Bayesian Reasoning and Machine Learning

Notation List

Generally a bold face $\mathbf{x}$ denotes a vector, and similarly $\mathbf{A}$ denotes a matrix.

$\mathcal{V}$	a calligraphic symbol typically denotes a set of random variables	3
$\mathrm{dom}(x)$	Domain of a variable	3
$x = \mathsf{x}$	The variable x is in the state x	3
$p(x = \mathsf{tr})$	probability of event/variable x being in the state true	3
$p(x = \mathsf{fa})$	probability of event/variable x being in the state false	3
$p(x, y)$	probability of x and y	4
$p(x \cap y)$	probability of x and y	4
$p(x \cup y)$	probability of x or y	4
$p(x\|y)$	The probability of x conditioned on y	4
$\int_x f(x)$	For continuous variables this is shorthand for $\int f(x)dx$ and for discrete variables means summation over the states of x, $\sum_x f(x)$	7
$\mathbb{I}[x = y]$	Indicator : has value 1 if $x = y$, 0 otherwise	11
$\mathrm{pa}(x)$	The parents of node x.	20
$\mathrm{ch}(x)$	The children of node x.	20
$\mathrm{ne}(x)$	Neighbours of node x	20
$\mathcal{X} \perp\!\!\!\perp \mathcal{Y} \| \mathcal{Z}$	Variables $\mathcal{X}$ are independent of variables $\mathcal{Y}$ conditioned on variables $\mathcal{Z}$.	33
$\mathcal{X} \top\!\!\!\top \mathcal{Y} \| \mathcal{Z}$	Variables $\mathcal{X}$ are dependent on variables $\mathcal{Y}$ conditioned variables $\mathcal{Z}$.	33
$\dim x$	For a discrete variable x, this denotes the number of states x can take	43
$\delta(a, b)$	Delta function. For discrete a, b, this is the Kronecker delta, $\delta_{a,b}$ and for continuous a, b the Dirac delta function $\delta(a - b)$	142
$\dim \mathbf{x}$	The dimension of the vector/matrix $\mathbf{x}$.	150
$\sharp(x = s, y = t)$	The number of times variable x is in state s and y in state t simultaneously.	172
$\mathcal{D}$	Dataset	251
n	Data index	251
N	Number of Dataset training points	251
$\sharp^x_y$	The number of times variable x is in state y	266
$\mathbf{S}$	Sample Covariance matrix	283
$\sigma(x)$	The logistic sigmoid $1/(1 + \exp(-x))$	319
$\mathrm{erf}(x)$	The (Gaussian) error function	319
$i \sim j$	The set of unique neighbouring edges on a graph	532
$\mathbf{I}_m$	The $m \times m$ identity matrix	550

Preface

Machine Learning

The last decade has seen considerable growth in interest in Artificial Intelligence and Machine Learning. In the broadest sense, these fields aim to 'learn something useful' about the environment within which the organism operates. How gathered information is processed leads to the development of algorithms – how to process high dimensional data and deal with uncertainty. In the early stages of research in Machine Learning and related areas, similar techniques were discovered in relatively isolated research communities. Whilst not all techniques have a natural description in terms of probability theory, many do, and it is the framework of Graphical Models (a marriage between graph and probability theory) that has enabled the understanding and transference of ideas from statistical physics, statistics, machine learning and information theory. To this extent it is now reasonable to expect that machine learning researchers are familiar with the basics of statistical modelling techniques.

This book concentrates on the probabilistic aspects of information processing and machine learning. Certainly no claim is made as to the correctness or that this is the only useful approach. Indeed, one might counter that this is unnecessary since "biological organisms don't use probability theory". Whether this is the case or not, it is undeniable that the framework of graphical models and probability has helped with the explosion of new algorithms and models in the machine learning community. One should also be clear that Bayesian viewpoint is not the only way to go about describing machine learning and information processing. Bayesian and probabilistic techniques really come into their own in domains where uncertainty is a necessary consideration.

The structure of the book

One aim of part I of the book is to encourage Computer Science students into this area. A particular difficulty that many modern students face is a limited formal training in calculus and linear algebra, meaning that minutiae of continuous and high-dimensional distributions can turn them away. In beginning with probability as a form of reasoning system, we hope to show the reader how ideas from logical inference and dynamical programming that they may be more familiar with have natural parallels in a probabilistic context. In particular, Computer Science students are familiar with the concept of algorithms as core. However, it is more common in machine learning to view the model as core, and how this is implemented is secondary. From this perspective, understanding how to translate a mathematical model into a piece of computer code is central.

Part II introduces the statistical background needed to understand continuous distributions and how learning can be viewed from a probabilistic framework. Part III discusses machine learning topics. Certainly some readers will raise an eyebrow to see their favourite statistical topic listed under machine learning. A difference viewpoint between statistics and machine learning is what kinds of systems we would ultimately like to construct (machines capable of ‘human/biological information processing tasks) rather than in some of the techniques. This section of the book is therefore what I feel would be useful for machine learners to know.

Part IV discusses dynamical models in which time is explicitly considered. In particular the Kalman Filter is treated as a form of graphical model, which helps emphasise what the model is, rather than focusing on it as a ‘filter’, as is more traditional in the engineering literature.

Part V contains a brief introduction to approximate inference techniques, including both stochastic (Monte Carlo) and deterministic (variational) techniques.

The references in the book are not generally intended as crediting authors with ideas, nor are they always to the most authoritative works. Rather, the references are largely to works which are at a level reasonably consistent with the book and which are readily available.

Whom this book is for

My primary aim was to write a book for final year undergraduates and graduates without significant experience in calculus and mathematics that gave an inroad into machine learning, much of which is currently phrased in terms of probabilities and multi-variate distributions. The aim was to encourage students that apparently unexciting statistical concepts are actually highly relevant for research in making intelligent systems that interact with humans in a natural manner. Such a research programme inevitably requires dealing with high-dimensional data, time-series, networks, logical reasoning, modelling and uncertainty.

Other books in this area

Whilst there are several excellent textbooks in this area, none currently meets the requirements that I personally need for teaching, namely one that contains demonstration code and gently introduces probability and statistics before leading on to more advanced topics in machine learning. This lead me to build on my lecture material from courses given at Aston, Edinburgh, EPFL and UCL and expand the demonstration software considerably. The book is due for publication by Cambridge University Press in 2010.

The literature on machine learning is vast, as is the overlap with the relevant areas of statistics, engineering and other physical sciences. In this respect, it is difficult to isolate particular areas, and this book is an attempt to integrate parts of the machine learning and statistics literature. The book is written in an informal style at the expense of rigour and detailed proofs. As an introductory textbook, topics are naturally covered to a somewhat shallow level and the reader is referred to more specialised books for deeper treatments. Amongst my favourites are:

- Graphical models
  - *Graphical models* by S. Lauritzen, Oxford University Press, 1996.
  - *Bayesian Networks and Decision Graphs* by F. Jensen and T. D. Nielsen, Springer Verlag, 2007.
  - *Probabilistic Networks and Expert Systems* by R. G. Cowell, A. P. Dawid, S. L. Lauritzen and D. J. Spiegelhalter, Springer Verlag, 1999.
  - *Probabilistic Reasoning in Intelligent Systems* by J. Pearl, Morgan Kaufmann, 1988.
  - *Graphical Models in Applied Multivariate Statistics* by J. Whittaker, Wiley, 1990.
- Machine Learning and Information Processing
  - *Information Theory, Inference and Learning Algorithms* by D. J. C. MacKay, Cambridge University Press, 2003.
  - *Pattern Recognition and Machine Learning* by C. M. Bishop, Springer Verlag, 2006.

- *An Introduction To Support Vector Machines*, N. Cristianini and J. Shawe-Taylor, Cambridge University Press, 2000.
- *Gaussian Processes for Machine Learning* by C. E. Rasmussen and C. K. I. Williams, MIT press, 2006.

How to use this book

Part I would be suitable for an introductory course on Graphical Models with a focus on inference. Part II contains enough material for a short lecture course on learning in probabilistic models. Part III is reasonably self-contained and would be suitable for a course on Machine Learning from a probabilistic perspective, particularly combined with the dynamical models material in part IV. Part V would be suitable for a short course on approximate inference.

Accompanying code

The MATLAB code is provided to help readers see how mathematical models translate into actual code. The code is not meant to be an industrial strength research tool, rather a reasonably lightweight toolbox that enables the reader to play with concepts in graph theory, probability theory and machine learning. In an attempt to retain readability, no extensive error and/or exception handling has been included. The code contains at the moment basic routines for manipulating discrete variable distributions, along with a set of routines that are more concerned with continuous variable machine learning. One could in principle extend the 'graphical models' part of the code considerably to support continuous variables. Limited support for continuous variables is currently provided so that, for example, inference in the linear dynamical system may be written in conducted of operations on Gaussian potentials. However, in general, potentials on continuous variables need to be manipulated with care and often specialised routines are required to ensure numerical stability.

Acknowledgements

Many people have helped this book along the way either in terms of reading, feedback, general insights, allowing me to present their work, or just plain motivation. Amongst these I would like to thank Massimiliano Pontil, Mark Herbster, John Shawe-Taylor, Vladimir Kolmogorov, Yuri Boykov, Tom Minka, Simon Prince, Silvia Chiappa, Bertrand Mesot, Robert Cowell, Ali Taylan Cemgil, David Blei, David Cohn, David Page, Peter Sollich, Chris Williams, Marc Toussaint, Amos Storkey, Zakria Hussain, Serafín Moral, Milan Studený, Tristan Fletcher, Tom Furmston, Ed Challis and Chris Bracegirdle. I would also like to thank the many students that have helped improve the material during lectures over the years. I'm particularly grateful to Tom Minka for allowing parts of his Lightspeed toolbox to be bundled with the BRMLtoolbox and am similarly indebted to Taylan Cemgil for his GraphLayout package.

A final thankyou to my family and friends.

Website

The code along with an electronic version of the book is available from

`http://www.cs.ucl.ac.uk/staff/D.Barber/brml`

Instructors seeking solutions to the exercises can find information at the website, along with additional teaching material. The website also contains a feedback form and errata list.

Contents

Part I

Inference in Probabilistic Models

CHAPTER 1

Probabilistic Reasoning

1.1 Probability Refresher

Variables, States and Notational Shortcuts

Variables will be denoted using either upper case X or lower case x and a set of variables will typically be denoted by a calligraphic symbol, for example $\mathcal{V} = \{a, B, c\}$.

The *domain* of a variable x is written $\text{dom}(x)$, and denotes the states x can take. States will typically be represented using sans-serif font. For example, for a coin c, we might have $\text{dom}(c) = \{\mathsf{heads}, \mathsf{tails}\}$ and $p(c = \mathsf{heads})$ represents the probability that variable c is in state heads.

The meaning of $p(\mathsf{state})$ will often be clear, without specific reference to a variable. For example, if we are discussing an experiment about a coin c, the meaning of $p(\mathsf{heads})$ is clear from the context, being shorthand for $p(c = \mathsf{heads})$. When summing (or performing some other operation) over a variable $\sum_x f(x)$, the interpretation is that all states of x are included, *i.e.* $\sum_x f(x) \equiv \sum_{\mathsf{s} \in \text{dom}(x)} f(x = \mathsf{s})$.

For our purposes, *events* are expressions about random variables, such as *Two heads in 6 coin tosses*. Two events are *mutually exclusive* if they cannot both simultaneously occur. For example the events *The coin is heads* and *The coin is tails* are mutually exclusive. One can think of defining a new variable named by the event so, for example, $p(\textit{The coin is tails})$ can be interpreted as $p(\textit{The coin is tails} = \mathsf{true})$.

The Rules of Probability

Definition 1 (Rules of Probability (Discrete Variables)).

The probability of an event x occurring is represented by a value between 0 and 1.

$p(x) = 1$ means that we are certain that the event does occur.

Conversely, $p(x) = 0$ means that we are certain that the event does not occur.

The summation of the probability over all the states is 1:

$$\sum_{\mathsf{x}} p(x = \mathsf{x}) = 1 \tag{1.1.1}$$

Such probabilities are normalised. We will usually more conveniently write $\sum_x p(x) = 1$.

Two events x and y can interact through

$$p(x \text{ or } y) = p(x) + p(y) - p(x \text{ and } y) \tag{1.1.2}$$

We will use the shorthand $p(x, y)$ for $p(x \text{ and } y)$. Note that $p(y, x) = p(x, y)$ and $p(x \text{ or } y) = p(y \text{ or } x)$.

Definition 2 (Set notation)**.** An alternative notation in terms of set theory is to write

$$p(x \text{ or } y) \equiv p(x \cup y), \qquad p(x, y) \equiv p(x \cap y) \tag{1.1.3}$$

Definition 3 (Marginals)**.** Given a *joint distribution* $p(x, y)$ the distribution of a single variable is given by

$$p(x) = \sum_y p(x, y) \tag{1.1.4}$$

Here $p(x)$ is termed a *marginal* of the joint probability distribution $p(x, y)$. The process of computing a marginal from a joint distribution is called *marginalisation*. More generally, one has

$$p(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = \sum_{x_i} p(x_1, \ldots, x_n) \tag{1.1.5}$$

An important definition that will play a central role in this book is conditional probability.

Definition 4 (Conditional Probability / Bayes' Rule)**.** The probability of event x conditioned on knowing event y (or more shortly, the probability of x given y) is defined as

$$p(x|y) \equiv \frac{p(x, y)}{p(y)} \tag{1.1.6}$$

If $p(y) = 0$ then $p(x|y)$ is not defined.

Probability Density Functions

Definition 5 (Probability Density Functions)**.** For a single continuous variable x, the probability density $p(x)$ is defined such that

$$p(x) \geq 0 \tag{1.1.7}$$

$$\int_{-\infty}^{\infty} p(x) dx = 1 \tag{1.1.8}$$

$$p(a < x < b) = \int_a^b p(x)dx \tag{1.1.9}$$

As shorthand we will sometimes write $\int_{x=a}^{b} p(x)$, particularly when we want an expression to be valid for either continuous or discrete variables. The multivariate case is analogous with integration over all real space, and the probability that x belongs to a region of the space defined accordingly.

For continuous variables, formally speaking, events are defined for the variable occurring within a defined region, for example

$$p(x \in [-1, 1.7]) = \int_{-1}^{1.7} f(x)dx \tag{1.1.10}$$

where here $f(x)$ is the probability density function (pdf) of the continuous random variable x. Unlike probabilities, probability densities can take positive values greater than 1.

Formally speaking, for a continuous variable, one should not speak of the probability that $x = 0.2$ since the probability of a single value is always zero. However, we shall often write $p(x)$ for continuous variables, thus not distinguishing between probabilities and probability density function values. Whilst this may appear strange, the nervous reader may simply replace our $p(X = x)$ notation for $\int_{x\in\Delta} f(x)dx$, where Δ is a small region centred on x. This is well defined in a probabilistic sense. In the limit Δ being very small, this would give approximately $\Delta f(x)$. If we consistently use the same Δ for all occurrences of pdfs, then we will simply have a common prefactor Δ in all expressions. Our strategy is to simply ignore these values (since in the end only relative probabilities will be relevant) and write $p(x)$. In this way, all the standard rules of probability carry over, including Bayes' Rule.

Interpreting Conditional Probability

Imagine a circular dart board, split into 20 equal sections, labelled from 1 to 20 and Randy, a dart thrower who hits any one of the 20 sections uniformly at random. Hence the probability that a dart thrown by Randy occurs in any one of the 20 regions is $p(\textit{region } i) = 1/20$. A friend of Randy tells him that he hasn't hit the 20 region. What is the probability that Randy has hit the 5 region? Conditioned on this information, only regions 1 to 19 remain possible and, since there is no preference for Randy to hit any of these regions, the probability is 1/19. The conditioning means that certain states are now inaccessible, and the original probability is subsequently distributed over the remaining accessible states. From the rules of probability :

$$p(\textit{region } 5|\textit{not region } 20) = \frac{p(\textit{region } 5, \textit{not region } 20)}{p(\textit{not region } 20)} = \frac{p(\textit{region } 5)}{p(\textit{not region } 20)} = \frac{1/20}{19/20} = \frac{1}{19}$$

giving the intuitive result. In the above $p(\textit{region } 5, \textit{not region } 20) = p(\textit{region } \{5 \cap 1 \cap 2\cap, \ldots, \cap 19\}) = p(\textit{region } 5)$.

An important point to clarify is that $p(A = \mathsf{a}|B = \mathsf{b})$ should not be interpreted as 'Given the event $B = \mathsf{b}$ has occurred, $p(A = \mathsf{a}|B = \mathsf{b})$ is the probability of the event $A = \mathsf{a}$ occurring'. In most contexts, no such explicit temporal causality is implied[1] and the correct interpretation should be ' $p(A = \mathsf{a}|B = \mathsf{b})$ is the probability of A being in state a under the constraint that B is in state b'.

The relation between the conditional $p(A = \mathsf{a}|B = \mathsf{b})$ and the joint $p(A = \mathsf{a}, B = \mathsf{b})$ is just a normalisation constant since $p(A = \mathsf{a}, B = \mathsf{b})$ is not a distribution in A – in other words, $\sum_{\mathsf{a}} p(A = \mathsf{a}, B = \mathsf{b}) \neq 1$. To make it a distribution we need to divide : $p(A = \mathsf{a}, B = \mathsf{b})/\sum_{\mathsf{a}} p(A = \mathsf{a}, B = \mathsf{b})$ which, when summed over a does sum to 1. Indeed, this is just the definition of $p(A = \mathsf{a}|B = \mathsf{b})$.

[1] We will discuss issues related to causality further in section(3.4).

Definition 6 (Independence)**.**

Events x and y are independent if knowing one event gives no extra information about the other event. Mathematically, this is expressed by

$$p(x, y) = p(x)p(y) \tag{1.1.11}$$

Provided that $p(x) \neq 0$ and $p(y) \neq 0$ independence of x and y is equivalent to

$$p(x|y) = p(x) \Leftrightarrow p(y|x) = p(y) \tag{1.1.12}$$

If $p(x|y) = p(x)$ for all states of x and y, then the variables x and y are said to be independent. If

$$p(x, y) = kf(x)g(y) \tag{1.1.13}$$

for some constant k, and positive functions $f(\cdot)$ and $g(\cdot)$ then x and y are independent.

Deterministic Dependencies

Sometimes the concept of independence is perhaps a little strange. Consider the following : variables x and y are both binary (their domains consist of two states). We define the distribution such that x and y are always both in a certain joint state:

$$p(x = \mathsf{a}, y = 1) = 1$$
$$p(x = \mathsf{a}, y = 2) = 0$$
$$p(x = \mathsf{b}, y = 2) = 0$$
$$p(x = \mathsf{b}, y = 1) = 0$$

Are x and y dependent? The reader may show that $p(x = \mathsf{a}) = 1$, $p(x = \mathsf{b}) = 0$ and $p(y = 1) = 1$, $p(y = 2) = 0$. Hence $p(x)p(y) = p(x, y)$ for all states of x and y, and x and y are therefore independent. This may seem strange – we know for sure the relation between x and y, namely that they are always in the same joint state, yet they are independent. Since the distribution is trivially concentrated in a single joint state, knowing the state of x tells you nothing that you didn't anyway know about the state of y, and vice versa.

This potential confusion comes from using the term 'independent' which, in English, suggests that there is no influence or relation between objects discussed. The best way to think about statistical independence is to ask whether or not knowing the state of variable y tells you something more than you knew before about variable x, where 'knew before' means working with the joint distribution of $p(x, y)$ to figure out what we can know about x, namely $p(x)$.

1.1.1 Probability Tables

Based on the populations 60776238, 5116900 and 2980700 of England (E), Scotland (S) and Wales (W), the a priori probability that a randomly selected person from these three countries would live in England, Scotland or Wales, would be approximately 0.88, 0.08 and 0.04 respectively. We can write this as a vector (or probability table) :

$$\begin{pmatrix} p(Cnt = \mathsf{E}) \\ p(Cnt = \mathsf{S}) \\ p(Cnt = \mathsf{W}) \end{pmatrix} = \begin{pmatrix} 0.88 \\ 0.08 \\ 0.04 \end{pmatrix} \tag{1.1.14}$$

who's component values sum to 1. The ordering of the components in this vector is arbitrary, as long as it is consistently applied.

For the sake of simplicity, let's assume that only three Mother Tongue languages exist : English (Eng), Scottish (Scot) and Welsh (Wel), with conditional probabilities given the country of residence, England (E), Scotland (S) and Wales (W)[2]. We write a conditional probability table

$$\begin{array}{lll} p(MT = \mathsf{Eng}|Cnt = \mathsf{E}) = 0.95 & p(MT = \mathsf{Scot}|Cnt = \mathsf{E}) = 0.04 & p(MT = \mathsf{Wel}|Cnt = \mathsf{E}) = 0.01 \\ p(MT = \mathsf{Eng}|Cnt = \mathsf{S}) = 0.7 & p(MT = \mathsf{Scot}|Cnt = \mathsf{S}) = 0.3 & p(MT = \mathsf{Wel}|Cnt = \mathsf{S}) = 0.0 \\ p(MT = \mathsf{Eng}|Cnt = \mathsf{W}) = 0.6 & p(MT = \mathsf{Scot}|Cnt = \mathsf{W}) = 0.0 & p(MT = \mathsf{Wel}|Cnt = \mathsf{W}) = 0.4 \end{array} \tag{1.1.15}$$

From this we can form a joint distribution $p(Cnt, MT) = p(MT|Cnt)p(Cnt)$. This could be written as a 3×3 matrix with (say) rows indexed by country and columns indexed by Mother Tongue:

$$\begin{pmatrix} 0.95 \times 0.88 & 0.7 \times 0.08 & 0.6 \times 0.04 \\ 0.04 \times 0.88 & 0.3 \times 0.08 & 0.0 \times 0.04 \\ 0.01 \times 0.88 & 0.0 \times 0.08 & 0.4 \times 0.04 \end{pmatrix} = \begin{pmatrix} 0.836 & 0.056 & 0.024 \\ 0.0352 & 0.024 & 0 \\ 0.0088 & 0 & 0.016 \end{pmatrix} \tag{1.1.16}$$

The joint distribution contains all the information about the model of this environment. By summing a column of this table, we have the marginal $p(Cnt)$. Summing the row gives the marginal $p(MT)$. Similarly, one could easily infer $p(Cnt|MT) \propto p(Cnt|MT)p(MT)$ from this joint distribution.

For joint distributions over a larger number of variables, $x_i, i = 1, \ldots, D$, with each variable x_i taking K_i states, the table describing the joint distribution is an array with $\prod_{i=1}^{D} K_i$ entries. Explicitly storing tables therefore requires space exponential in the number of variables, which rapidly becomes impractical for a large number of variables.

A probability distribution assigns a value to each of the joint states of the variables. For this reason, $p(T, J, R, S)$ is considered equivalent to $p(J, S, R, T)$ (or any such reordering of the variables), since in each case the joint setting of the variables is simply a different index to the same probability. This situation is more clear in the set theoretic notation $p(J \cap S \cap T \cap R)$. We abbreviate this set theoretic notation by using the commas – however, one should be careful not to confuse the use of this indexing type notation with functions $f(x, y)$ which are in general dependent on the variable order. Whilst the variables to the left of the conditioning bar may be written in any order, and equally those to the right of the conditioning bar may be written in any order, moving variables across the bar is not generally equivalent, so that $p(x_1|x_2) \neq p(x_2|x_1)$.

1.1.2 Interpreting Conditional Probability

Together with the rules of probability, conditional probability enables one to reason in a rational, logical and consistent way. One could argue that much of science deals with problems of the form : tell me something about the parameters θ given that I have observed data $\mathcal{D}$ and have some knowledge of the underlying data generating mechanism. From a modelling perspective, this requires

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} = \frac{p(D|\theta)p(\theta)}{\int_\theta p(\mathcal{D}|\theta)p(\theta)} \tag{1.1.17}$$

This shows how from a forward or *generative model* $p(\mathcal{D}|\theta)$ of the dataset, and coupled with a *prior* belief $p(\theta)$ about which parameter values are appropriate, we can infer the *posterior* distribution $p(\theta|\mathcal{D})$ of parameters in light of the observed data.

This use of a generative model sits well with physical models of the world which typically postulate how to generate observed phenomena, assuming we know the correct parameters of the model. For example, one might postulate how to generate a time-series of displacements for a swinging pendulum but with unknown mass, length and damping constant. Using this generative model, and given only the displacements, we could infer the unknown physical properties of the pendulum, such as its mass, length and friction damping constant.

[2]This information is fictitious.

Subjective Probability

Probability is a contentious topic and we do not wish to get bogged down too much here, apart from pointing out that it is not necessarily the axioms of probability that are contentious rather what interpretation we should place on them. In some cases potential repetitions of an experiment can be envisaged so that the 'long run' (or frequentist) definition of probability in which probabilities are defined with respect to a potentially infinite repetition of 'experiments' makes sense. For example, in coin tossing, the probability of heads might be interpreted as 'If I were to repeat the experiment of flipping coins (at 'random'), the limit of the number of heads that occurred over the number of tosses is defined as the probability of a head occurring.

Here's another problem that is typical of the kind of scenario one might face in a machine learning situation. A film enthusiast joins a new online film service. Based on expressing a few films a user likes and dislikes, the online company tries to estimate the probability that the user will like each of the 10000 films in their database. In a purely technical sense, if we were to define probability as a limiting case of infinite repetitions of the same experiment, this wouldn't make much sense in this case. We can't repeat the experiment – we just have a small number of films rated by the user. However, if we assume that the user behaves in a manner consistent with other users, we should be able to exploit the large amount of data from other users' ratings to make a reasonable 'guess' as to what this consumer likes. This *degree of belief* or *Bayesian* subjective interpretation of probability sidesteps non-repeatability issues – it's just a consistent framework for manipulating real values consistent with our intuition about probability.

The major use of probability in this book is as a consistent way to perform inference and reasoning in uncertain environments [143], and one which extends our traditional logical intuitions in deterministic environments.

1.2 Probabilistic Reasoning

The axioms of probability, combined with Bayes rule make for a complete reasoning system, one which includes traditional deductive logic as a special case[143].

Remark 1. The central paradigm of probabilistic reasoning is to identify all relevant variables $x_1, \ldots, x_N$ in the environment, and make a probabilistic model $p(x_1, \ldots, x_N)$ of their interaction. Reasoning (inference) is then performed by introducing *evidence* that sets variables in known states, and subsequently computing probabilities of interest, conditioned on this evidence.

Example 1 (Hamburgers)**.** Consider the following fictitious scientific information: Doctors find that people with Kreuzfeld-Jacob disease (KJ) almost invariably ate hamburgers, thus $p(\mathit{Hamburger\ Eater}|\mathit{KJ}) = 0.9$. The probability of an individual having KJ is currently rather low, about one in 100,000.

1. Assuming eating lots of hamburgers is rather widespread, say $p(\mathit{Hamburger\ Eater}) = 0.5$, what is the probability that a hamburger eater will have Kreuzfeld-Jacob disease?

   This may be computed as

$$p(\mathit{KJ}|\mathit{Hamburger\ Eater}) = \frac{p(\mathit{Hamburger\ Eater}, \mathit{KJ})}{p(\mathit{Hamburger\ Eater})} = \frac{p(\mathit{Hamburger\ Eater}|\mathit{KJ})p(\mathit{KJ})}{p(\mathit{Hamburger\ Eater})} \tag{1.2.1}$$

$$= \frac{\frac{9}{10} \times \frac{1}{100000}}{\frac{1}{2}} = 1.8 \times 10^{-5} \tag{1.2.2}$$

2. If the fraction of people eating hamburgers was rather small, $p(\textit{Hamburger Eater}) = 0.001$, what is the probability that a regular hamburger eater will have Kreuzfeld-Jacob disease? Repeating the above calculation, this is given by

$$\frac{\frac{9}{10} \times \frac{1}{100000}}{\frac{1}{1000}} \approx 1/100 \tag{1.2.3}$$

Intuitively, this is much higher than in scenario (1) since here we can be more sure that eating hamburgers is related to the illness. In this case only a small number of people in the population eat hamburgers, and most of them get ill.

Example 2 (Inspector Clouseau)**.** Inspector Clouseau arrives at the scene of a crime. The victim lies dead in the room and the inspector quickly finds the murder weapon, a Knife (K). The Butler (B) and Maid (M) are his main suspects. The inspector has a prior belief of 0.8 that the Butler is the murderer, and a prior belief of 0.2 that the Maid is the murderer. These probabilities are independent in the sense that $p(B, M) = p(B)p(M)$. (It is possible that both the Butler and the Maid murdered the victim or neither). The inspector's *prior* criminal knowledge can be formulated mathematically as follows:

$$\mathrm{dom}(B) = \mathrm{dom}(M) = \{\mathsf{murderer}, \mathsf{not\ murderer}\}, \mathrm{dom}(K) = \{\mathsf{knife\ used}, \mathsf{knife\ not\ used}\} \tag{1.2.4}$$

$$p(B = \mathsf{murderer}) = 0.8, \qquad p(M = \mathsf{murderer}) = 0.2 \tag{1.2.5}$$

$$\begin{array}{lll} p(\mathsf{knife\ used}|B = \mathsf{not\ murderer}, & M = \mathsf{not\ murderer}) & = 0.3 \\ p(\mathsf{knife\ used}|B = \mathsf{not\ murderer}, & M = \mathsf{murderer}) & = 0.2 \\ p(\mathsf{knife\ used}|B = \mathsf{murderer}, & M = \mathsf{not\ murderer}) & = 0.6 \\ p(\mathsf{knife\ used}|B = \mathsf{murderer}, & M = \mathsf{murderer}) & = 0.1 \end{array} \tag{1.2.6}$$

What is the probability that the Butler is the murderer? (Remember that it might be that neither is the murderer). Using b for the two states of B and m for the two states of M,

$$p(B|K) = \sum_m p(B, m|K) = \sum_m \frac{p(B, m, K)}{p(K)} = \frac{p(B) \sum_m p(K|B, m)p(m)}{\sum_b p(b) \sum_m p(K|b, m)p(m)} \tag{1.2.7}$$

Plugging in the values we have

$$p(B = \mathsf{murderer}|\mathsf{knife\ used}) = \frac{\frac{8}{10}\left(\frac{2}{10} \times \frac{1}{10} + \frac{8}{10} \times \frac{6}{10}\right)}{\frac{8}{10}\left(\frac{2}{10} \times \frac{1}{10} + \frac{8}{10} \times \frac{6}{10}\right) + \frac{2}{10}\left(\frac{2}{10} \times \frac{2}{10} + \frac{8}{10} \times \frac{3}{10}\right)} = \frac{200}{228} \approx 0.877 \tag{1.2.8}$$

The role of $p(\mathsf{knife\ used})$ in the Inspector Clouseau example can cause some confusion. In the above,

$$p(\mathsf{knife\ used}) = \sum_b p(b) \sum_m p(\mathsf{knife\ used}|b, m)p(m) \tag{1.2.9}$$

is computed to be 0.456. But surely, $p(\mathsf{knife\ used}) = 1$, since this is given in the question! Note that the quantity $p(\mathsf{knife\ used})$ relates to the *prior* probability the model assigns to the knife being used (in the absence of any other information). If we know that the knife is used, then the *posterior*

$$p(\mathsf{knife\ used}|\mathsf{knife\ used}) = \frac{p(\mathsf{knife\ used}, \mathsf{knife\ used})}{p(\mathsf{knife\ used})} = \frac{p(\mathsf{knife\ used})}{p(\mathsf{knife\ used})} = 1 \tag{1.2.10}$$

which, naturally, must be the case.

Another potential confusion is the choice

$$p(B = \mathsf{murderer}) = 0.8, \qquad p(M = \mathsf{murderer}) = 0.2 \tag{1.2.11}$$

which means that $p(B = \text{not murderer}) = 0.2$, $p(M = \text{not murderer}) = 0.8$. These events are not exclusive and it's just 'coincidence' that the numerical values are chosen this way. For example, we could have also chosen

$$p(B = \text{murderer}) = 0.6, \qquad p(M = \text{murderer}) = 0.9 \tag{1.2.12}$$

which means that $p(B = \text{not murderer}) = 0.4$, $p(M = \text{not murderer}) = 0.1$

1.3 Prior, Likelihood and Posterior

The prior, likelihood and posterior are all probabilities. They are assigned these names due to their role in Bayes' rule, described below.

Definition 7. Prior Likelihood and Posterior

For data $\mathcal{D}$ and variable θ, Bayes' rule tells us how to update our prior beliefs about the variable θ in light of the data to a posterior belief:

$$\underbrace{p(\theta|\mathcal{D})}_{\text{posterior}} = \frac{\overbrace{p(\mathcal{D}|\theta)}^{\text{likelihood}}\ \overbrace{p(\theta)}^{\text{prior}}}{\underbrace{p(\mathcal{D})}_{\textit{evidence}}} \tag{1.3.1}$$

The *evidence* is also called the *marginal likelihood*.

The term likelihood is used for the probability that a model generates observed data. More fully, if we condition on the model M, we have

$$p(\theta|\mathcal{D}, M) = \frac{p(\mathcal{D}|\theta, M)p(\theta|M)}{p(\mathcal{D}|M)}$$

where we see the role of the likelihood $p(\mathcal{D}|\theta, M)$ and marginal likelihood $p(\mathcal{D}|M)$. The marginal likelihood is also called the *model likelihood*.

The *most probable a posteriori* (*MAP*) setting is that which maximises the posterior, $\theta_* = \underset{\theta}{\text{argmax}}\ p(\theta|\mathcal{D}, M)$.

Bayes' rule tells us how to update our prior knowledge with the data generating mechanism. The prior distribution $p(\theta)$ describes the information we have about the variable before seeing any data. After data $\mathcal{D}$ arrives, we update the prior distribution to the posterior $p(\theta|\mathcal{D}) \propto p(\mathcal{D}|\theta)p(\theta)$. A clean example of this situation is given below.

1.3.1 Two dice : what were the individual scores?

Two fair dice are rolled. Someone tells you that the sum of the two scores is 9. What is the probability distribution of the two dice scores[3]?

The score of die a is denoted s_a with $\text{dom}(s_a) = \{1, 2, 3, 4, 5, 6\}$ and similarly for s_b. The three variables involved are then s_a, s_b and the total score, $t = s_a + s_b$. A model of these three variables naturally takes the form

$$p(t, s_a, s_b) = \underbrace{p(t|s_a, s_b)}_{\text{likelihood}}\ \underbrace{p(s_a, s_b)}_{\text{prior}} \tag{1.3.2}$$

[3]This example is due to Taylan Cemgil.

The prior $p(s_a, s_b)$ is the joint probability of score s_a and score s_b without knowing anything else. Assuming no dependency in the rolling mechanism,

$$p(s_a, s_b) = p(s_a)p(s_b) \tag{1.3.3}$$

Since the dice are fair both $p(s_a)$ and $p(s_b)$ are uniform distributions, $p(s_a = s) = 1/6$.

Here the likelihood term is

$$p(t|s_a, s_b) = \mathbb{I}\left[t = s_a + s_b\right] \tag{1.3.4}$$

which states that the total score is given by $s_a + s_b$. Here $\mathbb{I}\left[x = y\right]$ is the *indicator function* defined as $\mathbb{I}\left[x = y\right] = 1$ if $x = y$ and 0 otherwise.

Hence, our complete model is

$$p(t, s_a, s_b) = p(t|s_a, s_b)p(s_a)p(s_b) \tag{1.3.5}$$

where the terms on the right are explicitly defined.

Our interest is then obtainable using Bayes' rule,

$$p(s_a, s_b|t = 9) = \frac{p(t = 9|s_a, s_b)p(s_a)p(s_b)}{p(t = 9)} \tag{1.3.6}$$

where

$$p(t = 9) = \sum_{s_a, s_b} p(t = 9|s_a, s_b)p(s_a)p(s_b) \tag{1.3.7}$$

$p(s_a)p(s_b)$:

	$s_a = 1$	$s_a = 2$	$s_a = 3$	$s_a = 4$	$s_a = 5$	$s_a = 6$
$s_b = 1$	1/36	1/36	1/36	1/36	1/36	1/36
$s_b = 2$	1/36	1/36	1/36	1/36	1/36	1/36
$s_b = 3$	1/36	1/36	1/36	1/36	1/36	1/36
$s_b = 4$	1/36	1/36	1/36	1/36	1/36	1/36
$s_b = 5$	1/36	1/36	1/36	1/36	1/36	1/36
$s_b = 6$	1/36	1/36	1/36	1/36	1/36	1/36

$p(t = 9|s_a, s_b)$:

	$s_a = 1$	$s_a = 2$	$s_a = 3$	$s_a = 4$	$s_a = 5$	$s_a = 6$
$s_b = 1$	0	0	0	0	0	0
$s_b = 2$	0	0	0	0	0	0
$s_b = 3$	0	0	0	0	0	1
$s_b = 4$	0	0	0	0	1	0
$s_b = 5$	0	0	0	1	0	0
$s_b = 6$	0	0	1	0	0	0

$p(t = 9|s_a, s_b)p(s_a)p(s_b)$:

	$s_a = 1$	$s_a = 2$	$s_a = 3$	$s_a = 4$	$s_a = 5$	$s_a = 6$
$s_b = 1$	0	0	0	0	0	0
$s_b = 2$	0	0	0	0	0	0
$s_b = 3$	0	0	0	0	0	1/36
$s_b = 4$	0	0	0	0	1/36	0
$s_b = 5$	0	0	0	1/36	0	0
$s_b = 6$	0	0	1/36	0	0	0

$p(s_a, s_b|t = 9)$:

	$s_a = 1$	$s_a = 2$	$s_a = 3$	$s_a = 4$	$s_a = 5$	$s_a = 6$
$s_b = 1$	0	0	0	0	0	0
$s_b = 2$	0	0	0	0	0	0
$s_b = 3$	0	0	0	0	0	1/4
$s_b = 4$	0	0	0	0	1/4	0
$s_b = 5$	0	0	0	1/4	0	0
$s_b = 6$	0	0	1/4	0	0	0

The term $p(t = 9) = \sum_{s_a, s_b} p(t = 9|s_a, s_b)p(s_a)p(s_b) = 4 \times 1/36 = 1/9$. Hence the posterior is given by equal mass in only 4 non-zero elements, as shown.

1.4 Further worked examples

Example 3 (Who's in the bathroom?)**.** Consider a household of three people, Alice, Bob and Cecil. Cecil wants to go to the bathroom but finds it occupied. He then goes to Alice's room and sees she is there. Since Cecil knows that only either Alice or Bob can be in the bathroom, from this he infers that Bob must be in the bathroom.

To arrive at the same conclusion in a mathematical framework, let's define the following events

$$A = \text{Alice is in her bedroom}, \quad B = \text{Bob is in his bedroom}, \quad O = \text{Bathroom occupied} \tag{1.4.1}$$

We can encode the information that if either Alice or Bob are not in their bedrooms, then they must be in the bathroom (they might both be in the bathroom) as

$$p(O = \mathsf{tr}|A = \mathsf{fa}, B) = 1, \qquad p(O = \mathsf{tr}|A, B = \mathsf{fa}) = 1 \tag{1.4.2}$$

The first term expresses that the bathroom is occupied if Alice is not in her bedroom, wherever Bob is. Similarly, the second term expresses bathroom occupancy as long as Bob is not in his bedroom. Then

$$p(B = \mathsf{fa}|O = \mathsf{tr}, A = \mathsf{tr}) = \frac{p(B = \mathsf{fa}, O = \mathsf{tr}, A = \mathsf{tr})}{p(O = \mathsf{tr}, A = \mathsf{tr})} = \frac{p(O = \mathsf{tr}|A = \mathsf{tr}, B = \mathsf{fa})p(A = \mathsf{tr}, B = \mathsf{fa})}{p(O = \mathsf{tr}, A = \mathsf{tr})} \tag{1.4.3}$$

where

$$p(O = \mathsf{tr}, A = \mathsf{tr}) = p(O = \mathsf{tr}|A = \mathsf{tr}, B = \mathsf{fa})p(A = \mathsf{tr}, B = \mathsf{fa}) + p(O = \mathsf{tr}|A = \mathsf{tr}, B = \mathsf{tr})p(A = \mathsf{tr}, B = \mathsf{tr}) \tag{1.4.4}$$

Using the fact $p(O = \mathsf{tr}|A = \mathsf{tr}, B = \mathsf{fa}) = 1$ and $p(O = \mathsf{tr}|A = \mathsf{tr}, B = \mathsf{tr}) = 0$, which encodes that if if Alice is in her room and Bob is not, the bathroom must be occupied, and similarly, if both Alice and Bob are in their rooms, the bathroom cannot be occupied,

$$p(B = \mathsf{fa}|O = \mathsf{tr}, A = \mathsf{tr}) = \frac{p(A = \mathsf{tr}, B = \mathsf{fa})}{p(A = \mathsf{tr}, B = \mathsf{fa})} = 1 \tag{1.4.5}$$

This example is interesting since we are not required to make a full probabilistic model in this case thanks to the limiting nature of the probabilities (we don't need to specify $p(A, B)$). The situation is common in limiting situations of probabilities being either zero or 1, corresponding to traditional logic systems.

Example 4 (Aristotle : Resolution)**.** We can represent the statement 'All apples are fruit' by $p(F = \mathsf{tr}|A = \mathsf{tr}) = 1$. Similarly, 'All fruits grow on trees' may be represented by $p(T = \mathsf{tr}|F = \mathsf{tr}) = 1$. Additionally we assume that whether or not something grows on a tree depends only on whether or not it is a fruit, $p(T|A, F) = P(T|F)$. From these, we can compute

$$p(T = \mathsf{tr}|A = \mathsf{tr}) = \sum_F p(T = \mathsf{tr}|F, A = \mathsf{tr})p(F|A = \mathsf{tr}) = \sum_F p(T = \mathsf{tr}|F)p(F|A = \mathsf{tr}) \tag{1.4.6}$$

$$= p(T = \mathsf{tr}|F = \mathsf{fa}) \underbrace{p(F = \mathsf{fa}|A = \mathsf{tr})}_{=0} + \underbrace{p(T = \mathsf{tr}|F = \mathsf{tr})}_{=1} \underbrace{p(F = \mathsf{tr}|A = \mathsf{tr})}_{=1} = 1 \tag{1.4.7}$$

In other words we have deduced that 'All apples grow on trees' is a true statement, based on the information presented. (This kind of reasoning is called resolution and is a form of transitivity : from the statements $A \Rightarrow F$ and $F \Rightarrow T$ we can infer $A \Rightarrow T$).

Example 5 (Aristotle : Inverse Modus Ponens)**.** According to Logic, from the statement : 'If A is true then B is true', one may deduce that 'if B is false then A is false'. Let's see how this fits in with a probabilistic reasoning system. We can express the statement : 'If A is true then B is true' as $p(B = \mathsf{tr}|A = \mathsf{tr}) = 1$. Then we may infer

$$\begin{aligned} p(A = \mathsf{fa}|B = \mathsf{fa}) &= 1 - p(A = \mathsf{tr}|B = \mathsf{fa}) \\ &= 1 - \frac{p(B = \mathsf{fa}|A = \mathsf{tr})p(A = \mathsf{tr})}{p(B = \mathsf{fa}|A = \mathsf{tr})p(A = \mathsf{tr}) + p(B = \mathsf{fa}|A = \mathsf{fa})p(A = \mathsf{fa})} = 1 \end{aligned} \tag{1.4.8}$$

This follows since $p(B = \mathsf{fa}|A = \mathsf{tr}) = 1 - p(B = \mathsf{tr}|A = \mathsf{tr}) = 1 - 1 = 0$, annihilating the second term.

Both the above examples are intuitive expressions of deductive logic. The standard rules of Aristotelian logic are therefore seen to be limiting cases of probabilistic reasoning.

Example 6 (Soft XOR Gate)**.**
A standard XOR logic gate is given by the table on the right. If we observe that the output of the XOR gate is 0, what can we say about A and B? In this case, either A and B were both 0, or A and B were both 1. This means we don't know which state A was in – it could equally likely have been 1 or 0.

A	B	A xor B
0	0	0
0	1	1
1	0	1
1	1	0

Consider a 'soft' version of the XOR gate given on the right, with additionally $p(A = 1) = 0.65$, $p(B = 1) = 0.77$. What is $p(A = 1|C = 0)$?

A	B	$p(C = 1\|A, B)$
0	0	0.1
0	1	0.99
1	0	0.8
1	1	0.25

$$\begin{aligned} p(A=1,C=0) &= \sum_B p(A=1,B,C=0) = \sum_B p(C=0|A=1,B)p(A=1)p(B) \\ &= p(A=1)\left(p(C=0|A=1,B=0)p(B=0) + p(C=0|A=1,B=1)p(B=1)\right) \\ &= 0.65 \times (0.2 \times 0.23 + 0.75 \times 0.77) = 0.4053 \end{aligned} \tag{1.4.9, 1.4.10}$$

$$\begin{aligned} p(A=0,C=0) &= \sum_B p(A=0,B,C=0) = \sum_B p(C=0|A=0,B)p(A=0)p(B) \\ &= p(A=0)\left(p(C=0|A=0,B=0)p(B=0) + p(C=0|A=0,B=1)p(B=1)\right) \\ &= 0.35 \times (0.9 \times 0.23 + 0.01 \times 0.77) = 0.0751 \end{aligned}$$

Then

$$p(A=1|C=0) = \frac{p(A=1,C=0)}{p(A=1,C=0)+p(A=0,C=0)} = \frac{0.4053}{0.4053+0.0751} = 0.8437 \tag{1.4.11}$$

Example 7 (Larry)**.** Larry is typically late for school. If Larry is late, we denote this with $L = \mathsf{late}$, otherwise, $L = \mathsf{not\ late}$. When his mother asks whether or not he was late for school he never admits to being late. The response Larry gives R_L is represented as follows

$$p(R_L = \mathsf{not\ late}|L = \mathsf{not\ late}) = 1, \qquad p(R_L = \mathsf{late}|L = \mathsf{late}) = 0 \tag{1.4.12}$$

The remaining two values are determined by normalisation and are

$$p(R_L = \mathsf{late}|L = \mathsf{not\ late}) = 0, \qquad p(R_L = \mathsf{not\ late}|L = \mathsf{late}) = 1 \tag{1.4.13}$$

Given that $R_L = \mathsf{not\ late}$, what is the probability that Larry was late, *i.e.* $p(L = \mathsf{late}|R_L = \mathsf{not\ late})$?

Using Bayes' we have

$$\begin{aligned} p(L=\mathsf{late}|R_L=\mathsf{not\ late}) &= \frac{p(L=\mathsf{late},R_L=\mathsf{not\ late})}{p(R_L=\mathsf{not\ late})} \\ &= \frac{p(L=\mathsf{late},R_L=\mathsf{not\ late})}{p(L=\mathsf{late},R_L=\mathsf{not\ late})+p(L=\mathsf{not\ late},R_L=\mathsf{not\ late})} \end{aligned} \tag{1.4.14}$$

In the above

$$p(L=\mathsf{late},R_L=\mathsf{not\ late}) = \underbrace{p(R_L=\mathsf{not\ late}|L=\mathsf{late})}_{=1}p(L=\mathsf{late}) \tag{1.4.15}$$

and

$$p(L=\mathsf{not\ late},R_L=\mathsf{not\ late}) = \underbrace{p(R_L=\mathsf{not\ late}|L=\mathsf{not\ late})}_{=1}p(L=\mathsf{not\ late}) \tag{1.4.16}$$

Hence

$$p(L = \mathsf{late}|R_L = \mathsf{not\ late}) = \frac{p(L = \mathsf{late})}{p(L = \mathsf{late}) + p(L = \mathsf{not\ late})} = p(L = \mathsf{late}) \tag{1.4.17}$$

Where we used normalisation in the last step, $p(L = \mathsf{late}) + p(L = \mathsf{not\ late}) = 1$. This result is intuitive – Larry's mother knows that he never admits to being late, so her belief about whether or not he really was late is unchanged, regardless of what Larry actually says.

Example 8 (Larry and Sue)**.** Continuing the example above, Larry's sister Sue always tells the truth to her mother as to whether or not Larry was late for School.

$$p(R_S = \mathsf{not\ late}|L = \mathsf{not\ late}) = 1, \quad p(R_S = \mathsf{late}|L = \mathsf{late}) = 1 \tag{1.4.18}$$

The remaining two values are determined by normalisation and are

$$p(R_S = \mathsf{late}|L = \mathsf{not\ late}) = 0, \quad p(R_S = \mathsf{not\ late}|L = \mathsf{late}) = 0 \tag{1.4.19}$$

We also assume $p(R_S, R_L|L) = p(R_S|L)p(R_L|L)$. We can then write

$$p(R_L, R_S, L) = p(R_L|L)p(R_S|L)p(L) \tag{1.4.20}$$

Given that $R_S = \mathsf{late}$, what is the probability that Larry was late?

Using Bayes' rule, we have

$$p(L = \mathsf{late}|R_L = \mathsf{not\ late}, R_S = \mathsf{late}) = \frac{1}{Z}p(R_S = \mathsf{late}|L = \mathsf{late})p(R_L = \mathsf{not\ late}|L = \mathsf{late})p(L = \mathsf{late})$$

where the normalisation Z is given by

$$p(R_S = \mathsf{late}|L = \mathsf{late})p(R_L = \mathsf{not\ late}|L = \mathsf{late})p(L = \mathsf{late}) \\ + p(R_S = \mathsf{late}|L = \mathsf{not\ late})p(R_L = \mathsf{not\ late}|L = \mathsf{not\ late})p(L = \mathsf{not\ late})$$

Hence

$$p(L = \mathsf{late}|R_L = \mathsf{not\ late}, R_S = \mathsf{late}) = \frac{1 \times 1 \times p(L = \mathsf{late})}{1 \times 1 \times p(L = \mathsf{late}) + 0 \times 1 \times p(L = \mathsf{not\ late})} = 1 \tag{1.4.21}$$

This result is also intuitive – Since Larry's mother knows that Sue always tells the truth, no matter what Larry says, she knows he was late.

Example 9 (Luke)**.** Luke has been told he's lucky and has won a prize in the lottery. There are 5 prizes available of value £10, £100, £1000, £10000, £1000000. The prior probabilities of winning these 5 prizes are p_1, p_2, p_3, p_4, p_5, with p_0 being the prior probability of winning no prize. Luke asks eagerly 'Did I win £1000000?!'. "I'm afraid not sir", is the response of the lottery phone operator. "Did I win £10000?!" asks Luke. "Again, I'm afraid not sir". What is the probability that Luke has won £1000?

Note first that $p_0+p_1+p_2+p_3+p_4+p_5 = 1$. We denote $W = 1$ for the first prize of £10, and $W = 2, \ldots, 5$

for the remaining prizes and $W = 0$ for no prize. We need to compute

$$p(W = 3|W \neq 5, W \neq 4, W \neq 0) = \frac{p(W = 3, W \neq 5, W \neq 4, W \neq 0)}{p(W \neq 5, W \neq 4, W \neq 0)}$$
$$= \frac{p(W = 3)}{p(W = 1 \text{ or } W = 2 \text{ or } W = 3)} = \frac{p_3}{p_1 + p_2 + p_3} \tag{1.4.22}$$

where the term in the denominator is computed using the fact that the events W are mutually exclusive (one can only win one prize). This result makes intuitive sense : once we have removed the impossible states of W, the probability that Luke wins the prize is proportional to the prior probability of that prize, with the normalisation being simply the total set of possible probability remaining.

1.5 Code

The BRMLtoolbox code accompanying this book is intended to give the reader some insight into representing discrete probability tables and performing simple inference. The MATLAB[4] code is written with only minimal error trapping to keep the code as short and hopefully reasonably readable.

1.5.1 Basic Probability code

At the simplest level, we only need two basic routines. One for multiplying probability tables together (called potentials in the code), and one for summing a probability table. Potentials are represented using a structure. For example, in the code corresponding to the Inspector Clouseau example `demoClouseau.m`, we define a probability table as

```
>> pot(1)
ans =
    variables: [1 3 2]
        table: [2x2x2 double]
```

This says that the potential depends on the variables $1, 3, 2$ and the entries are stored in the array given by the table field. The size of the array informs how many states each variable takes in the order given by `variables`. The order in which the variables are defined in a potential is irrelevant provided that one indexes the array consistently. A routine that can help with setting table entries is `setstate.m`. For example,

```
>> pot(1) = setstate(pot(1),[2 1 3],[2 1 1],0.3)
```

means that for potential 1, the table entry for variable 2 being in state 2, variable 1 being in state 1 and variable 3 being in state 1 should be set to value 0.3.

The philosophy of the code is to keep the information required to perform computations to a minimum. Additional information about the labels of variables and their domains can be useful to check results, but is not actually required to carry out computations. One may also specify the name and domain of each variable, for example

```
>>variable(3)
ans =
    domain: {'murderer'  'not murderer'}
      name: 'butler'
```

The variable name and domain information in the Clouseau example is stored in the structure `variable`, which can be helpful to display the potential table:

[4] At the time of writing, some versions of MATLAB suffer from serious memory indexing bugs, some of which may appear in the array structures used in the code provided. To deal with this, turn off the JIT accelerator by typing `feature accel off`.

```
>> disptable(pot(1),variable);
knife   =   used        maid    = murderer        butler  =  murderer      0.100000
knife   =   not used    maid    = murderer        butler  =  murderer      0.900000
knife   =   used        maid    = not murderer    butler  =  murderer      0.600000
knife   =   not used    maid    = not murderer    butler  =  murderer      0.400000
knife   =   used        maid    = murderer        butler  =  not murderer  0.200000
knife   =   not used    maid    = murderer        butler  =  not murderer  0.800000
knife   =   used        maid    = not murderer    butler  =  not murderer  0.300000
knife   =   not used    maid    = not murderer    butler  =  not murderer  0.700000
```

Multiplying Potentials

In order to multiply potentials (as for arrays) the tables of each potential must be dimensionally consistent – that is the number of states of variable i in potential 1 must match the number of states of variable i in any other potential which can be checked using `potvariables.m`. This consistency is also required for other basic operations such as summing potentials.
`multpots.m`: Multiplying two or more potentials
`divpots.m`: Dividing a potential by another

Summing a Potential

`sumpot.m`: Sum (marginalise) a potential over a set of variables
`sumpots.m`: Sum a set of potentials together

Making a conditional Potential

`condpot.m`: Make a potential conditioned on variables

Setting a Potential

`setpot.m`: Set variables in a potential to given states
`setevpot.m`: Set variables in a potential to given states and return also an identity potential on the given states

The philosophy of BRMLtoolbox is that all information about variables is local and is read off from a potential. Using `setevpot.m` enables one to set variables in a state whilst maintaining information about the number of states of a variable.

Maximising a Potential

`maxpot.m`: Maximise a potential over a set of variables

See also `maxNarray.m` and `maxNpot.m` which return the N-highest values and associated states.

Other potential utilities

`setstate.m`: Set the a potential state to a given value
`table.m`: Return a table from a potential
`whichpot.m`: Return potentials which contain a set of variables
`potvariables.m`: Variables and their number of states in a set of potentials
`orderpotfields.m`: Order the fields of a potential structure
`uniquepots.m`: Merge redundant potentials and return only unique ones
`numstates.m`: Number of states of a variable in a domain
`squeezepots.m`: Remove redundant potentials by merging

`normpot.m`: Normalise a potential to form a distribution

1.5.2 General utilities

`condp.m`: Return a table $p(x|y)$ from $p(x, y)$
`condexp.m`: Form a conditional distribution from a log value
`logsumexp.m`: Compute the log of a sum of exponentials in a numerically precise way
`normp.m`: Return a normalised table from an unnormalised table
`assign.m`: Assign values to multiple variables
`maxarray.m`: Maximize a multi-dimensional array over a subset

1.5.3 An example

The following code highlights the use of the above routines in solving the Inspector Clouseau, example(2).
`demoClouseau.m`: Solving the Inspector Clouseau example

1.6 Notes

The interpretation of probability is contentious. We refer the reader to [143, 180, 176] for detailed discussions. A useful website that relates to understanding probability and Bayesian reasoning is understandinguncertainty.org.

1.7 Exercises

Exercise 1. *Prove*

$$p(x, y|z) = p(x|z)p(y|x, z) \tag{1.7.1}$$

and also

$$p(x|y, z) = \frac{p(y|x, z)p(x|z)}{p(y|z)} \tag{1.7.2}$$

Exercise 2. *Prove the* Bonferroni inequality

$$p(a, b) \geq p(a) + p(b) - 1 \tag{1.7.3}$$

Exercise 3 (Adapted from [164])**.** *There are two boxes. Box 1 contains three red and five white balls and box 2 contains two red and five white balls. A box is chosen at random $p(box = 1) = p(box = 2) = 0.5$ and a ball chosen at random from this box turns out to be red. What is the posterior probability that the red ball came from box 1?*

Exercise 4 (Adapted from [164])**.** *Two balls are placed in a box as follows: A fair coin is tossed and a white ball is placed in the box if a head occurs, otherwise a red ball is placed in the box. The coin is tossed again and a red ball is placed in the box if a tail occurs, otherwise a white ball is placed in the box. Balls are drawn from the box three times in succession (always with replacing the drawn ball back in the box). It is found that on all three occasions a red ball is drawn. What is the probability that both balls in the box are red?*

Exercise 5 (From David Spiegelhalter `understandinguncertainty.org`)**.** *A secret government agency has developed a scanner which determines whether a person is a terrorist. The scanner is fairly reliable; 95% of all scanned terrorists are identified as terrorists, and 95% of all upstanding citizens are identified as such. An informant tells the agency that exactly one passenger of 100 aboard an aeroplane in which you are seated is a terrorist. The agency decide to scan each passenger and the shifty looking man sitting next to you is the first to test positive. What are the chances that this man is a terrorist?*

Exercise 6 (The Monty Hall problem). *On a gameshow, there are three doors. Behind one door is a prize. The gameshow host asks you to pick a door. He then opens a different door to the one you chose and shows that there is no prize behind it. Is is better to stick with your original guess as to where the prize is, or better to change your mind?*

Exercise 7. *Consider three variable distributions which admit the factorisation*

$$p(a, b, c) = p(a|b)p(b|c)p(c) \tag{1.7.4}$$

where all variables are binary. How many parameters are needed to specify distributions of this form?

Exercise 8. *Repeat the Inspector Clouseau scenario, example(2), but with the restriction that either the Maid or the Butler is the murderer, but not both. Explicitly, the probability of the Maid being the Murder and not the Butler is 0.04, the probability of the Butler being the Murder and not the Maid is 0.64. Modify* `demoClouseau.m` *to implement this.*

Exercise 9. *Prove*

$$p(a, (b \text{ or } c)) = p(a, b) + p(a, c) - p(a, b, c) \tag{1.7.5}$$

Exercise 10. *Prove*

$$p(x|z) = \sum_y p(x|y, z)p(y|z) = \sum_{y,w} p(x|w, y, z)p(w|y, z)p(y|z) \tag{1.7.6}$$

Exercise 11. *As a young man Mr Gott visits Berlin in 1969. He's surprised that he cannot cross into East Berlin since there is a wall separating the two halves of the city. He's told that the wall was erected 8 years previously. He reasons that : The wall will have a finite lifespan; his ignorance means that he arrives uniformly at random at some time in the lifespan of the wall. Since only 5% of the time one would arrive in the first or last 2.5% of the lifespan of the wall he asserts that with 95% confidence the wall will survive between* $8/0.975 \approx 8.2$ *and* $8/0.025 = 320$ *years. In 1989 the now Professor Gott is pleased to find that his prediction was correct and promotes his prediction method in elite journals. This 'delta-t' method is widely adopted and used to form predictions in a range of scenarios about which researchers are 'totally ignorant'. Would you 'buy' a prediction from Prof. Gott? Explain carefully your reasoning.*

Exercise 12. *Implement the soft XOR gate, example(6) using* BRMLtoolbox*. You may find* `condpot.m` *of use.*

Exercise 13. *Implement the hamburgers, example(1) (both scenarios) using* BRMLtoolbox*. To do so you will need to define the joint distribution* $p(hamburgers, KJ)$ *in which* $dom(hamburgers) = dom(KJ) = \{\mathsf{tr}, \mathsf{fa}\}$.

Exercise 14. *Implement the two-dice example, section(1.3.1) using* BRMLtoolbox*.*

Exercise 15. *A redistribution lottery involves picking the correct four numbers from 1 to 9 (without replacement, so 3,4,4,1 for example is not possible). The order of the picked numbers is irrelevant. Every week a million people play this game, each paying* £1 *to enter, with the numbers 3,5,7,9 being the most popular (1 in every 100 people chooses these numbers). Given that the million pounds prize money is split equally between winners, and that any four (different) numbers come up at random, what is the expected amount of money each of the players choosing 3,5,7,9 will win each week? The least popular set of numbers is 1,2,3,4 with only 1 in 10,000 people choosing this. How much do they profit each week, on average? Do you think there is any 'skill' involved in playing this lottery?*

Exercise 16. *In a test of 'psychometry' the car keys and wrist watches of 5 people are given to a medium. The medium then attempts to match the wrist watch with the car key of each person. What is the expected number of correct matches that the medium will make (by chance)? What is the probability that the medium will obtain at least 1 correct match?*

CHAPTER 2

Basic Graph Concepts

2.1 Graphs

Definition 8 (Graph)**.** A *graph* G consists of vertices (nodes) and edges (links) between the vertices. Edges may be directed (they have an arrow in a single direction) or undirected. A graph with all edges directed is called a *directed graph*, and one with all edges undirected is called an *undirected graph*.

2.2 Directed Graphs

Definition 9 (Path, Ancestors, Descendants[165])**.** A *path* $A \mapsto B$ from node A to node B is a sequence of vertices $A_0 = A, A_1, \ldots, A_{n-1}, A_n = B$, with (A_n, A_{n+1}) an edge in the graph, thereby connecting A to B. For a directed graph this means that a path is a sequence of nodes which when we follow the direction of the arrows leads us from A to B.

The vertices A such that $A \mapsto B$ and $B \not\mapsto A$ are the *ancestors* of B.

The vertices B such that $A \mapsto B$ and $B \not\mapsto A$ are the *descendants* of A.

Definition 10 (Directed Acyclic Graph (DAG))**.** A DAG is a graph G with directed edges (arrows on each link) between the vertices (nodes) such that by following a path of vertices from one node to another along the direction of each edge no path will revisit a vertex.

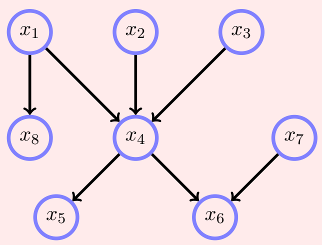
In a DAG the ancestors of B are those nodes who have a directed path ending at B. Conversely, the descendants of A are those nodes who have a directed path starting at A.

Definition 11 (Relationships in a DAG)**.**

Remark 2 (Graph Confusions)**.** Graphs are widely used, but differ markedly in what they represent. Here we try to cover two potential pitfalls.

State Transition Diagrams Such graphical representations are common in Markov Chains and Finite State Automata. A set of states is written as set of nodes(vertices) of a graph, and a directed edge between node i and node j (with an associated weight p_{ij}) represents that a transition from state i to state j can occur with probability p_{ij}. From the graphical models perspective we would simply write down a directed graph $x(t) \rightarrow x(t+1)$ to represent this Markov Chain. The state-transition diagram simply provides a graphical description of the conditional probability table $p(x(t+1)|x(t))$.

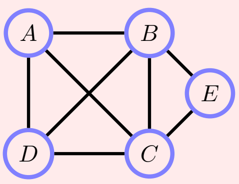
Neural Networks Neural networks also have vertices and edges. In general, however, neural networks are graphical representations of *functions*, whereas as graphical models are representations of distributions (a richer formalism). Neural networks (or any other parametric description) may be used to represent the conditional probability tables, as in sigmoid belief networks[202].



The *parents* of x_4 are $\text{pa}(x_4) = \{x_1, x_2, x_3\}$. The *children* of x_4 are $\text{ch}(x_4) = \{x_5, x_6\}$. The *family* of a node is itself and its parents. The *Markov blanket* of a node is itself, its parents, children and the parents of its children. In this case, the Markov blanket of x_4 is $x_1, x_2, \ldots, x_7$.

2.3 Undirected Graphs

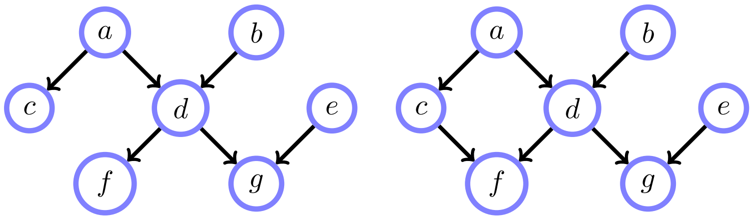
Definition 12 (Undirected graph)**.**



An undirected graph G consists of undirected edges between nodes.

Definition 13 (Neighbour)**.** For an undirected graph G the neighbours of x, $\text{ne}(x)$ are those nodes directly connected to x.

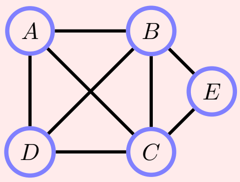
Definition 14 (Connected graph)**.** An undirected graph is *connected* if there is a path between every set of vertices (*i.e.* there are no isolated islands). For a graph which is not connected, the connected components are those subgraphs which are connected.



(a) (b)

Figure 2.1: (a) Singly-Connected graph. (b) Multiply-Connected graph.

Definition 15 (Clique)**.**



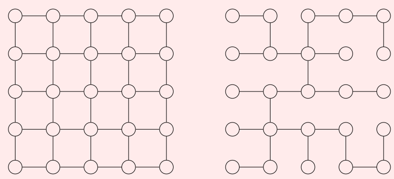
Given an undirected graph, a clique is a maximally connected subset of vertices. All the members of the clique are connected to each other; furthermore there is no larger clique that can be made from a clique. For example this graph has two cliques, $\mathcal{C}_1 = \{A, B, C, D\}$ and $\mathcal{C}_2 = \{B, C, E\}$. Whilst A, B, C are fully connected, this is a non-maximal clique since there is a larger fully connected set, A, B, C, D that contains this. A non-maximal clique is sometimes called a *cliquo*.

2.4 Graph Structure

Definition 16 (Singly-Connected Graph)**.** A graph is *singly-connected* if there is only one path from a vertex a to another vertex b. Otherwise the graph is *multiply-connected*. This definition applies regardless of whether or not the edges in the graph are directed. An alternative name for a singly-connected graph is a *tree*. A multiply-connected graph is also called *loopy*.

2.5 Spanning Tree

Definition 17 (Spanning Tree)**.**
A spanning tree of an undirected graph G is a singly-connected subset of the existing edges such that the resulting singly-connected graph covers all vertices of G. On the right is a a graph and an associated spanning tree. A maximum weight spanning tree is a spanning tree such that the sum of all weights on the edges of the tree is larger than for any other spanning tree of G.



Finding the maximal weight spanning tree

A simple algorithm to find a spanning tree with maximal weight is as follows: Start by picking the edge with the largest weight, and add this to the edge set. Then pick the next candidate edge which has the largest weight and add this to the edge set – if this results in an edge set with cycles, then reject the candidate edge, and find the next largest edge weight. Note that there may be more than one maximal weight spanning tree.

2.6 Numerically encoding Graphs

To express the structure of GMs we need to numerically encode the links on the graphs. For a graph of N vertices, we can describe the graph structure in various equivalent ways.

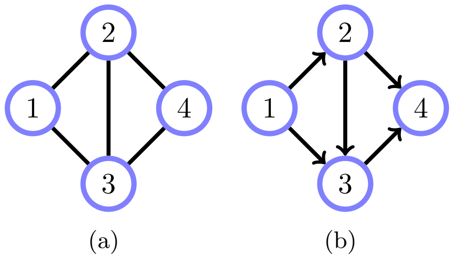


Figure 2.2: **(a)**: An undirected graph can be represented as a symmetric adjacency matrix. **(b)**: A directed graph with vertices labelled in ancestral order corresponds to a triangular adjacency matrix.

2.6.1 Edge List

As the name suggests, an *edge list* simply lists which vertex-vertex pairs are in the graph. For fig(2.2a), an edge list is $L = \{(1,2),(2,1),(1,3),(3,1),(2,3),(3,2),(2,4),(4,2),(3,4),(4,3)\}$ where an undirected edge is represented by a bidirectional edge.

2.6.2 Adjacency matrix

An alternative is to use an *adjacency matrix*

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \tag{2.6.1}$$

where $A_{ij} = 1$ if there is an edge from variable i to j in the graph, and 0 otherwise. Some authors include self-connections in this definition. An undirected graph has a symmetric adjacency matrix.

Provided that the vertices are labelled in *ancestral order* (parents always come before children) a directed graph fig(2.2b) can be represented as a triangular adjacency matrix:

$$\mathbf{T} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{2.6.2}$$

Adjacency Matrix powers

For an $N \times N$ adjacency matrix $\mathbf{A}$, powers of the adjacency matrix $\left[\mathbf{A}^k\right]_{ij}$ specify how many paths there are from node i to node j in k edge hops.

If we include 1's on the diagonal of $\mathbf{A}$ then $\left[\mathbf{A}^N\right]_{ij}$ is non-zero when there is a path connecting j to i in the graph. If $\mathbf{A}$ corresponds to a DAG the non-zero entries of the j^{th} row of $\left[\mathbf{A}^N\right]$ correspond to the descendants of node j.

2.6.3 Clique matrix

For an undirected graph with N vertices and maximal cliques $C_1, \ldots, C_K$ a clique matrix is an $n \times K$ matrix in which each column c_k has zeros expect for ones on entries describing the clique. A cliquo matrix relaxes the constraint that cliques are required to be maximal[1]. For example

$$\mathbf{C} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 1 \end{pmatrix} \tag{2.6.3}$$

[1]The term 'cliquo' for a non-maximal clique is attributed to Julian Besag.

is a clique matrix for fig(2.2a). A cliquo matrix containing only two-dimensional maximal cliques is called an *incidence matrix*. For example

$$\mathbf{C}_{inc} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix} \tag{2.6.4}$$

is an incidence matrix for fig(2.2b).

It is straightforward to show that $\mathbf{C}_{inc}\mathbf{C}_{inc}^{\mathsf{T}}$ is equal to the adjacency matrix except that the diagonals now contain the *degree* of each vertex (the number of edges it touches). Similarly, for any cliquo matrix the diagonal entry of $[\mathbf{C}\mathbf{C}^{\mathsf{T}}]_{ii}$ expresses the number of cliquos (columns) that vertex i occurs in. Off diagonal elements $[\mathbf{C}\mathbf{C}^{\mathsf{T}}]_{ij}$ contain the number of cliquos that vertices i and j jointly inhabit.

2.6.4 Utility Routines

`ancestors.m`: Find the ancestors of a node in a DAG
`edges.m`: Edge list from an adjacency matrix
`ancestralorder.m`: Ancestral order from a DAG
`connectedComponents.m`: Connected Components

`parents.m`: Parents of a node given an adjacency matrix
`children.m`: Children of a node given an adjacency matrix
`neigh.m`: Neighbours of a node given an adjacency matrix

A connected graph is a tree if the number of edges plus 1 is equal to the number of nodes. However, for a possibly disconnected graph this is not the case. The code below deals with the possibly disconnected case. The routine is based on the observation that any singly-connected graph must always possess a simplical node (a leaf node) which can be eliminated to reveal a smaller singly-connected graph.
`istree.m`: If graph is singly connected return 1 and elimination sequence
`spantree.m`: Return a spanning tree from an ordered edge list
`singleparenttree.m`: Find a directed tree with at most one parent from an undirected tree

Additional routines for basic manipulations in graphs are given at the end of chapter(6).

2.7 Exercises

Exercise 17. *Consider an adjacency matrix* $\mathbf{A}$ *with elements* $[\mathbf{A}]_{ij} = 1$ *if one can reach state* i *from state* j *in one timestep, and 0 otherwise. Show that the matrix* $\left[\mathbf{A}^k\right]_{ij}$ *represents the number of paths that lead from state* j *to* i *in* k *timesteps. Hence derive an algorithm that will find the minimum number of steps to get from state* j *to state* i.

Exercise 18. *For an* $N \times N$ *symmetric adjacency matrix* $\mathbf{A}$*, describe an algorithm to find the connected components. You may wish to examine* `connectedComponents.m`.

Exercise 19. *Show that for a connected graph that is singly-connected, the number of edges* E *must be equal to the number of vertices minus 1,* $E = V - 1$. *Give an example graph with* $E = V - 1$ *that is not singly-connected.*

Exercise 20. *Given an undirected graph with adjacency matrix* $\mathbf{A}$*, describe a procedure to find a spanning tree of the graph.*

CHAPTER 3

Belief Networks

3.1 Probabilistic Inference in structured distributions

Consider an environment composed of N variables, with a corresponding distribution $p(x_1, \ldots, x_N)$. Writing $\mathcal{E}$ as the set of evidential variables and using $\mathsf{evidence} = \{x_e = \mathsf{x}_e, e \in \mathcal{E}\}$ to denote all available evidence, then inference and reasoning can be carried out automatically by the 'brute force' method[1]

$$p(x_i = \mathsf{x}_i|\mathsf{evidence}) = \frac{\sum_{x_{\{\backslash\mathcal{E},\backslash i\}}} p(\{x_e = \mathsf{x}_e, e \in \mathcal{E}\}, x_i = \mathsf{x}_i, x_{\{\backslash\mathcal{E},\backslash i\}})}{\sum_{x_{\backslash\mathcal{E}}} p(\{x_e = \mathsf{x}_e, e \in \mathcal{E}\}, x_{\backslash\mathcal{E}})} \tag{3.1.1}$$

If all variables are binary (take two states), these summations require $O(2^{N-|\mathcal{E}|})$ operations. Such exponential computation is impractical and techniques that reduce this burden by exploiting any structure in the joint probability table are the topic of our discussions on efficient inference.

Naively specifying all the entries of a table $p(x_1, \ldots, x_N)$ over binary variables x_i takes $O(2^N)$ space. We will need to deal with large numbers of variables in machine learning and related application areas, with distributions on potentially hundreds if not millions of variables. The only way to deal with such large distributions is to constrain the nature of the variable interactions in some manner, both to render specification and ultimately inference in such systems tractable. The key idea is to specify which variables are independent of others, leading to a structured factorisation of the joint probability distribution. Belief Networks are a convenient framework for representing such factorisations into local conditional distributions. We will discuss Belief Networks more formally in section(3.3), first discussing their natural graphical representations of distributions.

Definition 18 (Belief Network)**.** A Belief Network is a distribution of the form

$$p(x_1, \ldots, x_D) = \prod_{i=1}^{D} p(x_i|\mathrm{pa}(x_i)) \tag{3.1.2}$$

where $\mathrm{pa}(x_i)$ represent the *parental* variables of variable x_i. Written as a Directed Graph, with an arrow pointing from a parent variable to child variable, a Belief Network is a Directed Acyclic Graph (DAG), with the i^{th} vertex in the graph corresponding to the factor $p(x_i|\mathrm{pa}(x_i))$.

[1]The extension to continuous variables is straightforward, replacing summation with integration over pdfs; we defer treatment of this to later chapters, since our aim is to here outline more the intuitions without needing to deal with integration of high dimensional distributions.

3.2 Graphically representing distributions

Belief Networks (also called Bayes' Networks or Bayesian Belief Networks) are a way to depict the independence assumptions made in a distribution [146, 165]. Their application domain is widespread, ranging from troubleshooting[50] and expert reasoning under uncertainty to machine learning. Before we more formally define a BN, an example will help motivate the development[2].

3.2.1 Constructing a simple Belief Network : Wet Grass

One morning Tracey leaves her house and realises that her grass is wet. Is it due to overnight rain or did she forget to turn off the sprinkler last night? Next she notices that the grass of her neighbour, Jack, is also wet. This *explains away* to some extent the possibility that her sprinkler was left on, and she concludes therefore that it has probably been raining.

Making a model

We can model the above situation using probability by following a general modelling approach. First we define the variables we wish to include in our model. In the above situation, the natural variables are

$R \in \{0, 1\}$ ($R = 1$ means that it has been raining, and 0 otherwise).

$S \in \{0, 1\}$ ($S = 1$ means that Tracey has forgotten to turn off the sprinkler, and 0 otherwise).

$J \in \{0, 1\}$ ($J = 1$ means that Jack's grass is wet, and 0 otherwise).

$T \in \{0, 1\}$ ($T = 1$ means that Tracey's Grass is wet, and 0 otherwise).

A model of Tracey's world then corresponds to a probability distribution on the joint set of the variables of interest $p(T, J, R, S)$ (the order of the variables is irrelevant).

Since each of the variables in this example can take one of two states, it would appear that we naively have to specify the values for each of the $2^4 = 16$ states, *e.g.* $p(T = 1, J = 0, R = 1, S = 1) = 0.7$ *etc.* However, since there are normalisation conditions for probabilities, we do not need to specify all the state probabilities. To see how many states need to be specified, consider the following decomposition. Without loss of generality (wlog) and repeatedly using Bayes' rule, we may write

$$p(T, J, R, S) = p(T|J, R, S)p(J, R, S) \tag{3.2.1}$$

$$= p(T|J, R, S)p(J|R, S)p(R, S) \tag{3.2.2}$$

$$= p(T|J, R, S)p(J|R, S)p(R|S)p(S) \tag{3.2.3}$$

That is, we may write the joint distribution as a product of conditional distributions. The first term $p(T|J, R, S)$ requires us to specify $2^3 = 8$ values – we need $p(T = 1|J, R, S)$ for the 8 joint states of J, R, S. The other value $p(T = 0|J, R, S)$ is given by normalisation : $p(T = 0|J, R, S) = 1 - p(T = 1|J, R, S)$. Similarly, we need $4 + 2 + 1$ values for the other factors, making a total of 15 values in all. In general, for a distribution on n binary variables, we need to specify $2^n - 1$ values in the range $[0, 1]$. The important point here is that the number of values that need to be specified in general scales exponentially with the number of variables in the model – this is impractical in general and motivates simplifications.

Conditional Independence

The modeler often knows constraints on the system. For example, in the scenario above, we may assume that Tracey's grass is wet depends only directly on whether or not is has been raining and whether or not her sprinkler was on. That is, we make a *conditional independence* assumption

$$p(T|J, R, S) = p(T|R, S) \tag{3.2.4}$$

[2] The scenario is adapted from [216].

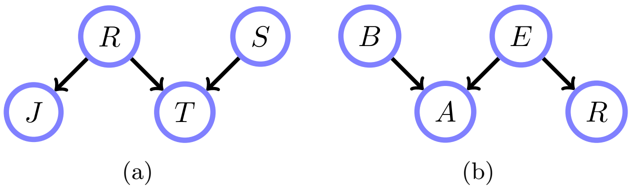


Figure 3.1: **(a)**: Belief network structure for the 'wet grass' example. Each node in the graph represents a variable in the joint distribution, and the variables which feed in (the parents) to another variable represent which variables are to the right of the conditioning bar. **(b)**: BN for the Burglar model.

Similarly, since whether or not Jack's grass is wet is influenced only directly by whether or not it has been raining, we write

$$p(J|R, S) = p(J|R) \tag{3.2.5}$$

and since the rain is not directly influenced by the sprinkler,

$$p(R|S) = p(R) \tag{3.2.6}$$

which means that our model now becomes :

$$p(T, J, R, S) = p(T|R, S)p(J|R)p(R)p(S) \tag{3.2.7}$$

We can represent these conditional independencies graphically, as in fig(3.1a). This reduces the number of values that we need to specify to $4 + 2 + 1 + 1 = 8$, a saving over the previous 15 values in the case where no conditional independencies had been assumed.

To complete the model, we need to numerically specify the values of each conditional probability table (CPT). Let the prior probabilities for R and S be $p(R = 1) = 0.2$ and $p(S = 1) = 0.1$. We set the remaining probabilities to $p(J = 1|R = 1) = 1$, $p(J = 1|R = 0) = 0.2$ (sometimes Jack's grass is wet due to unknown effects, other than rain), $p(T = 1|R = 1, S) = 1$, $p(T = 1|R = 0, S = 1) = 0.9$ (there's a small chance that even though the sprinkler was left on, it didn't wet the grass noticeably), $p(T = 1|R = 0, S = 0) = 0$.

Inference

Now that we've made a model of an environment, we can perform inference. Let's calculate the probability that the sprinkler was on overnight, given that Tracey's grass is wet: $p(S = 1|T = 1)$.
To do this, we use Bayes' rule:

$$p(S = 1|T = 1) = \frac{p(S = 1, T = 1)}{p(T = 1)} = \frac{\sum_{J,R} p(T = 1, J, R, S = 1)}{\sum_{J,R,S} p(T = 1, J, R, S)} \tag{3.2.8}$$

$$= \frac{\sum_{J,R} p(J|R)p(T = 1|R, S = 1)p(R)p(S = 1)}{\sum_{J,R,S} p(J|R)p(T = 1|R, S)p(R)p(S)} \tag{3.2.9}$$

$$= \frac{\sum_{R} p(T = 1|R, S = 1)p(R)p(S = 1)}{\sum_{R,S} p(T = 1|R, S)p(R)p(S)} \tag{3.2.10}$$

$$= \frac{0.9 \times 0.8 \times 0.1 + 1 \times 0.2 \times 0.1}{0.9 \times 0.8 \times 0.1 + 1 \times 0.2 \times 0.1 + 0 \times 0.8 \times 0.9 + 1 \times 0.2 \times 0.9} = 0.3382 \tag{3.2.11}$$

so the belief that the sprinkler is on increases above the prior probability 0.1, due to the fact that the grass is wet.

Let us now calculate the probability that Tracey's sprinkler was on overnight, given that her grass is wet and that Jack's grass is also wet, $p(S = 1|T = 1, J = 1)$. We use Bayes' rule again:

$$p(S = 1|T = 1, J = 1) = \frac{p(S = 1, T = 1, J = 1)}{p(T = 1, J = 1)} \tag{3.2.12}$$

$$= \frac{\sum_R p(T = 1, J = 1, R, S = 1)}{\sum_{R,S} p(T = 1, J = 1, R, S)} \tag{3.2.13}$$

$$= \frac{\sum_R p(J = 1|R)p(T = 1|R, S = 1)p(R)p(S = 1)}{\sum_{R,S} p(J = 1|R)p(T = 1|R, S)p(R)p(S)} \tag{3.2.14}$$

$$= \frac{0.0344}{0.2144} = 0.1604 \tag{3.2.15}$$

The probability that the sprinkler is on, given the extra evidence that Jack's grass is wet, is *lower* than the probability that the grass is wet given only that Tracey's grass is wet. That is, that the grass is wet due to the sprinkler is (partly) explained away by the fact that Jack's grass is also wet – this increases the chance that the rain has played a role in making Tracey's grass wet.

Naturally, we don't wish to carry out such inference calculations by hand all the time. General purpose algorithms exist for this, such as the Junction Tree Algorithm, and we shall introduce these in later chapters.

Example 10 (Was it the Burglar?)**.** Here's another example using binary variables, adapted from [216]. Sally comes home to find that the burglar alarm is sounding ($A = 1$). Has she been burgled ($B = 1$), or was the alarm triggered by an earthquake ($E = 1$)? She turns the car radio on for news of earthquakes and finds that the radio broadcasts an earthquake alert ($R = 1$).

Using Bayes' rule, we can write, without loss of generality,

$$p(B, E, A, R) = p(A|B, E, R)p(B, E, R) \tag{3.2.16}$$

We can repeat this for $p(B, E, R)$, and continue

$$p(B, E, A, R) = p(A|B, E, R)p(R|B, E)p(E|B)p(B) \tag{3.2.17}$$

However, the alarm is surely not directly influenced by any report on the Radio – that is, $p(A|B, E, R) = p(A|B, E)$. Similarly, we can make other conditional independence assumptions such that

$$p(B, E, A, R) = p(A|B, E)p(R|E)p(E)p(B) \tag{3.2.18}$$

Specifying Conditional Probability Tables

Alarm = 1	Burglar	Earthquake
0.9999	1	1
0.99	1	0
0.99	0	1
0.0001	0	0

Radio = 1	Earthquake
1	1
0	0

The remaining tables are $p(B = 1) = 0.01$ and $p(E = 1) = 0.000001$. The tables and graphical structure fully specify the distribution. Now consider what happens as we observe evidence.

Initial Evidence: The Alarm is sounding

$$p(B = 1|A = 1) = \frac{\sum_{E,R} p(B = 1, E, A = 1, R)}{\sum_{B,E,R} p(B, E, A = 1, R)} \tag{3.2.19}$$

$$= \frac{\sum_{E,R} p(A = 1|B = 1, E)p(B = 1)p(E)p(R|E)}{\sum_{B,E,R} p(A = 1|B, E)p(B)p(E)p(R|E)} \approx 0.99 \tag{3.2.20}$$

Additional Evidence: The Radio broadcasts an Earthquake warning: A similar calculation gives $p(B = 1|A = 1, R = 1) \approx 0.01$. Thus, initially, because the Alarm sounds, Sally thinks that she's been burgled. However, this probability drops dramatically when she hears that there has been an Earthquake. That is, the Earthquake 'explains away' to an extent the fact that the Alarm is ringing. See `demoBurglar.m`.

3.2.2 Uncertain Evidence

In soft or *uncertain evidence*, the variable is in more than one state, with the strength of our belief about each state being given by probabilities. For example, if x has the states $\text{dom}(x) = \{\mathsf{red}, \mathsf{blue}, \mathsf{green}\}$ the vector $(0.6, 0.1, 0.3)$ represents the probabilities of the respective states. In contrast, for *hard evidence* we are certain that a variable is in a particular state. In this case, all the probability mass is in one of the vector components, for example $(0, 0, 1)$.

Performing inference with soft-evidence is straightforward and can be achieved using Bayes' rule. Writing the soft evidence as $\tilde{y}$, we have

$$p(x|\tilde{y}) = \sum_y p(x|y)p(y|\tilde{y}) \tag{3.2.21}$$

where $p(y = \mathsf{i}|\tilde{y})$ represents the probability that y is in state i under the soft-evidence. This is a generalisation of hard-evidence in which the vector $p(y|\tilde{y})$ has all zero component values, except for all but a single component.

Note that the soft evidence $p(y = \mathsf{i}|\tilde{y})$ does not correspond to the marginal $p(y = \mathsf{i})$ in the original joint distribution $p(x, y)$. A procedure to form a joint distribution, known as *Jeffrey's rule* is to begin with an original distribution $p_1(x, y)$, from which we can define

$$p_1(x|y) = \frac{p_1(x, y)}{\sum_x p_1(x, y)} \tag{3.2.22}$$

Using the soft evidence $p(y|\tilde{y})$ we then define a new joint distribution

$$p_2(x, y|\tilde{y}) = p_1(x|y)p(y|\tilde{y}) \tag{3.2.23}$$

In the BN we use a dashed circle to represent that a variable is in a soft-evidence state.

Example 11 (soft-evidence)**.** Revisiting the earthquake scenario, example(10), imagine that we think we hear the burglar alarm sounding, but are not sure, specifically we are only 70% sure we heard the alarm. For this binary variable case we represent this soft-evidence for the states $(\mathsf{1}, \mathsf{0})$ as $\tilde{A} = (0.7, 0.3)$. What is the probability of a burglary under this soft-evidence?

$$p(B = \mathsf{1}|\tilde{A}) = \sum_A p(B = \mathsf{1}|A)p(A|\tilde{A}) = p(B = \mathsf{1}|A = \mathsf{1}) \times 0.7 + p(B = \mathsf{1}|A = \mathsf{0}) \times 0.3 \tag{3.2.24}$$

The probabilities $p(B = \mathsf{1}|A = \mathsf{1}) \approx 0.99$ and $p(B = \mathsf{1}|A = \mathsf{0}) \approx 0.0001$ are calculated using Bayes' rule as before to give

$$p(B = \mathsf{1}|\tilde{A}) \approx 0.6930 \tag{3.2.25}$$

Uncertain evidence versus unreliable modelling

An entertaining example of uncertain evidence is given by Pearl[216]:

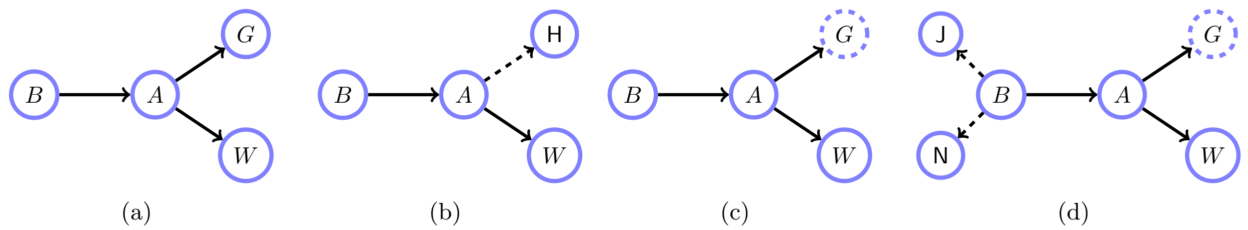


Figure 3.2: **(a)**: Mr Holmes' burglary worries as given in [216]: (B)urglar, (A)larm, (W)atson, Mrs (Gibbon). **(b)**: Virtual Evidence can be represented by a dashed line. **(c)**: Modified problem. Mrs Gibbon is not drinking but somewhat deaf; we represent such uncertain (soft-evidence) by a circle. **(d)**: Holmes gets additional information from his neighbour Mrs (N)osy and informant Dodgy (J)oe.

> Mr Holmes receives a telephone call from his neighbour Dr Watson, who states that he hears the sound of a burglar alarm from the direction of Mr Holmes' house. While preparing to rush home, Mr Holmes recalls that Dr Watson is known to be a tasteless practical joker, and he decides to first call another neighbour, Mrs Gibbon who, despite occasional drinking problems, is far more reliable.
>
> When Mr Holmes calls Mrs Gibbon, he soon realises that she is somewhat tipsy. Instead of answering his question directly, she goes on and on about her latest back operation and about how terribly noisy and crime-ridden the neighbourhood has become. When he finally hangs up, all Mr Holmes can glean from the conversation is that there is probably an 80% chance that Mrs Gibbon did hear an alarm sound from her window.

A BN for this scenario is depicted in fig(3.2a) which deals with four binary variables: House is (B)urgled, (A)larm has sounded, (W)atson hears alarm, and Mrs (G)ibbon hears alarm[3]:

$$p(B, A, G, W) = p(A|B)p(B)p(W|A)p(G|A) \tag{3.2.26}$$

Holmes is interested in the likelihood that his house has been burgled. Naively, Holmes' might calculate[4]

$$p(B = \mathsf{tr}|W = \mathsf{tr}, G = \mathsf{tr}) \tag{3.2.27}$$

However, after finding out about Mrs Gibbon's state, Mr Holmes no longer finds the above model reliable. He wants to ignore the effect that Mrs Gibbon's evidence has on the inference, and replace it with his own belief as to what Mrs Gibbon observed. Mr Holmes can achieve this by replacing the term $p(G = \mathsf{tr}|A)$ by a so-called *virtual evidence* term

$$p(G = \mathsf{tr}|A) \to p(\mathsf{H}|A), \qquad \text{where} \quad p(\mathsf{H}|A) = \begin{cases} 0.8 & A = \mathsf{tr} \\ 0.2 & A = \mathsf{fa} \end{cases} \tag{3.2.28}$$

Here the state H is arbitrary and fixed. This is used to modify the joint distribution to

$$p(B, A, \mathsf{H}, W) = p(A|B)p(B)p(W|A)p(\mathsf{H}|A), \tag{3.2.29}$$

see fig(3.2b). When we then compute $p(B = \mathsf{tr}|W = \mathsf{tr}, \mathsf{H})$ the effect of Mr Holmes' judgement will count for a factor of 4 times more in favour of the Alarm sounding than not. The values of the table entries are irrelevant up to normalisation since any constants can be absorbed into the proportionality constant. Note also that $p(\mathsf{H}|A)$ is not a distribution in A, and hence no normalisation is required. This form of evidence is also called *likelihood evidence*.

A twist on Pearl's scenario is that Mrs Gibbon has not been drinking. However, she is a little deaf and cannot be sure herself that she heard the alarm. She is 80% sure she heard it. In this case, Holmes would

[3]One might be tempted to include an additional (T)ipsy variable as a parent of G. This would then require us to specify the joint distribution $p(G|T, A)$ for the 4 parental joint states of T and A. Here we assume that we do not have access to such information.

[4]The notation tr is equivalent to $\mathsf{1}$ and fa to $\mathsf{0}$ from example(10).

trust the model – however, the observation itself is now uncertain, fig(3.2c). This can be dealt with using the soft evidence technique. From Jeffrey's rule, one uses the original model equation (3.2.26) to compute

$$p(B = \mathsf{tr}|W = \mathsf{tr}, G) = \frac{p(B = \mathsf{tr}, W = \mathsf{tr}, G)}{p(W = \mathsf{tr}, G)} = \frac{\sum_A p(G|A)p(W = \mathsf{tr}|A)p(A|B = \mathsf{tr})}{\sum_{B,A} p(G|A)p(W = \mathsf{tr}|A)p(A|B)} \tag{3.2.30}$$

and then uses the soft-evidence

$$p(G|\tilde{G}) = \begin{cases} 0.8 & G = \mathsf{tr} \\ 0.2 & G = \mathsf{fa} \end{cases} \tag{3.2.31}$$

to compute

$$p(B = \mathsf{tr}|W = \mathsf{tr}, \tilde{G}) = p(B = \mathsf{tr}|W = \mathsf{tr}, G = \mathsf{tr})p(G = \mathsf{tr}|\tilde{G}) + p(B = \mathsf{tr}|W = \mathsf{tr}, G = \mathsf{fa})p(G = \mathsf{fa}|\tilde{G}) \tag{3.2.32}$$

The reader may show that an alternative way to represent an uncertain observation such as Mrs Gibbon being non-tipsy but hard-of-hearing above is to use a virtual evidence child from G.

Uncertain evidence within an unreliable model

To highlight uncertain evidence in an unreliable model we introduce two additional characters. Mrs Nosy lives next door to Mr Holmes and is completely deaf, but nevertheless an incorrigible curtain-peeker who seems to notice most things. Unfortunately, she's also rather prone to imagining things. Based on his conversation with her, Mr Holmes counts her story as 3 times in favour of there not being a burglary to there being a burglary, and therefore uses a virtual evidence term

$$p(\mathsf{Nosy}|B) = \begin{cases} 1 & B = \mathsf{tr} \\ 3 & B = \mathsf{fa} \end{cases} \tag{3.2.33}$$

Mr Holmes also telephones Dodgy Joe, his contact in the criminal underworld to see if he's heard of any planned burglary on Mr Holmes' home. He summarises this information using a virtual evidence term

$$p(\mathsf{Joe}|B) = \begin{cases} 1 & B = \mathsf{tr} \\ 5 & B = \mathsf{fa} \end{cases} \tag{3.2.34}$$

When all this information is combined : Mrs Gibbon is not tipsy but somewhat hard of hearing, Mrs Nosy, and Dodgy Joe, we first deal with the unreliable model

$$p(B, A, W = \mathsf{tr}, G, \mathsf{Nosy}, \mathsf{Joe}) \propto p(B)p(\mathsf{Nosy}|B)p(\mathsf{Joe}|B)p(A|B)p(W = \mathsf{tr}|A)p(G|A) \tag{3.2.35}$$

from which we can compute

$$p(B = \mathsf{tr}|W = \mathsf{tr}, G, \mathsf{Nosy}, \mathsf{Joe}) \tag{3.2.36}$$

Finally we perform inference with the soft-evidence

$$p(B = \mathsf{tr}|W = \mathsf{tr}, \tilde{G}, \mathsf{Nosy}, \mathsf{Joe}) = \sum_G p(B = \mathsf{tr}|W = \mathsf{tr}, G, \mathsf{Nosy}, \mathsf{Joe})p(G|\tilde{G}) \tag{3.2.37}$$

An important consideration above is that the virtual evidence does not replace the prior $p(B)$ with another prior distribution – rather the virtual evidence terms modify the prior through the inclusion of extra factors. The usual assumption is that each virtual evidence acts independently, although one can consider dependent scenarios if required.

(a) (b)

Figure 3.3: Two BNs for a 4 variable distribution. Both graphs (a) and (b) represent the *same* distribution $p(x_1, x_2, x_3, x_4)$. Strictly speaking they represent the same (lack of) independence assumptions – the graphs say nothing about the content of the CPTs. The extension of this 'cascade' to many variables is clear and always results in a Directed Acyclic Graph.

3.3 Belief Networks

In the Wet Grass and Burglar examples, we had a choice as to how we recursively used Bayes' rule. In a general 4 variable case we could choose the factorisation,

$$p(x_1, x_2, x_3, x_4) = p(x_1|x_2, x_3, x_4)p(x_2|x_3, x_4)p(x_3|x_4)p(x_4) \tag{3.3.1}$$

An equally valid choice is (see fig(3.3))

$$p(x_1, x_2, x_3, x_4) = p(x_3|x_4, x_1, x_2)p(x_4|x_1, x_2)p(x_1|x_2)p(x_2). \tag{3.3.2}$$

In general, two different graphs may represent the same independence assumptions, as we will discuss further in section(3.3.1). If one wishes to make independence assumptions, then the choice of factorisation becomes significant.

The observation that any distribution may be written in the *cascade* form, fig(3.3), gives an algorithm for constructing a BN on variables $x_1, \ldots, x_n$: write down the $n-$variable cascade graph; assign any ordering of the variables to the nodes; you may then delete any of the directed connections. More formally, this corresponds to an ordering of the variables which, without loss of generality, we may write as $x_1, \ldots, x_n$. Then, from Bayes' rule, we have

$$\begin{aligned} p(x_1, \ldots, x_n) &= p(x_1|x_2, \ldots, x_n)p(x_2, \ldots, x_n) && (3.3.3)\\ &= p(x_1|x_2, \ldots, x_n)p(x_2|x_3, \ldots, x_n)p(x_3, \ldots, x_n) && (3.3.4)\\ &= p(x_n)\prod_{i=1}^{n-1} p(x_i|x_{i+1}, \ldots, x_n) && (3.3.5) \end{aligned}$$

The representation of any BN is therefore a *Directed Acyclic Graph (DAG)*.

Every probability distribution can be written as a BN, even though it may correspond to a fully connected 'cascade' DAG. The particular role of a BN is that the structure of the DAG corresponds to a set of conditional independence assumptions, namely which parental variables are sufficient to specify each conditional probability table. Note that this does not mean that non-parental variables have no influence. For example, for distribution $p(x_1|x_2)p(x_2|x_3)p(x_3)$ with DAG $x_1 \leftarrow x_2 \leftarrow x_3$, this does not imply $p(x_2|x_1, x_3) = p(x_2|x_3)$. The DAG specifies conditional independence statements of variables on their ancestors – namely which ancestors are 'causes' for the variable.

The DAG corresponds to a statement of conditional independencies in the model. To complete the specification of the BN we need to define all elements of the conditional probability tables $p(x_i|\text{pa}(x_i))$. Once the graphical structure is defined, the entries of the conditional probability tables (CPTs) $p(x_i|\text{pa}(x_i))$ can be expressed. For every possible state of the parental variables $\text{pa}(x_i)$, a value for each of the states of x_i needs to be specified (except one, since this is determined by normalisation). For a large number of parents, writing out a table of values is intractable, and the tables are usually parameterised in a low dimensional manner. This will be a central topic of our discussion on the application of BNs in machine learning.

3.3.1 Conditional Independence

Whilst a BN corresponds to a set of conditional independence assumptions, it is not always immediately clear from the DAG whether a set of variables is conditionally independent of a set of other variables. For example, in fig(3.4) are x_1 and x_2 independent, given the state of x_4? The answer is yes, since we have

$$p(x_1, x_2|x_4) = \frac{1}{p(x_4)} \sum_{x_3} p(x_1, x_2, x_3, x_4) = \frac{1}{p(x_4)} \sum_{x_3} p(x_1|x_4)p(x_2|x_3, x_4)p(x_3)p(x_4) \tag{3.3.6}$$

$$= p(x_1|x_4) \sum_{x_3} p(x_2|x_3, x_4)p(x_3) \tag{3.3.7}$$

Now

$$p(x_2|x_4) = \frac{1}{p(x_4)} \sum_{x_1,x_3} p(x_1, x_2, x_3, x_4) = \frac{1}{p(x_4)} \sum_{x_1,x_3} p(x_1|x_4)p(x_2|x_3, x_4)p(x_3)p(x_4) \tag{3.3.8}$$

$$= \sum_{x_3} p(x_2|x_3, x_4)p(x_3) \tag{3.3.9}$$

Combining the two results above we have

$$p(x_1, x_2|x_4) = p(x_1|x_4)p(x_2|x_4) \tag{3.3.10}$$

so that x_1 and x_2 are indeed independent conditioned on x_4.

Definition 19 (Conditional Independence)**.**

$$\mathcal{X} \perp\!\!\!\perp \mathcal{Y} | \mathcal{Z} \tag{3.3.11}$$

denotes that the two sets of variables $\mathcal{X}$ and $\mathcal{Y}$ are independent of each other provided we know the state of the set of variables $\mathcal{Z}$. For full conditional independence, $\mathcal{X}$ and $\mathcal{Y}$ must be independent given *all* states of $\mathcal{Z}$. Formally, this means that

$$p(\mathcal{X}, \mathcal{Y}|\mathcal{Z}) = p(\mathcal{X}|\mathcal{Z})p(\mathcal{Y}|\mathcal{Z}) \tag{3.3.12}$$

for all states of $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$. In case the conditioning set is empty we may also write $\mathcal{X} \perp\!\!\!\perp \mathcal{Y}$ for $\mathcal{X} \perp\!\!\!\perp \mathcal{Y} | \emptyset$, in which case $\mathcal{X}$ is (unconditionally) independent of $\mathcal{Y}$.

If $\mathcal{X}$ and $\mathcal{Y}$ are *not* conditionally independent, they are conditionally dependent. This is written

$$\mathcal{X} \top\!\!\!\top \mathcal{Y} | \mathcal{Z} \tag{3.3.13}$$

To develop intuition about conditional independence consider the three variable distribution $p(x_1, x_2, x_3)$. We may write this in any of the 6 ways

$$p(x_1, x_2, x_3) = p(x_{i_1}|x_{i_2}, x_{i_3})p(x_{i_2}|x_{i_3})p(x_{i_3}) \tag{3.3.14}$$

where (i_1, i_2, i_3) is any of the 6 permutations of $(1, 2, 3)$. Whilst all different DAGs, they represent the same distribution, namely that which makes no conditional independence statements.

To make an independence statement, we need to drop one of the links. This gives rise to the 4 DAGs in fig(3.5). Are any of these graphs equivalent, in the sense that they represent the same distribution?

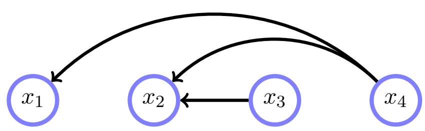


Figure 3.4: $p(x_1, x_2, x_3, x_4) = p(x_1|x_4)p(x_2|x_3, x_4)p(x_3)p(x_4)$.

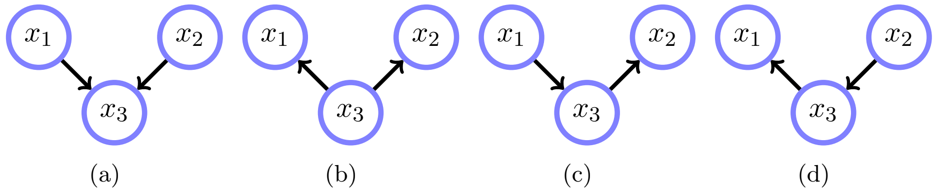


Figure 3.5: By dropping say the connection between variables x_1 and x_2, we reduce the 6 possible BN graphs amongst three variables to 4. (The 6 fully connected 'cascade' graphs correspond to (a) with $x_1 \to x_2$, (a) with $x_2 \to x_1$, (b) with $x_1 \to x_2$, (b) with $x_2 \to x_1$, (c) with $x_1 \to x_3$ and (d) with $x_3 \to x_1$. Any other graphs would be cyclic and therefore not distributions).

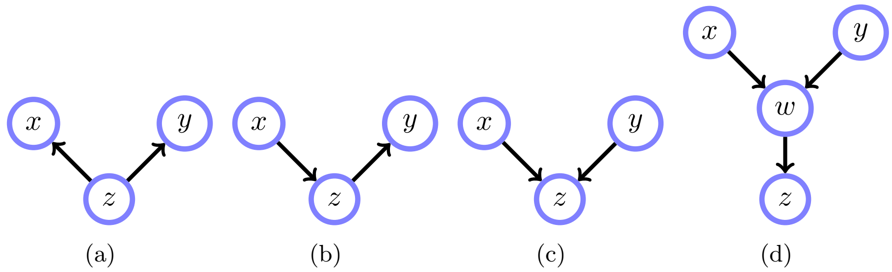


Figure 3.6: In graphs (a) and (b), variable z is not a collider. **(c)**: Variable z is a collider. Graphs (a) and (b) represent conditional independence $x \perp\!\!\!\perp y \mid z$. In graphs (c) and (d), x and y are conditionally dependent given variable z.

Applying Bayes' rule gives :

$$\underbrace{p(x_2|x_3)p(x_3|x_1)p(x_1)}_{graph(c)} = p(x_2, x_3)p(x_3, x_1)/p(x_3) = p(x_1|x_3)p(x_2, x_3) \tag{3.3.15}$$

$$= \underbrace{p(x_1|x_3)p(x_3|x_2)p(x_2)}_{graph(d)} = \underbrace{p(x_1|x_3)p(x_2|x_3)p(x_3)}_{graph(b)} \tag{3.3.16}$$

so that DAGs (b),(c) and (d) represent the same CI assumptions namely that, given the state of variable x_3, variables x_1 and x_2 are independent, $x_1 \perp\!\!\!\perp x_2 \mid x_3$.

However, graph (a) represents something fundamentally different, namely: $p(x_1, x_2) = p(x_1)p(x_2)$. There is no way to transform the distribution $p(x_3|x_1, x_2)p(x_1)p(x_2)$ into any of the others.

3.3.2 The impact of Collisions

In a general BN, how could we check if $x \perp\!\!\!\perp y \mid z$? In fig(3.6)(a,b), x and y are independent when conditioned on z. In fig(3.6)(c) they are dependent; in this situation, variable z is called a *collider* – the arrows of its neighbours are pointing towards it. What about fig(3.6)(d)? In (d), when we condition on z, then, in general, x and y will be dependent, since

$$p(x, y|z) = \frac{p(x, y, z)}{p(z)} = \frac{1}{p(z)} \sum_w p(z|w)p(w|x, y)p(x)p(y) \neq p(x|z)p(y|z) \tag{3.3.17}$$

– intuitively, variable w becomes dependent on the value of z, and since x and y are conditionally dependent on w, they are also conditionally dependent on z.

Roughly speaking, if there is a non-collider z which is conditioned on along the path between x and y (as in fig(3.6)(a,b)), then this path does not make x and y dependent. Similarly, if there is a path between x and y which contains a collider, provided that this collider is not in the conditioning set (and neither are any of its descendants) then this path does not make x and y dependent. If there is a path between x and

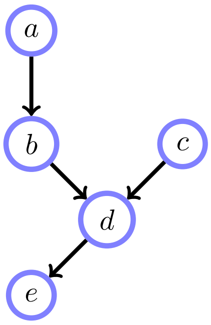


Figure 3.7: The variable d is a collider along the path $a-b-d-c$, but not along the path $a-b-d-e$. Is $a \perp\!\!\!\perp e | b$? a and b are *not* d-connected since there are no colliders on the only path between a and e, and since there is a non-collider b which is in the conditioning set. Hence a and b are d-separated, *i.e.* $a \perp\!\!\!\perp e | b$.

y which contains no colliders and no conditioning variables, then this path 'd-connects' x and y.

Note that a collider is defined *relative to a path*. In fig(3.7), the variable d is a collider along the path $a-b-d-c$, but not along the path $a-b-d-e$ (since, relative to this path, the two arrows do not point inwards to d).

Consider the BN: $A \to B \leftarrow C$. Here A and C are (unconditionally) independent. However, conditioning of B makes them dependent. Intuitively, whilst we believe the root causes are independent, given the value of the observation, this tells us something about the state of *both* the causes, coupling them and making them dependent.

3.3.3 d-Separation

The DAG concepts of d-separation and d-connection are central to determining conditional independence in any BN with structure given by the DAG[281].

Definition 20 (d-connection, d-separation)**.** If G is a directed graph in which $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$ are disjoint sets of vertices, then $\mathcal{X}$ and $\mathcal{Y}$ are d-connected by $\mathcal{Z}$ in G if and only if there exists an undirected path U between some vertex in $\mathcal{X}$ and some vertex in $\mathcal{Y}$ such that for every collider C on U, either C or a descendent of C is in $\mathcal{Z}$, and no non-collider on U is in $\mathcal{Z}$.

$\mathcal{X}$ and $\mathcal{Y}$ are d-separated by $\mathcal{Z}$ in G if and only if they are not d-connected by $\mathcal{Z}$ in G.

One may also phrase this as follows. For every variable $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, check every path U between x and y. A path U is said to be *blocked* if there is a node w on U such that either

1. w is a collider and neither w nor any of its descendants is in $\mathcal{Z}$.
2. w is not a collider on U and w is in $\mathcal{Z}$.

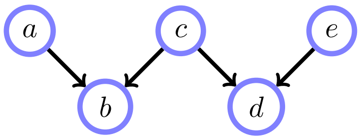
If all such paths are blocked then $\mathcal{X}$ and $\mathcal{Y}$ are d-separated by $\mathcal{Z}$.

If the variable sets $\mathcal{X}$ and $\mathcal{Y}$ are d-separated by $\mathcal{Z}$, they are independent conditional on $\mathcal{Z}$ in all probability distributions such a graph can represent.

The Bayes Ball algorithm[238] provides a linear time complexity algorithm which given a set of nodes $\mathcal{X}$ and $\mathcal{Z}$ determines the set of nodes $\mathcal{Y}$ such that $\mathcal{X} \perp\!\!\!\perp \mathcal{Y} | \mathcal{Z}$. $\mathcal{Y}$ is called the set of irrelevant nodes for $\mathcal{X}$ given $\mathcal{Z}$.

3.3.4 d-Connection and dependence

Given a DAG we can imply with certainty that two variables are (conditionally) independent, provided they are d-separated. Can we infer that they are dependent, provided they are d-connected? Consider the



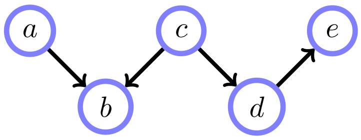


Figure 3.8: Examples for d-separation. **(a)**: b d-separates a from e. The joint variables $\{b, d\}$ d-connect a and e. **(b)**: c and e are (unconditionally) d-connected. b d-connects a and e.

following situation

$$p(a, b, c) = p(c|a, b)p(a)p(b) \tag{3.3.18}$$

for which we note that a and b are d-connected by c. For concreteness, we assume c is binary with states 1,2. The question is whether a and b are dependent, conditioned on c, $a \perp\!\!\!\perp b| c$. To answer this, consider

$$p(a, b|c = 1) = \frac{p(c = 1|a, b)p(a)p(b)}{\sum_{a,b} p(c = 1|a, b)p(a)p(b)} \tag{3.3.19}$$

In general, the first term $p(c = 1|a, b)$ does not need to be a factored function of a and b and therefore a and b are conditionally 'graphically' dependent. However, we can construct cases where this is not so. For example, let

$$p(c = 1|a, b) = \phi(a)\psi(b), \text{ and } p(c = 2|a, b) = 1 - p(c = 1|a, b) \tag{3.3.20}$$

where $\phi(a)$ and $\psi(b)$ are arbitrary potentials between 0 and 1. Then

$$p(a, b|c = 1) = \frac{1}{Z}\phi(a)p(a)\psi(b)p(b), \qquad Z = \sum_a \phi(a)p(a) \sum_b \psi(b)p(b) \tag{3.3.21}$$

which shows that $p(a, b|c = 1)$ is a product of a function in a and function in b, so that a and b are independent, conditioned on $c = 1$.

A second example is given by the distribution

$$p(a, b, c) = p(c|b)p(b|a)p(a) \tag{3.3.22}$$

in which a and c are d-connected by b. The question is, are a and c dependent, $a \perp\!\!\!\perp c| \emptyset$? For simplicity we assume b takes the two states 1,2. Then

$$p(a, c) = p(a) \sum_b p(c|b)p(b|a) = p(a) \left(p(c|b = 1)p(b = 1|a) + p(c|b = 2)p(b = 2|a)\right) \tag{3.3.23}$$

Using the shorthand $\phi(a) = p(b = 1|a)$, we can write this as

$$p(a, c) = p(a)\phi(a) \left[p(c|b = 1) + p(c|b = 2) \left(\frac{1}{\phi(a)} - 1\right)\right] \tag{3.3.24}$$

For the setting $\phi(a) = \gamma$, for some constant γ for all states of a, then

$$p(a, c) = p(a)\gamma \left(p(c|b = 1) + p(c|b = 2) \left(\frac{1}{\gamma} - 1\right)\right) \tag{3.3.25}$$

which is a product of a function of a and a function of c. Hence a and c are independent.

The moral of the story is that d-separation necessarily implies independence. However, d-connection does not necessarily imply dependence. It might be that there are numerical settings for which variables are independent, even though they are d-connected. For this reason we use the term 'graphical' dependence when the graph would suggest that variables are dependent, even though there may be numerical instantiations were dependence does not hold, see definition(21).

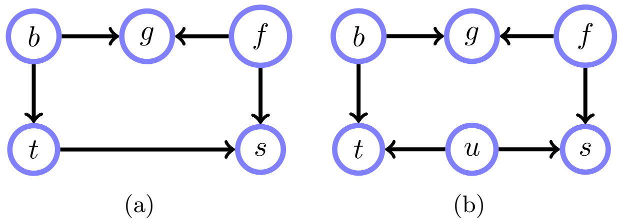


Figure 3.9: **(a)**: t and f are d-connected by g. **(b)**: b and f are d-separated by u.

Example 12. Consider fig(3.8a). Is $a \perp\!\!\!\perp e \mid b$? If we sum out variable d, then we see that a and e are independent given b, since the variable e will appear as an isolated factor independent of all other variables, hence indeed $a \perp\!\!\!\perp e \mid b$. Whilst b is a collider which is in the conditioning set, we need all colliders on the path to be in the conditioning set (or their descendants) for d-connectedness.

In fig(3.8b), if we sum out variable d, then c and e become intrinsically linked and $p(a, b, c, e)$ will not factorise into a function of a multiplied by a function of e – hence they are dependent.

Example 13. Consider the graph in fig(3.9a).

1. Are the variables t and f unconditionally independent, *i.e.* $t \perp\!\!\!\perp f \mid \emptyset$? Here there are two colliders, namely g and s – however, these are not in the conditioning set (which is empty), and hence they are d-separated and therefore unconditionally independent.

2. What about $t \perp\!\!\!\perp f \mid g$? There is a collider on the path between t and f which is in the conditioning set. Hence t and f are d-connected by g, and therefore t and f are not independent conditioned on g.

3. What about $b \perp\!\!\!\perp f \mid s$? Since there is a collider s in the conditioning set on the path between t and f, then b and f are conditionally dependent given s.

Example 14. Is $\{b, f\} \perp\!\!\!\perp u \mid \emptyset$? in fig(3.9b). Since the conditioning set is empty and every path from either b or f to u contains a collider, b, f are unconditionally independent of u.

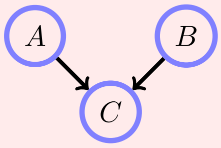
Definition 22 (Markov Equivalence)**.** Two graphs are Markov equivalent if they both represent the same set of conditional independence statements.

3.3.5 Markov Equivalence in Belief Networks

Define the *skeleton* of a graph as its undirected version with the directions on the arrows removed. Define an *immorality* in a DAG as a configuration of three nodes, A,B,C such that C is a child of both A and B, with A and B not directly connected. Two DAGs represent the same set of independence assumptions (they are *Markov equivalent*) if and only if they have the same skeleton and the same set of immoralities [73].

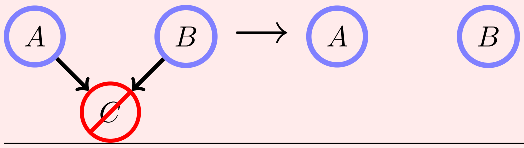
Using this rule we see that in fig(3.5), BNs (b,c,d) have the same skeleton with no immoralities and are therefore equivalent. However BN (a) has an immorality and is therefore not equivalent to DAGS (b,c,d).

Definition 21 (Some properties of Belief Networks)**.**

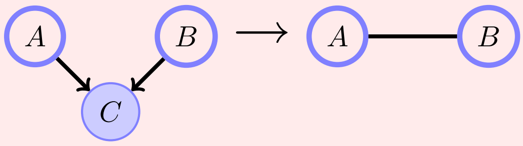


$$p(A, B, C) = p(C|A, B)p(A)p(B) \tag{3.3.26}$$

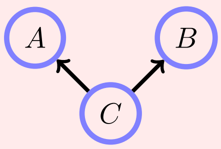
A and B are (unconditionally) independent : $p(A, B) = p(A)p(B)$.
A and B are conditionally dependent on C : $p(A, B|C) \neq p(A|C)p(B|C)$.



Marginalising over C makes A and B independent.

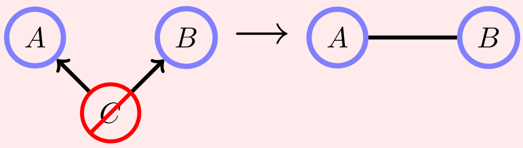


Conditioning on C makes A and B (graphically) dependent.

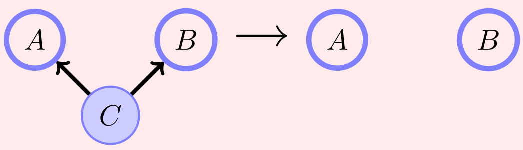


$$p(A, B, C) = p(A|c)p(B|C)p(C) \tag{3.3.27}$$

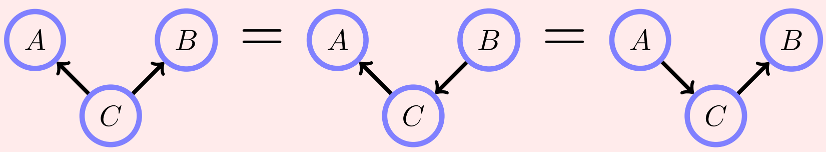
A and B are (unconditionally) dependent : $p(A, B) \neq p(A)p(B)$.
A and B are conditionally independent on C : $p(A, B|C) = p(A|C)p(B|C)$.



Marginalising over C makes A and B (graphically) dependent.



Conditioning on C makes A and B independent.



3.3.6 Belief networks have limited expressibility

Consider the DAG in fig(3.10a), (from [228]). This DAG could be used to represent two successive experiments where t_1 and t_2 are two treatments and y_1 and y_2 represent two outcomes of interest; h is the underlying health status of the patient; the first treatment has no effect on the second outcome hence there is no edge from y_1 to y_2. Now consider the implied independencies in the marginal distribution $p(t_1, t_2, y_1, y_2)$, obtained by marginalising the full distribution over h. There is no DAG containing only the vertices t_1, y_1, t_2, y_2 which represents the independence relations and does not also imply some other independence relation that is not implied by fig(3.10a). Consequently, any DAG on vertices t_1, y_1, t_2, y_2 alone will either fail to represent an independence relation of $p(t_1, t_2, y_1, y_2)$, or will impose some additional independence restriction that is not implied by the DAG. In the above example

$$p(t_1, t_2, y_1, y_2) = p(t_1)p(t_2) \sum_h p(y_1|t_1, h)p(y_2|t_2, h)p(h) \tag{3.3.28}$$

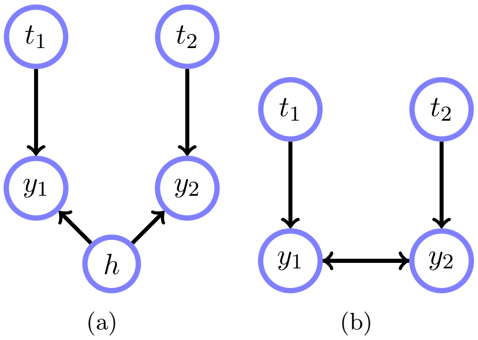


Figure 3.10: **(a)**: Two treatments t_1, t_2 and corresponding outcomes y_1, y_2. The health of a patient is represented by h. This DAG embodies the conditional independence statements $t_1 \perp\!\!\!\perp t_2, y_2 \mid \emptyset$, $t_2 \perp\!\!\!\perp t_1, y_1 \mid \emptyset$, namely that the treatments have no effect on each other. **(b)**: One could represent the marginalised latent variable using a bi-directional edge.

Figure 3.11: Both (a) and (b) represent the same distribution $p(a, b) = p(a|b)p(b) = p(b|a)p(a)$. **(c)**: The graph represents $p(rain, grasswet) = p(grasswet|rain)p(rain)$. **(d)**: We could equally have written $p(rain|grasswet)p(grasswet)$, although this appears to be causally non-sense.

cannot in general be expressed as a product of functions defined on a limited set of the variables. However, it *is* the case that the conditional independence conditions $t_1 \perp\!\!\!\perp t_2, y_2|\emptyset$, $t_2 \perp\!\!\!\perp t_1, y_1|\emptyset$ hold in $p(t_1, t_2, y_1, y_2)$ – they are there, encoded in the form of the conditional probability tables. It is just that we cannot 'see' this independence since it is not present in the structure of the marginalised graph (though one can naturally infer this in the larger graph $p(t_1, t_2, y_1, y_2, h)$).

This example demonstrates that BNs cannot express all the conditional independence statements that could be made on that set of variables (the set of conditional independence statements can be increased by considering extra latent variables however). This situation is rather general in the sense that any graphical model has limited expressibility in terms of independence statements[262]. It is worth bearing in mind that Belief Networks may not always be the most appropriate framework to express one's independence assumptions and intuitions.

A natural consideration is to use a bi-directional arrow when a latent variable is marginalised. For fig(3.10a), one could depict the marginal distribution using a bi-directional edge, fig(3.10b). Similarly a BN with a latent conditioned variable can be represented using an undirected edge. For a discussion of these and related issues, see [228].

3.4 Causality

Causality is a contentious topic and the purpose of this section is make the reader aware of some pitfalls that can occur and which may give rise to erroneous inferences. The reader is referred to [217] and [73] for further details.

The word 'causal' is contentious particularly in cases where the model of the data contains no explicit temporal information, so that formally only correlations or dependencies can be inferred. For a distribution $p(a, b)$, we could write this as either (i) $p(a|b)p(b)$ or (ii) $p(b|a)p(a)$. In (i) we might think that b 'causes' a, and in (ii) a 'causes' b. Clearly, this is not very meaningful since they both represent exactly the same distribution. Formally Belief Networks only make (in)dependence statements, not causal ones. Nevertheless, in constructing BNs, it can be helpful to think about dependencies in terms of causation since our intuitive understanding is usually framed in how one variable 'influences' another. First we discuss a classic conundrum that highlights potential pitfalls that can arise.

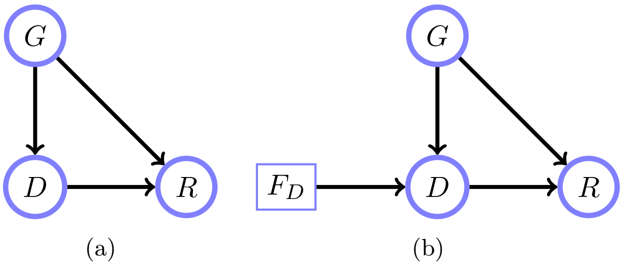


Figure 3.12: **(a)**: A DAG for the relation between Gender (G), Drug (D) and Recovery (R), see table(3.1). **(b)**: Influence diagram. No decision variable is required for G since G has no parents.

3.4.1 Simpson's Paradox

Simpson's 'paradox' is a cautionary tale in causal reasoning in BNs. Consider a medical trial in which patient treatment and outcome are recovered. Two trials were conducted, one with 40 females and one with 40 males. The data is summarised in table(3.1). The question is : Does the drug cause increased recovery? According to the table for males, the answer is no, since more males recovered when they were not given the drug than when they were. Similarly, more females recovered when not given the drug than recovered when given the drug. The conclusion appears that the drug cannot be beneficial since it aids neither subpopulation.

However, ignoring the gender information, and collating both the male and female data into one combined table, we find that more people recovered when given the drug than when not. Hence, even though the drug doesn't seem to work for either males or females, it does seem to work overall! Should we therefore recommend the drug or not?

Resolution of the Paradox

The 'paradox' occurs since we are asking a *causal* (or interventional) question. The question we are intuitively asking is, if we give someone the drug, what happens? However, the calculation we performed above was only an observational calculation. The calculation we really want is to first *intervene*, setting the drug state, and then observe what effect this has on recovery[5].

A model of the Gender, Drug and Recovery data (which makes no conditional independence assumptions) is

$$p(G, D, R) = p(R|G, D)p(D|G)p(G) \tag{3.4.1}$$

An observational calculation concerns computing $p(R|G, D)$ and $p(R|D)$. In a *causal* interpretation, however, if we intervene and give the drug, then the term $p(D|G)$ in equation (3.4.1) should play no role in the experiment (otherwise the distribution models that given the gender we select a drug with probability

[5]Pearl describes this as the difference between 'given that we see' (observational evidence), versus 'given that we do' (interventional evidence).

Males	Recovered	Not Recovered	Rec. Rate
Given Drug	18	12	60%
Not Given Drug	7	3	70%

Females	Recovered	Not Recovered	Rec. Rate
Given Drug	2	8	20%
Not Given Drug	9	21	30%

Combined	Recovered	Not Recovered	Rec. Rate
Given Drug	20	20	50%
Not Given Drug	16	24	40%

Table 3.1: Table for Simpson's Paradox (from [217])

$p(D|G)$, which is not the case – we decide to give the drug or not, independent of gender). In the causal case we are modelling the *causal experiment*; in this case the term $p(D|G)$ needs to be replaced by a term that reflects the setup of the experiment. In an *atomic intervention* a single variable is set in a particular state[6]. In our atomic causal intervention in setting D, we are dealing with the modified distribution

$$\tilde{p}(G, R|D) = p(R|G, D)p(G) \tag{3.4.2}$$

where the terms on the right hand side of this equation are taken from the original BN of the data. To denote an intervention we use $||$:

$$p(R||G, D) \equiv \tilde{p}(R|G, D) = \frac{p(R|G, D)p(G)}{\sum_R p(R|G, D)p(G)} = p(R|G, D) \tag{3.4.3}$$

(One can also consider here G as being interventional – in this case it doesn't matter since the fact that the variable G has no parents means that for any distribution conditional on G, the prior factor $p(G)$ will not be present). Using equation (3.4.3), for the males given the drug 60% recover, versus 70% recovery when not given the drug. For the females given the drug 20% recover, versus 30% recovery when not given the drug.

Similarly,

$$p(R||D) \equiv \tilde{p}(R|D) = \frac{\sum_G p(R|G, D)p(G)}{\sum_{R,G} p(R|G, D)p(G)} = \sum_G p(R|G, D)p(G) \tag{3.4.4}$$

Using the above post intervention distribution we have

$$p(\text{recovery}|\text{drug}) = 0.6 \times 0.5 + 0.2 \times 0.5 = 0.4 \tag{3.4.5}$$

and

$$p(\text{recovery}|\text{no drug}) = 0.7 \times 0.5 + 0.3 \times 0.5 = 0.5 \tag{3.4.6}$$

Hence we correctly infer that the drug is overall not helpful, as we intuitively expect, and is consistent with the results from both subpopulations.

Here $p(R||D)$ means that we first choose either a Male or Female patient at random, and then give them the drug, or not depending on the state of D. The point is that we do not randomly decide whether or not to give the drug, hence the absence of the term $p(D|G)$ from the joint distribution. One way to think about such models is to consider how to draw a sample from the joint distribution of the random variables – in most cases this should clarify the role of causality in the experiment.

In contrast to the interventional calculation, the observational calculation makes no conditional independence assumptions. This means that, for example, the term $p(D|G)$ plays a role in the calculation (the reader might wish to verify that the result given in the combined data in table(3.1) is equivalent to inferring with the full distribution equation (3.4.1)).

Definition 23 (Pearl's Do Operator)**.**

In a causal inference, in which the effect of setting variables $X_{c_1}, \ldots, X_{c_K}$, $c_k \in \mathcal{C}$, in states $\mathsf{x}_{c_1}, \ldots, \mathsf{x}_{c_K}$, is to be inferred, this is equivalent to standard evidential inference in the *post intervention distribution*:

$$p(X|do(X_{c_1} = \mathsf{x}_{c_1}), \ldots, do(X_{c_K} = \mathsf{x}_{c_K})) = \frac{p(X_1, \ldots, X_n|\mathsf{x}_{c_1}, \ldots, \mathsf{x}_{c_K})}{\prod_{i=1}^K p(X_{c_i}|\text{pa}(X_{c_i}))} = \prod_{j \notin \mathcal{C}} p(X_j|\text{pa}(X_j)) \tag{3.4.7}$$

[6] More general experimental conditions can be modelled by replacing $p(D|G)$ by an intervention distribution $\pi(D|G)$

where any parental states of $\text{pa}(X_j)$ of X_j are set in their evidential states. An alternative notation is $p(X||\mathsf{x}_{c_1}, \ldots, \mathsf{x}_{c_K})$.

In words, for those variables for which we causally intervene and set in a particular state, the corresponding terms $p(X_{c_i}|\text{pa}(X_{c_i}))$ are removed from the original Belief Network. For variables which are evidential but non-causal, the corresponding factors are not removed from the distribution. The interpretation is that the post intervention distribution corresponds to an experiment in which the causal variables are first set and non-causal variables are subsequently observed.

3.4.2 Influence Diagrams and the Do Calculus

In making causal inferences we must adjust the model to reflect any causal experimental conditions. In setting any variable into a particular state we need to surgically remove all parental links of that variable. Pearl calls this the *do operator*, and contrasts an observational ('see') inference $p(x|y)$ with a causal ('make' or 'do') inference $p(x|do(y))$.

A useful alternative representation is to append variables X upon which an intervention can possibly be made with a parental decision variable F_X[73]. For example[7]

$$\tilde{p}(D, G, R, F_D) = p(D|F_D, G)p(G)p(R|G, D)p(F_D) \tag{3.4.8}$$

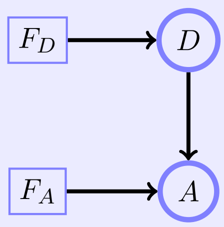
where

$$p(D|F_D = \emptyset, G) \equiv p(D|\text{pa}(D))$$
$$p(D|F_D = \mathsf{d}, G) = 1 \text{ for } D = \mathsf{d} \text{ and } 0 \text{ otherwise}$$

Hence, if the decision variable F_D is set to the empty state, the variable D is determined by the standard observational term $p(D|\text{pa}(D))$. If the decision variable is set to a state of D, then the variable puts all its probability in that single state of $D = \mathsf{d}$. This has the effect of replacing the conditional probability term a unit factor and any instances of D set to the variable in its interventional state[8].

A potential advantage of the influence diagram approach over the do-calculus is that deriving conditional independence statements can be made based on standard techniques for the augmented BN. Additionally, for parameter learning, standard techniques apply in which the decision variables are set to the condition under which each data sample was collected (a causal or non-causal sample).

Example 15 (Drivers and Accidents: A causal Belief Network)**.**



Consider the following CPT entries $p(D = \mathsf{bad}) = 0.3$, $p(A = \mathsf{tr}|D = \mathsf{bad}) = 0.9$. If intervene and use a bad driver, what is the probability of an accident?

$$p(A = \mathsf{tr}|D = \mathsf{bad}, F_D = \mathsf{tr}, F_A = \emptyset) = p(A = \mathsf{tr}|D = \mathsf{bad}) = 0.9 \tag{3.4.9}$$

On the other hand, if we intervene and make an accident, what is the probability the driver involved is bad? This is

$$p(D = \mathsf{bad}||A = \mathsf{tr}, F_D = \emptyset, F_A = \mathsf{tr}) = p(D = \mathsf{bad}) = 0.3$$

[7]Here the Influence Diagram is a distribution over variables in including decision variables, in contrast to the application of IDs in chapter(7).

[8]More general cases can be considered in which the variables are placed in a distribution of states [73].

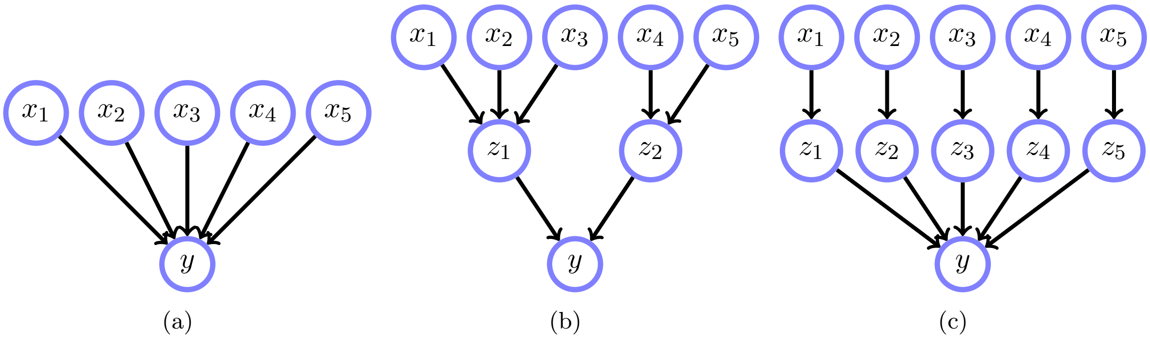


Figure 3.13: **(a)**: If all variables are binary $2^5 = 32$ states are required to specify $p(y|x_1, \ldots, x_5)$. **(b)**: Here only 16 states are required. **(c)**: Noisy logic gates.

3.4.3 Learning the direction of arrows

In the absence of data from causal experiments, one should be justifiably sceptical about learning 'causal' networks. Nevertheless, one might prefer a certain direction of a link based on assumptions of the 'simplicity' of the CPTs. This preference may come from a 'physical intuition' that whilst root 'causes' may be uncertain, the relationship from cause to effect is clear. In this sense a measure of the complexity of a CPT is required, such as entropy. Such heuristics can be numerically encoded and the 'directions' learned in an otherwise Markov equivalent graph.

3.5 Parameterising Belief Networks

Consider a variable y with many parental variables $x_1, \ldots, x_n$, fig(3.13a). Formally, the structure of the graph implies nothing about the form of the parameterisation of the table $p(y|x_1, \ldots, x_n)$. If each parent x_i has $\dim(x_i)$ states, and there is no constraint on the table, then the table $p(y|x_1, \ldots, x_n)$ contains $(\dim(y) - 1)\prod_i \dim(x_i)$ entries. If stored explicitly for each state, this would require potentially huge storage. An alternative is to constrain the table to have a simpler parametric form. For example, one might write a decomposition in which only a limited number of parental interactions are required (this is called *divorcing parents* in [146]). For example, in fig(3.13b), assuming all variables are binary, the number of states requiring specification is $2^3 + 2^2 + 2^2 = 16$, compared to the $2^5 = 32$ states in the unconstrained case. The distribution

$$p(y|x_1, \ldots, x_5) = \sum_{z_1, z_2} p(y|z_1, z_2)p(z_1|x_1, x_2, x_3)p(z_2|x_4, x_5) \tag{3.5.1}$$

can be stored using only 16 independent parameters.

Logic Gates

Another technique to constrain CPTs uses simple classes of conditional tables. For example, in fig(3.13c), one could use a logical OR gate on binary z_i, say

$$p(y|z_1, \ldots, z_5) = \begin{cases} 1 & \text{if at least one of the } z_i \text{ is in state 1} \\ 0 & \text{otherwise} \end{cases} \tag{3.5.2}$$

We can then make a CPT $p(y|x_1, \ldots, x_5)$ by including the additional terms $p(z_i = 1|x_i)$. When each x_i is binary there are in total only $2 + 2 + 2 + 2 + 2 = 10$ quantities required for specifying $p(y|x)$. In this case, fig(3.13c) can be used to represent any *noisy logic gate*, such as the *noisy OR* or *noisy AND*, where the number of parameters required to specify the noisy gate is linear in the number of parents x.

The noisy-OR is particularly common in disease-symptom networks in which many diseases x can give rise to the same symptom y– provided that at least one of the diseases is present, the probability that the symptom will be present is high.

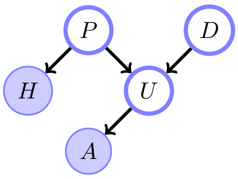


Figure 3.14: Party animal. Here all variables are binary. When set to 1 the statements are true: P = Been to Party, H = Got a Headache, D = Demotivated at work, U = Underperform at work, A =Boss Angry. Shaded variables are observed in the true state.

3.6 Further Reading

An introduction to Bayesian Networks and graphical models in expert systems is to be found in [255], which also discusses general inference techniques which will be discussed during later chapters.

3.7 Code

3.7.1 Naive Inference Demo

`demoBurglar.m`: Was it the Burglar demo
`demoChestClinic.m`: Naive Inference on Chest Clinic

3.7.2 Conditional Independence Demo

The following demo determines whether $\mathcal{X} \perp\!\!\!\perp \mathcal{Y} | \mathcal{Z}$ for the Chest Clinic network, and checks the result numerically[9]. The independence test is based on the Markov method of section(4.2.4). This is preferred over the d-separation method since it is arguably simpler to code and also more general in that it deals also with conditional independence in Markov Networks as well as Belief Networks.

Running the demo code below, it may happen that the numerical dependence is very low – that is

$$p(\mathcal{X}, \mathcal{Y}|\mathcal{Z}) \approx p(\mathcal{X}|\mathcal{Z})p(\mathcal{Y}|\mathcal{Z}) \tag{3.7.1}$$

even though $\mathcal{X} \top\!\!\!\top \mathcal{Y} | \mathcal{Z}$. This highlights the difference between 'structural' and 'numerical' independence.
`condindepPot.m`: Numerical measure of conditional independence
`demoCondindep.m`: Demo of conditional independence (using Markov method)

3.7.3 Utility Routines

`dag.m`: Find the DAG structure for a Belief Network

3.8 Exercises

Exercise 21 (Party Animal)**.** *The party animal problem corresponds to the network in fig(3.14). The boss is angry and the worker has a headache – what is the probability the worker has been to a party? To complete the specifications, the probabilities are given as follows:*

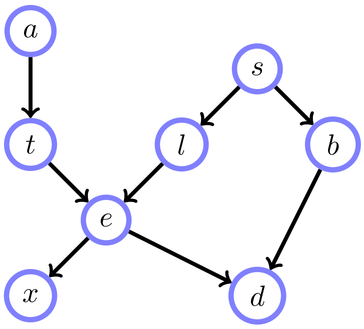
$p(U = \mathsf{tr}|P = \mathsf{tr}, D = \mathsf{tr}) = 0.999$ $\quad p(U = \mathsf{tr}|P = \mathsf{fa}, D = \mathsf{tr}) = 0.9$
$p(U = \mathsf{tr}|P = \mathsf{tr}, D = \mathsf{fa}) = 0.9$ $\quad p(U = \mathsf{tr}|P = \mathsf{fa}, D = \mathsf{fa}) = 0.01$

Exercise 22. *Consider the distribution $p(a, b, c) = p(c|a, b)p(a)p(b)$. (i) Is $a \perp\!\!\!\perp b | \emptyset$?. (ii) Is $a \perp\!\!\!\perp b | c$?*

Exercise 23. *The Chest Clinic network [167] concerns the diagnosis of lung disease (tuberculosis, lung cancer, or both, or neither). In this model a visit to Asia is assumed to increase the probability of tuberculosis. State if the following conditional independence relationships are true or false*

1. *$tuberculosis \perp\!\!\!\perp smoking | shortness\ of\ breath$,*

[9] The code for (structural) conditional independence is given in chapter(4).



x = Positive X-ray
d = Dyspnea (Shortness of breath)
e = Either Tuberculosis or Lung Cancer
t = Tuberculosis
l = Lung Cancer
b = Bronchitis
a = Visited Asia
s = Smoker

Figure 3.15: Belief network structure for the Chest Clinic example.

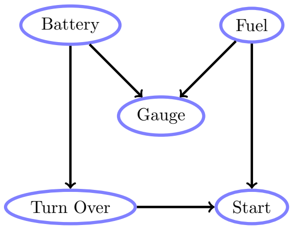


Figure 3.16: Belief Network of car not starting[126], see exercise(24).

2. *lung cancer* $\perp\!\!\!\perp$ *bronchitis*| *smoking*,

3. *visit to Asia* $\perp\!\!\!\perp$ *smoking*| *lung cancer*

4. *visit to Asia* $\perp\!\!\!\perp$ *smoking*| *lung cancer*, *shortness of breath*.

Exercise 24 ([126])**.** *Consider the network in fig(3.16), which concerns the probability of a car starting.*

$$
\begin{array}{ll}
p(b = \text{bad}) = 0.02 & p(f = \text{empty}) = 0.05 \\
p(g = \text{empty}|b = \text{good}, f = \text{not empty}) = 0.04 & p(g = \text{empty}|b = \text{good}, f = \text{empty}) = 0.97 \\
p(g = \text{empty}|b = \text{bad}, f = \text{not empty}) = 0.1 & p(g = \text{empty}|b = \text{bad}, f = \text{empty}) = 0.99 \\
p(t = \text{fa}|b = \text{good}) = 0.03 & p(t = \text{fa}|b = \text{bad}) = 0.98 \\
p(s = \text{fa}|t = \text{tr}, f = \text{not empty}) = 0.01 & p(s = \text{fa}|t = \text{tr}, f = \text{empty}) = 0.92 \\
p(s = \text{fa}|t = \text{fa}, f = \text{not empty}) = 1.0 & p(s = \text{fa}|t = \text{fa}, f = \text{empty}) = 0.99
\end{array}
$$

Calculate $P(f = \text{empty}|s = no)$*, the probability of the fuel tank being empty conditioned on the observation that the car does not start.*

Exercise 25. *Consider the Chest Clinic Bayesian Network in fig(3.15) [167]. Calculate by hand the values for* $p(D)$*,* $p(D|S = \text{tr})$*,* $p(D|S = \text{fa})$*. The table values are:*

$$
\begin{array}{llll}
p(a = \text{tr}) & = 0.01 & p(s = \text{tr}) & = 0.5 \\
p(t = \text{tr}|a = \text{tr}) & = 0.05 & p(t = \text{tr}|a = \text{fa}) & = 0.01 \\
p(l = \text{tr}|s = \text{tr}) & = 0.1 & p(l = \text{tr}|s = \text{fa}) & = 0.01 \\
p(b = \text{tr}|s = \text{tr}) & = 0.6 & p(b = \text{tr}|s = \text{fa}) & = 0.3 \\
p(x = \text{tr}|e = \text{tr}) & = 0.98 & p(x = \text{tr}|e = \text{fa}) & = 0.05 \\
p(d = \text{tr}|e = \text{tr}, b = \text{tr}) & = 0.9 & p(d = \text{tr}|e = \text{tr}, b = \text{fa}) & = 0.7 \\
p(d = \text{tr}|e = \text{fa}, b = \text{tr}) & = 0.8 & p(d = \text{tr}|e = \text{fa}, b = \text{fa}) & = 0.1
\end{array}
$$

$p(e = \text{tr}|t, l) = 0$ *only if both* t *and* l *are* fa*, 1 otherwise.*

Exercise 26. *If we interpret the Chest Clinic network exercise(25) causally, how can we help a doctor answer the question 'If I could cure my patients of Bronchitis, how would this affect my patients's chance of being short of breath?'. How does this compare with* $p(d = \text{tr}|b = \text{fa})$ *in a non-causal interpretation, and what does this mean?*

Exercise 27. *There is a synergistic relationship between Asbestos (A) exposure, Smoking (S) and Cancer (C). A model describing this relationship is given by*

$$p(A, S, C) = p(C|A, S)p(A)p(S) \tag{3.8.1}$$

1. *Is* $A \perp\!\!\!\perp S|\emptyset$*?*

2. *Is* $A \perp\!\!\!\perp S|C$*?*

3. *How could you adjust the model to account for the fact that people who work in the building industry have a higher likelihood to also be smokers and also a higher likelihood to asbestos exposure?*

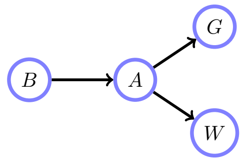
Exercise 28. *Consider the three variable distribution*

$$p(a, b, c) = p(a|b)p(b|c)p(c) \tag{3.8.2}$$

where all variables are binary. How many parameters are needed to specify distributions of this form?

Exercise 29.

Consider the Belief Network on the right which represents Mr Holmes' burglary worries as given in fig(3.2a) : (B)urglar, (A)larm, (W)atson, Mrs (Gibbon).



All variables take the two states $\{\mathsf{tr}, \mathsf{fa}\}$*. The table entries are*

$$\begin{array}{llll} p(B = \mathsf{tr}) & = 0.01 & & \\ p(A = \mathsf{tr}|B = \mathsf{tr}) & = 0.99 & p(A = \mathsf{tr}|B = \mathsf{fa}) & = 0.05 \\ p(W = \mathsf{tr}|A = \mathsf{tr}) & = 0.9 & p(W = \mathsf{tr}|A = \mathsf{fa}) & = 0.5 \\ p(G = \mathsf{tr}|A = \mathsf{tr}) & = 0.7 & p(G = \mathsf{tr}|A = \mathsf{fa}) & = 0.2 \end{array} \tag{3.8.3}$$

1. *Compute 'by hand' (*i.e. *show your working) :*
   (a) $p(B = \mathsf{tr}|W = \mathsf{tr})$
   (b) $p(B = \mathsf{tr}|W = \mathsf{tr}, G = \mathsf{fa})$

2. *Consider the same situation as above, except that now the evidence is uncertain. Mrs Gibbon thinks that the state is* $G = \mathsf{fa}$ *with probability 0.9. Similarly, Dr Watson believes in the state* $W = \mathsf{fa}$ *with value 0.7. Compute 'by hand' the posteriors under these uncertain (soft) evidences:*
   (a) $p(B = \mathsf{tr}|\tilde{W})$
   (b) $p(B = \mathsf{tr}|\tilde{W}, \tilde{G})$

Exercise 30. *A doctor gives a patient a (D)rug (drug or no drug) dependent on their (A)ge (old or young) and (G)ender (male or female). Whether or not the patient (R)ecovers (recovers or doesn't recover) depends on all* D, A, G*. In addition* $A \perp\!\!\!\perp G|\emptyset$*.*

1. *Write down the Belief Network for the above situation.*

2. *Explain how to compute* $p(\mathsf{recover}|\mathsf{drug})$*.*

3. *Explain how to compute* $p(\mathsf{recover}|do(\mathsf{drug}), \mathsf{young})$*.*

Exercise 31. *Implement the Wet Grass scenario numerically using the* BRMLtoolbox*.*

Exercise 32 (LA Burglar)**.** *Consider the Burglar scenario, example(10). We now wish to model the fact that in Los Angeles the probability of being burgled increases if there is an earthquake. Explain how to include this effect in the model.*

Exercise 33. *Given two Belief Networks represented as DAGs with associated adjacency matrices* $\mathbf{A}$ *and* $\mathbf{B}$*, write a* MATLAB *function* `MarkovEquiv(A,B).m` *that returns 1 if* $\mathbf{A}$ *and* $\mathbf{B}$ *are Markov equivalent, and zero otherwise.*

Exercise 34. *The adjacency matrices of two Belief Networks are given below. State if they are Markov equivalent.*

$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \qquad \mathbf{B} = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{3.8.4}$$

Exercise 35. *There are three computers indexed by $i \in \{1,2,3\}$. Computer i can send a message in one timestep to computer j if $C_{ij} = 1$, otherwise $C_{ij} = 0$. There is a fault in the network and the task is to find out some information about the communication matrix $\mathbf{C}$ ($\mathbf{C}$ is not necessarily symmetric). To do this, Thomas, the engineer, will run some tests that reveal whether or not computer i can send a message to computer j in t timesteps, $t \in \{1,2\}$. This is expressed as $C_{ij}(t)$, with $C_{ij}(1) \equiv C_{ij}$. For example, he might know that $C_{13}(2) = 1$, meaning that according to his test, a message sent from computer 1 will arrive at computer 3 in at most 2 timesteps. Note that this message could go via different routes – it might go directly from 1 to 3 in one timestep, or indirectly from 1 to 2 and then from 2 to 3, or both. You may assume $C_{ii} = 1$. A priori Thomas thinks there is a 10% probability that $C_{ij} = 1$. Given the test information $\mathcal{C} = \{C_{12}(2) = 1, C_{23}(2) = 0\}$, compute the a posteriori probability vector*

$$[p(C_{12} = 1|\mathcal{C}), p(C_{13} = 1|\mathcal{C}), p(C_{23} = 1|\mathcal{C}), p(C_{32} = 1|\mathcal{C}), p(C_{21} = 1|\mathcal{C}), p(C_{31} = 1|\mathcal{C})] \tag{3.8.5}$$

Exercise 36. *A Belief Network models the relation between the variables oil, inf, eh, bp, rt which stand for the price of oil, inflation rate, economy health, British Petroleum Stock price, retailer stock price. Each variable takes the states* low, high, *except for bp which has states* low, high, normal. *The Belief Network model for these variables has tables*

*p(eh=*low*)=0.2*	
*p(bp=*low*|oil=*low*)=0.9*	*p(bp=*normal*|oil=*low*)=0.1*
*p(bp=*low*|oil=*high*)=0.1*	*p(bp=*normal*|oil=*high*)=0.4*
*p(oil=*low*|eh=*low*)=0.9*	*p(oil=*low*|eh=*high*)=0.05*
*p(rt=*low*|inf=*low*,eh=*low*)=0.9*	*p(rt=*low*|inf=*low*,eh=*high*)=0.1*
*p(rt=*low*|inf=*high*,eh=*low*)=0.1*	*p(rt=*low*|inf=*high*,eh=*high*)=0.01*
*p(inf=*low*|oil=*low*,eh=*low*)=0.9*	*p(inf=*low*|oil=*low*,eh=*high*)=0.1*
*p(inf=*low*|oil=*high*,eh=*low*)=0.1*	*p(inf=*low*|oil=*high*,eh=*high*)=0.01*

1. *Draw a Belief Network for this distribution.*
2. *Given that BP stock price is normal and the retailer stock price is high, what is the probability that inflation is high?*

Exercise 37. *There are a set of C potentials with potential c defined on a subset of variables $\mathcal{X}_c$. If $\mathcal{X}_c \subseteq \mathcal{X}_d$ then can merge (multiply) potentials c and d since c is contained within d. With reference to suitable graph structures, describe an efficient algorithm to merge a set of potentials so that for the new set of potentials no potential is contained within the other.*

CHAPTER 4

Graphical Models

4.1 Graphical Models

Graphical Models (GMs) are depictions of independence/dependence relationships for distributions. Each form of GM is a particular union of graph and probability constructs and details the form of independence assumptions represented. GMs are useful since they provide a framework for studying a wide class of probabilistic models and associated algorithms. In particular they help to clarify modelling assumptions and provide a unified framework under which inference algorithms in different communities can be related.

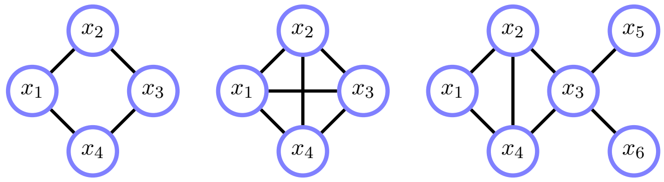
It needs to be emphasised that all forms of GM have a limited ability to graphically expresses conditional (in)dependence statements[262]. As we've seen, Belief Networks are useful for modelling ancestral conditional independence. In this chapter we'll introduce other types of GM that are more suited to representing different assumptions. Markov Networks, for example, are particularly suited to modelling marginal dependence and conditional independence. Here we'll focus on Markov Networks, Chain Graphs (which marry Belief and Markov networks) and Factor Graphs. There are many more inhabitants of the zoo of Graphical Models, see [69, 290].

The general viewpoint we adopt is to describe the problem environment using a probabilistic model, after which reasoning corresponds to performing probabilistic inference. This is therefore a two part process :

Modelling After identifying all potentially relevant variables of a problem environment, our task is to describe how these variables can interact. This is achieved using structural assumptions as to the form of the joint probability distribution of all the variables, typically corresponding to assumptions of independence of variables. Each class of graphical model corresponds to a factorisation property of the joint distribution.

Inference Once the basic assumptions as to how variables interact with each other is formed (*i.e.* the probabilistic model is constructed) all questions of interest are answered by performing inference on the distribution. This can be a computationally non-trivial step so that coupling GMs with accurate inference algorithms is central to successful graphical modelling.

Whilst not a strict separation, GMs tend to fall into two broad classes – those useful in modelling, and those useful in representing inference algorithms. For modelling, Belief Networks, Markov Networks, Chain Graphs and Influence Diagrams are some of the most popular. For inference one typically 'compiles' a model into a suitable GM for which an algorithm can be readily applied. Such inference GMs include Factor Graphs, Junction Trees and Region Graphs.



(a) (b) (c)

Figure 4.1: **(a)**: $p_a = \phi(x_1,x_2)\phi(x_2,x_3)\phi(x_3,x_4)\phi(x_4,x_1)/Z_a$. **(b)**: $p_b = \phi(x_1,x_2,x_3,x_4)/Z_b$. **(c)**: $p_c = \phi(x_1,x_2,x_4)\phi(x_2,x_3,x_4)\phi(x_3,x_5)\phi(x_3,x_6)/Z_c$.

4.2 Markov Networks

Belief Networks correspond to a special kind of factorisation of the joint probability distribution in which each of the factors is itself a distribution. An alternative factorisation is, for example

$$p(a,b,c) = \frac{1}{Z}\phi(a,b)\phi(b,c) \tag{4.2.1}$$

where $\phi(a,b)$ and $\phi(b,c)$ are *potentials* and Z is a constant which ensures normalisation, called the *partition function*

$$Z = \sum_{a,b,c} \phi(a,b)\phi(b,c) \tag{4.2.2}$$

We will typically use the convention that the ordering of the variables in the potential is not relevant (as for a distribution) – the joint variables simply index an element of the potential table. Markov Networks are defined as products of non-negative functions defined on maximal cliques of an undirected graph – see fig(4.1).

Definition 24 (Potential)**.** A potential $\phi(x)$ is a non-negative function of the variable x, $\phi(x) \geq 0$. A joint potential $\phi(x_1,\ldots,x_n)$ is a non-negative function of the set of variables. A distribution is a special case of a potential satisfying normalisation, $\sum_x \phi(x) = 1$. This holds similarly for continuous variables, with summation replaced by integration.

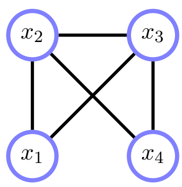
Definition 25 (Markov Network)**.** For a set of variables $\mathcal{X} = \{x_1,\ldots,x_n\}$ a Markov network is defined as a product of potentials on subsets of the variables $\mathcal{X}_c \subseteq \mathcal{X}$:

$$p(x_1,\ldots,x_n) = \frac{1}{Z}\prod_{c=1}^{C}\phi_c(\mathcal{X}_c) \tag{4.2.3}$$

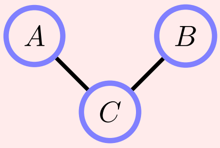
Graphically this is represented by an undirected graph G with $\mathcal{X}_c, c = 1\ldots,C$ being the maximal cliques of G. The constant Z ensures the distribution is normalised. The graph is said to satisfy the factorisation property (F). In the special case that the graph contains cliques of only size 2, the distribution is called a *pairwise Markov Network*, with potentials defined on each link between two variables.

For the case in which clique potentials are strictly positive, this is called a *Gibbs distribution*.

Remark 3 (*Pairwise Markov network*)**.** Whilst a Markov network is formally defined on maximal cliques, in practice authors often use the term to refer to non-maximal cliques.
For example, in the graph on the right, the maximal cliques are x_1,x_2,x_3 and x_2,x_3,x_4, so that the graph describes a distribution $p(x_2,x_2,x_3,x_4) = \phi(x_1,x_2,x_3)\phi(x_2,x_3,x_4)/Z$. In a pairwise network though the potentials are assumed to be over two-cliques, giving $p(x_2,x_2,x_3,x_4) = \phi(x_1,x_2)\phi(x_1,x_3)\phi(x_2,x_3)\phi(x_2,x_4)\phi(x_3,x_4)/Z$.

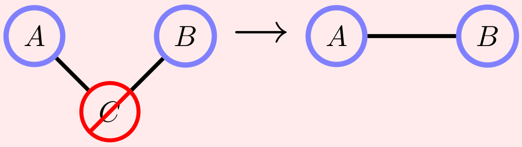


Definition 26 (Properties of Markov Networks)**.**

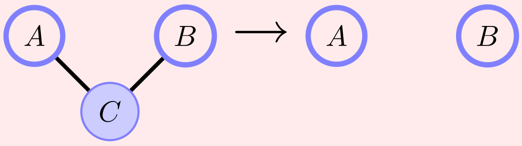


$$p(A, B, C) = \phi_{AC}(A, C)\phi_{BC}(B, C)/Z \tag{4.2.4}$$

A and B are unconditionally dependent : $p(A, B) \neq p(A)p(B)$.
A and B are conditionally independent on C : $p(A, B|C) = p(A|C)p(B|C)$.



Marginalising over C makes A and B (graphically) dependent.



Conditioning on C makes A and B independent.

4.2.1 Markov Properties

We here state some of the most useful results. The reader is referred to [165] for proofs and more detailed discussion. Consider the Markov Network in fig(4.2a). Here we use the shorthand $p(1) \equiv p(x_1)$, $\phi(1, 2, 3) \equiv \phi(x_1, x_2, x_3)$ *etc.* We will use this undirected graph to demonstrate conditional independence properties.

Local Markov Property

Definition 27 (Local Markov Property (L))**.**

$$p(x|\mathcal{X}\backslash x) = p(x|\text{ne}\,(x)) \tag{4.2.5}$$

When conditioned on its neighbours, x is independent of the remaining variables of the graph.

The conditional distribution $p(4|1, 2, 3, 5, 6, 7)$ is

$$p(4|1, 2, 3, 5, 6, 7) = \frac{\phi(1, 2, 3)\phi(2, 3, 4)\phi(4, 5, 6)\phi(5, 6, 7)}{\sum_4 \phi(1, 2, 3)\phi(2, 3, 4)\phi(4, 5, 6)\phi(5, 6, 7)} = \frac{\phi(2, 3, 4)\phi(4, 5, 6)}{\sum_4 \phi(2, 3, 4)\phi(4, 5, 6)} = p(4|2, 3, 5, 6) \tag{4.2.6}$$

The last line above follows since the variable x_4 only appears in the cliques that border x_4. The generalisation of the above example is clear: a MN with positive clique potentials ϕ, defined with respect to an undirected graph G entails[1] $p(x_i|x_{\backslash i}) = p(x_i|\text{ne}\,(x_i))$.

Pairwise Markov Property

Definition 28 (Pairwise Markov Property (P))**.** For any non-adjacent vertices x and y

$$x \perp\!\!\!\perp y|\,\mathcal{X}\backslash\,\{x, y\} \tag{4.2.7}$$

[1]The notation $x_{\backslash i}$ is shorthand for the set of all variables $\mathcal{X}$ excluding variable x_i, namely $\mathcal{X}\backslash x_i$ in set notation.

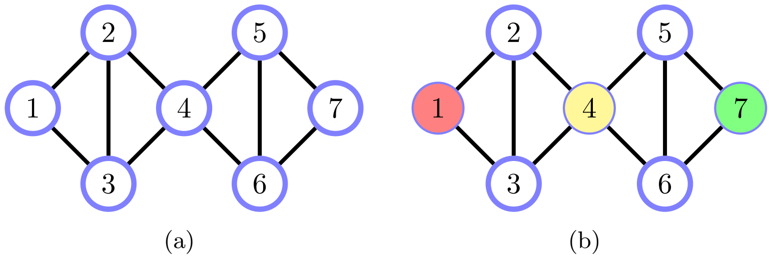


Figure 4.2: **(a)**: $\phi(1,2,3)\phi(2,3,4)\phi(4,5,6)\phi(5,6,7)$. **(b)**: By the global Markov property, since every path from 1 to 7 passes through 4, then $1 \perp\!\!\!\perp 7 | 4$.

$$p(1,4|2,3,5,6,7) = \frac{\phi(1,2,3)\phi(2,3,4)\phi(4,5,6)\phi(5,6,7)}{\sum_{1,4}\phi(1,2,3)\phi(2,3,4)\phi(4,5,6)\phi(5,6,7)} = \frac{\phi(1,2,3)\phi(2,3,4)\phi(4,5,6)}{\sum_{1,4}\phi(1,2,3)\phi(2,3,4)\phi(4,5,6)} \tag{4.2.8}$$

$$= p(1|2,3,4,5,6,7)p(4|1,2,3,5,6,7) \tag{4.2.9}$$

where the last line follows since for fixed $2,3,5,6,7$, the function $\phi(1,2,3)\phi(2,3,4)\phi(4,5,6)$ is a product of a function on 1 and a function on 4, implying independence.

Global Markov Property

Definition 29 (Separation)**.** A subset $\mathcal{S}$ separates a subset $\mathcal{A}$ from a subset $\mathcal{B}$ if every path from any member of $\mathcal{A}$ to any member of $\mathcal{B}$ passes though $\mathcal{S}$.

Definition 30 (Global Markov Property (G))**.** For a disjoint subset of variables, $(\mathcal{A},\mathcal{B},\mathcal{S})$ where $\mathcal{S}$ separates $\mathcal{A}$ from $\mathcal{B}$ in G, then $\mathcal{A} \perp\!\!\!\perp \mathcal{B} | \mathcal{S}$.

$$p(1,7|4) \propto \sum_{2,3,5,6} p(1,2,3,4,5,6,7) \tag{4.2.10}$$

$$= \sum_{2,3,5,6} \phi(1,2,3)\phi(2,3,4)\phi(4,5,6)\phi(5,6,7) \tag{4.2.11}$$

$$= \left\{\sum_{2,3} \phi(1,2,3)\phi(2,3,4)\right\}\left\{\sum_{5,6} \phi(4,5,6)\phi(5,6,7)\right\} \tag{4.2.12}$$

This implies that $p(1,7|4) = p(1|4)p(7|4)$.

Example 16 (*Boltzmann machine*)**.** A Boltzmann machine is a MN on binary variables $\text{dom}(x_i) = \{0,1\}$ of the form

$$p(\mathbf{x}) = \frac{1}{Z(\mathbf{w},b)} e^{\sum_{i<j} w_{ij}x_ix_j + \sum_i b_i x_i} \tag{4.2.13}$$

where the interactions w_{ij} are the 'weights' and the b_i the biases. This model has been studied in the machine learning community as a basic model of distributed memory and computation[2]. The graphical model of the BM is an undirected graph with a link been nodes i and j for $w_{ij} \neq 0$. Consequently, for all but specially constrained $\mathbf{W}$, the graph is multiply-connected and inference will be typically intractable.

4.2.2 Gibbs Networks

For simplicity we assume that the potentials are strictly positive in which case MNs are also termed *Gibbs Networks*. In this case, a GN satisfies the following independence relations:

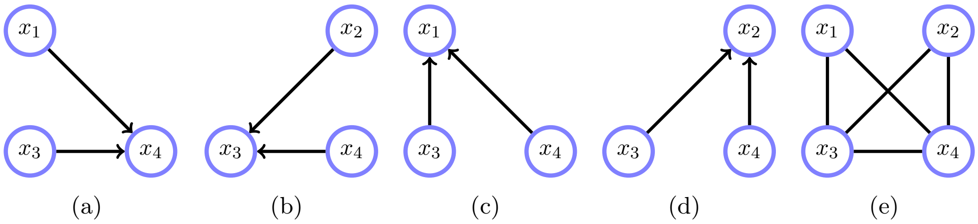


Figure 4.3: **(a-d)**: Local distributions. **(e)**: The Markov network consistent with the local distributions. If the local distributions are positive, by the Hammersley-Clifford theorem, the only joint distribution that can be consistent with the local distributions must be a Gibbs distribution with structure given by (e).

4.2.3 Markov random fields

Definition 31 (Markov Random Field)**.** A MRF is defined by a set of distributions $p(x_i|\text{ne}(x_i))$ where $i \in \{1, \ldots, n\}$ indexes the distributions and $\text{ne}(x_i)$ are the neighbours of variable x_i, namely that subset of the variables $x_1, \ldots, x_n$ that the distribution of variable x_i depends on. The term Markov indicates that this is a proper subset of the variables.

A distribution is an MRF with respect to an undirected graph G if

$$p(x_i|x_{\setminus i}) = p(x_i|\text{ne}(x_i)) \tag{4.2.14}$$

where $\text{ne}(x_i)$ are the neighbouring variables of variable x_i, according to the undirected graph G.

Hammersley Clifford Theorem

The Hammersley-Clifford theorem helps resolve questions as to when a set of positive local distributions $p(x_i|\text{ne}(x_i))$ could ever form a consistent joint distribution $p(x_1, \ldots, x_n)$. Local distributions $p(x_i|\text{ne}(x_i))$ can form a consistent joint distribution if and only if $p(x_1, \ldots, x_n)$ factorises according to

$$p(x_1, \ldots, x_n) = \frac{1}{Z} \exp\left(-\sum_c V_c(\mathcal{X}_c)\right) \tag{4.2.15}$$

where the sum is over all cliques and $V_c(\mathcal{X}_c)$ is a real function defined over the variables in the clique indexed by c. Equation (4.2.15) is equivalent to $\prod_c \phi(\mathcal{X}_c)$, namely a MN on positive clique potentials.

The graph over which the cliques are defined is an undirected graph with a link between x_i and x_j if

$$p(x_i|x_{\setminus i}) \neq p(x_i|x_{\setminus(i,j)}) \tag{4.2.16}$$

That is, if x_j has an effect on the conditional distribution of x_i, then add an undirected link between x_i and x_j. This is then repeated over all the variables x_i[35, 201], see fig(4.3). Note that the HC theorem does not mean that given a set of conditional distributions, we can always form a consistent joint distribution from them – rather it states what the functional form of a joint distribution must be if we are to have any hope that the conditionals are consistent with the joint, see exercise(46).

4.2.4 Conditional independence using Markov networks

For $\mathcal{X}$,$\mathcal{Y}$,$\mathcal{Z}$ each being collections of variables, in section(3.3.3) we discussed an algorithm to determine $\mathcal{X} \perp\!\!\!\perp \mathcal{Y}|\,\mathcal{Z}$. An alternative and more general method (since it handles directed and undirected graphs) uses the following steps: (see [73, 166])

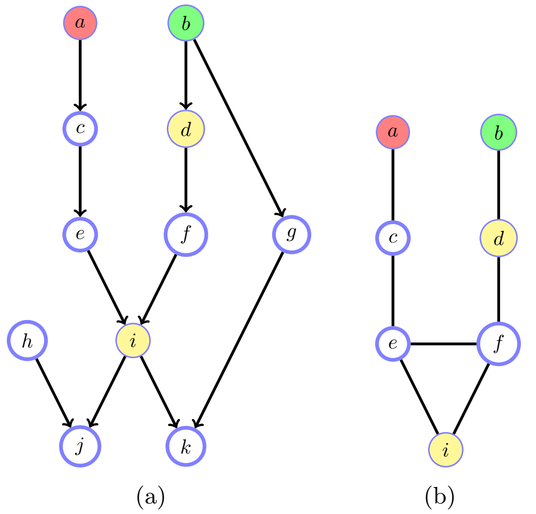


Figure 4.4: **(a)**: Belief Network for which we are interested in checking conditional independence $a \perp\!\!\!\perp b \mid \{d, i\}$. **(b)**: Ancestral moralised graph for $a \perp\!\!\!\perp b \mid \{d, i\}$. Every path from a red to green node passes through a yellow node, so a and b are independent given d, i. Alternatively, if we consider $a \perp\!\!\!\perp b \mid i$, the variable d is uncoloured, and we can travel from the red to the green without encountering a yellow node (using the $e - f$ path). In this case a is dependent on b, conditioned on i.

Ancestral Graph Remove from the DAG any node which is neither in $\mathcal{X} \cup \mathcal{Y} \cup \mathcal{Z}$ nor an ancestor of a node in this set, together with any edges in or out of such nodes.

Moralisation Add a line between any two remaining nodes which have a common child, but are not already connected by an arrow. Then remove remaining arrowheads.

Separation In the undirected graph so constructed, look for a path which joins a node in $\mathcal{X}$ to one in $\mathcal{Y}$ but does not intersect $\mathcal{Z}$. If there is no such path deduce that $\mathcal{X} \perp\!\!\!\perp \mathcal{Y} \mid \mathcal{Z}$.

For Markov Networks only the final separation criterion needs to be applied. See fig(4.4) for an example.

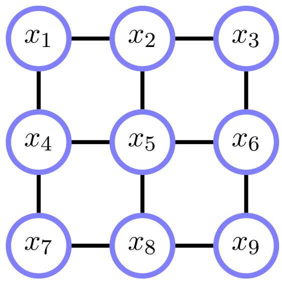
4.2.5 Lattice Models

Undirected models have a long history in different branches of science, especially statistical mechanics on lattices and more recently as models in visual processing in which the models encourage neighbouring variables to be in the same states[35, 36, 103].

Consider a model in which our desire is that states of the binary valued variables $x_1, \ldots, x_9$, arranged on a lattice (right) should prefer their neighbouring variables to be in the same state

$$p(x_1, \ldots x_9) = \frac{1}{Z} \prod_{i \sim j} \phi_{ij}(x_i, x_j) \tag{4.2.17}$$

where $i \sim j$ denotes the set of indices where i and j are neighbours in the undirected graph.



The Ising model

A set of potentials for equation (4.2.17) that encourages neighbouring variables to have the same state is

$$\phi_{ij}(x_i, x_j) = e^{-\frac{1}{2T}(x_i - x_j)^2} \tag{4.2.18}$$

This corresponds to a well-known model of the physics of magnetic systems, called the *Ising model* which consists of 'mini-magnets' which prefer to be aligned in the same state, depending on the temperature T. For high T the variables behave independently so that no global magnetisation appears. For low T, there is a strong preference for neighbouring mini-magnets to become aligned, generating a strong macro-magnet. Remarkably, one can show that, in a very large two-dimensional lattice, below the so-called Curie temperature, $T_c \approx 2.269$ (for ± 1 variables), the system admits a phase change in that a large fraction of the variables become aligned – above T_c, on average, the variables are unaligned. This is depicted in fig(4.5) where $M = \left| \sum_{i=1}^{N} x_i \right| / N$ is the average alignment of the variables. That this phase change happens for non-zero temperature has driven considerable research in this and related areas[40]. Global coherence

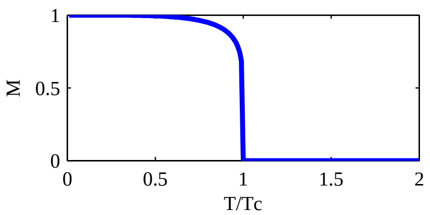


Figure 4.5: Onsagar magnetisation. As the temperature T decreases towards the critical temperature T_c a phase transition occurs in which a large fraction of the variables become aligned in the same state.

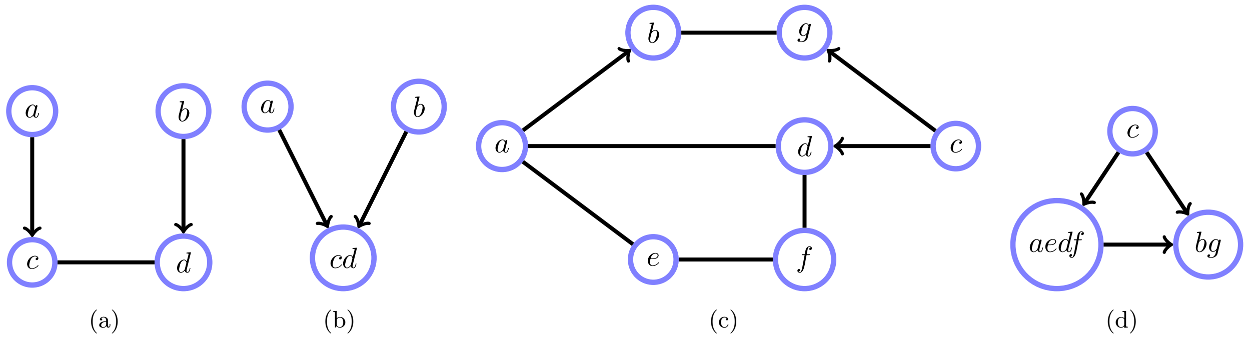


Figure 4.6: Chain graphs. The chain components are identified by deleting the directed edges and identifying the remaining connected components. **(a)**: Chain components are (a),(b),(c, d), which can be written as a BN on the cluster variables in (b). **(c)**: Chain components are (a, e, d, f), (b, g), (c), which has the cluster BN representation (d). (From [165])

effects such as this that arise from weak local constraints are present in systems that admit *emergent behaviour*. Similar local constraints are popular in image restoration algorithms to clean up noise, under the assumption that noise will not show any local spatial coherence, whilst 'signal' will. An example is given in section(28.8) where we discuss algorithms for inference under special constraints on the MRF.

4.3 Chain Graphical Models

Definition 32 (Chain Component)**.** The chain components of a graph G are obtained by :

1. Forming a graph G' with directed edges removed from G.
2. Then each connected component in G' constitutes a chain component.

Chain Graphs (CGs) contain both directed and undirected links. To develop the intuition, consider fig(4.6a). The only terms that we can unambiguously specify from this depiction are $p(a)$ and $p(b)$ since there is no mixed interaction of directed and undirected edges at the a and b vertices. By probability, therefore, we must have

$$p(a, b, c, d) = p(a)p(b)p(c, d|a, b) \tag{4.3.1}$$

Looking at the graph, we might expect the interpretation to be

$$p(c, d|a, b) = \phi(c, d)p(c|a)p(d|b) \tag{4.3.2}$$

However, to ensure normalisation, and also to retain generality, we interpret this chain component as

$$p(c, d|a, b) = \phi(c, d)p(c|a)p(d|b)\phi(a, b), \text{ with } \phi(a, b) \equiv 1/\sum_{c,d}\phi(c, d)p(c|a)p(d|b) \tag{4.3.3}$$

This leads to the interpretation of a CG as a DAG over the chain components. Each chain component represents a distribution over the variables of the component, conditioned on the parental components. The conditional distribution is itself a product over the cliques of the undirected component and moralised parental components, including also a factor to ensure normalisation over the chain component.

Definition 33 (Chain Graph distribution)**.** The distribution associated with a chain graph G is found by first identifying the chain components, τ. Then

$$p(x) = \prod_{\tau} p\left(\mathcal{X}_\tau|\mathrm{pa}\left(\mathcal{X}_\tau\right)\right) \tag{4.3.4}$$

and

$$p\left(\mathcal{X}_\tau|\mathrm{pa}\left(\mathcal{X}_\tau\right)\right) \propto \prod_{c\in\mathcal{C}_\tau} \phi\left(\mathcal{X}_{\mathcal{C}_\tau}\right) \tag{4.3.5}$$

where $\mathcal{C}_\tau$ denotes the union of the cliques in component τ together with the moralised parental components of τ, with ϕ being the associated functions defined on each clique. The proportionality factor is determined implicitly by the constraint that the distribution sums to 1.

BNs are CGs in which the connected components are singletons. MNs are CGs in which the chain components are simply the connected components of the undirected graph.

CGs can be useful since they are more expressive of CI statements than either Belief Networks or Markov Networks alone. The reader is referred to [165] and [96] for further details.

Example 17 (Chain Graphs are more expressive than Belief or Markov Networks)**.** Consider the chain graph in fig(4.7a), which has chain component decomposition

$$p(a, b, c, d, e, f) = p(a)p(b)p(c, d, e, f|a, b) \tag{4.3.6}$$

where

$$p(c, d, e, f|a, b) = \phi(a, c)\phi(c, e)\phi(e, f)\phi(d, f)\phi(d, b)\phi(a, b) \tag{4.3.7}$$

with the normalisation requirement

$$\phi(a, b) \equiv 1/ \sum_{c,d,e,f} \phi(a, c)\phi(c, e)\phi(e, f)\phi(d, f)\phi(d, b) \tag{4.3.8}$$

The marginal $p(c, d, e, f)$ is given by

$$\phi(c, e)\phi(e, f)\phi(d, f) \underbrace{\sum_{a,b} \phi(a, b)p(a)p(b)\phi(a, c)\phi(d, b)}_{\phi(c,d)} \tag{4.3.9}$$

Since the marginal distribution of $p(c, d, e, f)$ is an undirected 4-cycle, no DAG can express the CI statements contained in the marginal $p(c, d, e, f)$. Similarly no undirected distribution on the same skeleton as fig(4.7a) could express that a and b are independent (unconditionally), *i.e.* $p(a, b) = p(a)p(b)$.

4.4 Expressiveness of Graphical Models

It is clear that directed distributions can be represented as undirected distributions since one can associate each (normalised) factor in a directed distribution with a potential. For example, the distribution $p(a|b)p(b|c)p(c)$ can be factored as $\phi(a, b)\phi(b, c)$, where $\phi(a, b) = p(a|b)$ and $\phi(b, c) = p(b|c)p(c)$, with $Z = 1$. Hence every Belief Network can be represented as some MN by simple identification of the factors in the distributions. However, in general, the associated undirected graph (which corresponds to the

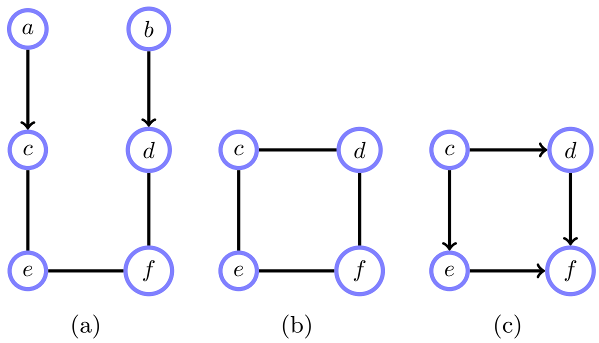


Figure 4.7: The CG (a) expresses $a \perp\!\!\!\perp b \mid \emptyset$ and $d \perp\!\!\!\perp e \mid (c, f)$. No directed graph could express both these conditions since the marginal distribution $p(c, d, e, f)$ is an undirected four cycle, (b). Any DAG on a 4 cycle must contain a collider, as in (c) and therefore express a different set of CI statements than (b). Similarly, no connected Markov network can express unconditional independence and hence (a) expresses CI statements that no Belief Network or Markov Network alone can express.

moralised directed graph) will contain additional links and independence information can be lost. For example, the MN of $p(c|a, b)p(a)p(b)$ if a single clique $\phi(a, b, c)$ from which one cannot graphically infer that $a \perp\!\!\!\perp b|\emptyset$.

The converse question is whether every undirected model can be represented by a BN with a readily derived link structure? Consider the example in fig(4.8). In this case, there is no directed model with the same link structure that can express the (in)dependencies in the undirected graph. Naturally, *every* probability distribution can be represented by some BN though it may not necessarily have a simple structure and be simply a 'fully connected' cascade style graph. In this sense the DAG cannot graphically represent the independence/dependence relations true in the distribution.

Definition 34 (Independence Maps)**.** A graph is an *independence map* (I-map) of a given distribution P if every conditional independence statement that one can derive from the graph G is true in the distribution P. That is

$$\mathcal{X} \perp\!\!\!\perp \mathcal{Y}| \mathcal{Z}_G \Rightarrow \mathcal{X} \perp\!\!\!\perp \mathcal{Y}| \mathcal{Z}_P \tag{4.4.1}$$

for all disjoint sets $\mathcal{X}$,$\mathcal{Y}$,$\mathcal{Z}$.

Similarly, a graph is a *dependence map* (D-map) of a given distribution P if every conditional dependence statement that one can derive from the graph G is true in the distribution P. That is

$$\mathcal{X} \top\!\!\!\top \mathcal{Y}| \mathcal{Z}_G \Rightarrow \mathcal{X} \top\!\!\!\top \mathcal{Y}| \mathcal{Z}_P \tag{4.4.2}$$

for all disjoint sets $\mathcal{X}$,$\mathcal{Y}$,$\mathcal{Z}$.

Due to Inverse Modus Ponens, example(5), the above is equivalent to

$$\mathcal{X} \perp\!\!\!\perp \mathcal{Y}| \mathcal{Z}_G \Leftarrow \mathcal{X} \perp\!\!\!\perp \mathcal{Y}| \mathcal{Z}_P \tag{4.4.3}$$

A graph G which is both an I-map and a D-map for P is called a *perfect map* and

$$\mathcal{X} \perp\!\!\!\perp \mathcal{Y}| \mathcal{Z}_G \Leftrightarrow \mathcal{X} \perp\!\!\!\perp \mathcal{Y}| \mathcal{Z}_P \tag{4.4.4}$$

for all disjoint sets $\mathcal{X}$,$\mathcal{Y}$,$\mathcal{Z}$. In this case, the set of all conditional independence and dependence statements expressible in the graph G are consistent with P and *vice versa*.

Note that the above definitions are not dependent on the graph being directed or undirected. Indeed, some distributions may have a perfect directed map, but no perfect undirected map. For example

$$p(x, y, z) = p(z|x, y)p(x)p(y) \tag{4.4.5}$$

has a directed perfect map $x \to z \leftarrow y$, but no perfect undirected map.

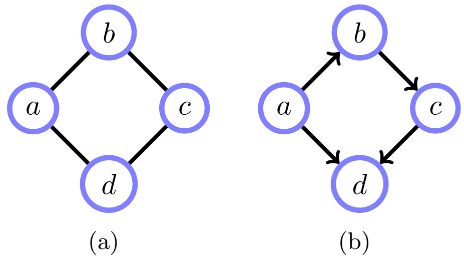


Figure 4.8: **(a)**: An undirected model for which we wish to find a directed equivalent. **(b)**: Every DAG with the same structure as the undirected model must have a situation where two arrows will point to a node, such as node d. Summing over the states of variable d will leave a DAG on the variables a, b, c with no link between a and c. This cannot represent the undirected model since when one marginals over d in the undirected this adds a link between a and c.

Example 18. Consider the distribution defined on variables t_1, t_2, y_1, y_2:

$$p(t_1, t_2, y_1, y_2) = p(t_1)p(t_2) \sum_h p(y_1|t_1, h)p(y_2|t_2, h)p(h) \tag{4.4.6}$$

The BN

$$p(y_2|y_1, t_2)p(y_1|t_1)p(t_1)p(t_2) \tag{4.4.7}$$

is an I-MAP for distribution (4.4.6). However, it is not a D-MAP since it implies $t_1 \perp\!\!\!\perp t_2 | y_2$ which is not true in (4.4.6). Similarly no undirected graph can represent all independence statements true in (4.4.6). In this case no perfect MAP (a BN or a MN) can represent (4.4.6).

4.5 Factor Graphs

Factor Graphs (FGs) are mainly used as part of inference algorithms[2].

Definition 35 (*Factor Graph*)**.** Given a function

$$f(x_1, \ldots, x_n) = \prod_i \psi_i \left(\mathcal{X}^i\right) \tag{4.5.1}$$

The FG has a node (represented by a square) for each factor ψ_i, and a variable node (represented by a circle) for each variable x_j. For each $x_j \in \mathcal{X}^i$ an undirected link is made between factor ψ_i and variable x_j.

For a factor $\psi_i \left(\mathcal{X}^i\right)$ which is a conditional distribution $p(x_i|\text{pa}\,(x_i))$, we may use directed links from the parents to the factor node, and a directed link from the factor node to the child. This has the same structure as an (undirected) FG, but preserves the information that the factors are distributions.

Factor Graphs are useful since they can preserve more information about the form of the distribution than either a Belief Network or a Markov Network (or Chain Graph) can do alone.
Consider the distribution

$$p(a, b, c) = \phi(a, b)\phi(a, c)\phi(b, c) \tag{4.5.2}$$

The MN representation is given in fig(4.9c). However, fig(4.9c) could equally represent some unfactored clique potential $\phi(a, b, c)$. In this sense, the FG representation in fig(4.9b) more precisely conveys the form of distribution equation (4.5.2). An unfactored clique potential $\phi(a, b, c)$ is represented by the FG

[2]Formally a FG is an alternative graphical depiction of a hypergraph[80] in which the vertices represent variables, and a hyperedge a factor as a function of the variables associated with the hyperedge. A FG is therefore a hypergraph with the additional interpretation that the graph represents a function defined as products over the associated hyperedges. Many thanks to Robert Cowell for this observation.

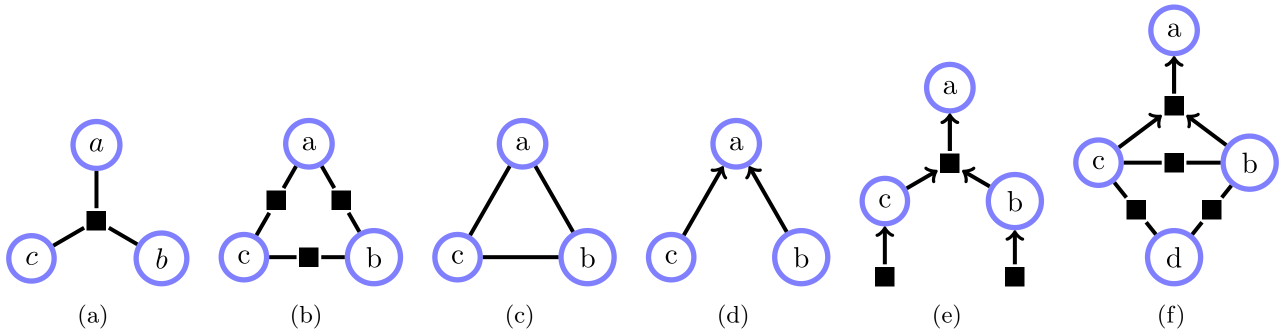


Figure 4.9: **(a)**: $\phi(a, b, c)$. **(b)**: $\phi(a, b)\phi(b, c)\phi(c, a)$. **(c)**: $\phi(a, b, c)$. Both (a) and (b) have the same undirected graphical model, (c). **(e)**: Directed FG of the BN in (d). (a) is an undirected FG of (d). The advantage of (e) over (a) is that information regarding the marginal independence of variables b and c is clear from graph (e), whereas one could only ascertain this by examination of the numerical entries of the factors in graph (a). **(f)**: A partially directed FG of $p(a|b, c)\phi(d, c)\phi(b, d)$. No directed, undirected or chain graph can represent both the conditional and marginal independence statements expressed by this graph and also the factored structure of the undirected terms.

fig(4.9a). Hence different FGs can have the same MN since information regarding the structure of the clique potential is lost in the MN.

4.5.1 Conditional independence in factor graphs

A rule which works with both directed and undirected (and partially directed) FGs is as follows[93]. To determine whether two variables are independent given a set of conditioned variables, consider all paths connecting the two variables. If all paths are blocked, the variables are conditionally independent.

A path is blocked if any one or more of the following conditions are satisfied:

- One of the variables in the path is in the conditioning set.
- One of the variables or factors in the path has two incoming edges that are part of the path, and neither the variable or factor nor any of its descendants are in the conditioning set.

4.6 Notes

A detailed discussion of the axiomatic and logical basis of conditional independence is given in [45] and [261].

4.7 Code

`condindep.m`: Conditional Independence test $p(X, Y|Z) = p(X|Z)p(Y|Z)$?

4.8 Exercises

Exercise 38. *1. Consider the pairwise Markov Network,*

$$p(x) = \phi(x_1, x_2)\phi(x_2, x_3)\phi(x_3, x_4)\phi(x_4, x_1) \tag{4.8.1}$$

Express in terms of ϕ the following:

$$p(x_1|x_2, x_4), \qquad p(x_2|x_1, x_3), \qquad p(x_3|x_2, x_4), \qquad p(x_4|x_1, x_3) \tag{4.8.2}$$

2. *For a set of local distributions defined as*

$$p_1(x_1|x_2, x_4), \qquad p_2(x_2|x_1, x_3), \qquad p_3(x_3|x_2, x_4), \qquad p_4(x_4|x_1, x_3) \tag{4.8.3}$$

is it always possible to find a joint distribution $p(x_1, x_2, x_3, x_4)$ consistent with these local conditional distributions?

Exercise 39. *Consider the Markov network*

$$p(a, b, c) = \phi_{ab}(a, b)\phi_{bc}(b, c) \tag{4.8.4}$$

Nominally, by summing over b, the variables a and c are dependent. For binary b, explain a situation in which this is not the case, so that marginally, a and c are independent.

Exercise 40. *Show that for the Boltzmann machine*

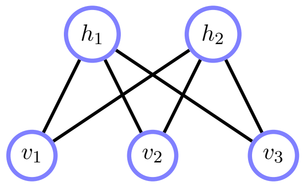
$$p(\mathbf{x}) = \frac{1}{Z(\mathbf{W}, \mathbf{b})} e^{\mathbf{x}^\mathsf{T}\mathbf{W}\mathbf{x}+\mathbf{x}^\mathsf{T}\mathbf{b}} \tag{4.8.5}$$

one may assume, without loss of generality, $\mathbf{W} = \mathbf{W}^\mathsf{T}$.

Exercise 41.

The restricted Boltzmann machine (or Harmonium[250]) is a specially constrained Boltzmann machine on a bipartite graph, consisting of a layer of visible variables $\mathbf{v} = (v_1, \ldots, v_V)$ and hidden variables $\mathbf{h} = (h_1, \ldots, h_H)$:

$$p(\mathbf{v}, \mathbf{h}) = \frac{1}{Z(\mathbf{W}, \mathbf{a}, \mathbf{b})} e^{\mathbf{v}^\mathsf{T}\mathbf{W}\mathbf{h}+\mathbf{a}^\mathsf{T}\mathbf{v}+\mathbf{b}^\mathsf{T}\mathbf{h}} \tag{4.8.6}$$



All variables are binary taking states $0, 1$.

1. *Show that the distribution of hidden units conditional on the visible units factorises as*

$$p(\mathbf{h}|\mathbf{v}) = \prod_i p(h_i|\mathbf{v}), \qquad \text{with } p(h_i|\mathbf{v}) = \sigma\left(b_i + \sum_j W_{ji}v_j\right) \tag{4.8.7}$$

where $\sigma(x) = e^x/(1 + e^x)$.

2. *By symmetry arguments, write down the form of the conditional $p(\mathbf{v}|\mathbf{h})$.*

3. *Is $p(\mathbf{h})$ factorised?*

4. *Can the partition function $Z(\mathbf{W}, \mathbf{a}, \mathbf{b})$ be computed efficiently for the RBM?*

Exercise 42. *Consider*

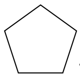
$$p(x) = \phi(x_1, x_{100}) \prod_{i=1}^{99} \phi(x_i, x_{i+1}) \tag{4.8.8}$$

Is it possible to compute $\underset{x_1,\ldots,x_{100}}{\operatorname{argmax}}\, p(x)$ efficiently?

Exercise 43. *You are given that*

$$x \perp\!\!\!\perp y|\,(z, u)\,, \qquad u \perp\!\!\!\perp z|\emptyset \tag{4.8.9}$$

Derive the most general form of probability distribution $p(x, y, z, u)$ consistent with these statements. Does this distribution have a simple graphical model?

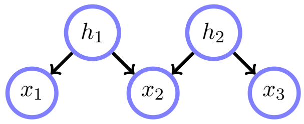
Exercise 44. *The undirected graph*  *represents a Markov Network with nodes x_1, x_2, x_3, x_4, x_5, counting clockwise around the pentagon with potentials $\phi(x_i, x_{1+mod(i,5)})$. Show that the joint distribution can be written as*

$$p(x_1, x_2, x_3, x_4, x_5) = \frac{p(x_1, x_2, x_5)p(x_2, x_4, x_5)p(x_2, x_3, x_4)}{p(x_2, x_5)p(x_2, x_4)} \tag{4.8.10}$$

and express the marginal probability tables explicitly as functions of the potentials $\phi(x_i, x_j)$.

Exercise 45.

Consider the Belief Network on the right.



1. *Write down a Markov Network of $p(x_1, x_2, x_3)$.*
2. *Is your Markov Network a perfect map of $p(x_1, x_2, x_3)$?*

Exercise 46. *Two research labs work independently on the relationship between discrete variables x and y. Lab A proudly announces that they have ascertained distribution $p_A(x|y)$ from data. Lab B proudly announces that they have ascertained $p_B(y|x)$ from data.*

1. *Is it always possible to find a joint distribution $p(x, y)$ consistent with the results of both labs?*
2. *Is it possible to define consistent marginals $p(x)$ and $p(y)$, in the sense that $p(x) = \sum_y p_A(x|y)p(y)$ and $p(y) = \sum_x p_B(y|x)p(x)$? If so, explain how to find such marginals. If not, explain why not.*

Exercise 47. *Research lab A states its findings about a set of variables $x_1, \ldots, x_n$ as a list L_A of conditional independence statements. Lab B similarly provides a list of conditional independence statements L_B.*

1. *Is it possible to find a distribution which is consistent with L_A and L_B?*
2. *If the lists also contain dependence statements, how could one attempt to find a distribution that is consistent with both lists?*

Exercise 48.
Consider the distribution

$$p(x, y, w, z) = p(z|w)p(w|x, y)p(x)p(y) \tag{4.8.11}$$

1. *Write $p(x|z)$ using a formula involving (all or some of) $p(z|w)$, $p(w|x, y)$, $p(x)$, $p(y)$.*
2. *Write $p(y|z)$ using a formula involving (all or some of) $p(z|w)$, $p(w|x, y)$, $p(x)$, $p(y)$.*
3. *Using the above results, derive an explicit condition for $x \perp\!\!\!\perp y|\, z$ and explain if this is satisfied for this distribution.*

Exercise 49. *Consider the distribution*

$$p(t_1, t_2, y_1, y_2, h) = p(y_1|t_1, h)p(y_2|t_2, h)p(t_1)p(t_2)p(h) \tag{4.8.12}$$

1. *Draw a Belief Network for this distribution.*
2. *Can the distribution*

$$p(t_1, t_2, y_1, y_2) = \sum_h p(y_1|t_1, h)p(y_2|t_2, h)p(t_1)p(t_2)p(h) \tag{4.8.13}$$

   be written as a ('non-complete') Belief Network?
3. *Show that for $p(t_1, t_2, y_1, y_2)$ as defined above $t_1 \perp\!\!\!\perp y_2|\emptyset$.*

Exercise 50. *Consider the distribution*

$$p(a, b, c, d) = \phi_{ab}(a, b)\phi_{bc}(b, c)\phi_{cd}(c, d)\phi_{da}(d, a) \tag{4.8.14}$$

where the ϕ *are potentials.*

1. *Draw a Markov Network for this distribution.*

2. *Explain if the distribution can be represented as a ('non-complete') Belief Network.*

3. *Derive explicitly if* $a \perp\!\!\!\perp c | \emptyset$.

Exercise 51. *Show how for any singly-connected Markov network, one may construct a Markov equivalent Belief Network.*

Exercise 52. *Consider a pairwise binary Markov network defined on variables* $s_i \in \{0, 1\}$, $i = 1, \ldots, N$, *with* $p(s) = \prod_{ij \in \mathcal{E}} \phi_{ij}(s_i, s_j)$, *where* $\mathcal{E}$ *is a given edge set and the potentials* ϕ_{ij} *are arbitrary. Explain how to translate such a Markov network into a Boltzmann machine.*

CHAPTER 5

Efficient Inference in Trees

5.1 Marginal Inference

Given a distribution $p(x_1, \ldots, x_n)$, inference is the process of computing functions of the distribution. For example, computing a marginal conditioned on a subset of variables being in a particular state would be an inference task. Similarly, computing the mean of a variable can be considered an inference task. The main focus of this chapter is on efficient inference algorithms for marginal inference in singly-connected structures. An efficient algorithm for multiply-connected graphs will be considered in chapter(6). Marginal *inference* is concerned with the computation of the distribution of a subset of variables, possibly conditioned on another subset. For example, given a joint distribution $p(x_1, x_2, x_3, x_4, x_5)$, a marginal inference given evidence calculation is

$$p(x_5|x_1 = \mathsf{tr}) \propto \sum_{x_2,x_3,x_4} p(x_1 = \mathsf{tr}, x_2, x_3, x_4, x_5) \tag{5.1.1}$$

Marginal inference for discrete models involves summation and will be the focus of our development. In principle the algorithms carry over to continuous variable models although the lack of closure of most continuous distributions under marginalisation (the Gaussian being a notable exception) can make the direct transference of these algorithms to the continuous domain problematic.

5.1.1 Variable elimination in a Markov chain and message passing

A key concept in efficient inference is *message passing* in which information from the graph is summarised by local edge information. To develop this idea, consider the four variable Markov chain (Markov chains are discussed in more depth in section(23.1))

$$p(a, b, c, d) = p(a|b)p(b|c)p(c|d)p(d) \tag{5.1.2}$$

as given in fig(5.1), for which our task is to calculate the marginal $p(a)$. For simplicity, we assume that each of the variables has domain $\{0, 1\}$. Then

$$p(a = 0) = \sum_{b\in\{0,1\},c\in\{0,1\},d\in\{0,1\}} p(a = 0, b, c, d) = \sum_{b\in\{0,1\},c\in\{0,1\},d\in\{0,1\}} p(a = 0|b)p(b|c)p(c|d)p(d) \tag{5.1.3}$$

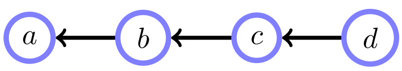


Figure 5.1: A Markov chain is of the form $p(x_T) \prod_{t=1}^{T-1} p(x_t|x_{t+1})$ for some assignment of the variables to labels x_t. Variable Elimination can be carried out in time linear in the number of variables in the chain.

We could carry out this computation by simply summing each of the probabilities for the $2 \times 2 \times 2 = 8$ states of the variables b,c and d.

A more efficient approach is to push the summation over d as far to the right as possible:

$$p(a = 0) = \sum_{b\in\{0,1\},c\in\{0,1\}} p(a = 0|b)p(b|c) \underbrace{\sum_{d\in\{0,1\}} p(c|d)p(d)}_{\gamma_d(c)} \tag{5.1.4}$$

where $\gamma_d(c)$ is a (two state) *potential*. Similarly, we can distribute the summation over c as far to the right as possible:

$$p(a = 0) = \sum_{b\in\{0,1\}} p(a = 0|b) \underbrace{\sum_{c\in\{0,1\}} p(b|c)\gamma_d(c)}_{\gamma_c(b)} \tag{5.1.5}$$

Then, finally,

$$p(a = 0) = \sum_{b\in\{0,1\}} p(a = 0|b)\gamma_c(b) \tag{5.1.6}$$

By distributing the summations we have made $2 + 2 + 2 = 6$ additions, compared to 8 from the naive approach. Whilst this saving may not appear much, the important point is that the number of computations for a chain of length T would scale linearly with T, as opposed to exponentially for the naive approach.

This procedure is naturally enough called *variable elimination*, since each time we sum over the states of a variable, we eliminate it from the distribution. We can always perform variable elimination in a chain efficiently since there is a natural way to distribute the summations, working inwards from the edges. Note that in the above case, the potentials are in fact always distributions – we are just recursively computing the marginal distribution of the right leaf of the chain.

One can view the elimination of a variable as passing a *message* (information) to a neighbouring vertex on the graph. We can calculate a univariate-marginal of any singly-connected graph by starting at a leaf of the tree, eliminating the variable there, and then working inwards, nibbling off each time a leaf of the remaining tree. Provided we perform elimination from the leaves inwards, then the structure of the remaining graph is simply a subtree of the original tree, albeit with the conditional probability table entries modified to potentials which update under recursion. This is guaranteed to enable us to calculate any marginal $p(x_i)$ using a number of summations which scales linearly with the number of variables in the graph.

Finding conditional marginals for a chain

Consider the following inference problem, fig(5.1) : Given

$$p(a, b, c, d) = p(a|b)p(b|c)p(c|d)p(d), \tag{5.1.7}$$

find $p(d|\mathsf{a})$. This can be computed using

$$p(d|\mathsf{a}) \propto \sum_{b,c} p(\mathsf{a}, b, c, d) \propto \sum_{b,c} p(\mathsf{a}|b)p(b|c)p(c|d)p(d) \propto \sum_{c} \underbrace{\sum_{b} p(\mathsf{a}|b)p(b|c)}_{\gamma_b(c)} p(c|d)p(d) \equiv \gamma_c(d) \tag{5.1.8}$$

The missing proportionality constant is found by repeating the computation for all states of variable d. Since we know that $p(d|\mathsf{a}) = k\gamma_c(d)$, where $\gamma_c(d)$ is the unnormalised result of the summation, we can use the fact that $\sum_d p(d|\mathsf{a}) = 1$ to infer that $k = 1/\sum_d \gamma_c(d)$.

In this example, the potential $\gamma_b(c)$ is not a distribution in c, nor is $\gamma_c(d)$. In general, one may view variable elimination as the passing of messages in the form of potentials from nodes to their neighbours. For Belief Networks, variable elimination passes messages that are distributions when following the direction of the edge, and non-normalised potentials when passing messages against the direction of the edge.

Remark 4. Variable Elimination in Trees as Matrix Multiplication

Variable Elimination is related to the associativity of matrix multiplication. For equation (5.1.2) above, we can define matrices

$$[\mathbf{M}_{ab}]_{i,j} = p(a = i|b = j), \quad [\mathbf{M}_{bc}]_{i,j} = p(b = i|c = j),$$
$$[\mathbf{M}_{cd}]_{i,j} = p(c = i|d = j), \quad [\mathbf{M}_d]_i = p(d = i), \quad [\mathbf{M}_a]_i = p(a = i) \tag{5.1.9}$$

Then the marginal $\mathbf{M}_a$ can be written

$$\mathbf{M}_a = \mathbf{M}_{ab}\mathbf{M}_{bc}\mathbf{M}_{cd}\mathbf{M}_d = \mathbf{M}_{ab}(\mathbf{M}_{bc}(\mathbf{M}_{cd}\mathbf{M}_d)) \tag{5.1.10}$$

since matrix multiplication is associative. This matrix formulation of calculating marginals is called the *transfer matrix* method, and is particularly popular in the physics literature[26].

Example 19 (Where will the fly be?).

You live in a house with three rooms, labelled $1, 2, 3$. There is a door between rooms 1 and 2 and another between rooms 2 and 3. One cannot directly pass between rooms 1 and 3 in one time-step. An annoying fly is buzzing from one room to another and there is some smelly cheese in room 1 which seems to attract the fly more. Using $x(t)$ for which room the fly is in at time t, with $\text{dom}(x(t)) = \{1, 2, 3\}$, the movement of the fly can be described by a transition

$$p(x(t + 1) = i|x(t) = j) = M_{ij} \tag{5.1.11}$$

where $\mathbf{M}$ is a transition matrix

$$\mathbf{M} = \begin{pmatrix} 0.7 & 0.5 & 0 \\ 0.3 & 0.3 & 0.5 \\ 0 & 0.2 & 0.5 \end{pmatrix} \tag{5.1.12}$$

The transition matrix is stochastic in the sense that, as required of a conditional probability distribution $\sum_{i=1}^{3} M_{ij} = 1$. Given that the fly is in room 1 at time 1, what is the probability of room occupancy at time $t = 5$? Assume a Markov chain which is defined by the joint distribution

$$p(x(1), \ldots, x(T)) = \prod_{t=1}^{T-1} p(x(t + 1)|x(t)) \tag{5.1.13}$$

We are asked to compute $p(x(t = 5)|x(t = 1) = 1)$ which is given by

$$\sum_{x(t=4),x(t=3),x(t=2)} p(x(t = 5)|x(t = 4))p(x(t = 4)|x(t = 3))p(x(t = 3)|x(t = 2))p(x(t = 2)|x(t = 1) = 1) \tag{5.1.14}$$

Since the graph of the distribution is a Markov chain, we can easily distribute the summation over the terms. This is most easily done using the transfer matrix method, giving

$$p(x(t = 5) = i|x(t = 1) = 1) = [\mathbf{M}^4\mathbf{v}]_i \tag{5.1.15}$$

where $\mathbf{v}$ is a vector with components $(1,0,0)^\mathsf{T}$, reflecting the evidence that at time 1 the fly is in room 1. Computing this we have (to 4 decimal places of accuracy)

$$\mathbf{M}^4\mathbf{v} = \begin{pmatrix} 0.5746 \\ 0.3180 \\ 0.1074 \end{pmatrix} \tag{5.1.16}$$

Similarly, after 5 time-steps, the occupancy probabilities are $(0.5612, 0.3215, 0.1173)$. The room occupancy probability is converging to a particular distribution – the *stationary* distribution of the Markov chain. One might ask where the fly is after an *infinite* number of time-steps. That is, we are interested in the large t behaviour of

$$p(x(t+1)) = \sum_{x(t)} p(x(t+1)|x(t))p(x(t)) \tag{5.1.17}$$

At convergence $p(x(t+1)) = p(x(t))$. Writing $\mathbf{p}$ for the vector describing the stationary distribution, this means

$$\mathbf{p} = \mathbf{M}\mathbf{p} \tag{5.1.18}$$

In other words, $\mathbf{p}$ is the eigenvector of $\mathbf{M}$ with eigenvalue 1. It is a well known property of stochastic matrices that they always possess such an eigenvector[120]. Computing this numerically, the stationary distribution is $(0.5435, 0.3261, 0.1304)$. Note that MATLAB returns eigenvectors with $\mathbf{e}^\mathsf{T}\mathbf{e} = 1$ – the unit eigenvector therefore will usually require normalisation to make this a probability.

5.1.2 The Sum-Product algorithm on Factor Graphs

Both Markov and Belief Networks can be represented using Factor Graphs. For this reason it is convenient to derive a marginal inference algorithm for the FG since this then applies to both Markov and Belief Networks. This is termed the sum-product algorithm since to compute marginals we need to distribute the sum over variable states over the product of factors. In older texts, this is referred to as *belief propagation*.

Non-branching graphs : variable to variable messages

Consider the distribution

$$p(a,b,c,d) = f_1(a,b)\, f_2(b,c)\, f_3(c,d)\, f_4(d) \tag{5.1.19}$$

which has the factor graph represented in fig(5.2) with factors as defined by the f above. To compute the marginal $p(a,b,c)$, since the variable d only occurs locally, we use

$$p(a,b,c) = \sum_d p(a,b,c,d) = \sum_d f_1(a,b)\, f_2(b,c)\, f_3(c,d)\, f_4(d) = f_1(a,b)\, f_2(b,c) \underbrace{\sum_d f_3(c,d)\, f_4(d)}_{\mu_{d\to c}(c)} \tag{5.1.20}$$

Similarly,

$$p(a,b) = \sum_c p(a,b,c) = f_1(a,b) \underbrace{\sum_c f_2(b,c)\, \mu_{d\to c}(c)}_{\mu_{c\to b}(b)} \tag{5.1.21}$$

Hence

$$\mu_{c\to b}(b) = \sum_c f_2(b,c)\, \mu_{d\to c}(c) \tag{5.1.22}$$

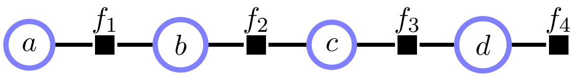


Figure 5.2: For singly-connected structures without branches, simple messages from one variable to its neighbour may be defined to form an efficient marginal inference scheme.

It is clear how one can recurse this definition of messages so that for a chain of length n variables the marginal of the first node can be computed in time linear in n. The term $\mu_{c\to b}(b)$ can be interpreted as carrying marginal information from the graph beyond c.

For any singly-connected structure the factors at the edge of the graph can be replaced with messages that reflect marginal information from the graph beyond that factor. For simple linear structures with no branching, messages from variables to variables are sufficient. However, as we will see below, it is useful in more general structures with branching to consider two types of messages, namely those from variables to factors and vice versa.

General singly-connected Factor Graphs

The slightly more complex example,

$$p(a|b)p(b|c,d)p(c)p(d)p(e|d) \tag{5.1.23}$$

has the factor graph, fig(5.3)

$$f_1(a,b)\, f_2(b,c,d)\, f_3(c)\, f_4(d,e)\, f_5(d) \tag{5.1.24}$$

If the marginal $p(a,b)$ is to be represented by the amputated graph with messages on the edges, then

$$p(a,b) = f_1(a,b) \underbrace{\sum_{c,d} f_2(b,c,d)\, f_3(c)\, f_5(d) \sum_e f_4(d,e)}_{\mu_{f_2\to b}(b)} \tag{5.1.25}$$

In this case it is natural to consider messages from factors to variables. Similarly, we can break the message from the factor f_2 into messages arriving from the two branches through c and d, namely

$$\mu_{f_2\to b}(b) = \sum_{c,d} f_2(b,c,d) \underbrace{f_3(c)}_{\mu_{c\to f_2}(c)} \underbrace{f_5(d) \sum_e f_4(d,e)}_{\mu_{d\to f_2}(d)} \tag{5.1.26}$$

Similarly, we can interpret

$$\mu_{d\to f_2}(d) = \underbrace{f_5(d)}_{\mu_{f_5\to d}(d)} \underbrace{\sum_e f_4(d,e)}_{\mu_{f_4\to d}(d)} \tag{5.1.27}$$

To complete the interpretation we identify $\mu_{c\to f_2}(c) \equiv \mu_{f_3\to c}(c)$. In a non-branching link, one can more simply use a variable to variable message.

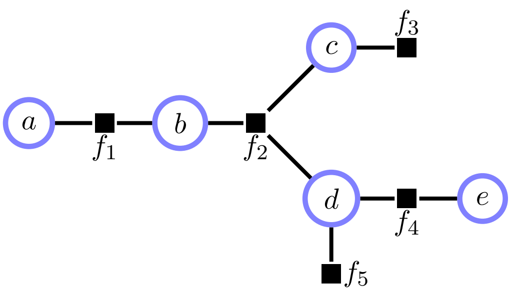


Figure 5.3: For a branching singly-connected graph, it is useful to define messages from both factors to variables, and variables to factors.

To compute the marginal $p(a)$, we then have

$$p(a) = \underbrace{\sum_b f_1(a,b)\,\mu_{f_2 \to b}(b)}_{\mu_{f_1 \to a}(a)} \tag{5.1.28}$$

For consistency of interpretation, one also can view the above as

$$\mu_{f_1 \to a}(a) = \sum_b f_1(a,b)\,\underbrace{\mu_{f_2 \to b}(b)}_{\mu_{b \to f_1}(b)} \tag{5.1.29}$$

A convenience of this approach is that the messages can be reused to evaluate other marginal inferences. For example, it is clear that $p(b)$ is given by

$$p(b) = \underbrace{\sum_a f_1(a,b)}_{\mu_{f_1 \to b}(b)}\,\mu_{f_2 \to b}(b) \tag{5.1.30}$$

If we additionally desire $p(c)$, we need to define the message from f_2 to c,

$$\mu_{f_2 \to c}(c) = \sum_{b,d} f_2(b,c,d)\,\mu_{b \to f_2}(b)\,\mu_{d \to f_2}(d) \tag{5.1.31}$$

where $\mu_{b \to f_2}(b) \equiv \mu_{f_1 \to b}(b)$. This demonstrates the reuse of already computed message from d to f_2 to compute the marginal $p(c)$.

Definition 36 (*Message schedule*)**.** A message schedule is a specified sequence of message updates. A valid schedule is that a message can be sent from a node only when that node has received all requisite messages from its neighbours. In general, there is more than one valid updating schedule.

Sum-Product algorithm

The sum-product algorithm is described below in which messages are updated as a function of incoming messages. One then proceeds by computing the messages in a schedule that allows the computation of a new message based on previously computed messages, until all messages from all factors to variables and vice-versa have been computed.

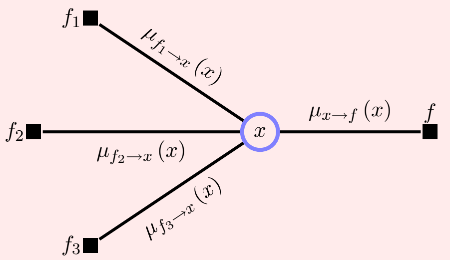
Definition 37 (Sum-Product messages on Factor Graphs)**.**

Given a distribution defined as a product on subsets of the variables, $p(\mathcal{X}) = \frac{1}{Z}\prod_f \phi_f\left(\mathcal{X}^f\right)$, provided the factor graph is singly-connected we can carry out summation over the variables efficiently.

Initialisation Messages from extremal (simplical) node factors are initialised to the factor. Messages from extremal (simplical) variable nodes are set to unity.

Variable to Factor message

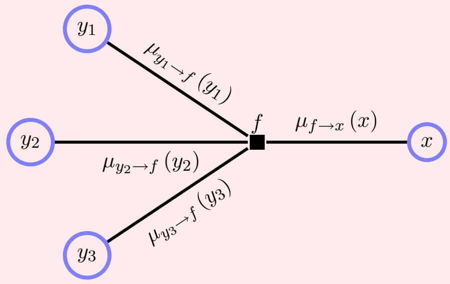
$$\mu_{x \to f}(x) = \prod_{g \in \{\text{ne}(x) \setminus f\}} \mu_{g \to x}(x)$$



Factor to Variable message

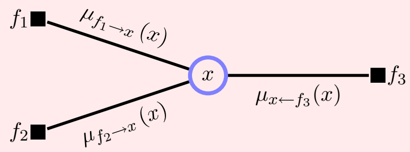
$$\mu_{f \to x}(x) = \sum_{\mathsf{y} \in \mathcal{X}^f \setminus x} \phi_f(\mathcal{X}^f) \prod_{y \in \{\mathrm{ne}(f) \setminus x\}} \mu_{y \to f}(y)$$

We write $\sum_{\mathsf{y} \in \mathcal{X}^f \setminus x}$ to emphasise that we sum over all states in the set of variables $\mathcal{X}^f \setminus x$.



Marginal

$$p(x) \propto \prod_{f \in \mathrm{ne}(x)} \mu_{f \to x}(x)$$



For marginal inference, the important information is the relative size of the message states so that we may renormalise messages as we wish. Since the marginal will be proportional to the incoming messages for that variable, the normalisation constant is trivially obtained using the fact that the marginal must sum to 1. However, if we wish to also compute any normalisation constant using these messages, we cannot normalise the messages since this global information will then be lost. To resolve this one may work with log messages to avoid numerical under/overflow problems.

The sum-product algorithm is able to perform efficient marginal inference in both Belief and Markov Networks, since both are expressible as Factor Graphs. This is the reason for the preferred use of the Factor Graph since it requires only a single algorithm and is agnostic to whether or not the graph is a locally or globally normalised distribution.

5.1.3 Computing the marginal likelihood

For a distribution defined as products over potentials $\phi_f\left(\mathcal{X}^f\right)$

$$p(x) = \frac{1}{Z} \prod_f \phi_f\left(\mathcal{X}^f\right) \tag{5.1.32}$$

the normalisation is given by

$$Z = \sum_{\mathcal{X}} \prod_f \phi_f\left(\mathcal{X}^f\right) \tag{5.1.33}$$

To compute this summation efficiently we take the product of all incoming messages to an arbitrarily chosen variable x and then sum over the states of that variable:

$$Z = \sum_{\mathsf{x}} \prod_{f \in \mathrm{ne}(x)} \mu_{f \to x}(x) \tag{5.1.34}$$

If the factor graph is derived from setting a subset of variables of a BN in evidential states

$$p(\mathcal{X}, \mathcal{V}) = \frac{p(\mathcal{X}|\mathcal{V})}{p(\mathcal{V})} \tag{5.1.35}$$

then the summation over all non-evidential variables will yield the marginal on the visible (evidential) variables, $p(\mathcal{V})$.

For this method to work, the absolute (not relative) values of the messages are required, which prohibits renormalisation at each stage of the message passing procedure. However, without normalisation the

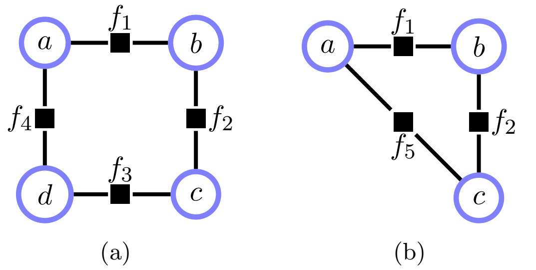


Figure 5.4: (a) Factor Graph with a loop. (b) Eliminating the variable d adds an edge between a and c, demonstrating that, in general, one cannot perform marginal inference in loopy graphs by simply passing messages along existing edges in the original graph.

numerical value of messages can become very small, particularly for large graphs, and numerical precision issues can occur. A remedy in this situation is to work with log messages,

$$\lambda = \log \mu \tag{5.1.36}$$

For this, the variable to factor messages

$$\mu_{x\to f}(x) = \prod_{g\in\{\text{ne}(x)\backslash f\}} \mu_{g\to x}(x) \tag{5.1.37}$$

become simply

$$\lambda_{x\to f}(x) = \sum_{g\in\{\text{ne}(x)\backslash f\}} \lambda_{g\to x}(x) \tag{5.1.38}$$

More care is required for the factors to variable messages, which are defined by

$$\mu_{f\to x}(x) = \sum_{\text{y}\in\mathcal{X}^f\backslash x} \phi_f(\mathcal{X}^f) \prod_{y\in\{\text{ne}(f)\backslash x\}} \mu_{y\to f}(y) \tag{5.1.39}$$

Naively, one may write

$$\lambda_{f\to x}(x) = \log\left(\sum_{\text{y}\in\mathcal{X}^f\backslash x} \phi_f(\mathcal{X}^f) e^{\sum_{y\in\{\text{ne}(f)\backslash x\}}\lambda_{y\to f}(y)}\right) \tag{5.1.40}$$

However, the exponentiation of the log messages will cause potential numerical precision problems. A solution to this numerical difficulty is obtained by finding the largest value of the incoming log messages,

$$\lambda^*_{y\to f} = \max_{y\in\{\text{ne}(f)\backslash x\}} \lambda_{y\to f}(y) \tag{5.1.41}$$

Then

$$\lambda_{f\to x}(x) = \lambda^*_{y\to f} + \log\left(\sum_{y\in\mathcal{X}^f\backslash x} \phi_f(\mathcal{X}^f) e^{\sum_{y\in\{\text{ne}(f)\backslash x\}}\lambda_{y\to f}(y)-\lambda^*_{y\to f}}\right) \tag{5.1.42}$$

By construction the terms $e^{\sum_{y\in\{\text{ne}(f)\backslash x\}}\lambda_{y\to f}(y)-\lambda^*_{y\to f}}$ will be ≤ 1. This ensures that the dominant numerical contributions to the summation are computed accurately.

Log marginals are readily found using

$$\log p(x) = \sum_{f\in\{\text{ne}(x)\}} \lambda_{f\to x}(x) \tag{5.1.43}$$

5.1.4 The problem with loops

Loops cause a problem with variable elimination (or message passing) techniques since once a variable is eliminated the structure of the 'amputated' graph in general changes. For example, consider the FG

$$p(a, b, c, d) = f_1(a, b) f_2(b, c) f_3(c, d) f_4(a, d) \tag{5.1.44}$$

The marginal $p(a, b, c)$ is given by

$$p(a, b, c) = f_1(a, b) f_2(b, c) \underbrace{\sum_d f_3(c, d) f_4(a, d)}_{f_5(a,c)} \tag{5.1.45}$$

which adds a link ac in the amputated graph, see fig(5.4). This means that one cannot account for information from variable d by simply updating potentials on links in the original graph – one needs to account for the fact that the structure of the graph changes. The Junction Tree algorithm, chapter(6) is a widely used technique to deal with this and essentially combines variables together in order to make a new singly-connected graph for which the graph structure remains singly-connected under variable elimination.

5.2 Other forms of Inference

5.2.1 Max-Product

A common interest is the most likely state of distribution. That is

$$\underset{x_1, x_2, \ldots, x_n}{\operatorname{argmax}} \; p(x_1, x_2, \ldots, x_n) \tag{5.2.1}$$

To compute this efficiently we exploit any factorisation structure of the distribution, analogous to the sum-product algorithm. That is, we aim to distribute the maximization so that only local computations are required.

To develop the algorithm, consider a function which can be represented as an undirected chain,

$$f(x_1, x_2, x_3, x_4) = \phi(x_1, x_2)\phi(x_2, x_3)\phi(x_3, x_4) \tag{5.2.2}$$

for which we wish to find the joint state x^* which maximises f. Firstly, we calculate the maximum *value* of f. Since potentials are non-negative, we may write

$$\max_{\mathbf{x}} f(\mathbf{x}) = \max_{x_1,x_2,x_3,x_4} \phi(x_1, x_2)\phi(x_2, x_3)\phi(x_3, x_4) = \max_{x_1,x_2,x_3} \phi(x_1, x_2)\phi(x_2, x_3) \underbrace{\max_{x_4} \phi(x_3, x_4)}_{\gamma(x_3)}$$

$$= \max_{x_1,x_2} \phi(x_1, x_2) \underbrace{\max_{x_3} \phi(x_2, x_3)\gamma(x_3)}_{\gamma(x_2)} = \max_{x_1,x_2} \phi(x_1, x_2)\gamma(x_2) = \max_{x_1} \underbrace{\max_{x_2} \phi(x_1, x_2)\gamma(x_2)}_{\gamma(x_1)}$$

The final equation corresponds to solving a single variable optimisation and determines both the optimal value of the function f and also the optimal state $x_1^* = \underset{x_1}{\operatorname{argmax}} \, \gamma(x_1)$. Given x_1^*, the optimal x_2 is given by $x_2^* = \underset{x_2}{\operatorname{argmax}} \, \phi(x_1^*, x_2)\gamma(x_2)$, and similarly $x_3^* = \underset{x_3}{\operatorname{argmax}} \, \phi(x_2^*, x_3)\gamma(x_3)$, and so on. This procedure is called *backtracking*. Note that we could have equally started at the other end of the chain by defining messages γ that pass information from x_i to x_{i+1}.

The chain structure of the function ensures that the maximal value (and its state) can be computed in time which scales *linearly* with the number of factors in the function. There is no requirement here that the function f corresponds to a probability distribution (though the factors must be non-negative).

Example 20. Consider a distribution defined over binary variables:

$$p(a, b, c) \equiv p(a|b)p(b|c)p(c) \tag{5.2.3}$$

with

$$p(a = \mathsf{tr}|b = \mathsf{tr}) = 0.3, p(a = \mathsf{tr}|b = \mathsf{fa}) = 0.2, p(b = \mathsf{tr}|c = \mathsf{tr}) = 0.75$$
$$p(b = \mathsf{tr}|c = \mathsf{fa}) = 0.1, p(c = \mathsf{tr}) = 0.4$$

What is the most likely joint configuration, $\underset{a,b,c}{\text{argmax}}\ p(a, b, c)$?

Naively, we could evaluate $p(a, b, c)$ over all the 8 joint states of a, b, c and select that states with highest probability. A message passing approach is to define

$$\gamma(b) \equiv \max_c p(b|c)p(c) \tag{5.2.4}$$

For the state $b = \mathsf{tr}$,

$$p(b = \mathsf{tr}|c = \mathsf{tr})p(c = \mathsf{tr}) = 0.75 \times 0.4, \qquad p(b = \mathsf{tr}|c = \mathsf{fa})p(c = \mathsf{fa}) = 0.1 \times 0.6 \tag{5.2.5}$$

Hence, $\gamma(b = \mathsf{tr}) = 0.75 \times 0.4 = 0.3$. Similarly, for $b = \mathsf{fa}$,

$$p(b = \mathsf{fa}|c = \mathsf{tr})p(c = \mathsf{tr}) = 0.25 \times 0.4 \qquad p(b = \mathsf{fa}|c = \mathsf{fa})p(c = \mathsf{fa}) = 0.9 \times 0.6 \tag{5.2.6}$$

Hence, $\gamma(b = \mathsf{fa}) = 0.9 \times 0.6 = 0.54$.

We now consider

$$\gamma(a) \equiv \max_b p(a|b)\gamma(b) \tag{5.2.7}$$

For $a = \mathsf{tr}$, the state $b = \mathsf{tr}$ has value

$$p(a = \mathsf{tr}|b = \mathsf{tr})\gamma(b = \mathsf{tr}) = 0.3 \times 0.3 = 0.09 \tag{5.2.8}$$

and state $b = \mathsf{fa}$ has value

$$p(a = \mathsf{tr}|b = \mathsf{fa})\gamma(b = \mathsf{fa}) = 0.2 \times 0.54 = 0.108 \tag{5.2.9}$$

Hence $\gamma(a = \mathsf{tr}) = 0.108$. Similarly, for $a = \mathsf{fa}$, the state $b = \mathsf{tr}$ has value

$$p(a = \mathsf{fa}|b = \mathsf{tr})\gamma(b = \mathsf{tr}) = 0.7 \times 0.3 = 0.21 \tag{5.2.10}$$

and state $b = \mathsf{fa}$ has value

$$p(a = \mathsf{fa}|b = \mathsf{fa})\gamma(b = \mathsf{fa}) = 0.8 \times 0.54 = 0.432 \tag{5.2.11}$$

giving $\gamma(a = \mathsf{fa}) = 0.432$. Now we can compute the optimal state

$$a^* = \underset{a}{\text{argmax}}\ \gamma(a) = \mathsf{fa} \tag{5.2.12}$$

Given this optimal state, we can backtrack, giving

$$b^* = \underset{b}{\text{argmax}}\ p(a = \mathsf{fa}|b)\gamma(b) = \mathsf{fa}, \quad c^* = \underset{c}{\text{argmax}}\ p(b = \mathsf{fa}|c)p(c) = \mathsf{fa} \tag{5.2.13}$$

Note that in the backtracking process, we already have all the information required from the computation of the messages γ.

Using a factor graph

One can also use the factor graph to compute the joint most probable state. Provided that a full schedule of message passing has occurred, the product of messages into a variable equals the maximum value of the joint function with respect to all other variables. One can then simply read off the most probable state by maximising this local potential.

One then proceeds in computing the messages in a schedule that allows the computation of a new message based on previously computed messages, until all messages from all factors to variables and vice-versa have been computed. The message updates are given below.

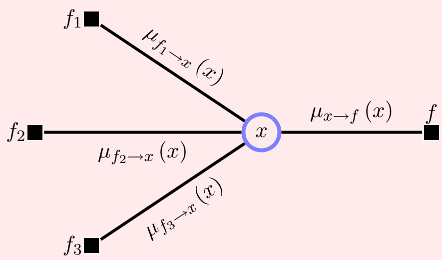
Definition 38 (Max-Product messages on Factor Graphs)**.**

Given a distribution defined as a product on subsets of the variables, $p(\mathcal{X}) = \frac{1}{Z}\prod_f \phi_f\left(\mathcal{X}^f\right)$, provided the factor graph is singly-connected we can carry out maximisation over the variables efficiently.

Initialisation Messages from extremal (simplical) node factors are initialised to the factor. Messages from extremal (simplical) variable nodes are set to unity.

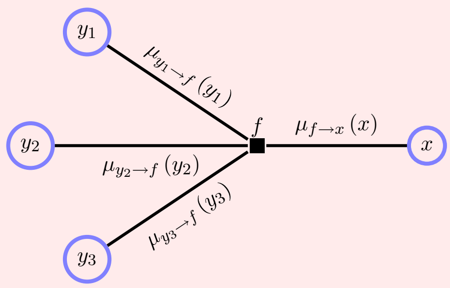
Variable to Factor message

$$\mu_{x\to f}\left(x\right) = \prod_{g\in\{\text{ne}(x)\setminus f\}} \mu_{g\to x}\left(x\right)$$



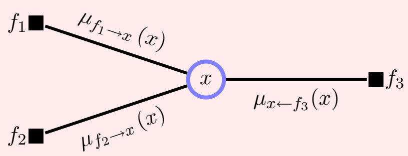
Factor to Variable message

$$\mu_{f\to x}\left(x\right) = \max_{\mathrm{y}\in\mathcal{X}^f\setminus x} \phi_f(\mathcal{X}^f) \prod_{y\in\{\text{ne}(f)\setminus x\}} \mu_{y\to f}\left(y\right)$$



Maximal State

$$\mathrm{x}^* = \underset{\mathrm{x}}{\text{argmax}} \prod_{f\in\text{ne}(x)} \mu_{f\to x}\left(x\right)$$



In earlier literature, this algorithm is called *belief revision*.

5.2.2 Finding the N most probable states

It is often of interest to calculate not just the most likely joint state, but the N most probable states, particularly in cases where the optimal state is only slightly more probable than other states. This is an interesting problem in itself and can be tackled with a variety of methods. A general technique is given by Nilson[208] which is based on the Junction Tree formalism, chapter(6), and the construction of candidate lists, see for example [68].

For singly-connected structures, several approaches have been developed. A special case of Nilsson's approach is available for hidden Markov models[209] which is the particularly efficient for large state spaces.

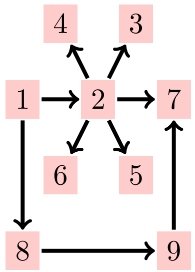


Figure 5.5: State transition diagram (weights not shown). The shortest (unweighted) path from state 1 to state 7 is $1-2-7$. Considered as a Markov chain (random walk), the most probable path from state 1 to state 7 is $1-8-9-7$. The latter path is longer but more probable since for the path $1-2-7$, the probability of exiting from state 2 into state 7 is $1/5$ (assuming each transition is equally likely). See `demoMostProbablePath.m`

For the hidden Markov model, section(23.2) a simple algorithm is the N-Viterbi approach which stores the N-most probable messages at each stage of the propagation, see for example [253].

For more general singly-connected graphs one can extend the max-product algorithm to an N-max-product algorithm by retaining at each stage the N most probable messages, see below. These techniques require N to be specified a-priori compared to anytime alternatives, [295]. An alternative approach for singly-connected networks was developed in [266]. Of particular interest is the application of the singly-connected algorithms as an approximation when for example Nilsson's approach on a multiply-connected graph is intractable[295].

N-max-product

The algorithm for *N-max-product* is a straightforward modification of the standard max-product algorithm in which one retains the N-most likely messages computed at each stage. At a junction of the Factor Graph, all the messages from the neighbours, along with their N-most probable tables are multiplied together into a large table. During the subsequent maximisation over the product on incoming messages, the N-most probable messages are retained, see `maxNprodFG.m`.

5.2.3 Most probable path and shortest path

What is the most likely path from state a to state b for an N state Markov chain? Note that this is not necessarily the same as the shortest path, as explained in fig(5.5).

If assume that a length T path exists, this has probability

$$p(s_2|s_1 = \mathsf{a})p(s_3|s_2) \ldots p(s_T = \mathsf{b}|s_{T-1}) \tag{5.2.14}$$

Finding the most probable path can then be readily solved using the max-product (or max-sum algorithm for the log-transitions) on a simple serial factor graph. To deal with the issue that we don't know the optimal T, one approach is to redefine the probability transitions such that the desired state b is an *absorbing state* of the chain (that is, one can enter this state but not leave it). With this redefinition, the most probable joint state will correspond to the most probable state on the product of N transitions – once the absorbing state is reached the chain will stay in this state, and hence the most probable path can be read off from the sequence of states up to the first time the chain hits the absorbing state. This approach is demonstrated in `demoMostProbablePath.m`, along with the more direct approaches described below.

An alternative, cleaner approach is as follows: for the Markov chain we can dispense with variable-to-factor and factor-to-variable messages and use only variable-to-variable messages. If we want to find the most likely set of states $\mathsf{a}, s_2, \ldots, s_{T-1}, \mathsf{b}$ to get us there, then this can be computed by defining the maximal path probability $E(\mathsf{a} \to \mathsf{b}, T)$ to get from a to b in T-timesteps:

$$E(\mathsf{a} \to \mathsf{b}, T) = \max_{s_2,\ldots,s_{T-1}} p(s_2|s_1 = \mathsf{a})p(s_3|s_2)p(s_4|s_3) \ldots p(s_T = \mathsf{b}|s_{T-1}) \tag{5.2.15}$$

$$= \max_{s_3,\ldots,s_{T-1}} \underbrace{\max_{s_2} p(s_2|s_1 = \mathsf{a})p(s_3|s_2)}_{\gamma_{2\to 3}(s_3)} p(s_4|s_3) \ldots p(s_T = \mathsf{b}|s_{T-1}) \tag{5.2.16}$$

To compute this efficiently we define messages

$$\gamma_{t\to t+1}(s_{t+1}) = \max_{s_t} \gamma_{t-1\to t}(s_t)\, p(s_{t+1}|s_t), \quad t \geq 2, \qquad \gamma_{1\to 2}(s_2) = p(s_2|s_1 = \mathsf{a}) \tag{5.2.17}$$

until the point

$$E(\mathsf{a} \to \mathsf{b}, T) = \max_{s_{T-1}} \gamma_{T-2\to T-1}(s_{T-1})\, p(s_T = \mathsf{b}|s_{T-1}) = \gamma_{T-1\to T}(s_T = \mathsf{b}) \tag{5.2.18}$$

We can now proceed to find the maximal path probability for timestep $T+1$. Since the messages up to time $T-2$ will be the same as before, we need only compute one additional message, $\gamma_{T-1\to T}(s_T)$, from which

$$E(\mathsf{a} \to \mathsf{b}, T+1) = \max_{s_T} \gamma_{T-1\to T}(s_T)\, p(s_{T+1} = \mathsf{b}|s_T) = \gamma_{T\to T+1}(s_{T+1} = \mathsf{b}) \tag{5.2.19}$$

We can proceed in this manner until we reach $E(\mathsf{a} \to \mathsf{b}, N)$ where N is the number of nodes in the graph. We don't need to go beyond this number of steps since those that do must necessarily contain non-simple paths. (A *simple path* is one that does not include the same state more than once.) The optimal time t^* is then given by which of $E(\mathsf{a} \to \mathsf{b}, 2), \ldots, E(\mathsf{a} \to \mathsf{b}, N)$ is maximal. Given t^* one can begin to backtrack[1]. Since

$$E(\mathsf{a} \to \mathsf{b}, t^*) = \max_{s_{t^*-1}} \gamma_{t^*-2\to t^*-1}(s_{t^*-1})\, p(s_{t^*} = \mathsf{b}|s_{t^*-1}) \tag{5.2.20}$$

we know the optimal state

$$\mathsf{s}^*_{t^*-1} = \operatorname*{argmax}_{s_{t^*-1}} \gamma_{t^*-2\to t^*-1}(s_{t^*-1})\, p(s_{t^*} = \mathsf{b}|s_{t^*-1}) \tag{5.2.21}$$

We can then continue to backtrack:

$$\mathsf{s}^*_{t^*-2} = \operatorname*{argmax}_{s_{t^*-2}} \gamma_{t^*-3\to t^*-2}(s_{t^*-2})\, p(\mathsf{s}^*_{t^*-1}|s_{t^*-2}) \tag{5.2.22}$$

and so on. See `mostprobablepath.m`.

- In the above derivation we do not use any properties of probability, except that p must be non-negative (otherwise sign changes can flip a whole sequence 'probability' and the local message recursion no longer applies). One can consider the algorithm as finding the optimal 'product' path from a to b.

- It is straightforward to modify the algorithm to solve the (single-source, single-sink) *shortest weighted path* problem. One way to do this is to replace the Markov log transition probabilities with edge weights $-u(s_t|s_{t-1})$, where $u(s_t|s_{t-1})$ is infinite if there is no edge from s_{t-1} to s_t. This approach is taken in `shortestpath.m` which is able to either positive or negative edge weights. If a negative edge cycle exists, the code returns the shortest weighted length N path, where N is the number of nodes in the graph. See `demoShortestPath.m`. This method is therefore more general than the well-known Dijkstra's algorithm [108] which requires weights to be positive.

- The above algorithm is efficient for the single-source, single-sink scenario, since the messages contain only N states, meaning that the overall storage is $O(N^2)$.

- As it stands, the algorithm is numerically impractical since the messages are recursively multiplied by values usually less than 1 (at least for the case of probabilities). One will therefore quickly run into numerical underflow (or possibly overflow in the case of non-probabilities) with this method.

[1] An alternative to finding t^* is to allow define self-transitions with probability 1, and then use a fixed time $T = N$. Once the desired state b is reached, the self-transition then preserves the chain in state b for the remaining timesteps. This procedure is used in `mostprobablepathmult.m`

To fix the final point above, it is best to work by defining the logarithm of E. Since this is a monotonic transformation, the most probable path defined through $\log E$ is the same as that obtained from E. In this case

$$L(\mathsf{a} \to \mathsf{b}, T) = \max_{s_2,\ldots,s_{T-1}} \log \left[p(s_2|s_1 = \mathsf{a})p(s_3|s_2)p(s_4|s_3)\ldots p(s_T = b|s_{T-1})\right] \tag{5.2.23}$$

$$= \max_{s_2,\ldots,s_{T-1}} \left[\log p(s_2|s_1 = \mathsf{a}) + \sum_{t=2}^{T-1} \log p(s_t|s_{t-1}) + \log p(s_T = \mathsf{b}|s_{T-1})\right] \tag{5.2.24}$$

We can therefore define new messages

$$\lambda_{t\to t+1}(s_{t+1}) = \max_{s_t} \left[\lambda_{t-1\to t}(s_t) + \log p(s_{t+1}|s_t)\right] \tag{5.2.25}$$

One then proceeds as before by finding the most probable t^* defined on L, and backtracks.

Remark 5. A possible confusion is that optimal paths can be efficiently found 'when the graph is loopy'. Note that the graph in fig(5.5) is a state-transition diagram, not a graphical model. The graphical model corresponding to this simple Markov chain is the Belief Network $\prod_t p(s_t|s_{t-1})$, a linear serial structure. Hence the underlying graphical model is a simple chain, which explains why computation is efficient.

Most probable path (multiple-source, multiple-sink)

If we need the most probable path between all states a and b, one could re-run the above single-source-single-sink algorithm for all a and b. A computationally more efficient approach is to observe that one can define a message for each starting state a:

$$\gamma_{t\to t+1}(s_{t+1}|\mathsf{a}) = \max_{s_t} \gamma_{t-1\to t}(s_t|\mathsf{a})\, p(s_{t+1}|s_t) \tag{5.2.26}$$

and continue until we find the maximal path probability matrix for getting from any state a to any state b in T timesteps:

$$E(\mathsf{a} \to \mathsf{b}, T) = \max_{s_{T-1}} \gamma_{T-2\to T-1}(s_{T-1}|\mathsf{a})\, p(s_T = \mathsf{b}|s_{T-1}) \tag{5.2.27}$$

Since we know the message $\gamma_{T-2\to T-1}(s_{T-1}|\mathsf{a})$ for all states a, we can readily compute the most probable path from all starting states a to all states b after T steps. This requires passing an $N \times N$ matrix message γ. We can then proceed to the next timestep $T+1$. Since the messages up to time $T-2$ will be the same as before, we need only compute one additional message, $\gamma_{T-1\to T}(s_T)$, from which

$$E(\mathsf{a} \to \mathsf{b}, T+1) = \max_{s_T} \gamma_{T-1\to T}(s_T|\mathsf{a})\, p(s_{T+1} = \mathsf{b}|s_T) \tag{5.2.28}$$

In this way one can then efficiently compute the optimal path probabilities for any starting state a and end state b after t timesteps. To find the optimal corresponding path, backtracking proceeds as before, see `mostprobablepathmult.m`. One can also use the same algorithm to solve the multiple-source, multiple sink shortest path problem. This algorithm is a variant of the Floyd-Warshall-Roy algorithm[108] for finding shortest weighted summed paths on a directed graph (the above algorithm enumerates through time, whereas the FWR algorithm enumerates through states).

5.2.4 Mixed Inference

An often encountered situation is to infer the most likely state of a joint marginal, possibly given some evidence. For example, given a distribution $p(x_1, \ldots, x_n)$, find

$$\operatorname*{argmax}_{x_1,x_2,\ldots,x_m} p(x_1, x_2, \ldots, x_m) = \operatorname*{argmax}_{x_1,x_2,\ldots,x_m} \sum_{x_{m+1},\ldots,x_n} p(x_1, \ldots, x_n) \tag{5.2.29}$$

Algorithm 1 Compute marginal $p(x_1|\text{evidence})$ from distribution $p(x) = \prod_f \phi_f(\{x\}_f)$. Assumes non-evidential variables are ordered $x_1, \ldots, x_n$.

procedure BUCKET ELIMINATION($p(x) = \prod_f \phi_f(\{x\}_f)$.)
  Initialize all Bucket potentials to unity. ▷ Fill Buckets
  while There are potentials left in the distribution **do**
    For each potential ϕ_f, its highest variable x_j (according to the ordering).
    Multiply ϕ_f with the potential in Bucket j and remove ϕ_f the distribution.
  end while
  for $i =$ Bucket n to 1 **do** ▷ Empty Buckets
    For Bucket i sum over the states of variable x_i and call this potential γ_i
    Identify the highest variable x_h of potential γ_i
    Multiply the existing potential in Bucket h by γ_i
  end for
  The marginal $p(x_1|\text{evidence})$ is proportional to γ_1.
  return $p(x_1|\text{evidence})$ ▷ The conditional marginal.
end procedure

In general, even for tree structured $p(x_1, \ldots, x_n)$, the optimal marginal state cannot be computed efficiently. One way to see this is that due to the summation the resulting joint marginal does not have a structured factored form as products of simpler functions of the marginal variables. Finding the most probable joint marginal then requires a search over all the joint marginal states – an task exponential in m. An approximate solution is provided by the EM algorithm (see section(11.2) and exercise(58)).

5.3 Inference in Multiply-Connected graphs

5.3.1 Bucket Elimination

We consider here a general conditional marginal variable elimination method that works for *any* distribution (including multiply connected graphs). The algorithm assumes the distribution is in the form

$$p(x_1, \ldots, x_n) \propto \prod_f \phi(\mathcal{X}_f) \tag{5.3.1}$$

and that the task is to compute $p(x_1|\text{evidence})$. For example, for

$$p(x_1, x_2, x_3, x_4) = p(x_1|x_2)p(x_2|x_3)p(x_3|x_4)p(x_4) \tag{5.3.2}$$

we could use

$$\phi_1(x_1, x_2) = p(x_1|x_2), \;\; \phi_2(x_2, x_3) = p(x_2|x_3), \;\; \phi_3(x_3, x_4) = p(x_3|x_4)p(x_4) \tag{5.3.3}$$

The sets of variables here are $\mathcal{X}_1 = (x_1, x_2), \mathcal{X}_2 = (x_2, x_3), \mathcal{X}_3 = (x_3, x_4)$. In general, the construction of potentials for a distribution is not unique. The task of computing a marginal in which a set of variables $x_{n+1}, \ldots, \ldots.$ are 'clamped' to their evidential states is

$$p(x_1|\text{evidence}) \propto p(x_1, \text{evidence}) = \sum_{x_2,\ldots,x_n} \prod_f \phi_f(\mathcal{X}_f) \tag{5.3.4}$$

The algorithm is given in algorithm(11) and can be considered a way to organise the distributed summation[78]. The algorithm is best explained by a simple example, as given below.

Example 21 (Bucket Elimination)**.** Consider the problem of calculating the marginal $p(f)$ of

$$p(a, b, c, d, e, f, g) = p(f|d)p(g|d, e)p(c|a)p(d|a, b)p(a)p(b)p(e), \tag{5.3.5}$$

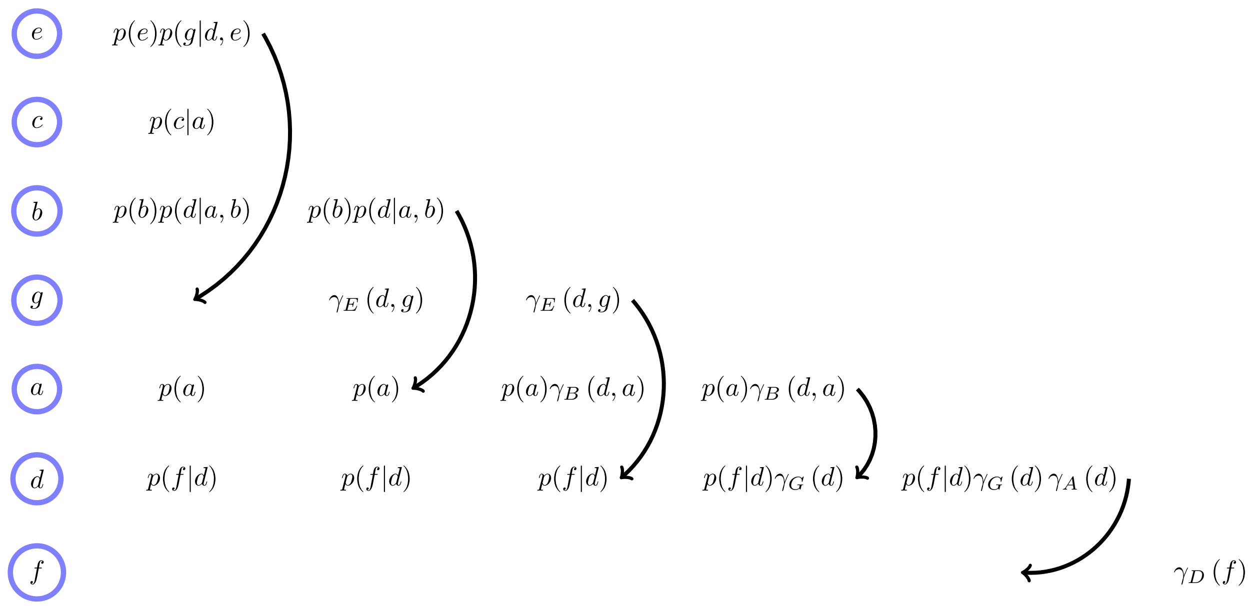


Figure 5.6: The bucket elimination algorithm applied to the graph fig(2.1). At each stage, at least one node is eliminated from the graph. The second stage of eliminating c is trivial since $\sum_c p(c|a) = 1$ and has therefore been skipped over since this bucket does not send any message.

see fig(2.1).

$$p(f) = \sum_{a,b,c,d,e,g} p(a,b,c,d,e,f,g) = \sum_{a,b,c,d,e,g} p(f|d)p(g|d,e)p(c|a)p(d|a,b)p(a)p(b)p(e) \tag{5.3.6}$$

We can distribute the summation over the various terms as follows: e,b and c are end nodes, so that we can sum over their values:

$$p(f) = \sum_{a,d,g} p(f|d)p(a) \left(\sum_b p(d|a,b)p(b)\right) \left(\sum_c p(c|a)\right) \left(\sum_e p(g|d,e)p(e)\right) \tag{5.3.7}$$

For convenience, lets write the terms in the brackets as $\sum_b p(d|a,b)p(b) \equiv \gamma_B(a,d)$, $\sum_e p(g|d,e)p(e) \equiv \gamma_E(d,g)$. The term $\sum_c p(c|a)$ is equal to unity, and we therefore eliminate this node directly. Rearranging terms, we can write

$$p(f) = \sum_{a,d,g} p(f|d)p(a)\gamma_B(a,d)\gamma_E(d,g) \tag{5.3.8}$$

If we think of this graphically, the effect of summing over b,c,e is effectively to remove or 'eliminate' those variables. We can now carry on summing over a and g since these are end points of the new graph:

$$p(f) = \sum_d p(f|d) \left(\sum_a p(a)\gamma_B(a,d)\right) \left(\sum_g \gamma_E(d,g)\right) \tag{5.3.9}$$

Again, this defines new functions $\gamma_A(d)$, $\gamma_G(d)$, so that the final answer can be found from

$$p(f) = \sum_d p(f|d)\gamma_A(d)\gamma_G(d) \tag{5.3.10}$$

We illustrate this in fig(5.6). Initially, we define an ordering of the variables, beginning with the one that we wish to find the marginal for – a suitable ordering is therefore, f,d,a,g,b,c,e. Then starting with the highest bucket e (according to our ordering f,d,a,g,b,c,e), we put all the functions that mention e in the e bucket. Continuing with the next highest bucket, c, we put all the remaining functions that mention c

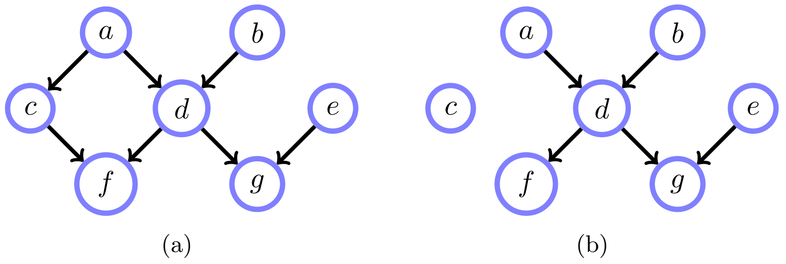


Figure 5.7: A multiply-connected graph (a) reduced to a singly-connected graph (b) by conditioning on the variable c.

in this c bucket, etc. The result of this initialisation procedure is that terms (conditional distributions) in the DAG are distributed over the buckets, as shown in the left most column of fig(5.6). Eliminating then the highest bucket e, we pass a message to node g. Immediately, we can also eliminate bucket c since this sums to unity. In the next column, we have now two less buckets, and we eliminate the highest remaining bucket, this time b, passing a message to bucket a.

There are some important observations we can make about bucket elimination:

1. To compute say $p(x_2|\mathsf{evidence})$ we need to re-order the variables (so that the required marginal variable is labelled x_1) and repeat Bucket Elimination. Hence each query (calculation of a marginal in this case) requires re-running the algorithm. It would be more efficient to reuse messages, rather than recalculating them each time.

2. In general, Bucket Elimination constructs multi-variable messages γ from Bucket to Bucket. The storage requirements of a multi-variable message are exponential in the number of variables of the message.

3. For trees we can always choose a variable ordering to render the computational complexity to be linear in the number of variables. Such an ordering is called perfect, definition(49), and indeed it can be shown that a perfect ordering can always easily be found for singly-connected graphs (see [85]). However, orderings exist for which Bucket Elimination will be extremely inefficient.

5.3.2 Loop-cut conditioning

For distributions which contain a loop (there is more than one path between two nodes in the graph when the directions are removed), we run into some difficulty with the message passing routines such as the sum-product algorithm which are designed to work on singly-connected graphs only. One way to solve the difficulties of multiply connected (loopy) graphs is to identify nodes that, when removed, would reveal a singly-connected subgraph[216]. Consider the example if fig(5.7). Imagine that we wish to calculate a marginal, say $p(d)$. Then

$$p(d) = \sum_c \sum_{a,b,e,f,g} \underbrace{p(c|a)p(a)}_{p^*(a)} p(d|a,b)p(b) \underbrace{p(f|c,d)}_{p^*(f|d)} p(g|d,e) \tag{5.3.11}$$

where the p^* definitions are not necessarily distributions. For each state of c, the form of the products of factors remaining as a function of a, b, e, f, g is singly-connected, so that standard singly-connected message passing can be used to perform inference. We will need to do perform inference for each state of variable c, each state defining a new singly-connected graph (with the same structure) but with modified potentials.

More generally, we can define a set of variables $\mathcal{C}$, called the *loop cut set* and run singly-connected inference for each joint state of the cut-set variables $\mathcal{C}$. This can also be used for finding the most likely state of a multiply-connected joint distribution as well. Hence, for a computational price exponential in the loop-cut size, we can calculate the marginals (or the most likely state) for a multiply-connected distribution. However, determining a small cut set is in general difficult, and there is no guarantee that this will anyway be small for a given graph. Whilst this method is able to handle loops in a general manner, it is not particularly elegant since the concept of messages now only applies conditioned on the cut set variables, and how to re-use messages for inference of additional quantities of interest becomes unclear. We will discuss an alternative method for handling multiply connected distributions in chapter(6).

5.4 Message Passing for Continuous Distributions

For parametric continuous distributions $p(x|\theta_x)$, message passing corresponds to passing parameters θ of the distributions. For the sum-product algorithm, this requires that the operations of multiplication and integration over the variables are closed with respect to the family of distributions. This is the case, for example, for the Gaussian distribution – the marginal (integral) of a Gaussian is another Gaussian, and the product of two Gaussians is a Gaussian, see section(8.6). This means that we can then implement the sum-product algorithm based on passing mean and covariance parameters. To implement this requires some tedious algebra to compute the appropriate message parameter updates. At this stage, the complexities from performing such calculations are a potential distraction, though the interested reader may refer to `demoSumprodGaussMoment.m`, `demoSumprodGaussCanon.m` and `demoSumprodGaussCanonLDS.m` and also chapter(24) for examples of message passing with Gaussians. For more general exponential family distributions, message passing is essentially straightforward, though again the specifics of the updates may be tedious to work out. In cases where the operations of marginalisation and products are not closed within the family, the distributions need to be projected back to the chosen message family. Expectation propagation, section(28.7) is relevant in this case.

5.5 Notes

A take-home message from this chapter is that (non-mixed) inference in singly-connected structures is usually computationally tractable. Notable exceptions are when the message passing operations are not-closed within the message family, or representing messages explicitly requires an exponential amount of space. This happens for example when the distribution can contain both discrete and continuous variables, such as the Switching Linear Dynamical system, which we discuss in chapter(25).

Broadly speaking, inference in multiply-connected structures is more complex and may be intractable. However, we do not want to give the impression that this is always the case. Notable exceptions are: MAP state in an attractive pairwise MRF, section(28.8), MAP and MPM state in a binary planar MRF with pure interactions, see for example [112, 240]. Of interest is *bond propagation*[174] which is an intuitive node elimination method to arrive at the MPM inference in pure-interaction Ising models.

5.6 Code

The code below implements message passing on a tree structured Factor Graph. The FG is stored as an adjacency matrix with the message between FG node i and FG node j given in $A_{i,j}$.
`FactorGraph.m`: Return a Factor Graph adjacency matrix and message numbers
`sumprodFG.m`: Sum-Product algorithm on a Factor Graph

In general it is recommended to work in log-space in the Max-Product case, particularly for large graphs since the produce of messages can become very small. The code provided does not work in log space and as such may not work on large graphs; writing this using log-messages is straightforward but leads to less readable code. An implementation based on log-messages is left as an exercise for the interested reader.
`maxprodFG.m`: Max-Product algorithm on a Factor Graph
`maxNprodFG.m`: N-Max-Product algorithm on a Factor Graph

5.6.1 Factor Graph Examples

For the distribution from fig(5.3), the following code finds the marginals and most likely joint states. The number of states of each variable is chosen at random.

`demoSumprod.m`: Test the Sum-Product algorithm
`demoMaxprod.m`: Test the Max-Product algorithm
`demoMaxNprod.m`: Test the Max-N-Product algorithm

5.6.2 Most probable and shortest path

`mostprobablepath.m`: Most Probable Path
`demoMostProbablePath.m`: Most probable versus shortest path demo

The shortest path demo works for both positive and negative edge weights. If negative weight cycles exist, the code finds the best length N shortest path. `demoShortestPath.m`: Shortest path demo
`mostprobablepathmult.m`: Most Probable Path – multi-source, multi-sink
`demoMostProbablePathMult.m`: Demo of most probable path – multi-source, multi-sink

5.6.3 Bucket Elimination

The efficacy of Bucket Elimination depends critically on the elimination sequence chosen. In the demonstration below we find the marginal of a variable in the Chest Clinic exercise using a randomly chosen elimination order. The desired marginal variable is specified as the last to be eliminated. For comparison we use an elimination sequence based on decimating a triangulated graph of the model, as discussed in section(6.5.1), again under the constraint that the last variable to be 'decimated' is the marginal variable of interest. For this smarter choice of elimination sequence, the complexity of computing this single marginal is roughly the same as that for the Junction Tree algorithm, using the same triangulation.
`bucketelim.m`: Bucket Elimination
`demoBucketElim.m`: Demo Bucket Elimination

5.6.4 Message passing on Gaussians

The following code hints at how message passing may be implemented for continuous distributions. The reader is referred to the BRMLtoolbox for further details and also section(8.6) for the algebraic manipulations required to perform marginalisation and products of Gaussians. The same principal holds for any family of distributions which is closed under products and marginalisation, and the reader may wish to implement specific families following the method outlined for Gaussians.
`demoSumprodGaussMoment.m`: Sum-product message passing based on Gaussian Moment parameterisation

5.7 Exercises

Exercise 53. *Given a pairwise singly connected Markov Network of the form*

$$p(x) = \frac{1}{Z} \prod_{i \sim j} \phi(x_i, x_j) \tag{5.7.1}$$

explain how to efficiently compute the normalisation factor (also called the partition function) Z as a function of the potentials ϕ.

Exercise 54. *You are employed by a web start up company that designs virtual environments, in which players can move between rooms. The rooms which are accessible from another in one time step is given by the 100×100 matrix $\mathbf{M}$, stored in `virtualworlds.mat`, where $M_{ij} = 1$ means that there is a door between rooms i and j ($M_{ij} = M_{ji}$). $M_{ij} = 0$ means that there is no door between rooms i and j. $M_{ii} = 1$ meaning that in one time step, one can stay in the same room. You can visualise this matrix by typing* `imagesc(M)`.

1. *Write a list of rooms which cannot be reached from room 2 after 10 time steps.*

2. *The manager complains that takes at least 13 time steps to get from room 1 to room 100. Is this true?*

3. *Find the most likely path (sequence of rooms) to get from room 1 to room 100.*

4. *If a single player were to jump randomly from one room to another (or stay in the same room), with no preference between rooms, what is the probability at time $t \gg 1$ the player will be in room 1? Assume that effectively an infinite amount of time has passed and the player began in room 1 at $t = 1$.*

5. *If two players are jumping randomly between rooms (or staying in the same room), explain how to compute the probability that, after an infinite amount of time, at least one of them will be in room 1? Assume that both players begin in room 1.*

Exercise 55. *Consider the hidden Markov model:*

$$p(v_1, \ldots, v_T, h_1, \ldots, h_T) = p(h_1)p(v_1|h_1)\prod_{t=2}^{T} p(v_t|h_t)p(h_t|h_{t-1}) \tag{5.7.2}$$

in which $dom(h_t) = \{1, \ldots, H\}$ and $dom(v_t) = \{1, \ldots, V\}$ for all $t = 1, \ldots, T$.

1. *Draw a Belief Network representation of the above distribution.*

2. *Draw a Factor Graph representation of the above distribution.*

3. *Use the Factor Graph to derive a Sum-Product algorithm to compute marginals $p(h_t|v_1, \ldots, v_T)$. Explain the sequence order of messages passed on your Factor Graph.*

4. *Explain how to compute $p(h_t, h_{t+1}|v_1, \ldots, v_T)$.*

Exercise 56. *For a singly connected Markov Network, $p(x) = p(x_1, \ldots, x_n)$, the computation of a marginal $p(x_i)$ can be carried out efficiently. Similarly, the most likely joint state $x^* = \arg\max_{x_1,\ldots,x_n} p(x)$ can be computed efficiently. Explain when the most likely joint state of a marginal can be computed efficiently,* i.e. *under what circumstances could one efficiently (in $O(m)$ time) compute $\underset{x_1,x_2,\ldots,x_m}{\text{argmax}}\ p(x_1, \ldots, x_m)$ for $m < n$?*

Exercise 57. *Consider the internet with webpages labelled $1, \ldots, N$. If webpage j has a link to webpage i, then we place an element of the matrix $L_{ij} = 1$, otherwise $L_{ij} = 0$. By considering a random jump from webpage j to webpage i to be given by the transition probability*

$$M_{ij} = \frac{L_{ij}}{\sum_i L_{ij}} \tag{5.7.3}$$

what is the probability that after an infinite amount of random surfing, one ends up on webpage i? How could you relate this to the potential 'relevance' of a webpage in terms of a search engine?

Exercise 58. *A special time-homogeneous hidden Markov model is given by*

$$p(x_1, \ldots, x_T, y_1, \ldots, y_T, h_1, \ldots, h_T) = p(x_1|h_1)p(y_1|h_1)p(h_1)\prod_{t=2}^{T} p(h_t|h_{t-1})p(x_t|h_t)p(y_t|h_t) \tag{5.7.4}$$

The variable x_t has 4 states, $dom(x_t) = \{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$ (numerically labelled as states 1,2,3,4). The variable y_t has 4 states, $dom(y_t) = \{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$. The hidden or latent variable h_t has 5 states, $dom(h_t) = \{1, \ldots, 5\}$. The HMM models the following (fictitious) process:

In humans, Z-factor proteins are a sequence on states of the variables $x_1, x_2, \ldots, x_T$. In bananas Z-factor proteins are also present, but represented by a different sequence $y_1, y_2, \ldots, y_T$. Given a sequence $x_1, \ldots, x_T$ from a human, the task is to find the corresponding sequence $y_1, \ldots, y_T$ in the banana by first *finding the most likely joint latent sequence, and then the most likely banana sequence given this optimal latent sequence. That is, we require*

$$\underset{y_1,\ldots,y_T}{\operatorname{argmax}}\ p(y_1, \ldots, y_T | h_1^*, \ldots, h_T^*) \tag{5.7.5}$$

where

$$h_1^*, \ldots, h_T^* = \underset{h_1,\ldots,h_T}{\operatorname{argmax}}\ p(h_1, \ldots, h_T | x_1, \ldots, x_T) \tag{5.7.6}$$

The file `banana.mat` *contains the emission distributions* `pxgh` *($p(x|h)$),* `pygh` *($p(y|h)$) and transition* `phtghtm` *($p(h_t|h_{t-1})$). The initial hidden distribution is given in* `ph1` *($p(h_1)$). The observed x sequence is given in* `x`*.*

1. *Explain mathematically and in detail how to compute the optimal y-sequence, using the two-stage procedure as stated above.*
2. *Write a* MATLAB *routine that computes and displays the optimal y-sequence, given the observed x-sequence. Your routine must make use of the Factor Graph formalism.*
3. *Explain whether or not it is computationally tractable to compute*

$$\arg \max_{y_1,\ldots,y_T} p(y_1, \ldots, y_T | x_1, \ldots, x_T) \tag{5.7.7}$$

4. *Bonus question: By considering $y_1, \ldots, y_T$ as parameters, explain how the EM algorithm may be used to find most likely marginal states. Implement this approach with a suitable initialisation for the optimal parameters $y_1, \ldots, y_T$.*

CHAPTER 6

The Junction Tree Algorithm

6.1 Clustering variables

In chapter(5) we discussed efficient inference for singly-connected graphs, for which variable elimination and message passing schemes are appropriate. In the multiply-connected case, however, one cannot in general perform inference by passing messages only along existing links in the graph. The idea behind the Junction Tree algorithm is to form a new representation of the graph in which variables are clustered together, resulting in a singly-connected graph in the cluster variables (albeit on a different graph). The main focus of the development will be on marginal inference, though similar techniques apply to difference inferences, such as finding the maximal state of the distribution.

At this stage it is important to point out that the Junction Tree Algorithm is not a magic method to deal with intractabilities resulting from multiply connected graphs; it is simply a way to perform correct inference on a multiply connected graph by transforming to a singly connected structure. Carrying out the inference on the resulting Junction Tree may still be computationally intractable. For example, the Junction Tree representation of a general two-dimensional Ising model is a trivial Junction Tree with a single supernode containing all the variables. Inference in this case is exponentially complex in the number of variables. Nevertheless, even in cases where implementing the JTA (or any other exact inference algorithm) may be intractable, the JTA provides useful insight into the representation of distributions that can form the basis for approximate inference. In this sense the JTA is key to understanding issues related to representations and complexity of inference and is central to the development of efficient inference algorithms.

6.1.1 Reparameterisation

Consider the chain

$$p(a, b, c, d) = p(a|b)p(b|c)p(c|d)p(d) \tag{6.1.1}$$

Using Bayes' rule, we can reexpress this as

$$p(a, b, c, d) = \frac{p(a, b)}{p(b)} \frac{p(b, c)}{p(c)} \frac{p(c, d)}{p(d)} p(d) = \frac{p(a, b)p(b, c)p(c, d)}{p(b)p(c)} \tag{6.1.2}$$

A useful insight is that the distribution can therefore be written as a product of marginal distributions, divided by a product of the intersection of the marginal distributions: Looking at the numerator $p(a, b)p(b, c)p(c, d)$ this cannot be a distribution over a, b, c, d since we are overcounting b and c, where this overcounting of b arises from the overlap between the sets a, b and b, c, which have b as their intersection. Similarly, the overcounting of c arises from the overlap between the sets b, c and c, d. Roughly speaking we need to correct for this overcounting by dividing by the distribution on the intersections. Given the

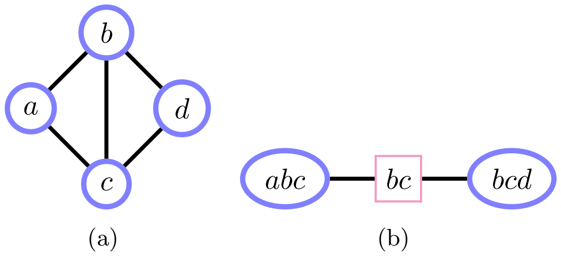


Figure 6.1: (a) Markov Network $\phi(a,b,c)\phi(b,c,d)$. (b) Equivalent Clique Graph of (a).

transformed representation, a marginal such as $p(a,b)$ can be read off directly from the factors in the new expression.

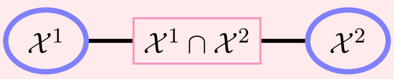
The aim of the Junction Tree algorithm is to form a representation of the distribution which contains the marginals explicitly. We want to do this in a way that works for Belief and Markov Networks, and also deals with the multiply-connected case. In order to do so, an appropriate way to parameterise the distribution is in terms of a clique graph, as described in the next section.

6.2 Clique Graphs

Definition 39 (Clique Graph)**.** A clique graph consists of a set of potentials, $\phi_1(\mathcal{X}^1), \ldots, \phi_n(\mathcal{X}^n)$ each defined on a set of variables $\mathcal{X}^i$. For neighbouring cliques on the graph, defined on sets of variables $\mathcal{X}^i$ and $\mathcal{X}^j$, the intersection $\mathcal{X}^s = \mathcal{X}^i \cap \mathcal{X}^j$ is called the *separator* and has a corresponding potential $\phi_s(\mathcal{X}^s)$. A clique graph represents the function

$$\frac{\prod_c \phi_c(\mathcal{X}^c)}{\prod_s \phi_s(\mathcal{X}^s)} \tag{6.2.1}$$

For notational simplicity we will usually drop the clique potential index c. Graphically clique potentials are represented by circles/ovals, and separator potentials by squares.



The graph on the left represents $\phi(\mathcal{X}^1)\phi(\mathcal{X}^2)/\phi(\mathcal{X}^1 \cap \mathcal{X}^2)$.

Clique graphs translate Markov Networks into structures convenient for carrying out inference. Consider the Markov Network in fig(6.1a)

$$p(a,b,c,d) = \frac{\phi(a,b,c)\phi(b,c,d)}{Z} \tag{6.2.2}$$

which contains two clique potentials sharing the variables b, c. An equivalent representation is given by the clique graph in fig(6.1b), defined as the product of the numerator clique potentials, divided by the product of the separator potentials. In this case the separator potential may be set to the normalisation constant Z. By summing we have

$$Zp(a,b,c) = \phi(a,b,c)\sum_d \phi(b,c,d), \qquad Zp(b,c,d) = \phi(b,c,d)\sum_a \phi(a,b,c) \tag{6.2.3}$$

Multiplying the two expressions, we have

$$Z^2 p(a,b,c)p(b,c,d) = \left(\phi(a,b,c)\sum_d \phi(b,c,d)\right)\left(\phi(b,c,d)\sum_a \phi(a,b,c)\right) = Z^2 p(a,b,c,d)\sum_{a,d} p(a,b,c,d) \tag{6.2.4}$$

In other words

$$p(a, b, c, d) = \frac{p(a, b, c)p(b, c, d)}{p(c, b)} \tag{6.2.5}$$

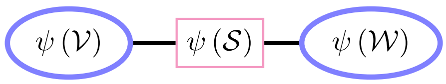
The important observation is that the distribution can be written in terms of its marginals on the variables in the original cliques and that, as a clique graph, it has the same structure as before. All that has changed is that the original clique potentials have been replaced by the marginals of the distribution and the separator by the marginal defined on the separator variables $\phi(a, b, c) \to p(a, b, c)$, $\phi(b, c, d) \to p(b, c, d)$, $Z \to p(c, b)$. The usefulness of this representation is that if we are interested in the marginal $p(a, b, c)$, this can be read off from the transformed clique potential. To make use of this representation, we require a systematic way of transforming the clique graph potentials so that at the end of the transformation the new potentials contain the marginals of the distribution.

Remark 6. Note that, whilst visually similar, a Factor Graph and a Clique Graph are different representations. In a Clique Graph the nodes contain sets of variables, which may share variables with other nodes.

6.2.1 Absorption

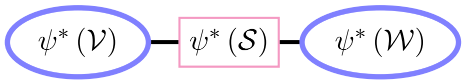
Consider neighbouring cliques $\mathcal{V}$ and $\mathcal{W}$, sharing the variables $\mathcal{S}$ in common. In this case, the distribution on the variables $\mathcal{X} = \mathcal{V} \cup \mathcal{W}$ is

$$p(\mathcal{X}) = \frac{\psi(\mathcal{V})\psi(\mathcal{W})}{\psi(\mathcal{S})} \tag{6.2.6}$$



and our aim is to find a new representation

$$p(\mathcal{X}) = \frac{\psi^*(\mathcal{V})\psi^*(\mathcal{W})}{\psi^*(\mathcal{S})} \tag{6.2.7}$$



in which the potentials are given by

$$\psi^*(\mathcal{V}) = p(\mathcal{V}), \quad \psi^*(\mathcal{W}) = p(\mathcal{W}), \quad \psi^*(\mathcal{S}) = p(\mathcal{S}) \tag{6.2.8}$$

In this example, we can explicitly work out the new potentials as function of the old potentials by computing the marginals as follows:

$$p(\mathcal{W}) = \sum_{\mathcal{V}\backslash\mathcal{S}} p(\mathcal{X}) = \sum_{\mathcal{V}\backslash\mathcal{S}} \frac{\psi(\mathcal{V})\psi(\mathcal{W})}{\psi(\mathcal{S})} = \psi(\mathcal{W})\frac{\sum_{\mathcal{V}\backslash\mathcal{S}}\psi(\mathcal{V})}{\psi(\mathcal{S})} \tag{6.2.9}$$

and

$$p(\mathcal{V}) = \sum_{\mathcal{W}\backslash\mathcal{S}} p(\mathcal{X}) = \sum_{\mathcal{W}\backslash\mathcal{S}} \frac{\psi(\mathcal{V})\psi(\mathcal{W})}{\psi(\mathcal{S})} = \psi(\mathcal{V})\frac{\sum_{\mathcal{W}\backslash\mathcal{S}}\psi(\mathcal{W})}{\psi(\mathcal{S})} \tag{6.2.10}$$

There is a symmetry present in the two equations above – they are the same under interchanging $\mathcal{V}$ and $\mathcal{W}$. One way to describe these equations is through 'absorption'. We say that the cluster $\mathcal{W}$ 'absorbs' information from cluster $\mathcal{V}$ by the following updating procedure. First we define a new separator

$$\psi^*(\mathcal{S}) = \sum_{\mathcal{V}\backslash\mathcal{S}} \psi(\mathcal{V}) \tag{6.2.11}$$

and the refine the $\mathcal{W}$ potential using

$$\psi^*(\mathcal{W}) = \psi(\mathcal{W})\frac{\psi^*(\mathcal{S})}{\psi(\mathcal{S})} \tag{6.2.12}$$

The advantage of this interpretation is that the new representation is still a valid clique graph representation of the distribution since

$$\frac{\psi(\mathcal{V})\psi^*(\mathcal{W})}{\psi^*(\mathcal{S})} = \frac{\psi(\mathcal{V})\psi(\mathcal{W})\frac{\psi^*(\mathcal{S})}{\psi(\mathcal{S})}}{\psi^*(\mathcal{S})} = \frac{\psi(\mathcal{V})\psi(\mathcal{W})}{\psi(\mathcal{S})} = p(\mathcal{X}) \tag{6.2.13}$$

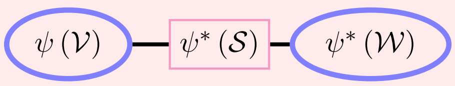
After $\mathcal{W}$ absorbs information from $\mathcal{V}$ then $\psi^*(\mathcal{W})$ contains the marginal $p(\mathcal{W})$. Similarly, after $\mathcal{V}$ absorbs information from $\mathcal{W}$ then $\psi^*(\mathcal{V})$ contains the marginal $p(\mathcal{V})$. After the separator $\mathcal{S}$ has participated in absorption along both directions, then the separator potential will contain $p(\mathcal{S})$ (this is not the case after only a single absorption). To see this, consider

$$\psi^{**}(\mathcal{S}) = \sum_{\mathcal{W}\backslash\mathcal{S}} \psi^*(\mathcal{W}) = \sum_{\mathcal{W}\backslash\mathcal{S}} \frac{\psi(\mathcal{W})\psi^*(\mathcal{S})}{\psi(\mathcal{S})} = \sum_{\{\mathcal{W}\cup\mathcal{V}\}\backslash\mathcal{S}} \frac{\psi(\mathcal{W})\psi(\mathcal{V})}{\psi(\mathcal{S})} = p(\mathcal{S}) \tag{6.2.14}$$

Continuing, we have the new potential $\psi^*(\mathcal{V})$ given by

$$\psi^*(\mathcal{V}) = \frac{\psi(\mathcal{V})\psi^{**}(\mathcal{S})}{\psi^*(\mathcal{S})} = \frac{\psi(\mathcal{V})\sum_{\mathcal{W}\backslash\mathcal{S}}\psi(\mathcal{W})\psi^*(\mathcal{S})/\psi(s)}{\psi^*(\mathcal{S})} = \frac{\sum_{\mathcal{W}\backslash\mathcal{S}}\psi(\mathcal{V})\psi(\mathcal{W})}{\psi(\mathcal{S})} = p(\mathcal{V}) \tag{6.2.15}$$

Definition 40 (Absorption)**.**



Let $\mathcal{V}$ and $\mathcal{W}$ be neighbours in a clique graph, let $\mathcal{S}$ be their separator, and let $\psi(\mathcal{V})$, $\psi(\mathcal{W})$ and $\psi(\mathcal{S})$ be their potentials. Absorption from $\mathcal{V}$ to $\mathcal{W}$ through $\mathcal{S}$ replaces the tables $\psi^*(\mathcal{S})$ and $\psi^*(\mathcal{W})$ with

$$\psi^*(\mathcal{S}) = \sum_{\mathcal{V}\backslash\mathcal{S}} \psi(\mathcal{V}) \qquad \psi^*(\mathcal{W}) = \psi(\mathcal{W})\frac{\psi^*(\mathcal{S})}{\psi(\mathcal{S})} \tag{6.2.16}$$

We say that clique $\mathcal{W}$ absorbs information from clique $\mathcal{V}$.

6.2.2 Absorption schedule on clique trees

Having defined the local message propagation approach, we need to define an update ordering for absorption. In general, a node $\mathcal{V}$ can send exactly one message to a neighbour $\mathcal{W}$, and it may only be sent when $\mathcal{V}$ has received a message from each of its other neighbours. We continue this sequence of absorptions until a message has been passed in both directions along every link. See, for example, fig(6.2). Note that the message passing scheme is not unique.

Definition 41 (Absorption Schedule)**.** A clique can send a message to a neighbour, provided it has already received messages from all other neighbours.

6.3 Junction Trees

There are a few stages we need to go through in order to transform a distribution into an appropriate structure for inference. Initially we explain how to do this for singly-connected structures before moving onto the multiply-connected case.

Consider the singly-connected Markov network, fig(6.3a)

$$p(x_1, x_2, x_3, x_4) = \phi(x_1, x_4)\phi(x_2, x_4)\phi(x_3, x_4) \tag{6.3.1}$$

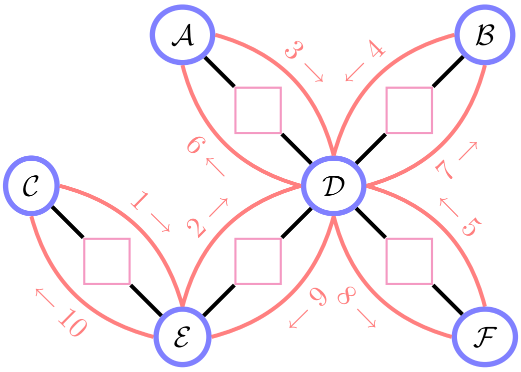


Figure 6.2: An example absorption schedule on a clique tree. Many valid schedules exist, with the only constraint that messages can only be passed to a neighbour when all other messages have been received.

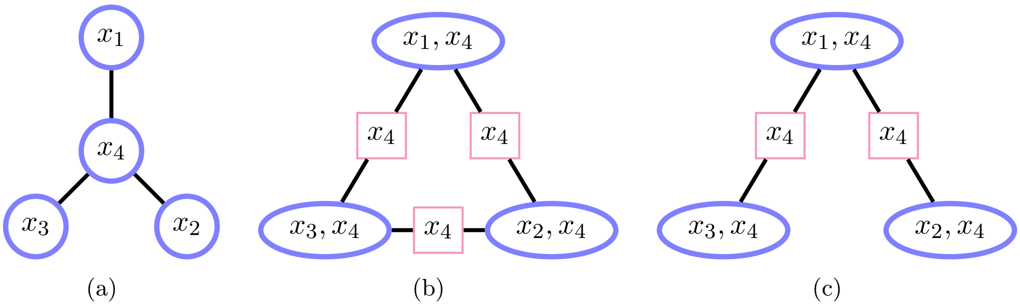


Figure 6.3: **(a)**: Singly-connected Markov Network. **(b)**: Clique graph. **(c)**: Clique tree.

The clique graph of this singly-connected Markov Network is multiply-connected, fig(6.3b), where the separator potentials are all set to unity. Nevertheless, let's try to reexpress the Markov Network in terms of marginals. First we have the relations

$$p(x_1, x_4) = \sum_{x_2,x_3} p(x_1, x_2, x_3, x_4) = \phi(x_1, x_4) \sum_{x_2} \phi(x_2, x_4) \sum_{x_3} \phi(x_3, x_4) \tag{6.3.2}$$

$$p(x_2, x_4) = \sum_{x_1,x_3} p(x_1, x_2, x_3, x_4) = \phi(x_2, x_4) \sum_{x_1} \phi(x_1, x_4) \sum_{x_3} \phi(x_3, x_4) \tag{6.3.3}$$

$$p(x_3, x_4) = \sum_{x_1,x_2} p(x_1, x_2, x_3, x_4) = \phi(x_3, x_4) \sum_{x_1} \phi(x_1, x_4) \sum_{x_2} \phi(x_2, x_4) \tag{6.3.4}$$

Taking the product of the three marginals, we have

$$p(x_1, x_4)p(x_2, x_4)p(x_3, x_4) = \phi(x_1, x_4)\phi(x_2, x_4)\phi(x_3, x_4) \underbrace{\left(\sum_{x_1} \phi(x_1, x_4) \sum_{x_2} \phi(x_2, x_4) \sum_{x_3} \phi(x_3, x_4)\right)^2}_{p(x_4)^2} \tag{6.3.5}$$

This means that the Markov Network can be expressed in terms of marginals as

$$p(x_1, x_2, x_3, x_4) = \frac{p(x_1, x_4)p(x_2, x_4)p(x_3, x_4)}{p(x_4)p(x_4)} \tag{6.3.6}$$

Hence a valid clique graph is also given by the representation fig(6.3c). Indeed, if a variable (here x_4) occurs on every separator in a clique graph loop, one can remove that variable from an arbitrarily chosen separator in the loop. This leaves an empty separator, which we can simply remove. This shows that in such cases we can transform the clique graph into a clique tree (*i.e.* a singly-connected clique graph). Provided that the original Markov Network is singly-connected, one can always form a clique tree in this manner.

6.3.1 The running intersection property

Sticking with the above example, consider the clique tree in fig(6.3)

$$\frac{\phi(x_3, x_4)\phi(x_1, x_4)\phi(x_2, x_4)}{\phi_1(x_4)\phi_2(x_4)} \tag{6.3.7}$$

as a representation of the distribution (6.3.1) where we set $\phi_1(x_4) = \phi_2(x_4) = 1$ to make this match. Now perform absorption on this clique tree:

We absorb $(x_3, x_4) \rightsquigarrow (x_1, x_4)$. The new separator is

$$\phi_1^*(x_4) = \sum_{x_3} \phi(x_3, x_4) \tag{6.3.8}$$

and the new potential is

$$\phi^*(x_1, x_4) = \phi(x_1, x_4)\frac{\phi_1^*(x_4)}{\phi_1(x_4)} = \phi(x_1, x_4)\phi_1^*(x_4) \tag{6.3.9}$$

Now $(x_1, x_4) \rightsquigarrow (x_2, x_4)$. The new separator is

$$\phi_2^*(x_4) = \sum_{x_1} \phi^*(x_1, x_4) \tag{6.3.10}$$

and the new potential is

$$\phi^*(x_2, x_4) = \phi(x_2, x_4)\frac{\phi_2^*(x_4)}{\phi_2(x_4)} = \phi(x_2, x_4)\phi_2^*(x_4) \tag{6.3.11}$$

Since we've 'hit the buffers' in terms of message passing, the potential $\phi(x_2, x_4)$ cannot be updated further. Let's examine more carefully the value of this new potential,

$$\phi^*(x_2, x_4) = \phi(x_2, x_4)\phi_2^*(x_4) = \phi(x_2, x_4)\sum_{x_1} \phi^*(x_1, x_4) \tag{6.3.12}$$

$$= \phi(x_2, x_4)\sum_{x_1} \phi(x_1, x_4)\sum_{x_3} \phi(x_3, x_4) = \sum_{x_1,x_3} p(x_1, x_2, x_3, x_4) = p(x_2, x_4) \tag{6.3.13}$$

Hence the new potential $\phi^*(x_2, x_4)$ contains the marginal $p(x_2, x_4)$.

To complete a full round of message passing we need to have passed messages in a valid schedule along both directions of each separator. To do so, we continue as follows:

We absorb $(x_2, x_4) \rightsquigarrow (x_1, x_4)$. The new separator is

$$\phi_2^{**}(x_4) = \sum_{x_2} \phi^*(x_2, x_4) \tag{6.3.14}$$

and

$$\phi^{**}(x_1, x_4) = \phi^*(x_1, x_4)\frac{\phi_2^{**}(x_4)}{\phi_2^*(x_4)} \tag{6.3.15}$$

Note that $\phi_2^{**}(x_4) = \sum_{x_2} \phi^*(x_2, x_4) = \sum_{x_2} p(x_2, x_4) = p(x_4)$ so that now, after absorbing through both directions, the separator contains the marginal $p(x_4)$. The reader may show that $\phi^{**}(x_1, x_4) = p(x_1, x_4)$.

Finally, we absorb $(x_1, x_4) \rightsquigarrow (x_3, x_4)$. The new separator is

$$\phi_1^{**}(x_4) = \sum_{x_1} \phi^{**}(x_1, x_4) = p(x_4) \tag{6.3.16}$$

and

$$\phi^*(x_3, x_4) = \phi(x_3, x_4)\frac{\phi_1^{**}(x_4)}{\phi_1^*(x_4)} = p(x_3, x_4) \tag{6.3.17}$$

Hence, after a full round of message passing, the new potentials all contain the correct marginals.

The new representation is *consistent* in the sense that for any (not necessarily neighbouring) cliques $\mathcal{V}$ and $\mathcal{W}$ with intersection $\mathcal{I}$, and corresponding potentials $\psi(\mathcal{V})$ and $\psi(\mathcal{W})$,

$$\sum_{\mathcal{V}\backslash\mathcal{I}} \psi(\mathcal{V}) = \sum_{\mathcal{W}\backslash\mathcal{I}} \psi(\mathcal{W}) \tag{6.3.18}$$

Note that bidirectional absorption guarantees consistency for neighbouring cliques, as in the example above, provided that we started with a clique tree which is a correct representation of the distribution.

In general, the only possible source of non-consistency is if a variable occurs in two non-neighbouring cliques and is not present in all cliques on any path connection them. An extreme example would be if we removed the link between cliques (x_3, x_4) and (x_1, x_4). In this case this is still a Clique Tree; however global consistency could not be guaranteed since the information required to make clique (x_3, x_4) consistent with the rest of the graph cannot reach this clique.

Formally, the requirement for the propagation of local to global consistency is that the clique tree is a junction tree, as defined below.

Definition 42 (Junction Tree)**.** A Clique Tree is a Junction Tree if, for each pair of nodes, $\mathcal{V}$ and $\mathcal{W}$, all nodes on the path between $\mathcal{V}$ and $\mathcal{W}$ contain the intersection $\mathcal{V} \cap \mathcal{W}$. This is also called the *running intersection property*.

From this definition local consistency will be passed on to any neighbours and the distribution will be globally consistent. Proofs for these results are contained in [146].

Example 22 (A consistent Junction Tree)**.** To gain some intuition about the meaning of consistency, consider the Junction Tree in fig(6.4d). After a full round of message passing on this tree, each link is consistent, and the product of the potentials divided by the product of the separator potentials is just the original distribution itself. Imagine that we are interested in calculating the marginal for the node abc. That requires summing over all the other variables, $defgh$. If we consider summing over h then, because the link is consistent,

$$\sum_h \psi^*(e, h) = \psi^*(e) \tag{6.3.19}$$

so that the ratio $\sum_h \frac{\psi^*(e,h)}{\psi^*(e)}$ is unity, and the effect of summing over node h is that the link between eh and dce can be removed, along with the separator. The same happens for the link between node eg and dce, and also for cf to abc. The only nodes remaining are now dce and abc and their separator c, which have so far been unaffected by the summations. We still need to sum out over d and e. Again, because the link is consistent,

$$\sum_{de} \psi^*(d, c, e) = \psi^*(c) \tag{6.3.20}$$

so that the ratio $\sum_{de} \frac{\psi^*(d,c,e)}{\psi^*(c)} = 1$. The result of the summation of all variables not in abc therefore produces unity for the cliques and their separators, and the summed potential representation reduces simply to the potential $\psi^*(a, b, c)$ which is the marginal $p(a, b, c)$. It is clear that a similar effect will happen for other nodes. We can then obtain the marginals for individual variables by simple brute force summation over the other variables in that potential, for example $p(f) = \sum_c \psi^*(c, f)$.

6.4 Constructing a Junction Tree for singly-connected distributions

6.4.1 Moralisation

For Belief Networks, an initial step is required, which is not required in the case of undirected graphs.

Definition 43 (Moralisation)**.** For each variable x add an undirected link between all parents of x and replace the directed link from x to its parents by undirected links. This creates a 'moralised' Markov Network.

6.4.2 Forming the clique graph

The clique graph is formed by identifying the cliques in the Markov Network and adding a link between cliques that have a non-empty intersection. Add a separator between the intersecting cliques.

6.4.3 Forming a junction tree from a clique graph

For a singly-connected distribution, any maximal weight spanning tree of a clique graph is a junction tree.

Definition 44 (Junction Tree)**.** A junction tree is obtained by finding a maximal weight spanning tree of the clique graph. The weight of the tree is defined as the sum of all the separator weights of the tree, where the separator weight is the number of variables in the separator.

If the clique graph contains loops, then all separators on the loop contain the same variable. By continuing to remove loop links until you have a tree is revealed, we obtain a Junction Tree.

Example 23 (Forming a Junction Tree)**.** Consider the Belief Network in fig(6.4a). The moralisation procedure gives fig(6.4b). Identifying the cliques in this graph, and linking them together gives the clique graph in fig(6.4c). There are several possible Junction Trees one could obtain from this clique graph, and one is given in fig(6.4d).

6.4.4 Assigning Potentials to Cliques

Definition 45 (Clique Potential Assignment)**.** Given a Junction Tree and a set of potentials $\psi\left(\mathcal{X}^1\right), \ldots, \psi\left(\mathcal{X}^n\right)$, a valid clique potential assignment places potentials in cliques whose variables can contain them such that the product of the clique potentials, divided by the separator potentials, is equal to the function.

Example 24. For the belief network of fig(6.4a), we wish to assign its potentials to the junction tree fig(6.4d). In this case the assignment is unique and is given by

$$\begin{aligned}
\psi\left(abc\right) &= p(a)p(b)p(c|a,b) \\
\psi\left(dce\right) &= p(d)p(e|d,c) \\
\psi\left(cf\right) &= p(f|c) \\
\psi\left(eg\right) &= p(g|e) \\
\psi\left(eh\right) &= p(h|e)
\end{aligned} \tag{6.4.1}$$

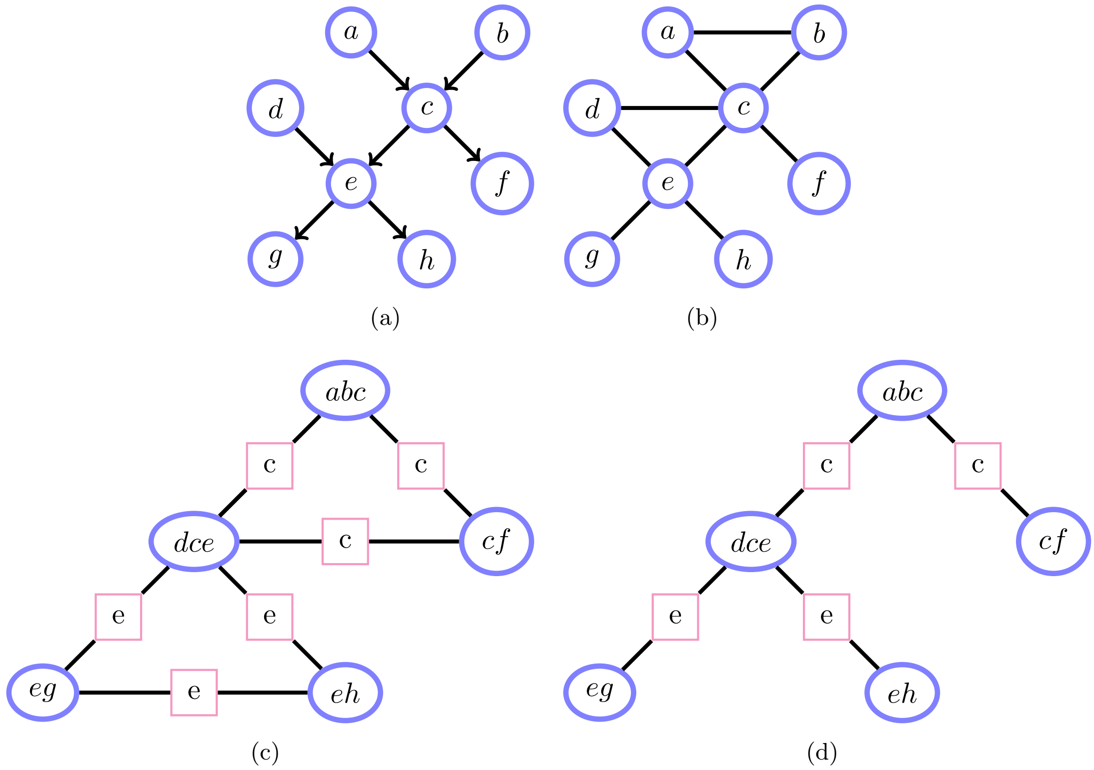


Figure 6.4: **(a)**: Belief Network. **(b)**: Moralised version of (a). **(c)**: Clique Graph of (b). **(d)**: A Junction Tree. This satisfies the running intersection property that for any two nodes which contain a variable in common, any clique on the path linking the two nodes also contains that variable.

All separator potentials are initialised to unity. Note that in some instances it can be that a junction tree clique is assigned to unity.

6.5 Junction Trees for Multiply-Connected distributions

When the distribution contains loops, the construction outlined in section(6.4) does not result in a Junction Tree. The reason is that, due to the loops, variable elimination changes the structure of the remaining graph. To see this, consider the following distribution,

$$p(a, b, c, d) = \phi(a, b)\phi(b, c)\phi(c, d)\phi(d, a) \tag{6.5.1}$$

as shown in fig(6.5a). Let's first try to make a clique graph. We have a choice about which variable first to marginalise over. Let's choose d:

$$p(a, b, c) = \phi(a, b)\phi(b, c)\sum_d \phi(c, d)\phi(d, a) \tag{6.5.2}$$

The remaining subgraph therefore has an extra connection between a and c, see fig(6.5b). We can express the joint in terms of the marginals using

$$p(a, b, c, d) = \frac{p(a, b, c)}{\sum_d \phi(c, d)\phi(d, a)}\phi(c, d)\phi(d, a) \tag{6.5.3}$$

To continue the transformation into marginal form, let's try to replace the numerator terms with probabilities. We can do this by considering

$$p(a, c, d) = \phi(c, d)\phi(d, a)\sum_b \phi(a, b)\phi(b, c) \tag{6.5.4}$$

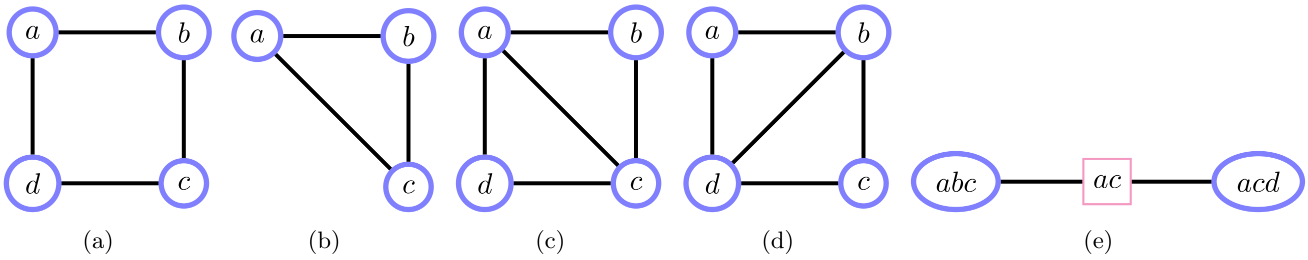


Figure 6.5: **(a)**: An undirected graph with a loop. **(b)**: Eliminating node d adds a link between a and c in the subgraph. **(c)**: The induced representation for the graph in (a). **(d)**: Equivalent induced representation. **(e)**: Junction Tree for (a).

Plugging this into the above equation, we have

$$p(a,b,c,d) = \frac{p(a,b,c)p(a,c,d)}{\sum_d \phi(c,d)\phi(d,a) \sum_b \phi(a,b)\phi(b,c)} \tag{6.5.5}$$

We recognise that the denominator is simply $p(a,c)$, hence

$$p(a,b,c,d) = \frac{p(a,b,c)p(a,c,d)}{p(a,c)}. \tag{6.5.6}$$

This means that a valid clique graph for the distribution fig(6.5a) must contain cliques larger than those in the original distribution. To form a JT based on products of cliques divided by products of separators, we could start from the *induced representation* fig(6.5c). Alternatively, we could have marginalised over variables a and c, and ended up with the equivalent representation fig(6.5d).

Generally, the result from variable elimination and re-representation in terms of the induced graph is that a link is added between any two variables on a loop (of length 4 or more) which does not have a chord. This is called *triangulation*. A Markov Network on a triangulated graph can always be written in terms of the product of marginals divided by the product of separators. Armed with this new induced representation, we can form a junction tree.

Example 25. A slightly more complex loopy distribution is depicted in fig(6.6a),

$$p(a,b,c,d,e,f) = \phi(a,b)\phi(b,c)\phi(c,d)\phi(d,e)\phi(e,f)\phi(a,f)\phi(b,e) \tag{6.5.7}$$

There are different induced representations depending on which variables we decide to eliminate. The reader may convince herself that one such induced representation is given by fig(6.6b).

Definition 46 (Chord)**.** This is a link joining two non-consecutive vertices of a loop.

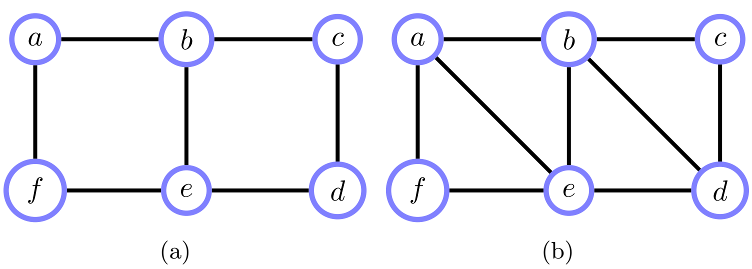


Figure 6.6: **(a)**: Loopy 'Ladder' Markov Network. **(b)**: Induced representation.

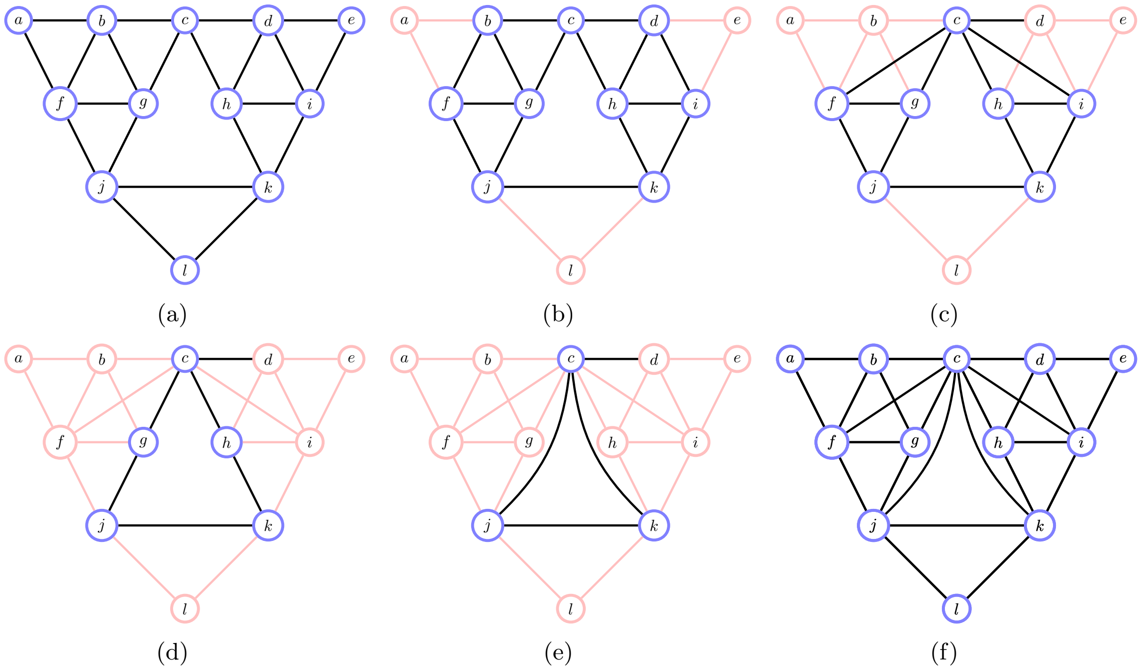


Figure 6.7: **(a)**: Markov Network for which we seek a triangulation via greedy variable elimination. We first eliminate the simplical nodes a, e, l. **(b)**: We then eliminate variables b, d since these only add a single extra link to the induced graph. **(c)**: f and i are now simplical and are eliminated. **(d)**: We eliminate g and h since this adds only single extra links. **(e)**: The remaining variables $\{c, j, k\}$ may be eliminated in any order. **(f)**: Final triangulation. The variable elimination (partial) order is $\{a, e, l\}, \{b, d\}, \{f, i\}, \{g, h\}, \{c, j, k\}$ where the brackets indicate that the order in which the variables inside the bracket are eliminated is irrelevant. Compared with the triangulation produced by the max-cardinality checking approach in fig(6.9d), this triangulation is more parsimonious.

Definition 47 (Triangulated (Decomposable) Graph)**.** An undirected graph is triangulated if every loop of length 4 or more has a chord. An equivalent term is that the graph is *decomposable* or *chordal*. An undirected graph is triangulated if and only if its clique graph has a junction tree.

Page 114, on the greedy elimination example "(d): f and i are now simplical and are eliminated." is false, which don't change the final result but the next step don't add links because they have already been add.((e): We eliminate g and h since this adds only single extra links." is false also).

6.5.1 Triangulation Algorithms

When a variable is eliminated from a graph, links are added between all the neighbours of the eliminated variable. A triangulation algorithm is one that produces a graph for which there exists a variable elimination order that introduces no extra links in the graph.

For discrete variables the complexity of inference scales exponentially with clique sizes in the triangulated graph since absorption requires computing tables on the cliques. It is therefore of some interest to find a triangulated graph with small clique sizes. However, finding the triangulated graph with the smallest maximal clique is an NP-hard problem for a general graph, and heuristics are unavoidable. Below we describe two simple algorithms that are generically reasonable, although there may be cases where an alternative algorithm may be considerably more efficient[52, 28, 188].

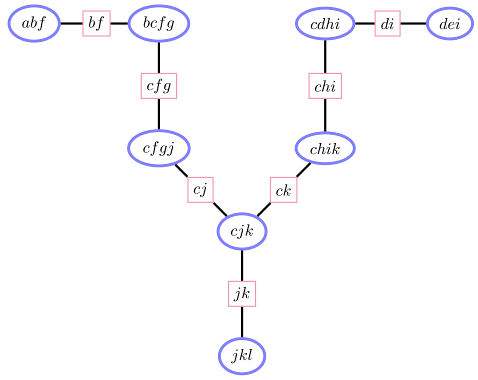


Figure 6.8: Junction tree formed from the triangulation fig(6.7)f. One verify that this satisfies the running intersection property.

Remark 7 (Triangulation does not mean putting 'triangles' on the original graph)**.** Note that a triangulated graph is not one in which 'squares in the original graph have triangles within them in the triangulated graph'. Whilst this is the case for fig(6.6b), this is not true for fig(6.9d). The term triangulation refers to the fact that *every* 'square' (*i.e.* loop of length 4) must have a 'triangle', with edges added until this criterion is satisfied.

Greedy variable elimination

An intuitive way to think of triangulation is to first start with *simplical nodes*, namely those which, when eliminated do not introduce any extra links in the remaining graph. Next consider a non-simplical node of the remaining graph that has the minimal number of neighbours. Then add a link between all neighbours of this node and finally eliminate this node from the graph. Continue until all nodes have been eliminated. (This procedure corresponds to Rose-Tarjan Elimination[229] with a particular node elimination choice). By labelling the nodes eliminated in sequence, we obtain a perfect ordering (see below) in reverse. In the case that (discrete) variables have different numbers of states, a more refined version is to choose the non-simplical node i which, when eliminated, leaves the smallest clique table size (the product of the size of all the state dimensions of the neighbours of node i). See fig(6.7) for an example.

Definition 48 (Variable Elimination)**.** In Variable Elimination, one simply picks any non-deleted node x in the graph, and then adds links to all the neighbours of x. Node x is then deleted. One repeats this until all nodes have been deleted[229].

Whilst this procedure guarantees a triangulated graph, its efficiency depends heavily on the sequence of nodes chosen to be eliminated. Several heuristics for this have been proposed, including the one below, which corresponds to choosing x to be the node with the minimal number of neighbours.

Maximum Cardinality Checking

Algorithm(2) terminates with success if the graph is triangulated. Not only is this a sufficient condition for a graph to be triangulated, but is also necessary [268]. It processes each node and the time to process a node is quadratic in the number of adjacent nodes. This triangulation checking algorithm also suggests a triangulation construction algorithm – we simply add a link between the two neighbours that caused the algorithm to FAIL, and then restart the algorithm. The algorithm is restarted from the beginning, not just continued from the current node. This is important since the new link may change the connectivity between previously labelled nodes. See fig(6.9) for an example[1].

[1] This example is due to David Page `www.cs.wisc.edu/~dpage/cs731`

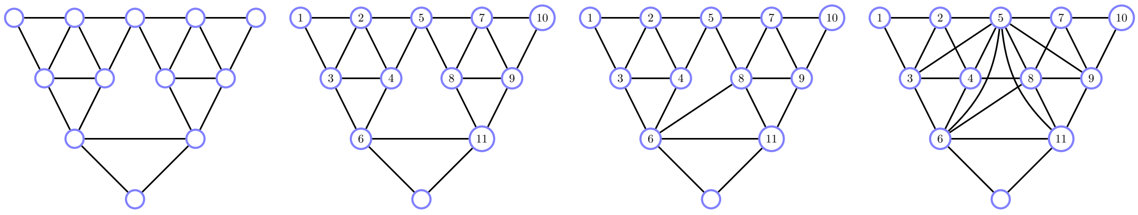


Figure 6.9: Starting with the Markov Network in (a), the maximum cardinality check algorithm proceeds until (b). where an additional link is required (c). One continues the algorithm until the fully triangulated graph (d) is found.

Definition 49 (Perfect Elimination Order)**.** Let the n variables in a Markov Network be ordered from 1 to n. The ordering is perfect if, for each node i, the neighbours of i that are later in the ordering, and i itself, form a (maximal) clique. This means that when we eliminate the variables in sequence from 1 to n, no additional links are induced in the remaining marginal graph. A graph which admits a perfect elimination order is decomposable, and vice versa.

Algorithm 2 A check if a graph is decomposable (triangulated). The graph is triangulated if, after cycling through all the n nodes in the graph, the FAIL criterion is not encountered.

```
1: Choose any node in the graph and label it 1.
2: for i = 2 to n do
3:     Choose the node with the most labeled neighbours and label it i.
4:     If any two labeled neighbours of i are not adjacent to each other, FAIL.
5: end for
```

Where there is more than one node with the most labeled neighbours, the tie may be broken arbitrarily.

6.5.2 Motivating the JT

Since the cliques of a JT

TODO

6.6 The Junction Tree Algorithm

We now have all the steps required for inference in multiply-connected graphs:

Moralisation This is required only for directed distributions.

Triangulation Ensure that every loop of length 4 or more has a chord.

Junction Tree Form a Junction Tree from cliques of the triangulated graph, removing any unnecessary links in a loop on the cluster graph. Algorithmically, this can be achieved by finding a tree with maximal spanning weight with weight w_{ij} given by the number of variables in the separator between cliques i and j. Alternatively, given a clique elimination order (with the lowest cliques eliminated first), one may connect each clique i to the single neighbouring clique $j > i$ with greatest edge weight w_{ij}.

Potential Assignment Assign potentials to Junction Tree cliques and set the separator potentials to unity.

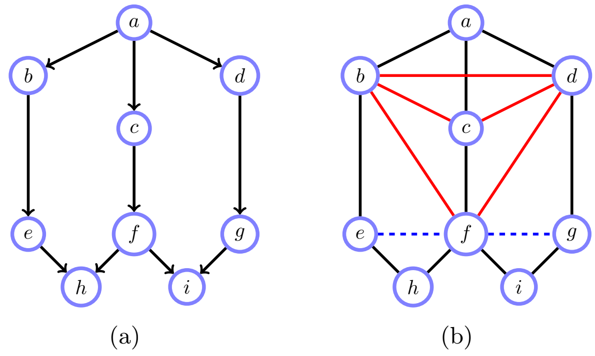


Figure 6.10: **(a)**: Original loopy Belief Network. **(b)**: The moralisation links (dashed) are between nodes e and f and between nodes f and g. The other additional links come from triangulation. The clique size of the resulting clique tree (not shown) is four. From [54].

Message Propagation Carry out absorption until updates have been passed along both directions of every link on the JT.

The clique marginals can then be read off from the JT. An example is given in fig(6.10).

6.6.1 Remarks on the JTA

- The algorithm provides an upper bound on the computation required to calculate marginals in the graph. There may exist more efficient algorithms in particular cases, although generally it is believed that there cannot be much more efficient approaches than the JTA since every other approach must perform a triangulation[145, 170]. One particular special case is that of marginal inference for a binary variable MRF on a two-dimensional lattice containing only pure quadratic interactions. In this case the complexity of computing a marginal inference is $O\left(n^3\right)$ where n is the number of variables in the distribution. This is in contrast to the pessimistic exponential complexity suggested by the JTA. Such cases are highly specialised and it is unlikely that a general purpose algorithm that could consistently outperform the JTA exists.

- One might think that the only class of distributions for which essentially a linear time algorithm is available are singly-connected distributions. However, there are decomposable graphs for which the cliques have limited size meaning that inference is tractable. For example an extended version of the 'ladder' in fig(6.6a) has a simple induced decomposable representation fig(6.6b), for which marginal inference would be linear in the number of rungs in the ladder. Effectively these structures are *hyper trees* in which the complexity is then related to the *tree width* of the graph[80].

- Ideally, we would like to find a triangulated graph which has minimal clique size. However, it can be shown to be a hard-computation problem (NP-hard) to find the most efficient triangulation. In practice, most general purpose triangulation algorithms are somewhat heuristic and chosen to provide reasonable, but clearly not optimal, generic performance.

- Numerical over/under flow issues can occur in large cliques, where many probability values are multiplied together. Similarly in long chains since absorption will tend to reduce the numerical size of potential entries in a clique. If we only care about marginals we can avoid numerical difficulties by normalising potentials at each step; these missing normalisation constants can always be found under the normalisation constraint. If required one can always store the values of these local renormalisations, should, for example, the global normalisation constant of a distribution be required, see section(6.6.2).

- After clamping variables in evidential states, running the JTA returns the joint distribution on the non-evidential variables in a clique with all the evidential variables clamped in their evidential states. From this conditionals are straightforward to calculate.

- Imagine that we have run the JT algorithm and want to afterwards find the marginal $p(\mathcal{X}|\mathsf{evidence})$. We could do so by clamping the evidential variables. However, if both $\mathcal{X}$ and the set of evidential variables are all contained within a single clique of the JT, then we may use the consistent JT cliques to compute $p(\mathcal{X}|\mathsf{evidence})$. The reason is that since the JT clique contains the marginal on the set of variables which includes $\mathcal{X}$ and the evidential variables, we can obtain the required marginal by considering the single JT clique alone.

- Representing the marginal distribution of a set of variables $\mathcal{X}$ which are not contained within a single clique is in general computationally difficult. Whilst the probability of any state of $p(\mathcal{X})$ may be computed efficiently, there are in general an exponential number of such states. A classical example in this regard is the HMM, section(23.2) with singly-connected joint distribution $p(\mathcal{V}, \mathcal{H})$. However the marginal distribution $p(\mathcal{H})$ is fully connected. This means that for example whilst the entropy of $p(\mathcal{V}, \mathcal{H})$ is straightforward to compute, the entropy of the marginal $p(\mathcal{H})$ is intractable.

6.6.2 Computing the normalisation constant of a distribution

For a Markov Network

$$p(\mathcal{X}) = \frac{1}{Z}\prod_i \phi(\mathcal{X}_i) \tag{6.6.1}$$

how can we find Z efficiently? If we used the JTA on the unnormalised distribution $\prod_i \phi(\mathcal{X}_i)$, we would have the equivalent representation:

$$p(\mathcal{X}) = \frac{1}{Z}\frac{\prod_C \phi(\mathcal{X}_C)}{\prod_S \phi(\mathcal{X}_S)} \tag{6.6.2}$$

Since the distribution must normalise, we can obtain Z from

$$Z = \sum_x \frac{\prod_C \phi(\mathcal{X}_C)}{\prod_S \phi(\mathcal{X}_S)} \tag{6.6.3}$$

For a consistent JT, summing first over the variables of a simplical JT clique (not including the separator variables), the marginal clique will cancel with the corresponding separator to give a unity term so that the clique and separator can be removed. This forms a new JT for which we then eliminate another simplical clique. Continuing in this manner we will be left with a single numerator potential so that

$$Z = \sum_{\mathcal{X}_C} \phi(\mathcal{X}_C) \tag{6.6.4}$$

This is true for any clique C, so it makes sense to choose one with a small number of states so that the resulting raw summation is efficient. Hence in order to compute the normalisation constant of a distribution one runs the JT algorithm on an unnormalised distribution and the global normalisation is then given by the local normalisation of any clique. Note that if the graph is disconnected (there are isolated cliques), the normalisation is the product of the connected component normalisation constants. A computationally convenient way to find this is to compute the product of all clique normalisations divided by the product of all separator normalisations.

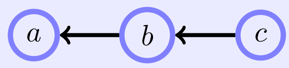
6.6.3 The marginal likelihood

Our interest here is the computation of $p(\mathcal{V})$ where $\mathcal{V}$ is a subset of the full variable set. Naively, one could carry out this computation by summing over all the non-evidential variables (hidden variables $\mathcal{H} = \mathcal{X}\backslash\mathcal{V}$) explicitly. In cases where this is computationally impractical an alternative is to use

$$p(\mathcal{H}|\mathcal{V}) = \frac{p(\mathcal{V}, \mathcal{H})}{p(\mathcal{V})} \tag{6.6.5}$$

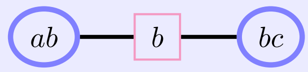
One can view this as a product of clique potentials divided by the normalisation $p(\mathcal{V})$, for which the general method of section(6.6.2) may be directly applied. See `demoJTree.m`.

Example 26 (A simple example of the JTA)**.**
Consider running the JTA on the simple graph



$$p(a, b, c) = p(a|b)p(b|c)p(c) \tag{6.6.6}$$

The moralisation and triangulation steps are trivial, and the JTA is given immediately by the figure on the right. A valid assignment is



$$\psi\,(a, b) = p(a|b),\ \psi\,(b) = 1,\ \psi\,(b, c) = p(b|c)p(c) \tag{6.6.7}$$

To find a marginal $p(b)$ we first run the JTA:

- Absorbing from ab through b, the new separator is $\psi^*\,(b) = \sum_a \psi\,(a, b) = \sum_a p(a|b) = 1$.
- The new potential on (b, c) is given by

$$\psi^*\,(b, c) = \frac{\psi\,(b, c)\psi^*\,(b)}{\psi\,(b)} = \frac{p(b|c)p(c) \times 1}{1} \tag{6.6.8}$$

- Absorbing from bc through b, the new separator is

$$\psi^{**}\,(b) = \sum_c \psi^*\,(b, c) = \sum_c p(b|c)p(c) \tag{6.6.9}$$

- The new potential on (a, b) is given by

$$\psi^*\,(a, b) = \frac{\psi\,(a, b)\psi^{**}\,(b)}{\psi^*\,(b)} = \frac{p(a|b)\sum_c p(b|c)p(c)}{1} \tag{6.6.10}$$

  This is therefore indeed equal to the marginal since $\sum_c p(a, b, c) = p(a, b)$.

The new separator $\psi^{**}\,(b)$ contains the marginal $p(b)$ since

$$\psi^{**}\,(b) = \sum_c p(b|c)p(c) = \sum_c p(b, c) = p(b) \tag{6.6.11}$$

Example 27 (Finding a conditional marginal)**.** Continuing with the distribution in example(26), we consider how to compute $p(b|a = 1, c = 1)$. First we clamp the evidential variables in their states. Then we claim that the effect of running the JTA is to produce on a set of clique variables $\mathcal{X}$ the marginals on the cliques $p(\mathcal{X}, \mathcal{V})$. We demonstrate this below:

- In general, the new separator is given by $\psi^*\,(b) = \sum_a \psi\,(a, b) = \sum_a p(a|b) = 1$. However, since a is clamped in state $a = 1$, then the summation is not carried out over a, and we have instead $\psi^*\,(b) = p(a = 1|b)$.
- The new potential on the (b, c) clique is given by

$$\psi^*\,(b, c) = \frac{\psi\,(b, c)\psi^*\,(b)}{\psi\,(b)} = \frac{p(b|c = 1)p(c = 1)p(a = 1|b)}{1} \tag{6.6.12}$$

- The new separator is normally given by

$$\psi^{**}\,(b) = \sum_c \psi^*\,(b, c) = \sum_c p(b|c)p(c) \tag{6.6.13}$$

  However, since c is clamped in state 1, we have instead

$$\psi^{**}\,(b) = p(b|c = 1)p(c = 1)p(a = 1|b) \tag{6.6.14}$$

- The new potential on (a, b) is given by

$$\psi^* (a, b) = \frac{\psi (a, b)\psi^{**} (b)}{\psi^* (b)} = \frac{p(a = 1|b)p(b|c = 1)p(c = 1)p(a = 1|b)}{p(a = 1|b)} = p(a = 1|b)p(b|c = 1)p(c = 1) \tag{6.6.15}$$

The effect of clamping a set of variables $\mathcal{V}$ in their evidential states and running the JTA is that, for a clique i which contains the set of non-evidential variables $\mathcal{H}^i$, the consistent potential from the JTA contains the marginal $p(\mathcal{H}^i, \mathcal{V})$. Finding a conditional marginal is then straightforward by ensuring normalisation.

Example 28 (finding the likelihood $p(a = 1, c = 1)$)**.** The effect of clamping the variables in their evidential states and running the JTA produces the joint marginals, such as $\psi^* (a, b) = p(a = 1, b, c = 1)$. Then calculating the likelihood is easy since we just sum out over the non-evidential variables of any converged potential : $p(a = 1, c = 1) = \sum_b \psi^* (a, b) = \sum_b p(a = 1, b, c = 1)$.

6.7 Finding the Most Likely State

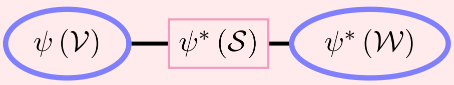
A quantity of interest is the most likely joint state of a distribution:

$$\underset{x_1,\dots,x_n}{\operatorname{argmax}}\ p(\mathcal{X}) \tag{6.7.1}$$

and it is natural to wonder how this can be efficiently computed in the case of a loopy distribution. Since the development of the JTA is based around a variable elimination procedure and the max operator distributes over the distribution as well, eliminating a variable by maximising over that variable will have the same effect on the graph structure as summation did. This means that a Junction Tree is again an appropriate structure on which to perform max operations. Once a JT has been constructed, one then uses the Max Absorption procedure (see below), to perform maximisation over the variables. After a full round of absorption has been carried out, the cliques contain the distribution on the variables of the clique with all remaining variables set to their optimal states. The optimal local states can be found by explicit optimisation of each clique potential separately.

Note that this procedure holds also for non-distributions – in this sense this is an example of a more general dynamic programming procedure applied in a case where the underlying graph is multiply-connected. This demonstrates how to efficiently compute the optimum of a multiply-connected function defined as the product on potentials.

Definition 50 (Max Absorption)**.**



Let $\mathcal{V}$ and $\mathcal{W}$ be neighbours in a clique graph, let $\mathcal{S}$ be their separator, and let $\psi (\mathcal{V})$, $\psi (\mathcal{W})$ and $\psi (\mathcal{S})$ be their potentials. Absorption replaces the tables $\psi (\mathcal{S})$ and $\psi (\mathcal{W})$ with

$$\psi^* (\mathcal{S}) = \max_{\mathcal{V}\backslash\mathcal{S}} \psi (\mathcal{V}) \qquad \psi^* (\mathcal{W}) = \psi (\mathcal{W})\frac{\psi^* (\mathcal{S})}{\psi (\mathcal{S})}$$

Once messages have been passed in both directions over all separators, according to a valid schedule, the most-likely joint state can be read off from maximising the state of the clique potentials. This is implemented in `absorb.m` and `absorption.m` where a flag is used to switch between either sum or max absorption.

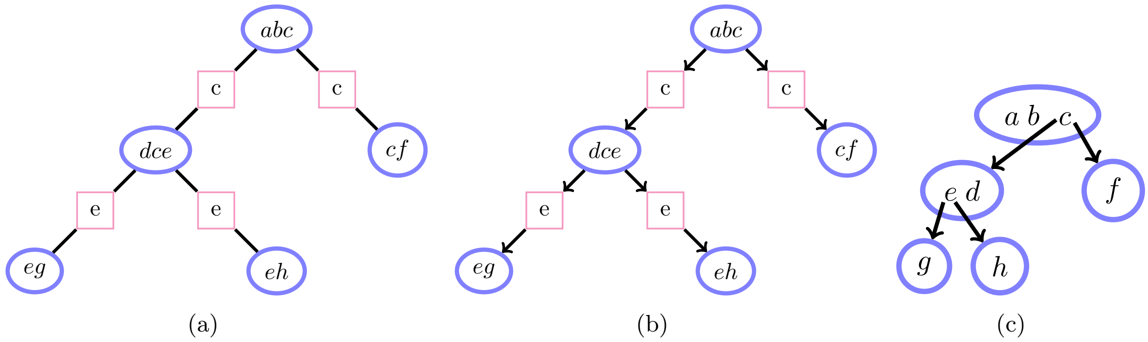


Figure 6.11: **(a)**: Junction tree. **(b)**: Directed junction tree in which all edges are consistently oriented away from the clique (abc). **(c)**: A set chain formed from the junction tree by reabsorbing each separator into its child clique.

6.8 Reabsorption : Converting a Junction Tree to a Directed Network

It is sometimes useful to be able to convert the JT back to a BN of a desired form. For example, if one wishes to draw samples from a Markov network, this can be achieved by ancestral sampling on an equivalent directed structure, see section(27.2.2).

Revisiting the example from fig(6.4), we have the JT given in fig(6.11a). To find a valid directed representation we first orient the JT edges consistently away from a chosen root node (see `singleparenttree.m`), thereby forming a directed JT which has the property that each clique has at most one parent clique.

Definition 51 (Reabsorption)**.**

Let $\mathcal{V}$ and $\mathcal{W}$ be neighbouring cliques in a directed JT in which each clique in the tree has at most one parent. Furthermore, let $\mathcal{S}$ be their separator, and $\psi(\mathcal{V})$, $\psi(\mathcal{W})$ and $\psi(\mathcal{S})$ be the potentials. Reabsorption into $\mathcal{W}$ removes the separator and forms a (set) conditional distribution

$$p(\mathcal{W}|\mathcal{V}) = \frac{\psi(\mathcal{W})}{\psi(\mathcal{S})} \tag{6.8.1}$$

We say that clique $\mathcal{W}$ reabsorbs the separator $\mathcal{S}$.

In fig(6.11) where one amongst many possible directed representations is formed from the JT. Specifically, fig(6.11a) represents

$$p(a, b, c, d, e, f, g, h) = \frac{p(e, g)p(d, c, e)p(a, b, c)p(c, f)p(e, h)}{p(e)p(c)p(c)p(e)} \tag{6.8.2}$$

We now have many choices as to which clique re-absorbs a separator. One such choice would give

$$p(a, b, c, d, e, f, g, h) = p(g|e)p(d, e|c)p(a, b, c,)p(f|c)p(h|e) \tag{6.8.3}$$

This can be represented using a so-called *set chain*[167] in fig(6.11c) (set chains generalise Belief Networks to a product of clusters of variables conditioned on parents). By writing each of the set conditional probabilities as local conditional BNs, one may also write full BN. For example, one such would be given from the decomposition

$$p(c|a, b)p(b|a)p(a)p(g|e)p(f|c)p(h|e)p(d|e, c)p(e|c) \tag{6.8.4}$$

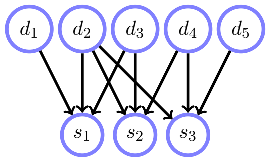


Figure 6.12: 5 diseases giving rise to 3 symptoms. Assuming the symptoms are all instantiated, the triangulated graph of the diseases is a 5 clique.

6.9 The need for approximations

The JTA provides an upper bound on the complexity of (marginal) inference and attempts to exploit the structure of the graph to reduce computations. However, in a great deal of interesting applications the use of the JTA algorithm would result in clique-sizes in the triangulated graph that are prohibitively large.

A classical situation in which this can arise are disease-symptom networks. For example, for the graph in fig(6.12), the triangulated graph of the diseases is fully coupled, meaning that no simplification can occur in general. This situation is common in such bipartite networks, even when the children only have a small number of parents. Intuitively, as one eliminates each parent, links are added between other parents, mediated via the common children. Unless the graph is highly regular, analogous to a form of hidden Markov model, this fill-in effect rapidly results in large cliques and intractable computations.

Dealing with large clique in the triangulated graph is an active research topic and we'll discuss strategies to approximate the computations in chapter(28).

6.9.1 Bounded width Junction trees

In some applications we may be at liberty to choose the structure of the Markov network. For example, if we wish to fit a Markov network to data, we may wish to use as complex a Markov network as we can computationally afford. In such cases we desire that the clique sizes of the resulting triangulated Markov network are smaller than a specified 'tree width' (considering the corresponding junction tree as a hypertree). Constructing such bounded width or 'thin' junction trees is an active research topic. A simple way to do this is to start with a graph and include a randomly chosen edge provided that the size of all cliques in the resulting triangulated graph is below a specified maximal width. See `demoThinJT.m` and `makeThinJT.m` which assumes an initial graph G and a graph of candidate edges C, iteratively expanding G until a maximal tree width limit is reached. See also [11] for a discussion on learning an appropriate Markov structure based on data.

6.10 Code

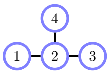
`absorb.m`: Absorption update $\mathcal{V} \rightarrow \mathcal{S} \rightarrow \mathcal{W}$
`absorption.m`: Full Absorption Schedule over Tree
`jtree.m`: Form a Junction Tree
`triangulate.m`: Triangulation based on simple node elimination

6.10.1 Utility Routines

Knowing if an undirected graph is a tree, and returning a valid elimination sequence is useful. A connected graph is a tree if the number of edges plus 1 is equal to the number of nodes. However, for a possibly disconnected graph this is not the case. The code deals with the possibly disconnected case, returning a valid elimination sequence if the graph is singly-connected. The routine is based on the observation that any singly-connected graph must always possess a simplical node which can be eliminated to reveal a smaller singly-connected graph.
`istree.m`: If graph is singly connected return 1 and elimination sequence
`elimtri.m`: Vertex/Node Elimination on a Triangulated Graph, with given end node
`demoJTree.m`: Junction Tree : Chest Clinic

6.11 Notes

6.12 Exercises

Exercise 59. *Show that the Markov network*  *is not perfect elimination ordered and give a perfect elimination labelling for this graph.*

Exercise 60. *Consider the following distribution:*

$$p(x_1, x_2, x_3, x_4) = \phi(x_1, x_2)\phi(x_2, x_3)\phi(x_3, x_4) \tag{6.12.1}$$

1. *Draw a clique graph that represents this distribution and indicate the separators on the graph.*

2. *Write down an alternative formula for the distribution* $p(x_1, x_2, x_3, x_4)$ *in terms of the marginal probabilities* $p(x_1, x_2)$, $p(x_2, x_3)$, $p(x_3, x_4)$, $p(x_2)$, $p(x_3)$

Exercise 61. *Consider the distribution*

$$p(x_1, x_2, x_3, x_4) = \phi(x_1, x_2)\phi(x_2, x_3)\phi(x_3, x_4)\phi(x_4, x_1) \tag{6.12.2}$$

1. *Write down a Junction Tree for the above graph.*

2. *Carry out the absorption procedure and demonstrate that this gives the correct result for the marginal* $p(x_1)$.

Exercise 62. *Consider the distribution*

$$p(a, b, c, d, e, f, g, h, i) = p(a)p(b|a)p(c|a)p(d|a)p(e|b)p(f|c)p(g|d)p(h|e, f)p(i|f, g) \tag{6.12.3}$$

1. *Draw the Belief Network for this distribution.*

2. *Draw the moralised graph.*

3. *Draw the triangulated graph. Your triangulated graph should contain cliques of the smallest size possible.*

4. *Draw a Junction Tree for the above graph and verify that it satisfies the running intersection property.*

5. *Describe a suitable initialisation of clique potentials.*

6. *Describe the absorption procedure and write down an appropriate message updating schedule.*

Exercise 63. *This question concerns the distribution*

$$p(a, b, c, d, e, f) = p(a)p(b|a)p(c|b)p(d|c)p(e|d)p(f|a, e) \tag{6.12.4}$$

1. *Draw the Belief Network for this distribution.*

2. *Draw the moralised graph.*

3. *Draw the triangulated graph. Your triangulated graph should contain cliques of the smallest size possible.*

4. *Draw a Junction Tree for the above graph and verify that it satisfies the running intersection property.*

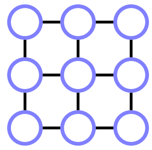
5. *Describe a suitable initialisation of clique potentials.*

6. *Describe the Absorption procedure and an appropriate message updating schedule.*

7. *Show that the distribution can be expressed in the form*

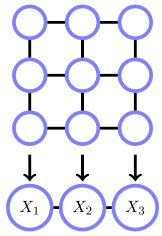
$$p(a|f)p(b|a, c)p(c|a, d)p(d|a, e)p(e|a, f)p(f) \tag{6.12.5}$$

Exercise 64.



For the undirected graph on the square lattice as shown, draw a triangulated graph with the smallest clique sizes possible.

Exercise 65.



Consider a binary variable Markov Random Field $p(x) = Z^{-1} \prod_{i>j} \phi(x_i, x_j)$*, defined on the* $n \times n$ *lattice with* $\phi(x_i, x_j) = e^{\mathbb{I}[x_i = x_j]}$ *for* i *a neighbour of* j *on the lattice and* $i > j$*. A naive way to perform inference is to first stack all the variables in the* t^{th} *column and call this cluster variable* X_t*, as shown. The resulting graph is then singly-connected. What is the complexity of computing the normalisation constant based on this cluster representation? Compute* $\log Z$ *for* $n = 10$*.*

Exercise 66. *Given a consistent Junction Tree on which a full round of message passing has occurred, explain how to form a Belief Network from the Junction Tree.*

Exercise 67. *The file* `diseaseNet.mat` *contains the potentials for a disease bi-partite Belief network, with 20 diseases* $d_1, \ldots, d_{20}$ *and 40 symptoms,* $s_1, \ldots, s_{40}$*. Each disease and symptom is a binary variable, and each symptom connects to 3 parent diseases.*

1. *Using the* BRMLtoolbox*, construct a Junction Tree for this distribution and use it to compute all the marginals of the symptoms,* $p(s_i = 1)$*.*
2. *On most standard computers, computing the marginal* $p(s_i = 1)$ *by raw summation of the joint distribution is computationally infeasible. Explain how to compute the marginals* $p(s_i = 1)$ *in a tractable way without using the Junction Tree formalism. By implementing this method, compare it with the results from the Junction Tree algorithm.*
3. *Consider the (unlikely) scenario in which all the 40 symptom variables are instantiated. Using the Junction Tree, estimate an upper bound on the number of seconds that computing a marginal* $p(d_1|s_{1:40})$ *takes, assuming that for a two clique table containing* S *(joint) states, absorption takes* $O(S\delta)$ *seconds, for an unspecified* δ*. Compare this estimate with the time required to compute the marginal by raw summation of the instantiated Belief network.*

CHAPTER 7

Making Decisions

7.1 Expected Utility

This chapter concerns situations in which decisions need to be taken under uncertainty. Consider the following scenario : you are asked if you wish to take a bet on the outcome of tossing a fair coin. If you bet and win, you gain £100. If you bet and lose, you lose £200. If you don't bet, the cost to you is zero. We can set this up using a two state variable x, with $\text{dom}(x) = \{\mathsf{win}, \mathsf{lose}\}$, a decision variable d with $\text{dom}(d) = \{\mathsf{bet}, \mathsf{no\ bet}\}$ and utilities as follows:

$$U(\mathsf{win}, \mathsf{bet}) = 100, \quad U(\mathsf{lose}, \mathsf{bet}) = -200, \quad U(\mathsf{win}, \mathsf{no\ bet}) = 0, \quad U(\mathsf{lose}, \mathsf{no\ bet}) = 0 \tag{7.1.1}$$

Since we don't know the state of x, in order to make a decision about whether or not to bet, arguably the best we can do is work out our expected winnings/losses under the situations of betting and not betting[237]. If we bet, we would expect to gain

$$U(\mathsf{bet}) = p(\mathsf{win}) \times U(\mathsf{win}, \mathsf{bet}) + p(\mathsf{lose}) \times U(\mathsf{lose}, \mathsf{bet}) = 0.5 \times 100 - 0.5 \times 200 = -50$$

If we don't bet, the expected gain is zero, $U(\mathsf{no\ bet}) = 0$. Based on taking the decision which maximises expected utility, we would therefore be advised not to bet.

Definition 52 (Subjective Expected Utility)**.** The utility of a decision is

$$U(d) = \langle U(d, x) \rangle_{p(x)} \tag{7.1.2}$$

where $p(x)$ is the distribution of the outcome x and d represents the decision.

7.1.1 Utility of money

You are a wealthy individual, with £1,000,000 in your bank account. You are asked if you would like to participate in a fair coin tossing bet in which, if you win, your bank account will become £1,000,000,000. However, if you lose, your bank account will contain only £1000. Assuming the coin is fair, should you take the bet?

If we take the bet our expected bank balance would be

$$U(\mathsf{bet}) = 0.5 \times 1,000,000,000 + 0.5 \times 1000 = 500,000,500.00 \tag{7.1.3}$$

If we don't bet, our bank balance will remain at £1,000,000. Based on expected utility, we are therefore advised to take the bet. (Note that if one considers instead the amount one will win or lose, one may show that the difference in expected utility between betting and not betting is the same, exercise(74)).

Whilst the above is a correct mathematical derivation, few people who are millionaires are likely to be willing to risk losing almost everything in order to become a billionaire. This means that the subjective utility of money is not simply the quantity of money. In order to better reflect the situation, the utility of money would need to be a non-linear function of money, growing slowly for large quantities of money and decreasing rapidly for small quantities of money, exercise(69).

7.2 Decision Trees

Decision trees are a way to graphically organise a sequential decision process. A *decision tree* contains decision nodes, each with branches for each of the alternative decisions. Chance nodes (random variables) also appear in the tree, with the utility of each branch computed at the leaf of each branch. The expected utility of any decision can then be computed on the basis of the weighted summation of all branches from the decision to all leaves from that branch.

Example 29 (Party)**.** Consider the decision problem as to whether or not to go ahead with a fund-raising garden party. The crux is that, if we go ahead with the party and it subsequently rains, then we will lose money (since very few people will show up); on the other hand, if we don't go ahead with the party and it doesn't rain we're free to go and do something else fun. To characterise this numerically, we use:

$$p(Rain = \mathsf{rain}) = 0.6,\ p(Rain = \mathsf{no\ rain}) = 0.4 \tag{7.2.1}$$

The utility is defined as

$$U(\mathsf{party}, \mathsf{rain}) = -100, \quad U(\mathsf{party}, \mathsf{no\ rain}) = 500, \quad U(\mathsf{no\ party}, \mathsf{rain}) = 0, \quad U(\mathsf{no\ party}, \mathsf{no\ rain}) = 50 \tag{7.2.2}$$

We represent this situation in fig(7.1). The question is, should we go ahead with the party? Since we don't know what will actually happen to the weather, we compute the expected utility of each decision:

$$U(\mathsf{party}) = \sum_{Rain} U(\mathsf{party}, Rain)p(Rain) = -100 \times 0.6 + 500 \times 0.4 = 140 \tag{7.2.3}$$

$$U(\mathsf{no\ party}) = \sum_{Rain} U(\mathsf{no\ party}, Rain)p(Rain) = 0 \times 0.6 + 50 \times 0.4 = 20 \tag{7.2.4}$$

Based on expected utility, we are therefore advised to go ahead with the party. The maximal expected utility is given by (see `demoDecParty.m`)

$$\max_{Party} \sum_{Rain} p(Rain)U(Party, Rain) = 140 \tag{7.2.5}$$

Example 30 (Party-Friend)**.** An extension of the Party problem is that if we decide not to go ahead with the party, we have the opportunity to visit a friend. However, we're not sure if this friend will be in. The question is should we still go ahead with the party?

We need to quantify all the uncertainties and utilities. If we go ahead with the party, the utilities are as before:

$$U_{party}(\mathsf{party}, \mathsf{rain}) = -100, \quad U_{party}(\mathsf{party}, \mathsf{no\ rain}) = 500 \tag{7.2.6}$$

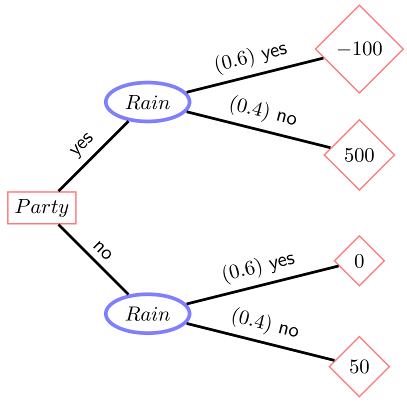


Figure 7.1: A decision tree containing chance nodes (denoted with ovals), decision nodes (denoted with squares) and utility nodes (denoted with diamonds). Note that a decision tree is *not* a graphical representation of a Belief Network with additional nodes. Rather, a decision tree is an explicit enumeration of the possible choices that can be made, beginning with the leftmost decision node, with probabilities on the links out of 'chance' nodes.

with

$$p(Rain = \mathsf{rain}) = 0.6, \quad p(Rain = \mathsf{no\ rain}) = 0.4 \tag{7.2.7}$$

If we decide not to go ahead with the party, we will consider going to visit a friend. In making the decision not to go ahead with the party we have utilities

$$U_{party}\left(\mathsf{no\ party}, \mathsf{rain}\right) = 0, \quad U_{party}\left(\mathsf{no\ party}, \mathsf{no\ rain}\right) = 50 \tag{7.2.8}$$

The probability that the friend is in depends on the weather according to

$$p(Friend = \mathsf{in}|\mathsf{rain}) = 0.8, \quad p(Friend = \mathsf{in}|\mathsf{no\ rain}) = 0.1, \tag{7.2.9}$$

The other probabilities are determined by normalisation. We additionally have

$$U_{visit}\left(\mathsf{friend\ in}, \mathsf{visit}\right) = 200, \quad U_{visit}\left(\mathsf{friend\ out}, \mathsf{visit}\right) = -100 \tag{7.2.10}$$

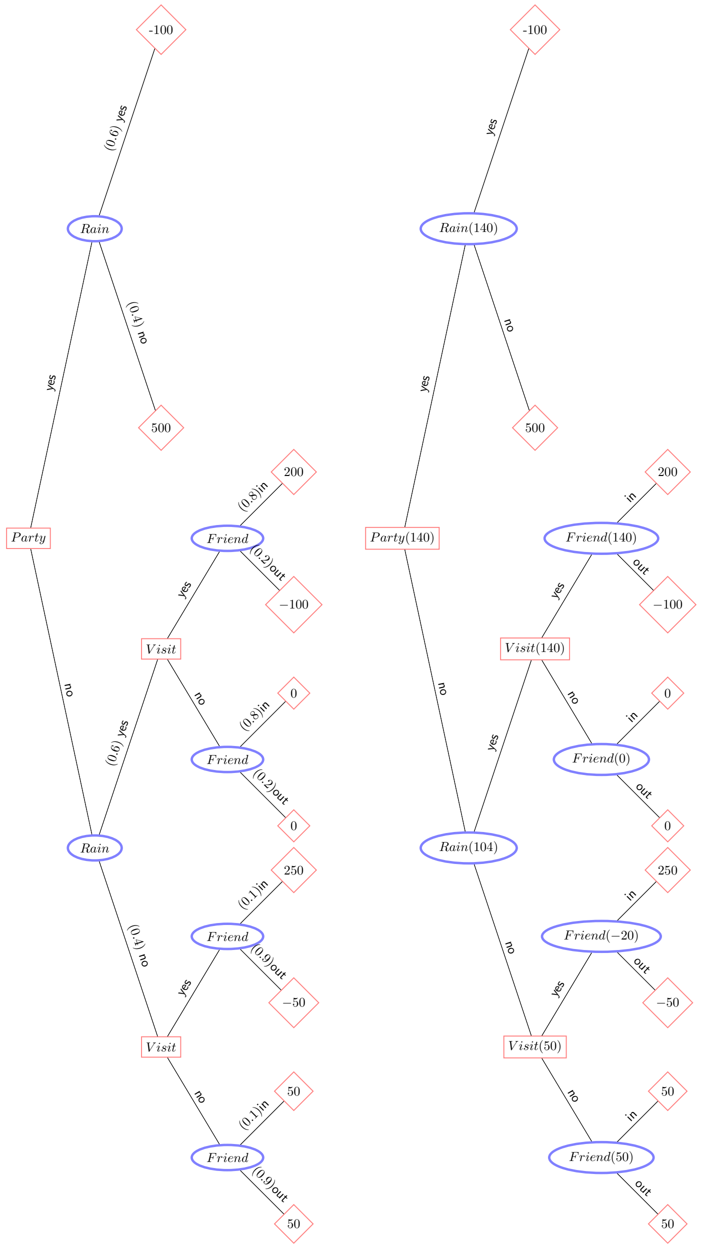
with the remaining utilities zero. The two sets of utilities add up so that the overall utility of any decision sequence is $U_{party} + U_{visit}$. The decision tree for the Party-Friend problem is shown is fig(7.2). For each decision sequence the utility of that sequence is given at the corresponding leaf of the DT. Note that the leaves contain the total utility $U_{party} + U_{visit}$. Solving the DT corresponds to finding for each decision node the maximal expected utility possible (by optimising over future decisions). At any point in the tree choosing that action which leads to the child with highest expected utility will lead to the optimal strategy. Using this, we find that the optimal expected utility has value 140 and is given by going ahead with the party, see `demoDecPartyFriend.m`.

- In DTs the same nodes are often repeated throughout the tree. For a longer sequence of decisions, the number of branches in the tree can grow exponentially with the number of decisions, making this representation impractical.
- In this example the DT is asymmetric since if we decide to go ahead with the party we will not visit the friend, curtailing the further decisions present in the lower half of the tree.

Mathematically, we can express the optimal expected utility U for the Party-Visit example by summing over un-revealed variables and optimising over future decisions:

$$\max_{Party} \sum_{Rain} p(Rain) \max_{Visit} \sum_{Friend} p(Friend|Rain) \left[U_{party}(Party, Rain) + U_{visit}(Visit, Friend)\mathbb{I}\left[Party = \mathsf{no}\right]\right] \tag{7.2.11}$$

where the term $\mathbb{I}\left[Party = \mathsf{no}\right]$ has the effect of curtailing the DT if the party goes ahead. To answer the question as to whether or not to go ahead with the party, we take that state of *Party* that corresponds to



(a) (b)

Figure 7.2: Solving a Decision Tree. **(a)**: Decision Tree for the Party-Visit, example(30). **(b)**: Solving the DT corresponds to making the decision with the highest expected future utility. This can be achieved by starting at the leaves (utilities). For a chance parent node x, the utility of the parent is the expected utility of that variable. For example, at the top of the DT we have the *Rain* variable with the children -100 (probability 0.6) and 500 (probability 0.4). Hence the expected utility of the *Rain* node is $-100 \times 0.6 + 500 \times 0.4 = 140$. For a decision node, the value of the node is the optimum of its child values. One recurses thus backwards from the leaves to the root. For example, the value of the *Rain* chance node in the lower branch is given by $140 \times 0.6 + 50 \times 0.4 = 104$. The optimal decision sequence is then given at each decision node by finding which child node has the maximal value. Hence the overall best decision is to decide to go ahead with the party. If we decided not to do so, and it does not rain, then the best decision we could take would be to not visit the friend (which has an expected utility of 50). A more compact description of this problem is given by the influence diagram, fig(7.4). See also `demoDecPartyFriend.m`.

the maximal expected utility above. The way to read equation (7.2.11) is to start from the last decision that needs to be taken, in this case $Visit$. When we are at the $Visit$ stage we assume that we will have previously made a decision about $Party$ and also will have observed whether or not is is raining. However, we don't know whether or not our friend will be in, so we compute the expected utility by averaging over this unknown. We then take the optimal decision by maximising over $Visit$. Subsequently we move to the next-to-last decision, assuming that what we will do in the future is optimal. Since in the future we will have taken a decision under the uncertain $Friend$ variable, the current decision can then be taken under uncertainty about $Rain$ and maximising this expected optimal utility over $Party$. Note that the sequence of maximisations and summations matters – changing the order will in general result in a different problem with a different expected utility[1].

[1] If one only had a sequence of summations, the order of the summations is irrelevant – likewise for the case of all maximisations. However, summation and maximisation operators do not in general commute.

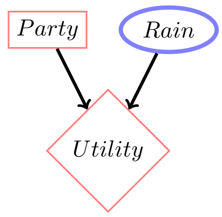


Figure 7.3: An influence diagram which contains random variables (denoted with ovals/circles) Decision nodes (denoted with squares) and Utility nodes (denoted with diamonds). Contrasted with fig(7.1) this is a more compact representation of the structure of the problem. The diagram represents the expression $p(rain)u(party, rain)$. In addition the diagram denotes an ordering of the variables with $party \prec rain$ (according to the convention given by equation (7.3.1)).

7.3 Extending Bayesian Networks for Decisions

An *influence diagram* is a Bayesian Network with additional Decision nodes and Utility nodes [136, 146, 159]. The decision nodes have no associated distribution; the utility nodes are deterministic functions of their parents. The utility and decision nodes can be either continuous or discrete; for simplicity, in the examples here the decisions will be discrete.

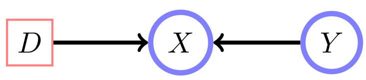
A benefit of decision trees is that they are general and explicitly encode the utilities and probabilities associated with each decision and event. In addition, we can readily solve small decision problems using decision trees. However, when the sequence of decisions increases, the number of leaves in the decision tree grows and representing the tree can become an exponentially complex problem. In such cases it can be useful to use an Influence Diagram (ID). An ID states which information is required in order to make each decision, and the order in which these decisions are to be made. The details of the probabilities and rewards are not specified in the ID, and this can enable a more compact description of the decision problem.

7.3.1 Syntax of influence diagrams

Information Links An *information link* from a random variable into a decision node:

indicates that the state of the variable X will be known before decision D is taken. Information links from another decision node d in to D similarly indicate that decision d is known before decision D is taken. We use a dashed link to denote that decision D is not functionally related to its parents.

Random Variables Random Variables may depend on the states of parental random variables (as in Belief Networks), but also Decision Node states:



As decisions are taken, the states of some random variables will be revealed. To emphasise this we typically shade a node to denote that its state will be revealed during the sequential decision process.

Utilities A utility node is a deterministic function of its parents. The parents can be either random variables or decision nodes.

In the party example, the BN trivially consists of a single node, and the Influence Diagram is given in fig(7.3). The more complex Party-Friend problem is depicted in fig(7.4). The ID generally provides a more compact representation of the structure of problem than a DT, although details about the specific probabilities and utilities are not present in the ID.

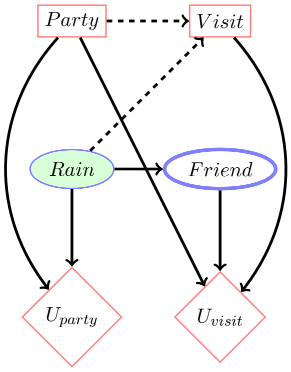


Figure 7.4: An influence diagram for the party-visit problem, example(30). The partial ordering is $Party^* \prec Rain \prec Visit^* \prec Friend$. The dashed-link from party to visit is not strictly necessary but retained in order to satisfy the convention that there is a directed path connecting all decision nodes.

Partial Ordering

An ID defines a *partial ordering* of the nodes. We begin by writing those variables $\mathcal{X}_0$ whose states are known (evidential variables) before the first decision D_1. We then find that set of variables $\mathcal{X}_1$ whose states are revealed before the second decision D_2. Subsequently the set of variables $\mathcal{X}_t$ is revealed before decision D_{t+1}. The remaining fully-unobserved variables are placed at the end of the ordering:

$$\mathcal{X}_0 \prec D_1 \prec \mathcal{X}_1 \prec D_2, \ldots, \prec \mathcal{X}_{n-1} \prec D_n \prec \mathcal{X}_n \tag{7.3.1}$$

with $\mathcal{X}_k$ being the variables revealed between decision D_k and D_{k+1}. The term 'partial' refers to the fact that there is no order implied amongst the variables within the set $\mathcal{X}_n$. For notational clarity, at points below we will indicate decision variables with * to reinforce that we maximise over these variables, and sum over the non-starred variables. Where the sets are empty we omit writing them. For example, in fig(7.5a) the ordering is $Test^* \prec Seismic \prec Drill^* \prec Oil$.

The optimal first decision D_1 is determined by computing

$$U(D_1|\mathcal{X}_0) \equiv \sum_{\mathcal{X}_1} \max_{D_2} \ldots \sum_{\mathcal{X}_{n-1}} \max_{D_n} \sum_{\mathcal{X}_n} \prod_{i\in\mathcal{I}} p\left(x_i|\mathrm{pa}\left(x_i\right)\right) \sum_{j\in\mathcal{J}} U_j\left(\mathrm{pa}\left(u_j\right)\right) \tag{7.3.2}$$

for each state of the decision D_1, given $\mathcal{X}_0$. In equation (7.3.2) above $\mathcal{I}$ denotes the set of indices for the random variables, and $\mathcal{J}$ the indices for the utility nodes. For each state of the conditioning variables, the optimal decision D_1 is found using

$$\operatorname*{argmax}_{D_1} U(D_1|\mathcal{X}_0) \tag{7.3.3}$$

Remark 8 (Reading off the partial ordering)**.** Sometimes it can be tricky to read the partial ordering from the ID. A method is to identify the first decision D_1 and then any variables $\mathcal{X}_0$ that need to be observed to make that decision. Then identify the next decision D_2 and the variables $\mathcal{X}_1$ that are revealed after decision D_1 is taken and before decision D_2 is taken, *etc.* This gives the partial ordering $\mathcal{X}_0 \prec D_1 \prec \mathcal{X}_1 \prec D_2, \ldots$. Place any unrevealed variables at the end of the ordering.

Implicit and explicit information links

The information links are a potential source of confusion. An information link specifies explicitly which quantities are known before that decision is taken[2]. We also implicitly assume the *no forgetting principle* that all past decisions and revealed variables are available at the current decision (the revealed variables are necessarily the parents of all past decision nodes). If we were to include all such information links, IDs would get potentially rather messy. In fig(7.5), both explicit and implicit information links are demonstrated. We call an information link *fundamental* if its removal would alter the partial ordering.

[2]Some authors prefer to write all information links where possible, and others prefer to leave them implicit. Here we largely take the implicit approach. For the purposes of computation, all that is required is a partial ordering; one can therefore view this as 'basic' and the information links as superficial (see [68]).

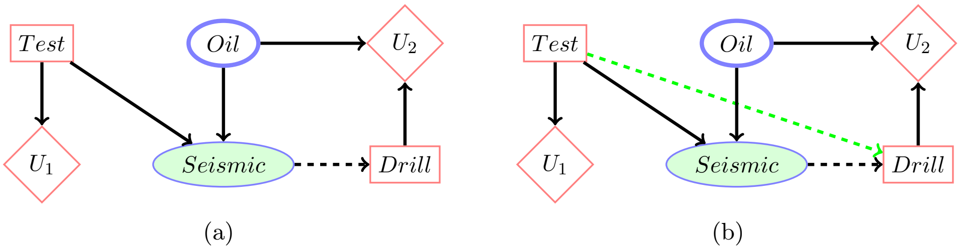


Figure 7.5: **(a)**: The partial ordering is $Test^* \prec Seismic \prec Drill^* \prec Oil$. The explicit information links from $Test$ to $Seismic$ and from $Seismic$ to $Drill$ are both fundamental in the sense that removing either results in a different partial ordering. The shaded node emphasises that the state of this variable will be revealed during the sequential decision process. Conversely, the non-shaded node will never be observed. **(b)**: Based on the ID in (a), there is an implicit link from $Test$ to $Drill$ since the decision about $Test$ is taken before $Seismic$ is revealed.

Causal Consistency

For an Influence Diagram to be consistent a current decision cannot affect the past. This means that any random variable descendants of a decision D in the ID must come later in the partial ordering. Assuming the no-forgetting principle, this means that for any valid ID there must be a directed path connecting all decisions. This can be a useful check on the consistency of an ID.

Asymmetry

IDs are most convenient when the corresponding DT is symmetric. However, some forms of asymmetry are relatively straightforward to deal with in the ID framework. For our party-visit example, the DT is asymmetric. However, this is easily dealt with in the ID by using a link from $Party$ to U_{visit} which removes the contribution from U_{visit} when the party goes ahead.

More complex issues arise when the set of variables than can be observed depends on the decision sequence taken. In this case the DT is asymmetric. In general, Influence Diagrams are not well suited to modelling such asymmetries, although some effects can be mediated either by careful use of additional variables, or extending the ID notation. See [68] and [146] for further details of these issues and possible resolutions.

Example 31 (Should I do a PhD?)**.** Consider a decision whether or not to do PhD as part of our education (E). Taking a PhD incurs costs, U_C both in terms of fees, but also in terms of lost income. However, if we have a PhD, we are more likely to win a Nobel Prize (P), which would certainly be likely to boost our Income (I), subsequently benefitting our finances (U_B). This setup is depicted in fig(7.6a). The ordering is (eliding empty sets)

$$E^* \prec \{I, P\} \tag{7.3.4}$$

and

$$\text{dom}(E) = (\textsf{do PhD}, \textsf{no PhD}), \quad \text{dom}(I) = (\textsf{low}, \textsf{average}, \textsf{high}), \quad \text{dom}(P) = (\textsf{prize}, \textsf{no prize}) \tag{7.3.5}$$

The probabilities are

$$p(\textsf{win Nobel prize}|\textsf{no PhD}) = 0.0000001 \qquad p(\textsf{win Nobel prize}|\textsf{do PhD}) = 0.001 \tag{7.3.6}$$

$$\begin{array}{lll}
p(\textsf{low}|\textsf{do PhD}, \textsf{no prize}) = 0.1 & p(\textsf{average}|\textsf{do PhD}, \textsf{no prize}) = 0.5 & p(\textsf{high}|\textsf{do PhD}, \textsf{no prize}) = 0.4 \\
p(\textsf{low}|\textsf{no PhD}, \textsf{no prize}) = 0.2 & p(\textsf{average}|\textsf{no PhD}, \textsf{no prize}) = 0.6 & p(\textsf{high}|\textsf{no PhD}, \textsf{no prize}) = 0.2 \\
p(\textsf{low}|\textsf{do PhD}, \textsf{prize}) = 0.01 & p(\textsf{average}|\textsf{do PhD}, \textsf{prize}) = 0.04 & p(\textsf{high}|\textsf{do PhD}, \textsf{prize}) = 0.95 \\
p(\textsf{low}|\textsf{no PhD}, \textsf{prize}) = 0.01 & p(\textsf{average}|\textsf{no PhD}, \textsf{prize}) = 0.04 & p(\textsf{high}|\textsf{no PhD}, \textsf{prize}) = 0.95
\end{array}$$

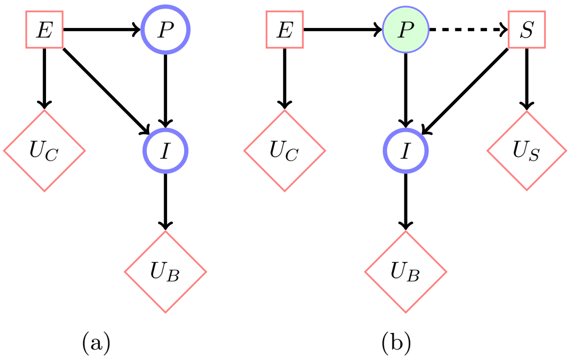


Figure 7.6: **(a)**: Education E incurs some cost, but also gives a chance to win a prestigious science prize. Both of these effect our likely Incomes, with corresponding long term financial benefits. **(b)**: The start-up scenario.

$$ \tag{7.3.7} $$

The utilities are

$$U_C\,(\textsf{do PhD}) = -50000, \quad U_C\,(\textsf{no PhD}) = 0, \tag{7.3.8}$$

$$U_B\,(\textsf{low}) = 100000, \quad U_B\,(\textsf{average}) = 200000, \quad U_B\,(\textsf{high}) = 500000 \tag{7.3.9}$$

The expected utility of Education is

$$U(E) = \sum_{I,P} p(I|E,P)p(P|E)\left[U_C(E) + U_B(I)\right] \tag{7.3.10}$$

so that $U(\textsf{do phd}) = 260174.000$, whilst not taking a PhD is $U(\textsf{no phd}) = 240000.0244$, making it on average beneficial to do a PhD. See `demoDecPhD.m`.

Example 32 (PhDs and Start-up companies)**.** Influence Diagrams are particularly useful when a sequence of decisions is taken. For example, in fig(7.6b) we model a new situation in which someone has first decided whether or not to take a PhD. Ten years later in their career they decide whether or not to make a start-up company. This decision is based on whether or not they won the Nobel Prize. The start-up decision is modelled by S with $\mathrm{dom}(S) = (\textsf{tr}, \textsf{fa})$. If we make a start-up, this will cost some money in terms of investment. However, the potential benefit in terms of our income could be high.

We model this with (the other required table entries being taken from example(31)):

$$\begin{array}{lll}
p(\textsf{low}|\textsf{start up}, \textsf{no prize}) = 0.1 & p(\textsf{average}|\textsf{start up}, \textsf{no prize}) = 0.5 & p(\textsf{high}|\textsf{start up}, \textsf{no prize}) = 0.4 \\
p(\textsf{low}|\textsf{no start up}, \textsf{no prize}) = 0.2 & p(\textsf{average}|\textsf{no start up}, \textsf{no prize}) = 0.6 & p(\textsf{high}|\textsf{no start up}, \textsf{no prize}) = 0.2 \\
p(\textsf{low}|\textsf{start up}, \textsf{prize}) = 0.005 & p(\textsf{average}|\textsf{start up}, \textsf{prize}) = 0.005 & p(\textsf{high}|\textsf{start up}, \textsf{prize}) = 0.99 \\
p(\textsf{low}|\textsf{no start up}, \textsf{prize}) = 0.05 & p(\textsf{average}|\textsf{no start up}, \textsf{prize}) = 0.15 & p(\textsf{high}|\textsf{no start up}, \textsf{prize}) = 0.8
\end{array} \tag{7.3.11}$$

and

$$U_S\,(\textsf{start up}) = -200000, \qquad U_S\,(\textsf{no start up}) = 0 \tag{7.3.12}$$

Our interest is to advise whether or not it is desirable (in terms of expected utility) to take a PhD, now bearing in mind that later one may or may not win the Nobel Prize, and may or may not make a start-up company.

The ordering is (eliding empty sets)

$$E^* \prec P \prec S^* \prec I \tag{7.3.13}$$

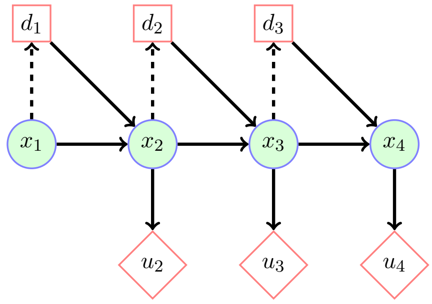


Figure 7.7: Markov Decision Process. These can be used to model planning problems of the form 'how do I get to where I want to be incurring the lowest total cost?'. They are readily solvable using a message passing algorithm.

The expected optimal utility for any state of E is

$$U(E) = \sum_P \max_S \sum_I p(I|S,P)p(P|E)\left[U_S(S) + U_C(E) + U_B(I)\right] \tag{7.3.14}$$

where we assume that the optimal decisions are taken in the future. Computing the above, we find

$$U(\textsf{do PhD}) = 190195.00, \qquad U(\textsf{no PhD}) = 240000.02 \tag{7.3.15}$$

Hence, we are better off not doing a PhD. See `demoDecPhd.m`.

7.4 Solving Influence Diagrams

Solving an influence diagram means computing the optimal decision or sequence of decisions. Here we focus on finding the optimal first decision. The direct approach is to take equation (7.3.2) and perform the required sequence of summations and maximisations explicitly. However, we may be able to exploit the structure of the problem to for computational efficiency. To develop this we first derive an efficient algorithm for a highly structured ID, the Markov Decision Process, which we will discuss further in section(7.5).

7.4.1 Efficient Inference

Consider the following function from the ID of fig(7.7)

$$\phi(x_4, x_3, d_3)\phi(x_3, x_2, d_2)\phi(x_2, x_1, d_1)\left(u(x_2) + u(x_3) + u(x_4)\right) \tag{7.4.1}$$

where the ϕ represent conditional probabilities and the u are utilities. We write this in terms of potentials since this will facilitate the generalisation to other cases. Our task is to take the optimal first decision, based on the expected optimal utility

$$U(d_1) = \sum_{x_2} \max_{d_2} \sum_{x_3} \max_{d_3} \sum_{x_4} \phi(x_4, x_3, d_3)\phi(x_3, x_2, d_2)\phi(x_2, x_1, d_1)\left(u(x_2) + u(x_3) + u(x_4)\right) \tag{7.4.2}$$

Whilst we could carry out the sequence of maximisations and summations naively, our interest is to derive a computationally efficient approach. Let's see how to distribute these operations 'by hand'. Since only $u(x_4)$ depends on x_4 explicitly we can write

$$\begin{aligned} U(d_1) = &\sum_{x_2} \phi(x_2, x_1, d_1) \max_{d_2} \sum_{x_3} \phi(x_3, x_2, d_2) \max_{d_3} \sum_{x_4} \phi(x_4, x_3, d_3) u(x_4) \\ &+ \sum_{x_2} \phi(x_2, x_1, d_1) \max_{d_2} \sum_{x_3} \phi(x_3, x_2, d_2) u(x_3) \max_{d_3} \sum_{x_4} \phi(x_4, x_3, d_3) \\ &+ \sum_{x_2} \phi(x_2, x_1, d_1) u(x_2) \max_{d_2} \sum_{x_3} \phi(x_3, x_2, d_2) \max_{d_3} \sum_{x_4} \phi(x_4, x_3, d_3) \end{aligned} \tag{7.4.3}$$

Starting with the first line and carrying out the summation over x_4 and max over d_3, this gives a new function of x_3,

$$u_{3\leftarrow 4}(x_3) \equiv \max_{d_3} \sum_{x_4} \phi(x_4, x_3, d_3) u(x_4) \tag{7.4.4}$$

In addition we define the message (which in our particular example will be unity)

$$\phi_{3\leftarrow 4}(x_3) \equiv \max_{d_3} \sum_{x_4} \phi(x_4, x_3, d_3) \tag{7.4.5}$$

Using this we can write

$$U(d_1) = \sum_{x_2} \phi(x_2, x_1, d_1) \max_{d_2} \sum_{x_3} \phi(x_3, x_2, d_2) \left[u(x_3)\phi_{3\leftarrow 4}(x_3) + u_{3\leftarrow 4}(x_3)\right] \\ + \sum_{x_2} \phi(x_2, x_1, d_1) u(x_2) \max_{d_2} \sum_{x_3} \phi(x_3, x_2, d_2)\phi_{3\leftarrow 4}(x_3) \tag{7.4.6}$$

Now we carry out the sum over x_3 and max over d_2 for the first row above and define a utility message

$$u_{2\leftarrow 3}(x_2) \equiv \max_{d_2} \sum_{x_3} \phi(x_3, x_2, d_2) \left[u(x_3)\phi_{3\leftarrow 4}(x_3) + u_{3\leftarrow 4}(x_3)\right] \tag{7.4.7}$$

and probability message[3]

$$\phi_{2\leftarrow 3}(x_2) \equiv \max_{d_2} \sum_{x_3} \phi(x_3, x_2, d_2)\phi_{3\leftarrow 4}(x_3) \tag{7.4.8}$$

The optimal decision for d_1 can be obtained from

$$U(d_1) = \sum_{x_2} \phi(x_2, x_1, d_1) \left[u(x_2)\phi_{2\leftarrow 3}(x_2) + u_{2\leftarrow 3}(x_2)\right]$$

Since the probability message $\phi_{2\leftarrow 3}(x_2)$ represents information about the distribution passed to x_2 via x_3, it is more intuitive to write

$$U(d_1) = \sum_{x_2} \phi(x_2, x_1, d_1)\phi_{2\leftarrow 3}(x_2) \left[u(x_2) + \frac{u_{2\leftarrow 3}(x_2)}{\phi_{2\leftarrow 3}(x_2)}\right]$$

which has the interpretation of the average of a utility with respect to a distribution.

It is intuitively clear that we can continue along this line for richer structures than chains. Indeed, provided we have formed an appropriate Junction Tree, we can pass potential and utility messages from clique to neighbouring clique, as described in the following section.

7.4.2 Using a Junction Tree

In complex IDs computational efficiency in carrying out the series of summations and maximisations may be an issue and one therefore seeks to exploit structure in the ID. It is intuitive that some form of junction tree style algorithm is applicable. We can first represent an ID using decision potentials which consist of two parts, as defined below.

Definition 53 (Decision Potential)**.** A *decision potential* on a clique C contains two potentials: a *probability potential* ρ_C and a *utility potential* μ_C. The joint potentials for the junction tree are defined as

$$\rho = \prod_{C\in\mathcal{C}} \rho_C, \qquad \mu = \sum_{C\in\mathcal{C}} \mu_C \tag{7.4.9}$$

with the junction tree representing the term $\rho\mu$.

[3]For our MDP example all these probability messages are unity.

In this case there are constraints on the triangulation, imposed by the partial ordering which restricts the variables elimination sequence. This results in a so-called *strong Junction Tree*. The treatment here is inspired by [144]; a related approach which deals with more general chain graphs is given in [68]. The sequence of steps required to construct a JT for an ID is as follows:

Remove Information Edges Parental links of decision nodes are removed[4].

Moralization Marry all parents of the remaining nodes.

Remove Utility Nodes Remove the utility nodes and their parental links.

Strong Triangulation Form a triangulation based on an elimination order which obeys the partial ordering of the variables.

Strong Junction Tree From the strongly triangulated graph, form a Junction Tree and orient the edges towards the strong root (the clique that appears last in the elimination sequence).

The cliques are ordered according to the sequence in which they are eliminated. The separator probability cliques are initialised to the identity, with the separator utilities initialised to zero. The probability cliques are then initialised by placing conditional probability factors into the lowest available clique (according to the elimination order) that can contain them, and similarly for the utilities. Remaining probability cliques are set to the identity and utility cliques to zero.

Example 33 (Junction Tree)**.** An example of a junction tree for an ID is given in fig(7.8a). The moralisation and triangulation links are given in fig(7.8b). The orientation of the edges follows the partial ordering with the leaf cliques being the first to disappear under the sequence of summations and maximisations.

A by-product of the above steps is that the cliques describe the fundamental dependencies on previous decisions and observations. In fig(7.8a), for example, the information link from f to D_2 is not present in the moralised-triangulated graph fig(7.8b), nor in the associated cliques of fig(7.8c). This is because once e is revealed, the utility U_4 is independent of f, giving rise to the two-branch structure in fig(7.8b). Nevertheless, the information link from f to D_2 is fundamental since it specifies that f will be revealed – removing this link would therefore change the partial ordering.

Absorption

By analogy with the definition of messages in section(7.4.1), for two neighbouring cliques C_1 and C_2, where C_1 is closer to the strong root of the JT (the last clique defined through the elimination order), we define

$$\rho_S = \sum_{C_2 \backslash S}^{*} \rho_{C_2}, \qquad \mu_S = \sum_{C_2 \backslash S}^{*} \rho_{C_2}\mu_{C_2} \tag{7.4.10}$$

$$\rho_{C_1}^{new} = \rho_{C_1}\rho_S, \qquad \mu_{C_1}^{new} = \mu_{C_1} + \frac{\mu_S}{\rho_S} \tag{7.4.11}$$

In the above $\sum_C^*$ is a 'generalised marginalisation' operation – it sums over those elements of clique C which are random variables and maximises over the decision variables in the clique. The order of this sequence of sums and maximisations follows the partial ordering defined by $\prec$.

Absorption is then computed from the leaves inwards to the root of the strong Junction Tree. The optimal setting of a decision D_1 can then be computed from the root clique. Subsequently backtracking may be

[4]Note that for the case in which the domain is dependent on the parental variables, such links must remain.

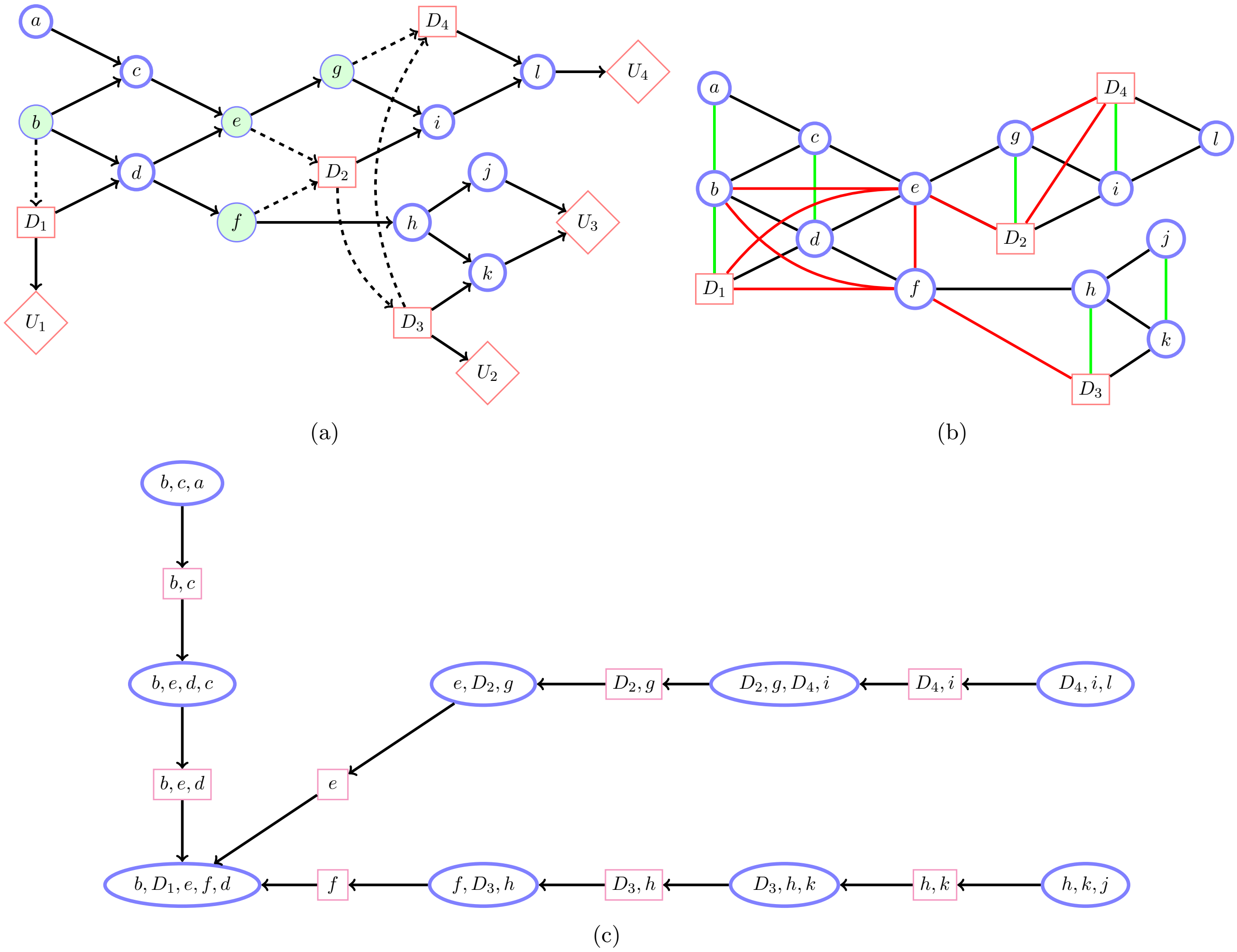
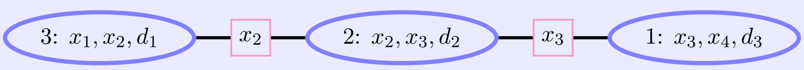


Figure 7.8: **(a)**: Influence Diagram, adapted from [144]. Causal consistency is satisfied since there is a directed path linking the all decisions in sequence. The partial ordering is $b \prec D_1 \prec (e, f) \prec D_2 \prec (\cdot) \prec D_3 \prec g \prec D_4 \prec (a, c, d, h, i, j, k, l)$. **(b)**: Moralised and strongly triangulated graph. Moralisation links are in green, strong triangulation links are in red. **(c)**: Strong Junction Tree. Absorption passes information from the leaves of the tree towards the root.

applied to infer the optimal decision trajectory. The optimal decision for D can be obtained by working with the clique containing D which is closest to the strong root and setting any previously taken decisions and revealed observations into their evidential states. See `demoDecAsia.m` for an example.

Example 34 (Absorption on a chain)**.** For the ID of fig(7.7), the moralisation and triangulation steps are trivial and give the JT:



where the cliques are indexed according the elimination order. The probability and utility cliques are initialised to

$$\begin{aligned} &\rho_3(x_1, x_2, d_1) = p(x_2|x_1, d_1) \quad \mu_3(x_1, x_2, d_1) = 0 \\ &\rho_2(x_2, x_3, d_2) = p(x_3|x_2, d_2) \quad \mu_2(x_2, x_3, d_2) = u(x_2) \\ &\rho_1(x_3, x_4, d_3) = p(x_4|x_3, d_3) \quad \mu_1(x_3, x_4, d_3) = u(x_3) + u(x_4) \end{aligned} \tag{7.4.12}$$

with the separator cliques initialised to

$$\begin{array}{ll} \rho_{1-2}(x_3) = 1 & \mu_{1-2}(x_3) = 0 \\ \rho_{2-3}(x_2) = 1 & \mu_{2-3}(x_2) = 0 \end{array} \tag{7.4.13}$$

Updating the separator we have the new probability potential

$$\rho_{1-2}(x_3)^* = \max_{d_3} \sum_{x_4} \rho_1(x_3, x_4, d_3) = 1 \tag{7.4.14}$$

and utility potential

$$\mu_{1-2}(x_3)^* = \max_{d_3} \sum_{x_4} \rho_1(x_3, x_4, d_3)\, \mu_1(x_3, x_4, d_3) = \max_{d_3} \sum_{x_4} p(x_4|x_3, d_3)\,(u(x_3) + u(x_4)) \tag{7.4.15}$$

$$= \max_{d_3} \left(u(x_3) + \sum_{x_4} p(x_4|x_3, d_3) u(x_4) \right) \tag{7.4.16}$$

At the next step we update the probability potential

$$\rho_2(x_2, x_3, d_2)^* = \rho_2(x_2, x_3, d_2)\, \rho_{1-2}(x_3)^* = 1 \tag{7.4.17}$$

and utility potential

$$\mu_2(x_2, x_3, d_2)^* = \mu_2(x_2, x_3, d_2) + \frac{\mu_{1-2}(x_3)^*}{\rho_{1-2}(x_3)} = u(x_2) + \max_{d_3} \left(u(x_3) + \sum_{x_4} p(x_4|x_3, d_3) u(x_4) \right) \tag{7.4.18}$$

The next separator decision potential is

$$\rho_{2-3}(x_2)^* = \max_{d_2} \sum_{x_3} \rho_2(x_2, x_3, d_2)^* = 1 \tag{7.4.19}$$

$$\mu_{2-3}(x_2)^* = \max_{d_2} \sum_{x_3} \rho_2(x_2, x_3, d_2)\, \mu_2(x_2, x_3, d_2)^* \tag{7.4.20}$$

$$= \max_{d_2} \sum_{x_3} p(x_3|x_2, d_2) \left(u(x_2) + \max_{d_3} \left(u(x_3) + \sum_{x_4} p(x_4|x_3, d_3) u(x_4) \right) \right) \tag{7.4.21}$$

Finally we end up with the root decision potential

$$\rho_3(x_1, x_2, d_1)^* = \rho_3(x_1, x_2, d_1)\, \rho_{2-3}(x_2)^* = p(x_2|x_1, d_1) \tag{7.4.22}$$

and

$$\mu_3(x_1, x_2, d_1)^* = \mu_3(x_2, x_1, d_1) + \frac{\mu_{2-3}(x_2)^*}{\rho_{2-3}(x_2)^*} \tag{7.4.23}$$

$$= \max_{d_2} \sum_{x_3} p(x_3|x_2, d_2) \left(u(x_2) + \max_{d_3} \left(u(x_3) + \sum_{x_4} p(x_4|x_3, d_3) u(x_4) \right) \right) \tag{7.4.24}$$

From the final decision potential we have the expression

$$\rho_3(x_1, x_2, d_1)^*\, \mu_3(x_1, x_2, d_1)^* \tag{7.4.25}$$

which is equivalent to that which would be obtained by simply distributing the summations and maximisations over the original ID. At least for this special case, we therefore have verified that the JT approach yields the correct root clique potentials.

7.5 Markov Decision Processes

Consider a Markov chain with transition probabilities $p(x_{t+1} = j|x_t = i)$. At each time t we consider an action (decision), which affects the state at time $t + 1$. We describe this by

$$p(x_{t+1} = i|x_t = j, d_t = k) \tag{7.5.1}$$

Associated with each state $x_t = i$ is a utility $u(x_t = i)$, and is schematically depicted in fig(7.7). One use of such an environment model would be to help plan a sequence of actions (decisions) required to reach a goal state in minimal total summed cost.

More generally one could consider utilities that depend on transitions and decisions, $u(x_{t+1} = i, x_t = j, d_t = k)$ and also time dependent versions of all of these, $p_t(x_{t+1} = i|x_t = j, d_t = k)$, $u_t(x_{t+1} = i, x_t = j, d_t = k)$. We'll stick with the time-independent (stationary) case here since the generalisations are conceptually straightforward at the expense of notational complexity.

MDPs can be used to solve planning tasks such as how can one get to a desired goal state as quickly as possible. By defining the utility of being in the goal state as high, and being in the non-goal state as a low value, at each time t, we have a utility $u(x_t)$ of being in state x_t. For positive utilities, the total utility of any state-decision path $x_{1:T}, d_{1:T}$ is defined as (assuming we know the initial state x_1)

$$U(x_{1:T}) \equiv \sum_{t=2}^{T} u(x_t) \tag{7.5.2}$$

and the probability with which this happens is given by

$$p(x_{2:T}|x_1, d_{1:T-1}) = \prod_{t=1}^{T-1} p(x_{t+1}|x_t, d_t) \tag{7.5.3}$$

At time $t = 1$ we want to make that decision d_1 that will lead to maximal expected total utility

$$U(d_1) \equiv \sum_{x_2} \max_{d_2} \sum_{x_3} \max_{d_3} \sum_{x_4} \ldots \max_{d_{T-1}} \sum_{x_T} p(x_{2:T}|x_1, d_{1:T-1})U(x_{1:T}) \tag{7.5.4}$$

Our task is to compute $U(d_1)$ for each state of d_1 and then choose that state with maximal expected total utility. To carry out the summations and maximisations efficiently, we could use the junction tree approach, as described in the previous section. However, in this case, the ID is sufficiently simple that a direct message passing approach can be used to compute the expected utility.

7.5.1 Maximising expected utility by message passing

Consider the MDP

$$\prod_{t=1}^{T-1} p(x_{t+1}|x_t, d_t) \sum_{t=2}^{T} u(x_t) \tag{7.5.5}$$

For the specific example in fig(7.7) the joint model of the BN and utility is

$$p(x_4|x_3, d_3)p(x_3|x_2, d_2)p(x_2|x_1, d_1) \left(u(x_2) + u(x_3) + u(x_4)\right) \tag{7.5.6}$$

To decide on how to take the first optimal decision, we need to compute

$$U(d_1) = \sum_{x_2} \max_{d_2} \sum_{x_3} \max_{d_3} \sum_{x_4} p(x_4|x_3, d_3)p(x_3|x_2, d_2)p(x_2|x_1, d_1) \left(u(x_2) + u(x_3) + u(x_4)\right) \tag{7.5.7}$$

Since only $u(x_4)$ depends on x_4 explicitly, we can write

$$\begin{aligned} U(d_1) = & \sum_{x_2} \max_{d_2} \sum_{x_3} \max_{d_3} \sum_{x_4} p(x_4|x_3, d_3)p(x_3|x_2, d_2)p(x_2|x_1, d_1)u(x_4) \\ & + \sum_{x_2} \max_{d_2} \sum_{x_3} p(x_3|x_2, d_2)p(x_2|x_1, d_1)u(x_3) \\ & + \sum_{x_2} p(x_2|x_1, d_1)u(x_2) \end{aligned} \tag{7.5.8}$$

For each line we distribute the operations:

$$U(d_1) = \sum_{x_2} p(x_2|x_1, d_1) \max_{d_2} \sum_{x_3} p(x_3|x_2, d_2) \max_{d_3} \sum_{x_4} p(x_4|x_3, d_3)u(x_4)$$
$$+ \sum_{x_2} p(x_2|x_1, d_1) \max_{d_2} \sum_{x_3} p(x_3|x_2, d_2)u(x_3)$$
$$+ \sum_{x_2} p(x_2|x_1, d_1)u(x_2) \tag{7.5.9}$$

We now start with the first line and carry out the summation over x_4 and maximisation over d_3. This gives a new function of x_3,

$$u_{3\leftarrow 4}(x_3) \equiv \max_{d_3} \sum_{x_4} p(x_4|x_3, d_3)u(x_4) \tag{7.5.10}$$

which we can incorporate in the next line

$$U(d_1) = \sum_{x_2} p(x_2|x_1, d_1) \max_{d_2} \sum_{x_3} p(x_3|x_2, d_2) \left[u(x_3) + u_{3\leftarrow 4}(x_3)\right]$$
$$+ \sum_{x_2} p(x_2|x_1, d_1)u(x_2) \tag{7.5.11}$$

Similarly, we can now carry out the sum over x_3 and max over d_2 to define a new function

$$u_{2\leftarrow 3}(x_2) \equiv \max_{d_2} \sum_{x_3} p(x_3|x_2, d_2) \left[u(x_3) + u_{3\leftarrow 4}(x_3)\right] \tag{7.5.12}$$

to give

$$U(d_1) = \sum_{x_2} p(x_2|x_1, d_1) \left[u(x_2) + u_{2\leftarrow 3}(x_2)\right] \tag{7.5.13}$$

Given $U(d_1)$ above, we can then find the optimal decision d_1 by

$$d_1^* = \underset{d_1}{\text{argmax}}\ U(d_1) \tag{7.5.14}$$

What about d_2^*? Bear in mind that when we come to make decision d_2 we will have observed x_1, x_2 and d_2. We can then find d_2^* by

$$\underset{d_2}{\text{argmax}} \sum_{x_3} p(x_3|x_2, d_2) \left[u(x_3) + u_{3\leftarrow 4}(x_3)\right] \tag{7.5.15}$$

Subsequently we can backtrack further to find d_3^*. In general, the optimal decision is given by

$$d_{t-1}^* = \underset{d_{t-1}}{\text{argmax}} \sum_{x_t} p(x_t|x_{t-1}, d_{t-1}) \left[u(x_t) + u_{t\leftarrow t+1}(x_t)\right] \tag{7.5.16}$$

7.5.2 Bellman's equation

In a Markov Decision Process, as above, we can define utility messages recursively as

$$u_{t-1\leftarrow t}(x_{t-1}) \equiv \max_{d_{t-1}} \sum_{x_t} p(x_t|x_{t-1}, d_{t-1}) \left[u(x_t) + u_{t\leftarrow t+1}(x_t)\right] \tag{7.5.17}$$

It is more common to define the *value* of being in state x_t as

$$v_t(x_t) \equiv u(x_t) + u_{t\leftarrow t+1}(x_t), \qquad v_T(x_T) = u(x_T) \tag{7.5.18}$$

and write then the equivalent recursion

$$v_{t-1}(x_{t-1}) = u(x_{t-1}) + \max_{d_{t-1}} \sum_{x_t} p(x_t|x_{t-1}, d_{t-1})v_t(x_t) \tag{7.5.19}$$

The optimal decision d_t^* is then given by

$$d_t^* = \underset{d_t}{\text{argmax}} \sum_{x_{t+1}} p(x_{t+1}|x_t, d_t)v(x_{t+1}) \tag{7.5.20}$$

Equation(7.5.19) is called Bellman's equation[30][5].

7.6 Temporally unbounded MDPs

In the previous discussion about MDPs we assumed a given end time, T, from which one can propagate messages back from the end of the chain. The infinite T case would appear to be ill-defined since the sum of utilities

$$u(x_1) + u(x_2) + \ldots + u(x_T) \tag{7.6.1}$$

will in general be unbounded. There is a simple way to avoid this difficulty. If we let $u^* = \max_s u(s)$ be the largest value of the utility and consider the sum of modified utilities for a chosen *discount factor* $0 < \gamma < 1$

$$\sum_{t=1}^{T} \gamma^t u(x_t) \leq u^* \sum_{t=1}^{T} \gamma^t = \gamma u^* \frac{1-\gamma^T}{1-\gamma} \tag{7.6.2}$$

where we used the result for a geometric series. In the limit $T \to \infty$ this means that the summed modified utility $\gamma^t u(x_t)$ is finite. The only modification required to our previous discussion is to include a factor γ in the message definition. Assuming that we are at convergence, we define a value $v(x_t = \mathsf{s})$ dependent only on the state s, and not the time. This means we replace the time-dependent Bellman's value recursion equation (7.5.19) with the stationary equation

$$v(\mathsf{s}) \equiv u(\mathsf{s}) + \gamma \max_{\mathsf{d}} \sum_{\mathsf{s}'} p(x_t = \mathsf{s}'|x_{t-1} = \mathsf{s}, d_{t-1} = \mathsf{d})v(\mathsf{s}') \tag{7.6.3}$$

We then need to solve equation (7.6.3) for the value $v(\mathsf{s})$ for all states s. The optimal decision *policy* when one is in state $x_t = \mathsf{s}$ is then given by

$$\mathsf{d}^*(\mathsf{s}) = \underset{\mathsf{d}}{\text{argmax}} \sum_{\mathsf{s}'} p(x_{t+1} = \mathsf{s}'|x_t = \mathsf{s}, d_t = \mathsf{d})v(\mathsf{s}') \tag{7.6.4}$$

For a deterministic transition p (*i.e.* for each decision d, only one state s' is available), this means that the best decision is the one that takes us to the accessible state with highest value.

Equation(7.6.3) seems straightforward to solve. However, the max operation means that the equations are non-linear in the value v and no closed form solution is available. Two popular techniques for solving equation (7.6.3), are Value and Policy iteration, which we describe below. When the number of states S is very large, approximate solutions are required. Sampling and state-dimension reduction techniques are described in [57].

7.6.1 Value iteration

A naive procedure is to iterate equation (7.6.3) until convergence, assuming some initial guess for the values (say uniform). One can show that this value iteration procedure is guaranteed to converge to a unique optimum[34]. The convergence rate depends somewhat on the discount γ – the smaller γ is, the faster is the convergence. An example of value iteration is given in fig(7.10).

[5]The continuous-time analog has a long history in physics and is called the Hamilton-Jacobi equation and enables one to solve MDPs by message passing, this being a special case of the more general junction tree approach described earlier in section(7.4.2).

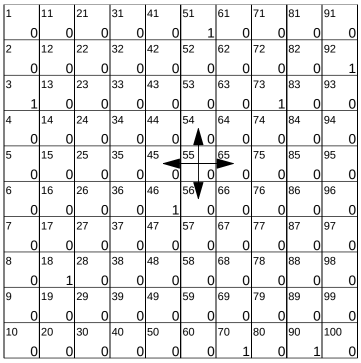


Figure 7.9: States defined on a two dimensional grid. In each square the top left value is the state number, and the bottom right is the utility of being in that state. An 'agent' can move from a state to a neighbouring state, as indicated. The task is to solve this problem such that for any position (state) one knows how to move optimally to maximise the expected utility. This means that we need to move towards the goal states (states with non-zero utility). See `demoMDP`.

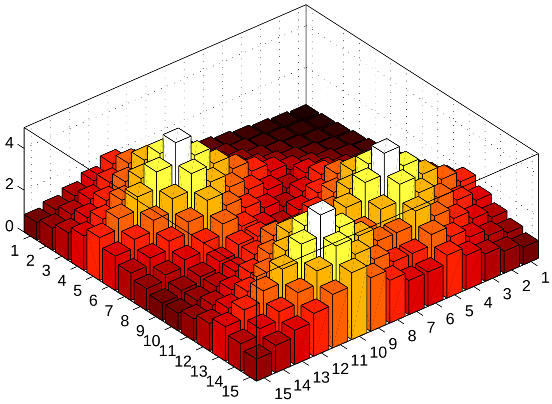


Figure 7.10: Value Iteration on a set of 225 states, corresponding to a 15×15 two dimensional grid. Deterministic transitions are allowed to neighbours on the grid, $\{\mathsf{stay}, \mathsf{left}, \mathsf{right}, \mathsf{up}, \mathsf{down}\}$. There are three goal states, each with utility 1 – all other states have utility 0. Plotted is the value $v(\mathsf{s})$ for $\gamma = 0.9$ after 30 updates of Value Iteration, where the states index a point on the $x - y$ grid. The optimal decision for any state on the grid is to go to the neighbouring state with highest value. See `demoMDP`.

7.6.2 Policy iteration

In policy iteration we first assume we know the optimal decision $\mathsf{d}^*(\mathsf{s})$ for any state s. We may use this in equation (7.6.3) to give

$$v(\mathsf{s}) = u(\mathsf{s}) + \gamma \sum_{\mathsf{s}'} p(x_t = \mathsf{s}' | x_{t-1} = \mathsf{s}, \mathsf{d}^*(\mathsf{s})) v(\mathsf{s}') \tag{7.6.5}$$

The maximisation over d has disappeared since we have assumed we already know the optimal decision for each state s. For fixed $\mathsf{d}^*(\mathsf{s})$, equation (7.6.5) is now linear in the value. Defining the value $\mathbf{v}$ and utility $\mathbf{u}$ vectors and transition matrix $\mathbf{P}$,

$$[\mathbf{v}]_{\mathsf{s}} = v(\mathsf{s}), \qquad [\mathbf{u}]_{\mathsf{s}} = u(\mathsf{s}), \qquad [\mathbf{P}]_{\mathsf{s}',\mathsf{s}} = p(\mathsf{s}'|\mathsf{s}, \mathsf{d}^*(\mathsf{s})) \tag{7.6.6}$$

in matrix notation, equation (7.6.5) becomes

$$\mathbf{v} = \mathbf{u} + \gamma \mathbf{P}^{\mathsf{T}} \mathbf{v} \Leftrightarrow \left(\mathbf{I} - \gamma \mathbf{P}^{\mathsf{T}}\right) \mathbf{v} = \mathbf{u} \Leftrightarrow \mathbf{v} = \left(\mathbf{I} - \gamma \mathbf{P}^{\mathsf{T}}\right)^{-1} \mathbf{u} \tag{7.6.7}$$

These linear equations are readily solved with Gaussian Elimination. Using this, the optimal policy is recomputed using equation (7.6.4). The two steps of solving for the value, and recomputing the policy are iterated until convergence.

In Policy Iteration we guess an initial $\mathsf{d}^*(\mathsf{s})$, then solve the linear equations (7.6.5) for the value, and then recompute the optimal decision. See `demoMDP.m` for a comparison of value and policy iteration, and also an EM style approach which we discuss in the next section.

Example 35 (A grid-world MDP)**.** A set of states defined on a grid, utilities for being in a grid state is given in fig(7.9), for which the agent deterministically moves to a neighbouring grid state at each time step. After initialising the value of each grid state to unity, the converged value for each state is given in fig(7.10). The optimal policy is then given by moving to the neighbouring grid state with highest value.

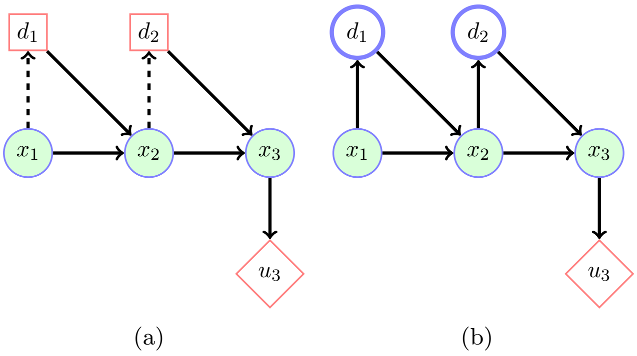


Figure 7.11: **(a)**: A Markov Decision Process. **(b)**: The corresponding probabilistic inference planner.

7.6.3 A curse of dimensionality

Consider the following Tower of Hanoi problem. There are 4 pegs a, b, c, d and 10 disks numbered from 1 to 10. You may move a single disk from one peg to another – however, you are not allowed to put a bigger numbered disk on top of a smaller numbered disk. Starting with all disks on peg a, how can you move them all to peg d in the minimal number of moves?

This would appear to be a straightforward Markov decision process in which the transitions are allowed disk moves. If we use x to represent the state of the disks on the 4 pegs, this has $4^{10} = 1048576$ states (some are equivalent up to permutation of the pegs, which reduces this by a factor of 2). This large number of states renders this naive approach computationally problematic.

Many interesting real-world problems suffer from this large number of states issue so that a naive approach based as we've described is computationally infeasible. Finding efficient exact and also approximate state representations is a key aspect to solving large scale MDPs, see for example [191].

7.7 Probabilistic Inference and Planning

An alternative to the classical MDP solution methods is to make use of the standard methods for training probabilistic models, such as the Expectation-Maximisation algorithm. In order to do so we first need to write the problem of maximising expected utility in a form that is suitable. To do this we first discuss how a MDP can be expressed as the maximisation of a form of Belief Network in which the parameters to be found relate to the policy.

7.7.1 Non-stationary Markov Decision Process

Consider the MDP in fig(7.11a) in which, for simplicity, we assume we know the initial state $x_1 = \mathrm{x}_1$. Our task is then to find the decisions that maximise the expected utility, based on a sequential decision process. The first decision d_1 is given by maximising the expected utility:

$$U(d_1) = \sum_{x_2} p(x_2|\mathrm{x}_1, d_1) \max_{d_2} \sum_{x_3} p(x_3|x_2, d_2) u_3(x_3) \tag{7.7.1}$$

More generally, this utility can be computed efficiently using a standard message passing routine:

$$u_{t\leftarrow t+1}(x_t) \equiv \max_{d_t} \sum_{x_{t+1}} p(x_{t+1}|x_t, d_t) u_{t+1\leftarrow t+2}(x_{t+1}) \tag{7.7.2}$$

where

$$u_{T\leftarrow T+1}(x_T) = u_T(x_T) \tag{7.7.3}$$

7.7.2 Non-stationary Probabilistic Inference Planner

As an alternative to the above MDP description, consider the Belief Network fig(7.11b) in which we have a utility associated with the last time-point[276]. Then the expected utility is given by

$$U(\pi_1, \pi_2) = \sum_{d_1,d_2,x_2,x_3} p(d_1|\mathsf{x}_1, \pi_1)p(x_2|\mathsf{x}_1, d_1)p(d_2|x_2, \pi_2)p(x_3|x_2, d_2)u_3(x_3) \tag{7.7.4}$$

$$= \sum_{d_1} p(d_1|\mathsf{x}_1, \pi_1) \sum_{x_2} p(x_2|\mathsf{x}_1, d_1) \sum_{d_2} p(d_2|x_2, \pi_2) \sum_{x_3} p(x_3|x_2, d_2)u_3(x_3) \tag{7.7.5}$$

Here the terms $p(d_t|x_t, \pi_t)$ are the 'policy distributions' that we wish to learn and π_t are the parameters of the t^{th} policy distribution. Let's assume that we have one per time so that π_t is a function that maps a state x to a probability distribution over decisions. Our interest is to find the policy distributions π_1, π_2 that maximise the expected utility. Since each time-step has its own π_t and for each state $x_2 = \mathsf{x}_2$ we have a separate unconstrained distribution $p(d_2|\mathsf{x}_2, \pi_2)$ to optimise over and we can write

$$\max_{\pi_1,\pi_2} U(\pi_1, \pi_2) = \max_{\pi_1} \sum_{d_1} p(d_1|\mathsf{x}_1, \pi_1) \sum_{x_2} p(x_2|\mathsf{x}_1, d_1) \max_{\pi_2} \sum_{d_2} p(d_2|x_2, \pi_2) \sum_{x_3} p(x_3|x_2, d_2)u_3(x_3) \tag{7.7.6}$$

This shows that provided there are no constraints on the policy distributions (there is a separate one for each timepoint), we are allowed to distribute the maximisations over the individual policies inside the summation.

More generally, for a finite time T one can define messages to solve for the optimal policy distributions

$$u_{t\leftarrow t+1}(x_t) \equiv \max_{\pi_t} \sum_{d_t} p(d_t|x_t, \pi_t) \sum_{x_{t+1}} p(x_{t+1}|x_t, d_t)u_{t+1\leftarrow t+2}(x_{t+1}) \tag{7.7.7}$$

with

$$u_{T+1\leftarrow T}(x_T) = u_T(x_T) \tag{7.7.8}$$

Deterministic policy

For a deterministic policy, only a single state is allowed, so that

$$p(d_t|x_t, \pi_t) = \delta\left(d_t, d_t^*(x_t)\right) \tag{7.7.9}$$

where $d_t^*(x)$ is a policy function that maps a state x to a single decision d. Since we have a separate policy function for each time t equation (7.7.7) reduces to

$$u_{t\leftarrow t+1}(x_t) \equiv \max_{d_t^*(x_t)} \sum_{x_{t+1}} p(x_{t+1}|x_t, d_t^*(x_t))u_{t+1\leftarrow t+2}(x_{t+1}) \tag{7.7.10}$$

which is equivalent to equation (7.7.2).

This shows that solving the MDP is equivalent to maximising a standard expected utility defined in terms of a Belief Network under the assumption that each time point has its own policy distribution, and that this is deterministic.

7.7.3 Stationary planner

If we reconsider our simple example, fig(7.11b) but now constrain the policy distributions to be the same for all time, $p(d_t|x_t, \pi_t) = p(d_t|x_t, \pi)$ (or more succinctly $\pi_t = \pi$), then equation (7.7.5) becomes

$$U(\pi) = \sum_{d_1} p(d_1|\mathsf{x}_1, \pi) \sum_{x_2} p(x_2|\mathsf{x}_1, d_1) \sum_{d_2} p(d_2|x_2, \pi) \sum_{x_3} p(x_3|x_2, d_2)u_3(x_3) \tag{7.7.11}$$

In this case we cannot distribute the maximisation over the policy π over the individual terms of the product. However, computing the expected utility for any given policy π is straightforward, using message passing. One may thus optimise the expected utility using standard numerical optimisation procedures, or alternatively an EM style approach as we discuss below.

A variational training approach

Without loss of generality, we assume that the utility is positive and define a distribution

$$\tilde{p}(d_1, d_2, d_3, x_2, x_3) = \frac{p(d_1|\mathsf{x}_1, \pi)p(x_2|\mathsf{x}_1, d_1)p(d_2|x_2, \pi)p(x_3|x_2, d_2)u_3(x_3)}{\sum_{d_1,d_2,d_3,x_2,x_3} p(d_1|\mathsf{x}_1, \pi)p(x_2|\mathsf{x}_1, d_1)p(d_2|x_2, \pi)p(x_3|x_2, d_2)u_3(x_3)} \tag{7.7.12}$$

Then for any variational distribution $q(d_1, d_2, d_3, x_2, x_3)$,

$$\begin{aligned}\mathrm{KL}(q(d_1, d_2, d_3, x_2, x_3)|\tilde{p}(d_1, d_2, d_3, x_2, x_3)) = &\left\langle \log q(d_1, d_2, d_3, x_2, x_3)\right\rangle_{q(d_1,d_2,d_3,x_2,x_3)} \\ &- \left\langle \log \tilde{p}(d_1, d_2, d_3, x_2, x_3)\right\rangle_{q(d_1,d_2,d_3,x_2,x_3)} \geq 0\end{aligned} \tag{7.7.13}$$

Using the definition of $\tilde{p}(d_1, d_2, d_3, x_2, x_3)$ and the fact that the denominator in equation (7.7.12) is equal to $U(\pi)$ we obtain the bound

$$\begin{aligned}\log U(\pi) \geq &- \left\langle \log q(d_1, d_2, d_3, x_2, x_3)\right\rangle_{q(d_1,d_2,d_3,x_2,x_3)} \\ &+ \left\langle \log p(d_1|\mathsf{x}_1, \pi)p(x_2|\mathsf{x}_1, d_1)p(d_2|x_2, \pi)p(x_3|x_2, d_2)u_3(x_3)\right\rangle_{q(d_1,d_2,d_3,x_2,x_3)}\end{aligned} \tag{7.7.14}$$

This then gives a two-stage EM style procedure:

M-step Isolating the dependencies on π, for a given variational distribution q^{old}, maximising the bound equation (7.7.14) is equivalent to maximising

$$E(\pi) \equiv \left\langle \log p(d_1|\mathsf{x}_1, \pi)\right\rangle_{q^{old}(d_1)} + \left\langle \log p(d_2|x_2, \pi)\right\rangle_{q^{old}(d_2,x_2)} \tag{7.7.15}$$

One then finds a policy π^{new} which maximises $E(\pi)$:

$$\pi^{new} = \underset{\pi}{\mathrm{argmax}}\ E(\pi) \tag{7.7.16}$$

E-step For fixed π the best q is given by the update

$$q^{new} \propto p(d_1|\mathsf{x}_1, \pi)p(x_2|\mathsf{x}_1, d_1)p(d_2|x_2, \pi)p(x_3|x_2, d_2)u_3(x_3) \tag{7.7.17}$$

From this joint distribution, in order to determine the M-step updates, we only require the marginals $q(d_1)$ and $q(d_2, x_2)$, both of which are straightforward to obtain since q is simply a first order Markov Chain in the joint variables x_t, d_t. For example one may write the q-distribution as a simple chain Factor Graph for which marginal inference can be performed readily using the sum-product algorithm.

This procedure is analogous to the standard EM procedure, section(11.2). The usual guarantees therefore carry over so that finding a policy that increases $E(\pi)$ is guaranteed to improve the expected utility.

In complex situations in which, for reasons of storage, the optimal q cannot be used, a structured constrained variational approximation may be applied. In this case, as in generalised EM, only a guaranteed improvement on the lower bound of the expected utility is achieved. Nevertheless, this may be of considerable use in practical situations, for which general techniques of approximate inference may be applied.

The deterministic case

For the special case that the policy π is deterministic, π simply maps each state x to single decision d. Writing this policy map as $d^*(x)$ equation (7.7.11) reduces to

$$U(d^*) = \sum_{x_2} p(x_2|\mathsf{x}_1, d^*(\mathsf{x}_1)) \sum_{x_3} p(x_3|x_2, d^*(x_2))u_3(x_3) \tag{7.7.18}$$

We now define a variational distribution only over x_2, x_3,

$$q(x_2, x_3) \propto p(x_2|\mathsf{x}_1, d^*(\mathsf{x}_1))p(x_3|x_2, d^*(x_2))u_3(x_3) \tag{7.7.19}$$

and the 'energy' term becomes

$$E(d^*) \equiv \left\langle \log p(x_2|\mathsf{x}_1, d^*(\mathsf{x}_1)) \right\rangle_{q(x_2)} + \left\langle \log p(x_3|x_2, d^*(x_2)) \right\rangle_{q(x_2,x_3)} \tag{7.7.20}$$

For a more general problem in which the utility is at the last time point T and no starting state is given we have (for a stationary transition $p(x_{t+1}|x_t, d_t)$)

$$E(\mathsf{d}(\mathsf{s})) \equiv \sum_{\mathsf{s}'} \left(\sum_t q(x_t = \mathsf{s}, x_{t+1} = \mathsf{s}') \right) \log p(x' = \mathsf{s}'|x = \mathsf{s}, d(\mathsf{s}) = \mathsf{d}) \tag{7.7.21}$$

and

$$\mathsf{d}^*(\mathsf{s}) = \underset{\mathsf{d}(\mathsf{s})}{\operatorname{argmax}}\ E(\mathsf{d}(\mathsf{s})) \tag{7.7.22}$$

This shows how to train a stationary MDP using EM in which there is a utility defined only at the last time-point. Below we generalise this to the case of utilities at each time for both the stationary and non-stationary cases.

7.7.4 Utilities at each timestep

Consider a generalisation in which we have an additive utility associated with each time-point.

Non-stationary policy

To help develop the approach, let's look at simply including utilities at times $t = 1, 2$ for the previous example. The expected utility is given by

$$\begin{aligned} U(\pi_1, \pi_2) = & \sum_{d_1,d_2,x_1,x_2,x_3} p(d_1|\mathsf{x}_1, \pi_1) p(x_2|\mathsf{x}_1, d_1) p(d_2|x_2, \pi_2) p(x_3|x_2, d_2) u_3(x_3) \\ & + \sum_{d_1,x_2} p(d_1|\mathsf{x}_1, \pi_1) p(x_2|\mathsf{x}_1, d_1) u_2(x_2) + u_1(\mathsf{x}_1) \end{aligned} \tag{7.7.23}$$

$$= u_1(\mathsf{x}_1) + \sum_{d_1,x_2} p(d_1|\mathsf{x}_1, \pi_1) p(x_2|\mathsf{x}_1) \left(u_2(x_2) + \sum_{d_2,x_3} p(d_2|x_2, \pi_2) p(x_3|x_2, d_2) u_3(x_3) \right) \tag{7.7.24}$$

Defining value messages

$$v_{\pi_2}(x_2) = u_2(x_2) + \sum_{d_2,x_3} p(d_2|x_2, \pi_2) p(x_3|x_2, d_2) u_3(x_3) \tag{7.7.25}$$

and

$$v_{\pi_1}(\mathsf{x}_1) = u_1(\mathsf{x}_1) + \sum_{d_1,x_2} p(d_1|\mathsf{x}_1, \pi_1) p(x_2|\mathsf{x}_1, d_1) v_{\pi_2}(x_2) \tag{7.7.26}$$

$$U(\pi_1, \pi_2) = v_{\pi_1}(\mathsf{x}_1) \tag{7.7.27}$$

For a more general case defined over T timesteps, we have analogously an expected utility $U(\pi_{1:T})$, and our interest is to maximise this expected utility with respect to all the policies

$$\max_{\pi_{1:T}} U(\pi_{1:T}) \tag{7.7.28}$$

As before, since each timestep has its own policy distribution for each state, we may distribute the maximisation using the recursion

$$v_{t,t+1}(x_t) \equiv u_t(x_t) + \max_{\pi_t} \sum_{d_t,x_{t+1}} p(d_t|x_t, \pi_t) p(x_{t+1}|x_t, d_t) v_{t+1,t+2}(x_{t+1}) \tag{7.7.29}$$

with

$$v_{T,T+1}(x_T) \equiv u(x_T) \tag{7.7.30}$$

Stationary deterministic policy

For an MDP the optimal policy is deterministic[264], so that methods which explicitly seek for deterministic policies are of interest. For a stationary deterministic policy π we have the expected utility

$$U(\pi) = \sum_{t=1}^{T} \sum_{x_t} u_t(x_t) \sum_{x_{1:t-1}} \prod_{\tau=1}^{t} p(x_\tau|x_{\tau-1}, d(x_{\tau-1})) \tag{7.7.31}$$

with the convention $p(x_1|x_0, d(x_0)) = p(x_1)$. Viewed as a Factor Graph, this is simply a chain, so that for any policy d, the expected utility can be computed easily. In principle one could then attempt to optimise U with respect to the decisions directly. An alternative is to use an EM style procedure[97]. To do this we need to define a (trans-dimensional) distribution

$$\hat{p}(x_{1:t}, t) = \frac{u_t(x_t)}{Z(d)} \prod_{\tau=1}^{t} p(x_\tau|x_{\tau-1}, d(x_{\tau-1})) \tag{7.7.32}$$

The normalisation constant $Z(d)$ of this distribution is

$$\sum_{t=1}^{T} \sum_{x_{1:t}} u_t(x_t) \prod_{\tau=1}^{t} p(x_\tau|x_{\tau-1}, d(x_{\tau-1})) = \sum_{t=1}^{T} \sum_{x_{1:t}} u_t(x_t) \prod_{\tau=1}^{t} p(x_\tau|x_{\tau-1}, d(x_{\tau-1})) = U(\pi) \tag{7.7.33}$$

If we now define a variational distribution $q(x_{1:t}, t)$, and consider

$$\mathrm{KL}(q(x_{1:t}, t)|\hat{p}(x_{1:t}, t)) \geq 0 \tag{7.7.34}$$

this gives the lower bound

$$\log U(\pi) \geq -H(q(x_{1:T}, t)) + \left\langle \log u_t(x_t) \prod_{\tau=1}^{t} p(x_\tau|x_{\tau-1}, d(x_{\tau-1})) \right\rangle_{q(x_{1:t},t)} \tag{7.7.35}$$

In terms of an EM algorithm, the M-step requires the dependency on d alone, which is

$$E(d) = \sum_{t=1}^{T} \sum_{\tau=1}^{t} \left\langle \log p(x_\tau|x_{\tau-1}, d(x_{\tau-1})) \right\rangle_{q(x_\tau, x_{\tau-1}, t)} \tag{7.7.36}$$

$$= \sum_{t=1}^{T} \sum_{\tau=1}^{t} q(x_\tau = \mathsf{s}', x_{\tau-1} = \mathsf{s}, t) \log p(x_\tau = \mathsf{s}'|x_{\tau-1} = \mathsf{s}, d(x_{\tau-1}) = \mathsf{d}) \tag{7.7.37}$$

$$\tag{7.7.38}$$

For each given state s we now attempt to find the optimal decision d, which corresponds to maximising

$$\hat{E}(\mathsf{d}|\mathsf{s}) = \sum_{\mathsf{s}'} \left\{ \sum_{t=1}^{T} \sum_{\tau=1}^{t} q(x_\tau = \mathsf{s}', x_{\tau-1} = \mathsf{s}, t) \right\} \log p(\mathsf{s}'|\mathsf{s}, \mathsf{d}) \tag{7.7.39}$$

Defining

$$q(\mathsf{s}'|\mathsf{s}) \propto \sum_{t=1}^{T} \sum_{\tau=1}^{t} q(x_\tau = \mathsf{s}', x_{\tau-1} = \mathsf{s}, t) \tag{7.7.40}$$

we see that for given s, up to a constant, $\hat{E}(\mathsf{d}|\mathsf{s})$ is the Kullback-Leibler divergence between $q(\mathsf{s}'|\mathsf{s})$ and $p(\mathsf{s}'|\mathsf{s}, \mathsf{d})$ so that the optimal decision d is given by the index of the distribution $p(\mathsf{s}'|\mathsf{s}, \mathsf{d})$ most closely aligned with $q(\mathsf{s}'|\mathsf{s})$:

$$\mathsf{d}^*(\mathsf{s}) = \underset{\mathsf{d}}{\mathrm{argmin}}\ \mathrm{KL}\big(q(\mathsf{s}'|\mathsf{s})|p(\mathsf{s}'|\mathsf{s}, \mathsf{d})\big) \tag{7.7.41}$$

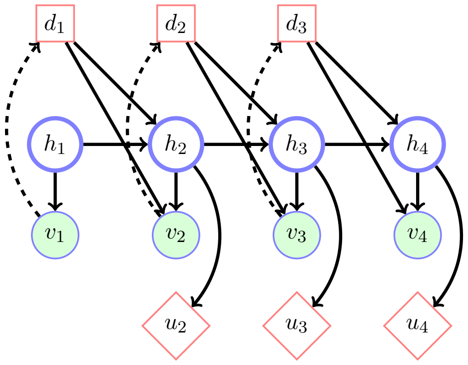


Figure 7.12: An example Partially Observable Markov Decision Process (POMDP). The 'hidden' variables h are never observed. In solving the Influence Diagram we are required to first sum over variables that are never observed; doing so will couple together all past observed variables and decisions that means any decision at time t will depend on all previous decisions. Note that the no-forgetting principle means that we do not need to explicitly write that each decision depends on all previous observations – this is implicitly assumed.

The E-step concerns the computation of the marginal distributions required in the M-step. The optimal q distribution is proportional to $\hat{p}$ evaluated at the previous decision function d:

$$q(x_{1:t}, t) \propto u_t(x_t) \prod_{\tau=1}^{t} p(x_\tau | x_{\tau-1}, d(x_{\tau-1})) \tag{7.7.42}$$

For a constant discount factor γ at each time-step and an otherwise stationary utility[6]

$$u_t(x_t) = \gamma^t u(x_t) \tag{7.7.43}$$

using this

$$q(x_{1:t}, t) \propto \gamma^t u(x_t) \prod_{\tau=1}^{t} p(x_\tau | x_{\tau-1}, d(x_{\tau-1})) \tag{7.7.44}$$

For each t this is a simple Markov chain for which the pairwise transition marginals required for the M-step, equation (7.7.40) are straightforward. This requires inference in a series of Markov models of different lengths. This can be done efficiently using a single forward and backward pass[276]. See `MDPemDeterministicPolicy.m` which also deals with the more general case of utilities dependent on the decision(action) as well as the state.

Note that this EM algorithm formally fails in the case of a deterministic environment (the transition $p(x_t|x_{t-1}, d_{t-1})$ is deterministic) – see exercise(76) for an explanation and exercise(77) for a possible resolution.

7.8 Further Topics

7.8.1 Partially Observable MDPs

In a *POMDP* there are states that are not observed. This seemingly innocuous extension of the MDP case can lead however to computational difficulties. Let's consider the situation in fig(7.12), and attempt to compute the optimal expected utility based on the sequence of summations and maximisations:

$$U = \max_{d_1} \sum_{v_2} \max_{d_2} \sum_{v_3} \max_{d_3} \sum_{h_{1:3}} p(h_4|h_3, d_3) p(v_3|h_3) p(h_3|h_2, d_2) p(v_2|h_2) p(h_2|h_1, d_1) p(v_1|h_1) p(h_1)$$

The sum over the hidden variables $h_{1:3}$ couples all the decisions and observations, meaning that we no longer have a simple chain structure for the remaining maximisations. For a POMDP of length t, this leads to intractable problem with complexity exponential in t. An alternative view is to recognise that all past decisions and observations $v_{1:t}, d_{1:t-1}$, can be summarised in terms of a belief in the current latent state, $p(h_t|v_{1:t}, d_{1:t-1})$. This suggests that instead of having an actual state, as in the MDP case, we need

[6]In the standard MDP framework it is more common to define $u_t(x_t) = \gamma^{t-1} u(x_t)$ so that for comparison with the standard Policy/Value routines one needs to divide the expected utility by γ.

to use a *distribution* over states to represent our current knowledge. One can therefore write down an effective MDP albeit over belief distributions, as opposed to finite states. Approximate techniques are required to solve the resulting 'infinite' state MDPs, and the reader is referred to more specialised texts for a study of approximation procedures. See for example [146, 149].

7.8.2 Restricted utility functions

An alternative to solving MDPs is to consider restricted utilities such that the policy can be found easily. Recently efficient solutions have been developed for classes of MDPs with utilities restricted to Kullback-Leibler divergences [150, 275].

7.8.3 Reinforcement learning

Reinforcement Learning deals mainly with stationary Markov Decision Processes. The added twist is that the transition $p(\mathsf{s}'|\mathsf{s}, \mathsf{d})$ (and possibly the utility) is unknown. Initially an 'agent' begins to explore the set of states and utilities (rewards) associated with taking decisions. The set of accessible states and their rewards populates as the agent traverses its environment. Consider for example a maze problem with a given start and goal state, though with an unknown maze structure. The task is to get from the start to the goal in the minimum number of moves on the maze. Clearly there is a balance required between curiosity and acting to maximise the expected reward. If we are too curious (don't take optimal decisions given the currently available information about the maze structure) and continue exploring the possible maze routes, this may be bad. On the other hand, if we don't explore the possible maze states, we might never realise that there is a much more optimal short-cut to follow than that based on our current knowledge. This exploration-exploitation tradeoff is central to the difficulties of RL. See [264] for an extensive discussion of reinforcement learning.

For a given set of environment data $\mathcal{X}$ (observed transitions and utilities) one aspect of RL problem can be considered as finding the policy that maximises expected reward, given only a prior belief about the environment and observed decisions and states. If we assume we know the utility function but not the transition, we may write

$$U(\pi|\mathcal{X}) = \langle U(\pi|\theta)\rangle_{p(\theta|\mathcal{X})} \tag{7.8.1}$$

where θ represents the environment state transition,

$$\theta = p(x_{t+1}|x_t, d_t) \tag{7.8.2}$$

Given a set of observed states and decisions,

$$p(\theta|\mathcal{X}) \propto p(\mathcal{X}|\theta)p(\theta) \tag{7.8.3}$$

where $p(\theta)$ is a prior on the transition. Similar techniques to the EM style training can be carried through in this case as well[76, 276]. Rather than the policy being a function of the state and the environment θ, optimally one needs to consider a policy $p(d_t|x_t, b(\theta))$ as a function of the state and the belief in the environment. This means that, for example, if the belief in the environment has high entropy, the agent can recognise this and explicitly carry out decisions/actions to explore the environment. A further complication in RL is that the data collected $\mathcal{X}$ depends on the policy π. If we write t for an 'episode' in which policy π_t is followed and data $\mathcal{X}_t$ collected, then the utility of the policy π given all the historical information is

$$U(\pi|\pi_{1:t}, \mathcal{X}_{1:t}) = \langle U(\pi|\theta)\rangle_{p(\theta|\mathcal{X}_{1:t}, \pi_{1:t})} \tag{7.8.4}$$

Depending on the priors on the environment, and also on how long each episode is, we will have different posteriors for the environment parameters. If we then set

$$\pi_{t+1} = \underset{\pi}{\operatorname{argmax}}\ U(\pi|\pi_{1:t}, \mathcal{X}_{1:t}) \tag{7.8.5}$$

this affects the data we collect at the next episode $\mathcal{X}_{t+1}$. In this way, the trajectory of policies $\pi_1, \pi_2, \ldots$ can be very different depending on these episode lengths and priors.

7.9 Code

7.9.1 Sum/Max under a partial order

`maxsumpot.m`: Generalised elimination operation according to a partial ordering
`sumpotID.m`: Sum/max an ID with probability and decision potentials
`demoDecParty.m`: Demo of summing/maxing an ID

7.9.2 Junction trees for influence diagrams

There is no need to specify the information links provided that a partial ordering is given. In the code `jtreeID.m` no check is made that the partial ordering is consistent with the influence diagram. In this case, the first step of the junction tree formulation in section(7.4.2) is not required. Also the moralisation and removal of utility nodes is easily dealt with by defining utility potentials and including them in the moralisation process.

The strong triangulation is found by a simple variable elimination scheme which seeks to eliminate a variable with the least number of neighbours, provided that the variable may be eliminated according to the specified partial ordering.

The junction tree is constructed based only on the elimination clique sequence $\mathcal{C}_1, \ldots, \mathcal{C}_N$. obtained from the triangulation routine. The junction tree is then obtained by connecting a clique $\mathcal{C}_\mathcal{I}$ to the first clique $j > i$ that is connected to this clique. Clique $\mathcal{C}_i$ is then eliminated from the graph. In this manner a junction tree of connected cliques is formed. We do not require the separators for the influence diagram absorption since these can be computed and discarded on the fly.

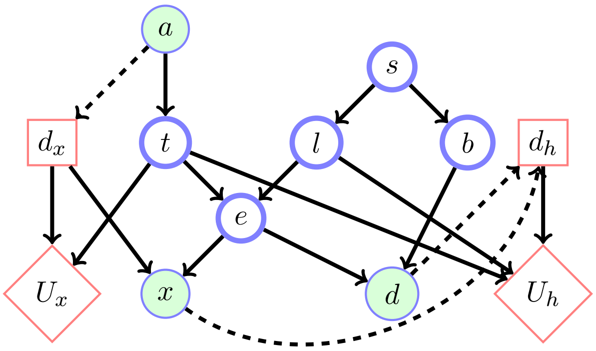
Note that the code only computes messages from the leaves to the root of the junction tree, which is sufficient for taking decisions at the root. If one desires an optimal decision at a non-root, one would need to absorb probabilities into a clique which contains the decision required. These extra forward probability absorptions are required because information about any unobserved variables can be affected by decisions and observations in the past. This extra forward probability schedule is not given in the code and left as an exercise for the interested reader.
`jtreeID.m`: Junction Tree for an Influence Diagram
`absorptionID.m`: Absorption on an Influence Diagram
`triangulatePorder.m`: Triangulation based on a partial ordering
`demoDecPhD.m`: Demo for utility of Doing PhD and Startup

7.9.3 Party-Friend example

The code below implements the Party-Friend example in the text. To deal with the asymmetry the $Visit$ utility is zero if $Party$ is in state yes.
`demoDecPartyFriend.m`: Demo for Party-Friend

7.9.4 Chest Clinic with Decisions

The table for the Chest Clinic Decision network, fig(7.13) is taken from exercise(25), see [116, 68]. There is a slight modification however to the $p(x|e)$ table. If an x-ray is taken, then information about x is available. However, if the decision is not to take an x-ray no information about x is available. This is a form of asymmetry. A straightforward approach in this case is to make d_x a parent of the x variable and



s = Smoking
x = Positive X-ray
d = Dyspnea (Shortness of breath)
e = Either Tuberculosis or Lung Cancer
t = Tuberculosis
l = Lung Cancer
b = Bronchitis
a = Visited Asia
d_h = Hospitalise?
d_x = Take X-ray?

Figure 7.13: Influence Diagram for the 'Chest Clinic' Decision example.

set the distribution of x to be uninformative if $d_x = \mathsf{fa}$.

$$\begin{array}{ll} p(a=\mathsf{tr})=0.01 & p(s=\mathsf{tr})=0.5 \\ p(t=\mathsf{tr}|a=\mathsf{tr})=0.05 & p(t=\mathsf{tr}|a=\mathsf{fa})=0.01 \\ p(l=\mathsf{tr}|s=\mathsf{tr})=0.1 & p(l=\mathsf{tr}|s=\mathsf{fa})=0.01 \\ p(b=\mathsf{tr}|s=\mathsf{tr})=0.6 & p(b=\mathsf{tr}|s=\mathsf{fa})=0.3 \\ p(x=\mathsf{tr}|e=\mathsf{tr},d_x=\mathsf{tr})=0.98 & p(x=\mathsf{tr}|e=\mathsf{fa},d_x=\mathsf{tr})=0.05 \\ p(x=\mathsf{tr}|e=\mathsf{tr},d_x=\mathsf{fa})=0.5 & p(x=\mathsf{tr}|e=\mathsf{fa},d_x=\mathsf{fa})=0.5 \\ p(d=\mathsf{tr}|e=\mathsf{tr},b=\mathsf{tr})=0.9 & p(d=\mathsf{tr}|e=\mathsf{tr},b=\mathsf{fa})=0.3 \\ p(d=\mathsf{tr}|e=\mathsf{fa},b=\mathsf{tr})=0.2 & p(d=\mathsf{tr}|e=\mathsf{fa},b=\mathsf{fa})=0.1 \end{array} \tag{7.9.1}$$

The two utilities are designed to reflect the costs and benefits of taking an x-ray and hospitalising a patient:

$$\begin{array}{lll|l} d_h=\mathsf{tr} & t=\mathsf{tr} & l=\mathsf{tr} & 180 \\ d_h=\mathsf{tr} & t=\mathsf{tr} & l=\mathsf{fa} & 120 \\ d_h=\mathsf{tr} & t=\mathsf{fa} & l=\mathsf{tr} & 160 \\ d_h=\mathsf{tr} & t=\mathsf{fa} & l=\mathsf{fa} & 15 \\ d_h=\mathsf{fa} & t=\mathsf{tr} & l=\mathsf{tr} & 2 \\ d_h=\mathsf{fa} & t=\mathsf{tr} & l=\mathsf{fa} & 4 \\ d_h=\mathsf{fa} & t=\mathsf{fa} & l=\mathsf{tr} & 0 \\ d_h=\mathsf{fa} & t=\mathsf{fa} & l=\mathsf{fa} & 40 \end{array} \tag{7.9.2}$$

$$\begin{array}{ll|l} d_x=\mathsf{tr} & t=\mathsf{tr} & 0 \\ d_x=\mathsf{tr} & t=\mathsf{fa} & 1 \\ d_x=\mathsf{fa} & t=\mathsf{tr} & 10 \\ d_x=\mathsf{fa} & t=\mathsf{fa} & 10 \end{array} \tag{7.9.3}$$

We assume that we know whether or not the patient has been to Asia, before deciding on taking an x-ray. The partial ordering is then

$$a \prec d_x \prec \{d, x\} \prec d_h \prec \{b, e, l, s, t\} \tag{7.9.4}$$

The demo `demoDecAsia.m` produces the results:

```
utility table:
asia = yes takexray = yes  49.976202
asia = no  takexray = yes  46.989441
asia = yes takexray = no   48.433043
asia = no  takexray = no   47.460900
```

which shows that optimally one should take an x-ray only if the patient has been to Asia.

`demoDecAsia.m`: Junction Tree Influence Diagram demo

7.9.5 Markov Decision Processes

In `demoMDP.m` we consider a simple two dimensional grid in which an 'agent' can move to a grid square either above, below, left, right of the current square, or stay in the current square. We defined goal states (grid squares) that have high utility, with others having zero utility.
`demoMDPclean.m`: Demo of Value and Policy Iteration for a simple MDP
`MDPsolve.m`: MDP solver using Value or Policy Iteration

MDP solver using EM and assuming a deterministic policy

The following code[7] is not fully documented in the text, although the method is reasonably straightforward and follows that described in section(7.7.3). The inference is carried out using a simple $\alpha-\beta$ style recursion. This could also be implemented using the general Factor Graph code, but was coded explicitly for reasons of speed. The code also handles the more general case of utilities (rewards) as a function of both the state and the action $u(x_t, d_t)$.
`MDPemDeterministicPolicy.m`: MDP solver using EM and assuming a deterministic policy
`EMqTranMarginal.m`: Marginal information required for the transition term of the energy
`EMqUtilMarginal.m`: Marginal information required for the utility term of the energy
`EMTotalBetaMessage.m`: Backward information required for inference in the MDP
`EMminimizeKL.m`: Find the optimal decision
`EMvalueTable.m`: Return the expected value of the policy

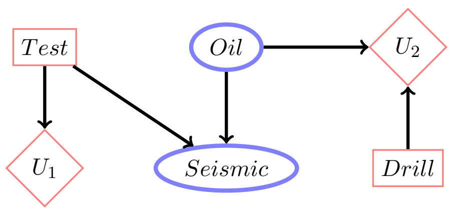
7.10 Exercises

Exercise 68. *You play a game in which you have a probability p of winning. If you win the game you gain an amount £S and if you lose the game you lose an amount £S. Show that the expected gain from playing the game is £$(2p-1)S$.*

Exercise 69. *It is suggested that the utility of money is based, not on the amount, but rather how much we have relative to other peoples. Assume a distribution $p(i)$, $i = 1, \ldots, 10$ of incomes using a histogram with 10 bins, each bin representing an income range. Use a histogram to roughly reflect the distribution of incomes in society, namely that most incomes are around the average with few very wealthy and few extremely poor people. Now define the utility of an income x as the chance that income x will be higher than a randomly chosen income y (under the distribution you defined) and relate this to the cumulative distribution of p. Write a program to compute this probability and plot the resulting utility as a function of income. Now repeat the coin tossing bet of section(7.1.1) so that if one wins the bet one's new income will be placed in the top histogram bin, whilst if one loses one's new income is in the lowest bin. Compare the optimal expect utility decisions under the situations in which one's original income is (i) average, and (ii) much higher than average.*

Exercise 70.

Derive a partial ordering for the ID on the right, and explain how this ID differs from that of fig(7.5).



Exercise 71. *This question follows closely* `demoMDP.m`*, and represents a problem in which a pilot wishes to land an airplane.*

The matrix $U(x, y)$ in the file `airplane.mat` *contains the utilities of being in position x, y and is a very crude model of a runway and taxiing area.*

[7]Thanks to Tom Furmston for coding this.

The airspace is represented by an 18×15 *grid (*$Gx = 18, Gy = 15$ *in the notation employed in* `demoMDP.m`*). The matrix* $U(8,4) = 2$ *represents that position* $(8,4)$ *is the desired parking bay of the airplane (the vertical height of the airplane is not taken in to account). The positive values in* U *represent runway and areas where the airplane is allowed. Zero utilities represent neutral positions. The negative values represent unfavourable positions for the airplane. By examining the matrix* U *you will see that the airplane should preferably not veer off the runway, and also should avoid two small villages close to the airport.*

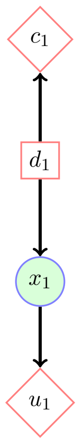
At each timestep the plane can perform one of the following actions stay up down left right*:*
For stay*, the airplane stays in the same* x, y *position.*
For up*, the airplane moves to the* $x, y+1$ *position.*
For down*, the airplane moves to the* $x, y-1$ *position.*
For left*, the airplane moves to the* $x-1, y$ *position.*
For right*, the airplane moves to the* $x+1, y$ *position.*

A move that takes the airplane out of the airspace is not allowed.

1. *The airplane begins in at point* $x = 1, y = 13$*. Assuming that an action deterministically results in the intended grid move, find the optimal* x_t, y_t *sequence for times* $t = 1, \ldots,$ *for the position of the aircraft.*

2. *The pilot tells you that there is a fault with the airplane. When the pilot instructs the plane to go* right *with probability 0.1 it actually goes* up *(provided this remains in the airspace). Assuming again that the airplane begins at point* $x = 1, y = 13$*, return the optimal* x_t, y_t *sequence for times* $t = 1, \ldots,$ *for the position of the aircraft.*

Exercise 72.



The influence diagram depicted describes the first stage of a game. The decision variable $dom(d_1) = \{\textsf{play}, \textsf{not play}\}$*, indicates the decision to either play the first stage or not. If you decide to play, there is a cost* $c_1(\textsf{play}) = C_1$*, but no cost otherwise,* $c_1(\textsf{no play}) = 0$*. The variable* x_1 *describes if you win or lose the game,* $dom(x_1) = \{\textsf{win}, \textsf{lose}\}$*, with probabilities:*

$$p(x_1 = \textsf{win}|d_1 = \textsf{play}) = p_1, \qquad p(x_1 = \textsf{win}|d_1 = \textsf{no play}) = 0 \tag{7.10.1}$$

The utility of winning/losing is

$$u_1(x_1 = \textsf{win}) = W_1, \qquad u_1(x_1 = \textsf{lose}) = 0 \tag{7.10.2}$$

Show that the expected utility gain of playing this game is

$$U(d_1 = \textsf{play}) = p_1 W_1 - C_1 \tag{7.10.3}$$

Exercise 73. *Exercise(72) above describes the first stage of a new two-stage game. If you win the first stage* $x_1 = \textsf{win}$*, you have to make a decision* d_2 *as to whether or not play in the second stage* $dom(d_2) = \{\textsf{play}, \textsf{not play}\}$*. If you do not win the first stage, you cannot enter the second stage.*

If you decide to play the second stage, you win with probability p_2*:*

$$p(x_2 = \textsf{win}|x_1 = \textsf{win}, d_2 = \textsf{play}) = p_2 \tag{7.10.4}$$

If you decide not to play the second stage there is no chance to win:

$$p(x_2 = \textsf{win}|x_1 = \textsf{win}, d_2 = \textsf{not play}) = 0 \tag{7.10.5}$$

The cost of playing the second stage is

$$c_2(d_2 = \textsf{play}) = C_2, \qquad c_2(d_2 = \textsf{no play}) = 0 \tag{7.10.6}$$

and the utility of winning/losing the second stage is

$$u_2(x_2 = \textsf{win}) = W_2, \qquad u_2(x_2 = \textsf{lose}) = 0 \tag{7.10.7}$$

1. *Draw an Influence Diagram that describes this two-stage game.*
2. *A gambler needs to decide if he should even enter the first stage of this two-stage game. Show that based on taking the optimal future decision d_2 the expected utility based on the first decision is:*

$$U(d_1 = \textsf{play}) = \begin{cases} p_1(p_2W_2 - C_2) + p_1W_1 - C_1 & \text{if} \;\; p_2W_2 - C_2 \geq 0 \\ p_1W_1 - C_1 & \text{if} \;\; p_2W_2 - C_2 \leq 0 \end{cases} \tag{7.10.8}$$

Exercise 74. *You have £B in your bank account. You are asked if you would like to participate in a bet in which, if you win, your bank account will become £W. However, if you lose, your bank account will contain only £L. You win the bet with probability p_w.*

1. *Assuming that the utility is given by the number of pounds in your bank account, write down a formula for the expected utility of taking the bet, $U(\textsf{bet})$ and also the expected utility of not taking the bet, $U(\textsf{no bet})$.*
2. *The above situation can be formulated differently. If you win the bet you gain £$(W - B)$. If you lose the bet you lose £$(B - L)$. Compute the expected amount of money you gain if you bet $U_{gain}(\textsf{bet})$ and if you don't bet $U_{gain}(\textsf{no bet})$.*
3. *Show that $U(\textsf{bet}) - U(\textsf{no bet}) = U_{gain}(\textsf{bet}) - U_{gain}(\textsf{no bet})$.*

Exercise 75. *Consider the Party-Friend scenario, example(30). An alternative is to replace the link from Party to U_{visit} by an information link from Party to Visit with the constraint that Visit can be in state* yes *only if Party is in state* no.

1. *Explain how this constraint can be achieved by including an additional additive term to the utilities and modify* `demoDecPartyFriend.m` *accordingly to demonstrate this.*
2. *For the case in which utilities are all positive, explain how the same constraint can be achieved using a multiplicative factor.*

Exercise 76. *Consider an objective*

$$F(\theta) = \sum_x U(x)p(x|\theta) \tag{7.10.9}$$

for a positive function $U(x)$ and that our task is to maximise F with respect to θ. An Expectation-Maximisation style bounding approach (see section(11.2)) can be derived by defining the auxiliary distribution

$$\tilde{p}(x|\theta) = \frac{U(x)p(x|\theta)}{F(\theta)} \tag{7.10.10}$$

so that by considering $KL(q(x)|\tilde{p}(x))$ for some variational distribution $q(x)$ we obtain the bound

$$\log F(\theta) \geq -\langle \log q(x) \rangle_{q(x)} + \langle \log U(x) \rangle_{q(x)} + \langle \log p(x|\theta) \rangle_{q(x)} \tag{7.10.11}$$

The M-step states that the optimal q distribution is given by

$$q(x) = \tilde{p}(x|\theta_{old}) \tag{7.10.12}$$

At the E-step of the algorithm the new parameters θ_{new} are given by maximising the 'energy' term

$$\theta_{new} = \underset{\theta}{\operatorname{argmax}} \; \langle \log p(x|\theta) \rangle_{\tilde{p}(x|\theta_{old})} \tag{7.10.13}$$

Show that for a deterministic distribution

$$p(x|\theta) = \delta(x, f(\theta)) \tag{7.10.14}$$

the E-step fails, giving $\theta_{new} = \theta_{old}$.

Exercise 77. *Consider an objective*

$$F_\epsilon(\theta) = \sum_x U(x) p_\epsilon(x|\theta) \tag{7.10.15}$$

for a positive function $U(x)$ *and*

$$p_\epsilon(x|\theta) = (1-\epsilon)\delta\left(x, f(\theta)\right) + \epsilon n(x), \quad 0 \leq \epsilon \leq 1 \tag{7.10.16}$$

and an arbitrary distribution $n(x)$*. Our task is to maximise* F *with respect to* θ*. As the previous exercise showed, if we attempt an EM algorithm in the limit of a deterministic model* $\epsilon = 0$*, then no-updating occurs and the EM algorithm fails to find* θ *that optimises* $F_0(\theta)$*.*

Show that

$$F_\epsilon(\theta) = (1-\epsilon)F_0(\theta) + \epsilon \sum_x n(x)U(x) \tag{7.10.17}$$

and hence

$$F_\epsilon(\theta_{new}) - F_\epsilon(\theta_{old}) = (1-\epsilon)\left[F_0(\theta_{new}) - F_0(\theta_{old}\right] \tag{7.10.18}$$

Show that if for $\epsilon > 0$ *we can find a* θ_{new} *such that* $F_\epsilon(\theta_{new}) > F_\epsilon(\theta_{old})$*, then necessarily* $F_0(\theta_{new}) > F_0(\theta_{old})$*.*

Using this result, derive an EM-style algorithm that guarantees to increase $F_\epsilon(\theta)$ *(unless we are already at the optimum) for* $\epsilon > 0$ *and therefore guarantees to increase* $F_0(\theta)$*. Hint: use*

$$\tilde{p}(x|\theta) = \frac{U(x)p_\epsilon(x|\theta)}{F_\epsilon(\theta)} \tag{7.10.19}$$

and consider

$$KL(q(x)|\tilde{p}(x)) \tag{7.10.20}$$

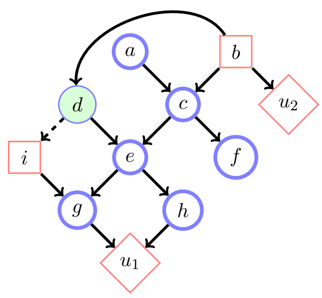
for some variational distribution $q(x)$*.*

Exercise 78. *The file* `IDjensen.mat` *contains probability and utility tables for the influence diagram of fig(7.8a). Using* BRMLtoolbox*, write a program that returns the maximal expected utility for this ID using a strong junction tree approach, and check the result by explicit summation and maximisation. Similarly, your program should output the maximal expected utility for both states of* d_1*, and check that the computation using the strong junction tree agrees with the result from explicit elimination summation and maximisation.*

Exercise 79. *For a POMDP, explain the structure of the strong junction tree, and relate this to the complexity of inference in the POMDP.*

Exercise 80.

(i) Define a partial order for the ID diagram depicted. (ii) Draw a (strong) junction tree for this ID.



Part II

Learning in Probabilistic Models

CHAPTER 8

Statistics for Machine Learning:

8.1 Distributions

Definition 54 (Cumulative distribution function)**.** For a univariate distribution $p(x)$, the CDF is defined as

$$cdf(y) \equiv p(x \leq y) = \langle \mathbb{I}[x \leq y] \rangle_{p(x)} \tag{8.1.1}$$

For an unbounded domain, $cdf(-\infty) = 0$ and $cdf(\infty) = 1$.

8.2 Summarising distributions

Definition 55 (Mode)**.** The mode x_* of a distribution $p(x)$ is the state of x at which the distribution takes its highest value, $x_* = \underset{x}{\operatorname{argmax}}\, p(x)$. A distribution could have more than one node (be multi-modal). A widespread abuse of terminology is to refer to any isolated local maximum of $p(x)$ to be a mode.

Definition 56 (Averages and Expectation)**.**

$$\langle f(x) \rangle_{p(x)} \tag{8.2.1}$$

denotes the average or expectation of $f(x)$ with respect to the distribution $p(x)$. A common alternative notation is

$$\mathbb{E}(x) \tag{8.2.2}$$

When the context is clear, one may drop the notational dependency on $p(x)$. The notation

$$\langle f(x)|y \rangle \tag{8.2.3}$$

is shorthand for the average of $f(x)$ conditioned on knowing the state of variable y, *i.e.* the average of $f(x)$ with respect to the distribution $p(x|y)$.

An advantage of the expectation notations is that they hold whether the distribution is over continuous or discrete variables. In the discrete case

$$\langle f(x) \rangle \equiv \sum_{\mathsf{x}} f(x = \mathsf{x}) p(x = \mathsf{x}) \tag{8.2.4}$$

and for continuous variables,

$$\langle f(x) \rangle \equiv \int_{-\infty}^{\infty} f(x) p(x) dx \tag{8.2.5}$$

The reader might wonder what $\langle x \rangle$ means when x is discrete. For example, if $\mathrm{dom}(x) = \{\mathsf{apple}, \mathsf{orange}, \mathsf{pear}\}$, with associated probabilities $p(x)$ for each of the states, what does $\langle x \rangle$ refer to? Clearly, $\langle f(x) \rangle$ makes sense if $f(x = \mathsf{x})$ maps the state x to a numerical value. For example $f(x = \mathsf{apple}) = 1$, $f(x = \mathsf{orange}) = 2$, $f(x = \mathsf{pear}) = 3$ for which $\langle f(x) \rangle$ is meaningful. Unless the states of the discrete variable are associated with a numerical value, then $\langle x \rangle$ has no meaning.

Definition 57 (Moments)**.** The k^{th} moment of a distribution is given by the average of x^k under the distribution:

$$\left\langle x^k \right\rangle_{p(x)} \tag{8.2.6}$$

For $k = 1$, we have the mean, typically denoted by μ,

$$\mu \equiv \langle x \rangle \tag{8.2.7}$$

Definition 58 (Variance and Correlation)**.**

$$\sigma^2 \equiv \left\langle (x - \langle x \rangle)^2 \right\rangle_{p(x)} \tag{8.2.8}$$

The square root of the variance, σ is called the *standard deviation*. The notation $\mathrm{var}(x)$ is also used to emphasise for which variable the variance is computed. The reader may show that an equivalent expression is

$$\sigma^2 \equiv \left\langle x^2 \right\rangle - \langle x \rangle^2 \tag{8.2.9}$$

For a multivariate distribution the matrix with elements

$$\Sigma_{ij} = \langle (x_i - \mu_i)(x_j - \mu_j) \rangle \tag{8.2.10}$$

where $\mu_i = \langle x_i \rangle$ is called the *covariance matrix*. The diagonal entries of the covariance matrix contain the variance of each variable. An equivalent expression is

$$\Sigma_{ij} = \langle x_i x_j \rangle - \langle x_i \rangle \langle x_j \rangle \tag{8.2.11}$$

The *correlation matrix* has elements

$$\rho_{ij} = \left\langle \frac{(x_i - \mu_i)}{\sigma_i} \frac{(x_j - \mu_j)}{\sigma_j} \right\rangle \tag{8.2.12}$$

where σ_i is the deviation of variable x_i. The correlation is a normalised form of the covariance so that each element is bounded $-1 \leq \rho_{ij} \leq 1$.

For independent variables x_i and x_j, $x_i \perp\!\!\!\perp x_j | \emptyset$ the covariance Σ_{ij} is zero. Similarly independent variables have zero correlation – they are 'uncorrelated'. Note however that the converse is not generally true – two variables can be uncorrelated but dependent. A special case is for when x_i and x_j are Gaussian distributed then independence is equivalent to being uncorrelated, see exercise(82).

Definition 59 (Skewness and Kurtosis)**.** The skewness is a measure of the asymmetry of a distribution:

$$\gamma_1 \equiv \frac{\left\langle (x - \langle x \rangle)^3 \right\rangle_{p(x)}}{\sigma^3} \tag{8.2.13}$$

where σ^2 is the variance of x with respect to $p(x)$. A positive skewness means the distribution has a heavy tail to the right. Similarly, a negative skewness means the distribution has a heavy tail to the left.

The kurtosis is a measure of how peaked around the mean a distribution is:

$$\gamma_2 \equiv \frac{\left\langle (x - \langle x \rangle)^4 \right\rangle_{p(x)}}{\sigma^4} - 3 \tag{8.2.14}$$

A distribution with positive kurtosis has more mass around its mean than would a Gaussian with the same mean and variance. These are also called *super Gaussian*. Similarly a negative kurtosis (*sub Gaussian*) distribution has less mass around its mean than the corresponding Gaussian. The kurtosis is defined such that a Gaussian has zero kurtosis (which accounts for the -3 term in the definition).

Definition 60 (Empirical Distribution)**.** For a set of datapoints $x^1, \ldots, x^N$, which are states of a random variable x, the empirical distribution has probability mass distributed evenly over the datapoints, and zero elsewhere.

For a discrete variable x the empirical distribution is

$$p(x) = \frac{1}{N} \sum_{n=1}^{N} \mathbb{I}\left[x = x^n\right] \tag{8.2.15}$$

where N is the number of datapoints.

For a continuous distribution we have

$$p(x) = \frac{1}{N} \sum_{n=1}^{N} \delta\left(x - x^n\right) \tag{8.2.16}$$

where $\delta(x)$ is the Dirac Delta function.

The sample mean of the datapoints is given by the

$$\hat{\mu} = \frac{1}{N} \sum_{n=1}^{N} x^n \tag{8.2.17}$$

and the sample variance is given by the

$$\hat{\sigma}^2 = \frac{1}{N} \sum_{n=1}^{N} (x^n - \hat{\mu})^2 \tag{8.2.18}$$

For vectors the sample mean vector has elements

$$\hat{\mu}_i = \frac{1}{N}\sum_{n=1}^{N} x_i^n \tag{8.2.19}$$

and sample covariance matrix has elements

$$\hat{\Sigma}_{ij} = \frac{1}{N}\sum_{n=1}^{N} \left(x_i^n - \hat{\mu}_i\right)\left(x_j^n - \hat{\mu}_j\right) \tag{8.2.20}$$

Definition 61 (Delta function)**.** For continuous x,

$$\delta(x - x_0) \tag{8.2.21}$$

is zero everywhere expect at x_0, where there is a spike. $\int_{-\infty}^{\infty} \delta(x - x_0)dx = 1$ and

$$\int_{-\infty}^{\infty} \delta(x - x_0) f(x) dx = f(x_0) \tag{8.2.22}$$

One can view the Delta function as an infinitely narrow Gaussian: $\delta(x - x_0) = \lim_{\sigma \to 0} \mathcal{N}\left(x|x_0, \sigma^2\right)$.

The *Kronecker delta*,

$$\delta_{x,x_0} \tag{8.2.23}$$

is similarly zero everywhere, except when $x = x_0$ when $\delta_{x_0,x_0} = 1$. The Kronecker delta is equivalent to $\delta_{x,x_0} = \mathbb{I}\left[x = x_0\right]$. We use the expression $\delta\left(x, x_0\right)$ to denote either the Dirac or Kronecker delta, depending on the context.

8.2.1 Estimator Bias

Definition 62 (Unbiased estimator)**.** Given data $\mathcal{X} = x^1, \ldots, x^N$, from a distribution $p(x|\theta)$ we can use the data $\mathcal{X}$ to estimate the parameter θ that was used to generate the data. The estimator is a function of the data, which we write $\hat{\theta}(\mathcal{X})$. For an *unbiased estimator*

$$\left\langle \hat{\theta}(\mathcal{X}) \right\rangle_{p(\mathcal{X}|\theta)} = \theta \tag{8.2.24}$$

More generally, one can consider any estimating function $\hat{\psi}(\mathcal{X})$ of data. This is an unbiased estimator of a quantity ψ if $\left\langle \hat{\psi}(\mathcal{X}) \right\rangle_{p(x)} = \psi$.

Figure 8.1: Empirical distribution over a discrete variable with 4 states. The empirical samples consist of n samples at each of states $1, 2, 4$ and $2n$ samples at state 3 where $n > 0$. On normalising this gives a distribution with values $0.2, 0.2, 0.4, 0.2$ over the 4 states.

A classical example for estimator bias are those of the mean and variance. Let

$$\hat{\mu}(\mathcal{X}) = \frac{1}{N}\sum_{n=1}^{N} x^n \tag{8.2.25}$$

This is an unbiased estimator of the mean $\langle x \rangle_{p(x)}$ since

$$\langle \hat{\mu}(\mathcal{X}) \rangle_{p(x)} = \frac{1}{N}\sum_{n=1}^{N} \langle x^n \rangle_{p(x)} = \frac{1}{N} N \langle x \rangle_{p(x)} = \langle x \rangle_{p(x)} \tag{8.2.26}$$

On the other hand, consider the estimator of the variance,

$$\hat{\sigma}^2(\mathcal{X}) = \frac{1}{N}\sum_{n=1}^{N} (x^n - \hat{\mu}(\mathcal{X}))^2 \tag{8.2.27}$$

This is biased since (omitting a few lines of algebra)

$$\left\langle \hat{\sigma}^2(\mathcal{X}) \right\rangle_{p(x)} = \frac{1}{N}\sum_{n=1}^{N} \left\langle (x^n - \hat{\mu}(\mathcal{X}))^2 \right\rangle = \frac{N-1}{N}\sigma^2 \tag{8.2.28}$$

8.3 Discrete Distributions

Definition 63 (Bernoulli Distribution)**.** The Bernoulli distribution concerns a discrete binary variable x, with $\mathrm{dom}(x) = \{0, 1\}$. The states are not merely symbolic, but real values 0 and 1.

$$p(x = 1) = \theta \tag{8.3.1}$$

From normalisation, it follows that $p(x = 0) = 1 - \theta$. From this

$$\langle x \rangle = 0 \times p(x = 0) + 1 \times p(x = 1) = \theta \tag{8.3.2}$$

The variance is given by $\mathrm{var}(x) = \theta\,(1 - \theta)$.

Definition 64 (Categorical Distribution)**.** The categorical distribution generalises the Bernoulli distribution to more than two (symbolic) states. For a discrete variable x, with symbolic states $\mathrm{dom}(x) = \{1, \ldots, C\}$,

$$p(x = c) = \theta_c, \qquad \sum_c \theta_c = 1 \tag{8.3.3}$$

The Dirichlet is conjugate to the categorical distribution.

Definition 65 (Binomial Distribution)**.** The Binomial describes the distribution of a discrete two-state variable x, with $\mathrm{dom}(x) = \{1, 0\}$ where the states are symbolic. The probability that in n Bernoulli Trials (independent samples), k 'success' states 1 will be observed is

$$p(y = k|\theta) = \binom{n}{k} \theta^k\,(1 - \theta)^{n-k} \tag{8.3.4}$$

where $\binom{n}{k}$ is the binomial coefficient. The mean and variance are

$$\langle y \rangle = n\theta, \qquad \text{var}(x) = n\theta\,(1-\theta) \tag{8.3.5}$$

The Beta distribution is the conjugate prior for the Binomial distribution.

Definition 66 (Multinomial Distribution)**.** Consider a multi-state variable x, with $\text{dom}(x) = \{1, \ldots, \mathsf{K}\}$, with corresponding state probabilities $\theta_1, \ldots, \theta_K$. We then draw n samples from this distribution. The probability of observing the state 1 y_1 times, state 2 y_2 times, ..., state K y_K times in the n samples is

$$p(y|\theta) = \frac{n!}{y_1! \ldots, y_K!} \prod_{i=1}^{n} \theta_i^{y_i} \tag{8.3.6}$$

where $n = \sum_{i=1}^{n} y_i$.

$$\langle y_i \rangle = n\theta_i, \quad \text{var}(y_i) = n\theta_i\,(1-\theta_i)\,, \quad \langle y_i y_j \rangle - \langle y_i \rangle \langle y_j \rangle = -n\theta_i\theta_j \;\; (i \neq j) \tag{8.3.7}$$

The Dirichlet distribution is the conjugate prior for the multinomial distribution.

Definition 67 (Poisson Distribution)**.** The Poisson distribution can be used to model situations in which the expected number of events scales with the length of the interval within which the events can occur. If λ is the expected number of events per unit interval, then the distribution of the number of events x within an interval $t\lambda$ is

$$p(x = k|\lambda) = \frac{1}{k!} e^{-\lambda t} (\lambda t)^k\,, \qquad k = 0, 1, 2, \ldots \tag{8.3.8}$$

For a unit length interval ($t = 1$),

$$\langle x \rangle = \lambda, \quad \text{var}(x) = \lambda \tag{8.3.9}$$

The Poisson can be derived as a limiting case of a Binomial distribution in which the success probability scales as $\theta = \lambda/n$, in the limit $n \to \infty$.

8.4 Continuous Distributions

8.4.1 Bounded distributions

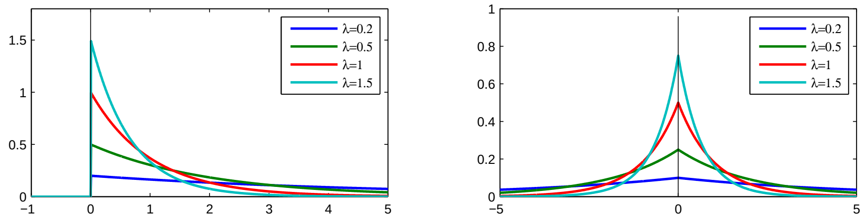
Definition 68 (Uniform distribution)**.** For a variable x, the distribution is uniform if $p(x) = \mathsf{const}$. over the domain of the variable.

Definition 69 (Exponential Distribution)**.** For $x \geq 0$,

$$p(x|\lambda) \equiv \lambda e^{-\lambda x} \tag{8.4.1}$$

One can show that for rate λ

$$\langle x \rangle = \frac{1}{\lambda}, \qquad \text{var}(x) = \frac{1}{\lambda^2} \tag{8.4.2}$$



(a) (b)

Figure 8.2: **(a)**: Exponential distribution. **(b)**: Laplace (double exponential) distribution.

The alternative parameterisation $b = 1/\lambda$ is called the scale.

Definition 70 (Gamma Distribution)**.**

$$p(x) = \frac{1}{\beta\Gamma(\gamma)}\left(\frac{x}{\beta}\right)^{\alpha-1} e^{-\frac{x}{\beta}}, \qquad x \geq 0, \alpha > 0, \beta > 0 \tag{8.4.3}$$

α is called the shape parameter, β is the scale parameter and

$$\Gamma(a) = \int_0^\infty t^{a-1}e^{-t}dt \tag{8.4.4}$$

The parameters are related to the mean and variance through

$$\alpha = \left(\frac{\mu}{s}\right)^2, \qquad \beta = \frac{s^2}{\mu} \tag{8.4.5}$$

where μ is the mean of the distribution and s is the standard deviation.

The mode is given by $(\alpha - 1)\,\beta$, for $\alpha \geq 1$.

Definition 71 (Beta Distribution)**.**

$$p(x|\alpha, \beta) = B\left(x|\alpha, \beta\right) = \frac{1}{B(\alpha, \beta)}x^{\alpha-1}\left(1-x\right)^{\beta-1}, \quad 0 \leq x \leq 1 \tag{8.4.6}$$

where the beta function is defined as

$$B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)} \tag{8.4.7}$$

and $\Gamma(x)$ is the gamma function. Note that the distribution can be flipped by interchanging x for $1 - x$, which is equivalent to interchanging α and β.

The mean is given by

$$\langle x \rangle = \frac{\alpha}{\alpha+\beta}, \qquad \text{var}(x) = \frac{\alpha\beta}{\left(\alpha+\beta\right)^2\left(\alpha+\beta+1\right)} \tag{8.4.8}$$

8.4.2 Unbounded distributions

Definition 72 (Laplace (Double Exponential) Distribution)**.**

$$p(x|\lambda) \equiv \lambda e^{-\frac{1}{b}|x-\mu|} \tag{8.4.9}$$

For scale b

$$\langle x \rangle = \mu, \qquad \text{var}(x) = 2b^2 \tag{8.4.10}$$

Univariate Gaussian Distribution

The Gaussian distribution is an important distribution in science. It's technical description is given in definition(73).

Definition 73 (Univariate Gaussian Distribution)**.**

$$p(x|\mu, \sigma^2) = \mathcal{N}\left(x|\mu, \sigma^2\right) \equiv \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2} \tag{8.4.11}$$

where μ is the mean of the distribution, and σ^2 the variance. This is also called the *normal distribution*.

One can show that the parameters indeed correspond to

$$\mu = \langle x \rangle_{\mathcal{N}(x|\mu,\sigma^2)}, \qquad \sigma^2 = \left\langle (x-\mu)^2 \right\rangle_{\mathcal{N}(x|\mu,\sigma^2)} \tag{8.4.12}$$

For $\mu = 0$ and $\sigma = 1$, the Gaussian is called the *standard normal distribution*.

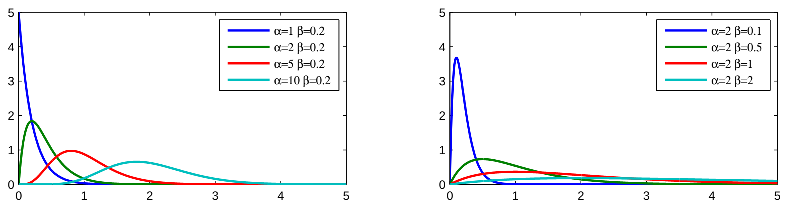
Definition 74 (Student's t-distribution)**.**

$$p(x|\mu, \lambda, \nu) = \frac{\Gamma(\frac{\nu+1}{2})}{\Gamma(\frac{\nu}{2})} \left(\frac{\lambda}{\nu\pi}\right)^{\frac{1}{2}} \left[1 + \frac{\lambda (x-\mu)^2}{\nu}\right]^{-\frac{\nu+1}{2}} \tag{8.4.13}$$

where μ is the mean, ν the degrees of freedom, and λ scales the distribution. The variance is given by

$$\text{var}(x) = \frac{\nu}{\lambda(\nu - 2)}, \quad \text{for } \nu > 2 \tag{8.4.14}$$

For $\nu \to \infty$ the distribution tends to a Gaussian with mean μ and variance $1/\lambda$. As ν decreases the tails of the distribution become fatter.



(a) (b)

Figure 8.3: **(a)**: Gamma distribution with varying β for fixed α. **(b)**: Gamma distribution with varying α for fixed β.

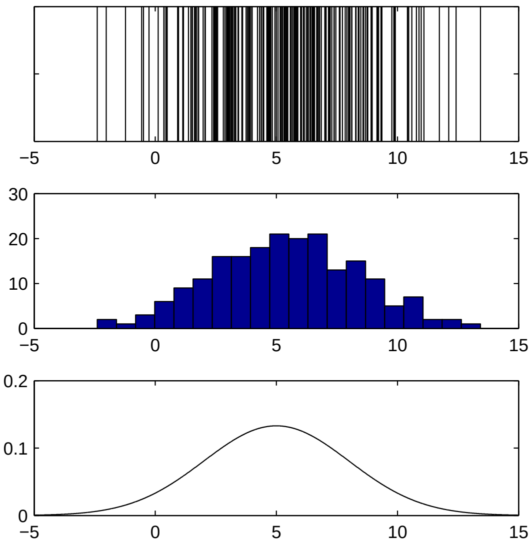


Figure 8.4: Top: 200 datapoints $x^1, \ldots, x^{200}$ drawn from a Gaussian distribution. Each vertical line denotes a datapoint at the corresponding x value on the horizontal axis. Middle: Histogram using 10 equally spaced bins of the datapoints. Bottom: Gaussian distribution $\mathcal{N}(x|\mu = 5, \sigma = 3)$ from which the datapoints were drawn. In the limit of an infinite amount of data, and limitingly small bin size, the normalised histogram tends to the Gaussian probability density function.

The t-distribution can be derived from a *scaled mixture*

$$p(x|a,b) = \int_{\tau=0}^{\infty} \mathcal{N}\left(x|\mu, \tau^{-1}\right) Gam\left(\tau|a,b\right) d\tau \tag{8.4.15}$$

$$= \left(\frac{\tau}{2\pi}\right)^{\frac{1}{2}} \int_{\tau=0}^{\infty} e^{-\frac{\tau}{2}(x-\mu)^2} b^a e^{-b\tau} \tau^{a-1} \frac{1}{\Gamma(a)} d\tau \tag{8.4.16}$$

$$= \frac{b^a}{\Gamma(a)} \frac{\Gamma(a+\frac{1}{2})}{\sqrt{2\pi}} \frac{1}{\left(b + \frac{1}{2}(x-\mu)^2\right)^{a+\frac{1}{2}}} \tag{8.4.17}$$

It is conventional to reparameterise using $\nu = 2a$ and $\lambda = a/b$.

Definition 75 (Inverse Gamma distribution)**.**

$$InvGam\left(x|\alpha, \beta\right) = \frac{\beta^\alpha}{\Gamma(\alpha)} \frac{1}{x^{\alpha+1}} e^{-\beta/x} \tag{8.4.18}$$

This has mean $\beta/(\alpha-1)$ for $\alpha > 1$ and variance $\frac{\beta^2}{(\alpha-1)^2(\alpha-2)}$ for $\alpha > 2$.

8.5 Multivariate distributions

Definition 76 (Dirichlet Distribution)**.** The Dirichlet distribution is a distribution on probability distributions:

$$p(\alpha) = \frac{1}{Z(\mathbf{u})} \delta\left(\sum_{i=1}^{Q} \alpha_i - 1\right) \prod_{q=1}^{Q} \alpha_q^{u_q-1} \tag{8.5.1}$$

where

$$Z(\mathbf{u}) = \frac{\prod_{q=1}^{Q} \Gamma(u_q)}{\Gamma\left(\sum_{q=1}^{Q} u_q\right)} \tag{8.5.2}$$

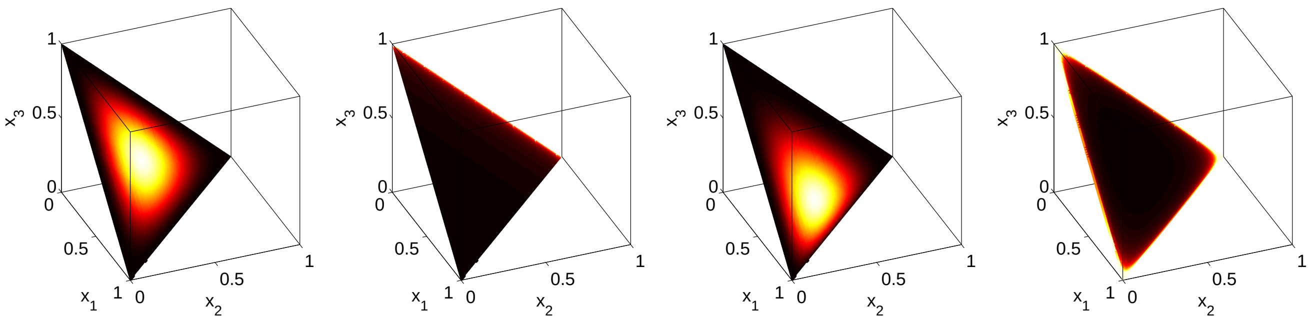


Figure 8.5: Dirichlet distribution with parameter (u_1, u_2, u_3) displayed on the simplex $x_1, x_2, x_3 \geq 0, x_1 + x_2 + x_3 = 1$. Black denotes low probability and white high probability. **(a)**: $(3, 3, 3)$ **(b)**: $(0.1, 1, 1)$. **(c)**: $(4, 3, 2)$. **(d)**: $(0.05, 0.05, 0.05)$.

It is conventional to denote the distribution as

$$\text{Dirichlet}\left(\alpha|\mathbf{u}\right) \tag{8.5.3}$$

The parameter $\mathbf{u}$ controls how strongly the mass of the distribution is pushed to the corners of the simplex. Setting $u_q = 1$ for all q corresponds to a uniform distribution, fig(8.5). In the binary case $Q = 2$, this is equivalent to a Beta distribution.

The product of two Dirichlet distributions is

$$\text{Dirichlet}\left(\theta|\mathbf{u}_1\right)\text{Dirichlet}\left(\theta|\mathbf{u}_2\right) = \text{Dirichlet}\left(\theta|\mathbf{u}_1 + \mathbf{u}_2\right) \tag{8.5.4}$$

Marginal of a Dirichlet:

$$\int_{\theta_j} \text{Dirichlet}\left(\theta|\mathbf{u}\right) = \text{Dirichlet}\left(\theta_{\setminus j}|\mathbf{u}_{\setminus j}\right) \tag{8.5.5}$$

The marginal of a single component θ_i is a Beta distribution:

$$p(\theta_i) = B\left(\theta_i|u_i, \sum_{j\neq i} u_j\right) \tag{8.5.6}$$

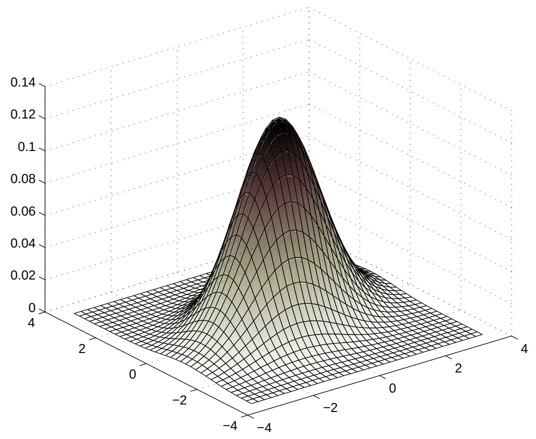
8.6 Multivariate Gaussian

The multivariate Gaussian plays a central role throughout this book and as such we discuss its properties in some detail.

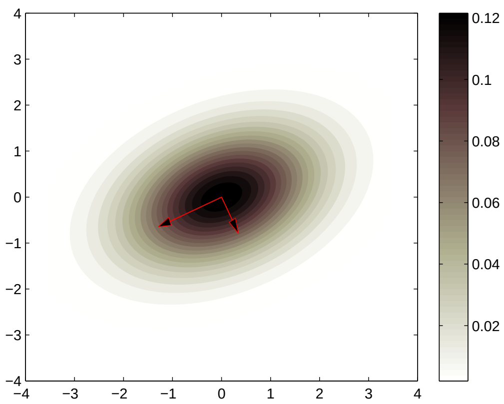
Definition 77 (Multivariate Gaussian Distribution)**.**

$$p(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \mathcal{N}\left(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}\right) \equiv \frac{1}{\sqrt{\det\left(2\pi\boldsymbol{\Sigma}\right)}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})} \tag{8.6.1}$$

where $\boldsymbol{\mu}$ is the mean vector of the distribution, and $\boldsymbol{\Sigma}$ the covariance matrix. The inverse covariance $\boldsymbol{\Sigma}^{-1}$ is called the *precision*.



(a)



(b)

Figure 8.6: **(a)**: Bivariate Gaussian with mean $(0, 0)$ and covariance $[1, 0.5; 0.5, 1.75]$. Plotted on the vertical axis is the probability density value $p(x)$. **(b)**: Probability density contours for the same bivariate Gaussian. Plotted are the unit eigenvectors scaled by the square root of their eigenvalues, $\sqrt{\lambda_i}$.

One may show

$$\boldsymbol{\mu} = \langle \mathbf{x} \rangle_{\mathcal{N}(\mathbf{x}|\boldsymbol{\mu},\boldsymbol{\Sigma})}, \qquad \boldsymbol{\Sigma} = \left\langle (\mathbf{x} - \boldsymbol{\mu}) (\mathbf{x} - \boldsymbol{\mu})^{\mathsf{T}} \right\rangle_{\mathcal{N}(\mathbf{x}|\boldsymbol{\mu},\boldsymbol{\Sigma})} \tag{8.6.2}$$

The multivariate Gaussian is given in definition(77). Note that $\det(\rho\mathbf{M}) = \rho^D \det(\mathbf{M})$, where $\mathbf{M}$ is a $D \times D$ matrix, which explains the dimension independent notation in the normalisation constant of definition(77).

The multivariate Gaussian is widely used and it is instructive to understand the geometric picture. This can be obtained by view the distribution in a different co-ordinate system. First we use that every real symmetric matrix $D \times D$ has an eigen-decomposition

$$\boldsymbol{\Sigma} = \mathbf{E}\boldsymbol{\Lambda}\mathbf{E}^{\mathsf{T}} \tag{8.6.3}$$

where $\mathbf{E}^{\mathsf{T}}\mathbf{E} = \mathbf{I}$ and $\boldsymbol{\Lambda} = \text{diag}(\lambda_1, \ldots, \lambda_D)$. In the case of a covariance matrix, all the eigenvalues λ_i are positive. This means that one can use the transformation

$$\mathbf{y} = \boldsymbol{\Lambda}^{\frac{1}{2}}\mathbf{E}^{\mathsf{T}}(\mathbf{x} - \boldsymbol{\mu}) \tag{8.6.4}$$

so that

$$(\mathbf{x} - \boldsymbol{\mu})^{\mathsf{T}} \boldsymbol{\Sigma} (\mathbf{x} - \boldsymbol{\mu}) = (\mathbf{x} - \boldsymbol{\mu})^{\mathsf{T}} \mathbf{E}\boldsymbol{\Lambda}\mathbf{E}^{\mathsf{T}} (\mathbf{x} - \boldsymbol{\mu}) = \mathbf{y}^{\mathsf{T}}\mathbf{y} \tag{8.6.5}$$

Under this transformation, the multivariate Gaussian reduces to a product of D univariate zero-mean unit variance Gaussians (since the Jacobian of the transformation is a constant). This means that we can view a multivariate Gaussian as a shifted, scaled and rotated version of an isotropic Gaussian in which the centre is given by the mean, the rotation by the eigenvectors, and the scaling by the square root of the eigenvalues, as depicted in fig(8.6b).

Isotropic means 'same under rotation'. For any isotropic distribution, contours of equal probability are spherical around the origin.

Some useful properties of the Gaussian are as follows:

Definition 78 (Partitioned Gaussian)**.** For a distribution $\mathcal{N}(\mathbf{z}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ defined jointly over two vectors $\mathbf{x}$ and $\mathbf{y}$ of potentially differing dimensions,

$$\mathbf{z} = \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} \tag{8.6.6}$$

with corresponding mean and partitioned covariance

$$\boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_x \\ \boldsymbol{\mu}_y \end{pmatrix} \qquad \boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}_{xx} & \boldsymbol{\Sigma}_{xy} \\ \boldsymbol{\Sigma}_{yx} & \boldsymbol{\Sigma}_{yy} \end{pmatrix} \tag{8.6.7}$$

where $\boldsymbol{\Sigma}_{yx} \equiv \boldsymbol{\Sigma}_{xy}^{\mathsf{T}}$. The marginal distribution is given by

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_{xx}) \tag{8.6.8}$$

and conditional

$$p(\mathbf{x}|\mathbf{y}) = \mathcal{N}\left(\mathbf{x}\middle|\boldsymbol{\mu}_x + \boldsymbol{\Sigma}_{xy}\boldsymbol{\Sigma}_{yy}^{-1}\left(\mathbf{y} - \boldsymbol{\mu}_y\right), \boldsymbol{\Sigma}_{xx} - \boldsymbol{\Sigma}_{xy}\boldsymbol{\Sigma}_{yy}^{-1}\boldsymbol{\Sigma}_{yx}\right) \tag{8.6.9}$$

Definition 79 (Product of two Gaussians)**.** The product of two Gaussians is another Gaussian, with a multiplicative factor, exercise(115):

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)\,\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})\,\frac{\exp\left(-\frac{1}{2}\left(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\right)^{\mathsf{T}}\mathbf{S}^{-1}\left(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\right)\right)}{\sqrt{\det\left(2\pi\mathbf{S}\right)}} \tag{8.6.10}$$

where $\mathbf{S} \equiv \boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2$ and the mean and covariance are given by

$$\boldsymbol{\mu} = \boldsymbol{\Sigma}_1\mathbf{S}^{-1}\boldsymbol{\mu}_2 + \boldsymbol{\Sigma}_2\mathbf{S}^{-1}\boldsymbol{\mu}_1 \qquad \boldsymbol{\Sigma} = \boldsymbol{\Sigma}_1\mathbf{S}^{-1}\boldsymbol{\Sigma}_2 \tag{8.6.11}$$

Definition 80 (Linear Transform of a Gaussian)**.** For the linear transformation

$$\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{b} \tag{8.6.12}$$

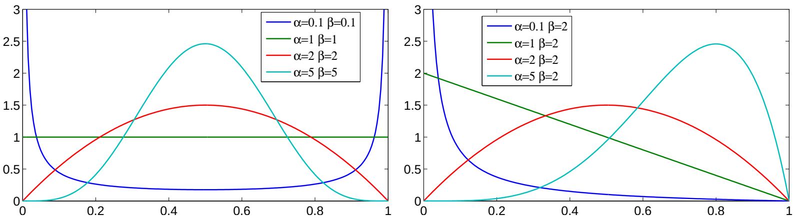
where $\mathbf{x} \sim \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$,

$$\mathbf{y} \sim \mathcal{N}\left(\mathbf{y}|\mathbf{A}\boldsymbol{\mu} + \mathbf{b}, \mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^{\mathsf{T}}\right) \tag{8.6.13}$$

Definition 81 (Entropy of a Gaussian)**.** The differential entropy of a multivariate Gaussian $p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is

$$H(\mathbf{x}) \equiv -\left\langle \log p(\mathbf{x})\right\rangle_{p(\mathbf{x})} = \frac{1}{2}\log\det\left(2\pi\boldsymbol{\Sigma}\right) + \frac{D}{2} \tag{8.6.14}$$

where $D = \dim\mathbf{x}$. Note that the entropy is independent of the mean $\boldsymbol{\mu}$.



(a) (b)

Figure 8.7: Beta distribution. The parameters α and β can also be witting in terms of the mean and variance, leading to an alternative parameterisation, see exercise(95).

8.6.1 Conditioning as system reversal

For a joint distribution $p(\mathbf{x}, \mathbf{y})$, consider the conditional $p(\mathbf{x}|\mathbf{y})$. The statistics of $p(x|y)$ can be obtained using a linear system of the form

$$\mathbf{x} = \overleftarrow{\mathbf{A}}\mathbf{y} + \overleftarrow{\boldsymbol{\eta}} \tag{8.6.15}$$

where

$$\overleftarrow{\boldsymbol{\eta}} \sim \mathcal{N}\left(\overleftarrow{\boldsymbol{\eta}} \middle| \overleftarrow{\boldsymbol{\mu}}, \overleftarrow{\boldsymbol{\Sigma}}\right) \tag{8.6.16}$$

and this reversed noise is uncorrelated with $\mathbf{y}$.

To show this, we need to make the statistics of $\mathbf{x}$ under this linear system match those given by the conditioning operation, (8.6.9). The mean of the linear system is given by

$$\boldsymbol{\mu}_x = \overleftarrow{\mathbf{A}}\boldsymbol{\mu}_y + \overleftarrow{\boldsymbol{\mu}} \tag{8.6.17}$$

and the covariances by (note that covariance of $\mathbf{y}$ remains unaffected by the system reversal)

$$\boldsymbol{\Sigma}_{xx} = \overleftarrow{\mathbf{A}}\boldsymbol{\Sigma}_{yy}\overleftarrow{\mathbf{A}}^{\mathsf{T}} + \overleftarrow{\boldsymbol{\Sigma}} \tag{8.6.18}$$

$$\boldsymbol{\Sigma}_{xy} = \overleftarrow{\mathbf{A}}\boldsymbol{\Sigma}_{yy} \tag{8.6.19}$$

From equation (8.6.19) we have

$$\overleftarrow{\mathbf{A}} = \boldsymbol{\Sigma}_{xy}\boldsymbol{\Sigma}_{yy}^{-1} \tag{8.6.20}$$

which using in equation (8.6.18) gives

$$\overleftarrow{\boldsymbol{\Sigma}} = \boldsymbol{\Sigma}_{xx} - \overleftarrow{\mathbf{A}}\boldsymbol{\Sigma}_{yy}\overleftarrow{\mathbf{A}}^{\mathsf{T}} = \boldsymbol{\Sigma}_{xx} - \boldsymbol{\Sigma}_{xy}\boldsymbol{\Sigma}_{yy}^{-1}\boldsymbol{\Sigma}_{yx} \tag{8.6.21}$$

Using equation (8.6.17) we similarly obtain

$$\overleftarrow{\boldsymbol{\mu}} = \boldsymbol{\mu}_x - \overleftarrow{\mathbf{A}}\boldsymbol{\mu}_y = \boldsymbol{\mu}_x - \boldsymbol{\Sigma}_{xy}\boldsymbol{\Sigma}_{yy}^{-1}\boldsymbol{\mu}_y \tag{8.6.22}$$

This means that we can write an explicit linear system of the form equation (8.6.15) where the parameters are given in terms of the statistics of the original system. These results are just a restatement of the conditioning results but shows how it may be interpreted as a linear system. This is useful in deriving results in inference with Linear Dynamical Systems.

8.6.2 Completing the square

A useful technique in manipulating Gaussians is completing the square. For example, the expression

$$e^{-\frac{1}{2}\mathbf{x}^{\mathsf{T}}\mathbf{A}\mathbf{x}+\mathbf{b}^{\mathsf{T}}\mathbf{x}} \tag{8.6.23}$$

can be transformed as follows. First we complete the square:

$$\frac{1}{2}\mathbf{x}^{\mathsf{T}}\mathbf{A}\mathbf{x} - \mathbf{b}^{\mathsf{T}}\mathbf{x} = \frac{1}{2}\left(\mathbf{x} - \mathbf{A}^{-1}\mathbf{b}\right)^{\mathsf{T}}\mathbf{A}\left(\mathbf{x} - \mathbf{A}^{-1}\mathbf{b}\right) - \frac{1}{2}\mathbf{b}^{\mathsf{T}}\mathbf{A}^{-1}\mathbf{b} \tag{8.6.24}$$

Hence

$$e^{-\frac{1}{2}\mathbf{x}^{\mathsf{T}}\mathbf{A}\mathbf{x}-\mathbf{b}^{\mathsf{T}}\mathbf{x}} = \mathcal{N}\left(\mathbf{x}|\mathbf{A}^{-1}\mathbf{b}, \mathbf{A}^{-1}\right)\sqrt{\det\left(2\pi\mathbf{A}^{-1}\right)}e^{\frac{1}{2}\mathbf{b}^{\mathsf{T}}\mathbf{A}^{-1}\mathbf{b}} \tag{8.6.25}$$

From this one can derive

$$\int e^{-\frac{1}{2}\mathbf{x}^{\mathsf{T}}\mathbf{A}\mathbf{x}+\mathbf{b}^{\mathsf{T}}\mathbf{x}}d\mathbf{x} = \sqrt{\det\left(2\pi\mathbf{A}^{-1}\right)}e^{\frac{1}{2}\mathbf{b}^{\mathsf{T}}\mathbf{A}^{-1}\mathbf{b}} \tag{8.6.26}$$

8.6.3 Gaussian Propagation

Let $\mathbf{y}$ be linearly related to $\mathbf{x}$ through

$$\mathbf{y} = \mathbf{M}\mathbf{x} + \boldsymbol{\eta} \tag{8.6.27}$$

where $\boldsymbol{\eta} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, and $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x)$.

Then the marginal $p(\mathbf{y}) = \int_{\mathbf{x}} p(\mathbf{y}|\mathbf{x})p(\mathbf{x})$ is a Gaussian

$$p(\mathbf{y}) = \mathcal{N}\left(\mathbf{y}\middle|\mathbf{M}\boldsymbol{\mu}_x + \boldsymbol{\mu}, \mathbf{M}\boldsymbol{\Sigma}_x\mathbf{M}^\mathsf{T} + \boldsymbol{\Sigma}\right) \tag{8.6.28}$$

8.6.4 Whitening and centering

For a set of data $\mathbf{x}^1, \ldots, \mathbf{x}^N$, with $\dim \mathbf{x}^n = D$, we can transform this data to $\mathbf{y}^1, \ldots, \mathbf{y}^N$ with zero mean using *centering*:

$$\mathbf{y}^n = \mathbf{x}^n - \mathbf{m} \tag{8.6.29}$$

where the mean $\mathbf{m}$ of the data is given by

$$\mathbf{m} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{x}^n \tag{8.6.30}$$

Furthermore, we can transform to a values $\mathbf{z}^1, \ldots, \mathbf{z}^N$ that have zero mean and unit covariance using *whitening*

$$\mathbf{z}^n = \mathbf{S}^{-\frac{1}{2}}\left(\mathbf{x}^n - \mathbf{m}\right) \tag{8.6.31}$$

where the covariance $\mathbf{S}$ of the data is given by

$$\mathbf{S} = \frac{1}{N}\sum_{n=1}^{N} \left(\mathbf{x}^n - \mathbf{m}\right)\left(\mathbf{x}^n - \mathbf{m}\right)^\mathsf{T} \tag{8.6.32}$$

An equivalent approach is to compute the SVD decomposition of the matrix of centered datapoints

$$\mathbf{U}\mathbf{S}\mathbf{V}^\mathsf{T} = \mathbf{Y}, \qquad \mathbf{Y} = \left[\mathbf{y}^1, \ldots, \mathbf{y}^N\right] \tag{8.6.33}$$

then

$$\mathbf{Z} = \sqrt{N}\mathrm{diag}\left(1/S_{1,1}, \ldots, 1/S_{D,D}\right)\mathbf{U}^\mathsf{T}\mathbf{Y} \tag{8.6.34}$$

has zero mean and unit covariance, see exercise(112).

8.6.5 Maximum Likelihood training

Given a set of training data $\mathcal{X} = \left\{\mathbf{x}^1, \ldots, \mathbf{x}^N\right\}$, drawn from a Gaussian $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ with unknown mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$, how can we find these parameters? Assuming the data are drawn i.i.d. the log likelihood is

$$L(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \equiv \sum_{n=1}^{N} \log p(x|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = -\frac{1}{2}\sum_{n=1}^{N} \left(\mathbf{x}^n - \boldsymbol{\mu}\right)^\mathsf{T} \boldsymbol{\Sigma}^{-1}\left(\mathbf{x}^n - \boldsymbol{\mu}\right) - \frac{N}{2}\log\det\left(2\pi\boldsymbol{\Sigma}\right) \tag{8.6.35}$$

Optimal $\boldsymbol{\mu}$

Taking the partial derivative with respect to the vector $\boldsymbol{\mu}$ we obtain the vector derivative

$$\nabla_{\boldsymbol{\mu}} L(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} \boldsymbol{\Sigma}^{-1} \left(\mathbf{x}^n - \boldsymbol{\mu}\right) \tag{8.6.36}$$

Equating to zero gives that at the optimum of the log likelihood,

$$\sum_{n=1}^{N} \boldsymbol{\Sigma}^{-1}\mathbf{x}^n = N\boldsymbol{\mu}\boldsymbol{\Sigma}^{-1} \tag{8.6.37}$$

and therefore, optimally

$$\boldsymbol{\mu} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{x}^n \tag{8.6.38}$$

Optimal $\boldsymbol{\Sigma}$

The derivative of L with respect to the matrix $\boldsymbol{\Sigma}$ requires more work. It is convenient to isolate the dependence on the covariance, and also parameterise using the inverse covariance, $\boldsymbol{\Sigma}^{-1}$,

$$L = -\frac{1}{2}\text{trace}\left(\boldsymbol{\Sigma}^{-1} \underbrace{\sum_{n=1}^{N} \left(\mathbf{x}^n - \boldsymbol{\mu}\right)\left(\mathbf{x}^n - \boldsymbol{\mu}\right)^{\mathsf{T}}}_{\equiv \mathbf{M}}\right) + \frac{N}{2}\log\det\left(2\pi\boldsymbol{\Sigma}^{-1}\right) \tag{8.6.39}$$

Using $\mathbf{M} = \mathbf{M}^{\mathsf{T}}$, we obtain

$$\frac{\partial}{\partial \boldsymbol{\Sigma}^{-1}} L = -\frac{1}{2}\mathbf{M} + \frac{N}{2}\boldsymbol{\Sigma} \tag{8.6.40}$$

Equating the derivative to the zero matrix and solving for $\boldsymbol{\Sigma}$ gives

$$\boldsymbol{\Sigma} = \frac{1}{N}\sum_{n=1}^{N} \left(\mathbf{x}^n - \boldsymbol{\mu}\right)\left(\mathbf{x}^n - \boldsymbol{\mu}\right)^{\mathsf{T}} \tag{8.6.41}$$

Equations (8.6.38) and (8.6.41) define the Maximum Likelihood solution mean and covariance for training data $\mathcal{X}$. Consistent with our previous results, in fact these equations simply set the parameters to their sample statistics of the empirical distribution. That is, the mean is set to the sample mean of the data and the covariance to the sample covariance.

8.6.6 Bayesian Inference of the mean and variance

For simplicity here we deal with the univariate case. Assuming i.i.d. data the likelihood is

$$p(\mathcal{X}|\mu, \sigma^2) = \frac{1}{\left(2\pi\sigma^2\right)^{N/2}} e^{-\frac{1}{2\sigma^2}\sum_{n=1}^{N}(x^n - \mu)^2} \tag{8.6.42}$$

For a Bayesian treatment, we require the posterior of the parameters

$$p(\mu, \sigma^2|\mathcal{X}) \propto p(\mathcal{X}|\mu, \sigma^2)p(\mu, \sigma^2) = p(\mathcal{X}|\mu, \sigma^2)p(\mu|\sigma^2)p(\sigma^2) \tag{8.6.43}$$

Our aim is to find conjugate priors for the mean and variance. A convenient choice for a prior on the mean μ is that it is a Gaussian centred on μ_0:

$$p(\mu|\mu_0, \sigma_0^2) = \frac{1}{\sqrt{2\pi\sigma_0^2}} e^{-\frac{1}{2\sigma_0^2}(\mu_0 - \mu)^2} \tag{8.6.44}$$

The posterior is then

$$p(\mu,\sigma^2|\mathcal{X}) \propto \frac{1}{\sqrt{\sigma_0^2}}\frac{1}{(\sigma^2)^{N/2}}e^{-\frac{1}{2\sigma_0^2}(\mu_0-\mu)^2-\frac{1}{2\sigma^2}\sum_n(x^n-\mu)^2}p(\sigma^2) \tag{8.6.45}$$

It is convenient to write this in the form

$$p(\mu,\sigma^2|\mathcal{X}) = p(\mu|\sigma^2,\mathcal{X})p(\sigma^2|\mathcal{X}) \tag{8.6.46}$$

Since equation (8.6.45) has quadratic contributions in μ in the exponent, the conditional posterior $p(\mu|\sigma^2,\mathcal{X})$ is Gaussian. To identify this Gaussian we multiply out the terms in the exponent to arrive at

$$\exp -\frac{1}{2}\left(a\mu^2 - 2b\mu + c\right) \tag{8.6.47}$$

with

$$a = \frac{1}{\sigma_0^2}+\frac{N}{\sigma^2}, \quad b = \frac{\mu_0}{\sigma_0^2}+\frac{\sum_n x^n}{\sigma^2}, \quad c = \frac{\mu_0^2}{\sigma_0^2}+\sum_n\frac{(x^n)^2}{\sigma^2} \tag{8.6.48}$$

Using the identity

$$a\mu^2 - 2b\mu + c = a\left(\mu - \frac{b}{a}\right)^2 + \left(c - \frac{b^2}{a}\right) \tag{8.6.49}$$

we can write

$$p(\mu,\sigma^2|\mathcal{X}) \propto \underbrace{\sqrt{a}e^{-\frac{1}{2}a\left(\mu-\frac{b}{a}\right)^2}}_{p(\mu|\mathcal{X},\sigma^2)}\underbrace{\frac{1}{\sqrt{a}}e^{-\frac{1}{2}\left(c-\frac{b^2}{a}\right)}\frac{1}{\sqrt{\sigma_0^2}}\frac{1}{(\sigma^2)^{N/2}}p(\sigma^2)}_{p(\sigma^2|\mathcal{X})} \tag{8.6.50}$$

We encounter a difficulty in attempting to find a conjugate prior for σ^2 because the term b^2/a is not a simple expression of σ^2. For this reason we constrain

$$\sigma_0^2 \equiv \gamma\sigma^2 \tag{8.6.51}$$

for some fixed hyperparameter γ. Defining the constants

$$\tilde{a} = \frac{1}{\gamma}+N, \quad \tilde{b} = \frac{\mu_0}{\gamma}+\sum_n x^n, \quad \tilde{c} = \frac{\mu_0^2}{\gamma}+\sum_n (x^n)^2 \tag{8.6.52}$$

we have

$$c - \frac{b^2}{a} = \frac{1}{\sigma^2}\left(\tilde{c} - \frac{\tilde{b}^2}{\tilde{a}}\right) \tag{8.6.53}$$

Using this expression in equation (8.6.50) we obtain

$$p(\sigma^2|\mathcal{X}) \propto \left(\sigma^2\right)^{-N/2}e^{-\frac{1}{2\sigma^2}\left(\tilde{c}-\frac{\tilde{b}^2}{\tilde{a}}\right)}p(\sigma^2) \tag{8.6.54}$$

If we therefore use an inverse gamma distribution we will have a conjugate prior for σ^2. For a Gauss-Inverse-Gamma prior:

$$p(\mu,\sigma^2) = \mathcal{N}\left(\mu|\mu_0,\gamma\sigma^2\right) InvGam\left(\sigma^2|\alpha,\beta\right) \tag{8.6.55}$$

the posterior is also *Gauss-Inverse-Gamma* with

$$p(\mu,\sigma^2|\mathcal{X}) = \mathcal{N}\left(\mu\middle|\frac{\tilde{b}}{\tilde{a}},\frac{\sigma^2}{\tilde{a}}\right) InvGam\left(\sigma^2|\alpha+\frac{N}{2},\beta+\frac{1}{2}\left(\tilde{c}-\frac{\tilde{b}^2}{\tilde{a}}\right)\right) \tag{8.6.56}$$

8.6.7 Gauss-Gamma distribution

It is common to to use a prior on the *precision*, defined as the inverse variance

$$\lambda \equiv \frac{1}{\sigma^2} \tag{8.6.57}$$

If we then use a Gamma prior

$$p(\lambda|\alpha, \beta) = Gam(\lambda|\alpha, \beta) = \frac{1}{\beta^\alpha \Gamma(\alpha)} \lambda^{\alpha-1} e^{-\lambda/\beta} \tag{8.6.58}$$

The posterior will be

$$p(\lambda|\mathcal{X}, \alpha, \beta) = Gam\left(\lambda|\alpha + N/2, \tilde{\beta}\right) \tag{8.6.59}$$

where

$$\frac{1}{\tilde{\beta}} = \frac{1}{\beta} + \frac{1}{2}\left(\tilde{c} - \frac{\tilde{b}^2}{\tilde{a}}\right) \tag{8.6.60}$$

The *Gauss-Gamma* prior distribution

$$p(\mu, \lambda|\mu_0, \alpha, \beta, \gamma) = \mathcal{N}\left(\mu|\mu_0, \gamma\lambda^{-1}\right) Gam(\lambda|\alpha, \beta) \tag{8.6.61}$$

is the conjugate prior for a Gaussian with unknown mean μ and precision λ.

The posterior for this prior is a Gauss-Gamma distribution with parameters

$$p(\mu, \lambda|\mathcal{X}, \mu_0, \alpha, \beta, \gamma) = \mathcal{N}\left(\mu \middle| \frac{\tilde{b}}{\tilde{a}}, \frac{1}{\tilde{a}\lambda}\right) Gam\left(\lambda|\alpha + N/2, \tilde{\beta}\right) \tag{8.6.62}$$

The marginal $p(\mu|\mathcal{X}, \mu_0, \alpha, \beta, \gamma)$ is a Student's t-distribution. An example of a Gauss-Gamma prior/posterior is given in fig(8.8).

The Maximum Likelihood solution is recovered in the limit of a 'flat' (improper) prior $\mu_0 = 0, \gamma \to \infty, \alpha = 1/2, \beta \to \infty$, see exercise(103). The unbiased estimators for the mean and variance are given using the proper prior $\mu_0 = 0, \gamma \to \infty, \alpha = 1, \beta \to \infty$, exercise(104).

For the multivariate case, the extension of these techniques uses a multivariate Gaussian distribution for the conjugate prior on the mean, and an Inverse Wishart distribution for the conjugate prior on the covariance[122].

8.7 Exponential Family

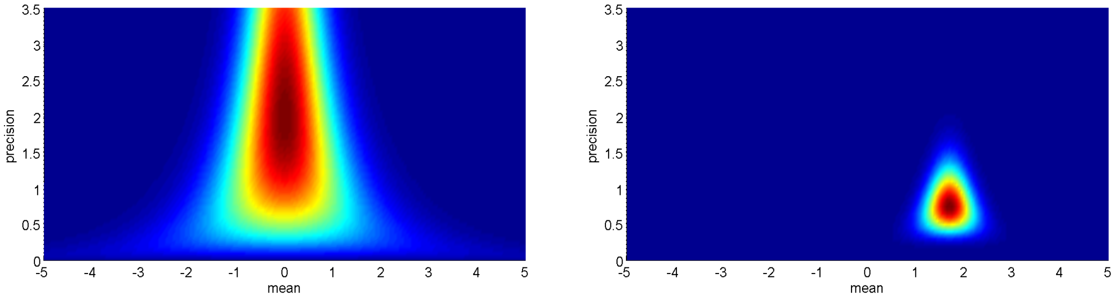
A theoretically convenient class of distributions are the exponential family, which contains many standard distributions, including the Gaussian, Gamma, Poisson, Dirichlet, Wishart, Multinomial, Markov Random Field.

Definition 82 (Exponential Family)**.** For a distribution on a (possibly multidimensional) variable x (continuous or discrete) an *exponential family* model is of the form

$$p(x|\boldsymbol{\theta}) = h(x) e^{\sum_i \eta_i(\boldsymbol{\theta}) T_i(x) - \psi(\boldsymbol{\theta})} \tag{8.7.1}$$

$\boldsymbol{\theta}$ are the parameters, $T_i(x)$ the test statistics, and $\psi(\boldsymbol{\theta})$ is the log partition function that ensure normalisation

$$\psi(\boldsymbol{\theta}) = \log \int_x h(x) e^{\sum_i \eta_i(\boldsymbol{\theta}) T_i(x)} \tag{8.7.2}$$



(a) Prior (b) Posterior

Figure 8.8: Bayesian approach to inferring the mean and precision (inverse variance) of a Gaussian based on $N = 10$ randomly drawn datapoints. **(a)**: A Gauss-Gamma prior with $\mu_0 = 0$, $\alpha = 2$, $\beta = 1$, $\gamma = 1$. **(b)**: Gauss-Gamma posterior conditional on the data. For comparison, the sample mean of the data is 1.87 and Maximum Likelihood optimal variance is 1.16 (computed using the N normalisation). The 10 datapoints were drawn from a Gaussian with mean 2 and variance 1. See `demoGaussBayes.m`.

One can always transform the parameters to the form $\boldsymbol{\eta}(\boldsymbol{\theta}) = \boldsymbol{\theta}$ in which case the distribution is in *canonical form*:

$$p(x|\boldsymbol{\theta}) = h(x)e^{\boldsymbol{\theta}^{\mathsf{T}}\mathbf{T}(x)-\psi(\boldsymbol{\theta})} \tag{8.7.3}$$

For example the univariate Gaussian can be written

$$\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{1}{2\sigma^2}(x-\mu)^2} = e^{-\frac{1}{2\sigma^2}x^2+\frac{\mu}{\sigma^2}x-\frac{\mu^2}{2\sigma^2}-\frac{1}{2}\log\pi\sigma^2} \tag{8.7.4}$$

Defining $t_1(x) = x$, $t_2(x) = -x^2/2$ and , $\theta_1 = \mu$, $\theta_2 = \sigma^2$, $h(x) = 1$, then

$$\eta_1(\theta) = \frac{\theta_1}{\theta_2}, \qquad \eta_2(\theta) = \frac{1}{\theta_2}, \qquad \psi(\theta) = \frac{1}{2}\left(\frac{\theta_1^2}{\theta_2} + \log\pi\theta_2\right) \tag{8.7.5}$$

Note that the parameterisation is not necessarily unique – we can for example rescale the functions $T_i(x)$ and inversely scale η_i by the same amount to arrive at an equivalent representation.

8.7.1 Conjugate priors

In principle, Bayesian learning for the exponential family is straightforward. In canonical form

$$p(x|\boldsymbol{\theta}) = h(x)e^{\boldsymbol{\theta}^{\mathsf{T}}\mathbf{T}(x)-\psi(\boldsymbol{\theta})} \tag{8.7.6}$$

For a prior with hyperparameters $\boldsymbol{\alpha}, \gamma$,

$$p(\boldsymbol{\theta}|\boldsymbol{\alpha},\gamma) \propto e^{\boldsymbol{\theta}^{\mathsf{T}}\boldsymbol{\alpha}-\gamma\psi(\boldsymbol{\theta})} \tag{8.7.7}$$

the posterior is

$$p(\boldsymbol{\theta}|x) \propto p(x|\boldsymbol{\theta})p(\boldsymbol{\theta}) \tag{8.7.8}$$

$$\propto h(x)e^{\boldsymbol{\theta}^{\mathsf{T}}\mathbf{T}(x)-\psi(\boldsymbol{\theta})}e^{\boldsymbol{\theta}^{\mathsf{T}}\boldsymbol{\alpha}-\gamma\psi(\boldsymbol{\theta})} \tag{8.7.9}$$

$$\propto e^{\boldsymbol{\theta}^{\mathsf{T}}[\mathbf{T}(x)+\boldsymbol{\alpha}]-[\gamma+1]\psi(\boldsymbol{\theta})} \tag{8.7.10}$$

so that the prior equation (8.7.7) is conjugate for the exponential family likelihood equation (8.7.6). Whilst the likelihood is in the exponential family, the conjugate prior is not necessarily in the exponential family.

8.8 The Kullback-Leibler Divergence KL$(q|p)$

The Kullback-Leibler divergence KL$(q|p)$ measures the 'difference' between distributions q and p[67].

Definition 83. KL divergence For two distributions $q(x)$ and $p(x)$

$$\mathrm{KL}(q|p) \equiv \langle \log q(x) - \log p(x) \rangle_{q(x)} \geq 0 \tag{8.8.1}$$

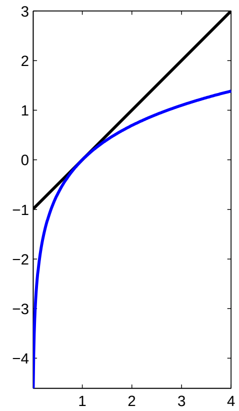
where $\langle f(x) \rangle_{r(x)}$ denotes average of the function $f(x)$ with respect to the distribution $r(x)$.

The KL divergence is ≥ 0

The KL divergence is widely used and it is therefore important to understand why the divergence is positive.

To see this, consider the following linear bound on the function $\log(x)$

$$\log(x) \leq x - 1 \tag{8.8.2}$$



as plotted in the figure on the right. Replacing x by $p(x)/q(x)$ in the above bound

$$\frac{p(x)}{q(x)} - 1 \geq \log \frac{p(x)}{q(x)} \tag{8.8.3}$$

Since probabilities are non-negative, we can multiply both sides by $q(x)$ to obtain

$$p(x) - q(x) \geq q(x) \log p(x) - q(x) \log q(x) \tag{8.8.4}$$

We now integrate (or sum in the case of discrete variables) both sides. Using $\int p(x)dx = 1$, $\int q(x)dx = 1$,

$$1 - 1 \geq \langle \log p(x) - \log q(x) \rangle_{q(x)} \tag{8.8.5}$$

Rearranging gives

$$\langle \log q(x) - \log p(x) \rangle_{q(x)} \equiv \mathrm{KL}(q|p) \geq 0 \tag{8.8.6}$$

The KL divergence is zero if and only if the two distributions are exactly the same.

8.8.1 Entropy

For both discrete and continuous variables, the *entropy* is defined as

$$H(p) \equiv - \langle \log p(x) \rangle_{p(x)} \tag{8.8.7}$$

For continuous variables, this is also called the *differential entropy*, see also exercise(114). The entropy is a measure of the uncertainty in a distribution. One way to see this is that

$$H(p) = -\mathrm{KL}(p|u) + \text{const.} \tag{8.8.8}$$

where u is a uniform distribution. Since the $\mathrm{KL}(p|u) \geq 0$, the less like a uniform distribution p is, the smaller will be the entropy. Or, vice versa, the more similar p is to a uniform distribution, the greater will be the entropy. Since the uniform distribution contains the least information a prior about which state $p(x)$ is in, the entropy is therefore a measure of the a priori uncertainty in the state occupancy. For a discrete distribution we can permute the state labels without changing the entropy. For a discrete distribution the entropy is positive, whereas the differential entropy can be negative.

8.9 Code

`demoGaussBayes.m`: Bayesian fitting of a univariate Gaussian
`logGaussGamma.m`: Plotting routine for a Gauss-Gamma distribution

8.10 Exercises

Exercise 81. *In a public lecture, the following phrase was uttered by a Professor of Experimental Psychology: 'In a recent data survey, 90% of people claim to have above average intelligence, which is clearly nonsense!' [Audience Laughs]. Is it theoretically possible for 90% of people to have above average intelligence? If so, give an example, otherwise explain why not. What about above median intelligence?*

Exercise 82. *Consider the distribution*

$$p(x, y) \propto (x^2 + y^2)^2 e^{-x^2-y^2} \tag{8.10.1}$$

Show that $\langle x \rangle = \langle y \rangle = 0$*. Furthermore show that* x *and* y *are uncorrelated,* $\langle xy \rangle = \langle x \rangle \langle y \rangle$*. Whilst* x *and* y *are uncorrelated, show that they are nevertheless dependent.*

Exercise 83. *For a variable* x *with* $dom(x) = \{0, 1\}$*, and* $p(x = 1) = \theta$*, show that in* n *independent draws* $x_1, \ldots, x_n$ *from this distribution, the probability of observing* k *states 1 is the Binomial distribution*

$$\binom{n}{k} \theta^k (1-\theta)^{n-k} \tag{8.10.2}$$

Exercise 84 (Normalisation constant of a Gaussian)**.** *The normalisation constant of a Gaussian distribution is related to the integral*

$$I = \int_{-\infty}^{\infty} e^{-\frac{1}{2}x^2} dx \tag{8.10.3}$$

By considering

$$I^2 = \int_{-\infty}^{\infty} e^{-\frac{1}{2}x^2} dx \int_{-\infty}^{\infty} e^{-\frac{1}{2}y^2} dy \tag{8.10.4}$$

show that

1. $I = \sqrt{2\pi}$
2. $\int_{-\infty}^{\infty} e^{-\frac{1}{2\sigma^2}(x-\mu)^2} dx = \sqrt{2\pi\sigma^2}$

Exercise 85. *For a univariate Gaussian distribution, show that*

1. $\mu = \langle x \rangle_{\mathcal{N}(x|\mu,\sigma^2)}$
2. $\sigma^2 = \left\langle (x-\mu)^2 \right\rangle_{\mathcal{N}(x|\mu,\sigma^2)}$

Exercise 86. *Show that the marginal of a Dirichlet distribution is another Dirichlet distribution:*

$$\int_{\theta_j} \text{Dirichlet}\,(\theta|\mathbf{u}) = \text{Dirichlet}\left(\theta_{\setminus j}|\mathbf{u}_{\setminus j}\right) \tag{8.10.5}$$

Exercise 87. *For a Beta distribution, show that*

$$\left\langle x^k \right\rangle = \frac{B(\alpha+k, \beta)}{B(\alpha, \beta)} \tag{8.10.6}$$

and, using $\Gamma(x+1) = x\Gamma(x)$*, derive an explicit expression for the* k^{th} *moment of a Beta distribution.*

Exercise 88. *Define the* moment generating function *as*

$$g(t) \equiv \left\langle e^{tx} \right\rangle_{p(x)} \tag{8.10.7}$$

Show that

$$\lim_{t \to 0} \frac{d^k}{dt^k} g(t) = \left\langle x^k \right\rangle_{p(x)} \tag{8.10.8}$$

Exercise 89 (Change of variables)**.** *Consider a one dimensional continuous random variable x with corresponding $p(x)$. For a variable $y = f(x)$, where $f(x)$ is a monotonic function, show that the distribution of y is*

$$p(y) = p(x) \left(\frac{df}{dx} \right)^{-1}, \quad x = f^{-1}(y) \tag{8.10.9}$$

Exercise 90 (Normalisation of a Multivariate Gaussian)**.** *Consider*

$$I = \int_{-\infty}^{\infty} e^{-\frac{1}{2}(x-\mu)^{\mathsf{T}} \mathbf{\Sigma}^{-1} (x-\mu)} dx \tag{8.10.10}$$

By using the transformation

$$z = \mathbf{\Sigma}^{-\frac{1}{2}} (x - \mu) \tag{8.10.11}$$

show that

$$I = \sqrt{\det (2\pi \mathbf{\Sigma})} \tag{8.10.12}$$

Exercise 91. *Consider the partitioned matrix*

$$\mathbf{M} = \begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{pmatrix} \tag{8.10.13}$$

for which we wish to find the inverse $\mathbf{M}^{-1}$. We assume that $\mathbf{A}$ is $m \times m$ and invertible, and D is $n \times n$ and invertible. By definition, the partitioned inverse

$$\mathbf{M}^{-1} = \begin{pmatrix} \mathbf{P} & \mathbf{Q} \\ \mathbf{R} & \mathbf{S} \end{pmatrix} \tag{8.10.14}$$

must satisfy

$$\begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{pmatrix} \begin{pmatrix} \mathbf{P} & \mathbf{Q} \\ \mathbf{R} & \mathbf{S} \end{pmatrix} = \begin{pmatrix} \mathbf{I}_m & 0 \\ 0 & \mathbf{I}_n \end{pmatrix} \tag{8.10.15}$$

where in the above I_m is the $m \times m$ identity matrix of the same dimension as A, and $\mathsf{0}$ the zero matrix of the same dimension as D. Using the above, derive the results

$$\mathbf{P} = \left(\mathbf{A} - \mathbf{B}\mathbf{D}^{-1}\mathbf{C}\right)^{-1}$$
$$\mathbf{Q} = -\mathbf{A}^{-1}\mathbf{B}\left(\mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B}\right)^{-1}$$
$$\mathbf{R} = -\mathbf{D}^{-1}\mathbf{C}\left(\mathbf{A} - \mathbf{B}\mathbf{D}^{-1}\mathbf{C}\right)^{-1}$$
$$\mathbf{S} = \left(\mathbf{D} - C\mathbf{A}^{-1}\mathbf{B}\right)^{-1}$$

Exercise 92. *Show that for Gaussian distribution $p(x) = \mathcal{N}\left(x|\mu, \sigma^2\right)$ the skewness and kurtosis are both zero.*

Exercise 93. *Consider a small interval of time δt and let the probability of an event occurring in this small interval be $\theta \delta t$. Derive a distribution that expresses the probability of at least one event in an interval from 0 to t.*

Exercise 94. *Consider a vector variable* $\mathbf{x} = (x_1, \ldots, x_n)$ *and set of functions defined on each component of* x, $\phi_i(x_i)$. *For example for* $\mathbf{x} = (x_1, x_2)$ *we might have*

$$\phi_1(x_1) = -|x_1|, \quad \phi_2(x_2) = -x_2^2 \tag{8.10.16}$$

Consider the distribution

$$p(\mathbf{x}|\theta) = \frac{1}{Z} e^{\theta^{\mathsf{T}} \phi(\mathbf{x})} \tag{8.10.17}$$

where $\phi(\mathbf{x})$ *is a vector function with* i^{th} *component* $\phi_i(x_i)$, *and* θ *is a parameter vector. Each component is tractably integrable in the sense that*

$$\int_{-\infty}^{\infty} e^{\theta_i \phi_i(x_i)} dx_i \tag{8.10.18}$$

can be computed either analytically or to an acceptable numerical accuracy. Show that

1. $x_i \perp\!\!\!\perp x_j | \emptyset$.
2. *The normalisation constant* Z *can be tractably computed.*
3. *Consider the transformation*

$$\mathbf{x} = \mathbf{M}\mathbf{y} \tag{8.10.19}$$

for an invertible matrix $\mathbf{M}$. *Show that the distribution* $p(\mathbf{y}|\mathbf{M}, \theta)$ *is tractable (its normalisation constant is known), and that, in general,* $y_i \top\!\!\!\top y_j \mid \emptyset$. *Explain the significance of this is deriving tractable multivariate distributions.*

Exercise 95. *Show that we may reparameterise the Beta distribution, definition(71) by writing the parameters* α *and* β *as functions of the mean* m *and variance* s *using*

$$\alpha = \beta\gamma, \qquad \gamma \equiv m/(1-m) \tag{8.10.20}$$

$$\beta = \frac{1}{1+\gamma}\left(\frac{\gamma}{s\,(1+\gamma)^2} - 1\right) \tag{8.10.21}$$

Exercise 96. *Consider the function*

$$f(\gamma + \alpha, \beta, \theta) \equiv \theta^{\gamma+\alpha-1}\,(1-\theta)^{\beta-1} \tag{8.10.22}$$

show that

$$\lim_{\gamma\to 0} \frac{\partial}{\partial\gamma} f(\gamma+\alpha, \beta, \theta) = \theta^{\alpha-1}\,(1-\theta)^{\beta-1} \log\theta \tag{8.10.23}$$

and hence that

$$\int \theta^{\alpha-1}\,(1-\theta)^{\beta-1} \log\theta d\theta = \lim_{\gamma\to 0} \frac{\partial}{\partial\gamma} \int f(\gamma+\alpha, \beta, \theta) d\theta = \frac{\partial}{\partial\alpha} \int f(\alpha, \beta, \theta) d\theta \tag{8.10.24}$$

Using this result, show therefore that

$$\langle \log\theta \rangle_{B(\theta|\alpha,\beta)} = \frac{\partial}{\partial\alpha} \log B(\alpha, \beta) \tag{8.10.25}$$

where $B(\alpha, \beta)$ *is the Beta function. Show additionally that*

$$\langle \log(1-\theta) \rangle_{B(\theta|\alpha,\beta)} = \frac{\partial}{\partial\beta} \log B(\alpha, \beta) \tag{8.10.26}$$

Using the fact that

$$B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)} \tag{8.10.27}$$

where $\Gamma(x)$ *is the gamma function, relate the above averages to the digamma function, defined as*

$$\psi(x) = \frac{d}{dx} \log\Gamma(x) \tag{8.10.28}$$

Exercise 97. *Using a similar 'generating function' approach as in exercise(96), explain how to compute*

$$\langle \log \theta_i \rangle_{\text{Dirichlet}(\theta|\mathbf{u})} \tag{8.10.29}$$

Exercise 98. *Consider the function*

$$f(x) = \int_0^\infty \delta\left(\sum_{i=1}^n \theta_i - x\right) \prod_i \theta_i^{u_i - 1} d\theta_1 \ldots d\theta_n \tag{8.10.30}$$

Show that the Laplace transform of $f(x)$, $\tilde{f}(s) \equiv \int_0^\infty e^{-sx} f(x) dx$ *is*

$$\tilde{f}(s) = \prod_{i=1}^n \left\{ \int_0^\infty e^{-s\theta_i} \theta_i^{u_i - 1} d\theta_i \right\} = \frac{1}{s^{\sum_i u_i}} \prod_{i=1}^n \Gamma(u_i) \tag{8.10.31}$$

By taking the inverse Laplace transform, show that

$$f(x) = \frac{\prod_{i=1}^n \Gamma(u_i)}{\Gamma\left(\sum_i u_i - 1\right)} x^{\sum_i u_i - 1} \tag{8.10.32}$$

Hence show that the normalisation constant of a Dirichlet distribution with parameters $\mathbf{u}$ *is given by*

$$\frac{\prod_{i=1}^n \Gamma(u_i)}{\Gamma\left(\sum_i u_i\right)} \tag{8.10.33}$$

Exercise 99. *By using the Laplace transform, as in exercise(98), show that the marginal of a Dirichlet distribution is a Dirichlet distribution.*

Exercise 100. *Derive the formula for the differential entropy of a multi-variate Gaussian.*

Exercise 101. *Show that for a gamma distribution* $Gam(x|\alpha, \beta)$ *the mode is given by*

$$x^* = (\alpha - 1)\beta \tag{8.10.34}$$

provided that $\alpha \geq 1$.

Exercise 102. *Consider a distribution* $p(x|\theta)$ *and a distribution with* θ *changed by a small amount,* δ. *Take the Taylor expansion of*

$$KL(p(x|\theta)|p(x|\theta + \delta)) \tag{8.10.35}$$

for small δ *and show that this is equal to*

$$-\frac{\delta^2}{2} \left\langle \frac{\partial^2}{\partial \theta^2} \log p(x|\theta) \right\rangle_{p(\theta)} \tag{8.10.36}$$

More generally for a distribution parameterised by a vector $\theta_i + \delta_i$, *show that a small change in the parameter results in*

$$\sum_{i,j} \frac{\delta_i \delta_j}{2} F_{ij} \tag{8.10.37}$$

where the Fisher Information matrix is defined as

$$F_{ij} = -\left\langle \frac{\partial^2}{\partial \theta_i \partial \theta_j} \log p(x|\theta) \right\rangle_{p(\theta)} \tag{8.10.38}$$

Show that the Fisher information matrix is positive (semi) definite by expressing it equivalently as

$$F_{ij} = \left\langle \frac{\partial}{\partial \theta_i} \log p(x|\theta) \frac{\partial}{\partial \theta_j} \log p(x|\theta) \right\rangle_{p(\theta)} \tag{8.10.39}$$

Exercise 103. *Consider the joint prior distribution*

$$p(\mu, \lambda|\mu_0, \alpha, \beta, \gamma) = \mathcal{N}\left(\mu|\mu_0, \gamma\lambda^{-1}\right) Gam\left(\lambda|\alpha, \beta\right) \tag{8.10.40}$$

Show that for $\mu_0 = 0$, $\gamma \to \infty$, $\beta \to \infty$, then the prior distribution becomes 'flat' (independent of μ and λ) for $\alpha = 1/2$. Show that for these settings the mean and variance that jointly maximise the posterior equation (8.6.62) are given by the standard Maximum Likelihood settings

$$\mu_* = \frac{1}{N}\sum_n x^n, \qquad \sigma_*^2 = \frac{1}{N}\sum_n (x^n - \mu_*)^2 \tag{8.10.41}$$

Exercise 104. *Show that in the limit $\mu_0 = 0, \gamma \to \infty, \alpha = 1, \beta \to \infty$, the jointly optimal mean and variance obtained from*

$$\underset{\mu,\lambda}{\operatorname{argmax}}\ p(\mu, \lambda|\mathcal{X}, \alpha, \beta, \gamma) \tag{8.10.42}$$

is given by

$$\mu_* = \frac{1}{N}\sum_n x^n, \qquad \sigma_*^2 = \frac{1}{N+1}\sum_n (x^n - \mu_*)^2 \tag{8.10.43}$$

where $\sigma_*^2 = 1/\lambda_*$. Note that these correspond to the standard 'unbiased' estimators of the mean and variance.

Exercise 105. *For the Gauss-Gamma posterior $p(\mu, \lambda|\mu_0, \alpha, \beta, \mathcal{X})$ given in equation (8.6.62) compute the marginal posterior $p(\mu|\mu_0, \alpha, \beta, \mathcal{X})$. What is the mean of this distribution?*

Exercise 106. *Derive equation (8.6.28).*

Exercise 107. *Consider the multivariate Gaussian distribution $p(\mathbf{x}) \sim \mathcal{N}\left(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}\right)$ on the vector $\mathbf{x}$ with components $x_1, \ldots, x_n$:*

$$p(\mathbf{x}) = \frac{1}{\sqrt{\det\left(2\pi\boldsymbol{\Sigma}\right)}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})} \tag{8.10.44}$$

Calculate $p(x_i|x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$.

Exercise 108. *Observations $y_0, \ldots, y_{n-1}$ are noisy i.i.d. measurements of an underlying variable x with $p(x) \sim \mathcal{N}\left(x|0, \sigma_0^2\right)$ and $p(y_i|x) \sim \mathcal{N}\left(y_i|x, \sigma^2\right)$ for $i = 0, \ldots, n-1$. Show that $p(x|y_0, \ldots, y_{n-1})$ is Gaussian with mean*

$$\mu = \frac{n\sigma_0^2}{n\sigma_0^2 + \sigma^2}\overline{y} \tag{8.10.45}$$

where $\overline{y} = (y_0 + y_1 + \ldots + y_{n-1})/n$ and variance σ_n^2 such that

$$\frac{1}{\sigma_n^2} = \frac{n}{\sigma^2} + \frac{1}{\sigma_0^2}. \tag{8.10.46}$$

Exercise 109. *Consider a set of data $x^1, \ldots, x^N$ drawn from a Gaussian with know mean μ and unknown variance σ^2. Assume a prior gamma distribution on $\tau = 1/\sigma^2$,*

$$p(\tau) = Gam\left(\tau|a, bs\right) \tag{8.10.47}$$

1. *Show that the posterior distribution is*

$$p(\tau|\mathcal{X}) = Gam\left(\tau|a + \frac{N}{2}, b + \frac{1}{2}\sum_{n=1}^{N}(x^n - \mu)^2\right) \tag{8.10.48}$$

2. *Show that the distribution for x is*

$$p(x|\mathcal{X}) = \int p(x|\tau)p(\tau|\mathcal{X})d\tau = Student\left(x|\mu, \lambda = \frac{a}{b}, \nu = 2a\right) \tag{8.10.49}$$

Exercise 110. *The Poisson distribution is a discrete distribution on the non-negative integers, with*

$$P(x) = \frac{e^{-\lambda}\lambda^x}{x!} \qquad x = 0,\ 1,\ 2, \ldots \tag{8.10.50}$$

You are given a sample of n observations $x_1, \ldots, x_n$ drawn from this distribution. Determine the maximum likelihood estimator of the Poisson parameter λ.

Exercise 111. *For a Gaussian mixture model*

$$p(\mathbf{x}) = \sum_i p_i \mathcal{N}\left(\mathbf{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right), \qquad p_i > 0, \sum_i p_i = 1 \tag{8.10.51}$$

show that $p(\mathbf{x})$ has mean

$$\langle \mathbf{x} \rangle = \sum_i p_i \boldsymbol{\mu}_i \tag{8.10.52}$$

and covariance

$$\sum_i p_i \left(\boldsymbol{\Sigma}_i + \boldsymbol{\mu}_i \boldsymbol{\mu}_i^\mathsf{T}\right) - \sum_i p_i \boldsymbol{\mu}_i \sum_j p_j \boldsymbol{\mu}_j^\mathsf{T} \tag{8.10.53}$$

Exercise 112. *Show that for the whitened data matrix, given in equation (8.6.34), $\mathbf{Z}\mathbf{Z}^\mathsf{T} = N\mathbf{I}$.*

Exercise 113. *Consider a uniform distribution $p_i = 1/N$ defined on states $i = 1, \ldots, N$. Show that the entropy of this distribution is*

$$H = -\sum_{i=1}^{N} p_i \log p_i = \log N \tag{8.10.54}$$

and that there for as the number of states N increases to infinity, the entropy diverges to infinity.

Exercise 114. *Consider a continuous distribution $p(x)$, $x \in [0, 1]$. We can form a discrete approximation with probabilities p_i to this continuous distribution by identifying a continuous value i/N for each state $i = 1, \ldots, N$. With this*

$$p_i = \frac{p(i/N)}{\sum_i p(i/N)} \tag{8.10.55}$$

show that the entropy $H = -\sum_i p_i \log p_i$ is given by

$$H = -\frac{1}{\sum_i p(i/N)} \sum_i p(i/N) \log p(i/N) + \log \sum_i p(i/N) \tag{8.10.56}$$

Since for a continuous distribution

$$\int_0^1 p(x)dx = 1 \tag{8.10.57}$$

a discrete approximation of this integral into bins of size $1/N$ gives

$$\frac{1}{N} \sum_{i=1}^{N} p(i/N) = 1 \tag{8.10.58}$$

Hence show that for large N,

$$H \approx -\int_0^1 p(x) \log p(x) dx + \textsf{const}. \tag{8.10.59}$$

where the constant tends to infinity as $N \to \infty$. Note that this result says that as a continuous distribution has essentially an infinite number of states, the amount of uncertainty in the distribution is infinite (alternatively, we would need an infinite number of bits to specify a continuous value). This motivates the definition of the differential entropy, which neglects the infinite constant of the limiting case of the discrete entropy.

Exercise 115. *Consider two multivariate Gaussians $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ and $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$.*

1. *Show that the log product of the two Gaussians is given by*

$$-\frac{1}{2}\mathbf{x}^{\textsf{T}}\left(\boldsymbol{\Sigma}_1^{-1} + \boldsymbol{\Sigma}_2^{-1}\right)\mathbf{x} + \mathbf{x}^{\textsf{T}}\left(\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_2^{-1}\boldsymbol{\mu}_2\right) - \frac{1}{2}\left(\boldsymbol{\mu}_1^{\textsf{T}}\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2^{\textsf{T}}\boldsymbol{\Sigma}_2^{-1}\boldsymbol{\mu}_2\right) - \frac{1}{2}\log\det\left(2\pi\boldsymbol{\Sigma}_1\right)\det\left(2\pi\boldsymbol{\Sigma}_2\right)$$

2. *Defining $\mathbf{A} = \boldsymbol{\Sigma}_1^{-1} + \boldsymbol{\Sigma}_2^{-1}$ and $\mathbf{b} = \boldsymbol{\Sigma}_1^{-1}\boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_2^{-1}\boldsymbol{\mu}_2$ we can write the above as*

$$-\frac{1}{2}\left(\mathbf{x} - \mathbf{A}^{-1}\mathbf{b}\right)^{\textsf{T}}\mathbf{A}\left(\mathbf{x} - \mathbf{A}^{-1}\mathbf{b}\right) + \frac{1}{2}\mathbf{b}^{\textsf{T}}\mathbf{A}^{-1}\mathbf{b} - \frac{1}{2}\left(\boldsymbol{\mu}_1^{\textsf{T}}\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2^{\textsf{T}}\boldsymbol{\Sigma}_2^{-1}\boldsymbol{\mu}_2\right) - \frac{1}{2}\log\det\left(2\pi\boldsymbol{\Sigma}_1\right)\det\left(2\pi\boldsymbol{\Sigma}_2\right)$$

   Writing $\boldsymbol{\Sigma} = \mathbf{A}^{-1}$ and $\boldsymbol{\mu} = \mathbf{A}^{-1}\mathbf{b}$ show that the product of Gaussians is a Gaussian with covariance

$$\boldsymbol{\Sigma} = \boldsymbol{\Sigma}_1\left(\boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2\right)^{-1}\boldsymbol{\Sigma}_2 \tag{8.10.60}$$

   mean

$$\boldsymbol{\mu} = \boldsymbol{\Sigma}_1\left(\boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2\right)^{-1}\boldsymbol{\mu}_2 + \boldsymbol{\Sigma}_2\left(\boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2\right)^{-1}\boldsymbol{\mu}_1 \tag{8.10.61}$$

   and log prefactor

$$\frac{1}{2}\mathbf{b}^{\textsf{T}}\mathbf{A}^{-1}\mathbf{b} - \frac{1}{2}\left(\boldsymbol{\mu}_1^{\textsf{T}}\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2^{\textsf{T}}\boldsymbol{\Sigma}_2^{-1}\boldsymbol{\mu}_2\right) - \frac{1}{2}\log\det\left(2\pi\boldsymbol{\Sigma}_1\right)\det\left(2\pi\boldsymbol{\Sigma}_2\right) + \frac{1}{2}\log\det\left(2\pi\boldsymbol{\Sigma}\right)$$

3. *Show that this can be written as*

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)\,\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})\,\frac{\exp\left(-\frac{1}{2}\left(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\right)^{\textsf{T}}\mathbf{S}^{-1}\left(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\right)\right)}{\sqrt{\det\left(2\pi\mathbf{S}\right)}} \tag{8.10.62}$$

CHAPTER 9

Learning as Inference

9.1 Learning as Inference

In previous chapters we largely assumed that all distributions are fully specified for the inference tasks. In Machine Learning and related fields, however, the distributions need to be learned on the basis of data. Learning is then the problem of integrating data with domain knowledge of the model environment.

Definition 84 (Priors and Posteriors)**.** Priors and posteriors typically refer to the parameter distributions before (prior to) and after (posterior to) seeing the data. Formally, Bayes' rule relates these via

$$p(\theta|\mathcal{V}) = \frac{p(\mathcal{V}|\theta)p(\theta)}{p(\mathcal{V})} \tag{9.1.1}$$

where θ is the parameter of interest and $\mathcal{V}$ represents the observed (visible) data.

9.1.1 Learning the bias of a coin

Consider data expressing the results of tossing a coin. We write $v^n = 1$ if on toss n the coin comes up heads, and $v^n = 0$ if it is tails. Our aim is to estimate the probability θ that the coin will be a head, $p(v^n = 1|\theta) = \theta$ – called the 'bias' of the coin. For a fair coin, $\theta = 0.5$. The variables in this environment are $v^1, \ldots, v^N$ and θ and we require a model of the probabilistic interaction of the variables, $p(v^1, \ldots, v^N, \theta)$. Assuming there is no dependence between the observed tosses, except through θ, we have the Belief Network

$$p(v^1, \ldots, v^N, \theta) = p(\theta) \prod_{n=1}^{N} p(v^n|\theta) \tag{9.1.2}$$

which is depicted in fig(9.1). The assumption that each observation is *identically and independently distributed* is called the i.i.d. assumption.

Learning refers to using the observations $v^1, \ldots, v^N$ to infer θ. In this context, our interest is

$$p(\theta|v^1, \ldots, v^N) = \frac{p(v^1, \ldots, v^N, \theta)}{p(v^1, \ldots, v^N)} = \frac{p(v^1, \ldots, v^N|\theta)p(\theta)}{p(v^1, \ldots, v^N)} \tag{9.1.3}$$

We still need to fully specify the prior $p(\theta)$. To avoid complexities resulting from continuous variables, we'll consider a discrete θ with only three possible states, $\theta \in \{0.1, 0.5, 0.8\}$. Specifically, we assume

$$p(\theta = 0.1) = 0.15, \qquad p(\theta = 0.5) = 0.8, \qquad p(\theta = 0.8) = 0.05 \tag{9.1.4}$$

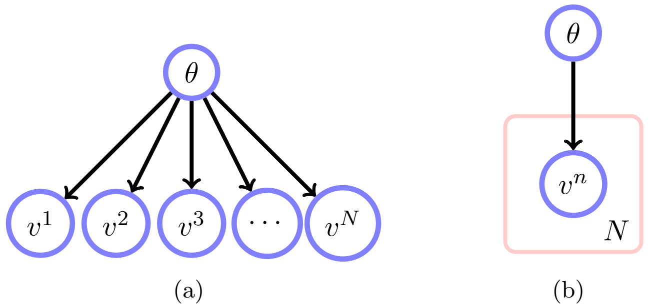


Figure 9.1: **(a)**: Belief Network for coin tossing model. **(b)**: *Plate* notation equivalent of (a). A plate replicates the quantities inside the plate a number of times as specified in the plate.

as shown in fig(9.2a). This prior expresses that we have 80% belief that the coin is 'fair', 5% belief the coin is biased to land heads (with $\theta = 0.8$), and 15% belief the coin is biased to land tails (with $\theta = 0.1$). The distribution of θ given the data and our beliefs is

$$p(\theta|v^1, \ldots, v^N) \propto p(\theta) \prod_{n=1}^{N} p(v^n|\theta) = p(\theta) \prod_{n=1}^{N} \theta^{\mathbb{I}[v^n=1]} (1-\theta)^{\mathbb{I}[v^n=0]} \tag{9.1.5}$$

$$\propto p(\theta)\theta^{\sum_{n=1}^{N} \mathbb{I}[v^n=1]} (1-\theta)^{\sum_{n=1}^{N} \mathbb{I}[v^n=0]} \tag{9.1.6}$$

In the above $\sum_{n=1}^{N} \mathbb{I}[v^n = 1]$ is the number of occurrences of heads, which we more conveniently denote as N_H. Likewise, $\sum_{n=1}^{N} \mathbb{I}[v^n = 0]$ is the number of tails, N_T. Hence

$$p(\theta|v^1, \ldots, v^N) \propto p(\theta)\theta^{N_H} (1-\theta)^{N_T} \tag{9.1.7}$$

For an experiment with $N_H = 2$, $N_T = 8$, the posterior distribution is

$$p(\theta = 0.1|\mathcal{V}) = k \times 0.15 \times 0.1^2 \times 0.9^8 = k \times 6.46 \times 10^{-4} \tag{9.1.8}$$

$$p(\theta = 0.5|\mathcal{V}) = k \times 0.8 \times 0.5^2 \times 0.5^8 = k \times 7.81 \times 10^{-4} \tag{9.1.9}$$

$$p(\theta = 0.8|\mathcal{V}) = k \times 0.05 \times 0.8^2 \times 0.2^8 = k \times 8.19 \times 10^{-8} \tag{9.1.10}$$

where $\mathcal{V}$ is shorthand for $v^1, \ldots, v^N$. From the normalisation requirement we have $1/k = 6.46 \times 10^{-4} + 7.81 \times 10^{-4} + 8.19 \times 10^{-8} = 0.0014$, so that

$$p(\theta = 0.1|\mathcal{V}) = 0.4525, \quad p(\theta = 0.5|\mathcal{V}) = 0.5475, \quad p(\theta = 0.8|\mathcal{V}) = 0.0001 \tag{9.1.11}$$

as shown in fig(9.2b). These are the 'posterior' parameter beliefs. In this case, if we were asked to choose a single *a posteriori* most likely value for θ, it would be $\theta = 0.5$, although our confidence in this is low since the posterior belief that $\theta = 0.1$ is also appreciable. This result is intuitive since, even though we observed more Tails than Heads, our prior belief was that it was more likely the coin is fair.

Repeating the above with $N_H = 20$, $N_T = 80$, the posterior changes to

$$p(\theta = 0.1|\mathcal{V}) = 1-1.93\times10^{-6}, \quad p(\theta = 0.5|\mathcal{V}) = 1.93\times10^{-6}, \quad p(\theta = 0.8|\mathcal{V}) = 2.13\times10^{-35} \tag{9.1.12}$$

fig(9.1c), so that the posterior belief in $\theta = 0.1$ dominates. This is reasonable since in this situation, there are so many more tails than heads that this is unlikely to occur from a fair coin. Even though we *a priori* thought that the coin was fair, *a posteriori* we have enough evidence to change our minds.

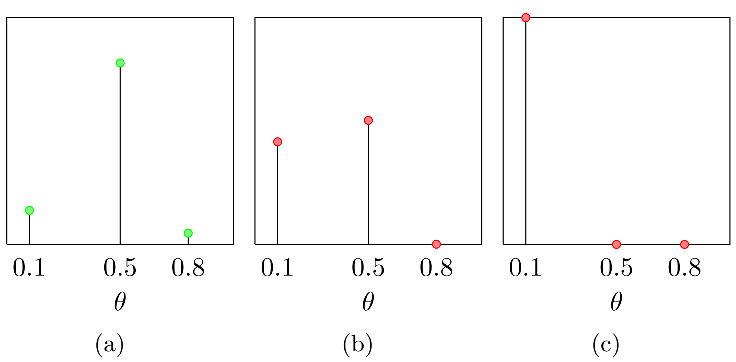


Figure 9.2: **(a)**: Prior encoding our beliefs about the amount the coin is biased to heads. **(b)**: Posterior having seen 2 heads and 8 tails. **(c)**: Posterior having seen 20 heads and 80 tails.

9.1.2 Making decisions

In itself, the Bayesian posterior merely represents our beliefs and says nothing about how best to summarise these beliefs. In situations in which decisions need to be taken under uncertainty we need to additionally specify what the utility of any decision is, as in chapter(7).

In the coin tossing scenario where θ is assumed to be either $0.1, 0.5$ or 0.8, we setup a decision problem as follows: If we correctly state the bias of the coin we gain 10 points; being incorrect, however, loses 20 points. We can write this using

$$U(\theta, \theta^0) = 10\mathbb{I}\left[\theta = \theta^0\right] - 20\mathbb{I}\left[\theta \neq \theta^0\right] \tag{9.1.13}$$

where θ^0 is the true value for the bias. The expected utility of the decision that the coin is $\theta = 0.1$ is

$$\begin{aligned} U(\theta = 0.1) = & \; U(\theta = 0.1, \theta^0 = 0.1)p(\theta^0 = 0.1|\mathcal{V}) \\ & + U(\theta = 0.1, \theta^0 = 0.5)p(\theta^0 = 0.5|\mathcal{V}) + U(\theta = 0.1, \theta^0 = 0.8)p(\theta^0 = 0.8|\mathcal{V}) \end{aligned} \tag{9.1.14}$$

Plugging in the numbers from equation (9.1.11), we obtain

$$U(\theta = 0.1) = 10 \times 0.4525 - 20 \times 0.5475 - 20 \times 0.0001 = -6.4270 \tag{9.1.15}$$

Similarly

$$U(\theta = 0.5) = 10 \times 0.5475 - 20 \times 0.4525 - 20 \times 0.0001 = -3.5770 \tag{9.1.16}$$

and

$$U(\theta = 0.8) = 10 \times 0.0001 - 20 \times 0.4525 - 20 \times 0.5475 = -19.999 \tag{9.1.17}$$

So that the best decision is to say that the coin is unbiased, $\theta = 0.5$.

Repeating the above calculations for $N_H = 20, N_T = 80$, we arrive at

$$U(\theta = 0.1) = 10 \times (1 - 1.93 \times 10^{-6}) - 20\left(1.93 \times 10^{-6} + 2.13 \times 10^{-35}\right) = 9.9999 \tag{9.1.18}$$

$$U(\theta = 0.5) = 10 \times 1.93 \times 10^{-6} - 20\left(1 - 1.93 \times 10^{-6} + 2.13 \times 10^{-35}\right) \approx -20.0 \tag{9.1.19}$$

$$U(\theta = 0.8) = 10 \times 2.13 \times 10^{-35} - 20\left(1 - 1.93 \times 10^{-6} + 1.93 \times 10^{-6}\right) \approx -20.0 \tag{9.1.20}$$

so that the best decision in this case is to choose $\theta = 0.1$.

As more information about the distribution $p(v, \theta)$ becomes available the posterior $p(\theta|\mathcal{V})$ becomes increasingly peaked, aiding our decision making process.

9.1.3 A continuum of parameters

In section(9.1.1) we considered only three possible values for θ. Here we discuss a continuum of parameters.

Using a flat prior

We first examine the case of a 'flat' or *uniform* prior $p(\theta) = k$ for some constant k. For continuous variables, normalisation requires

$$\int p(\theta)d\theta = 1 \tag{9.1.21}$$

Since $0 \leq \theta \leq 1$,

$$\int_0^1 p(\theta)d\theta = k = 1 \tag{9.1.22}$$

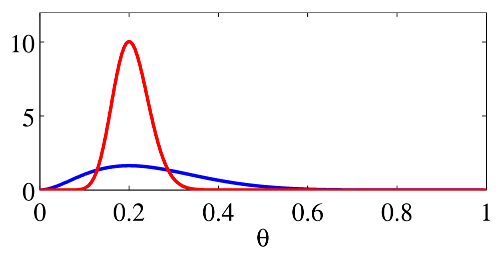


Figure 9.3: Posterior $p(\theta|\mathcal{V})$ assuming a flat prior on θ. (red) $N_H = 2$, $N_T = 8$ and (blue) $N_H = 20$, $N_T = 80$. In both cases, the most probable state of the posterior is 0.2, which makes intuitive sense, since the fraction of Heads to Tails in both cases is 0.2. Where there is more data, the posterior is more certain and sharpens around the most probable value. The Maximum A Posteriori setting is $\theta = 0.2$ in both cases, this being the value of θ for which the posterior attains its highest value.

Repeating the previous calculations with this flat continuous prior, we have

$$p(\theta|\mathcal{V}) = \frac{1}{c}\theta^{N_H}(1-\theta)^{N_T} \tag{9.1.23}$$

where c is a constant to be determined by normalisation,

$$c = \int_0^1 \theta^{N_H}(1-\theta)^{N_T} d\theta \equiv B(N_H+1, N_T+1) \tag{9.1.24}$$

where $B(\alpha, \beta)$ is the Beta function.

Definition 85 (conjugacy)**.** If the posterior is of the same parametric form as the prior, then we call the prior the conjugate distribution for the likelihood distribution.

Using a conjugate prior

Determining the normalisation constant of a continuous distribution requires that the integral of the unnormalised posterior can be carried out. For the coin tossing case, it is clear that if the prior is of the form of a Beta distribution, then the posterior will be of the same parametric form:

$$p(\theta) = \frac{1}{B(\alpha,\beta)}\theta^{\alpha-1}(1-\theta)^{\beta-1} \tag{9.1.25}$$

the posterior is

$$p(\theta|\mathcal{V}) \propto \theta^{\alpha-1}(1-\theta)^{\beta-1}\theta^{N_H}(1-\theta)^{N_T} \tag{9.1.26}$$

so that

$$p(\theta|\mathcal{V}) = \frac{1}{B(\alpha+N_H, \beta+N_T)}\theta^{\alpha+N_H-1}(1-\theta)^{\beta+N_T-1} \equiv B(\theta|\alpha+N_H, \beta+N_T) \tag{9.1.27}$$

The prior and posterior are of the same form (both Beta distributions) but simply with different parameters. Hence the Beta distribution is ‘conjugate’ to the Binomial distribution.

9.1.4 Decisions based on continuous intervals

The result of a coin tossing experiment is $N_H = 2$ heads and $N_T = 8$ tails. You now need to make a decision : you win 10 dollars if your guess that the coin is more likely to come up heads than tails is correct. If your guess is incorrect, you lose a million dollars. What is your decision? (Assume an uninformative prior).

We need two quantities, θ for our guess and θ^0 for the truth. Then the utility of saying Heads is

$$U(\theta > 0.5, \theta^0 > 0.5)p(\theta^0 > 0.5|\mathcal{V}) + U(\theta > 0.5, \theta^0 < 0.5)p(\theta^0 < 0.5|\mathcal{V}) \tag{9.1.28}$$

In the above,

$$p(\theta^0 < 0.5|\mathcal{V}) = \int_0^{0.5} p(\theta^0|\mathcal{V})d\theta^0 \tag{9.1.29}$$

$$= \frac{1}{B(\alpha + N_H, \beta + N_T)} \int_0^{0.5} \theta^{\alpha+N_H-1} (1-\theta)^{\beta+N_T-1} d\theta \tag{9.1.30}$$

$$\equiv I_{0.5}(\alpha + N_H, \beta + N_T) \tag{9.1.31}$$

where $I_x(a, b)$ is the *regularised incomplete Beta function*. For the former case of $N_H = 2, N_T = 8$, under a flat prior,

$$p(\theta^0 < 0.5|\mathcal{V}) = I_{0.5}(N_H + 1, N_T + 1) = 0.9673 \tag{9.1.32}$$

Since the events are exclusive, $p(\theta^0 \geq 0.5|\mathcal{V}) = 1 - 0.9673 = 0.0327$.

Hence the expected utility of saying heads is more likely is

$$10 \times 0.0327 - 1000000 \times 0.9673 = -9.673 \times 10^5. \tag{9.1.33}$$

Similarly, the utility of saying tails is more likely is

$$10 \times 0.9673 - 1000000 \times 0.0327 = -3.269 \times 10^4. \tag{9.1.34}$$

So we are better off taking the decision that the coin is more likely to come up tails.

If we modify the above so that we lose 100 million dollars if we guess tails when in fact it as heads, the expected utility of saying tails would be -3.27×10^6 in which case we would be better of saying heads. In this case, even though we are more confident that the coin is likely to come up tails, we would pay such a penalty of making a mistake in saying tails, that it is fact better to say heads.

9.2 Maximum A Posteriori and Maximum Likelihood

9.2.1 Summarising the posterior

Definition 86 (Maximum Likelihood and Maximum a Posteriori)**.** Maximum Likelihood sets parameter θ, given data $\mathcal{V}$, using

$$\theta^{ML} = \underset{\theta}{\operatorname{argmax}}\, p(\mathcal{V}|\theta) \tag{9.2.1}$$

Maximum A Posteriori uses that setting θ that maximises the posterior distribution of the parameter,

$$\theta^{MAP} = \underset{\theta}{\operatorname{argmax}}\, p(\mathcal{V}|\theta)p(\theta) \tag{9.2.2}$$

where $p(\theta)$ is the prior distribution.

A crude summary of the posterior is given by a distribution with all its mass in a single most likely state, $\delta\left(\theta, \theta^{MAP}\right)$. In making such an approximation, potentially useful information concerning the reliability of the parameter estimate is lost. In contrast the full posterior reflects our beliefs about the range of possibilities and their associated credibilities.

One can motivate MAP from a decision theoretic perspective. If we assume a utility that is zero for all but the correct θ,

$$U(\theta_{true}, \theta) = \mathbb{I}\left[\theta_{true} = \theta\right] \tag{9.2.3}$$

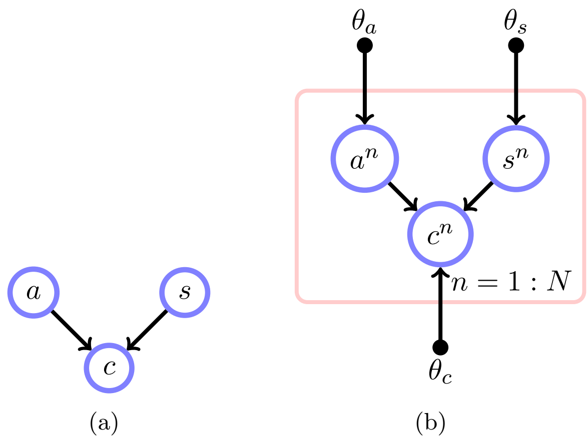


Figure 9.4: **(a)**: A model for the relationship between lung Cancer, Asbestos exposure and Smoking. **(b)**: Plate notation replicating the observed n datapoints and placing priors over the CPTs, tied across all datapoints.

then the expected utility of θ is

$$U(\theta) = \sum_{\theta_{true}} \mathbb{I}\left[\theta_{true} = \theta\right] p(\theta_{true}|\mathcal{V}) = p(\theta|\mathcal{V}) \tag{9.2.4}$$

This means that the maximum utility decision is to return that θ with the highest posterior value.

When a 'flat' prior $p(\theta) = \mathsf{const.}$ is used the MAP parameter assignment is equivalent to the Maximum Likelihood setting

$$\theta^{ML} = \underset{\theta}{\operatorname{argmax}}\ p(\mathcal{V}|\theta) \tag{9.2.5}$$

The term Maximum Likelihood refers to the parameter θ for which the observed data is most likely to be generated by the model.

Since the logarithm is a strictly increasing function, then for a positive function $f(\theta)$

$$\theta_{opt} = \underset{\theta}{\operatorname{argmax}}\ f(\theta) \Leftrightarrow \theta_{opt} = \underset{\theta}{\operatorname{argmax}}\ \log f(\theta) \tag{9.2.6}$$

so that the MAP parameters can be found either by optimising the MAP objective or, equivalently, its logarithm,

$$\log p(\theta|\mathcal{V}) = \log p(\mathcal{V}|\theta) + \log p(\theta) - \log p(\mathcal{V}) \tag{9.2.7}$$

where the normalisation constant, $p(\mathcal{V})$, is not a function of θ.

The log likelihood is convenient since under the i.i.d. assumption it is a summation of data terms,

$$\log p(\theta|\mathcal{V}) = \sum_n \log p(v^n|\theta) + \log p(\theta) - \log p(\mathcal{V}) \tag{9.2.8}$$

so that quantities such as derivatives of the log-likelihood *w.r.t.* θ are straightforward to compute.

Example 36. In the coin-tossing experiment of section(9.1.1) the ML setting is $\theta = 0.2$ in both $N_H = 2, N_T = 8$ and $N_H = 20, N_T = 80$.

9.2.2 Maximum likelihood and the empirical distribution

Given a dataset of discrete variables $\mathcal{X} = \left\{x^1, \ldots, x^N\right\}$ we define the empirical distribution as

$$q(x) = \frac{1}{N} \sum_{n=1}^{N} \mathbb{I}\left[x = x^n\right] \tag{9.2.9}$$

a	s	c
1	1	1
1	0	0
0	1	1
0	1	0
1	1	1
0	0	0
1	0	1

Figure 9.5: A database containing information about the Asbestos exposure (1 signifies exposure), being a Smoker (1 signifies the individual is a smoker), and lung Cancer (1 signifies the individual has lung Cancer). Each row contains the information for an individual, so that there are 7 individuals in the database.

in the case that x is a vector of variables,

$$\mathbb{I}[x = x^n] = \prod_i \mathbb{I}[x_i = x_i^n] \tag{9.2.10}$$

The Kullback-Leibler divergence between the empirical distribution $q(x)$ and a distribution $p(x)$ is

$$\mathrm{KL}(q|p) = \langle \log q(x) \rangle_{q(x)} - \langle \log p(x) \rangle_{q(x)} \tag{9.2.11}$$

Our interest is the functional dependence of $\mathrm{KL}(q|p)$ on p. Since the entropic term $\langle \log q(x) \rangle_{q(x)}$ is independent of $p(x)$ we may consider this constant and focus on the second term alone. Hence

$$\mathrm{KL}(q|p) = -\langle \log p(x) \rangle_{q(x)} + \mathsf{const.} = -\frac{1}{N}\sum_{n=1}^{N} \log p(x^n) + \mathsf{const.} \tag{9.2.12}$$

We recognise $\sum_{n=1}^{N} \log p(x^n)$ as the log likelihood under the model $p(x)$, assuming that the data is i.i.d. This means that setting parameters by maximum likelihood is equivalent to setting parameters by minimising the Kullback-Leibler divergence between the empirical distribution and the parameterised distribution. In the case that $p(x)$ is unconstrained, the optimal choice is to set $p(x) = q(x)$, namely the maximum likelihood optimal distribution corresponds to the empirical distribution.

9.2.3 Maximum likelihood training of belief networks

Consider the following model of the relationship between exposure to asbestos (a), being a smoker (s) and the incidence of lung cancer (c)

$$p(a, s, c) = p(c|a, s)p(a)p(s) \tag{9.2.13}$$

which is depicted in fig(9.4a). Each variable is binary, $\mathrm{dom}(a) = \{0, 1\}$, $\mathrm{dom}(s) = \{0, 1\}$, $\mathrm{dom}(c) = \{0, 1\}$. We assume that there is no direct relationship between Smoking and exposure to Asbestos. This is the kind of assumption that we may be able to elicit from medical experts. Furthermore, we assume that we have a list of patient records, fig(9.5), where each row represents a patient's data. To learn the table entries $p(c|a, s)$ we can do so by counting the number of c is in state 1 for each of the 4 parental states of a and s:

$$\begin{array}{ll} p(c=1|a=0,s=0)=0, & p(c=1|a=0,s=1)=0.5 \\ p(c=1|a=1,s=0)=0.5 & p(c=1|a=1,s=1)=1 \end{array} \tag{9.2.14}$$

Similarly, based on counting, $p(a = 1) = 4/7$, and $p(s = 1) = 4/7$. These three CPTs then complete the full distribution specification.

Setting the CPT entries in this way by counting the relative number of occurrences corresponds mathematically to maximum likelihood learning under the i.i.d. assumption, as we show below.

Maximum likelihood corresponds to counting

For a BN there is a constraint on the form of $p(x)$, namely

$$p(x) = \prod_{i=1}^{K} p(x_i|\mathrm{pa}(x_i)) \tag{9.2.15}$$

To compute the Maximum Likelihood setting of each term $p(x_i|\text{pa}(x_i))$, as shown in section(9.2.2), we can equivalently minimise the Kullback-Leibler divergence between the empirical distribution $q(x)$ and $p(x)$. For the BN $p(x)$, and empirical distribution $q(x)$ we have

$$\text{KL}(q|p) = -\left\langle \sum_{i=1}^{K} \log p(x_i|\text{pa}(x_i)) \right\rangle_{q(x)} + \text{const.} = -\sum_{i=1}^{K} \langle \log p(x_i|\text{pa}(x_i)) \rangle_{q(x_i,\text{pa}(x_i))} + \text{const.} \tag{9.2.16}$$

This follows using the general result

$$\langle f(\mathcal{X}_i) \rangle_{q(\mathcal{X})} = \langle f(\mathcal{X}_i) \rangle_{q(\mathcal{X}_i)} \tag{9.2.17}$$

which says that if the function f only depends on a subset of the variables, we only need to know the marginal distribution of this subset of variables in order to carry out the average.

Since $q(x)$ is fixed, we can add on entropic terms in q and equivalently mimimize

$$\text{KL}(q|p) = \sum_{i=1}^{K} \left[\langle \log q(x_i|\text{pa}(x_i)) \rangle_{q(x_i,\text{pa}(x_i))} - \langle \log p(x_i|\text{pa}(x_i)) \rangle_{q(x_i,\text{pa}(x_i))} \right] \tag{9.2.18}$$

$$= \sum_{i=1}^{K} \langle \text{KL}(q(x_i|\text{pa}(x_i))|p(x_i|\text{pa}(x_i))) \rangle_{q(\text{pa}(x_i))} \tag{9.2.19}$$

The final line is a positive weighted sum of individual Kullback-Leibler divergences. The minimal Kullback-Leibler setting, and that which corresponds to Maximum Likelihood, is therefore

$$p(x_i|\text{pa}(x_i)) = q(x_i|\text{pa}(x_i)) \tag{9.2.20}$$

In terms of the original data, this is

$$p(x_i = \mathsf{s}|\text{pa}(x_i) = \mathsf{t}) \propto \sum_{n=1}^{N} \mathbb{I}[x_i^n = \mathsf{s}] \prod_{x_j \in \text{pa}(x_i)} \mathbb{I}\left[x_j^n = \mathsf{t}^j\right] \tag{9.2.21}$$

This expression corresponds to the intuition that the table entry $p(x_i|\text{pa}(x_i))$ can be set by counting the number of times the state $\{x_i = \mathsf{s}, \text{pa}(x_i) = \mathsf{t}\}$ occurs in the dataset (where t is a vector of parental states). The table is then given by the relative number of counts of being in state s compared to the other states s', for fixed joint parental state t.

An alternative method to derive this intuitive result is to use Lagrange multipliers, see exercise(120). For reader less comfortable with the above Kullback-Leibler derivation, a more direct example is given below which makes use of the notation

$$\sharp(x_1 = s_1, x_2 = s_2, x_3 = s_3, \ldots) \tag{9.2.22}$$

to denote the number of times that states $x_1 = s_1, x_2 = s_2, x_3 = s_3, \ldots$ occur together in the training data.

Example 37. We wish to learn the table entries of the distribution $p(x_1, x_2, x_3) = p(x_1|x_2, x_3)p(x_2)p(x_3)$. We address here how to find the CPT entry $p(x_1 = 1|x_2 = 1, x_3 = 0)$ using Maximum Likelihood. For i.i.d. data, the contribution from $p(x_1|x_2, x_3)$ to the log likelihood is

$$\sum_n \log p(x_1^n|x_2^n, x_3^n)$$

The number of times $p(x_1 = 1|x_2 = 1, x_3 = 0)$ occurs in the log likelihood is $\sharp(x_1 = 1, x_2 = 1, x_3 = 0)$, the number of such occurrences in the training set. Since (by the normalisation constraint) $p(x_1 = 0|x_2 =$

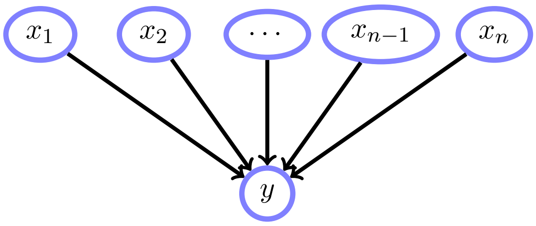


Figure 9.6: A variable y with a large number of parents $x_1, \ldots, x_n$ requires the specification of an exponentially large number of entries in the conditional probability $p(y|x_1, \ldots, x_n)$. One solution to this difficulty is to parameterise the conditional, $p(y|x_1, \ldots, x_n, \theta)$.

$1, x_3 = 0) = 1 - p(x_1 = 1|x_2 = 1, x_3 = 0)$, the total contribution of $p(x_1 = 1|x_2 = 1, x_3 = 0)$ to the log likelihood is

$$\sharp\,(x_1 = 1, x_2 = 1, x_3 = 0) \log p(x_1 = 1|x_2 = 1, x_3 = 0) \\ + \sharp\,(x_1 = 0, x_2 = 1, x_3 = 0) \log\left(1 - p(x_1 = 1|x_2 = 1, x_3 = 0)\right) \quad (9.2.23)$$

Using $\theta \equiv p(x_1 = 1|x_2 = 1, x_3 = 0)$ we have

$$\sharp\,(x_1 = 1, x_2 = 1, x_3 = 0) \log\theta + \sharp\,(x_1 = 0, x_2 = 1, x_3 = 0) \log(1 - \theta) \quad (9.2.24)$$

Differentiating the above expression *w.r.t.* θ and equating to zero gives

$$\frac{\sharp\,(x_1 = 1, x_2 = 1, x_3 = 0)}{\theta} - \frac{\sharp\,(x_1 = 0, x_2 = 1, x_3 = 0)}{1 - \theta} = 0 \quad (9.2.25)$$

The solution for optimal θ is then

$$p(x_1 = 1|x_2 = 1, x_3 = 0) = \frac{\sharp\,(x_1 = 1, x_2 = 1, x_3 = 0)}{\sharp\,(x_1 = 1, x_2 = 1, x_3 = 0) + \sharp\,(x_1 = 0, x_2 = 1, x_3 = 0)}, \quad (9.2.26)$$

corresponding to the intuitive counting procedure.

Conditional probability functions

Consider a binary variable y with n binary parental variables, $\mathbf{x} = (x_1, \ldots, x_n)$. There are 2^n entries in the CPT of $p(y|x)$ so that it is infeasible to explicitly store these entries for even moderate values of n. To reduce the complexity of this CPT we may constrain the form of the table. For example, one could use a function

$$p(y = 1|\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-\mathbf{w}^\mathsf{T}\mathbf{x}}} \quad (9.2.27)$$

where we only need to specify the n-dimensional parameter vector $\mathbf{w}$.

In this case, rather than using Maximum Likelihood to learn the entries of the CPTs directly, we instead learn the value of the parameter $\mathbf{w}$. Since the number of parameters in $\mathbf{w}$ is small (n, compared with 2^n in the unconstrained case), we also have some hope that with a small number of training examples we can learn a reliable value for $\mathbf{w}$.

Example 38. Consider the following 3 variable model $p(x_1, x_2, x_3) = p(x_1|x_2, x_3)p(x_2)p(x_3)$, where $x_i \in \{0, 1\}, i = 1, 2, 3$. We assume that the CPT is parameterised using

$$p(x_1 = 1|x_2, x_3, \theta) \equiv e^{-\theta_1^2 - \theta_2^2 (x_2 - x_3)^2} \quad (9.2.28)$$

One may verify that the above probability is always positive and lies between 0 and 1. Due to normalisation, we must have

$$p(x_1 = 0|x_2, x_3) = 1 - p(x_1 = 1|x_2, x_3) \quad (9.2.29)$$

For unrestricted $p(x_2)$ and $p(x_3)$, the Maximum Likelihood setting is $p(x_2 = 1) \propto \sharp(x_2 = 1)$, and $p(x_3 = 1) \propto \sharp(x_3 = 1)$. The contribution to the log likelihood from the term $p(x_1|x_2, x_3, \theta)$, assuming i.i.d. data, is

$$L(\theta_1, \theta_2) = \sum_{n=1}^{N} \mathbb{I}[x_1^n = 1] \left(-\theta_1^2 - \theta_2^2(x_2^n - x_3^n)^2\right) + \mathbb{I}[x_1^n = 0] \log\left(1 - e^{-\theta_1^2 - \theta_2^2(x_2^n - x_3^n)^2}\right) \tag{9.2.30}$$

This objective function needs to be optimised numerically to find the best θ_1 and θ_2. The gradient is

$$\frac{dL}{d\theta_1} = \sum_{n=1}^{N} -2\mathbb{I}[x_1^n = 1]\,\theta_1 + 2\mathbb{I}[x_1^n = 0] \frac{\theta_1 e^{-\theta_1^2 - \theta_2^2(x_2^n - x_3^n)^2}}{1 - e^{-\theta_1^2 - \theta_2^2(x_2^n - x_3^n)^2}} \tag{9.2.31}$$

$$\frac{dL}{d\theta_2} = \sum_{n=1}^{N} -2\mathbb{I}[x_1^n = 1]\,\theta_2\,(x_2^n - x_3^n)^2 + 2\theta_2\mathbb{I}[x_1^n = 0] \frac{(x_2^n - x_3^n)^2 e^{-\theta_1^2 - \theta_2^2(x_2^n - x_3^n)^2}}{1 - e^{-\theta_1^2 - \theta_2^2(x_2^n - x_3^n)^2}} \tag{9.2.32}$$

The gradient can be used as part of a standard optimisation procedure (such as conjugate gradients, see Appendix (A)) to aid finding the Maximum Likelihood parameters θ_1, θ_2.

9.3 Bayesian Belief Network Training

An alternative to Maximum Likelihood training of a BN is to use a Bayesian approach in which we maintain a distribution over parameters. We continue with the Asbestos, Smoking, Cancer scenario,

$$p(a, c, s) = p(c|a, s)p(a)p(s) \tag{9.3.1}$$

which can be represented as a Belief Network, fig(9.4a). So far we've only specified the independence structure, but not the entries of the tables $p(c|a, s)$, $p(a)$, $p(s)$. Given a set of visible observations, $\mathcal{V} = \{(a^n, s^n, c^n), n = 1, \dots, N\}$, we would like to learn appropriate distributions for the table entries.

To begin we need a notation for the table entries. With all variables binary we have parameters such as

$$p(a = 1|\theta_a) = \theta_a, \quad p(c = 1|a = 0, s = 1, \theta_c) = \theta_c^{0,1} \tag{9.3.2}$$

and similarly for the remaining parameters $\theta_c^{1,1}, \theta_c^{0,0}, \theta_c^{1,0}$. For our example, the parameters are

$$\theta_a, \theta_s, \underbrace{\theta_c^{0,0}, \theta_c^{0,1}, \theta_c^{1,0}, \theta_c^{1,1}}_{\theta_c} \tag{9.3.3}$$

9.3.1 Global and local parameter independence

In Bayesian learning of BNs, we need to specify a prior on the joint table entries. Since in general dealing with multi-dimensional continuous distributions is computationally problematic, it is useful to specify only uni-variate distributions in the prior. As we show below, this has a pleasing consequence that for i.i.d. data the posterior also factorises into uni-variate distributions.

Global parameter independence

A convenient assumption is that the prior factorises over parameters. For our Asbestos, Smoking, Cancer example, we assume

$$p(\theta_a, \theta_s, \theta_c) = p(\theta_a)p(\theta_s)p(\theta_c) \tag{9.3.4}$$

Assuming the data is i.i.d., we then have the joint model

$$p(\theta_a, \theta_s, \theta_c, \mathcal{V}) = p(\theta_a)p(\theta_s)p(\theta_c) \prod_n p(a^n|\theta_a)p(s^n|\theta_s)p(c^n|s^n, a^n, \theta_c) \tag{9.3.5}$$

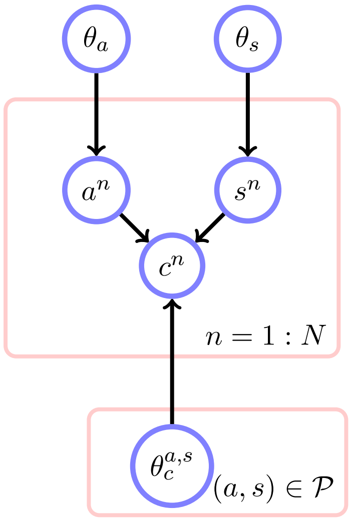


Figure 9.7: A Bayesian parameter model for the relationship between lung Cancer, Asbestos exposure and Smoking with factorised parameter priors. The global parameter independence assumption means that the prior over tables factorises into priors over each conditional probability table. The local independence assumption, which in this case comes into effect only for $p(c|a, s)$, means that $p(\theta_c)$ factorises in $\prod_{a,s\in\mathcal{P}} p(\theta_c^{a,s})$, where $\mathcal{P} = \{(0, 0), (0, 1), (1, 0), (1, 1)\}$.

the Belief Network for which is given in fig(9.7.) Learning then corresponds to inference of

$$p(\theta_a, \theta_s, \theta_c|\mathcal{V}) = \frac{p(\mathcal{V}|\theta_a, \theta_s, \theta_c)p(\theta_a, \theta_s, \theta_c)}{p(\mathcal{V})} = \frac{p(\mathcal{V}|\theta_a, \theta_s, \theta_c)p(\theta_a)p(\theta_s)p(\theta_c)}{p(\mathcal{V})} \tag{9.3.6}$$

A convenience of the factorised prior for a BN is that the posterior also factorises, since

$$\begin{aligned} p(\theta_a, \theta_s, \theta_c|\mathcal{V}) &\propto p(\theta_a, \theta_s, \theta_c, \mathcal{V}) \\ &\propto \left\{p(\theta_a)\prod_n p(a^n|\theta_a)\right\}\left\{p(\theta_s)\prod_n p(s^n|\theta_s)\right\}\left\{p(\theta_c)\prod_n p(c^n|s^n, a^n, \theta_c)\right\} \\ &= p(\theta_a|\mathcal{V}_a)p(\theta_s|\mathcal{V}_s)p(\theta_c|\mathcal{V}) \end{aligned} \tag{9.3.7}$$

so that one can consider each parameter posterior separately. In this case, 'learning' involves computing the posterior distributions $p(\theta_i|\mathcal{V}_i)$ where $\mathcal{V}_i$ is the set of training data restricted to the family of variable i.

The global independence assumption conveniently results in a posterior distribution that factorises over the conditional tables. However, the parameter θ_c is itself 4 dimensional. To simplify this we need to make a further assumption as to the structure of each local table.

Local parameter independence

If we further assume that the prior for the table factorises over all states a, c:

$$p(\theta_c) = p(\theta_c^{0,0})p(\theta_c^{1,0})p(\theta_c^{0,1})p(\theta_c^{1,1}) \tag{9.3.8}$$

then the posterior

$$p(\theta_c|\mathcal{V}) \propto p(\mathcal{V}|\theta_c)p(\theta_c^{0,0})p(\theta_c^{1,0})p(\theta_c^{0,1})p(\theta_c^{1,1}) \tag{9.3.9}$$

$$= \underbrace{\left[\theta_c^{0,0}\right]^{\sharp(a=0,s=0)} p(\theta_c^{0,0})}_{p(\theta_c^{0,0}|\mathcal{V})} \underbrace{\left[\theta_c^{0,1}\right]^{\sharp(a=0,s=1)} p(\theta_c^{0,1})}_{p(\theta_c^{0,1}|\mathcal{V})} \underbrace{\left[\theta_c^{1,0}\right]^{\sharp(a=1,s=0)} p(\theta_c^{1,0})}_{p(\theta_c^{1,0}|\mathcal{V})} \underbrace{\left[\theta_c^{1,1}\right]^{\sharp(a=1,s=1)} p(\theta_c^{1,1})}_{p(\theta_c^{1,1}|\mathcal{V})} \tag{9.3.10}$$

so that the posterior also factorises over the parental states of the local conditional table.

Posterior marginal table

A marginal probability table is given by, for example,

$$p(c = 1|a = 1, s = 0, \mathcal{V}) = \int_{\theta_c} p(c = 1|a = 1, s = 0, \theta_c^{1,0})p(\theta_c|\mathcal{V}) \tag{9.3.11}$$

The integral over all the other tables in equation (9.3.11) is unity, and we are left with

$$p(c = 1|a = 1, s = 0, \mathcal{V}) = \int_{\theta_c^{1,0}} p(c = 1|a = 1, s = 0, \theta_c^{1,0})p(\theta_c^{1,0}|\mathcal{V}) \tag{9.3.12}$$

9.3.2 Learning binary variable tables using a Beta prior

We continue the example of section(9.3.1) where all variables are binary, but using a continuous valued table prior. The simplest case is to start with $p(a|\theta_a)$ since this requires only a univariate prior distribution $p(\theta_a)$. The likelihood depends on the table variable via

$$p(a = 1|\theta_a) = \theta_a \tag{9.3.13}$$

so that the total likelihood term is

$$\theta_a^{\sharp(a=1)} (1 - \theta_a)^{\sharp(a=0)} \tag{9.3.14}$$

The posterior is therefore

$$p(\theta_a|\mathcal{V}_a) \propto p(\theta_a)\theta_a^{\sharp(a=1)} (1 - \theta_a)^{\sharp(a=0)} \tag{9.3.15}$$

This means that if the prior is also of the form $\theta_a^\alpha (1 - \theta_a)^\beta$ then conjugacy will hold, and the mathematics of integration will be straightforward. This suggests that the most convenient choice is a Beta distribution,

$$p(\theta_a) = B(\theta_a|\alpha_a, \beta_a) = \frac{1}{B(\alpha_a, \beta_a)}\theta_a^{\alpha_a-1} (1 - \theta_a)^{\beta_a-1} \tag{9.3.16}$$

for which the posterior is also a Beta distribution:

$$p(\theta_a|\mathcal{V}_a) = B(\theta_a|\alpha_a + \sharp(a = 1), \beta_a + \sharp(a = 0)) \tag{9.3.17}$$

The marginal table is given by

$$p(a = 1|\mathcal{V}_a) = \int_{\theta_a} p(\theta_a|\mathcal{V}_a)\theta_a = \frac{\alpha_a + \sharp(a = 1)}{\alpha_a + \sharp(a = 1) + \beta_a + \sharp(a = 0)} \tag{9.3.18}$$

using the result for the mean of a Beta distribution, definition(71).

The situation for the table $p(c|a, s)$ is slightly more complex since we need to specify a prior for each of the parental tables. As above, this is most convenient if we specify a Beta prior, one for each of the (four) parental states. Let's look at a specific table

$$p(c = 1|a = 1, s = 0) \tag{9.3.19}$$

Assuming the local independence property, we have $p(\theta_c^{1,0}|\mathcal{V}_c)$ given by

$$B\left(\theta_c^{1,0}|\alpha_c(a = 1, s = 0) + \sharp(c = 1, a = 1, s = 0), \beta_c(a = 1, s = 0) + \sharp(c = 0, a = 1, s = 0)\right) \tag{9.3.20}$$

As before, the marginal probability table is then given by

$$p(c = 1|a = 1, s = 0, \mathcal{V}_c) = \frac{\alpha_c(a = 1, s = 0) + \sharp(c = 1, a = 1, s = 0)}{\alpha_c(a = 1, s = 0) + \beta_c(a = 1, s = 0) + \sharp(a = 1, s = 0)} \tag{9.3.21}$$

since $\sharp(a = 1, s = 0) = \sharp(c = 0, a = 1, s = 0) + \sharp(c = 1, a = 1, s = 0)$.

The prior parameters $\alpha_c(a, s)$ are called *hyperparameters*. If one had no preference, one could set all of the $\alpha_c(a, s)$ to be equal to the same value α and similarly for β. A complete ignorance prior would correspond to setting $\alpha = \beta = 1$, see fig(8.7).

No data limit $N \to 0$ In the limit of no data, the marginal probability table corresponds to the prior, which is given in this case by

$$p(c = 1|a = 1, s = 0) = \frac{\alpha_c(a = 1, s = 0)}{\alpha_c(a = 1, s = 0) + \beta_c(a = 1, s = 0)} \tag{9.3.22}$$

For a flat prior $\alpha = \beta = 1$ for all states a, c, this would give a prior probability of $p(c = 1|a = 1, s = 0) = 0.5$.

Infinite data limit $N \to \infty$ In this limit the marginal probability tables are dominated by the data counts, since these will typically grow in proportion to the size of the dataset. This means that in the infinite (or very large) data limit,

$$p(c = 1|a = 1, s = 0, \mathcal{V}) \to \frac{\sharp(c = 1, a = 1, s = 0)}{\sharp(c = 1, a = 1, s = 0) + \sharp(c = 0, a = 1, s = 0)} \tag{9.3.23}$$

which corresponds to the Maximum Likelihood solution.

This effect that the large data limit of a Bayesian procedure corresponds to the Maximum Likelihood solution is general unless the prior has a pathologically strong effect.

Example 39 (Asbestos-Smoking-Cancer)**.**

Consider the binary variable network

$$p(c, a, s) = p(c|a, s)p(a)p(s) \tag{9.3.24}$$

The data $\mathcal{V}$ is given in fig(9.5). Using a flat Beta prior $\alpha = \beta = 1$ for all conditional probability tables, the marginal posterior tables are given by

$$p(a = 1|\mathcal{V}) = \frac{1 + \sharp(a = 1)}{2 + N} = \frac{1 + 4}{2 + 7} = \frac{5}{9} \approx 0.556 \tag{9.3.25}$$

By comparison, the Maximum Likelihood setting is $4/7 = 0.571$. The Bayesian result is a little more cautious than the Maximum Likelihood, which squares with our prior belief that any setting of the probability is equally likely, pulling the posterior towards 0.5.

Similarly,

$$p(s = 1|\mathcal{V}) = \frac{1 + \sharp(s = 1)}{2 + N} = \frac{1 + 4}{2 + 7} = \frac{5}{9} \approx 0.556 \tag{9.3.26}$$

and

$$p(c = 1|a = 1, s = 1, \mathcal{V}) = \frac{1 + \sharp(c = 1, a = 1, s = 1)}{2 + \sharp(c = 1, a = 1, s = 1) + \sharp(c = 0, a = 1, s = 1)} = \frac{1 + 2}{2 + 2} = \frac{3}{4} \tag{9.3.27}$$

$$p(c = 1|a = 1, s = 0, \mathcal{V}) = \frac{1 + \sharp(c = 1, a = 1, s = 0)}{2 + \sharp(c = 1, a = 1, s = 0) + \sharp(c = 0, a = 1, s = 0)} = \frac{1 + 1}{2 + 1} = \frac{2}{3} \tag{9.3.28}$$

$$p(c = 1|a = 0, s = 1, \mathcal{V}) = \frac{1 + \sharp(c = 1, a = 0, s = 1)}{2 + \sharp(c = 1, a = 0, s = 1) + \sharp(c = 0, a = 0, s = 1)} = \frac{1 + 1}{2 + 2} = \frac{1}{2} \tag{9.3.29}$$

$$p(c = 1|a = 0, s = 0, \mathcal{V}) = \frac{1 + \sharp(c = 1, a = 0, s = 0)}{2 + \sharp(c = 1, a = 0, s = 0) + \sharp(c = 0, a = 0, s = 0)} = \frac{1 + 0}{2 + 1} = \frac{1}{3} \tag{9.3.30}$$

9.3.3 Learning multivariate discrete tables using a Dirichlet prior

The natural generalisation to more than two-state variables is given by using a Dirichlet prior, again assuming i.i.d. data and the local and global parameter prior independencies. Since under the global parameter independence assumption the posterior factorises over variables (as in equation (9.3.7)), we can concentrate on the posterior of a single variable.

No Parents

Let's consider the contribution of a variable v with $\mathrm{dom}(v) = \{1, \ldots I\}$. The contribution to the posterior from a datapoint v^n is

$$p(v^n|\theta) = \prod_{i=1}^{I} \theta_i^{\mathbb{I}[v^n=i]}, \qquad \sum_{i=1}^{I} \theta_i = 1 \tag{9.3.31}$$

so that the posterior is proportional to

$$p(\theta) \prod_{n=1}^{N} \prod_{i=1}^{I} \theta_i^{\mathbb{I}[v^n=i]} = p(\theta) \prod_{i=1}^{I} \theta_i^{\sum_{n=1}^{N} \mathbb{I}[v^n=i]} \tag{9.3.32}$$

For a Dirichlet prior distribution with hyperparameters $\mathbf{u}$

$$p(\theta) \propto \prod_{i=1}^{I} \theta_i^{u_i - 1} \tag{9.3.33}$$

Using this prior the posterior becomes

$$p(\theta|\mathcal{V}) \propto \prod_{i=1}^{I} \theta_i^{u_i-1} \prod_{i=1}^{I} \theta_i^{\sum_{n=1}^{N} \mathbb{I}[v^n=i]} = \prod_{i=1}^{I} \theta_i^{u_i - 1 + \sum_{n=1}^{N} \mathbb{I}[v^n=i]} \tag{9.3.34}$$

which means that the posterior is given by

$$p(\theta|\mathcal{V}) = \text{Dirichlet}\left(\theta|\mathbf{u} + \mathbf{c}\right) \tag{9.3.35}$$

where $\mathbf{c}$ is a count vector with components

$$c_i = \sum_{n=1}^{N} \mathbb{I}\left[v^n = i\right] \tag{9.3.36}$$

being the number of times state i was observed in the training data.

The marginal table is given by integrating

$$p(v = i|\mathcal{V}) = \int_\theta p(v = i|\theta) p(\theta|\mathcal{V}) = \int_{\theta_i} \theta_i p(\theta_i|\mathcal{V}) \tag{9.3.37}$$

Since the single-variable marginal distribution of a Dirichlet is a Beta distribution, section(8.5), the marginal table is the mean of a Beta distribution. Given that the marginal $p(\theta|\mathcal{V})$ is Beta distribution with parameters $\alpha = u_i + c_i$, $\beta = \sum_{j \neq i} u_j + c_j$, the marginal table is given by

$$p(v = i|\mathcal{V}) = \frac{u_i + c_i}{\sum_j u_j + c_j} \tag{9.3.38}$$

which generalises the binary state formula equation (9.3.18).

9.3.4 Parents

To deal with the general case of a variable v with parents $\mathrm{pa}(v)$ we denote the probability of v being in state i, conditioned on the parents being in state j as

$$p(v = \mathsf{i}|\mathrm{pa}(v) = \mathsf{j}, \theta) = \theta_{\mathsf{i}}(v; \mathsf{j}) \tag{9.3.39}$$

where $\sum_{\mathsf{i}} \theta_{\mathsf{i}}(v; \mathsf{j}) = 1$. This forms the components of a vector $\boldsymbol{\theta}(v; \mathsf{j})$. Note that if v has K parents then the number of states j will be exponential in K.

Local (state) independence means

$$p(\boldsymbol{\theta}(v)) = \prod_{\mathsf{j}} p(\boldsymbol{\theta}(v; \mathsf{j})) \tag{9.3.40}$$

And global independence means

$$p(\boldsymbol{\theta}) = \prod_{v} p(\boldsymbol{\theta}(v)) \tag{9.3.41}$$

where $\boldsymbol{\theta} = (\boldsymbol{\theta}(v), v = 1, \ldots, V)$ represents the combined table of all the variables. We drop the explicit sans-serif font on the states from here on in.

Parameter posterior

Thanks to the global parameter independence the posterior distribution over the tables θ factorises, with one posterior table per variable. Each posterior table for a variable v depends only on the information local to the family of each variable $\mathcal{D}(v)$. Assuming a Dirichlet distribution prior

$$p(\boldsymbol{\theta}(v; j)) = \text{Dirichlet}\left(\boldsymbol{\theta}(v; j)|\mathbf{u}(v; j)\right) \tag{9.3.42}$$

the posterior is proportional to the joint distribution

$$p(\boldsymbol{\theta}(v), \mathcal{D}(v)) = p(\boldsymbol{\theta}(v))p(\mathcal{D}(v)|\boldsymbol{\theta}(v)) \tag{9.3.43}$$

$$= \prod_j \frac{1}{Z(\mathbf{u}(v; j))} \prod_i \theta_i(v; j)^{u_i(v;j)-1} \prod_n \prod_j \prod_i \theta_i(v; j)^{\mathbb{I}[v^n = i, \mathrm{pa}(v^n) = j]} \tag{9.3.44}$$

$$= \prod_j \frac{1}{Z(\mathbf{u}(v; j))} \prod_i \theta_i(v; j)^{u_i(v;j)-1+\sharp(v=i,\mathrm{pa}(v)=j)} \tag{9.3.45}$$

where $Z(\mathbf{u})$ is the normalisation constant of a Dirichlet distribution.
Hence the posterior is

$$p(\boldsymbol{\theta}(v)|\mathcal{D}(v)) = \prod_j \text{Dirichlet}\left(\boldsymbol{\theta}(v; j)|\mathbf{u}'(v; j)\right) \tag{9.3.46}$$

where the hyperparameter prior term is updated by the observed counts,

$$u_i'(v; j) \equiv u_i(v; j) + \sharp\left(v = i, \mathrm{pa}(v) = j\right) \tag{9.3.47}$$

By analogy with the no-parents case, the marginal table is given by (writing the states explicitly)

$$p(v = \mathsf{i}|\mathrm{pa}(v) = \mathsf{j}, \mathcal{D}(v)) \propto u_{\mathsf{i}}'(v; \mathsf{j}) \tag{9.3.48}$$

a	s	c
1	1	2
1	0	0
0	1	1
0	1	0
1	1	2
0	0	0
1	0	1

Figure 9.8: A database of patient records about the Asbestos exposure (1 signifies exposure), being a Smoker (1 signifies the individual is a smoker), and lung Cancer (0 signifies no cancer, 1 signifies early stage cancer, 2 signifies late state cancer). Each row contains the information for an individual, so that there are 7 individuals in the database.

Example 40. Consider the $p(c|a,s)p(s)p(a)$ asbestos example with $\text{dom}(a) = \text{dom}(s) = \{0,1\}$, except now with the variable c taking three states, $\text{dom}(c) = \{0,1,2\}$, accounting for different kinds of cancer. The marginal table under a Dirichlet prior is then given by

$$p(c=0|a=1,s=1,\mathcal{V}) = \frac{u_0(a=1,s=1) + \sharp(c=0,a=1,s=1)}{\sum_{i\in\{0,1,2\}} u_i(a=1,s=1) + \sharp(c=i,a=1,s=1)} \tag{9.3.49}$$

Assuming a flat Dirichlet prior, which corresponds to setting all components of $\mathbf{u}$ to 1, this gives

$$p(c=0|a=1,s=1,\mathcal{V}) = \frac{1+0}{3+2} = \frac{1}{5} \tag{9.3.50}$$

$$p(c=1|a=1,s=1,\mathcal{V}) = \frac{1+0}{3+2} = \frac{1}{5} \tag{9.3.51}$$

$$p(c=2|a=1,s=1,\mathcal{V}) = \frac{1+2}{3+2} = \frac{3}{5} \tag{9.3.52}$$

and similarly for the other three tables $p(c|a=1,s=0), p(c|a=0,s=1), p(c|a=1,s=1)$.

Model likelihood

For a Belief Network M, the joint probability of all variables factorises into the local probabilities of each variable conditioned on its parents:

$$p(\mathcal{V}|M) = \prod_v p(v|\text{pa}(v), M) \tag{9.3.53}$$

For i.i.d. data $\mathcal{D}$, the likelihood under the network M is

$$p(\mathcal{D}|M) = \prod_v \prod_n p(v^n|\text{pa}(v^n), M) = \prod_v \prod_j \frac{Z(\mathbf{u}'(v;j))}{Z(\mathbf{u}(v;j))} \tag{9.3.54}$$

where $\mathbf{u}$ are the Dirichlet hyperparameters and $\mathbf{u}'$ is given by equation (9.3.47). Expression (9.3.54) can be written explicitly in terms of Gamma functions, see exercise(125). In the above expression in general the number of parental states differs for each variable v, so that implicit in the above formula is that the state product over j goes from 1 to the number of parental states of variable v. Due to the local and global parameter independence assumptions, the logarithm of the model likelihood splits into terms, one for each variable v and parental configuration. This is called the *likelihood decomposable* property.

9.3.5 Structure Learning

Up to this point we have assumed that we are given both the structure of the distribution and a dataset‘ $\mathcal{D}$. A more complex task is when we need to learn the structure of the network as well. We'll consider the case in which the data is complete (*i.e.* there are no missing observations). Since for D variables, there is an exponentially large number (in D) of BN structures, it's clear that we cannot search over all possible structures. For this reason structure learning is a computationally challenging problem and we must rely on constraints and heuristics to help guide the search. Furthermore, for all but the sparsest

Algorithm 3 PC algorithm for skeleton learning.

Start with a complete undirected graph G on the set $\mathcal{V}$ of all vertices.
$i = 0$
repeat
  for $x \in \mathcal{V}$ **do**
    for $y \in Adj\{x\}$ **do**
      Determine if there a subset $\mathcal{S}$ of size i of the neighbours of x (not including y) for which $x \perp\!\!\!\perp y | \mathcal{S}$. If this set exists remove the $x - y$ link from the graph G and set $\mathcal{S}_{xy} = \mathcal{S}$.
    end for
  end for
  $i = i + 1$.
until all nodes have $\leq i$ neighbours.

networks, estimating the dependencies to any accuracy requires a large amount of data, making testing of dependencies difficult. Indeed, for a finite amount of data, two variables will always have non-zero mutual information, so that a threshold needs to be set to decide if the measured dependence is significant under the finite sample, see section(9.3.6). Other complexities arise from the concern that a Belief or Markov Network on the visible variables alone may not be a parsimonious way to represent the observed data if, for example, there may be latent variables which are driving the observed dependencies. For these reasons we will not discuss this topic in detail here and limit the discussion to two central approaches.

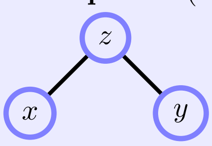
A special case that is computationally tractable is when the network is constrained to have at most one parent. We defer discussion of this to section(10.4.1).

PC algorithm

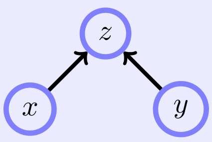
The PC algorithm[256] first learns the skeleton of a graph, after which edges may be oriented to form a (partially oriented) DAG.

The PC algorithm begins at the first round with a complete skeleton G and attempts to remove as many links as possible. At the first step we test all pairs $x \perp\!\!\!\perp y | \emptyset$. If an x and y pair are deemed independent then the link $x - y$ is removed from the complete graph. One repeats this for all the pairwise links. In the second round, for the remaining graph, one examines each $x - y$ link and conditions on a single neighbour z of x. If $x \perp\!\!\!\perp y | z$ then remove the link $x - y$. One repeats in this way through all the variables. At each round the number of neighbours in the conditioning set is increased by one. See algorithm(3), fig(9.9)[1] and `demoPCoracle.m`. A refinement of this algorithm, known as NPC for necessary path PC[258] attempts to limit the number of independence checks which may otherwise result in inconsistencies due to the empirical estimates of conditional mutual information. Given a learned skeleton, a partial DAG can be constructed using algorithm(4). Note that this is necessary since the undirected graph G is a skeleton – not a Belief Network of the independence assumptions discovered. For example, we may have a graph G with $x - z - y$ in which the $x - y$ link was removed on the basis $x \perp\!\!\!\perp y | \emptyset \rightarrow \mathcal{S}_{xy} = \emptyset$. As a MN the graph $x - z - y$ implies $x \top\!\!\!\top y$, although this is inconsistent with the discovery in the first round $x \perp\!\!\!\perp y$. This is the reason for the orientation part: for consistency, we must have $x \rightarrow z \leftarrow y$, for which $x \perp\!\!\!\perp y$ and $x \top\!\!\!\top y | z$. Note that in algorithm(4) we have for the 'unmarried collider' test, $z \notin \emptyset$, which in this case is true, resulting in a collider forming. See also fig(9.10).

Example 41 (Skeleton orienting)**.**



$x \perp\!\!\!\perp y | \emptyset \Rightarrow$



If x is (unconditionally) independent of y, it must be that z is a collider since otherwise marginalising over z would introduce a dependence between x and y.

[1]This example appears in [146] and [206] – thanks also to Serafín Moral for his online notes.

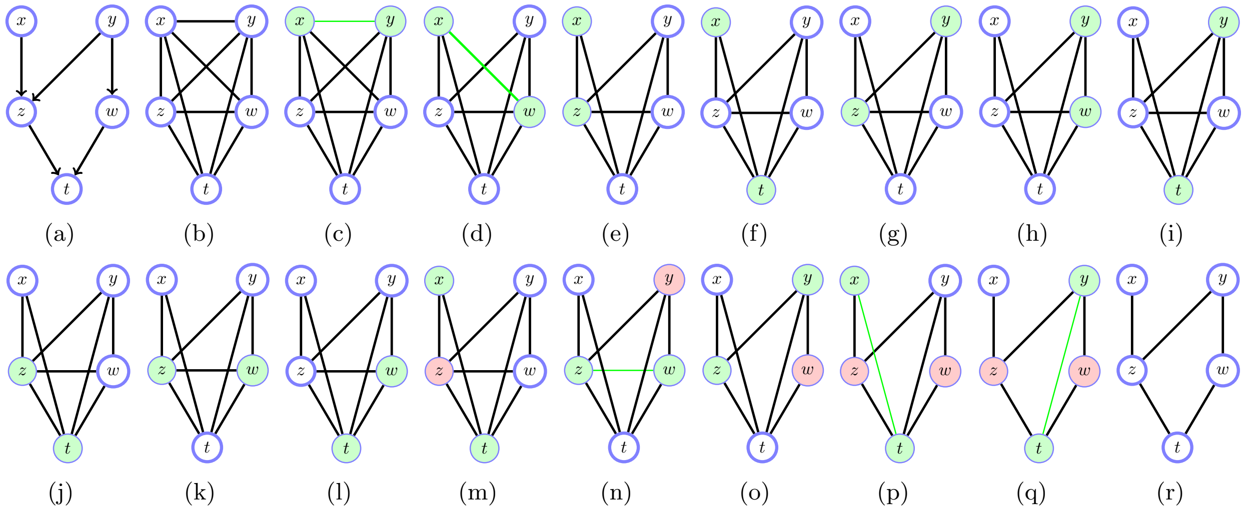
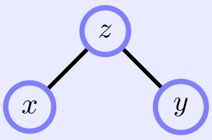
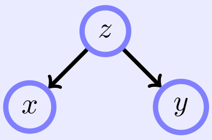


Figure 9.9: PC algorithm. **(a)**: The BN from which data is assumed generated and against which conditional independence tests will be performed. **(b)**: The initial skeleton is fully connected. **(c-l)**: In the first round ($i = 0$) all the pairwise mutual informations $x \perp\!\!\!\perp y|\emptyset$ are checked, and the link between x and y removed if deemed independent (green line). **(m-o)**: $i = 1$. We now look at connected subsets on the three variables x, y, z of the remaining graph, removing the link $x - y$ if $x \perp\!\!\!\perp y|z$ is true. Not all steps are shown. **(p,q)**: $i = 2$. We now examine all $x \perp\!\!\!\perp y|\{a, b\}$. The algorithm terminates after this round (when i gets incremented to 3) since there are no nodes with 3 or more neighbours. **(r)**: Final skeleton. During this process the sets $S_{x,y} = \emptyset, S_{x,w} = \emptyset, S_{z,w} = y, S_{x,t} = \{z, w\}, S_{y,t} = \{z, w\}$ were found. See also `demoPCoracle.m`



$x \perp\!\!\!\perp y|z \Rightarrow$



If x is independent of y conditioned on z, z must not be a collider. Any other orientation is appropriate.

9.3.6 Empirical Independence

Given a data set $\mathcal{D}$, containing variables x, y, z, our interest is to measure if $x \perp\!\!\!\perp y|z$. One approach is to use the conditional mutual information which is the average of conditional Kullback-Leibler divergences.

Definition 87 (Mutual Information)**.**

$$\mathrm{MI}(x; y|z) \equiv \langle \mathrm{KL}(p(x, y|z)|p(x|z)p(y|z))\rangle_{p(z)} \geq 0 \tag{9.3.55}$$

where this expression is equally valid for sets of variables. If $x \perp\!\!\!\perp y|z$ is true, then $\mathrm{MI}(x; y|z)$ is zero, and vice versa. When $z = \emptyset$, the average over $p(z)$ is absent and one writes $\mathrm{MI}(x; y)$.

Given data we can obtain an estimate of the conditional mutual information by using the empirical distribution $p(x, y, z)$ estimated by simply counting occurrences in the data. In practice, however, we only have a finite amount of data to estimate the empirical distribution so that for data sampled from distribution for which the variables truly are independent, the empirical mutual information will typically be greater than zero. An issue therefore is what threshold to use for the empirical conditional mutual information to decide if this is sufficiently far from zero to be caused by dependence. A frequentist approach is to compute the distribution of the conditional mutual information and then see where the sample value is compared to the distribution. According to [161] $2N\mathrm{MI}(x; y|z)$ is Chi-square distributed

Algorithm 4 Skeleton orientation algorithm (returns a DAG).

Unmarried Collider: Examine all undirected links $x - z - y$. If $z \notin \mathcal{S}_{xy}$ set $x \to z \leftarrow y$.
repeat
  $x \to z - y \Rightarrow x \to z \to y$
  For $x - y$, if there is a directed path from x to y orient $x \to y$
  If for $x - z - y$ there is a w such that $x \to w$, $y \to w$, $z - w$ then orient $z \to w$
until No more edges can be oriented.
The remaining edges can be arbitrarily oriented provided that the graph remains a DAG and no additional colliders are introduced.

with $(X-1)(Y-1)Z$ degrees of freedom, although this test does not work well in the case of small amounts of data. An alternative pragmatic approach is to estimate the threshold based on empirical samples of the MI under controlled independent/dependent conditions – see `demoCondindepEmp.m` for a comparison of these approaches.

Bayesian conditional independence test

A Bayesian approach to testing for independence can be made by comparing the likelihood of the data under the independence hypothesis, versus the likelihood under the dependent hypothesis. For the independence hypothesis we have a joint distribution over variables and parameters:

$$p(x, y, z, \theta|\mathcal{H}_{indep}) = p(x|z, \theta_{x|z})p(y|z, \theta_{y|z})p(z|\theta_z)p(\theta_{x|z})p(\theta_{y|z})p(\theta_z) \tag{9.3.56}$$

For multinomial distributions, it is convenient to use a Dirichlet prior Dirichlet $(\theta|u)$ on the parameters θ, assuming also local as well as global parameter independence. For a set of assumed i.i.d. data $(\mathcal{X}, \mathcal{Y}, \mathcal{Z}) = (x^n, y^n, z^n), n = 1, \ldots, N$, the likelihood is then given by integrating

$$p(\mathcal{X}, \mathcal{Y}, \mathcal{Z}|\mathcal{H}_{indep}) = \int_\theta \prod_n p(x^n, y^n, z^n, \theta|\mathcal{H}_{indep})$$

over the parameters θ. Thanks to conjugacy, this is straightforward, and gives the expression

$$p(\mathcal{X}, \mathcal{Y}, \mathcal{Z}|\mathcal{H}_{indep}) = \frac{Z(u_z + \sharp(z))}{Z(u_z)} \prod_z \frac{Z(u_{x|z} + \sharp(x, z))}{Z(u_{x|z})} \frac{Z(u_{y|z} + \sharp(y, z))}{Z(u_{y|z})} \tag{9.3.57}$$

where $u_{x|z}$ is a hyperparameter matrix of pseudo counts for each state of x given each state of z. $Z(v)$ is the normalisation constant of a Dirichlet distribution with vector parameter v.

For the dependent hypothesis we have

$$p(x, y, z, \theta|\mathcal{H}_{dep}) = p(x, y, z|\theta_{x,y,z})p(\theta_{x,y,z}) \tag{9.3.58}$$

The likelihood is then

$$p(\mathcal{X}, \mathcal{Y}, \mathcal{Z}|\mathcal{H}_{dep}) = \frac{Z(u_{x,y,z} + \sharp(x, y, z))}{Z(u_{x,y,z})} \tag{9.3.59}$$

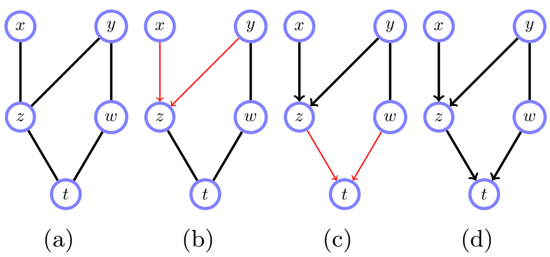


Figure 9.10: Skeleton orientation algorithm. **(a)**: The skeleton along with $S_{x,y} = \emptyset, S_{x,w} = \emptyset, S_{z,w} = y, S_{x,t} = \{z, w\}, S_{y,t} = \{z, w\}$. **(b)**: $z \notin S_{x,y}$, so form collider. **(c)**: $t \notin S_{z,w}$, so form collider. **(d)**: Final partially oriented DAG. The remaining edge may be oriented as desired, without violating the DAG condition. See also `demoPCoracle.m`.

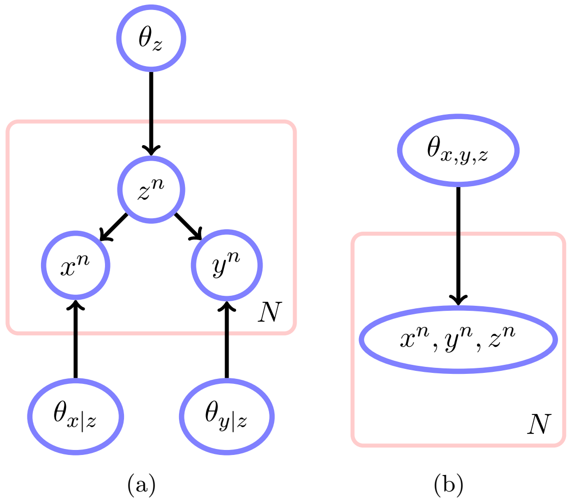


Figure 9.11: Bayesian conditional independence test using Dirichlet priors on the tables. **(a)**: A model $\mathcal{H}_{indep}$ for conditional independence $x \perp\!\!\!\perp y \,|\, z$. **(b)**: A model $\mathcal{H}_{dep}$ for conditional dependence $x \top\!\!\!\top y \,|\, z$. By computing the likelihood of the data under each model, a numerical score for the whether the data is more consistent with the conditional independence assumption can be formed. See `demoCondindepEmp.m`.

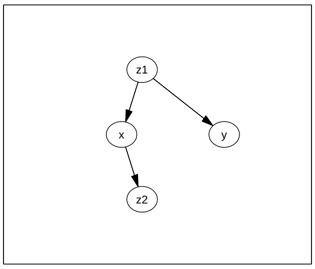


Figure 9.12: Conditional independence test of $x \perp\!\!\!\perp y \,|\, z_1, z_2$ with x, y, z_1, z_2 having $3, 2, 4, 2$ states respectively. From the oracle Belief network shown, in each experiment the tables are drawn at random and 20 examples are sampled to form a dataset. For each dataset a test is carried out to determine if x and y are independent conditioned on z_1, z_2 (the correct answer being that they are independent). Over 500 experiments, the Bayesian conditional independence test correctly states that the variables are conditionally independent 74% of the time, compared with only 50% accuracy using the chi-square mutual information test. See `demoCondindepEmp.m`.

Assuming each hypothesis is equally likely, for a Bayes' Factor

$$\frac{p(\mathcal{X}, \mathcal{Y}, \mathcal{Z}|\mathcal{H}_{indep})}{p(\mathcal{X}, \mathcal{Y}, \mathcal{Z}|\mathcal{H}_{dep})} \tag{9.3.60}$$

greater than 1, we assume that conditional independence holds, otherwise we assume the variables are conditionally dependent. `demoCondindepEmp.m` suggests that the Bayesisan hypothesis test tends to outperform the conditional mutual information approach, particularly in the small sample size case, see fig(9.12).

9.3.7 Network scoring

An alternative to local methods such as the PC algorithm is to evaluate the whole network structure. In a probabilistic context, given a model structure M, we wish to compute $p(M|\mathcal{D}) \propto p(\mathcal{D}|M)p(M)$. Some care is needed here since we have to first 'fit' each model with parameters θ, $p(\mathcal{V}|\theta, M)$ to the data $\mathcal{D}$. If we do this using Maximum Likelihood alone, with no constraints on θ, we will always end up favouring that model M with the most complex structure (assuming $p(M) = \text{const.}$). This can be remedied by using the Bayesian technique

$$p(\mathcal{D}|M) = \int_\theta p(\mathcal{D}|\theta, M)p(\theta|M) \tag{9.3.61}$$

In the case of directed networks, however, as we saw in section(9.3), the assumptions of local and global parameter independence make the integrals tractable. For a discrete state network and Dirichlet priors, we have $p(\mathcal{D}|M)$ given explicitly by the *Bayesian Dirichlet score* equation (9.3.54). First we specify the hyperparameters $\mathbf{u}(v; j)$, and then search over structures M, to find the one with the best score $p(\mathcal{D}|M)$. The simplest setting for the hyperparameters is set them all to unity[65]. Another setting is the 'uninformative prior'[51]

$$u_i(v; j) = \frac{\alpha}{\dim v \dim \text{pa}\,(v)} \tag{9.3.62}$$

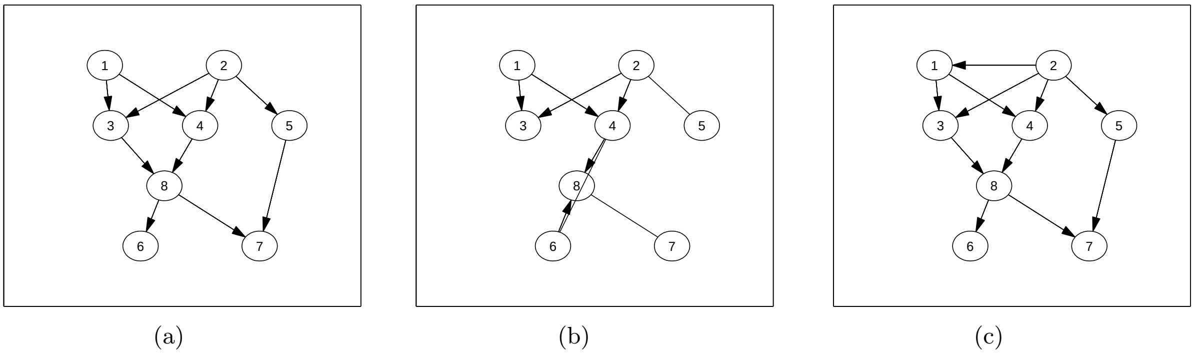


Figure 9.13: Learning the structure of a Bayesian network. **(a)**: The correct structure in which all variables are binary. The ancestral order is $2, 1, 5, 4, 3, 8, 7, 6$. The dataset is formed from 1000 samples from this network. **(b)**: The learned structure based on the PC algorithm using the Bayesian empirical conditional independence test. Undirected edges may be oriented arbitrarily. **(c)**: The learned structure based on the Bayes Dirichlet network scoring method. See `demoPCdata.m` and `demoBDscore.m`.

where $\dim x$ is the number of states of the variable(s) x, giving rise to the *BDeu* score, for an 'equivalent sample size' parameter α. A discussion of these settings is given in [127] under the concept of likelihood equivalence, namely that two networks which are Markov equivalent should have the same score. How dense the resulting network is can be sensitive to α[260, 247, 259]. Including an explicit prior $p(M)$ on the networks to favour those with sparse connections is also a sensible idea, for which one modified the score to $p(\mathcal{D}|M)p(M)$.

Searching over structures is a computationally demanding task. However, since the log-score decomposes into terms involving each family of v, we can compare two networks differing in a single arc efficiently. Search heuristics based on local addition/removal/reversal of links [65, 127] that increase the score are popular[127]. In `learnBayesNet.m` we simplify the problem for demonstration purposes in which we assume we know the ancestral order of the variables, and also the maximal number of parents of each variable.

Example 42 (PC algorithm versus network scoring)**.** In fig(9.13) we compare the PC algorithm with BD network scoring based (with Dirichlet hyperparameters set to unity) on 1000 samples from a known Belief Network. The PC algorithm conditional independence test is based on the Bayesian factor (9.3.60) in which Dirichlet priors with $\alpha = 0.1$ were used throughout. In fig(9.13) the network scoring technique outperforms the PC algorithm. This is partly explained by the network scoring technique being provided with the correct ancestral order and the constraint that each variable has maximally two parents.

9.4 Maximum Likelihood for Undirected models

Consider a Markov network distribution $p(\mathcal{X})$ defined on (not necessarily maximal) cliques $\mathcal{X}_c \subseteq \mathcal{X}$

$$p(\mathcal{X}|\theta) = \frac{1}{Z(\theta)} \prod_c \phi_c(\mathcal{X}_c|\theta_c) \tag{9.4.1}$$

where

$$Z(\theta) = \sum_{\mathcal{X}} \prod_c \phi_c(\mathcal{X}_c|\theta_c) \tag{9.4.2}$$

ensures normalisation. Given a set of data, $\mathcal{X}^n, n = 1, \ldots, N$, and assuming i.i.d. data, the log likelihood is

$$L(\theta) = \sum_n \sum_c \log \phi_c(\mathcal{X}_c^n|\theta_c) - N \log Z(\theta) \tag{9.4.3}$$

In general learning the optimal parameters $\theta_c, c = 1, \ldots, C$ is awkward since they are coupled via $Z(\theta)$. Unlike the BN, the objective function does not split into a set of isolated parameter terms and in general we need to resort to numerical methods. In special cases, however, exact results still apply, in particular when the MN is decomposable and no constraints are placed on the form of the clique potentials, as we discuss in section(9.4.2). More generally, however, gradient based techniques may be used and also give insight into properties of the Maximum Likelihood solution.

9.4.1 The likelihood gradient

$$\frac{\partial}{\partial \theta_c} L(\theta) = \sum_n \frac{\partial}{\partial \theta_c} \log \phi_c(\mathcal{X}_c^n|\theta_c) - N \left\langle \frac{\partial}{\partial \theta_c} \log \phi_c(\mathcal{X}_c|\theta_c) \right\rangle_{p(\mathcal{X}_\mathcal{C}|\theta)} \tag{9.4.4}$$

where we used the result

$$\frac{\partial}{\partial \theta_c} \log Z(\theta) = \frac{1}{Z(\theta)} \sum_{\mathcal{X}} \frac{\partial}{\partial \theta_c} \phi_c(\mathcal{X}_c|\theta_c) \prod_{c \neq c'} \phi_{c'}(\mathcal{X}_{c'}|\theta_{c'}) = \left\langle \frac{\partial}{\partial \theta_c} \log \phi_c(\mathcal{X}_c|\theta_c) \right\rangle_{p(\mathcal{X}_c|\theta)} \tag{9.4.5}$$

The gradient can then be used as part of a standard numerical optimisation package.

Exponential form potentials

A common form of parameterisation is to use an exponential form

$$\phi_c(\mathcal{X}_c) = e^{\boldsymbol{\theta}^\mathsf{T} \boldsymbol{\psi}_c(\mathcal{X}_\mathcal{C})} \tag{9.4.6}$$

where $\boldsymbol{\theta}$ are the parameters and $\boldsymbol{\psi}_c(\mathcal{X}_\mathcal{C})$ is a fixed 'feature function' defined on the variables of clique c. Differentiating with respect to $\boldsymbol{\theta}$ and equating to zero, we obtain that the Maximum Likelihood solution satisfies that the empirical average of a feature function matches the average of the feature function with respect to the model:

$$\langle \boldsymbol{\psi}_c(\mathcal{X}_\mathcal{C}) \rangle_{\epsilon(\mathcal{X}_c)} = \langle \boldsymbol{\psi}_c(\mathcal{X}_\mathcal{C}) \rangle_{p(\mathcal{X}_c)} \tag{9.4.7}$$

$$\epsilon(\mathcal{X}_c) = \frac{1}{N} \sharp (\mathcal{X}_c) \tag{9.4.8}$$

where $\sharp (\mathcal{X}_c)$ is the number of times the clique state $\mathcal{X}_c$ is observed in the dataset. An intuitive interpretation is to sample states $\mathcal{X}$ from the trained model $p(\mathcal{X})$ and use these to compute the average of each feature function. In the limit of an infinite number of samples, for a Maximum Likelihood optimal model, these sample averages will match those based on the empirical average.

Unconstrained potentials

For unconstrained potentials we have a separate table for each of the states defined on the clique. This means we may write

$$\phi_c(\mathcal{X}_c^n) = \prod_{\mathcal{Y}_c} \phi(\mathcal{Y}_c)^{\mathbb{I}[\mathcal{Y}_c = \mathcal{X}_c^n]} \tag{9.4.9}$$

where the product is over all states of potential c. This expression follows since the indicator is zero for all but the single observed state $\mathcal{X}_c^n$. The log likelihood is then

$$L(\theta) = \sum_c \sum_{\mathcal{Y}_c} \sum_n \mathbb{I}\left[\mathcal{Y}_c = \mathcal{X}_c^n\right] \log \phi_c(\mathcal{Y}_c) - N \log Z(\phi) \tag{9.4.10}$$

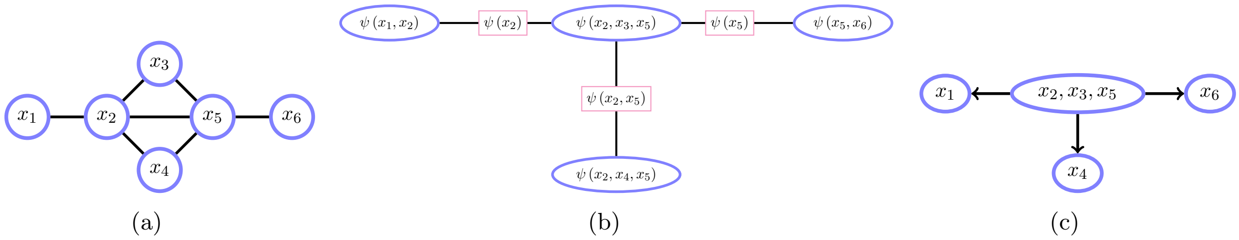


Figure 9.14: **(a)**: A decomposable Markov network. **(b)**: A junction tree for (a). **(c)**: Set chain for (a) formed by choosing clique x_2, x_3, x_5 as root and orienting edges consistently away from the root. Each separator is absorbed into its child clique to form the set chain.

where

$$Z(\phi) = \sum_{\mathcal{Y}} \prod_c \phi_c(\mathcal{Y}_c) \tag{9.4.11}$$

Differentiating the log likelihood with respect to a specific table entry $\phi(\mathcal{Y}_c)$ we obtain

$$\sum_n \mathbb{I}\left[\mathcal{Y}_c = \mathcal{X}_c^n\right] \frac{1}{\phi_c(\mathcal{Y}_c)} - N \frac{p(\mathcal{Y}_c)}{\phi_c(\mathcal{Y}_c)} \tag{9.4.12}$$

Equating to zero, the Maximum Likelihood solution is obtained when

$$p(\mathcal{Y}_c) = \epsilon(\mathcal{Y}_c) \equiv \frac{1}{N} \sum_n \mathbb{I}\left[\mathcal{Y}_c = \mathcal{X}_c^n\right] \tag{9.4.13}$$

That is, the unconstrained optimal Maximum Likelihood solution is given by setting the clique potentials such that the marginal distribution on each clique $p(\mathcal{Y}_c)$ matches the empirical distribution on each clique $\epsilon(\mathcal{Y}_c)$.

9.4.2 Decomposable Markov networks

In the case that there is no constraint placed on the form of the factors ϕ_c and if the MN corresponding to these potentials is decomposable, then we know (from the junction tree representation) that we can express the distribution in the form of a product of local marginals divided by the separator distributions

$$p(\mathcal{X}) = \frac{\prod_c p(\mathcal{X}_\mathcal{C})}{\prod_s p(\mathcal{X}_\mathcal{S})} \tag{9.4.14}$$

By reabsorbing the separators into the numerator terms, we can form a set chain distribution, section(6.8)

$$p(\mathcal{X}) = \prod_c p(\mathcal{X}_\mathcal{C} | \mathcal{X}_\mathcal{S}) \tag{9.4.15}$$

Since this is directed, the Maximum Likelihood solution to learning the tables is trivial since we assign each set chain factor $p(\mathcal{X}_\mathcal{C} | \mathcal{X}_\mathcal{S})$ by counting the instances in the dataset[165], see `learnMarkovDecom.m`. The procedure is perhaps best explained by an example, as given below. See algorithm(5) for a general description.

Example 43. Given a dataset $\mathcal{V} = \{\mathcal{X}^n, n = 1, \ldots, N\}$, we wish to fit by Maximum Likelihood a MN of the form

$$p(x_1, \ldots, x_6) = \frac{1}{Z} \phi(x_1, x_2) \phi(x_2, x_3, x_5) \phi(x_2, x_4, x_5) \phi(x_5, x_6) \tag{9.4.16}$$

Algorithm 5 Learning of an unconstrained decomposable Markov network using Maximum Likelihood. We have a triangulated (decomposable) Markov network on cliques $\phi_c(\mathcal{X}_c)$, $c = 1, \ldots, C$ and the empirical marginal distributions on all cliques and separators, $\epsilon(\mathcal{X}_c)$, $\epsilon(\mathcal{X}_s)$

1: Form a junction tree from the cliques.
2: Initialise each clique $\psi_c(\mathcal{X}_c)$ to $\epsilon(\mathcal{X}_c)$ and each separator $\psi_s(\mathcal{X}_s)$ to $\epsilon(\mathcal{X}_s)$.
3: Choose a root clique on the junction tree and orient edges consistently away from this root.
4: For this oriented junction tree, divide each clique by its parent separator.
5: Return the new potentials on each clique as the Maximum Likelihood solution.

where the potentials are unconstrained tables, see fig(9.14a). Since the graph is decomposable, we know it admits a factorisation of clique potentials divided by the separators:

$$p(x_1, \ldots, x_6) = \frac{p(x_1, x_2)p(x_2, x_3, x_5)p(x_2, x_4, x_5)p(x_5, x_6)}{p(x_2)p(x_2, x_5)p(x_5)} \tag{9.4.17}$$

We can convert this to a set chain by reabsorbing the denominators into numerator terms, see section(6.8). For example, by choosing the clique x_2, x_3, x_5 as root,we can write

$$p(x_1, \ldots, x_6) = \underbrace{p(x_1|x_2)}_{\psi(x_1,x_2)} \underbrace{p(x_2, x_3, x_5)}_{\psi(x_2,x_3,x_5)} \underbrace{p(x_4|x_2, x_5)}_{\psi(x_2,x_4,x_5)} \underbrace{p(x_6|x_5)}_{\psi(x_5,x_6)} \tag{9.4.18}$$

where we identified the factors with clique potentials, and the normalisation constant Z is unity, see fig(9.14b). The advantage is that in this representation, the clique potentials are independent since the distribution is a BN on cluster variables. The log likelihood for an i.i.d. dataset $\mathcal{X} = \{x^n, n = 1, \ldots, N\}$ is

$$L = \sum_n \log p(x_1^n|x_2^n) + \log p(x_2^n, x_3^n, x_5^n) + \log p(x_4^n|x_2^n, x_5^n) + \log p(x_6^n|x_5^n) \tag{9.4.19}$$

where each of the terms is an independent parameter of the model. The Maximum Likelihood solution then corresponds (as for the BN case) to simply setting each factor to the datacounts. For example

$$\phi(x_2, x_4, x_5) = p(x_4|x_2, x_5) = \frac{\sharp\,(x_2, x_4, x_5)}{\sharp\,(x_2, x_5)}, \qquad \phi(x_2, x_3, x_5) = p(x_2, x_3, x_5) = \frac{\sharp\,(x_2, x_3, x_5)}{N} \tag{9.4.20}$$

9.4.3 Non-decomposable Markov networks

In the non-decomposable or constrained case, no closed form Maximum Likelihood solution generally exists and one needs to resort to numerical methods. According to equation (9.4.13) the Maximum Likelihood solution is such that the clique marginals match the empirical marginals. Assuming that we can absorb the normalisation constant into an arbitrarily chosen clique, we can drop explicitly representing the normalisation constant. For a clique c, the requirement that the marginal of p matches the empirical marginal on the variables in the clique is

$$\phi(\mathcal{X}_c) \sum_{\mathcal{X}_{\backslash c}} \prod_{d \neq c} \phi(\mathcal{X}_d) = \epsilon(\mathcal{X}_c) \tag{9.4.21}$$

Given an initial setting for the potentials we can then update $\phi(\mathcal{X}_c)$ to satisfy the above marginal requirement,

$$\phi^{new}(\mathcal{X}_c) = \frac{\epsilon(\mathcal{X}_c)}{\sum_{\mathcal{X}_{\backslash c}} \prod_{d \neq c} \phi(\mathcal{X}_d)} \tag{9.4.22}$$

which is required for each of the states of $\mathcal{X}_c$. By multiplying and dividing the right hand side by $\phi(\mathcal{X}_c)$ this is equivalent to

$$\phi^{new}(\mathcal{X}_c) = \frac{\phi(\mathcal{X}_c)\epsilon(\mathcal{X}_c)}{p(\mathcal{X}_c)} \tag{9.4.23}$$

One can view this IPF update as coordinate-wise optimisation of the log likelihood in which the coordinate corresponds to $\phi_c(\mathcal{X}_c)$, with all other parameters fixed. In this case this conditional optimum is analytically given by the above setting. One proceeds by selecting another potential to update. Note that in general, with each update, the marginal $p(\mathcal{X}_c)$ need to be recomputed. Computing these marginals may be expensive unless the width of the junction tree formed from the graph is suitably limited.

Example 44 (Boltzmann Machine learning)**.** We define the BM as

$$p(\mathbf{v}|\mathbf{W}) = \frac{1}{Z(\mathbf{W})}e^{\frac{1}{2}\mathbf{v}^\mathsf{T}\mathbf{W}\mathbf{v}}, \qquad Z(\mathbf{W}) = \sum_{\mathbf{v}} e^{\frac{1}{2}\mathbf{v}^\mathsf{T}\mathbf{W}\mathbf{v}} \tag{9.4.24}$$

for symmetric $\mathbf{W}$ and binary variables $\text{dom}(v_i) = \{0, 1\}$. Given a set of training data, $\mathcal{D} = \{\mathbf{v}^1, \ldots, \mathbf{v}^N\}$, the log likelihood is

$$L(\mathbf{W}) = \frac{1}{2}\sum_{n=1}^{N} (\mathbf{v}^n)^\mathsf{T} \mathbf{W}\mathbf{v}^n - N \log Z(\mathbf{W}) \tag{9.4.25}$$

Differentiating *w.r.t.* w_{ij}, $i \neq j$ we have the gradient

$$\frac{\partial L}{\partial w_{ij}} = \sum_{n=1}^{N} \left(v_i^n v_j^n - \langle v_i v_j \rangle_{p(\mathbf{v}|\mathbf{W})} \right) \tag{9.4.26}$$

A simple algorithm to optimise the weight matrix $\mathbf{W}$ is to use gradient ascent,

$$w_{ij}^{new} = w_{ij}^{old} + \eta \sum_{n=1}^{N} \left(v_i^n v_j^n - \langle v_i v_j \rangle_{p(\mathbf{v}|\mathbf{W})} \right) \tag{9.4.27}$$

for a learning rate $\eta > 0$. The intuitive interpretation is that learning will stop (the gradient is zero) when the second order statistics of the model $\langle v_i v_j \rangle_{p(\mathbf{v}|\mathbf{W})}$ match those of the empirical distribution, $\sum_n v_i^n v_j^n / N$. BM learning however is difficult since $\langle v_i v_j \rangle_{p(\mathbf{v}|\mathbf{W})}$ is typically computationally intractable for an arbitrary interaction matrix $\mathbf{W}$ and therefore needs to be approximated. Indeed, one cannot compute the likelihood $L(\mathbf{W})$ exactly so that monitoring performance is also difficult.

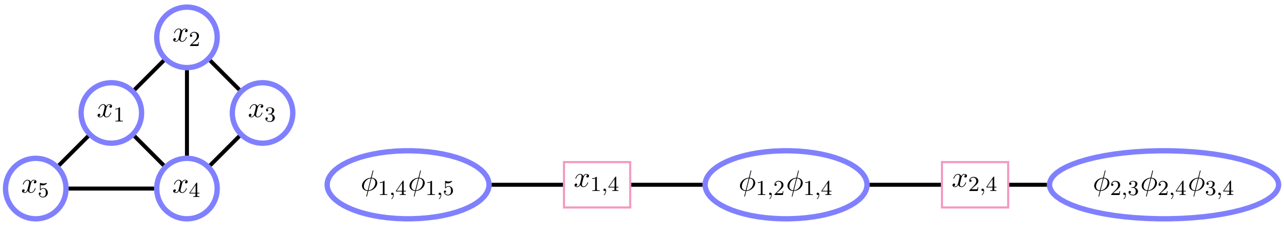
9.4.4 Constrained decomposable Markov networks

If there are no constraints on the forms of the maximal clique potentials of the Markov network, as we've seen, learning is straightforward. Here our interest is when the functional form of the maximal clique is constrained to be a product of potentials on smaller cliques:

$$\phi_c(\mathcal{X}_c) = \prod_i \phi_c^i(\mathcal{X}_c^i) \tag{9.4.28}$$

with no constraint[2] being placed on the non-maximal clique potentials $\phi_c^i(\mathcal{X}_c^i)$. In general, in this case one cannot write down directly the Maximum Likelihood solution for the non-maximal clique potentials $\phi_c^i(\mathcal{X}_c^i)$.

[2] A Boltzmann machine is of this form since any unconstrained binary pairwise potentials can be converted into a BM. For other cases in which the ϕ_c^i are constrained, then Iterative scaling may be used in place of IPF.



(a) (b)

Figure 9.15: **(a)**: Interpreted as a Markov network, the graph represents the distribution $\phi(x_1, x_4, x_5)\phi(x_1, x_2, x_4)\phi(x_2, x_4, x_3)$. As a pairwise MN, the graph represents $\phi(x_4, x_5)\phi(x_1, x_4)\phi(x_4, x_5)\phi(x_1, x_2)\phi(x_2, x_4)\phi(x_2, x_3)\phi(x_3, x_4)$. **(b)**: A junction tree for the pairwise MN in (a). We have a choice were to place the pairwise cliques, and this is one valid choice, using the shorthand $\phi_{a,b} = \phi_{a,b}(x_a, x_b)$ and $x_{a,b} = \{x_a, x_b\}$.

Consider the graph in fig(9.15). In the constrained case, in which we interpret the graph as a pairwise MN, IPF may be used to learn the pairwise tables. Since the graph is decomposable, there are however, computational savings that can be made in this case[11]. For an empirical distribution ϵ, Maximum Likelihood requires that all the pairwise marginals of the MN match the corresponding marginals obtained from ϵ. As explained in fig(9.15) we have a choice as to which junction tree clique each potential is assigned to, with one valid choice being given in fig(9.15b). Keeping the potentials of the cliques $\phi_{1,4}\phi_{1,5}$ and $\phi_{2,3}\phi_{2,4}\phi_{3,4}$ fixed we can update the potentials of clique $\phi_{1,2}\phi_{1,4}$. Using a bar to denote fixed potentials, we the marginal requirement that the MN matches the empirical marginal $\epsilon(x_1, x_2)$ can be written in shorthand as

$$p(x_1, x_2) = \sum_{x_3,x_4,x_5} \bar{\phi}_{1,5}\bar{\phi}_{4,5}\phi_{1,4}\phi_{1,2}\bar{\phi}_{2,4}\bar{\phi}_{2,3}\bar{\phi}_{3,4} = \epsilon(x_1, x_2) \tag{9.4.29}$$

which can be expressed as

$$\sum_{x_4} \underbrace{\left(\sum_{x_5} \bar{\phi}_{1,5}\bar{\phi}_{4,5}\right)}_{\gamma_{1,4}} \phi_{1,4}\phi_{1,2} \underbrace{\left(\sum_{x_3} \bar{\phi}_{2,4}\bar{\phi}_{2,3}\bar{\phi}_{3,4}\right)}_{\gamma_{2,4}} = \epsilon(x_1, x_2) \tag{9.4.30}$$

The 'messages' $\gamma_{1,4}$ and $\gamma_{1,2}$ are the boundary separator tables when we choose the central clique as root and carry out absorption towards the root. Given these fixed messages we can then perform updates of the root clique using

$$\phi_{1,2}^{new} = \frac{\epsilon(x_1, x_2)}{\sum_{x_4} \gamma_{1,4}\phi_{1,4}\gamma_{2,4}} \tag{9.4.31}$$

After making this update, we can subsequently update $\phi_{1,4}$ similarly using the constraint

$$\sum_{x_2} \underbrace{\left(\sum_{x_5} \bar{\phi}_{1,5}\bar{\phi}_{4,5}\right)}_{\gamma_{1,4}} \phi_{1,4}\phi_{1,2} \underbrace{\left(\sum_{x_3} \bar{\phi}_{2,4}\bar{\phi}_{2,3}\bar{\phi}_{3,4}\right)}_{\gamma_{2,4}} = \epsilon(x_1, x_4) \tag{9.4.32}$$

so that

$$\phi_{1,4}^{new} = \frac{\epsilon(x_1, x_4)}{\sum_{x_2} \gamma_{1,4}\phi_{1,2}\gamma_{2,4}} \tag{9.4.33}$$

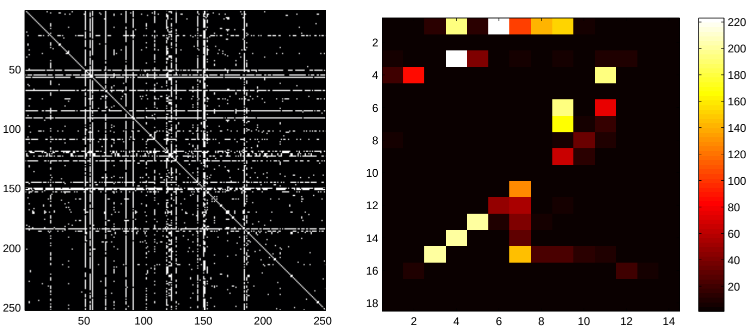
Given converged updates for this clique, we can choose another clique as root, propagate towards the root and compute the separator cliques on the boundary of the root. Given these fixed boundary clique potentials we perform IPF within the clique.

Algorithm 6 Efficient Iterative Proportional Fitting. Given a set of ϕ_i, $i = 1, \ldots, I$ and a corresponding set of reference (empirical) marginals distributions on the variables of each potential, ϵ_i, we aim to set all ϕ such that all marginals of the Markov network match the given empirical marginals.

Given a Markov network on potentials ϕ_i, $i = 1, \ldots, I$, triangulate the graph and form the cliques $\mathcal{C}_1, \ldots, \mathcal{C}_C$.
Assign potentials to cliques. Thus each clique has a set of associated potentials $\mathcal{F}_c$
Initialise all potentials (for example to unity).
repeat
    Choose a clique c as root.
    Propagate messages towards the root and compute the separators on the boundary of the root.
    repeat
        Choose a potential ϕ_i in clique c, $i \in \mathcal{F}_c$.
        Perform an IPF update for ϕ_i, given fixed boundary separators and other potentials in c.
    until Potentials in clique c converge.
until All Markov network marginals converge to the reference marginals.

This 'efficient' IPF procedure is described more generally in algorithm(6) for an empirical distribution ϵ. More generally, IPF minimises the Kullback-Leibler divergence between a given reference distribution ϵ and the Markov network. See `demoIPFeff.m` and `IPF.m`.

Example 45. In fig(9.17) 36 examples of $18 \times 12 = 252$ binary pixel handwritten twos are presented, forming the training set from which we wish to fit a Markov network. First all pairwise empirical entropies $H(x_i, x_j), i, j = 1, \ldots, 252$ were computed and used to rank edges, with highest entropy edges ranked first. Edges were included in a graph G, highest ranked first, provided the triangulated G had all cliques less than size 15. This resulted in 238 unique cliques and an adjacency matrix for the triangulated G as presented in fig(9.16a). In fig(9.16b) the number of times that a pixel appears in the 238 cliques is shown, and indicates the degree of importance of each pixel in distinguishing between the 36 examples. Two models were then trained and used to compute the most likely reconstruction based on missing data $p(x_{missing}|x_{visible})$. The first model was a Markov network on the maximal cliques of the graph, for which essentially no training is required, and the settings for each clique potential can be obtained as explained in algorithm(5). The model makes 3.8% errors in reconstruction of the missing pixels. Note that the unfortunate effect of reconstructing a white pixel surrounded by black pixels is an effect of the limited training data. With larger amounts of data the model would recognise that such effects do not occur. In the second model, the same maximal cliques were used, but the maximal clique potentials restricted to be the product of all pairwise two-cliques within the maximal clique. This is equivalent to using a Boltzmann machine, and was trained using the efficient IPF approach of algorithm(6). The corresponding reconstruction error is 20%. This performance is worse than the unconstrained network since the Boltzmann machine is a highly constrained Markov network. See `demoLearnDecMN.m`.



(a) (b)

Figure 9.16: **(a)**: Based on the pairwise empirical entropies $H(x_i, x_j)$ edges are ordered, high entropy edges first. Shown is the adjacency matrix of the resulting Markov network whose junction tree has cliques ≤ 15 in size (white represents an edge). **(b)**: Indicated are the number of cliques that each pixel is a member of, indicating a degree of importance. Note that the lowest clique membership value is 1, so that each pixel is a member of at least one clique.

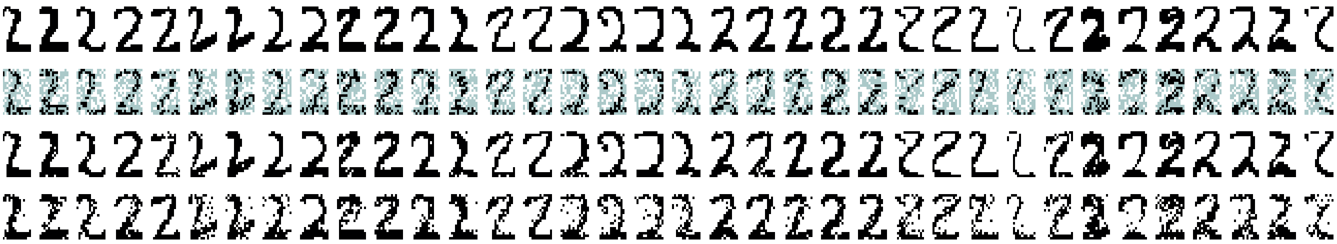


Figure 9.17: Learning digits using a Markov network. Top row: the 36 training examples. Each example is a binary image on 18×12 pixels. Second row: the training data with 50% missing pixels (grey represents a missing pixel). Third row: Reconstructions from the missing data using a thin-junction-tree MN with maximum clique size 15. Bottom row: Reconstructions using a thin-junction-tree Boltzmann machine with maximum clique size 15, trained using efficient IPF.

9.4.5 Iterative scaling

We consider Markov networks of the exponential form

$$p(\mathcal{V}|\theta) = \frac{1}{Z(\theta)} \prod_c e^{\theta_c f_c(\mathcal{V}_c)} \tag{9.4.34}$$

where $f_c(\mathcal{V}_c) \geq 0$ and c ranges of the non-maximal cliques $\mathcal{V}_c \subset \mathcal{V}$. The normalisation requirement is

$$Z(\theta) = \sum_{\mathcal{V}} \prod_c e^{\theta_c f_c(\mathcal{V}_c)} \tag{9.4.35}$$

A Maximum Likelihood training algorithm for a Markov network, somewhat analogous to the EM approach of section(11.2) can be derived as follows[32]:

Consider the bound, for positive x:

$$\log x \leq x - 1 \Rightarrow -\log x \geq 1 - x \tag{9.4.36}$$

Hence

$$-\log \frac{Z(\theta)}{Z(\theta^{old})} \geq 1 - \frac{Z(\theta)}{Z(\theta^{old})} \Rightarrow -\log Z(\theta) \geq -\log Z(\theta^{old}) + 1 - \frac{Z(\theta)}{Z(\theta^{old})} \tag{9.4.37}$$

Then we can write a bound on the log likelihood

$$\frac{1}{N} L(\theta) \geq \frac{1}{N} \sum_{c,n} \theta_c f_c(\mathcal{V}_c^n) - \log Z(\theta^{old}) + 1 - \frac{Z(\theta)}{Z(\theta^{old})} \tag{9.4.38}$$

As it stands, the bound (9.4.38) is in general not straightforward to optimise since the parameters of each potential are coupled through the $Z(\theta)$ term. For convenience it is useful to first reparmameterise and write

$$\theta_c = \underbrace{\theta_c - \theta_c^{old}}_{\alpha_c} + \theta_c^{old} \tag{9.4.39}$$

Then

$$Z(\theta) = \sum_{\mathcal{V}} e^{\sum_c f_c(\mathcal{V}_c)\theta_c} = \sum_{\mathcal{V}} e^{\sum_c f_c(\mathcal{V}_c)\theta_c^{old}} e^{\sum_c f_c(\mathcal{V}_c)\alpha_c} \tag{9.4.40}$$

One can decouple this using an additional bound derived by first considering:

$$e^{\sum_c \alpha_c f_c(\mathcal{V}_c)} = e^{\sum_c p_c \left[\alpha_c \sum_d f_d(\mathcal{V}_d)\right]} \tag{9.4.41}$$

where

$$p_c \equiv \frac{f_c(\mathcal{V}_c)}{\sum_d f_d(\mathcal{V}_d)} \tag{9.4.42}$$

Since $p_c \geq 0$ and $\sum_c p_c = 1$ we may apply Jensen's inequality to give

$$e^{\sum_c \alpha_c f_c(\mathcal{V}_c)} \leq \sum_c p_c e^{\sum_d f_d(\mathcal{V}_d)\alpha_c} \tag{9.4.43}$$

Hence

$$Z(\theta) \leq \sum_{\mathcal{V}} e^{\sum_c f_c(\mathcal{V}_c)\theta_c^{old}} \sum_c p_c e^{\alpha_c \sum_f f_d(\mathcal{V}_c)} \tag{9.4.44}$$

Plugging this bound into (9.4.38) we have

$$\frac{1}{N}L(\theta) \geq \sum_c \underbrace{\left\{ \frac{1}{N}\sum_n f_c(\mathcal{V}_c^n)\theta_c - \left\langle p_c e^{\alpha_c \sum_d f_d(\mathcal{V}_c)} \right\rangle_{p(\mathcal{V}|\theta^{old})} \right\}}_{LB(\theta_c)} + 1 - \log Z(\theta^{old}) \tag{9.4.45}$$

The term in curly brackets contains the potential parameters θ_c in an uncoupled fashion. Differentiating with respect to θ_c the gradient of each lower bound is given by

$$\frac{\partial LB(\theta_c)}{\partial \theta_c} = \frac{1}{N}\sum_n f_c(\mathcal{V}_c^n) - \left\langle f_c(\mathcal{V}_c) e^{\left(\theta_c - \theta_c^{old}\right)\sum_d f_d(\mathcal{V}_d)} \right\rangle_{p(\mathcal{V}|\theta^{old})} \tag{9.4.46}$$

This can be used as part of a gradient based optimisation procedure to learn the parameters θ_c. A potential advantage over IPF is that all the parameters may be updated simultaneously, whereas in IPF they must be updated sequentially. Intuitively, the parameters converge when the empirical average of the functions f match the average of the functions with respect to samples drawn from the distribution, in line with our general condition for Maximum Likelihood optimal solution.

In the special case that the functions sum to 1, $\sum_c f_c(\mathcal{V}_c) = 1$, the zero of the gradient can be found analytically, giving the update

$$\theta_c = \theta_c^{old} + \log \frac{1}{N}\sum_n f_c(\mathcal{V}_c^n) - \log \langle f_c(\mathcal{V}_c) \rangle_{p(\mathcal{V}_c|\theta^{old})} \tag{9.4.47}$$

The constraint that the features f_c need to be non-negative can be relaxed at the expense of additional variational parameters, see exercise(129). In cases where the zero of the gradient cannot be computed analytically, there may be little advantage in general in using IS over standard gradient based procedures on the log likelihood directly [194].

If the junction tree formed from this exponential form Markov network has limited tree width, computational savings can be made by performing IPF over the cliques of the junction tree and updating the parameters θ within each clique using IS[11]. This is a modified version of the constrained decomposable case. See also [271] for a unified treatment of propagation and scaling on junction trees.

9.4.6 Conditional random fields

For an input x and output y, a CRF is defined by a conditional distribution [263, 163]

$$p(y|x) = \frac{1}{Z(x)}\prod_k \phi_k(y, x) \tag{9.4.48}$$

for (positive) potentials $\phi_k(y, x)$.To make learning more straightforward, the potentials are usually defined as $e^{\lambda_k f_k(y,x)}$ for fixed functions $f(y, x)$ and parameters λ_k. In this case the distribution of the output conditioned on the input is

$$p(y|x, \lambda) = \frac{1}{Z(x, \lambda)}\prod_k e^{\lambda_k f_k(y,x)} \tag{9.4.49}$$

For an i.i.d. dataset of input-outputs, $\mathcal{D} = \{(x^n, y^n), n = 1, \ldots, N\}$, training based on conditional Maximum Likelihood requires the maximisation of

$$L(\lambda) \equiv \sum_{n=1}^{N} \log p(y^n|x^n, \lambda) = \sum_{n=1}^{N} \sum_k \lambda_k f_k(y^n, x^n) - \log Z(x^n, \lambda) \tag{9.4.50}$$

In general no closed form solution for the optimal λ exists and this needs to be determined numerically. First we note that equation (9.4.49) is equivalent to equation (9.4.34) where the parameters θ are here denoted by λ and the variables v are here denoted by y. In the CRF case the inputs simply have the effect of determining the feature $f_k(y, x)$. In this sense iterative scaling, or any related method for Maximum Likelihood training of constrained Markov networks, may be readily adapted, taking advantage also of any computational savings from limited width junction trees.

As an alternative here we briefly describe gradient based training. The gradient has components

$$\frac{\partial}{\partial \lambda_i} L = \sum_n \left(f_i(y^n, x^n) - \langle f_i(y, x^n) \rangle_{p(y|x^n,\lambda)} \right) \tag{9.4.51}$$

The terms $\langle f_i(y, x^n) \rangle_{p(y|x^n,\lambda)}$ can be problematic and their tractability depends on the structure of the potentials. For a multivariate y, provided the structure of the cliques defined on subsets of y is singly-connected, then computing the average is generally tractable. More generally, provided the cliques of the resulting junction tree have limited width, then exact marginals are avaiable. An example of this is given for a linear-chain CRF in section(23.4.4) – see also example(46) below.

Another quantity often useful for numerical optimisation is the Hessian which has components

$$\frac{\partial^2}{\partial \lambda_i \partial \lambda_j} L = \sum_n \left(\langle f_i(y, x^n) \rangle \langle f_j(y, x^n) \rangle - \langle f_i(y, x^n) f_j(y, x^n) \rangle \right) \tag{9.4.52}$$

where the averages above are with respect to $p(y|x^n, \lambda)$. This expression is a (negated) sum of covariance elements, and is therefore negative (semi) definite. Hence the function $L(\lambda)$ is concave and has only a single global optimum. Whilst no closed form solution for the optimal λ exists, the optimal solutions can be found easily using a numerical technique such as conjugate gradients.

In practice regularisation terms are often added to prevent overfitting (see section(13.2.3) for a discussion of regularisation) . Using a term

$$-\sum_k c_k^2 \lambda_k^2 \tag{9.4.53}$$

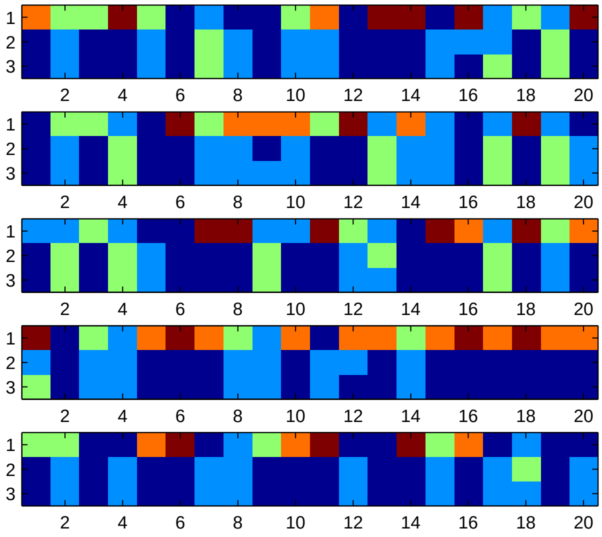
for positive regularisation constants c_k^2 discourages the weights λ from being too large. This term is also negative definite and hence the overall objective function remains concave. Iterative Scaling may also be used to train a CRF though in practice gradient based techniques are to be preferred[194].

Once trained a CRF can be used for predicting the output distribution for a novel input x^*. The most likely output y^* is equivalently given by

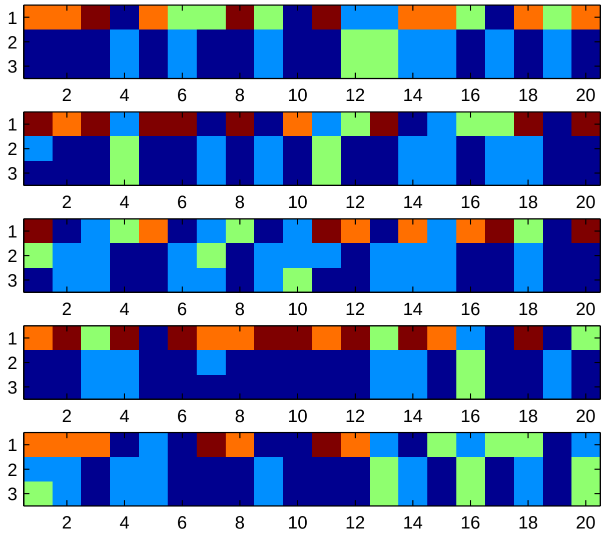
$$y^* = \underset{y}{\operatorname{argmax}} \; \log p(y|x^*) = \underset{y}{\operatorname{argmax}} \; \sum_k \lambda_k f_k(y, x^*) - \log Z(x^*, \lambda) \tag{9.4.54}$$

Since the normalisation term is independent of y, finding the most likely output is equivalent to

$$y^* = \underset{y}{\operatorname{argmax}} \; \sum_k \lambda_k f_k(y, x^*) \tag{9.4.55}$$



(a)



(b)

Figure 9.18: **(a)**: Training results for a linear chain CRF. There are 5 training sequences, one per subpanel. In each the top row corresponds to the input sequence $x_{1:20}$, $x_t \in \{1, \ldots, 5\}$ (each state represented by a different colour) the middle row, the correct output sequence $y_{1:20}$, $y_t \in \{1, 2, 3\}$ (each state represented by a different colour). Together the input and output sequences make the training data $\mathcal{D}$. The bottom row contains the most likely output sequence given the trained CRF, $\arg\max_{y_{1:20}} p(y_{1:20}|x_{1:20}, \mathcal{D})$. **(b)**: Five additional test sequences along with the correct output and predicted output sequence.

Natural language processing

In a natural language processing application, x_t might represent a word and y_t a corresponding linguistic tag ('noun','verb', *etc.*). A more suitable form in this case is to constrain the CRF to be of the form

$$\exp\left(\sum_k \mu_k g_k(y_t, y_{t-1}) + \sum_l \rho_l h_l(y_t, x_t)\right) \tag{9.4.56}$$

for binary functions g_k and h_l and parameters μ_k and ρ_l. The grammatical structure of tag-tag transitions is encoded in $g_k(y_t, y_{t-1})$ and linguistic tag information in $h_k(y_t, x_t)$, with the importance of these being determined by the corresponding parameters[163]. In this case inference of the marginals $\langle y_t y_{t-1} | x_{1:T}\rangle$ is straightforward since the factor graph corresponding to the inference problem is a linear chain.

Variants of the linear chain CRF are used heavily in natural language processing, including part-of-speech tagging and machine translation (in which the input sequence x represents a sentence say in English and the output sequence y the corresponding translation into French). See, for example, [211].

Example 46 (Linear chain CRF)**.** We consider a CRF with $X = 5$ input states and $Y = 3$ output states of the form

$$p(y_{1:T}|x_{1:T}) = \prod_{t=2}^{T} e^{\sum_k \mu_k g_k(y_t, y_{t-1}) + \sum_l \rho_l h_l(y_t, x_t)} \tag{9.4.57}$$

Here the binary functions $g_k(y_t, y_{t-1}) = \mathbb{I}[y_t = a_k]\,\mathbb{I}[y_{t-1} = b_k]$, $k = 1, \ldots, 9$ model the transitions between two consecutive outputs. The binary functions $h_l(y_t, x_t) = \mathbb{I}[y_t = a_l]\,\mathbb{I}[x_t = c_l]$, $l = 1, \ldots, 15$ model the translation of the input to the output. There are therefore $9 + 15 = 24$ parameters in total. In fig(9.18) we plot the training and test results based on a small set of data. The training of the CRF is obtained using 50 iterations of gradient ascent with a learning rate of 0.1. See `demoLinearCRF.m`.

9.4.7 Pseudo likelihood

Consider a MN on variables $\mathbf{x}$ with $\dim \mathbf{x} = D$ of the form

$$p(\mathbf{x}|\theta) = \frac{1}{Z}\prod_c \phi_c(\mathcal{X}_c|\theta_c) \tag{9.4.58}$$

For all but specially constrained ϕ_c, the partition function Z will be intractable and the likelihood of a set of i.i.d. data intractable as well. A surrogate is to use the *pseudo likelihood* of the probability of each variable conditioned on all other variables (which is equivalent to conditioning on only the variable's neighbours for a MN)

$$L'(\theta) = \sum_{n=1}^{N}\sum_{i=1}^{D} \log p(x_i^n|\mathbf{x}_{\setminus i}^n|\theta) \tag{9.4.59}$$

The terms $p(x_i^n|\mathbf{x}_{\setminus i}^n|\theta)$ are usually straightforward to work out since they require finding the normalisation of a univariate distribution only. In this case the gradient can be computed exactly, and learning of the parameters θ carried out. At least for the case of the Boltzmann machine, this forms a consistent estimator[138].

9.4.8 Learning the structure

Learning the structure of a Markov network can also be based on independence tests, as for Belief networks. A criterion for finding a MN on a set of nodes $\mathcal{V}$ is to use the fact that no edge exits between x and y if, conditioned on all other nodes, x and y are deemed independent. This is the pairwise Markov property described in section(4.2.1). By checking $x \perp\!\!\!\perp y|\mathcal{V}\setminus\{x,y\}$ for every pair of variables x and y, this edge deletion approach in principle reveals the structure of the network[216]. For learning the structure from an oracle, this method is sound. However, a practical difficulty in the case where the independencies are determined from data is that checking if $x \perp\!\!\!\perp y|\mathcal{V}\setminus\{x,y\}$ requires in principle enormous amounts of data. The reason for this is that the conditioning selects only those parts of the dataset consistent with the conditioning. In practice this will result in very small numbers of remaining datapoints, and estimating independencies on this basis is unreliable.

The Markov boundary criterion[216] uses the local Markov property,section(4.2.1), namely that conditioned on its neighbours, a variable is independent of all other variables in the graph. By starting with a variable x and an empty neighbourhood set, one can progressively include neighbours, testing if their inclusion renders the remaining non-neighbours independent of x. A difficultly with this is that, if one doesn't have the correct Markov boundary, then including a variable in the neighbourhood set may be deemed necessary. To see this, consider a network which corresponds to a linear chain and that x is at the edge of the chain. In this case, only the nearest neighbour of x is in the Markov boundary of x. However, if this nearest neighbour were not currently in the set, then any other non-nearest neighbour would be included, even though this is not strictly required. To counter this, the neighbourhood variables included in the neighbourhood of x may be later removed if they are deemed superfluous to the boundary[99].

In cases where specific constraints are imposed, such as learning structures whose resulting triangulation has a bounded tree-width, whilst still formally difficult, approximate procedures are available[257].

In terms of network scoring methods for undirected networks, computing a score is hampered by the fact that the parameters of each clique become coupled in the normalisation constant of the distribution. This issue can be addressed using hyper Markov priors[74].

9.5 Properties of Maximum Likelihood

9.5.1 Training assuming the correct model class

Consider a dataset $\mathcal{X} = \{x^n, n = 1, \ldots, N\}$ generated from an underlying parametric model $p(x|\theta^0)$. Our interest is to fit a model $p(x|\theta)$ of the same form as the correct underlying model $p(x|\theta^0)$ and examine whether if, in the limit of a large amount of data, the parameter θ learned by Maximum Likelihood matches the correct parameter θ^0. Our derivation below is non-rigorous, but highlights the essence of the argument.

Assuming the data is i.i.d., the log likelihood $L(\theta) \equiv \log p(\mathcal{X}|\theta)$ is

$$L(\theta) = \frac{1}{N}\sum_{n=1}^{N} \log p(x^n|\theta) \tag{9.5.1}$$

In the limit $N \to \infty$, the sample average can be replaced by an average with respect to the distribution generating the data

$$L(\theta) \overset{N\to\infty}{=} \langle \log p(x|\theta)\rangle_{p(x|\theta^0)} = -\mathrm{KL}\big(p(x|\theta^0)|p(x|\theta)\big) + \big\langle \log p(x|\theta^0)\big\rangle_{p(x|\theta^0)} \tag{9.5.2}$$

Up to a negligible constant, this is the Kullback-Leibler divergence between two distributions in x, just with different parameter settings. The θ that maximises $L(\theta)$ is that which minimises the Kullback-Leibler divergence, namely $\theta = \theta^0$. In the limit of a large amount of data we can, in principle, learn the correct parameters (assuming we know the correct model class). The property of an estimator such that the parameter θ converges to the true model parameter θ^0 as the sequence of data increase is termed a *consistency*.

9.5.2 Training when the assumed model is incorrect

We write $q(x|\theta)$ for the assumed model, and $p(x|\phi)$ for the correct generating model. Repeating the above calculations in the case of the assumed model being correct, we have that, in the limit of a large amount of data, the likelihood is

$$L(\theta) = \langle \log q(x|\theta)\rangle_{p(x|\phi)} = -\mathrm{KL}(p(x|\phi)|q(x|\theta)) + \langle \log p(x|\phi)\rangle_{p(x|\phi)} \tag{9.5.3}$$

Since q and p are not of the same form, setting θ to ϕ does not necessarily minimise $\mathrm{KL}(p(x|\phi)|q(x|\theta))$, and therefore does not necessarily optimize $L(\theta)$.

9.6 Notes

9.7 Code

`condindepEmp.m`: Bayes test and Mutual Information for empirical conditional independence
`condMI.m`: Conditional Mutual Information
`condMIemp.m`: Conditional Mutual Information of Empirical distribution
`MIemp.m`: Mutual Information of Empirical distribution

9.7.1 PC algorithm using an oracle

This demo uses an oracle to determine $x \perp\!\!\!\perp y \,|\, z$, rather than using data to determine the empirical dependence. The oracle is itself a Belief Network. For the partial orientation only the first 'unmarried collider' rule is implemented.
`demoPCoracle.m`: Demo of PC algorithm with an oracle
`PCskeletonOracle.m`: PC algorithm using an oracle
`PCorient.m`: Orient a skeleton

9.7.2 Demo of empirical conditional independence

For half of the experiments, the data is drawn from a distribution for which $x \perp\!\!\!\perp y | z$ is true. For the other half of the experiments, the data is drawn from a random distribution for which $x \perp\!\!\!\perp y | z$ is false. We then

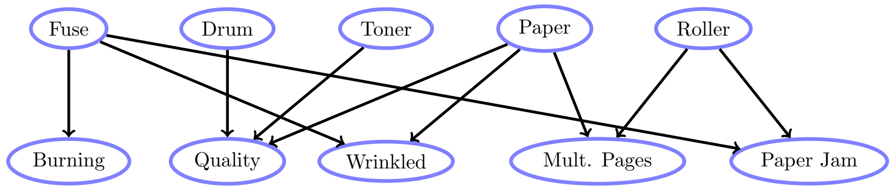


Figure 9.19: Printer Nightmare Belief Network. All variables are binary. The upper variables without parents are possible problems (diagnoses), and the lower variables consequences of problems (faults).

measure the fraction of experiments for which the Bayes test correctly decides $x \perp\!\!\!\perp y | z$. We also measure the fraction of experiments for which the Mutual Information test correctly decides $x \perp\!\!\!\perp y | z$, based on setting the threshold equal to the median of all the empirical conditional mutual information values. A similar empirical threshold can also be obtained for the Bayes' factor (although this is not strictly kosher in the pure Bayesian spirit since one should in principle set the threshold to zero). The test based on the assumed chi-squared distributed MI is included for comparison, although it seems to be impractical in these small data cases.

`demoCondIndepEmp.m`: Demo of empirical conditional independence based on data

9.7.3 Bayes Dirichlet structure learning

It is interesting to compare the result of `demoPCdata.m` with `demoBDscore.m`.

`PCskeletonData.m`: PC algorithm using empirical conditional independence
`demoPCdata.m`: Demo of PC algorithm with data
`BDscore.m`: Bayes Dirichlet (BD) score for a node given parents
`learnBayesNet.m`: Given an ancestral order and maximal parents, learn the network
`demoBDscore.m`: Demo of structure learning

9.8 Exercises

Exercise 116 (Printer Nightmare)**.** *Cheapco is, quite honestly, a pain in the neck. Not only did they buy a dodgy old laser printer from StopPress and use it mercilessly, but try to get away with using substandard components and materials. Unfortunately for StopPress, they have a contract to maintain Cheapco's old warhorse, and end up frequently sending the mechanic out to repair the printer. After the* 10^{th} *visit, they decide to make a statistical model of Cheapco's printer, so that they will have a reasonable idea of the fault based only on the information that Cheapco's secretary tells them on the phone. In that way, StopPress hopes to be able to send out to Cheapco only a junior repair mechanic, having most likely diagnosed the fault over the phone.*

Based on the manufacturer's information, StopPress has a good idea of the dependencies in the printer, and what is likely to directly affect other printer components. The Belief Network in fig(9.19) represents these assumptions. However, the specific way that Cheapco abuse their printer is a mystery, so that the exact probabilistic relationships between the faults and problems is idiosyncratic to Cheapco. StopPress has the following table of faults for each of the 10 visits. Each column represents a visit.

fuse assembly malfunction	0	0	0	1	0	0	0	0	0	0	0	0	1	0	1
drum unit	0	0	0	0	1	0	0	1	0	0	1	1	0	0	0
toner out	1	1	0	0	0	1	0	1	0	0	0	1	0	0	0
poor paper quality	1	0	1	0	1	0	1	0	1	1	0	1	1	0	0
worn roller	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1
burning smell	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0
poor print quality	1	1	1	0	1	1	0	1	0	0	1	1	0	0	0
wrinkled pages	0	0	1	0	0	0	0	0	1	0	0	0	1	1	1
multiple pages fed	0	0	1	0	0	0	1	0	1	0	0	0	0	0	1
paper jam	0	0	1	1	0	0	1	1	1	1	0	0	0	1	0

1. *The above table is contained in* `printer.mat`. *Learn all table entries on the basis of Maximum Likelihood.*

2. *Program the Belief Network using the tables Maximum Likelihood tables and* BRMLtoolbox. *Compute the probability that there is a fuse assembly malfunction given that the secretary complains there is a burning smell and that the paper is jammed, and that there are no other problems.*

3. *Repeat the above calculation using a Bayesian method in which a flat Beta prior is used on all tables.*

4. *Given the above information from the secretary, what is the most likely joint diagnosis over the diagnostic variables – that is the joint most likely* $p(Fuse, Drum, Toner, Paper, Roller|\mathsf{evidence})$*? Use the max-absorption method on the associated junction tree.*

5. *Compute the joint most likely state of the distribution*

$$p(Fuse, Drum, Toner, Paper, Roller|\mathsf{burning\ smell}, \mathsf{paper\ jammed})$$

Explain how to compute this efficiently using the max-absorption method.

Exercise 117. *Explain how to use a factorised Beta prior in the case of learning table entries in Belief Networks in which each variable has maximally a single parent. Consider the issues around Bayesian Learning of binary table entries when the number of parental variables is not restricted.*

Exercise 118. *Consider data* $x^n, n = 1, \ldots, N$*. Show that for a Gaussian distribution, the Maximum Likelihood estimator of the mean is* $\hat{m} = \frac{1}{N}\sum_{n=1}^{N} x^n$ *and variance is* $\hat{\sigma}^2 = \frac{1}{N}\sum_{n=1}^{N}(x^n - \hat{m})^2$.

Exercise 119. *A training set consists of one dimensional examples from two classes. The training examples from class 1 are*

$$0.5, 0.1, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1, 0.35, 0.25 \tag{9.8.1}$$

and from class 2 are

$$0.9, 0.8, 0.75, 1.0 \tag{9.8.2}$$

Fit a (one dimensional) Gaussian using Maximum Likelihood to each of these two classes. Also estimate the class probabilities p_1 *and* p_2 *using Maximum Likelihood. What is the probability that the test point* $x = 0.6$ *belongs to class 1?*

Exercise 120. *For a set of* N *observations (training data),* $\mathcal{X} = \mathbf{x}^1, \ldots, \mathbf{x}^N$*, and independently gathered observations, the log likelihood for a Belief network to generate* $\mathcal{X}$ *is*

$$\log p(\mathcal{X}) = \sum_{n=1}^{N}\sum_{i=1}^{K} \log p\left(x_i^n|\mathrm{pa}\left(x_i^n\right)\right) \tag{9.8.3}$$

We define the notation

$$\theta_{\mathsf{s}}^i(\mathsf{t}) = p(x_i = \mathsf{s}|\mathrm{pa}\left(x_i\right) = \mathsf{t}) \tag{9.8.4}$$

meaning variable x_i *is in state* s*, and the parents of variable* x_i *are in the vector of states* $\mathbf{t}$*. Using a Lagrangian*

$$L \equiv \sum_{n=1}^{N} \sum_{i=1}^{K} \log p\left(x_i^n | \mathrm{pa}\left(x_i^n\right)\right) + \sum_{i=1}^{K} \sum_{\mathbf{t}^i} \lambda_{\mathbf{t}^i}^i \left(1 - \sum_{\mathsf{s}} \theta_{\mathsf{s}}^i\left(\mathbf{t}^i\right)\right) \tag{9.8.5}$$

Show that the Maximum Likelihood setting of $\theta_{\mathsf{s}}^i(\mathbf{t})$ *is*

$$\theta_{\mathsf{s}}^j(\mathbf{t}^j) = \frac{\sum_{n=1}^{N} \mathbb{I}\left[x_j^n = \mathsf{s}\right] \mathbb{I}\left[\mathrm{pa}\left(x_j^n\right) = \mathbf{t}^j\right]}{\sum_{n=1}^{N} \sum_{\mathsf{s}} \mathbb{I}\left[x_j^n = \mathsf{s}\right] \mathbb{I}\left[\mathrm{pa}\left(x_j^n\right) = \mathbf{t}^j\right]} \tag{9.8.6}$$

Exercise 121 (Conditional Likelihood training)**.** *Consider a situation in which we partition observable variables into disjoint sets* x *and* y *and that we want to find the parameters that maximize the conditional likelihood,*

$$CL(\theta) = \frac{1}{N} \sum_{n=1}^{N} p(y^n | x^n, \theta), \tag{9.8.7}$$

for a set of training data $\{(x^n, y^n), n = 1, \ldots, N\}$*. All data is assumed generated from the same distribution* $p(x, y|\theta^0) = p(y|x, \theta^0)p(x|\theta^0)$ *for some unknown parameter* θ^0*. In the limit of a large amount of i.i.d. training data, does* $CL(\theta)$ *have an optimum at* θ^0*?*

Exercise 122 (Moment Matching)**.** *One way to set parameters of a distribution is to match the moments of the distribution to the empirical moments. This sometimes corresponds to Maximum Likelihood (for the Gaussian distribution for example), though generally this is not consistent with Maximum Likelihood. For data with mean* m *and variance* s*, show that to fit a Beta distribution by moment matching, we use*

$$\alpha = \frac{m^2(1-m)}{s} - m, \quad \beta = \alpha \frac{1-m}{m} \tag{9.8.8}$$

Exercise 123. *For data* $0 \le x^n \le 1$*,* $n = 1, \ldots, N$*, generated from a Beta distribution* $B(x|a, b)$*, show that the log likelihood is given by*

$$L(a, b) \equiv (a-1) \sum_{n=1}^{N} \log x^n + (b-1) \sum_{n=1}^{N} \log(1 - x^n) - N \log B(a, b) \tag{9.8.9}$$

where $B(a, b)$ *is the Beta function. Show that the derivatives are*

$$\frac{\partial}{\partial a} L = \sum_{n=1}^{N} \log x^n - \psi(a) - \psi(a+b), \quad \frac{\partial}{\partial b} L = \sum_{n=1}^{N} \log(1 - x^n) - \psi(b) - \psi(a+b) \tag{9.8.10}$$

where $\psi(x) \equiv d \log \Gamma(x)/dx$ *is the digamma function, and suggest a method to learn the parameters a,b.*

Exercise 124. *Consider the Boltzmann machine as defined in example(44).*

1. *Derive the gradient with respect to the 'biases'* w_{ii}*.*
2. *Write down the pseudo likelihood for a set of i.i.d. data* $\mathbf{v}^1, \ldots, \mathbf{v}^N$ *and derive the gradient of this with respect to* w_{ij}*,* $i \neq j$*.*

Exercise 125. *Show that the model likelihood equation (9.3.54) can be written explicitly as*

$$p(\mathcal{D}|M) = \prod_v \prod_j \frac{\Gamma\left(\sum_i u_i(v; j)\right)}{\Gamma\left(\sum_i u_i'(v; j)\right)} \prod_i \left[\frac{\Gamma\left(u_i'(v; j)\right)}{\Gamma\left(u_i(v; j)\right)}\right] \tag{9.8.11}$$

Exercise 126. *Define the set* $\mathcal{N}$ *as consisting of 8 node Belief Networks in which each node has at most 2 parents. For a given ancestral order* a*, the restricted set is written* $\mathcal{N}_a$

1. *How many Belief Networks are in* $\mathcal{N}_a$*?*
2. *What is the computational time to find the optimal member of* $\mathcal{N}_a$ *using the Bayesian Dirichlet score, assuming that computing the BD score of any member of* $\mathcal{N}_a$ *takes 1 second and bearing in mind the decomposability of the BD score.*
3. *What is the time to find the optimal member of* $\mathcal{N}$*?*

Exercise 127. *For the Markov network*

$$p(x, y, z) = \frac{1}{Z}\phi_1(x, y)\phi_2(y, z) \tag{9.8.12}$$

derive an iterative scaling algorithm to learn the unconstrained tables $\phi_1(x, y)$ *and* $\phi_2(x, y)$ *based on a set of i.i.d. data* $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$.

Exercise 128. *Given training data* $x^1, \ldots, x^n$, *derive an iterative scaling algorithm for Maximum Likelihood training of CRFs of the form*

$$p(x|\lambda) = \frac{1}{Z(\lambda)}\prod_c e^{\lambda_c f_c(x)}$$

where $Z(\lambda) = \sum_x \prod_c e^{\lambda_c f_c(x)}$ *and non-negative features,* $f_c(x) \geq 0$ *(you may assume that the features cannot all be zero for any given* x*).*

Exercise 129. *For data* $\mathcal{X}^1, \ldots, \mathcal{X}^N$, *consider Maximum Likelihood learning of a Markov network* $p(\mathcal{X}) = \prod_c \phi_c(\mathcal{X}_c)$ *with potentials of the form*

$$\phi_c(\mathcal{X}_c) = e^{\theta_c f_c(\mathcal{X}_c)} \tag{9.8.13}$$

with $f_c(\mathcal{X}_c)$ *being general real valued functions and* θ_c *real valued parameters. By considering*

$$\sum_c \theta_c f_c(\mathcal{X}_c) = \sum_c p_c \frac{\theta_c f_c(\mathcal{X}_c)}{p_c} \tag{9.8.14}$$

for auxiliary variables $p_c > 0$ *such that* $\sum_c p_c = 1$, *explain how to derive a form of iterative scaling training algorithm in which each parameter* θ_c *can be learned separately.*

CHAPTER 10

Naive Bayes

10.1 Naive Bayes and Conditional Independence

Naive Bayes (NB) is a popular classification method and aids our discussion of conditional independence, overfitting and Bayesian methods. In NB, we form a joint model of observations $\mathbf{x}$ and the corresponding class label c using a Belief network of the form

$$p(\mathbf{x}, c) = p(c) \prod_{i=1}^{D} p(x_i|c) \tag{10.1.1}$$

whose Belief Network is depicted in fig(10.1a). Coupled with a suitable choice for each conditional distribution $p(x_i|c)$, we can then use Bayes' rule to form a classifier for a novel attribute vector $\mathbf{x}^*$:

$$p(c|\mathbf{x}^*) = \frac{p(\mathbf{x}^*|c)p(c)}{p(\mathbf{x}^*)} = \frac{p(\mathbf{x}^*|c)p(c)}{\sum_c p(\mathbf{x}^*|c)p(c)} \tag{10.1.2}$$

In practice it is common to consider only two classes $\text{dom}(c) = \{0, 1\}$. The theory we describe below is valid for any number of classes c, though our examples are restricted to the binary class case. Also, the attributes x_i are often taken to be binary, as we shall do initially below as well. The extension to more than two attribute states, or continuous attributes is straightforward.

Example 47. EZsurvey.org considers Radio station listeners conveniently fall into two groups – the 'young' and 'old'. They assume that, given the knowledge that a customer is either 'young' or 'old', this is sufficient to determine whether or not a customer will like a particular Radio station, independent of their likes or dislikes for any other stations:

$$p(R1, R2, R3, R4|age) = p(R1|age)p(R2|age)p(R3|age)p(R4|age) \tag{10.1.3}$$

where each of the variables $R1, R2, R3, R4$ can take the states either like or dislike, and the 'age' variable can take the value either young or old. Thus the information about the age of the customer determines the individual product preferences without needing to know anything else. To complete the specification, given that a customer is young, she has a 95% chance to like Radio1, a 5% chance to like Radio2, a 2% chance to like Radio3 and a 20% chance to like Radio4. Similarly, an old listener has a 3% chance to like Radio1, an 82% chance to like Radio2, a 34% chance to like Radio3 and a 92% chance to like Radio4. They know that 90% of the listeners are old.

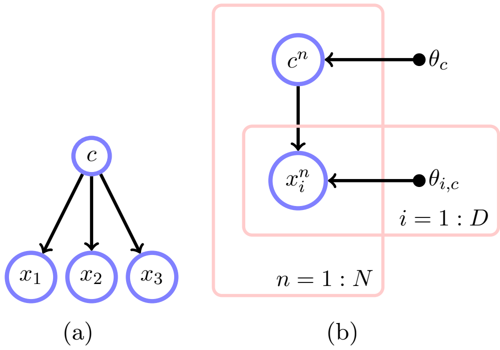


Figure 10.1: Naive Bayes classifer. **(a)**: The central assumption is that given the class c, the attributes x_i are independent. **(b)**: Assuming the data is i.i.d., Maximum Likelihood learns the optimal parameters of the distribution $p(c)$ and the class-dependent attribute distributions $p(x_i|c)$.

Given this model, and a new customer that likes Radio1, and Radio3, but dislikes Radio2 and Radio4, what is the probability that they are young? This is given by

$$p(age = \text{young}|R1 = \text{like}, R2 = \text{dislike}, R3 = \text{like}, R4 = \text{dislike})$$
$$= \frac{p(R1 = \text{like}, R2 = \text{dislike}, R3 = \text{like}, R4 = \text{dislike}|age = \text{young})p(age = \text{young})}{\sum_{age} p(R1 = \text{like}, R2 = \text{dislike}, R3 = \text{like}, R4 = \text{dislike}|age)p(age)} \quad (10.1.4)$$

Using the Naive Bayes structure, the numerator above is given by

$$p(R1 = \text{like}|age = \text{young})p(R2 = \text{dislike}|age = \text{young})$$
$$\times p(R3 = \text{like}|age = \text{young})p(R4 = \text{dislike}|age = \text{young})p(age = \text{young}) \quad (10.1.5)$$

Plugging in the values we obtain

$$0.95 \times 0.95 \times 0.02 \times 0.8 \times 0.1 = 0.0014$$

The denominator is given by this value plus the corresponding term evaluated under assuming the customer is old,

$$0.03 \times 0.18 \times 0.34 \times 0.08 \times 0.9 = 1.3219 \times 10^{-4}$$

Which gives

$$p(age = \text{young}|R1 = \text{like}, R2 = \text{dislike}, R3 = \text{like}, R4 = \text{dislike}) = \frac{0.0014}{0.0014 + 1.3219 \times 10^{-4}} = 0.9161 \quad (10.1.6)$$

10.2 Estimation using Maximum Likelihood

Learning the table entries for NB is a straightforward application of the more general BN learning discussed in section(9.2.3). For a fully observed dataset, Maximum Likelihood learning of the table entries corresponds to counting the number of occurrences in the training data, as we show below.

10.2.1 Binary attributes

Consider a dataset $\{\mathbf{x}^n, n = 1, \dots, N\}$ of binary attributes, $x_i^n \in \{0, 1\}$, $i = 1, \dots, D$. Each datapoint $\mathbf{x}^n$ has an associated class label c^n. The number of datapoints from class $c = 0$ is n_0 and the number from class $c = 1$ denoted is n_1. For each attribute of the two classes, we need to estimate the values $p(x_i = 1|c) \equiv \theta_i^c$. The other probability, $p(x_i = 0|c)$ is given by the normalisation requirement, $p(x_i = 0|c) = 1 - p(x_i = 1|c) = 1 - \theta_i^c$.

Based on the NB conditional independence assumption the probability of observing a vector $\mathbf{x}$ can be compactly written

$$p(\mathbf{x}|c) = \prod_{i=1}^{D} p(x_i|c) = \prod_{i=1}^{D} (\theta_i^c)^{x_i}(1-\theta_i^c)^{1-x_i} \tag{10.2.1}$$

In the above expression, x_i is either 0 or 1 and hence each i term contributes a factor θ_i^c if $x_i = 1$ or $1 - \theta_i^c$ if $x_i = 0$. Together with the assumption that the training data is i.i.d. generated, the log likelihood of the attributes and class labels is

$$L = \sum_n \log p(\mathbf{x}^n, c^n) = \sum_n \log p(c^n) \prod_i p(x_i^n|c^n) \tag{10.2.2}$$

$$= \sum_{i,n} x_i^n \log \theta_i^{c^n} + (1 - x_i^n) \log(1 - \theta_i^{c^n}) + n_0 \log p(c = 0) + n_1 \log p(c = 1) \tag{10.2.3}$$

This can be written more explicitly in terms of the parameters as

$$L = \sum_{i,n} \left\{ \mathbb{I}\left[x_i^n = 1, c^n = 0\right] \log \theta_i^0 + \mathbb{I}\left[x_i^n = 0, c^n = 0\right] \log(1 - \theta_i^0) + \mathbb{I}\left[x_i^n = 1, c^n = 1\right] \log \theta_i^1 \right.$$
$$\left. + \, \mathbb{I}\left[x_i^n = 0, c^n = 1\right] \log(1 - \theta_i^1) \right\} + n_0 \log p(c = 0) + n_1 \log p(c = 1) \tag{10.2.4}$$

We can find the Maximum Likelihood optimal θ_i^c by differentiating *w.r.t.* θ_i^c and equating to zero, giving

$$\theta_i^c = p(x_i = 1|c) = \frac{\sum_n \mathbb{I}\left[x_i^n = 1, c^n = c\right]}{\sum_n \mathbb{I}\left[x_i^n = 0, c^n = c\right] + \mathbb{I}\left[x_i^n = 1, c^n = c\right]} \tag{10.2.5}$$

$$= \frac{\text{number of times } x_i = 1 \text{ for class c}}{\text{number of datapoints in class c}} \tag{10.2.6}$$

Similarly, optimising equation (10.2.3) with respect to $p(c)$ gives

$$p(c) = \frac{\text{number of times class c occurs}}{\text{total number of data points}} \tag{10.2.7}$$

Classification boundary

We classify a novel input $\mathbf{x}^*$ as class 1 if

$$p(c = 1|\mathbf{x}^*) > p(c = 0|\mathbf{x}^*) \tag{10.2.8}$$

Using Bayes' rule and writing the log of the above expression, this is equivalent to

$$\log p(\mathbf{x}^*|c = 1) + \log p(c = 1) - \log p(\mathbf{x}^*) > \log p(\mathbf{x}^*|c = 0) + \log p(c = 0) - \log p(\mathbf{x}^*) \tag{10.2.9}$$

From the definition of the classifier, this is equivalent to (the normalisation constant $-\log p(\mathbf{x}^*)$ can be dropped from both sides)

$$\sum_i \log p(x_i^*|c = 1) + \log p(c = 1) > \sum_i \log p(x_i^*|c = 0) + \log p(c = 0) \tag{10.2.10}$$

Using the binary encoding $x_i \in \{0, 1\}$, we classify $\mathbf{x}^*$ as class 1 if

$$\sum_i \left\{ x_i^* \log \theta_i^1 + (1 - x_i^*) \log(1 - \theta_i^1) \right\} + \log p(c = 1) > \sum_i \left\{ x_i^* \log \theta_i^0 + (1 - x_i^*) \log(1 - \theta_i^0) \right\} + \log p(c = 0) \tag{10.2.11}$$

This decision rule can be expressed in the form: classify $\mathbf{x}^*$ as class 1 if $\sum_i w_i x_i^* + a > 0$ for some suitable choice of weights w_i and constant a, see exercise(133). The interpretation is that $\mathbf{w}$ specifies a hyperplane in the attribute space and $\mathbf{x}^*$ is classified as 1 if it lies on the positive side of the hyperplane.

0	1	1	1	0	0
0	0	1	1	1	0
1	1	0	0	0	0
1	1	0	0	0	1
1	0	1	0	1	0

(a)

1	1	1	1	1	1	1
0	1	1	1	1	0	0
0	0	1	0	0	1	1
1	0	1	1	1	1	0
1	1	0	0	1	0	0

(b)

Figure 10.2: **(a)**: English tastes over attributes $(shortbread, lager, whiskey, porridge, football)$. Each column represents the tastes of an individual. **(b)**: Scottish tastes.

Example 48 (Are they Scottish?)**.** Consider the following vector of attributes:

$$\text{(likes shortbread, likes lager, drinks whiskey, eats porridge, watched England play football)} \quad (10.2.12)$$

A vector $\mathbf{x} = (1, 0, 1, 1, 0)^{\mathsf{T}}$ would describe that a person likes shortbread, does not like lager, drinks whiskey, eats porridge, and has not watched England play football. Together with each vector $\mathbf{x}$, there is a label nat describing the nationality of the person, $\text{dom}(nat) = \{\mathsf{scottish}, \mathsf{english}\}$, see fig(10.2).

We wish to classify the vector $\mathbf{x} = (1, 0, 1, 1, 0)^{\mathsf{T}}$ as either scottish or english. We can use Bayes' rule to calculate the probability that $\mathbf{x}$ is Scottish or English:

$$p(\mathsf{scottish}|\mathbf{x}) = \frac{p(\mathbf{x}|\mathsf{scottish})p(\mathsf{scottish})}{p(\mathbf{x})} = \frac{p(\mathbf{x}|\mathsf{scottish})p(\mathsf{scottish})}{p(\mathbf{x}|\mathsf{scottish})p(\mathsf{scottish}) + p(\mathbf{x}|\mathsf{english})p(\mathsf{english})} \quad (10.2.13)$$

By Maximum Likelihood the 'prior' class probability $p(\mathsf{scottish})$ is given by the fraction of people in the database that are Scottish, and similarly $p(\mathsf{english})$ is given as the fraction of people in the database that are English. This gives $p(\mathsf{scottish}) = 7/13$ and $p(\mathsf{english}) = 6/13$.

For $p(\mathbf{x}|nat)$ under the Naive Bayes assumption:

$$p(\mathbf{x}|nat) = p(x_1|nat)p(x_2|nat)p(x_3|nat)p(x_4|nat)p(x_5|nat) \quad (10.2.14)$$

so that knowing whether not someone is Scottish, we don't need to know anything else to calculate the probability of their likes and dislikes. Based on the table in fig(10.2) and using Maximum Likelihood we have:

$$\begin{aligned} p(x_1 = 1|\mathsf{english}) &= 1/2 & p(x_1 = 1|\mathsf{scottish}) &= 1 \\ p(x_2 = 1|\mathsf{english}) &= 1/2 & p(x_2 = 1|\mathsf{scottish}) &= 4/7 \\ p(x_3 = 1|\mathsf{english}) &= 1/3 & p(x_3 = 1|\mathsf{scottish}) &= 3/7 \\ p(x_4 = 1|\mathsf{english}) &= 1/2 & p(x_4 = 1|\mathsf{scottish}) &= 5/7 \\ p(x_5 = 1|\mathsf{english}) &= 1/2 & p(x_5 = 1|\mathsf{scottish}) &= 3/7 \end{aligned} \quad (10.2.15)$$

For $\mathbf{x} = (1, 0, 1, 1, 0)^{\mathsf{T}}$, we get

$$p(\mathsf{scottish}|\mathbf{x}) = \frac{1 \times \frac{3}{7} \times \frac{3}{7} \times \frac{5}{7} \times \frac{4}{7} \times \frac{7}{13}}{1 \times \frac{3}{7} \times \frac{3}{7} \times \frac{5}{7} \times \frac{4}{7} \times \frac{7}{13} + \frac{1}{2} \times \frac{1}{2} \times \frac{1}{3} \times \frac{1}{2} \times \frac{1}{2} \times \frac{6}{13}} = 0.8076 \quad (10.2.16)$$

Since this is greater than 0.5, we would classify this person as being Scottish.

Small data counts

In example(48), consider trying to classify the vector $\mathbf{x} = (0, 1, 1, 1, 1)^{\mathsf{T}}$. In the training data, all Scottish people say they like shortbread. This means that for this particular $\mathbf{x}$, $p(\mathbf{x}, \mathsf{scottish}) = 0$, and therefore that we make the extremely confident classification $p(\mathsf{scottish}|\mathbf{x}) = 0$. This demonstrates a difficulty using Maximum Likelihood with sparse data. One way to ameliorate this is to smooth the probabilities, for example by adding a certain small number to the frequency counts of each attribute. This ensures that there are no zero probabilities in the model. An alternative is to use a Bayesian approach that discourages extreme probabilities, as discussed in section(10.3).

Potential pitfalls with encoding

In many off-the-shelf packages implementing Naive Bayes, binary attributes are assumed. In practice, however, the case of non-binary attributes often occurs. Consider the following attribute : age. In a survey, a person's age is marked down using the variable $a \in 1, 2, 3$. $a = 1$ means the person is between 0 and 10 years old, $a = 2$ means the person is between 10 and 20 years old, $a = 3$ means the person is older than 20. One way to transform the variable a into a binary representation would be to use three binary variables (a_1, a_2, a_3) with $(1, 0, 0)$, $(0, 1, 0)$, $(0, 0, 1)$ representing $a = 1, a = 2, a = 3$ respectively. This is called $1 - of - M$ *coding* since only 1 of the binary variables is active in encoding the M states. By construction, means that the variables a_1, a_2, a_3 are dependent – for example, if we know that $a_1 = 1$, we know that $a_2 = 0$ and $a_3 = 0$. Regardless of any class conditioning, these variables will always be dependent, contrary to the assumption of Naive Bayes. A correct approach is to use variables with more than two states, as explained in section(10.2.2).

10.2.2 Multi-state variables

For a variable x_i with more than two states, $\text{dom}(x_i) = \{1, \ldots, S\}$, the likelihood of observing a state $x_i = s$ is denoted

$$p(x_i = s|c) = \theta^i_s(c) \tag{10.2.17}$$

with $\sum_s p(x_i = s|c) = 1$. For a set of data vectors $\mathbf{x}^n$,$n = 1, \ldots N$, belonging to class c, under the i.i.d. assumption, the likelihood of the NB model generating data from class c is

$$\prod_{n=1}^{N} p(\mathbf{x}^n|c^n) = \prod_{n=1}^{N}\prod_{i=1}^{D}\prod_{s=1}^{S}\prod_{c=1}^{C} \theta^i_s(c)^{\mathbb{I}[x_i^n=s]\mathbb{I}[c^n=c]} \tag{10.2.18}$$

which gives the class conditional log-likelihood

$$L = \sum_{n=1}^{N}\sum_{i=1}^{D}\sum_{s=1}^{S}\sum_{c=1}^{C} \mathbb{I}[x_i^n = s]\,\mathbb{I}[c^n = c] \log \theta^i_s(c) \tag{10.2.19}$$

We can optimize with respect to the parameters θ using a Lagrange multiplier (one for each of the attributes i and classes c) to ensure normalisation:

$$L(\theta) = \sum_{n=1}^{N}\sum_{i=1}^{D}\sum_{s=1}^{S}\sum_{c=1}^{C} \mathbb{I}[x_i^n = s]\,\mathbb{I}[c^n = c] \log \theta^i_s(c) + \sum_{c=1}^{C}\sum_{i=1}^{D} \lambda_i^c \left(1 - \sum_{s=1}^{S} \theta^i_s(c)\right) \tag{10.2.20}$$

To find the optimum of this function we may differentiate with respect to $\theta^i_s(c)$ and equate to zero. Solving the resulting equation we obtain

$$\sum_{n=1}^{N} \frac{\mathbb{I}[x_i^n = s]\,\mathbb{I}[c^n = c]}{\theta^i_s(c)} = \lambda_i^c \tag{10.2.21}$$

Hence, by normalisation,

$$\theta^i_s(c) = p(x_i = s|c) = \frac{\sum_n \mathbb{I}[x_i^n = s]\,\mathbb{I}[c^n = c]}{\sum_{s',n'} \mathbb{I}\left[x_i^{n'} = s'\right]\mathbb{I}[c^{n'} = c]} \tag{10.2.22}$$

The Maximum Likelihood setting for the parameter $p(x_i = s|c)$ equals the relative number of times that attribute i is in state s for class c.

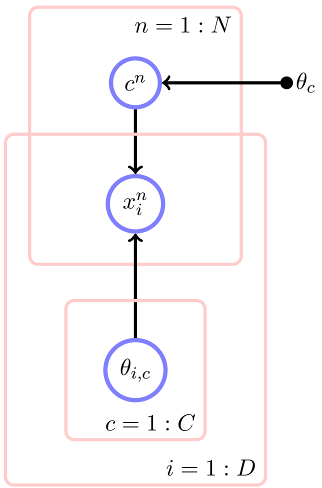


Figure 10.3: Bayesian Naive Bayes with a factorised prior on the class conditional attribute probabilities $p(x_i = s|c)$. For simplicity we assume that the class probability $\theta_c \equiv p(c)$ is learned with Maximum Likelihood, so that no distribution is placed over this parameter.

10.2.3 Text classification

Consider a set of documents about politics, and another set about sport. Our interest is to make a method that can automatically classify a new document as pertaining to either sport or politics. We search through both sets of documents to find the 100 most commonly occurring words. Each document is then represented by a 100 dimensional vector representing the number of times that each of the words occurs in that document – the so called *bag of words* representation (this is a crude representation of the document since it discards word order). A Naive Bayes model specifies a distribution of these number of occurrences $p(x_i|c)$, where x_i is the count of the number of times word i appears in documents of type c. One can achieve this using either a multistate representation (as discussed in section(10.2.2)) or using a continuous x_i to represent the frequency of word i in the document. In this case $p(x_i|c)$ could be conveniently modelled using for example a Beta distribution.

Despite the simplicity of Naive Bayes, it can classify documents surprisingly well[123]. Intuitively a potential justification for the conditional independence assumption is that if we know a document is about politics, this is a good indication of the kinds of other words we will find in the document. Because Naive Bayes is a reasonable classifier in this sense, and has minimal storage and fast training, it has been applied to time-storage critical applications, such as automatically classifying webpages into types[286], and spam filtering[9].

10.3 Bayesian Naive Bayes

To predict the class c of an input $\mathbf{x}$ we use

$$p(c|\mathbf{x}, \mathcal{D}) \propto p(\mathbf{x}, \mathcal{D}, c)p(c|\mathcal{D}) \propto p(\mathbf{x}|\mathcal{D}, c)p(c|\mathcal{D}) \tag{10.3.1}$$

For convenience we will simply set $p(c|\mathcal{D})$ using Maximum Likelihood

$$p(c|\mathcal{D}) = \frac{1}{N}\sum_n \mathbb{I}[c^n = c] \tag{10.3.2}$$

However, as we've seen, setting the parameters of $p(\mathbf{x}|\mathcal{D}, c)$ using Maximum Likelihood training can yield over-confident predictions in the case of sparse data. A Bayesian approach that addresses this difficulty is to use priors on the probabilities $p(x_i = s|c) \equiv \theta_s^i(c)$ that discourage extreme values. The model is depicted in fig(10.3).

The prior

We will use a prior on the table entries and make the global factorisation assumption (see section(9.3))

$$p(\theta) = \prod_{i,c} p(\theta^i(c)) \tag{10.3.3}$$

We consider discrete x_i each of which take states from $1, \ldots, S$. In this case $p(x_i = s|c)$ corresponds to a multinomial distribution, for which the conjugate prior is a Dirichlet distribution. Under the factorised prior assumption (10.3.3) we define a prior for each attribute i and class c,

$$p(\theta^i(c)) = \text{Dirichlet}\left(\theta^i(c)|\mathbf{u}^i(c)\right) \tag{10.3.4}$$

where $\mathbf{u}^i(c)$ is the hyperparameter vector of the Dirichlet distribution for table $p(x_i|c)$.

The posterior

First let's see how the Bayesian approach is used to classify a novel point $\mathbf{x}^*$. Let $\mathcal{D}$ denote the training data $(\mathbf{x}^n, c^n), n = 1, \ldots, N$. From equation (10.3.14), the term $p(\mathbf{x}^*, \mathcal{D}|c^*)$ is computed using the following decomposition:

$$p(\mathbf{x}^*, \mathcal{D}|c^*) = \int_\theta p(\mathbf{x}^*, \mathcal{D}, \theta|c^*) = \int_\theta p(\mathbf{x}^*|\theta, \cancel{\mathcal{D}}, c^*)p(\theta, \mathcal{D}|\cancel{c^*}) \propto \int_{\theta(c^*)} p(\mathbf{x}^*|\theta(c^*))p(\theta(c^*)|\mathcal{D}) \tag{10.3.5}$$

Hence in order to make a prediction, we require the parameter posterior. Consistent with our general Bayesian BN training result in section(9.3), the parameter posterior factorises

$$p(\theta(c^*)|\mathcal{D}) = \prod_i p(\theta^i(c^*)|\mathcal{D}) \tag{10.3.6}$$

where

$$p(\theta^i(c^*)|\mathcal{D}) \propto p(\theta^i(c^*)) \prod_{n:c^n=c^*} p(x_i^n|\theta^i(c^*)) \tag{10.3.7}$$

By conjugacy, the posterior for class c^* is a Dirichlet distribution,

$$p(\theta^i(c^*)|\mathcal{D}) = \text{Dirichlet}\left(\theta^i(c^*)|\hat{\mathbf{u}}^i(c^*)\right) \tag{10.3.8}$$

where the vector $\hat{\mathbf{u}}^i(c^*)$ has components

$$\left[\hat{\mathbf{u}}^i(c^*)\right]_s = u_s^i(c^*) + \sum_{n:c^n=c^*} \mathbb{I}\left[x_i^n = s\right] \tag{10.3.9}$$

For Dirichlet hyperparameters $\mathbf{u}^i(c^*)$ the above equation updates the hyperparameter by the number of times variable i is in state s for class c^* data. A common default setting is to take all components of $\mathbf{u}$ to be 1.

Classification

The class distribution is given by

$$p(c^*|\mathbf{x}^*, \mathcal{D}) \propto p(c^*|\mathcal{D})p(\mathbf{x}^*|\mathcal{D}, c^*) = p(c^*|\mathcal{D}) \prod_i p(x_i^*|\mathcal{D}, c^*) \tag{10.3.10}$$

To compute $p(\mathbf{x}^*|\mathcal{D}, c^*)$ we use

$$p(x_i^* = s|\mathcal{D}, c^*) = \int_{\theta(c^*)} p(x_i^* = s, \theta(c^*)|\mathcal{D}, c^*) = \int_{\theta(c^*)} p(x_i^* = s|\theta(c^*))p(\theta(c^*)|\mathcal{D}) = \int_{\theta(c^*)} \theta_s^i(c^*)p(\theta(c^*)|\mathcal{D}) \tag{10.3.11}$$

Using the general identity

$$\int \theta_s \text{Dirichlet}\left(\theta|\mathbf{u}\right) d\theta = \frac{1}{Z(\mathbf{u})} \int \prod_{s'} \theta^{u_{s'}-1+\mathbb{I}[s'=s]} d\theta = \frac{Z(\mathbf{u}')}{Z(\mathbf{u})} \tag{10.3.12}$$

where $Z(\mathbf{u})$ is the normalisation constant of the Dirichlet distribution $\text{Dirichlet}\left(\cdot|\mathbf{u}\right)$ and

$$u'_s = \begin{cases} u_s & s \neq s' \\ u_s + 1 & s = s' \end{cases} \tag{10.3.13}$$

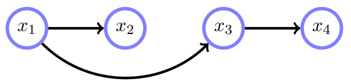


Figure 10.4: A Chow-Liu Tree in which each variable x_i has at most one parent. The variables may be indexed such that $1 \leq i \leq D$.

we obtain

$$p(c^*|\mathbf{x}^*, \mathcal{D}) \propto p(c^*|\mathcal{D}) \prod_i \frac{Z(\mathbf{u}^{*i}(c^*))}{Z(\hat{\mathbf{u}}^i(c^*))} \tag{10.3.14}$$

where

$$u_s^{*i}(c^*) = \hat{u}_s^i(c^*) + \mathbb{I}\left[x_i^* = s\right] \tag{10.3.15}$$

Example 49 (Bayesian Naive Bayes)**.** Repeating the previous analysis for the 'Are they Scottish?' data from example(48), the probability under a uniform Dirichlet prior for all the tables, gives a value of 0.236 for the probability that $(1, 0, 1, 1, 0)$ is Scottish, compared with a value of 0.192 under the standard Naive Bayes assumption.

10.4 Tree augmented Naive Bayes

A natural extension of Naive Bayes is to relax the assumption that the attributes are independent given the class:

$$p(\mathbf{x}|c) \neq \prod_{i=1}^{D} p(x_i|c) \tag{10.4.1}$$

The question then arises – which structure should we choose for $p(x|c)$? As we saw in section(9.3.5), learning a structure is computationally infeasible for all but very small numbers of attributes. A practical algorithm requires a specific form of constraint on the structure. To do this we first make a digression into the Maximum Likelihood learning of trees constrained to have at most a single parent.

10.4.1 Chow-Liu Trees

Consider a multivariate distribution $p(\mathbf{x})$ that we wish to approximate with a distribution $q(\mathbf{x})$. Furthermore, we constrain the approximation $q(x)$ to be a Belief Network in which each node has at most one parent. First we assume that we have chosen a particular labelling of the variables $1 \leq i \leq D$, for which the DAG single parent constraint means

$$q(x) = \prod_{i=1}^{D} q(x_i|x_{pa(i)}), \qquad pa(i) < i, \quad \text{or } pa(i) = \emptyset \tag{10.4.2}$$

where $pa(i)$ is the single parent index of node i. To find the best approximating distribution q in this constrained class, we may minimise the Kullback-Leibler divergence

$$\text{KL}(p|q) = \langle \log p(x) \rangle_{p(x)} - \sum_{i=1}^{D} \left\langle \log q(x_i|x_{pa(i)}) \right\rangle_{p(x_i, x_{pa(i)})} \tag{10.4.3}$$

Since $p(\mathbf{x})$ is fixed, the first term is constant. By adding a term $\left\langle \log p(x_i|x_{pa(i)}) \right\rangle_{p(x_i, x_{pa(i)})}$ that depends on $p(\mathbf{x})$ alone, we can write

$$\text{KL}(p|q) = \mathsf{const.} - \sum_{i=1}^{D} \left\langle \left\langle \log q(x_i|x_{pa(i)}) \right\rangle_{p(x_i|x_{pa(i)})} - \left\langle \log p(x_i|x_{pa(i)}) \right\rangle_{p(x_i|x_{pa(i)})} \right\rangle_{p(x_{pa(i)})} \tag{10.4.4}$$

Algorithm 7 Chow-Liu Trees

1: **for** $i = 1$ to D **do**
2:     **for** $j = 1$ to D **do**
3:         Compute the Mutual Information for the pair of variables x_i, x_j: $w_{ij} = \text{MI}(x_i; x_j)$
4:     **end for**
5: **end for**
6: For the undirected graph $\mathcal{G}$ with edge weights w, find a maximum weight undirected spanning tree $\mathcal{T}$
7: Choose an arbitrary variable as the root node of the tree $\mathcal{T}$.
8: Form a directed tree by orienting all edges away from the root node.

This enables us to recognise that, up to a negligible constant, the overall Kullback-Leibler divergence is a positive sum of individual Kullback-Leibler divergences so that the optimal setting is therefore

$$q(x_i|x_{pa(i)}) = p(x_i|x_{pa(i)}) \tag{10.4.5}$$

Plugging this solution into equation (10.4.3) and using $\log p(x_i|x_{pa(i)}) = \log p(x_i, x_{pa(i)}) - \log p(x_{pa(i)})$ we obtain

$$\text{KL}(p|q) = \text{const.} - \sum_{i=1}^{D} \left\langle \log p(x_i, x_{pa(i)}) \right\rangle_{p(x_i, x_{pa(i)})} + \sum_{i=1}^{D} \left\langle \log p(x_{pa(i)}) \right\rangle_{p(x_{pa(i)})} \tag{10.4.6}$$

We still need to find the optimal parental structure $pa(i)$ that minimises the above expression. If we add and subtract an entropy term we can write

$$\text{KL}(p|q) = -\sum_{i=1}^{D} \left\langle \log p(x_i, x_{pa(i)}) \right\rangle_{p(x_i, x_{pa(i)})} + \sum_{i=1}^{D} \left\langle \log p(x_{pa(i)}) \right\rangle_{p(x_{pa(i)})} + \sum_{i=1}^{D} \left\langle \log p(x_i) \right\rangle_{p(x_i)} - \sum_{i=1}^{D} \left\langle \log p(x_i) \right\rangle_{p(x_i)} + \text{const.} \tag{10.4.7}$$

For two variables x_i and x_j and distribution $p(x_i, x_j)$, the *mutual information* definition(87) can be written as

$$\text{MI}(x_i; x_j) = \left\langle \log \frac{p(x_i, x_j)}{p(x_i)p(x_j)} \right\rangle_{p(x_i, x_j)} \tag{10.4.8}$$

which can be seen as the Kullback-Leibler divergence $\text{KL}(p(x_i, x_j)|p(x_i)p(x_j))$ and is therefore non-negative. Using this, equation (10.4.7) is

$$\text{KL}(p|q) = -\sum_{i=1}^{D} \text{MI}\big(x_i; x_{pa(i)}\big) - \sum_{i=1}^{D} \left\langle \log p(x_i) \right\rangle_{p(x_i)} + \text{const.} \tag{10.4.9}$$

Since our task is to find the parental indices $pa(i)$, and the entropic term $\sum_i \left\langle \log p(x_i) \right\rangle_{p(x_i)}$ is independent of this mapping, finding the optimal mapping is equivalent to maximising the summed mutual informations

$$\sum_{i=1}^{D} \text{MI}\big(x_i; x_{pa(i)}\big) \tag{10.4.10}$$

under the constraint that $pa(i) \leq i$. Since we also need to choose the optimal initial labelling of the variables as well, the problem is equivalent to computing all the pairwise mutual informations

$$w_{ij} = \text{MI}(x_i; x_j) \tag{10.4.11}$$

and then finding a maximal spanning tree for the graph with edge weights w (see `spantree.m`). This can be thought of as a form of breadth-first-search[60]. Once found, we need to identify a directed tree with at most one parent. This is achieved by choosing an arbitrary node and then orienting edges consistently away from this node.

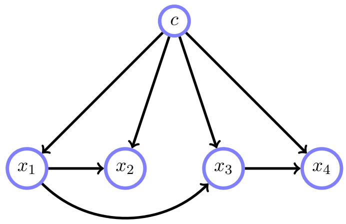


Figure 10.5: Tree Augmented Naive (TAN) Bayes. Each variable x_i has at most one parent. The Maximum Likelihood optimal TAN structure is computed using a modified Chow-Liu algorithm in which the conditional mutual information $\text{MI}(x_i; x_j|c)$ is computed for all i, j. A maximum weight spanning tree is then found and turned into a directed graph by orienting the edges outwards from a chosen root node. The table entries can then be read off using the usual Maximum Likelihood counting argument.

Maximum likelihood Chow-Liu trees

If $p(x)$ is the empirical distribution

$$p(x) = \frac{1}{N}\sum_{n=1}^{N} \delta(x, x^n) \tag{10.4.12}$$

then

$$\text{KL}(p|q) = \text{const.} - \frac{1}{N}\sum_{n} \log q(x^n) \tag{10.4.13}$$

Hence the approximation q that minimises the Kullback-Leibler divergence between the empirical distribution and p is equivalent to that which maximises the likelihood of the data. This means that if we use the mutual information found from the empirical distribution, with

$$p(x_i = \mathsf{a}, x_j = \mathsf{b}) \propto \sharp(x_i = \mathsf{a}, x_j = \mathsf{b}) \tag{10.4.14}$$

then the Chow-Liu tree produced corresponds to the Maximum Likelihood solution amongst all single-parent trees. An outline of the algorithm is given in algorithm(7). An efficient algorithm for sparse data is also available[187].

Remark 9 (Learning Tree structured Belief Networks)**.** The Chow-Liu algorithm pertains to the discussion in section(9.3.5) on learning the structure of Belief Networks from data. Under the special constraint that each variable has at most one parent, the Chow-Liu algorithm returns the Maximum Likelihood structure to fit the data.

10.4.2 Learning tree augmented Naive Bayes networks

For a distribution $p(x|c)$ of the form of a tree structure with a single-parent constraint we can readily find the class conditional Maximum Likelihood solution by computing the Chow-Liu tree for each class. One then adds links from the class node c to each variable and learns the class conditional probabilities from c to x, which can be read off for Maximum Likelihood using the usual counting argument. Note that this would generally result in a different Chow-Liu tree for each class.

Practitioners typically constrain the network to have the same structure for all classes. The Maximum Likelihood objective under the TAN constraint then corresponds to maximising the *conditional mutual information*[94]

$$\text{MI}(x_i; x_j|c) = \langle \text{KL}(p(x_i, x_j|c)|p(x_i|c)p(x_j|c))\rangle_{p(c)} \tag{10.4.15}$$

see exercise(136). Once the structure is learned one subsequently sets parameters by Maximum Likelihood counting. Techniques to prevent overfitting are discussed in [94] and can be addressed using Dirichlet priors, as for the simpler Naive Bayes structure.

One can readily consider less restrictive structures than single-parent Belief Networks. However, the complexity of finding optimal BN structures is generally computationally infeasible and heuristics are required to limit the search space.

10.5 Code

`NaiveBayesTrain.m`: Naive Bayes trained with Maximum Likelihood
`NaiveBayesTest.m`: Naive Bayes test
`NaiveBayesDirichletTrain.m`: Naive Bayes trained with Bayesian Dirichlet
`NaiveBayesDirichletTest.m`: Naive Bayes testing with Bayesian Dirichlet

`demoNaiveBayes.m`: Demo of Naive Bayes

10.6 Exercises

Exercise 130. *A local supermarket specializing in breakfast cereals decides to analyze the buying patterns of its customers. They make a small survey asking 6 randomly chosen people their age (older or younger than 60 years) and which of the breakfast cereals (Cornflakes, Frosties, Sugar Puffs, Branflakes) they like. Each respondent provides a vector with entries 1 or 0 corresponding to whether they like or dislike the cereal. Thus a respondent with* (1101) *would like Cornflakes, Frosties and Branflakes, but not Sugar Puffs. The older than 60 years respondents provide the following data* $(1000), (1001), (1111), (0001)$*. The younger than 60 years old respondents responded* $(0110), (1110)$*. A novel customer comes into the supermarket and says she only likes Frosties and Sugar Puffs. Using Naive Bayes trained with maximum likelihood, what is the probability that she is younger than 60?*

Exercise 131. *A psychologist does a small survey on 'happiness'. Each respondent provides a vector with entries 1 or 0 corresponding to whether they answer 'yes' to a question or 'no', respectively. The question vector has attributes*

$$\mathbf{x} = (rich, married, healthy) \tag{10.6.1}$$

Thus, a response $(1, 0, 1)$ *would indicate that the respondent was 'rich', 'unmarried', 'healthy'. In addition, each respondent gives a value* $c = 1$ *if they are content with their lifestyle, and* $c = 0$ *if they are not. The following responses were obtained from people who claimed also to be 'content' :* $(1, 1, 1), (0, 0, 1), (1, 1, 0), (1, 0, 1)$ *and for 'not content':* $(0, 0, 0), (1, 0, 0), (0, 0, 1), (0, 1, 0), (0, 0, 0)$.

1. *Using Naive Bayes, what is the probability that a person who is 'not rich', 'married' and 'healthy' is 'content'?*

2. *What is the probability that a person who is 'not rich' and 'married' is 'content'? (That is, we do not know whether or not they are 'healthy').*

3. *Consider the following vector of attributes :*

$$x_1 = 1 \text{ if customer is younger than 20 } ; x_1 = 0 \text{ otherwise} \tag{10.6.2}$$

$$x_2 = 1 \text{ if customer is between 20 and 30 years old } ; x_2 = 0 \text{ otherwise} \tag{10.6.3}$$

$$x_3 = 1 \text{ if customer is older than 30 } ; x_3 = 0 \text{ otherwise} \tag{10.6.4}$$

$$x_4 = 1 \text{ if customer walks to work } ; x_4 = 0 \text{ otherwise} \tag{10.6.5}$$

Each vector of attributes has an associated class label 'rich' or 'poor'. Point out any potential difficulties with using your previously described approach to training using Naive Bayes. Hence describe how to extend your previous Naive Bayes method to deal with this dataset.

Exercise 132. *Whizzco decide to make a text classifier. To begin with they attempt to classify documents as either sport or politics. They decide to represent each document as a (row) vector of attributes describing the presence or absence of words.*

$$\mathbf{x} = (\text{goal}, \text{football}, \text{golf}, \text{defence}, \text{offence}, \text{wicket}, \text{office}, \text{strategy}) \tag{10.6.6}$$

Training data from sport documents and from politics documents is represented below in MATLAB *using a matrix in which each row represents the 8 attributes.*

```
xP=[1 0 1 1 1 0 1 1;  % Politics
    0 0 0 1 0 0 1 1;
    1 0 0 1 1 0 1 0;
    0 1 0 0 1 1 0 1;
    0 0 0 1 1 0 1 1;
    0 0 0 1 1 0 0 1]
```

```
xS=[1 1 0 0 0 0 0 0; % Sport
    0 0 1 0 0 0 0 0;
    1 1 0 1 0 0 0 0;
    1 1 0 1 0 0 0 1;
    1 1 0 1 1 0 0 0;
    0 0 0 1 0 1 0 0;
    1 1 1 1 1 0 1 0]
```

Using a Naive Bayes classifier, what is the probability that the document $\mathbf{x} = (1, 0, 0, 1, 1, 1, 1, 0)$ *is about politics?*

Exercise 133. *A Naive Bayes Classifier for binary attributes* $x_i \in \{0, 1\}$ *is parameterised by* $\theta_i^1 = p(x_i = 1|class = 1)$*,* $\theta_i^0 = p(x_i = 1|class = 0)$*, and* $p_1 = p(class = 1)$ *and* $p_0 = p(class = 0)$*. Show that the decision boundary to classify a datapoint* $\mathbf{x}$ *can be written as* $\mathbf{w}^{\mathsf{T}}\mathbf{x} + b > 0$*, and state explicitly* $\mathbf{w}$ *and* b *as a function of* $\boldsymbol{\theta}^1, \boldsymbol{\theta}^0, p_1, p_0$*.*

Exercise 134. *This question concerns spam filtering. Each email is represented by a vector*

$$x = (x_1, \ldots, x_D) \tag{10.6.7}$$

where $x_i \in \{0, 1\}$*. Each entry of the vector indicates if a particular symbol or word appears in the email. The symbols/words are*

$$money, cash, !!!, viagra, \ldots, \text{etc.} \tag{10.6.8}$$

So that $x_2 = 1$ *if the word 'cash' appears in the email. The training dataset consists of a set of vectors along with the class label* c*, where* $c = 1$ *indicates the email is spam, and* $c = 0$ *not spam. Hence, the training set consists of a set of pairs* $(x^n, c^n), n = 1, \ldots, N$*. The Naive Bayes model is given by*

$$p(c, x) = p(c) \prod_{i=1}^{D} p(x_i|c) \tag{10.6.9}$$

1. *Draw a Belief Network for this distribution.*

2. *Derive expressions for the parameters of this model in terms of the training data using Maximum Likelihood. Assume that the data is independent and identically distributed*

$$p(c^1, \ldots, c^N, x^1, \ldots, x^N) = \prod_{n=1}^{N} p(c^n, x^n) \tag{10.6.10}$$

   Explicitly, the parameters are

$$p(c = 1), p(x_i = 1|c = 1), p(x_i = 1|c = 0), i = 1, \ldots, D \tag{10.6.11}$$

3. *Given a trained model* $p(x, c)$*, explain how to form a classifier* $p(c|x)$*.*

4. *If 'viagra' never appears in the spam training data, discuss what effect this will have on the classification for a new email that contains the word 'viagra'.*

5. *Write down an expression for the decision boundary*

$$p(c = 1|x) = p(c = 0|x) \tag{10.6.12}$$

   and show that it can be written in the form

$$\sum_{d=1}^{D} u_d x_d - b = 0 \tag{10.6.13}$$

   for suitably defined u *and* b*.*

Exercise 135. *For a distribution $p(x, c)$ and an approximation $q(x, c)$, show that when $p(x, c)$ corresponds to the empirical distribution, finding $q(x, c)$ that minimises the Kullback-Leibler divergence*

$$KL(p(x, c)|q(x, c)) \tag{10.6.14}$$

corresponds to Maximum Likelihood training of $q(x, c)$.

Exercise 136. *Consider a distribution $p(x, c)$ and a Tree Augmented approximation*

$$q(x, c) = q(c) \prod_i q(x_i|x_{pa(i)}, c), \qquad pa(i) < i \quad \text{or } pa(i) = \emptyset \tag{10.6.15}$$

Show that for the optimal $q(x, c)$ constrained as above, the solution $q(x, c)$ that minimises $KL(p(x, c)|q(x, c))$ when plugged back into the Kullback-Leibler expression gives, as a function of the parental structure,

$$KL(p(x, c)|q(x, c)) = -\sum_i \left\langle \log \frac{p(x_i, x_{pa(i)}|c)}{p(x_{pa(i)}|c)p(x_i|c)} \right\rangle_{p(x_i, x_{pa(i)}, c)} + \mathsf{const.} \tag{10.6.16}$$

This shows that under the single-parent constraint and that each tree $q(x|c)$ has the same structure, minimising the Kullback-Leibler divergence is equivalent to maximising the sum of conditional mutual information terms.

Exercise 137. *Write a* MATLAB *routine* `A = ChowLiu(X)` *where X is a $D \times N$ data matrix containing a multivariate datapoint on each column that returns a Chow-Liu Maximum Likelihood tree for X. The tree structure is to be returned in the sparse matrix A. You may find the routine* `spantree.m` *useful. The file* `ChowLiuData.mat` *contains a data matrix for 10 variables. Use your routine to find the Maximum Likelihood Chow Liu tree, and draw a picture of the resulting DAG with edges oriented away from variable 1.*

CHAPTER 11

Learning with Hidden Variables

11.1 Hidden Variables and Missing Data

In practice data entries are often missing resulting in incomplete information to specify a likelihood. Observational variables may be split into *visible* (those for which we actually know the state) and *missing* (those whose states would nominally be known but are missing for a particular datapoint).

Another scenario in which not all variables in the model are observed are the so-called hidden or *latent variable models*. In this case there are variables which are essential for the model description but never observed. For example, the underlying physics of a model may contain latent processes which are essential to describe the model, but cannot be directly measured.

11.1.1 Why hidden/missing variables can complicate proceedings

In learning the parameters of models as previously described in chapter(9), we assumed we have complete information to define all variables of the joint model of the data $p(v|\theta)$. Consider the Asbestos-Smoking-Cancer network of section(9.2.3). In the case of complete data, the likelihood is

$$p(v^n|\theta) = p(a^n, s^n, c^n|\theta) = p(c^n|a^n, s^n, \theta_c)p(a^n|\theta_a)p(s^n|\theta_s) \tag{11.1.1}$$

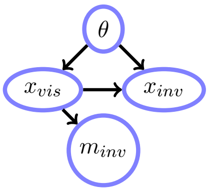
which is factorised in terms of the table entry parameters. We exploited this property to show that table entries θ can be learned by considering only local information, both in the Maximum Likelihood and Bayesian frameworks.

Now consider the case that for some of the patients, only partial information is available. For example, for patient n with record $v^n = \{c = 1, s = 1\}$ it is known that the patient has cancer and is a smoker, but whether or not they had exposure to asbestos is unknown. Since we can only use the 'visible' available information is it would seem reasonable to assess parameters using the marginal likelihood

$$p(v^n|\theta) = \sum_a p(a, s^n, c^n|\theta) = \sum_a p(c^n|a, s^n, \theta_c)p(a|\theta_a)p(s^n|\theta_s) \tag{11.1.2}$$

We will discuss when this approach is valid in section(11.1.2). Using the marginal likelihood may result in computational difficulties since equation (11.1.2) is not factorised over the tables. This means that the likelihood function cannot be written as a product of functions, one for each separate parameter. In this case the maximisation of the likelihood is more complex since the parameters of different tables are coupled.

A similar complication holds for Bayesian learning. As we saw in section(13), under a prior factorised over each CPT θ, the posterior is also factorised. However, in the case of unknown asbestos exposure, a term



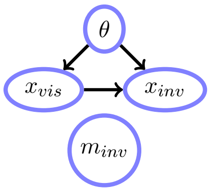


Figure 11.1: **(a)**: Missing at random assumption. **(b)**: Missing completely at random assumption.

is introduced of the form

$$p(v^n|\theta) = \sum_a p(c^n|a, s^n, \theta_c)p(a|\theta_a)p(s^n|\theta_s) = p(s^n|\theta_s)\sum_a p(c^n|a, s^n, \theta_c)p(a|\theta_a) \tag{11.1.3}$$

which cannot be written as a product of a functions of $f_s(\theta_s)f_a(\theta_a)f_c(\theta_c)$. The missing variable therefore introduces dependencies in the posterior parameter distribution, making the posterior more complex.

In both the Maximum Likelihood and Bayesian cases, one has a well defined likelihood function of the table parameters/posterior. The difficulty is therefore not conceptual, but rather computational – how are we to find the optimum of the likelihood/summarise the posterior?

Note that missing data does not always make the parameter posterior non-factorised. For example, if the cancer state is unobserved above, because cancer is a collider with no descendants, the conditional distribution simply sums to 1, and one is left with a factor dependent on a and another on s.

11.1.2 The missing at random assumption

Under what circumstances is it valid to use the marginal likelihood to assess parameters? We partition the variables x into those that are 'visible', x_{vis} and 'invisible', x_{inv}, so that the set of all variables can be written $x = [x_{vis}, x_{inv}]$. For the visible variables we have an observed state $x_{vis} = \mathsf{v}$, whereas the state of the invisible variables is unknown. We use an indicator $m_{inv} = 1$ to denote that the state of the invisible variables is unknown. Then for a datapoint which contains both visible and invisible information,

$$p(x_{vis} = \mathsf{v}, m_{inv} = 1|\theta) = \sum_{x_{inv}} p(x_{vis} = \mathsf{v}, x_{inv}, m_{inv} = 1|\theta) \tag{11.1.4}$$

$$= \sum_{x_{inv}} p(m_{inv} = 1|x_{vis} = \mathsf{v}, x_{inv}, \theta)p(x_{vis} = \mathsf{v}, x_{inv}|\theta) \tag{11.1.5}$$

If we assume that the mechanism which generates invisible data has the form

$$p(m_{inv} = 1|x_{vis} = \mathsf{v}, x_{inv}, \theta) = p(m_{inv} = 1|x_{vis} = \mathsf{v}) \tag{11.1.6}$$

then

$$p(x_{vis} = \mathsf{v}, m_{inv} = 1|\theta) = p(m_{inv} = 1|x_{vis} = \mathsf{v})\sum_{x_{inv}} p(x_{vis} = \mathsf{v}, x_{inv}|\theta) \tag{11.1.7}$$

$$= p(m_{inv} = 1|x_{vis} = \mathsf{v})p(x_{vis} = \mathsf{v}|\theta) \tag{11.1.8}$$

Only the term $p(x_{vis} = \mathsf{v}|\theta)$ conveys information about the model. Therefore, provided the mechanism by which the data is missing depends only on the visible states, we may simply use the marginal likelihood to assess parameters. This is called the *missing at random* assumption.

Example 50 (Not missing at random)**.** EZsurvey.org stop men on the street and ask them their favourite colour. All men whose favourite colour is pink decline to respond to the question – for any other colour, all men respond to the question. Based on the data, EZsurvey.org produce a histogram of men's favourite colour, based on the likelihood of the visible data alone, confidently stating that none of them likes pink.

For simplicity, assume there are only three colours, blue, green and pink. EZsurvey.org attempts to find the histogram with probabilities $\theta_b, \theta_g, \theta_p$ with $\theta_b + \theta_g + \theta_p = 1$. Each respondent produces a visible response x_c with $\text{dom}(x_c) = \{\textsf{blue}, \textsf{green}, \textsf{pink}\}$, otherwise $m_c = 1$ if there is no response. Three men are asked their favourite colour, giving data

$$\left\{x_c^1, x_c^2, x_c^3\right\} = \{\textsf{blue}, \textsf{missing}, \textsf{green}\} \tag{11.1.9}$$

Based on the likelihood of the visible data alone we have the log likelihood for i.i.d. data

$$L(\theta_b, \theta_g, \theta_p) = \log \theta_b + \log \theta_g + \lambda \left(1 - \theta_b - \theta_g - \theta_p\right) \tag{11.1.10}$$

where the last Lagrange term ensures normalisation. Maximising the expression we arrive at (see exercise(145))

$$\theta_b = \frac{1}{2}, \theta_g = \frac{1}{2}, \theta_p = 0 \tag{11.1.11}$$

The unreasonable result that EZsurvey.org produce is due to not accounting correctly for the mechanism which produces the data.

The correct mechanism that generates the data (including the missing data is)

$$p(c^1 = \textsf{blue}|\theta)p(m_c^2 = 1|\theta)p(c^3 = \textsf{green}|\theta) = \theta_b\theta_p\theta_g = \theta_b\left(1 - \theta_b - \theta_g\right)\theta_g \tag{11.1.12}$$

where we used $p(m_c^2 = 1|\theta) = \theta_p$ since the probability that a datapoint is missing is the same as the probability that the favourite colour is pink. Maximising the likelihood, we arrive at

$$\theta_b = \frac{1}{3}, \theta_g = \frac{1}{3}, \theta_p = \frac{1}{3} \tag{11.1.13}$$

as we would expect. On the other hand if there is another visible variable, t, denoting the time of day, and the probability that men respond to the question depends only on the time t alone (for example the missing probability is high during rush hour), then we may indeed treat the missing data as missing at random.

A stronger assumption than MAR is

$$p(m_{inv} = 1|x_{vis} = \textsf{v}, x_{inv}, \theta) = p(m_{inv} = 1) \tag{11.1.14}$$

which is called *missing completely at random*. This applies for example to latent variable models in which the variable state is always missing, independent of anything else.

11.1.3 Maximum Likelihood

Throughout the remaining discussion we will assume any missing data is MAR or missing completely at random. This means that we can treat any unobserved variables by summing (or integrating) over their states. For Maximum Likelihood we learn model parameters θ by optimising the *marginal likelihood*

$$p(v|\theta) = \sum_h p(v, h|\theta) \tag{11.1.15}$$

with respect to θ.

11.1.4 Identifiability issues

The marginal likelihood objective function depends on the parameters only through $p(v|\theta)$, so that equivalent parameter solutions may exist. For example, consider a latent variable problem with distribution

$$p(x_1, x_2|\theta) = \theta_{x_1,x_2} \tag{11.1.16}$$

in which variable x_2 is never observed. This means that the marginal likelihood only depends on the entry $p(x_1|\theta) = \sum_{x_2} \theta_{x_1,x_2}$. Given a Maximum Likelihood solution θ^*, we can then always find an equivalent Maximum Likelihood solution θ' provided (see exercise(146))

$$\sum_{x_2} \theta'_{x_1,x_2} = \sum_{x_2} \theta^*_{x_1,x_2} \tag{11.1.17}$$

In other cases there is an inherent symmetry in the parameter space of the marginal likelihood. For example, consider the network over binary variables

$$p(c, a, s) = p(c|a, s)p(a)p(s) \tag{11.1.18}$$

Our aim is to learn the table

$$\hat{p}(a = 1) \tag{11.1.19}$$

and the four tables

$$\hat{p}(c = 1|a = 1, s = 1), \quad \hat{p}(c = 1|a = 1, s = 0), \quad \hat{p}(c = 1|a = 0, s = 1), \quad \hat{p}(c = 1|a = 0, s = 0) \tag{11.1.20}$$

where we used a $\hat{\cdot}$ to denote that these are parameter estimates.

We assume that we have missing data such that the states of variable a are never observed. In this case an equivalent solution (in the sense that it has the same marginal likelihood) is given by interchanging the states of a:

$$\hat{p}'(a = 0) = \hat{p}(a = 1) \tag{11.1.21}$$

and the four tables

$$\hat{p}'(c = 1|a = 0, s = 1) = \hat{p}(c = 1|a = 1, s = 1), \qquad \hat{p}'(c = 1|a = 0, s = 0) = \hat{p}(c = 1|a = 1, s = 0)$$
$$\hat{p}'(c = 1|a = 1, s = 1) = \hat{p}(c = 1|a = 0, s = 1), \qquad \hat{p}'(c = 1|a = 1, s = 0) = \hat{p}(c = 1|a = 0, s = 0)$$

A similar situation occurs in a more general setting in which the state of a variable is consistently unobserved (mixture models are a case in point) yielding an inherent symmetry in the solution space. A well known characteristic of Maximum Likelihood algorithms is that 'jostling' occurs in the initial stages of training in which these symmetric solutions compete.

11.2 Expectation Maximisation

The EM algorithm is a convenient and general purpose iterative approach to maximising the likelihood under missing data/hidden variables[184]. It is generally straightforward to implement and can achieve large jumps in parameter space, particularly in the initial iterations.

11.2.1 Variational EM

The key feature of the EM algorithm is to form an alternative objective function for which the parameter coupling effect discussed in section(11.1.1) is removed, meaning that individual parameter updates can be achieved, akin to the case of fully observed data. The way this works is to replace the marginal likelihood with a lower bound – it is this lower bound that has the decoupled form.

We first consider a single variable pair (v, h), where v stands for 'visible' and h for 'hidden'. To derive the bound on the marginal likelihood, consider the Kullback-Leibler divergence between a 'variational' distribution $q(h|v)$ and the parametric model $p(h|v, \theta)$:

$$\mathrm{KL}(q(h|v)|p(h|v, \theta)) \equiv \langle \log q(h|v) - \log p(h|v, \theta) \rangle_{q(h|v)} \geq 0 \tag{11.2.1}$$

The term 'variational' refers to the fact that this distribution will be a parameter of an optimisation problem. Using Bayes' rule, $p(h|v,\theta) = p(h,v|\theta)/p(v|\theta)$ and the fact that $p(v|\theta)$ does not depend on h,

$$\langle \log q(h|v)\rangle_{q(h|v)} - \langle \log p(h,v|\theta)\rangle_{q(h|v)} + \log p(v|\theta) = \mathrm{KL}(q(h|v)|p(h|v,\theta)) \geq 0 \tag{11.2.2}$$

Rearranging, we obtain a bound on the marginal likelihood[1]

$$\log p(v|\theta) \geq \underbrace{- \langle \log q(h|v)\rangle_{q(h|v)}}_{\text{Entropy}} + \underbrace{\langle \log p(h,v|\theta)\rangle_{q(h|v)}}_{\text{Energy}} \tag{11.2.3}$$

The bound is potentially useful since it is similar in form to the fully observed case, except that terms with missing data have their log likelihood weighted by a prefactor. Equation(11.2.3) is a marginal likelihood bound for a single training example. Under the i.i.d. assumption, the log likelihood of all training data $\mathcal{V} = \left\{v^1, \ldots, v^N\right\}$ is the sum of the individual log likelihoods:

$$\log p(\mathcal{V}|\theta) = \sum_{n=1}^{N} \log p(v^n|\theta) \tag{11.2.4}$$

Summing over the training data, we obtain a bound on the log (marginal) likelihood

$$\log p(\mathcal{V}|\theta) \geq \underbrace{- \sum_{n=1}^{N} \langle \log q(h^n|v^n)\rangle_{q(h^n|v^n)}}_{Entropy} + \underbrace{\sum_{n=1}^{N} \langle \log p(h^n,v^n|\theta)\rangle_{q(h^n|v^n)}}_{Energy} \tag{11.2.5}$$

Note that the bound is exact (that is, the right hand side is equal to the log likelihood) when we set $q(h^n|v^n) = p(h^n|v^n,\theta), n = 1, \ldots, N$.

The bound suggests an iterative procedure to optimise θ:

E-step For fixed θ, find the distributions $q(h^n|v^n)$ that maximise equation (11.2.5).

M-step For fixed $\{q(h^n|v^n), n = 1, \ldots, N\}$, find the parameters θ that maximise equation (11.2.5).

11.2.2 Classical EM

In the variational E-step above, the fully optimal setting is

$$q(h^n|v^n) = p(h^n|v^n,\theta) \tag{11.2.6}$$

Since q does not depend on θ_{new} the M-step is equivalent to maximising the energy term alone, see algorithm(8).

Example 51 (A one-parameter one-state example)**.** We consider a model small enough that we can plot fully the evolution of the EM algorithm. The model is on a single visible variable v and single two-state hidden variable $h \in \{1,2\}$. Defining a model $p(v,h) = p(v|h)p(h)$ with

$$p(v|h,\theta) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(v-\theta h)^2} \tag{11.2.7}$$

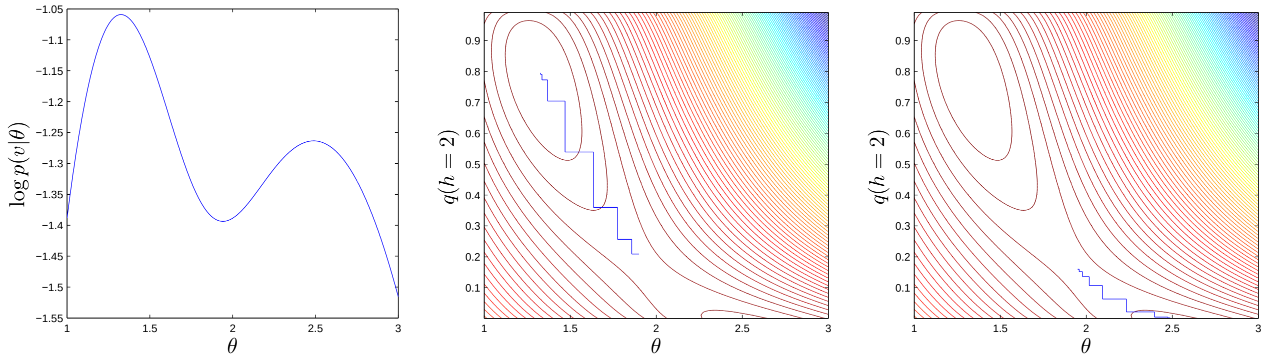
and $p(h=1) = p(h=2) = 0.5$. For an observation $v = 2.75$ and $\sigma^2 = 0.5$ our interest is to find the parameter θ that optimises the likelihood

$$p(v=2.75|\theta) = \frac{1}{2\sqrt{\pi}} \sum_{h=1,2} e^{-(2.75-\theta h)^2} \tag{11.2.8}$$

[1] This is analogous to a standard partition function bound in statistical physics, from where the terminology 'energy' and 'entropy' hails.

Algorithm 8 Expectation Maximisation. Compute Maximum Likelihood value for data with hidden variables. Input: a distribution $p(x|\theta)$ and dataset $\mathcal{V}$. Returns ML setting of θ.

$t = 0$ ▷ Iteration counter
Choose an initial setting for the parameters θ^0. ▷ Initialisation
while θ not converged (or likelihood not converged) **do**
  $t \leftarrow t + 1$
  for $n = 1$ to N **do** ▷ Run over all datapoints
    $q_t^n(h^n|v^n) = p(h^n|v^n, \theta^{t-1})$ ▷ E step
  end for
  $\theta^t = \arg\max_\theta \sum_{n=1}^{N} \langle \log p(h^n, v^n|\theta) \rangle_{q_t^n(h^n|v^n)}$ ▷ M step
end while
return θ^t ▷ The max likelihood parameter estimate.



(a) (b) (c)

Figure 11.2: **(a)**: The log likelihood for the model described in example(51). **(b)**: Contours of the lower bound $LB(q(h = 2), \theta)$. For an initial choice $q(h = 2) = 0.5$ and $\theta = 1.9$, successive updates of the E (vertical) and M (horizontal) steps are plotted. **(c)**: Starting at $\theta = 1.95$, the EM algorithm converges to a local optimum.

The log likelihood is plotted in fig(11.2a) with optimum at $\theta = 1.325$. The EM procedure iteratively optimises the lower bound

$$\log p(v = 2.75|\theta) \geq LB(q(h = 2), \theta) \equiv -q(h = 1) \log q(h = 1) - q(h = 2) \log q(h = 2) - \sum_{h=1,2} q(h) (2.75 - \theta h)^2 + \log 2 \quad (11.2.9)$$

where $q(h = 1) = 1 - q(h = 2)$. From an initial starting θ, the EM algorithm finds the q distribution that optimises $L(q, \theta)$ (E-step) and then updates θ (M-step). Depending on the initial θ, the solution found is either a global or local optimum of the likelihood, see fig(11.2a,b).

The M-step is easy to work out analytically in this case with $\theta^{new} = v \langle h \rangle_{q(h)} / \langle h^2 \rangle_{q(h)}$. Similarly, the E-step sets $q^{new}(h) = p(h|v, \theta)$.

Example 52. Consider a simple model

$$p(x_1, x_2|\theta) \quad (11.2.10)$$

where $\mathrm{dom}(x_1) = \mathrm{dom}(x_2) = \{1, 2\}$. Assuming an unconstrained distribution

$$p(x_1, x_2|\theta) = \theta_{x_1,x_2}, \qquad \theta_{1,1} + \theta_{1,2} + \theta_{2,1} + \theta_{2,2} = 1 \tag{11.2.11}$$

our aim is to learn θ from the data $\mathbf{x}^1 = (1, 1)\,, \mathbf{x}^2 = (1, ?)\,, \mathbf{x}^3 = (?, 2)$. The energy term for the classical EM is

$$\log p(x_1 = 1, x_2 = 1|\theta) + \langle \log p(x_1 = 1, x_2|\theta)\rangle_{p(x_2|x_1=1,\theta^{old})} + \langle \log p(x_1, x_2 = 2|\theta)\rangle_{p(x_1|x_2=2,\theta^{old})} \tag{11.2.12}$$

Writing out fully each of the above terms on a separate line gives the energy

$$\log \theta_{1,1} \tag{11.2.13}$$
$$+p(x_2 = 1|x_1 = 1, \theta^{old}) \log \theta_{1,1} + p(x_2 = 2|x_1 = 1, \theta^{old}) \log \theta_{1,2} \tag{11.2.14}$$
$$+p(x_1 = 1|x_2 = 2, \theta^{old}) \log \theta_{1,2} + p(x_1 = 2|x_2 = 2, \theta^{old}) \log \theta_{2,2} \tag{11.2.15}$$

This expression resembles the standard log likelihood of fully observed data except that terms with missing data have their weighted log parameters. The parameters are conveniently decoupled in this bound (apart from the trivial normalisation constraint) so that finding the optimal parameters is straightforward. This is achieved by the M-step update which gives

$$\begin{array}{ll} \theta_{1,1} \propto 1 + p(x_2 = 1|x_1 = 1, \theta^{old}) & \theta_{1,2} \propto p(x_2 = 2|x_1 = 1, \theta^{old}) + p(x_1 = 1|x_2 = 2, \theta^{old}) \\ \theta_{2,1} = 0 & \theta_{2,2} \propto p(x_1 = 2|x_2 = 2, \theta^{old}) \end{array} \tag{11.2.16}$$

where $p(x_2|x_1, \theta^{old}) \propto \theta^{old}_{x_1,x_2}$ (E-step) *etc.* The E and M-steps are iterated till convergence.

The EM algorithm increases the likelihood

Whilst, by construction, the EM algorithm cannot decrease the bound on the likelihood, an important question is whether or not the log likelihood itself is necessarily increased by this procedure.

We use θ' for the new parameters, and θ for the previous parameters in two consecutive iterations. Using $q(h^n|v^n) = p(h^n|v^n, \theta)$ we see that as a function of the parameters, the lower bound for a single variable pair (v, h) depends on θ and θ':

$$LB(\theta'|\theta) \equiv -\langle \log p(h|v, \theta)\rangle_{p(h|v,\theta)} + \left\langle \log p(h, v|\theta')\right\rangle_{p(h|v,\theta)} \tag{11.2.17}$$

and

$$\log p(v|\theta') = LB(\theta'|\theta) + KL(p(h|v, \theta)|p(h|v, \theta')) \tag{11.2.18}$$

That is, the Kullback-Leibler divergence is the difference between the lower bound and the true likelihood. We may write

$$\log p(v|\theta) = LB(\theta|\theta) + \underbrace{KL(p(h|v, \theta)|p(h|v, \theta))}_{0} \tag{11.2.19}$$

Hence

$$\log p(v|\theta') - \log p(v|\theta) = \underbrace{LB(\theta'|\theta) - LB(\theta|\theta)}_{\geq 0} + \underbrace{KL(p(h|v, \theta)|p(h|v, \theta'))}_{\geq 0} \tag{11.2.20}$$

The first assertion is true since, by definition of the M-step, we search for a θ' which has a higher value for the bound than our starting value θ. The second assertion is true by the property of the Kullback-Leibler

s	c
1	1
0	0
1	1
1	0
1	1
0	0
0	1

Figure 11.3: A database containing information about being a Smoker (1 signifies the individual is a smoker), and lung Cancer (1 signifies the individual has lung Cancer). Each row contains the information for an individual, so that there are 7 individuals in the database.

divergence.

For more than a single datapoint, we simply sum each individual bound for $\log p(v^n|\theta)$. Hence we reach the important conclusion that the EM algorithm increases, not only the lower bound on the marginal likelihood, but the marginal likelihood itself (more correctly, the EM cannot decrease these quantities).

Shared Parameters and tables

The case of tables sharing parameters is essentially straightforward. According to the energy term, we need to identify all those terms in which the shared parameter occurs. The objective for the shared parameter is then the sum over all energy terms containing the shared parameter.

11.2.3 Application to Belief networks

Conceptually, the application of EM to training Belief Networks with missing data is straightforward. The battle is more notational than conceptual. We begin the development with an example, from which intuition about the general case can be gleaned.

Example 53. Consider the network.

$$p(a, c, s) = p(c|a, s)p(a)p(s) \tag{11.2.21}$$

for which we have a set of data, but that the states of variable a are never observed, see fig(11.3). Our goal is to learn the CPTs $p(c|a, s)$ and $p(a)$ and $p(s)$. To apply EM, algorithm(8) to this case, we first assume initial parameters θ_a^0, θ_s^0, θ_c^0.

The first E-step, for iteration $t = 1$ then defines a set of distributions on the hidden variables (here the hidden variable is a)

$$q_{t=1}^{n=1}(a) = p(a|c = 1, s = 1, \theta^0), \qquad q_{t=1}^{n=2}(a) = p(a|c = 0, s = 0, \theta^0) \tag{11.2.22}$$

and so on for the 7 training examples, $n = 2, \ldots, 7$. For notational convenience, we write $q_t^n(a)$ in place of $q_t^n(a|v^n)$.

We now move to the first M-step. The energy term for any iteration t is:

$$E(\theta) = \sum_{n=1}^{7} \langle \log p(c^n|a^n, s^n) + \log p(a^n) + \log p(s^n)\rangle_{q_t^n(a)} \tag{11.2.23}$$

$$= \sum_{n=1}^{7} \left\{ \langle \log p(c^n|a^n, s^n)\rangle_{q_t^n(a)} + \langle \log p(a^n)\rangle_{q_t^n(a)} + \log p(s^n) \right\} \tag{11.2.24}$$

The final term is the log likelihood of the variable s, and $p(s)$ appears explicitly only in this term. Hence, the usual maximum likelihood rule applies, and $p(s = 1)$ is simply given by the relative number of times that $s = 1$ occurs in the database, giving $p(s = 1) = 4/7$, $p(s = 0) = 3/7$.

The parameter $p(a = 1)$ occurs in the terms

$$\sum_n \{q_t^n(a=0) \log p(a=0) + q_t^n(a=1) \log p(a=1)\} \tag{11.2.25}$$

which, using the normalisation constraint is

$$\log p(a=0) \sum_n q_t^n(a=0) + \log(1 - p(a=0)) \sum_n q_t^n(a=1) \tag{11.2.26}$$

Differentiating with respect to $p(a = 0)$ and solving for the zero derivative we get

$$p(a=0) = \frac{\sum_n q_t^n(a=0)}{\sum_n q_t^n(a=0) + \sum_n q_t^n(a=1)} = \frac{1}{N} \sum_n q_t^n(a=0) \tag{11.2.27}$$

That is, whereas in the standard Maximum Likelihood estimate, we would have the real counts of the data in the above formula, here they have been replaced with our guessed values $q_t^n(a = 0)$ and $q_t^n(a = 1)$.

A similar story holds for $p(c = 1|a = 0, s = 1)$. The contribution of this term to the energy is

$$\sum_{n:c^n=1,s^n=1} q_t^n(a=0) \log p(c=1|a=0,s=1) + \sum_{n:c^n=0,s^n=1} q_t^n(a=0) \log (1 - p(c=1|a=0,s=1))$$

which is

$$\log p(c=1|a=0,s=1) \sum_{n:c^n=1,s^n=1} q_t^n(a=0) + \log(1 - p(c=1|a=0,s=1)) \sum_{n:c^n=0,s^n=1} q_t^n(a=0) \tag{11.2.28}$$

Optimising with respect to $p(c = 1|a = 0, s = 1)$ gives

$$p(c=1|a=0,s=1) = \frac{\sum_n \mathbb{I}[c^n=1] \mathbb{I}[s^n=1] q_t^n(a=0)}{\sum_n \mathbb{I}[c^n=1] \mathbb{I}[s^n=1] q_t^n(a=0) + \sum_n \mathbb{I}[c^n=0] \mathbb{I}[s^n=1] q_t^n(a=0)} \tag{11.2.29}$$

For comparison, the setting in the complete data case is

$$p(c=1|a=0,s=1) = \frac{\sum_n \mathbb{I}[c^n=1] \mathbb{I}[s^n=1] \mathbb{I}[a^n=0]}{\sum_n \mathbb{I}[c^n=1] \mathbb{I}[s^n=1] \mathbb{I}[a^n=0] + \sum_n \mathbb{I}[c^n=0] \mathbb{I}[s^n=1] \mathbb{I}[a^n=0]} \tag{11.2.30}$$

There is an intuitive relationship between these updates: in the missing data case we replace the indicators by the assumed distributions q.

Iterating the E and M steps, these equations will converge to a local likelihood optimum.

To minimise the notational burden, we assume that the structure of the missing variables is fixed throughout, this being equivalent therefore to a latent variable model. The form of the energy term for Belief Networks is

$$\sum_n \langle \log p(x^n) \rangle_{q_t(h^n|v^n)} = \sum_n \sum_i \langle \log p(x_i^n | \mathrm{pa}(x_i^n)) \rangle_{q_t(h^n|v^n)} \tag{11.2.31}$$

It is useful to define the following notation:

$$q_t^n(x) = q_t(h|v^n)\delta(v, v^n) \tag{11.2.32}$$

where $x = (v, h)$ represents all the variables in the distribution. This means that $q_t^n(x)$ sets the visible variables in the observed state, and defines a conditional distribution on the unobserved variables. We then define the mixture distribution

$$q_t(x) = \frac{1}{N} \sum_{n=1}^{N} q_t^n(x) \tag{11.2.33}$$

Algorithm 9 EM for Belief Networks. Input: a BN structure $p(x_i|\text{pa}(x_i))$, $i = 1, \ldots, K$ with empty tables, and dataset on the visible variables $\mathcal{V}$. Returns the Maximum Likelihood setting of tables.

```
t = 1                                                          ▷ Iteration counter
Set p_t(x_i|pa(x_i)) to initial values.                        ▷ Initialisation
while p(x_i|pa(x_i)) not converged (or likelihood not converged) do
    t ← t + 1
    for n = 1 to N do                                          ▷ Run over all datapoints
        q_t^n(x) = p_t(h^n|v^n) δ(v, v^n)                       ▷ E step
    end for
    for i = 1 to K do                                          ▷ Run over all variables
        p_{t+1}(x_i|pa(x_i)) = (1/N) Σ_{n=1}^N q_t^n(x_i|pa(x_i))  ▷ M step
    end for
end while
return p_t(x_i|pa(x_i))                   ▷ The max likelihood parameter estimate.
```

The energy term in equation (11.2.5) can be written more compactly as

$$\sum_n \langle \log p(x^n) \rangle_{q_t(h|v^n)} = N \langle \log p(x) \rangle_{q_t(x)} \tag{11.2.34}$$

To see this consider the right hand side of the above

$$N \langle \log p(x) \rangle_{q_t(x)} = N \sum_x [\log p(x)] \frac{1}{N} \sum_n q_t(h|v^n)\delta(v, v^n) = \sum_n \langle \log p(x^n) \rangle_{q_t(h|v^n)} \tag{11.2.35}$$

Using the structure of the Belief Network, we have

$$\langle \log p(x) \rangle_{q_t(x)} = \sum_i \langle \log p(x_i|\text{pa}(x_i)) \rangle_{q_t(x)} = \sum_i \left\langle \langle \log p(x_i|\text{pa}(x_i)) \rangle_{q_t(x_i|\text{pa}(x_i))} \right\rangle_{q_t(\text{pa}(x_i))} \tag{11.2.36}$$

This means that maximising the energy is equivalent to minimising

$$\sum_i \left\langle \langle \log q_t(x_i|\text{pa}(x_i)) \rangle_{q_t(x_i|\text{pa}(x_i))} - \langle \log p(x_i|\text{pa}(x_i)) \rangle_{q_t(x_i|\text{pa}(x_i))} \right\rangle_{q_t(\text{pa}(x_i))} \tag{11.2.37}$$

where we added the constant first term to make this into the form of a Kullback-Leibler divergence. Since this is a sum of independent Kullback-Leibler divergences, optimally the M-step is given by setting

$$p(x_i|\text{pa}(x_i)) = q_t(x_i|\text{pa}(x_i)) \tag{11.2.38}$$

In practice, storing the $q_t(x)$ over the states of all variables x is prohibitively expense. Fortunately, since the M-step only requires the distribution on the family of each variable x_i, one only requires the local distributions $q^n_{old}(x_i|\text{pa}(x_i))$. We may therefore dispense with the global $q_{old(x)}$ and equivalently use

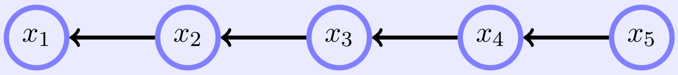
$$p^{new}(x_i|\text{pa}(x_i)) = \frac{\sum_n q^n_{old}(x_i, \text{pa}(x_i))}{\sum_{n'} q^{n'}_{old}(\text{pa}(x_i))} \tag{11.2.39}$$

Using the EM algorithm, the optimal setting for the E-step is to use $q_t(h^n|v^n) = p^{old}(h^n|v^n)$. With this notation, the EM algorithm can be compactly stated as in algorithm(9). See also `EMbeliefnet.m`.

Example 54 (More General Belief Networks). Consider a five variable distribution with discrete variables,

$$p(x_1, x_2, x_3, x_4, x_5) = p(x_1|x_2)p(x_2|x_3)p(x_3|x_4)p(x_4|x_5)p(x_5) \tag{11.2.40}$$

in which the variables x_2 and x_4 are consistently hidden in the training data, and training data for x_1, x_3, x_5 are always present. The distribution can be represented as a Belief network



In this case, the contributions to the energy have the form

$$\sum_n \langle \log p(x_1^n|x_2)p(x_2|x_3^n)p(x_3^n|x_4)p(x_4|x_5^n)p(x_5^n)\rangle_{q^n(x_2,x_4|x_1,x_3,x_5)} \tag{11.2.41}$$

which may be written as

$$\sum_n \langle \log p(x_1^n|x_2)\rangle_{q^n(x_2,x_4|x_1,x_3,x_5)} + \sum_n \langle \log p(x_2|x_3^n)\rangle_{q^n(x_2,x_4|x_1,x_3,x_5)} + \sum_n \langle \log p(x_3^n|x_4)\rangle_{q^n(x_2,x_4|x_1,x_3,x_5)} + \sum_n \langle \log p(x_4|x_5^n)\rangle_{q^n(x_2,x_4|x_1,x_3,x_5)} + \sum_n \log p(x_5^n) \tag{11.2.42}$$

A useful property can now be exploited, namely that each term depends on only those hidden variables in the family that that term represents. Thus we may write

$$\sum_n \langle \log p(x_1^n|x_2)\rangle_{q^n(x_2|x_1,x_3,x_5)} + \sum_n \langle \log p(x_2|x_3^n)\rangle_{q^n(x_2|x_1,x_3,x_5)} + \sum_n \langle \log p(x_3^n|x_4)\rangle_{q^n(x_4|x_1,x_3,x_5)} + \sum_n \langle \log p(x_4|x_5^n)\rangle_{q^n(x_4|x_1,x_3,x_5)} + \sum_n \log p(x_5^n)$$

The final term can be set using Maximum Likelihood. Let us consider therefore a more difficult table, $p(x_1|x_2)$. When will the table entry $p(x_1 = i|x_2 = j)$ occur in the energy? This happens whenever x_1^n is in state i. Since there is a summation over all the states of variables x_2 (due to the average), there is also a term with variable x_2 in state j. Hence the contribution to the energy from terms of the form $p(x_1 = i|x_2 = j)$ is

$$\sum_n \mathbb{I}[x_1^n = i]\, q^n(x_2 = j|x_1, x_3, x_5) \log p(x_1 = i|x_2 = j) \tag{11.2.43}$$

where the indicator function $\mathbb{I}[x_1^n = i]$ equals 1 if x_1^n is in state i and is zero otherwise. To ensure normalisation of the table, we add a Lagrange term:

$$\sum_n \mathbb{I}[x_1^n = i]\, q^n(x_2 = j|x_1, x_3, x_5) \log p(x_1 = i|x_2 = j) + \lambda \left\{1 - \sum_k p(x_1 = k|x_2 = j)\right\} \tag{11.2.44}$$

Differentiating with respect to $p(x_1 = i|x_2 = j)$ we get

$$\sum_n \mathbb{I}[x_1^n = i] \frac{q^n(x_2 = j|x_1, x_3, x_5)}{p(x_1 = i|x_2 = j)} = \lambda \tag{11.2.45}$$

or

$$p(x_1 = i|x_2 = j) \propto \sum_n \mathbb{I}[x_1^n = i]\, q^n(x_2 = j|x_1, x_3, x_5). \tag{11.2.46}$$

Hence

$$p(x_1 = i|x_2 = j) = \frac{\sum_n \mathbb{I}[x_1^n = i]\, q^n(x_2 = j|x_1, x_3, x_5)}{\sum_{n,k} \mathbb{I}[x_1^n = k]\, q^n(x_2 = j|x_1, x_3, x_5)} \tag{11.2.47}$$

Using the EM algorithm, we have

$$q^n(x_2 = j|x_1, x_3, x_5) = p(x_2 = j|x_1^n, x_3^n, x_5^n) \tag{11.2.48}$$

This optimal distribution is easy to compute since this is the marginal on the family, given some evidential variables. Hence, the M-step update for the table is

$$p^{new}(x_1 = i|x_2 = j) = \frac{\sum_n \mathbb{I}[x_1^n = i]\, p^{old}(x_2 = j|x_1^n, x_3^n, x_5^n)}{\sum_{n,k} \mathbb{I}[x_1^n = k]\, p^{old}(x_2 = j|x_1^n, x_3^n, x_5^n)} \tag{11.2.49}$$

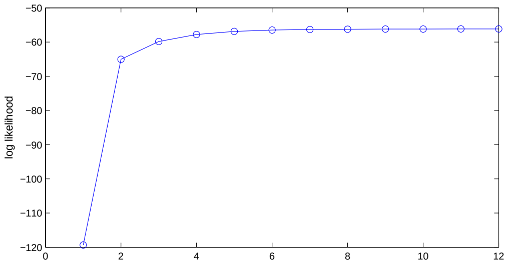


Figure 11.4: Evolution of the log-likelihood versus iterations under the EM training procedure (from solving the Printer Nightmare with missing data, exercise(138). Note how rapid progress is made at the beginning, but convergence can be slow.

What about the table $p(x_2 = i|x_3 = j)$? To ensure normalisation of the table, we add a Lagrange term:

$$\sum_n \mathbb{I}[x_3^n = j]\, q^n(x_2 = i|x_1, x_3, x_5) \log p(x_2 = i|x_3 = j) + \lambda \left\{ 1 - \sum_k p(x_2 = k|x_3 = j) \right\} \tag{11.2.50}$$

As before, differentiating, and using the EM settings, we have

$$p^{new}(x_2 = i|x_3 = j) = \frac{\sum_n \mathbb{I}[x_3^n = j]\, p^{old}(x_2 = i|x_1^n, x_3^n, x_5^n)}{\sum_{n,k} \mathbb{I}[x_3^n = j]\, p^{old}(x_2 = k|x_1^n, x_3^n, x_5^n)} \tag{11.2.51}$$

There is a simple intuitive pattern to equation (11.9.2) and equation (11.2.51) : If there were no hidden data, equation (11.9.2) would read

$$p^{new}(x_1 = i|x_2 = j) \propto \sum_n \mathbb{I}[x_1^n = i]\, \mathbb{I}[x_2^n = j] \tag{11.2.52}$$

and equation (11.2.51) would be

$$p^{new}(x_2 = i|x_3 = j) \propto \sum_n \mathbb{I}[x_3^n = j]\, \mathbb{I}[x_2^n = i] \tag{11.2.53}$$

All that we do, therefore, in the general EM case, is to replace those deterministic functions such as $\mathbb{I}[x_2^n = i]$ by their missing variable equivalents $p^{old}(x_2 = i|x_1^n, x_3^n, x_5^n)$. This is merely a restatement of the general update given in equation (11.2.39) under the definition (11.2.32).

11.2.4 Application to Markov networks

For a MN defined over visible and hidden variables $p(v, h|\theta) = \frac{1}{Z(\theta)} \prod_c \phi_c(h, v)$ the EM variational bound is

$$\log p(v|\theta) \geq -H(q(h)) + \sum_c \langle \log \phi_c(h, v|\theta) \rangle_{q(h)} - \log Z(\theta) \tag{11.2.54}$$

Whilst the bound decouples the parameters in the second term, the parameters are nevertheless coupled in the normalisation $Z(\theta)$. Because of this we cannot optimise the above bound on a parameter by parameter basis. One approach is to use an additional bound $\log Z(\theta)$ from above, as for iterative scaling.

11.2.5 Convergence

Convergence of EM can be slow, particularly when the number of missing observations is greater than the number of visible observations. In practice, one often combines the EM with gradient based procedures to improve convergence, see section(11.7). Note also that the log likelihood is typically a non-convex function of the parameters. This means that there may be multiple local optima and the solution found often depends on the initialisation.

11.3 Extensions of EM

11.3.1 Partial M step

It is not necessary to find the full optimum of the energy term at each iteration. As long as one finds a parameter θ' which has a higher energy than that of the current parameter θ, then the conditions required in section(11.2.2) still hold, and the likelihood cannot decrease at each iteration.

11.3.2 Partial E step

The E-step requires us to find the optimum of

$$\log p(\mathcal{V}|\theta) \geq -\sum_{n=1}^{N} \langle \log q(h^n|v^n) \rangle_{q(h^n|v^n)} + \sum_{n=1}^{N} \langle \log p(h^n, v^n|\theta) \rangle_{q(h^n|v^n)} \tag{11.3.1}$$

with respect to $q(h^n|v^n)$. The fully optimal setting is

$$q(h^n|v^n) = p(h^n|v^n) \tag{11.3.2}$$

For a guaranteed increase in likelihood at each iteration, from section(11.2.2) we required that the fully optimal setting of q is used. Unfortunately, therefore, one cannot in general guarantee that such a partial E step would always increase the likelihood. Of course, it *is* guaranteed to increase the lower bound on the likelihood, though not the likelihood itself.

Intractable Energy

The EM algorithm assumes that we can calculate

$$\langle \log p(h, v|\theta) \rangle_{q(h|v)} \tag{11.3.3}$$

However, in general, it may be that we can only carry out the average over q for a very restricted class of distributions - for example, factorised distributions $q(h|v) = \prod_j q(h_j|v)$. In such cases one may use a simpler class of distributions, $\mathcal{Q}$, *e.g.* $\mathcal{Q}$ = factorised $q(h|v) = \prod_i q(h_i|v)$, for which the averaging required for the energy may be simpler.

We can find the best distribution in class $\mathcal{Q}$ by minimising the KL divergence between $q(h|v, \theta_Q)$ and $p(h|v, \theta)$ numerically using a non-linear optimisation routine:

$$q^{opt} = \underset{q \in \mathcal{Q}}{\operatorname{argmin}} \ \mathrm{KL}(q(h)|p(h|v, \theta)) \tag{11.3.4}$$

Alternatively, one can assume a certain structured form for the q distribution, and learn the optimal factors of the distribution by free form functional calculus.

Viterbi Training

An extreme case is to restrict $q(h^n|v^n)$ to a delta-function. In this case, the entropic term $\langle \log q(h^n|v^n) \rangle_{q(h^n|v^n)}$ is constant, so that the optimal delta function q is to set

$$q(h^n|v^n) = \delta\left(h^n, h^n_*\right) \tag{11.3.5}$$

where

$$h^n_* = \underset{h}{\operatorname{argmax}} \ p(h, v^n|\theta) \tag{11.3.6}$$

This is called Viterbi training and is common in training HMMs, see section(23.2). EM training with this restricted class of q distribution is therefore guaranteed to increase the lower bound on the log likelihood, though not the likelihood itself. A practical advantage of Viterbi training is that the energy is always tractable to compute, becoming simply

$$\sum_{n=1}^{N} \log p(h_*^n, v^n|\theta) \tag{11.3.7}$$

which is amenable to optimisation.

Provided there is sufficient data, one might hope that the likelihood as a function of the parameter θ will be sharply peaked around the optimum value. This means that at convergence the approximation of the posterior $p(h|v, \theta^{opt})$ by a delta function will be reasonable, and an update of EM using Viterbi training will produce a new θ approximately the same as θ^{opt}. For any highly suboptimal θ, however, $p(h|v, \theta)$ will be far from a delta function, and therefore a Viterbi update is less reliable in terms of leading to an increase in the likelihood itself. This suggests that the initialisation of θ for Viterbi training is more critical than for the standard EM.

Stochastic Training

Another approximate $q(h^n|v^n)$ distribution would be to use an empirical distribution formed by samples from the fully optimal distribution $p(h^n|v^n, \theta)$. That is one draws samples (see chapter(27) for a discussion on sampling) $h_1^n, \ldots, h_L^n$ from $p(h^n|v^n, \theta)$ and forms a q distribution

$$q(h^n|v^n) = \frac{1}{L}\sum_{l=1}^{L} \delta\left(h^n, h_l^n\right) \tag{11.3.8}$$

The energy becomes then proportional to

$$\sum_{n=1}^{N}\sum_{l=1}^{L} \log p(h_l^n, v^n|\theta) \tag{11.3.9}$$

so that, as in Viterbi training, the energy is always computationally tractable for this restricted q class. Provided that the samples from $p(h^n|v^n)$ are reliable, stochastic training will produce an energy function with (on average) the same characteristics as the true energy under the classical EM algorithm. This means that the solution obtained from stochastic training should tend to that from the classical EM as the number of samples increases.

11.4 A failure case for EM

Consider a likelihood of the form

$$p(v|\theta) = \int_h \delta\left(v, f(h|\theta)\right) p(h) \tag{11.4.1}$$

If we attempt an EM approach for this, this will fail (see also exercise(76)). For a more general model of the form

$$p(v|\theta) = \int_h p\left(v|h, \theta\right) p(h) \tag{11.4.2}$$

The E-step is

$$q(h|\theta_{old}) \propto p\left(v|h, \theta_{old}\right) p(h) \tag{11.4.3}$$

and the M-step sets

$$\theta_{new} = \underset{\theta}{\operatorname{argmax}} \left\langle \log p(v, h|\theta)\right\rangle_{p(h|\theta_{old})} = \underset{\theta}{\operatorname{argmax}} \left\langle \log p(v|h, \theta)\right\rangle_{p(h|\theta_{old})} \tag{11.4.4}$$

where we used the fact that for this model $p(h)$ is independent of θ. In the case that $p(v|h, \theta) = \delta(v, f(h|\theta))$ then

$$p(h|\theta_{old}) \propto \delta(v, f(h|\theta))\, p(h) \tag{11.4.5}$$

so that optimising the energy requires

$$\theta_{new} = \underset{\theta}{\operatorname{argmax}} \ \langle \log \delta(v, f(h|\theta)) \rangle_{p(h|\theta_{old})} \tag{11.4.6}$$

Since $p(h|\theta_{old})$ is zero everywhere expect that h for which $v = f(h|\theta)$, then the energy is effectively negative infinity if $\theta \neq \theta_{old}$. However, when $\theta = \theta_{old}$ the energy is maximal[2] . This is therefore the optimum of the energy, and represents therefore a failure in updating for EM. This situation occurs in practice, and has been noted in particular in the context of Independent Component Analysis[219].

One can attempt to heal this behaviour by deriving an EM algorithm based on the distribution

$$p_\epsilon(v|h, \theta) = (1-\epsilon)\delta(v, f(h|\theta)) + \epsilon n(h), \quad 0 \leq \epsilon \leq 1 \tag{11.4.7}$$

where $n(h)$ is an arbitrary distribution on the hidden variable h. The original deterministic model corresponds to $p_0(v|h, \theta)$. Defining

$$p_\epsilon(v|\theta) = \int_h p_\epsilon(v|h, \theta)p(h) \tag{11.4.8}$$

we have

$$p_\epsilon(v|\theta) = (1-\epsilon)p_0(v|\theta) + \epsilon \langle n(h) \rangle_{p(h)} \tag{11.4.9}$$

An EM algorithm for $p_\epsilon(v|\theta)$, $0 < \epsilon < 1$ satisfies

$$p_\epsilon(v|\theta_{new}) - p_\epsilon(v|\theta_{old}) = (1-\epsilon)\left(p_0(v|\theta_{new}) - p_0(v|\theta_{old})\right) > 0 \tag{11.4.10}$$

which implies

$$p_0(v|\theta_{new}) - p_0(v|\theta_{old}) > 0 \tag{11.4.11}$$

This means that the EM algorithm for the non-deterministic case $0 < \epsilon < 1$ is guaranteed to increase the likelihood under the deterministic model $p_0(v|\theta)$ at each iteration (unless we are at convergence). See [97] for an application of this 'antifreeze' technique to learning Markov Decision Processes with EM.

11.5 Variational Bayes

As discussed in section(9.2) Maximum Likelihood corresponds to a form of Bayesian approach in which the parameter posterior distribution (under a flat prior) is approximated with a delta function $p(\theta|\mathcal{V}) \approx \delta(\theta, \theta_{opt})$. Variational Bayes is analogous to EM in that it attempts to deal with hidden variables but using a distribution that better represents the posterior distribution than given by Maximum Likelihood.

To keep the notation simple, we'll initially assume only a single datapoint with observation v. Our interest is then the parameter posterior

$$p(\theta|v) \propto p(v|\theta)p(\theta) \propto \sum_h p(v, h|\theta)p(\theta) \tag{11.5.1}$$

The VB approach assumes a factorised approximation of the joint hidden and parameter posterior:

$$p(h, \theta|v) \approx q(h)q(\theta) \tag{11.5.2}$$

[2]For discrete variables and the Kronecker delta, the energy attains the maximal value of zero when $\theta = \theta_{old}$. In the case of continuous variables, however, the log of the Dirac delta function is not well defined. Considering the delta function as the limit of a narrow width Gaussian, for any small but finite width, the energy is largest when $\theta = \theta_{old}$.

A bound on the marginal likelihood

By minimising the KL divergence,

$$\mathrm{KL}(q(h)q(\theta)|p(h,\theta|v)) = \langle \log q(h)\rangle_{q(h)} + \langle \log q(\theta)\rangle_{q(\theta)} - \langle \log p(h,\theta|v)\rangle_{q(h)q(\theta)} \geq 0 \tag{11.5.3}$$

we arrive at the bound

$$\log p(v) \geq -\langle \log q(h)\rangle_{q(h)} - \langle \log q(\theta)\rangle_{q(\theta)} + \langle \log p(v,h,\theta)\rangle_{q(h)q(\theta)} \tag{11.5.4}$$

For fixed $q(\theta)$ if we minimize the Kullback-Leibler divergence, we get the tightest lower bound on $\log p(v)$. If then for fixed $q(h)$ we minimise the Kullback-Leibler divergence *w.r.t.* $q(\theta)$ we are maximising the term $-\langle \log q(\theta)\rangle_{q(\theta)} + \langle \log p(v,h,\theta)\rangle_{q(h)q(\theta)}$ and hence pushing up the bound on the marginal likelihood. This simple co-ordinate wise procedure in which we first fix the $q(\theta)$ and solve for $q(h)$ and then vice versa is analogous to the E and M step of the EM algorithm:

E-step

$$q^{new}(h) = \underset{q(h)}{\operatorname{argmin}}\ \mathrm{KL}\Big(q(h)q^{old}(\theta)|p(h,\theta|v)\Big) \tag{11.5.5}$$

M-step

$$q^{new}(\theta) = \underset{q(\theta)}{\operatorname{argmin}}\ \mathrm{KL}(q^{new}(h)q(\theta)|p(h,\theta|v)) \tag{11.5.6}$$

In full generality for a set of observations $\mathcal{V}$ and hidden variables $\mathcal{H}$, algorithm(7). For distributions $q(\mathcal{H})$ and $q(\theta)$ which are parametersised/constrained, the best distributions in the minimal KL sense are returned. In general, each iteration of VB is guaranteed to increase the bound on the marginal likelihood, but not the marginal likelihood itself. Like the EM algorithm, VB can (and often does) suffer from local maxima issues. This means that the converged solution can be dependent on the initialisation.

Unconstrained approximations

For fixed $q(\theta)$ the contribution to the KL divergence is

$$\langle \log q(h)\rangle_{q(h)} - \langle \log p(v,h,\theta)\rangle_{q(h)q(\theta)} = \mathrm{KL}(q(h)|\tilde{p}(h)) + \mathsf{const.} \tag{11.5.7}$$

where

$$\tilde{p}(h) \equiv \frac{1}{\tilde{Z}} e^{\langle \log p(v,h,\theta)\rangle_{q(\theta)}} \tag{11.5.8}$$

where $\tilde{Z}$ is a normalising constant. Hence, for fixed $q(\theta)$, the optimal $q(h)$ is given by $\tilde{p}$,

$$q(h) \propto e^{\langle \log p(v,h,\theta)\rangle_{q(\theta)}} \tag{11.5.9}$$

Similarly, for fixed $q(h)$, optimally

$$q(\theta) \propto e^{\langle \log p(v,h,\theta)\rangle_{q(h)}} \tag{11.5.10}$$

i.i.d. Data

Under the i.i.d. assumption, we obtain a bound on the marginal likelihood for the whole dataset:

$$\log p(\mathcal{V}|\theta) \geq \sum_n \Big\{ -\langle \log q(h^n)\rangle_{q(h^n)} - \langle \log q(\theta)\rangle_{q(\theta)} + \langle \log p(v^n,h^n,\theta)\rangle_{q(h^n)q(\theta)} \Big\} \tag{11.5.11}$$

The bound holds for any $q(h^n)$ and $q(\theta)$ but is tightest for the converged estimates from the VB procedure.

For an i.i.d. dataset, it is straightforward to show that without loss of generality we may assume

$$q(h^1,\ldots,h^N) = \prod_n q(h^n) \tag{11.5.12}$$

Under this we arrive at algorithm(11).

Algorithm 10 Variational Bayes.

$t = 0$ ▷ Iteration counter
Choose an initial distribution $q_0(\theta)$. ▷ Initialisation
while θ not converged (or likelihood bound not converged) **do**
  $t \leftarrow t + 1$
  $q_t^n(\mathcal{H}) = \arg\min_{q(\mathcal{H})} \mathrm{KL}(q(\mathcal{H})q_{t-1}(\theta)|p(\mathcal{H},\theta|\mathcal{V}))$ ▷ E step
  $q_t^n(\theta) = \arg\min_{q(\theta)} \mathrm{KL}(q_t^n(\mathcal{H})q(\theta)|p(\mathcal{H},\theta|\mathcal{V}))$ ▷ M step
end while
return $q_t^n(\theta)$ ▷ The posterior parameter approximation.

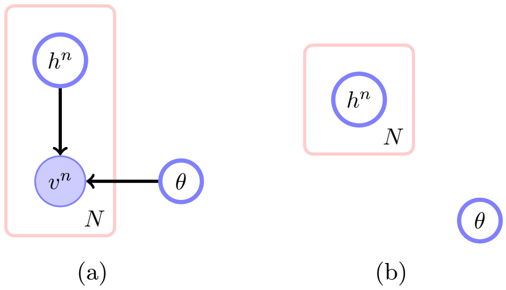


Figure 11.5: **(a)**: Generic form of a model with hidden variables. **(b)**: A factorised posterior approximation uses in Variational Bayes.

11.5.1 EM is a special case of Variational Bayes

If we wish to find a summary of the parameter distribution corresponding to only the most likely point θ, then

$$q(\theta) = \delta(\theta, \theta_*) \tag{11.5.13}$$

where θ_* is the single optimal value of the parameter. If we plug this assumption into equation (11.5.4) we obtain the bound

$$\log p(v|\theta_*) \geq -\langle \log q(h) \rangle_{q(h)} + \langle \log p(v, h, \theta_*) \rangle_{q(h)} + \text{const.} \tag{11.5.14}$$

The M-step is then given by

$$\theta_* = \underset{\theta}{\operatorname{argmax}} \left(\langle \log p(v|h,\theta)p(h|\theta) \rangle_{q(h)} + \log p(\theta) \right) \tag{11.5.15}$$

For a flat prior $p(\theta) = \text{const.}$, this is therefore equivalent to energy maximisation in the EM algorithm. Using this single optimal value in the VB update for $q(h^n)$ we have

$$q_t^n(h) \propto p(v, h|\theta_*) \propto p(h|v, \theta_*) \tag{11.5.16}$$

which is the standard E-step of EM. Hence EM is a special case of VB, under a flat prior $p(\theta) = \text{const.}$ and a delta function approximation of the parameter posterior.

11.5.2 Factorising the parameter posterior

Let's reconsider Bayesian learning in the binary variable network

$$p(a, c, s) = p(c|a, s)p(a)p(s) \tag{11.5.17}$$

in which we use a factorised parameter prior

$$p(\theta_c)(\theta_a)p(\theta_s) \tag{11.5.18}$$

When all the data is observed, the parameter posterior factorises. As we discussed in section(11.1.1) if the state of a is not observed, the parameter posterior no longer factorises:

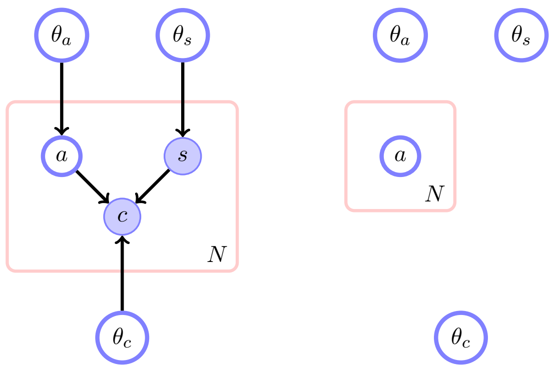
$$p(\theta_a, \theta_s, \theta_c|\mathcal{V}) \propto p(\theta_a)p(\theta_s)p(\theta_c)p(\mathcal{V}|\theta_a, \theta_s, \theta_c) \tag{11.5.19}$$

$$\propto p(\theta_a)p(\theta_s)p(\theta_c) \prod_n p(v^n|\theta_a, \theta_s, \theta_c) \tag{11.5.20}$$

$$\propto p(\theta_a)p(\theta_s)p(\theta_c) \prod_n p(s^n|\theta_s) \sum_{a^n} p(c^n|s^n, a^n, \theta_c)p(a^n|\theta_a) \tag{11.5.21}$$

Algorithm 11 Variational Bayes (i.i.d. data).

$t = 0$ ▷ Iteration counter
Choose an initial distribution $q_0(\theta)$. ▷ Initialisation
while θ not converged (or likelihood bound not converged) **do**
    $t \leftarrow t + 1$
    for $n = 1$ to N **do** ▷ Run over all datapoints
        $q_t^n(h^n) \propto e^{\langle \log p(v^n, h^n, \theta) \rangle_{q_{t-1}(\theta)}}$ ▷ E step
    end for
    $q_t(\theta) \propto p(\theta) e^{\sum_n \langle \log p(v^n, h^n|\theta) \rangle_{q_t^n(h^n)}}$ ▷ M step
end while
return $q_t^n(\theta)$ ▷ The posterior parameter approximation.



(a) (b)

Figure 11.6: **(a)**: A model for the relationship between lung Cancer, Asbestos exposure and Smoking with factorised parameter priors. Variables c and s are observed, but variable a is consistently missing. **(b)**: A factorised parameter posterior approximation.

where the summation over a prevents the factorisation into a product of the individual table parameters.

Since it is convenient in terms of representations to work with factorised posteriors, we can apply VB but with a factorised constraint on the form of the q. In VB we define a distribution over the visible and hidden variables. In this case the hidden variables are the a^n and the visible are s^n, c^n. The joint posterior over all unobserved variables (parameters and missing observations) is

$$p(\theta_a, \theta_s, \theta_c, a^1, \ldots, a^N|\mathcal{V}) \propto p(\theta_a)p(\theta_s)p(\theta_c) \prod_n p(c^n|s^n, a^n, \theta_c)p(s^n|\theta_s)p(a^n|\theta_a) \tag{11.5.22}$$

To make a factorised posterior approximation we use

$$p(\theta_a, \theta_s, \theta_c, a^1, \ldots, a^N|\mathcal{V}) \approx q(\theta_a)q(\theta_c)q(\theta_s) \prod_n q(a^n) \tag{11.5.23}$$

and minimise the Kullback-Leibler divergence between the left and right of the above.

M-step

Hence

$$q(\theta_a) \propto p(\theta_a) \prod_n e^{\langle \log p(a^n|\theta_a) \rangle_{q(a^n)}} \tag{11.5.24}$$

$$\langle \log p(a^n|\theta_a) \rangle_{q(a^n)} = q(a^n = 1) \log \theta_a + q(a^n = 0) \log (1 - \theta_a) \tag{11.5.25}$$

Hence

$$e^{\langle \log p(a^n|\theta_a) \rangle_{q(a^n)}} = \theta_a^{q(a^n=1)} (1 - \theta_a)^{q(a^n=0)} \tag{11.5.26}$$

It is convenient to use a Beta distribution prior,

$$p(\theta_a) \propto \theta_a^{\alpha-1} (1 - \theta_a)^{\beta-1} \tag{11.5.27}$$

since the posterior approximation is then also a Beta distribution:

$$q(\theta_a) = B\left(\theta_a|\alpha + \sum_n q(a^n = 1), \beta + \sum_n q(a^n = 0\right) \tag{11.5.28}$$

A similar calculation gives

$$q(\theta_s) = B\left(\theta_s|\alpha + \sum_n \mathbb{I}\left[s^n = 1\right], \beta + \sum_n \mathbb{I}\left[s^n = 0\right]\right) \tag{11.5.29}$$

and four tables, one for each of the parental states of c. For example

$$q(\theta_c(a = 0, s = 1)) = B\left(\theta_c|\alpha + \sum_n \mathbb{I}\left[s^n = 1\right] q(a^n = 0), \beta + \sum_n \mathbb{I}\left[s^n = 0\right] q(a^n = 1)\right) \tag{11.5.30}$$

These are reminiscent of the standard Bayesian equations, equation (9.3.17) except that the counts have been replaced by q's.

E-step

We still need to determine $q(a^n)$. The optimal value is given by minimising the Kullback-Leibler divergence with respect to $q(a^n)$. This gives the solution that optimally,

$$q(a^n) \propto e^{\langle \log p(c^n|s^n,a^n,\theta_c)\rangle_{q(\theta_c)} + \langle \log p(a^n|\theta_a)\rangle_{q(\theta_a)}} \tag{11.5.31}$$

For example, if assume that for datapoint n, s is in state 1 and c in state 0, then

$$q(a^n = 1) \propto e^{\langle \log(1-\theta_c(s=1,a=1))\rangle_{q(\theta_c(s=1,a=1))} + \langle \log \theta_a\rangle_{q(\theta_a)}} \tag{11.5.32}$$

and

$$q(a^n = 0) \propto e^{\langle \log(1-\theta_c(s=1,a=0))\rangle_{q(\theta_c(s=1,a=1))} + \langle \log(1-\theta_a)\rangle_{q(\theta_a)}} \tag{11.5.33}$$

To compute such quantities explicitly, we need the values $\langle \log \theta\rangle_{B(\theta|\alpha,\beta)}$ and $\langle \log(1-\theta)\rangle_{B(\theta|\alpha,\beta)}$. For a Beta distribution, these are straightforward to compute, see exercise(96).

The complete VB procedure is then given by iterating equations (11.5.28,11.5.29,11.5.30) and (11.5.32,11.5.33) until convergence.

Given a converged factorised approximation, computing a marginal table $p(a = 1|\mathcal{V})$ is then straightforward under the approximation

$$p(a = 1|\mathcal{V}) \approx \int_{\theta_a} q(a = 1|\theta_a) q(\theta_a|\mathcal{V}) = \int_{\theta_a} \theta_a q(\theta_a|\mathcal{V}) \tag{11.5.34}$$

Since $q(\theta_a|\mathcal{V})$ is a Beta distribution $B(\theta_a|\alpha,\beta)$, the mean is straightforward. Using this for both states of a leads to

$$p(a = 1|\mathcal{V}) = \frac{\alpha + \sum_n q(a^n = 1)}{\alpha + \sum_n q(a^n = 0) + \beta + \sum_n q(a^n = 1)} \tag{11.5.35}$$

The application of VB to learning the tables in arbitrarily structured BNs is a straightforward extension of the technique outlined here. Under the factorised approximation, $q(h, \theta) = q(h)q(\theta)$, one will always obtain a simple updating equation analogous to the full data case, but with the missing data replaced by variational approximations. Nevertheless, if a variable has many missing parents, the number of states in the average with respect to the q distribution can become intractable, and further constraints on the form of the approximation, or additional bounds are required.

One may readily extend the above to the case of Dirichlet distributions on multinomial variables, see exercise(142). Indeed, the extension to the exponential family is straightforward.

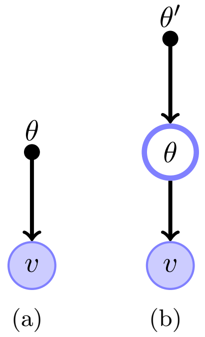


Figure 11.7: **(a)**: Standard ML learning. The best parameter θ is found by maximising the probability that the model generates the observed data $\theta_{opt} = \arg\max_\theta p(v|\theta)$. **(b)**: ML-II learning. In cases where we have a prior preference for the parameters θ, but with unspecified hyperparameter θ', we can find θ' by $\theta'_{opt} = \arg\max_{\theta'} p(v|\theta') = \arg\max_{\theta'} \langle p(v|\theta)\rangle_{p(\theta|\theta')}$.

11.6 Bayesian Methods and ML-II

Consider a parameterised distribution $p(v|\theta)$, for which we wish to the learn the optimal parameters θ given some data. The model $p(v|\theta)$ is depicted in fig(11.7a), where a dot indicates that no distribution is present on that variable. For a single observed datapoint v, setting θ by Maximum Likelihood corresponds to finding the parameter θ that maximises $p(v|\theta)$.

In some cases we may have an idea about which parameters θ are more appropriate and can express this prior preference using a distribution $p(\theta)$. If the prior were fully specified, then there is nothing to 'learn' since $p(\theta|v)$ is now fully known. However, in many cases in practice, we are unsure of the exact parameter settings of the prior, and hence specify a parametersised prior using a distribution $p(\theta|\theta')$ with hyperparameter θ'. This is depicted in fig(11.7b). The learning corresponds to finding the optimal θ' that maximises the likelihood $p(v|\theta') = \int_\theta p(v|\theta)p(\theta|\theta')$. This is known as an ML-II procedure since it corresponds to maximum likelihood, but at the higher, hyperparameter level[33, 180]. This is a form of approximate Bayesian analysis since, although θ' is set using maximum likelihood, after training, we have a distribution over parameters, $p(\theta|v, \theta')$.

11.7 Optimising the Likelihood by Gradient methods

11.7.1 Directed models

The EM algorithm typically works well when the amount of missing information is small compared to the complete information. In this case EM exhibits approximately the same convergence as Newton based gradient method[233]. However, if the fraction of missing information approaches unity, EM can converge very slowly. In the case of continuous parameters θ, an alternative is to compute the gradient of the likelihood directly and use this as part of a standard continuous variable optimisation routine. The gradient is straightforward to compute using the following identity. Consider the log likelihood

$$L(\theta) = \log p(v|\theta) \tag{11.7.1}$$

The derivative can be written

$$\partial_\theta L(\theta) = \frac{1}{p(v|\theta)}\partial_\theta p(v|\theta) = \frac{1}{p(v|\theta)}\partial_\theta \int_h p(v, h|\theta) \tag{11.7.2}$$

At this point, we take the derivative inside the integral

$$\partial_\theta L(\theta) = \frac{1}{p(v|\theta)}\int_h \partial_\theta p(v, h|\theta) = \int_h p(h|v, \theta)\partial_\theta \log p(v, h|\theta) = \langle \partial_\theta \log p(v, h|\theta)\rangle_{p(h|v,\theta)} \tag{11.7.3}$$

where we used $\partial \log f(x) = (1/f(x))\partial f(x)$. The right hand side is the average of the derivative of the log complete likelihood. This is closely related to the derivative of the energy term in the EM algorithm, though note that the average here is performed with respect the current distribution parameters θ and not θ^{old} as in the EM case. Used in this way, computing the derivatives of latent variable models is relatively straightforward. These derivatives may then be used as part of a standard optimisation routine such as conjugate gradients[233].

11.7.2 Undirected Models

Consider an undirected model which contains both hidden and visible variables

$$p(v, h|\theta) = \frac{1}{Z(\theta)} e^{\phi(v,h)} \tag{11.7.4}$$

For i.i.d. data, the log likelihood on the visible variables is (assuming discrete v and h)

$$L(\theta) = \sum_n \left(\log \sum_h e^{\phi(v^n,h|\theta)} - \log \sum_{h,v} e^{\phi(v,h|\theta)} \right) \tag{11.7.5}$$

which has gradient

$$\frac{\partial}{\partial \theta} L = \sum_n \left(\underbrace{\left\langle \frac{\partial}{\partial \theta} \phi(v^n, h|\theta) \right\rangle_{p(h|v^n)}}_{\text{clamped average}} - \underbrace{\left\langle \frac{\partial}{\partial \theta} \phi(v, h|\theta) \right\rangle_{p(h,v)}}_{\text{free average}} \right) \tag{11.7.6}$$

For a Markov Network that is intractable (the partition function Z cannot be computed efficiently), the gradient is particularly difficult to estimate since it is the difference of two quantities, each of which needs to be estimated. Even getting the sign of the gradient correct can therefore be computationally difficult. For this reason learning in models, such as the Boltzmann machine with hidden units, is particularly difficult.

11.8 Code

In the demo code we take the original Chest Clinic network [167] and draw data samples from this network. Our interest is then to see if we can use the EM algorithm to estimate the tables based on the data (with some parts of the data missing at random). We assume that we know the correct BN structure, only that the CPTs are unknown. We assume the logic gate table is known, so we do not need to learn this.
`demoEMchestclinic.m`: Demo of EM in learning the Chest Clinic Tables

The following code implements Maximum Likelihood learning of BN tables based on data with possibly missing values.
`EMbeliefnet.m`: EM training of a Belief Network

11.9 Exercises

Exercise 138 (Printer Nightmare continued)**.** *Continuing with the BN given in fig(9.19), the following table represents data gathered on the printer, where ? indicates that the entry is missing. Each column represents a datapoint. Use the EM algorithm to learn all CPTs of the network.*

fuse assembly malfunction	?	?	?	1	0	0	?	0	?	0	0	?	1	?	1
drum unit	?	0	?	0	1	0	0	1	?	?	1	1	?	0	0
toner out	1	1	0	?	?	1	0	1	0	?	0	1	?	0	?
poor paper quality	1	0	1	0	1	?	1	0	1	1	?	1	1	?	0
worn roller	0	0	?	?	?	0	1	?	0	0	?	0	?	1	1
burning smell	0	?	?	1	0	0	0	0	0	?	0	?	1	0	?
poor print quality	1	1	1	0	1	1	0	1	0	0	1	1	?	?	0
wrinkled pages	0	0	1	0	0	0	?	0	1	?	0	0	1	1	1
multiple pages fed	0	?	1	0	?	0	1	0	1	?	0	0	?	0	1
paper jam	?	0	1	1	?	0	1	1	1	1	0	?	0	1	?

The table is contained in `EMprinter.mat`*, using states* `1,2,nan` *in place of* `0,1,?` *(since* BRMLtoolbox *requires states to be numbered 1,2,....). Given no wrinkled pages, no burning smell and poor print quality, what is the probability there is a drum unit problem?*

Exercise 139. *Consider the following distribution over discrete variables,*

$$p(x_1, x_2, x_3, x_4, x_5) = p(x_1|x_2)p(x_2|x_3)p(x_3|x_4)p(x_4|x_5)p(x_5), \tag{11.9.1}$$

in which the variables x_2 and x_4 are consistently hidden in the training data, and training data for x_1, x_3, x_5 are always present. Show that the EM update for the table $p(x_1|x_2)$ is given by

$$p^{new}(x_1 = i|x_2 = j) = \frac{\sum_n \mathbb{I}[x_1^n = i]\, p^{old}(x_2 = j|x_1^n, x_3^n, x_5^n)}{\sum_{n,k} \mathbb{I}[x_1^n = k]\, p^{old}(x_2 = j|x_1^n, x_3^n, x_5^n)} \tag{11.9.2}$$

Exercise 140. *Consider a simple two variable BN*

$$p(y, x) = p(y|x)p(x) \tag{11.9.3}$$

where both y and x are binary variables, $dom(x) = \{1, 2\}$, $dom(y) = \{1, 2\}$. You have a set of training data $\{(y^n, x^n), n = 1, \ldots, N\}$, in which for some cases x^n may be missing. We are specifically interested in learning the table $p(x)$ from this data. A colleague suggests that one can set $p(x)$ by simply looking at datapoints where x is observed, and then setting $p(x = 1)$ to be the fraction of observed x that is in state 1. Explain how this suggested procedure relates to Maximum Likelihood and EM.

Exercise 141. *Assume that a sequence is generated by a Markov chain. For a single chain of length T, we have*

$$p(v_1, \ldots, v_T) = p(v_1) \prod_{t=1}^{T-1} p(v_{t+1}|v_t) \tag{11.9.4}$$

For simplicity, we denote the sequence of visible variables as

$$\mathbf{v} = (v_1, \ldots, v_T) \tag{11.9.5}$$

For a single Markov chain labelled by h,

$$p(\mathbf{v}|h) = p(v_1|h) \prod_{t=1}^{T-1} p(v_{t+1}|v_t, h) \tag{11.9.6}$$

In total there are a set of H such Markov chains ($h = 1, \ldots, H$). The distribution on the visible variables is therefore

$$p(\mathbf{v}) = \sum_{h=1}^{H} p(\mathbf{v}|h)p(h) \tag{11.9.7}$$

1. *There are a set of training sequences, $\mathbf{v}^n, n = 1, \ldots, N$. Assuming that each sequence $\mathbf{v}^n$ is independently and identically drawn from a Markov chain mixture model with H components, derive the Expectation Maximisation algorithm for training this model.*

2. *Write a general* MATLAB *function in the form*

```
function [q,ph,pv,A]=mchain_mix(v,V,H,num_em_loops)
```

to perform EM learning for any set of (the same length) sequences of integers $v_t^n \in [1 : V]$, $t = 1, \ldots, T$. `v` *is a cell array of the training data:* `v{2}(4)` *is the 4th time element of the second training sequence. Each element, say* `v{2}(4)` *must be an integer from 1 to V.* `V` *is the number of states of the visible variables (in the bio-sequence case below, this will be 4).* `H` *is the* *number of mixture components.* `num_em_loops` *is the number of EM iterations.* `A` *is the transition matrix* `A{h}(i,j)=p(v(t+1)=i|v(t)=j,h)`. `pv` *is the prior state of the first visible variable,* `pv{h}(i)=p(v(t=1)=i|h)`. `ph` *is a vector of prior probabilities for the mixture state* `ph(h)=p(h)`. `q` *is the cell array of posterior probabilities* `q{mu}(h)=p(h|v{mu})`. *Your routine must also display, for each EM iteration, the value of the log likelihood. As a check on your routine, the log likelihood must increase at each iteration.*

3. *The file* `sequences.mat` *contains a set of fictitious bio-sequence in a cell array* `sequences{mu}(t)`. *Thus* `sequences{3}(:)` *is the third sequence,* `GTCTCCTGCCCTCTCTGAAC` *which consists of 20 timesteps. There are 20 such sequences in total. Your task is to cluster these sequences into two clusters, assuming that each cluster is modelled by a Markov chain. State which of the sequences belong together by assigning a sequence $\mathbf{v}^n$ to that state for which $p(h|\mathbf{v}^n)$ is highest.*

Exercise 142. *Write a general purpose routine* `VBbeliefnet(pot,x,pars)` *along the lines of* `EMbeliefnet.m` *that performs Variational Bayes under a Dirichlet prior, using a factorised parameter approximation. Assume both global and local parameter independence for the prior and the approximation q, section(9.3.1).*

Exercise 143. *Consider a 3 'layered' Boltzmann Machine which has the form*

$$p(\mathbf{v}, \mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3|\theta) = \frac{1}{Z}\phi(\mathbf{v}, \mathbf{h}_1|\theta^1)\phi(\mathbf{h}_1, \mathbf{h}_2|\theta^2)\phi(\mathbf{h}_2, \mathbf{h}_3|\theta^3) \tag{11.9.8}$$

where $\dim \mathbf{v} = \dim \mathbf{h}_1 = \dim \mathbf{h}_2 = \dim \mathbf{h}_3 = V$

$$\phi(\mathbf{x}, \mathbf{y}|\theta) = e^{\sum_{i,j=1}^{V} W_{ij}x_iy_j + A_{ij}x_ix_j + B_{ij}y_iy_j} \tag{11.9.9}$$

All variables are binary with states $0, 1$ and the parameters for each layer l are $\theta^l = \left\{\mathbf{W}^l, \mathbf{A}^l, \mathbf{B}^l\right\}$.

1. *In terms of fitting the model to visible data $\mathbf{v}^1, \ldots, \mathbf{v}^N$, is the 3 layered model above any more powerful than fitting a two-layered model (the factor $\phi(\mathbf{h}_2, \mathbf{h}_3|\theta^3)$ is not present in the two-layer case)?*

2. *If we use a restricted potential*

$$\phi(\mathbf{x}, \mathbf{y}|\theta) = e^{\sum_{i,j} W_{ij}x_iy_j} \tag{11.9.10}$$

is the three layered model more powerful in being able to fit the visible data than the two-layered model?

Exercise 144. *The* sigmoid Belief Network *is defined by the layered network*

$$p(\mathbf{x}^L)\prod_{l=1}^{L} p(\mathbf{x}^{l-1}|\mathbf{x}^l) \tag{11.9.11}$$

where vector variables have binary components $\mathbf{x}^l \in \{0, 1\}^{w_l}$ and the width of layer l is given by w_l. In addition

$$p(\mathbf{x}^{l-1}|\mathbf{x}^l) = \prod_{i=1}^{w_l} p(x_i^{l-1}|\mathbf{x}^l) \tag{11.9.12}$$

and

$$p(x_i^{l-1} = 1|\mathbf{x}^l) = \sigma\left(\mathbf{w}_{i,l}^T\mathbf{x}^l\right), \qquad \sigma(x) = 1/(1 + e^{-x}) \tag{11.9.13}$$

for a weight vector $\mathbf{w}_{i,l}$ describing the interaction from the parental layer. The top layer, $p(\mathbf{x}^L)$ describes a factorised distribution $p(x_1^L), \ldots, p(x_{w_L}^L)$.

1. *Draw the Belief Network structure of this distribution.*

2. *For the layer $\mathbf{x}^0$, what is the computational complexity of computing the likelihood $p(\mathbf{x}^0)$, assuming that all layers have equal width w?*

3. *Assuming a fully factorised approximation for an equal width network,*

$$p(\mathbf{x}^1, \ldots, \mathbf{x}^L|\mathbf{x}^0) \approx \prod_{l=1}^{L} \prod_{i=1}^{w} q(x_i^l) \tag{11.9.14}$$

write down the energy term of the Variational EM procedure for a single data observation $\mathbf{x}^0$*, and discuss the tractability of computing the energy.*

Exercise 145. *Show how to find the components* $0 \le (\theta_b, \theta_g, \theta_p) \le 1$ *that maximise equation (11.1.10).*

Exercise 146. *A* 2×2 *probability table,* $p(x_1 = i, x_2 = j) = \theta_{i,j}$*, with* $0 \le \theta_{i,j} \le 1$*,* $\sum_{i=1}^{2} \sum_{j=1}^{2} \theta_{i,j} = 1$ *is learned using maximal marginal likelihood in which* x_2 *is never observed. Show that if*

$$\theta = \begin{pmatrix} 0.3 & 0.3 \\ 0.2 & 0.2 \end{pmatrix} \tag{11.9.15}$$

is given as a maximal marginal likelihood solution, then

$$\theta = \begin{pmatrix} 0.2 & 0.4 \\ 0.4 & 0 \end{pmatrix} \tag{11.9.16}$$

has the same marginal likelihood score.

CHAPTER 12

Bayesian Model Selection

12.1 Comparing models the Bayesian way

Given two models M_1 and M_2 with parameters θ_1, θ_2 and associated parameter priors,

$$p(x, \theta_1|M_1) = p(x|\theta_1, M_1)p(\theta_1|M_1), \qquad p(x, \theta_2|M_2) = p(x|\theta_2, M_2)p(\theta_2|M_2) \tag{12.1.1}$$

how can we compare the performance of the models in fitting a set of data $\mathcal{D} = \{x_1, \ldots, x_N\}$? The application of Bayes' rule to models gives a framework for answering questions like this – a form of Bayesian Hypothesis testing, applied at the model level. More generally, given an indexed set of models $M_1, \ldots, M_m$, and associated prior beliefs in the appropriateness of each model $p(M_i)$, our interest is the model posterior probability

$$p(M_i|\mathcal{D}) = \frac{p(\mathcal{D}|M_i)p(M_i)}{p(\mathcal{D})} \tag{12.1.2}$$

where

$$p(\mathcal{D}) = \sum_{i=1}^{m} p(\mathcal{D}|M_i)p(M_i) \tag{12.1.3}$$

Model M_i is parameterised by θ_i, and the model likelihood is given by

$$p(\mathcal{D}|M_i) = \int p(\mathcal{D}|\theta_i, M_i)p(\theta_i|M_i)d\theta_i \tag{12.1.4}$$

In discrete parameter spaces, the integral is replaced with summation. Note that the number of parameters $\dim(\theta_i)$ need not be the same for each model.

A point of caution here is that $p(M_i|\mathcal{D})$ only refers to the probability relative to the set of models specified $M_1, \ldots, M_m$. This is not the *absolute* probability that model M fits 'well'. To compute such a quantity would require one to specify *all* possible models. Whilst interpreting the posterior $p(M_i|\mathcal{D})$ requires some care, comparing two competing model hypotheses M_i and M_j is straightforward and only requires the *Bayes' factor*

$$\frac{p(M_i|\mathcal{D})}{p(M_j|\mathcal{D})} = \underbrace{\frac{p(\mathcal{D}|M_i)}{p(\mathcal{D}|M_j)}}_{\text{Bayes' Factor}} \frac{p(M_i)}{p(M_j)} \tag{12.1.5}$$

which does not require integration/summation over all possible models.

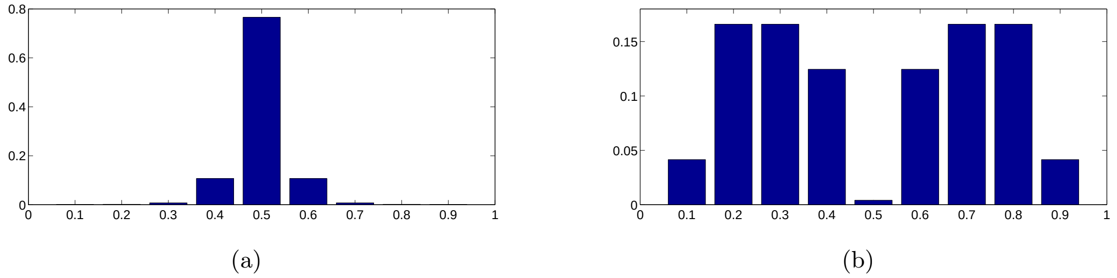


Figure 12.1: **(a)**: Discrete prior model of a 'fair' coin. **(b)**: Prior for a biased 'unfair' coin. In both cases we are making explicit choices here about what we consider to be a 'fair' and and 'unfair'.

12.2 Illustrations : coin tossing

We'll consider two illustrations. The first uses a discrete parameter space to keep the mathematics simple. In the second we use a continuous parameter space.

12.2.1 A discrete parameter space

A simple choice would be to consider two competing models, one corresponding to a fair coin, and the other a biased coin. The bias of the coin, namely the probability that the coin will land heads, is specified by θ, so that a truly fair coin has $\theta = 0.5$. For simplicity we assume $\mathrm{dom}(\theta) = \{0.1, 0.2, \ldots, 0.9\}$. For the fair coin we use the distribution $p(\theta|M_{fair})$ in fig(12.1a) and for the biased coin the distribution $p(\theta|M_{biased})$ in fig(12.1b).

For each model M, the likelihood is given by

$$p(\mathcal{D}|M) = \sum_{\theta} p(\mathcal{D}|\theta, M)p(\theta|M) = \sum_{\theta} \theta^{N_H} (1-\theta)^{N_T} p(\theta|M) \tag{12.2.1}$$

$$= 0.1^{N_H} (1-0.1)^{N_T} p(\theta = 0.1|M) + \ldots 0.9^{N_H} (1-0.9)^{N_T} p(\theta = 0.9|M) \tag{12.2.2}$$

Assuming that $p(M_{fair}) = p(M_{biased})$ the Bayes' factor is given by the ratio of the two model likelihoods.

Example 55 (Discrete parameter space)**.**

5 Heads and 2 Tails Here $p(\mathcal{D}|M_{fair}) = 0.00786$ and $p(\mathcal{D}|M_{biased}) = 0.0072$. The Bayes' factor is

$$\frac{p(M_{fair}|\mathcal{D})}{p(M_{biased}|\mathcal{D})} = 1.09 \tag{12.2.3}$$

indicating that there is little to choose between the two models.

50 Heads and 20 Tails Here $p(\mathcal{D}|M_{fair}) = 1.5 \times 10^{-20}$ and $p(\mathcal{D}|M_{biased}) = 1.4 \times 10^{-19}$. The Bayes' factor is

$$\frac{p(M_{fair}|\mathcal{D})}{p(M_{biased}|\mathcal{D})} = 0.109 \tag{12.2.4}$$

indicating that have around 10 times the belief in the biased model as opposed to the fair model.

12.2.2 A continuous parameter space

Here we repeat the above calculation but for continuous parameter spaces.

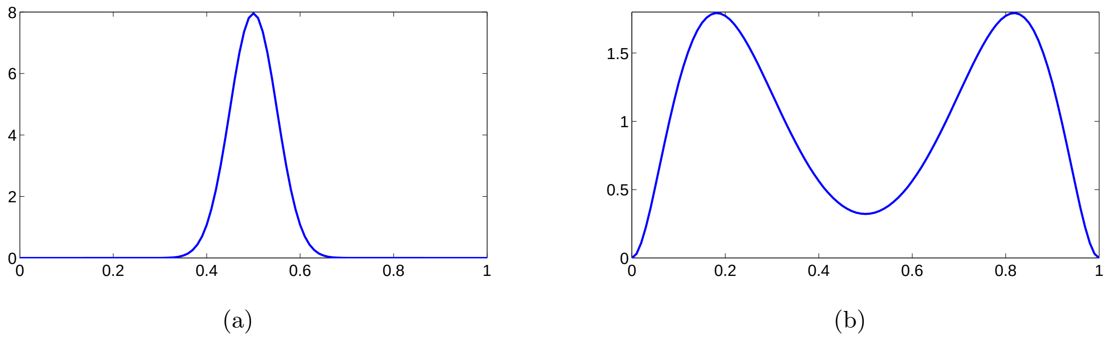


Figure 12.2: Probability density priors on the probability of a Head $p(\theta)$. **(a)**: For a fair coin, $p(\theta|M_{fair}) = B(\theta|50, 50)$. **(b)**: For an biased coin, $p(\theta|M_{biased}) = 0.5\,(B(\theta|3, 10) + B(\theta|10, 3))$. Note the different vertical scales in the two cases.

Fair coin

For the fair coin, a uni-modal prior is appropriate. We use Beta distribution

$$p(\theta) = B(\theta|a, b), \qquad B(\theta|a, b) \equiv \frac{1}{B(a, b)}\theta^{a-1}(1-\theta)^{b-1} \tag{12.2.5}$$

for convenience since as this is conjugate to the binomial distribution the required integrations are trivial. A reasonable choice for a fair coin is $a = 50$, $b = 50$, as shown in fig(12.2a).

In general,

$$p(\mathcal{D}|M_{fair}) = \int_\theta p(\theta)\theta^{N_H}(1-\theta)^{N_T} = \frac{1}{B(a, b)}\int_\theta \theta^{a-1}(1-\theta)^{b-1}\theta^{N_H}(1-\theta)^{N_T} \tag{12.2.6}$$

$$= \frac{1}{B(a, b)}\int_\theta \theta^{N_H+a-1}(1-\theta)^{N_T+b-1} = \frac{B(N_H + a, N_T + b)}{B(a, b)} \tag{12.2.7}$$

Biased coin

For the biased coin, we use a bimodal distribution formed, for convenience, as a mixture of two Beta distributions:

$$p(\theta|M_{biased}) = \frac{1}{2}\left[B(\theta|a_1, b_1) + B(\theta|a_2, b_2)\right] \tag{12.2.8}$$

as shown in fig(12.2b). The model likelihood $p(\mathcal{D}|M_{biased})$ is given by

$$\int_\theta p(\theta|M_{biased})\theta^{N_H}(1-\theta)^{N_T} \tag{12.2.9}$$

$$= \frac{1}{2}\left\{\frac{1}{B(a_1, b_1)}\int_\theta \theta^{a_1-1}(1\theta)^{b_1-1}\theta^{N_H}(1-\theta)^{N_T} + \frac{1}{B(a_2, b_2)}\int_\theta \theta^{a_2-1}(1-\theta)^{b_2-1}\theta^{N_H}(1-\theta)^{N_T}\right\} \tag{12.2.10}$$

$$= \frac{1}{2}\left\{\frac{B(N_H + a_1, N_T + b_1)}{B(a_1, b_1)} + \frac{B(N_H + a_2, N_T + b_2)}{B(a_2, b_2)}\right\} \tag{12.2.11}$$

Assuming no prior preference for either a fair or biased coin $p(M) = \text{const.}$, and repeating the above scenario in the discrete parameter case:

Example 56 (Continuous parameter space)**.**

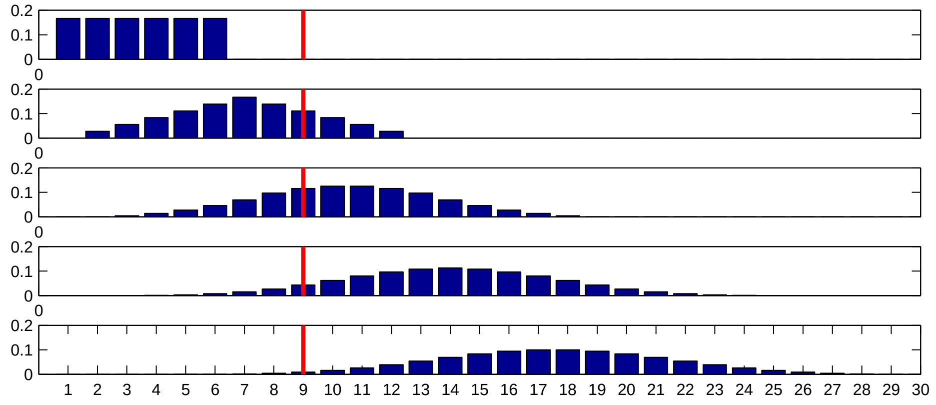


Figure 12.3: The likelihood of the total dice score, $p(t|n)$ for $n = 1$ (top) to $n = 5$ (bottom) die. Plotted along the horizontal axis is the total score t. The vertical line marks the comparison for $p(t = 9|n)$ for the different number of die. The more complex models, which can reach more states, have lower likelihood, due to normalisation over t.

5 Heads and 2 Tails Here $p(\mathcal{D}|M_{fair}) = 0.0079$ and $p(\mathcal{D}|M_{biased}) = 0.00622$. The Bayes' factor is

$$\frac{p(M_{fair}|\mathcal{D})}{p(M_{biased}|\mathcal{D})} = 1.27 \tag{12.2.12}$$

indicating that there is little to choose between the two models.

50 Heads and 20 Tails Here $p(\mathcal{D}|M_{fair}) = 9.4 \times 10^{-21}$ and $p(\mathcal{D}|M_{biased}) = 1.09 \times 10^{-19}$. The Bayes' factor is

$$\frac{p(M_{fair}|\mathcal{D})}{p(M_{biased}|\mathcal{D})} = 0.087 \tag{12.2.13}$$

indicating that have around 11 times the belief in the biased model as opposed to the fair model.

12.3 Occam's Razor and Bayesian Complexity Penalisation

We return to the dice scenario of section(1.3.1). There we assumed there are two die whose scores s_1 and s_2 are not known. Only the sum of the two scores $t = s_1 + s_2$ is known. We then computed the posterior joint score distribution $p(s_1, s_2|t = 9)$ for the two die. We repeat the calculation but now for multiple dice and with the twist that we don't know how many dice there are[1], only that the sum of the scores is 9. That is, we know $t = \sum_{i=1}^{n} s_i$ and are given the value $t = 9$. However, we are not told the number of die involved n. Assuming that any number n is equally likely, what is the posterior distribution over n?

From Bayes' rule, we need to compute the posterior distribution over models

$$p(n|t) = \frac{p(t|n)p(n)}{p(t)} \tag{12.3.1}$$

In the above

$$p(t|n) = \sum_{s_1,\ldots,s_n} p(t, s_1, \ldots, s_n|n) = \sum_{s_1,\ldots,s_n} p(t|s_1, \ldots, s_n) \prod_i p(s_i) = \sum_{s_1,\ldots,s_n} \mathbb{I}\left[t = \sum_{i=1}^{n} s_i\right] \prod_i p(s_i) \tag{12.3.2}$$

[1]This description of Occam's razor is due to Taylan Cemgil.

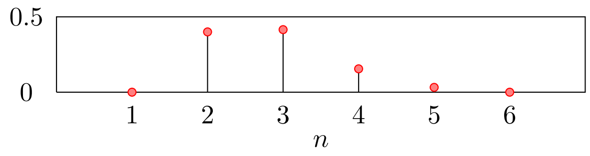


Figure 12.4: The posterior distribution $p(n|t = 9)$ of the number of die given the observed summed score of 9.

where $p(s_i) = 1/6$ for all scores s_i. By enumerating all 6^n states, we can explicitly compute $p(t|n)$, as displayed in fig(12.3). The important observation is that as the models explaining the data become more 'complex' (n increases), more states become accessible and the probability mass typically reduces. We see this effect at $p(t = 9|n)$ where, apart from $n = 1$, the value of $p(t = 9|n)$ decreases with increasing n since the higher n have mass in more states, becoming more spread out. Assuming $p(n) = \mathsf{const.}$, the posterior $p(n|t = 9)$ is plotted in fig(12.4). A posteriori, there are only 3 plausible models, namely $n = 2, 3, 4$ since the rest are either too complex, or impossible. This demonstrates the *Occam's razor* effect which penalises models which are over complex.

12.4 A continuous example : curve fitting

Consider an additive set of periodic functions

$$y^0 = w_0 + w_1 \cos(x) + w_2 \cos(2x) + \ldots + w_K \cos(Kx) \tag{12.4.1}$$

This can be conveniently written in vector form

$$y^0 = \mathbf{w}^{\mathsf{T}} \boldsymbol{\phi}(x) \tag{12.4.2}$$

where $\boldsymbol{\phi}(x)$ is a $K + 1$ dimensional vector with elements $(1, \cos(x), \cos(2x), \ldots, \cos(Kx))^{\mathsf{T}}$ and the vector $\mathbf{w}$ contains the weights of the additive function. We are given a set of data $\mathcal{D} = \{(x^n, y^n), n = 1, \ldots, N\}$ drawn from this distribution, where y is the clean $y^0(x)$ corrupted with additive zero mean Gaussian noise with variance σ^2,

$$y^n = y^0(x^n) + \epsilon^n, \quad \epsilon^n \sim \mathcal{N}\left(\epsilon^n|0, \sigma^2\right) \tag{12.4.3}$$

see fig(12.5). Assuming i.i.d. data, we are interested in the posterior probability of the number of coefficients, given the observed data:

$$p(K|\mathcal{D}) = \frac{p(\mathcal{D}|K)p(K)}{p(\mathcal{D})} = \frac{p(K)\prod_n p(x^n)}{p(\mathcal{D})} p(y^1, \ldots y^N|x^1, \ldots, x^N, K) \tag{12.4.4}$$

We will assume $p(K) = \mathsf{const}$. The likelihood term above is given by the integral

$$p(y^1, \ldots y^N|x^1, \ldots, x^N, K) = \int_{\mathbf{w}} p(\mathbf{w}|K) \prod_{n=1}^{N} p(y^n|x^n, \mathbf{w}, K) \tag{12.4.5}$$

For $p(\mathbf{w}|K) = \mathcal{N}(\mathbf{w}|0, \mathbf{I}_K/\alpha)$, the integrand is a Gaussian in $\mathbf{w}$ for which it is straightforward to evaluate the integral, (see section(8.6) and exercise(149))

$$2 \log p(y^1, \ldots, y^N|x^1, \ldots, y^N, K) = N \log\left(2\pi\sigma^2\right) - \sum_{n=1}^{N} \frac{(y^n)^2}{\sigma^2} + \mathbf{b}^{\mathsf{T}}\mathbf{A}^{-1}\mathbf{b} - \log\det\left(2\pi\mathbf{A}\right) + K \log\left(2\pi\alpha\right)$$

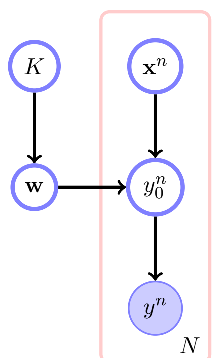
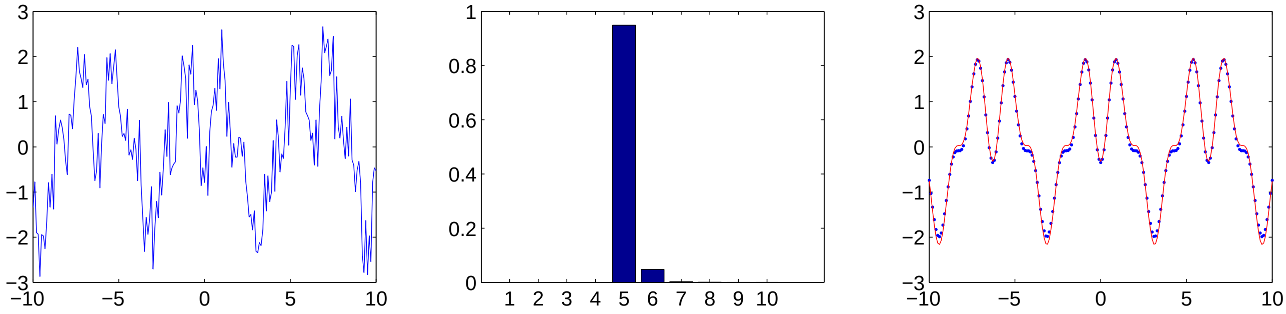


Figure 12.5: Belief Network representation of a Hierarchical Bayesian Model for regression under the i.i.d. data assumption. Note that the intermediate nodes on y_0^n are included to highlight the role of the 'clean' underlying model. Since $p(y|\mathbf{w}, \mathbf{x}) = \int_{y_0} p(y|y_0)p(y_0|\mathbf{w}, \mathbf{x}) = \int_{y_0} \mathcal{N}\left(y|y_0, \sigma^2\right) \delta\left(y_0 - \mathbf{w}^{\mathsf{T}}\mathbf{x}\right) = \mathcal{N}\left(y|\mathbf{w}^{\mathsf{T}}\mathbf{x}, \sigma^2\right)$, we can if desired do away with the intermediate node y_0 and place directly arrows from $\mathbf{w}$ and $\mathbf{x}^n$ to y^n.



(a) (b) (c)

Figure 12.6: (a) The data generated with additive Gaussian noise $\sigma = 0.5$ from a $K = 5$ component model. (b) The posterior $p(K|\mathcal{D})$. (c) The reconstruction of the data using $\langle \mathbf{w} \rangle^{\mathsf{T}} \boldsymbol{\phi}(x)$ where $\langle \mathbf{w} \rangle$ is the mean posterior vector of the optimal dimensional model $p(\mathbf{w}|\mathcal{D}, K = 5)$. Plotted in the continuous line is the reconstruction. Plotted in dots is the true underlying clean data.

(12.4.6)

where

$$\mathbf{A} \equiv \alpha \mathbf{I} + \frac{1}{\sigma^2} \sum_{n=1}^{N} \boldsymbol{\phi}(x^n) \boldsymbol{\phi}^{\mathsf{T}}(x^n), \qquad \mathbf{b} \equiv \frac{1}{\sigma^2} \sum_{n=1}^{N} y^n \boldsymbol{\phi}(x^n) \tag{12.4.7}$$

Assuming $\alpha = 1$ and $\sigma = 0.5$, we sampled some data from a model with $K = 5$ components, fig(12.6a). We assume that we know the correct noise level σ. The posterior $p(K|\mathcal{D})$ plotted in fig(12.6b) is sharply peaked at $K = 5$, which is the 'correct' value used to generate the data. The clean reconstructions for $K = 5$ are plotted in fig(12.6c).

12.5 Approximating the Model Likelihood

For a model with continuous parameter vector $\boldsymbol{\theta}$, $\dim(\boldsymbol{\theta}) = K$ and data $\mathcal{D}$, the model likelihood is

$$p(\mathcal{D}|M) = \int_{\boldsymbol{\theta}} p(\mathcal{D}|\boldsymbol{\theta}, M) p(\boldsymbol{\theta}|M) d\boldsymbol{\theta} \tag{12.5.1}$$

For a generic expression

$$p(\mathcal{D}|\boldsymbol{\theta}, M) p(\boldsymbol{\theta}|M) = e^{-f(\boldsymbol{\theta})} \tag{12.5.2}$$

unless f is of a particularly simple form (quadratic in $\boldsymbol{\theta}$ for example), one cannot compute the integral in (12.5.1) and approximations are required.

12.5.1 Laplace's method

A simple approximation of (12.5.1) is given by Laplace's method, section(28.2),

$$\log p(\mathcal{D}|M) \approx -f(\boldsymbol{\theta}^*) + \frac{1}{2} \log \det \left(2\pi \mathbf{H}^{-1} \right) \tag{12.5.3}$$

where $\boldsymbol{\theta}^*$ is the MAP solution

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\; p(\mathcal{D}|\boldsymbol{\theta}, M) p(\boldsymbol{\theta}|M) \tag{12.5.4}$$

and $\mathbf{H}$ is the Hessian of $f(\boldsymbol{\theta})$ at $\boldsymbol{\theta}^*$.

For data $\mathcal{D} = \{x^1, \ldots, x^N\}$ that is i.i.d. generated the above specialises to

$$p(\mathcal{D}|M) = \int_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|M) \prod_{n=1}^{N} p(x^n|\boldsymbol{\theta}, M) d\boldsymbol{\theta} \tag{12.5.5}$$

In this case Laplace's method computes the optimum of the function

$$-f(\boldsymbol{\theta}) = \log p(\boldsymbol{\theta}|M) + \sum_{n=1}^{N} \log p(x^n|\boldsymbol{\theta}, M) \tag{12.5.6}$$

12.5.2 Bayes Information Criterion (BIC)

For i.i.d. data the Hessian scales with the number of training examples, N, and a crude approximation is to set $\mathbf{H} \approx N\mathbf{I}_K$ where $K = \dim \boldsymbol{\theta}$. In this case one may take as a model comparison procedure the function

$$\log p(\mathcal{D}|M) \approx \log p(\mathcal{D}|\boldsymbol{\theta}^*, M) + \log p(\boldsymbol{\theta}^*|M) + \frac{K}{2}\log 2\pi - \frac{K}{2}\log N \tag{12.5.7}$$

For a simple prior that penalises the length of the parameter vector, $p(\boldsymbol{\theta}|M) = \mathcal{N}(\boldsymbol{\theta}|\mathbf{0}, \mathbf{I})$, the above reduces to

$$\log p(\mathcal{D}|M) \approx \log p(\mathcal{D}|\boldsymbol{\theta}^*, M) - \frac{1}{2}(\boldsymbol{\theta}^*)^{\mathsf{T}} \boldsymbol{\theta}^* - \frac{K}{2}\log N \tag{12.5.8}$$

The Bayes Information Criterion[241] approximates (12.5.7) by ignoring the penalty term, giving

$$BIC = \log p(\mathcal{D}|\boldsymbol{\theta}^*, M) - \frac{K}{2}\log N \tag{12.5.9}$$

The BIC criterion may be used as an approximate way to compare models, where the term $-\frac{K}{2}\log N$ penalises model complexity. In general, the Laplace approximation, equation (12.5.3), is to be preferred to the BIC criterion since it more correctly accounts for the uncertainty in the posterior parameter estimate.

12.6 Exercises

Exercise 147. *Write a program to implement the fair/biased coin tossing model selection example of section(12.2.1) using a discrete domain for θ. Explain how to overcome potential numerical issues in dealing with large N_H and N_T (of the order of 1000).*

Exercise 148. *You work at Dodder's hedge fund and the manager wants to model next day 'returns' y_{t+1} based on current day information $\mathbf{x}_t$. The vector of 'factors' each day, $\mathbf{x}_t$ captures essential aspects of the market. He argues that a simple linear model*

$$y_{t+1} = \sum_{k=1}^{K} w_k x_{kt} \tag{12.6.1}$$

should be reasonable and asks you to find the weight vector $\mathbf{w}$, based on historical information $\mathcal{D} = \{(\mathbf{x}_t, y_{t+1}), t = 1, \ldots, T-1\}$. In addition he also gives you a measure of the 'volatility' σ_t^2 for each day.

1. Under the assumption that the returns are i.i.d. Gaussian distributed

$$p(y_{1:T}|\mathbf{x}_{1:T}, \mathbf{w}) = \prod_{t=2}^{T} p(y_t|\mathbf{x}_{t-1}, \mathbf{w}) = \prod_{t=2}^{T} \mathcal{N}\left(y_t|\mathbf{w}^{\mathsf{T}}\mathbf{x}_{t-1}, \sigma_t^2\right) \tag{12.6.2}$$

   explain how to set the weight vector $\mathbf{w}$ by Maximum Likelihood.

2. Your hedge fund manager is however convinced that some of the factors are useless for prediction and wishes to remove as many as possible. To do this you decide to use a Bayesian model selection method in which you use a prior

$$p(\mathbf{w}|M) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \mathbf{I}) \tag{12.6.3}$$

where $M = 1, \ldots, 2^K - 1$ indexes the model. Each model uses only a subset of the factors. By translating the integer M into a binary vector representation, the model describes which factors are to be used. For example if $K = 3$, there would be 7 models

$$\{0,0,1\}, \{0,1,0\}, \{1,0,0\}, \{0,1,1\}, \{1,0,1\}, \{1,1,0\}, \{1,1,1\} \tag{12.6.4}$$

where the first model is $y_t = w_3 x_3$ with weight prior $p(w_3) = \mathcal{N}(w_3|0,1)$. Similarly model 7 would be $y_t = w_1 x_1 + w_2 x_2 + w_3 x_3$ with $p(w_1, w_2, w_3) = \mathcal{N}((w_1, w_2, w_3)|(0,0,0), \mathbf{I}_3)$. You decide to use a flat prior $p(M) = \mathsf{const}$. Draw the hierarchical Bayesian network for this model and explain how to find the best model for the data using Bayesian model selection by suitably adapting equation (12.4.6).

3. Using the data `dodder.mat`, perform Bayesian model selection as above for $K = 6$ and find which of the factors $x_1, \ldots, x_6$ are most likely to explain the data.

Exercise 149. *Here we will derive the expression (12.4.6) and also an alternative form.*

1. *Starting from*

$$p(\mathbf{w}) \prod_{n=1}^{N} p(y^n|\mathbf{w}, x^n, K) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \mathbf{I}/\alpha) \prod_n \mathcal{N}\left(y^n|\mathbf{w}^{\mathsf{T}}\boldsymbol{\phi}(x^n), \sigma^2\right) \tag{12.6.5}$$

$$= \frac{1}{\sqrt{2\pi\alpha^{-1}}} e^{-\frac{\alpha}{2}\mathbf{w}^{\mathsf{T}}\mathbf{w}} \frac{1}{(2\pi\sigma^2)^{N/2}} e^{-\frac{1}{2\sigma^2}\sum_n \left(y^n - \mathbf{w}^{\mathsf{T}}\boldsymbol{\phi}(x^n)\right)^2} \tag{12.6.6}$$

Show that this can be expressed as

$$\frac{1}{\sqrt{2\pi\alpha^{-1}}} \frac{1}{(2\pi\sigma^2)^{N/2}} e^{-\frac{1}{2\sigma^2}\sum_n (y^n)^2} e^{-\frac{1}{2}\mathbf{w}^{\mathsf{T}}\mathbf{A}\mathbf{w} + \mathbf{b}^{\mathsf{T}}\mathbf{w}} \tag{12.6.7}$$

where

$$\mathbf{A} = \alpha\mathbf{I} + \frac{1}{\sigma^2}\sum_n \boldsymbol{\phi}(x^n)\boldsymbol{\phi}^{\mathsf{T}}(x^n) \qquad \mathbf{b} = \frac{1}{\sigma^2}\sum_n y^n \boldsymbol{\phi}(x^n) \tag{12.6.8}$$

2. *By completing the square (see section(8.6.2)), derive (12.4.6).*

3. *Since each y^n, $n = 1, \ldots, N$ is linearly related through $\mathbf{w}$ and $\mathbf{w}$ is Gaussian distributed, the joint vector $y^1, \ldots, y^N$ is Gaussian distributed. Using the Gaussian propagation results, section(8.6.3), derive an alternative expression for $\log p(y^1, \ldots, y^N|x^1, \ldots, x^N)$.*

Part III

Machine Learning

CHAPTER 13

Machine Learning Concepts

13.1 Styles of Learning

Broadly speaking the main two subfields of Machine Learning are *supervised learning* and *unsupervised learning*. In supervised learning the focus is on accurate prediction, whereas in unsupervised learning the aim is to find accurate compact descriptions of the data.

Particularly in supervised learning, one is interested in methods that perform well on previously unseen data. That is, the method 'generalises' to unseen data. In this sense, one distinguishes between data that is used to train a model, and data that is used to test the performance of the trained model, see fig(13.1).

13.1.1 Supervised Learning

Consider a database of face images, each represented by a vector[1] $\mathbf{x}$. Along with each image $\mathbf{x}$ is an output class $y \in \{\mathsf{male}, \mathsf{female}\}$ that states if the image is of a male or female. A database of 10000 such image-class pairs is available, $\mathcal{D} = \{(\mathbf{x}^n, y^n), n = 1, \ldots, 10000\}$. The task is to make an accurate predictor $y(\mathbf{x}^*)$ of the sex of a novel image $\mathbf{x}^*$. This is an example application that would be hard to program in a traditional 'programming' manner since formally specifying how male faces differ from female faces is difficult. An alternative is to give examples faces and their gender labels and let a machine automatically 'learn' a rule to differentiate male from female faces.

Definition 88 (Supervised Learning)**.** Given a set of data $\mathcal{D} = \{(x^n, y^n), n = 1, \ldots, N\}$ the task is to 'learn' the relationship between the input x and output y such that, when given a new input x^* the predicted output y^* is accurate. To specify explicitly what accuracy means one defines a loss function $L(y^{pred}, y^{true})$ or, conversely, a utility function $U = -L$.

In supervised learning our interest is describing y conditioned on knowing x. From a probabilistic modelling perspective, we are therefore concerned primarily with the conditional distribution $p(y|x, \mathcal{D})$. The term 'supervised' indicates that there is a 'supervisor' specifying the output y for each input x in the available data $\mathcal{D}$. The output is also called a 'label', particularly when discussing classification.

Predicting tomorrow's stock price $y(T+1)$ based on past observations $y(1), \ldots, y(T)$ is a form of supervised learning. We have a collection of times and prices $\mathcal{D} = \{(t, y(t)), t = 1, \ldots, T\}$ where time t is the 'input' and the price $y(t)$ is the output.

[1] For an $m \times n$ face image with elements F_{mn} we can form a vector by stacking the entries of the matrix. In MATLAB one may achieve this using `x=F(:)`.

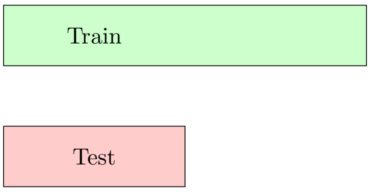


Figure 13.1: In training and evaluating a model, conceptually there are two sources of data. The parameters of the model are set on the basis of the train data only. If the test data is generated from the same underlying process that generated the train data, an unbiased estimate of the generalisation performance can be obtained by measuring the test data performance of the trained model. Importantly, the test performance should not be used to adjust the model parameters since we would then no longer have an independent measure of the performance of the model.

Example 57. A father decides to teach his young son what a sports car is. Finding it difficult to explain in words, he decides to give some examples. They stand on a motorway bridge and, as each car passes underneath, the father cries out 'that's a sports car!' when a sports car passes by. After ten minutes, the father asks his son if he's understood what a sports car is. The son says, 'sure, it's easy'. An old red VW Beetle passes by, and the son shouts – 'that's a sports car!'. Dejected, the father asks – 'why do you say that?'. 'Because all sports cars are red!', replies the son.

This story is an example scenario for supervised learning. Here the father plays the role of the supervisor, and his son is the 'student' (or 'learner'). It's indicative of the kinds of problems encountered in machine learning in that it is not really clear anyway what a sports car is – if we knew that, then we wouldn't need to go through the process of learning. This example also highlights the issue that there is a difference between performing well on training data and performing well on novel test data. The main interest in supervised learning is to discover an underlying rule that will generalise well, leading to accurate prediction on new inputs.

For an input x, if the output is one of a discrete number of possible 'classes', this is called a *classification problem*. In classification problems we will generally use c for the output.

For an input x, if the output is continuous, this is called a *regression problem*. For example, based on historical information of demand for sun-cream in your supermarket, you are asked to predict the demand for the next month. In some cases it is possible to discretise a continuous output and then consider a corresponding classification problem. However, in other cases it is impractical or unnatural to do this; for example if the output y is a high dimensional continuous valued vector, or if the ordering of states of the variable is meaningful.

13.1.2 Unsupervised Learning

Definition 89 (Unsupervised learning)**.** Given a set of data $\mathcal{D} = \{x^n, n = 1, \ldots, N\}$ in unsupervised learning we aim to to 'learn' a plausible compact description of the data. An objective is used to quantify the accuracy of the description.

In unsupervised learning there is no special 'prediction' variable. From a probabilistic perspective we are interested in modelling the distribution $p(x)$. The likelihood of the data under the i.i.d. assumption, for example, would be one objective measure of the accuracy of the description.

Example 58. A supermarket chain wishes to discover how many different basic consumer buying behaviours there are based on a large database of supermarket checkout data. Items brought by a customer on a visit to a checkout are represented by a (very sparse) 10,000 dimensional vector $\mathbf{x}$ which contains a 1 in the i^{th} element if the customer bought product i and 0 otherwise. Based on 10 million such checkout vectors from stores across the country, $\mathcal{D} = \{\mathbf{x}^n, n = 1, \ldots, 10^7\}$ the supermarket chain wishes to discover patterns of buying behaviour.

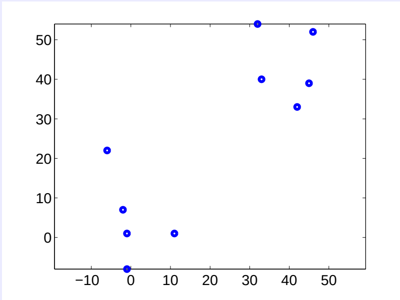
In the table each column represents the buying patterns of a customer (7 customer records and just the first 6 of the 10,000 products are shown). A 1 indicates that the customer bought that item. We wish to find common patterns in the data, such as if someone buys coffee they are also likely to buy milk.

coffee	1	0	0	1	0	0	0	··
tea	0	0	1	0	0	0	0	··
milk	1	0	1	1	0	1	1	··
beer	0	0	0	1	1	0	1	··
diapers	0	0	1	0	1	0	1	··
aspirin	0	1	0	0	1	0	1	··

Example 59 (Clustering)**.**

The table on the right represents a collection of unlabelled two-dimensional points. We can visualise this data by plotting it in 2 dimensions.

x_1	-2	-6	-1	11	-1	46	33	42	32	45
x_2	7	22	1	1	-8	52	40	33	54	39



By simply eye-balling the data, we can see that there are two apparent clusters here, one centred around (0,5) and the other around (35,45). A reasonable model to describe this data might therefore be to describe it as two clusters, centred at (0,0) and (35,35), each with a standard deviation of around 10.

13.1.3 Anomaly detection

A baby processes a mass of initially confusing sensory data. After a while the baby begins to understand her environment in the sense that novel sensory data from the same environment is familiar or expected. When a strange face presents itself, the baby recognises that this is not familiar and may be upset. The baby has learned a representation of the familiar and can distinguish the expected from the unexpected; this is an example of unsupervised learning. Models that can detect irregular events are used in *plant monitoring* and require a model of normality which will in most cases be based on unlabelled data.

13.1.4 Online (sequential) learning

In the above situations, we assumed that the data $\mathcal{D}$ was given beforehand. In *online learning* data arrives sequentially and we want to continually update our model as new data becomes available. Online learning may occur in either a supervised or unsupervised context.

13.1.5 Interacting with the environment

In many real-world situations, an agent is able to interact in some manner with its environment.

Query (Active) Learning Here the agent has the ability to request data from the environment. For example, a predictor might recognise that it is less confidently able to predict in certain regions of the space x and therefore requests more training data in this region. Active Learning can also be considered in an unsupervised context in which the agent might request information in regions where $p(x)$ looks uninformative or 'flat'.

Reinforcement Learning One might term this also 'survival learning'. One has in mind scenarios such as encountered in real-life where an organism needs to learn the best actions to take in its environment in order to survive as long as possible. In each situation in which the agent finds itself it needs to take an action. Some actions may eventually be beneficial (lead to food for example), whilst others may be disastrous (lead to being eaten for example). Based on accumulated experience, the agent needs to learn which action to take in a given situation in order to obtain a desired long term goal. Essentially actions that lead to long term rewards need to reinforced. Reinforcement learning has connections with control theory, Markov decision processes and game theory. Whilst we discussed MDPs and briefly mentioned how an environment can be learned based on delayed rewards in section(7.8.3), we will not discuss this topic further in this book.

13.1.6 Semi-Supervised Learning

In machine learning, a common scenario is to have a small amount of labelled and a large amount of unlabelled data. For example, it may be that we have access to many images of faces; however, only a small number of them may have been labelled as instances of known faces. In semi-supervised learning, one tries to use the unlabelled data to make a better classifier than that based on the labelled data alone.

13.2 Supervised Learning

Supervised and unsupervised learning are mature fields with a wide range of practical tools and associated theoretical analyses. Our aim here is to give a brief introduction to the issues and 'philosophies' behind the approaches. We focus here mainly on supervised learning and classification in particular.

13.2.1 Utility and Loss

To more fully specify a supervised problem we need to be clear what 'cost' is involved in making a correct or incorrect prediction. In a two class problem $\text{dom}(c) = \{1, 2\}$, we assume here that everything we know about the environment is contained in a model $p(x, c)$. Given a new input x^*, the optimal prediction also depends on how costly making an error is. This can be quantified using a loss function (or conversely a utility). In forming a *decision function* $c(x^*)$ that will produce a class label for the new input x^*, we don't know the true class, only our presumed distribution $p(c|x^*)$. The expected utility for the decision function is

$$U(c(x^*)) = \sum_{c^{true}} U(c^{true}, c(x^*))p(c^{true}|x^*) \tag{13.2.1}$$

and the optimal decision is that which maximises the expected utility.

Zero-one Loss

A 'count the correct predictions' measure of prediction performance is based on the 'zero-one' utility (or conversely the *zero-one loss*):

$$U(c^{true}, c^*) = \begin{cases} 1 \text{ if } c^* = c^{true} \\ 0 \text{ if } c^* \neq c^{true} \end{cases} \tag{13.2.2}$$

For the two class case, we then have

$$U(c(x^*)) = \begin{cases} p(c^{true} = 1|x^*) \text{ for } c(x^*) = 1 \\ p(c^{true} = 2|x^*) \text{ for } c(x^*) = 2 \end{cases} \tag{13.2.3}$$

Hence, in order to have the highest expected utility, the decision function $c(x^*)$ should correspond to selecting the highest class probability $p(c|x^*)$:

$$c(x^*) = \begin{cases} 1 & \text{if} \quad p(c = 1|x^*) \geq 0.5 \\ 2 & \text{if} \quad p(c = 2|x^*) \geq 0.5 \end{cases} \tag{13.2.4}$$

In the case of a tie, either class is selected at random with equal probability.

General Loss functions

In general, for a two-class problem, we have

$$U(c(x^*)) = \begin{cases} U(c^{true}=1, c^*=1)p(c^{true}=1|x^*) + U(c^{true}=2, c^*=1)p(c^{true}=2|x^*) & \text{for } c(x^*)=1 \\ U(c^{true}=1, c^*=2)p(c^{true}=1|x^*) + U(c^{true}=2, c^*=2)p(c^{true}=2|x^*) & \text{for } c(x^*)=2 \end{cases} \quad (13.2.5)$$

and the optimal decision function $c(x^*)$ chooses that class with highest expected utility.

One can readily generalise this to multiple-class situations using a *utility matrix* with elements

$$U_{i,j} = U(c^{true}=i, c^{pred}=j) \quad (13.2.6)$$

where the i, j element of the matrix contains the utility of predicting class j when the true class is i. Conversely one could think of a loss-matrix with entries $L_{ij} = -U_{ij}$. The expected loss with respect to $p(c|x)$ is then termed the *risk*.

In some applications the utility matrix is highly non-symmetric. Consider a medical scenario in which we are asked to predict whether or not the patient has cancer $\text{dom}(c) = \{\text{cancer}, \text{benign}\}$. If the true class is cancer yet we predict benign, this could have terrible consequences for the patient. On the other hand, if the class is benign yet we predict cancer, this may be less disastrous for the patient. Such asymmetric utilities can bias the predictions in favour of conservative decisions – in the cancer case, we would be more inclined to decide the sample is cancerous than benign, even if the predictive probability of the two classes is equal.

13.2.2 What's the catch?

In solving for the optimal decision function $c(x^*)$ above we are assuming that the model $p(c|x)$ is 'correct'. The catch is therefore that in practice :

- We typically *don't* know the correct model underlying the data – all we have is a dataset of examples $\mathcal{D} = \{(x^n, y^n), n = 1, \ldots, N\}$ and our domain knowledge.
- We want our method to perform well not just on a specifically chosen x^*, but any new input that could come along – that is we want it to *generalise* to novel inputs. This means we also need a model for $p(x)$ in order to measure what the expected performance of our decision function would be. Hence we require knowledge of the joint distribution $p(c, x) = p(c|x)p(x)$.

We therefore need to form a distribution $p(x, c|\mathcal{D})$ which should ideally be close to the true but unknown joint data distribution. Communities of researchers in Machine Learning form around different strategies to address the lack of knowledge about the true $p(c, x)$.

13.2.3 Using the empirical distribution

A direct approach to not knowing the correct model $p^{true}(c, x)$ is to replace it with the *empirical distribution*

$$p(x, c|\mathcal{D}) = \frac{1}{N}\sum_{n=1}^{N} \delta(x, x^n)\, \delta(c, c^n) \quad (13.2.7)$$

That is, we assume that the underlying distribution is approximated by placing equal mass on each of the points (x^n, c^n) in the dataset. Using this gives the empirical utility

$$\langle U(c, c(x))\rangle_{p(c,x|\mathcal{D})} = \frac{1}{N}\sum_{n} U(c^n, c(x^n)) \quad (13.2.8)$$

or conversely the *empirical risk*

$$R = \frac{1}{N}\sum_{n} L(c^n, c(x^n)) \quad (13.2.9)$$

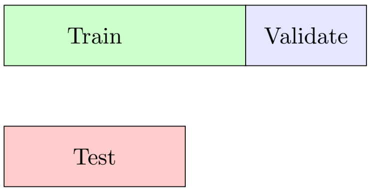


Figure 13.2: Models can be trained using the train data based on different regularisation parameters. The optimal regularisation parameter is determined by the empirical performance on the validation data. An independent measure of the generalisation performance is obtained by using a separate test set.

Assuming the loss is minimal when the correct class is predicted, the optimal decision $c(x)$ for any input in the train set is trivially given by $c(x^n) = c^n$. However, for any new x^* not contained in $\mathcal{D}$ then $c(x^*)$ is undefined. In order to define the class of a novel input, one may use a parametric function

$$c(x) = f(x|\theta) \tag{13.2.10}$$

For example for a two class problem $\mathrm{dom}(c) = \{1, 2\}$, a linear decision function is given by

$$f(\mathbf{x}|\boldsymbol{\theta}) = \begin{cases} 1 \text{ if } \boldsymbol{\theta}^\mathsf{T}\mathbf{x} + \theta_0 \geq 0 \\ 2 \text{ if } \boldsymbol{\theta}^\mathsf{T}\mathbf{x} + \theta_0 < 0 \end{cases} \tag{13.2.11}$$

If the vector input $\mathbf{x}$ is on the positive side of a hyperplane defined by the vector $\boldsymbol{\theta}$ and bias θ_0, we assign it to class 1, otherwise to class 2. (We return to the geometric interpretation of this in chapter(17)). The empirical risk then becomes a function of the parameters $\theta = \{\boldsymbol{\theta}, \theta_0\}$,

$$R(\theta|\mathcal{D}) = \frac{1}{N}\sum_n L(c^n, f(x^n|\theta)) \tag{13.2.12}$$

The optimal parameters θ are given by minimising the empirical risk with respect to θ,

$$\theta_{opt} = \underset{\theta}{\mathrm{argmin}}\ R(\theta|\mathcal{D}) \tag{13.2.13}$$

The decision for a new datapoint x^* is then given by $f(x^*|\theta_{opt})$.

In this *empirical risk minimisation* approach, as we make the decision function $f(x|\theta)$ more complex, the empirical risk goes down. If we make $f(x|\theta)$ too complex we will have no confidence $f(x|\theta)$ will perform well on a novel input x^*. To constrain the complexity of $f(x|\theta)$ we may minimise the *penalised empirical risk*

$$R'(\theta|\mathcal{D}) = R(\theta|\mathcal{D}) + \lambda P(\theta) \tag{13.2.14}$$

For the linear decision function above, it is reasonable to penalise wildly changing classifications in the sense that if we change the input x by only a small amount we expect (on average) minimal change in the class label. The squared difference in $\theta^\mathsf{T}x + \theta_0$ for two inputs x_1 and x_2 is $\left(\theta^\mathsf{T}\Delta x\right)^2$ where $\Delta x \equiv x_2 - x_1$. By constraining the length of θ to be small we would then limit the ability of the classifier to change class for only a small change in input space[2]. This motivates a penalised risk of the form

$$R'(\theta, \theta_0|\mathcal{D}) = R(\theta, \theta_0|\mathcal{D}) + \lambda\theta^\mathsf{T}\theta \tag{13.2.15}$$

where λ is a *regularising constant*,. We subsequently minimise this penalised empirical risk with respect to θ, θ_0. We discuss how to find an appropriate setting for the regularisation constant λ below.

Validation

In penalised empirical risk minimisation we need to set the regularisation parameter λ. This can be achieved by evaluating the performance of the learned classifier $f(x|\theta)$ on validation data $\mathcal{D}_{validate}$ for several different λ values, and choosing the one with the best performance. It's important that the validation data is not the data on which the model was trained since we know that the optimal setting for λ in that case is zero, and again we will have no confidence in the generalisation ability.

[2]Assuming the distance between two datapoints is distributed according to an isotropic multivariate Gaussian with zero mean and covariance $\sigma^2\mathbf{I}$, the average squared change is $\left\langle \left(\boldsymbol{\theta}^\mathsf{T}\Delta\mathbf{x}\right)^2\right\rangle = \sigma^2\boldsymbol{\theta}^\mathsf{T}\boldsymbol{\theta}$, motivating the choice of the Euclidean squared length of the parameter $\boldsymbol{\theta}$ as the penalty term.

Algorithm 12 Setting regularisation parameters using cross-validation.

Choose a set of regularisation parameters $\lambda_1, \ldots, \lambda_A$
Choose a set of training and validation set splits $\left\{\mathcal{D}^i_{train}, \mathcal{D}^i_{validate}\right\}, i = 1, \ldots, I$
for $a = 1$ to A **do**
  for $i = 1$ to I **do**
    $\theta^i_a = \underset{\theta}{\text{argmin}} \left[R(\theta|\mathcal{D}^i_{train}) + \lambda_a P(\theta)\right]$
  end for
  $L(\lambda_a) = \frac{1}{I}\sum_{i=1}^{I} R(\theta^i_a|\mathcal{D}^i_{validate})$
end for
$\lambda_{opt} = \underset{\lambda_a}{\text{argmin}}\ L(\lambda_a)$

Given an original dataset $\mathcal{D}$ we split this into disjoint parts, $\mathcal{D}_{train}, \mathcal{D}_{validate}$, where the size of the validation set is usually chosen to be smaller than the train set. For each parameter λ_a one then finds the minimal empirical risk parameter θ_a. This splitting procedure is repeated, each time producing a separate training $\mathcal{D}^i_{train}$ and validation $\mathcal{D}^i_{validation}$ set, along with an optimal penalised empirical risk parameter θ^i_a and associated (unregularised) validation performance $R(\theta^i_a|\mathcal{D}^i_{validate})$. The performance of regularisation parameter λ_a is taken as the average of the validation performances over i. The best regularisation parameter is then given as that with the minimal average validation error, see algorithm(12) and fig(13.2). Using the optimal regularisation parameter λ, many practitioners retrain θ on the basis of the whole dataset $\mathcal{D}$.

In cross-validation a dataset is broken into training and validation sets. In K-fold cross validation the data $\mathcal{D}$ is split into K equal sized disjoint parts $\mathcal{D}_1, \ldots, \mathcal{D}_K$. Then $\mathcal{D}^i_{validate} = \mathcal{D}_i$ and $\mathcal{D}^i_{train} = \mathcal{D}\backslash\mathcal{D}^i_{validate}$. This gives a total of K different training-validation sets over which performance is averaged, see fig(13.3). In practice 10-fold cross validation is popular, as is leave-one-out cross validation in which the validation sets consist of only a single example.

A heuristic justification for penalty terms

Our aim is to make a method that generalises well on novel examples. Leaving the utility approach and classification aside for a moment, we consider the task of making a prediction distribution $\tilde{p}(y|x, \theta)$ of an output y given an input x. We additionally assume that the training data x^n, y^n are generated from some unknown distribution $p(x, y)$. A simple generic measure of the accuracy of prediction is given by the overlap between the predictive distribution and the true distribution:

$$A(\theta) = \sum_x p(x) \sum_y p(y|x)\tilde{p}(y|x, \theta) = \sum_{x,y} p(x, y)\tilde{p}(y|x, \theta) \tag{13.2.16}$$

Our aim is to set θ such that the accuracy is maximal. However, we don't have access to $p(x, y)$, so we cannot directly achieve this. A reasonable generic assumption is that the unknown true distribution can

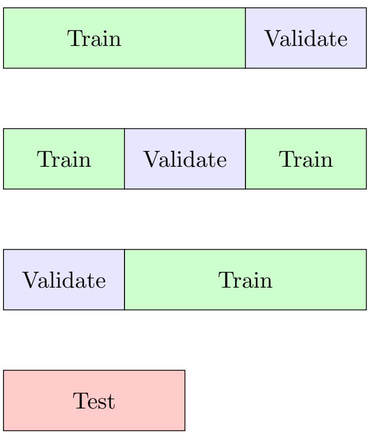


Figure 13.3: In cross-validation the original dataset is split into several train-validation sets. Depicted is 3-fold cross-validation. For a range of regularisation parameters, the optimal regularisation parameter is found based on the empirical validation performance averaged across the different splits.

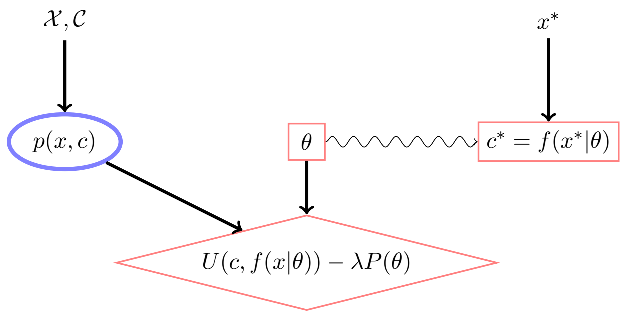


Figure 13.4: Empirical risk approach. Given the dataset $\mathcal{X}, \mathcal{C}$, a model of the data $p(x, c)$ is made, usually using the empirical distribution. For a classifier $f(x|\theta)$, the parameter θ is learned by maximising the penalised empirical utility (or minimising empirical risk) with respect to θ. The penalty parameter λ is set by validation. A novel input x^* is then assigned to class $f(x^*|\theta)$, given this optimal θ.

be approximated as

$$p(x, y) \approx \alpha q(x, y) + (1 - \alpha)\epsilon(x, y) \tag{13.2.17}$$

where $q(x, y)$ is the empirical distribution

$$q(x, y) = \frac{1}{N}\sum_{n=1}^{N} \mathbb{I}\left[x = x^n, y = y^n\right] \tag{13.2.18}$$

and $\epsilon(x, y)$ is a 'noise' distribution that accounts for our lack of knowledge about the true $p(x, y)$. The parameter $0 \leq \alpha \leq 1$ controls how much we believe the empirical distribution matches the true data generating distribution $p(x, y)$. Under this approximation

$$\frac{1}{\alpha}A(\theta) \approx \underbrace{\frac{1}{N}\sum_{n=1}^{N} \tilde{p}(y^n|x^n, \theta)}_{\text{training accuracy}} + \frac{1-\alpha}{\alpha}\underbrace{\sum_{x,y} \epsilon(x, y)\tilde{p}(y|x, \theta)}_{\text{penalty}} \tag{13.2.19}$$

The second term

$$\sum_{x,y} \epsilon(x, y)\tilde{p}(y|x, \theta) \tag{13.2.20}$$

measures the overlap between the predictions and a 'random' distribution. Since little is assumed about $\epsilon(x, y)$, to have a high overlap between the predictions and this 'random' distribution we need the predictions to look essentially random. This motivates the idea that one should set prediction parameters θ based both on maximising training accuracy whilst also minimising the commitment of the predictor, see also exercise(155). One can carry through a similar reasoning for more general loss functions.

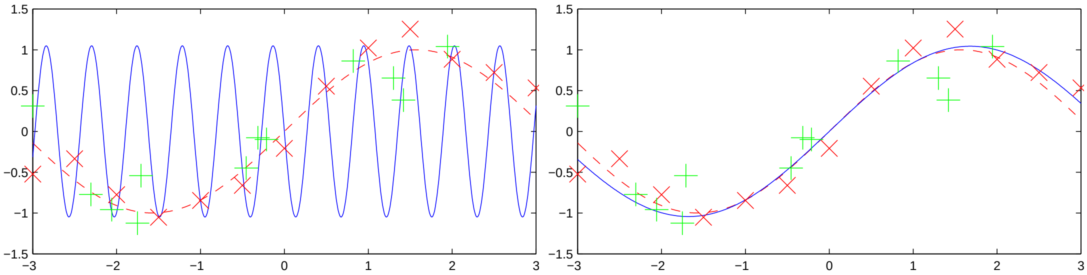
Benefits of the empirical risk approach

For a utility $U(c^{true}, c^{pred})$ and penalty $P(\theta)$, the empirical risk approach is summarised in fig(13.4).

- In the limit of a large amount of training data the empirical distribution will tend towards the correct distribution.
- The discriminant function is chosen on the basis of minimal risk, which is the quantity we are ultimately interested in.
- The procedure is conceptually straightforward.

Drawbacks of the empirical risk approach

- It seems extreme to assume that the data follows the empirical distribution, particularly for small amounts of training data. To generalise well, we need to make sensible assumptions as to $p(x)$ – that is the distribution for all x that could arise.
- If the utility (or loss) function changes, the discriminant function needs to be retrained.



(a) (b)

Figure 13.5: **(a)**: The unregularised fit ($\lambda = 0$) to training given by $\times$. Whilst the training data is well fitted, the error on the validation examples, $+$ is high. **(b)**: The regularised fit ($\lambda = 0.5$). Whilst the train error is high, the validation error (which is all important) is low. The true function which generated this noisy data is the dashed line; the function learned from the data is given by the solid line.

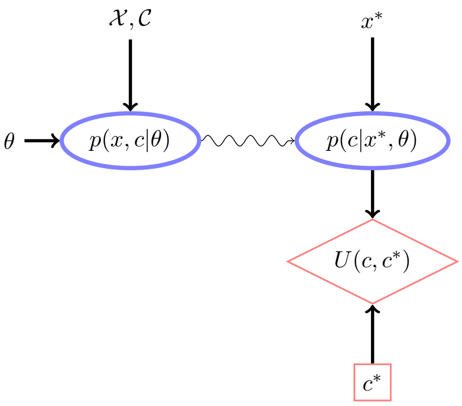


Figure 13.6: Bayesian decision approach. A model $p(x, c|\theta)$ is fitted to the data. After leaning, this model is used to compute $p(c|x, \theta)$. For a novel x^*, we then find the distribution of the assumed 'truth', $p(c|x^*, \theta)$. The prediction (decision) is then given by that c^* which maximises the expected utility $\langle U(c, c^*)\rangle_{p(c|x^*,\theta)}$.

- Some problems require an estimate of the confidence of the prediction. Whilst there may be heuristic ways to evaluating confidence in the prediction, this is not inherent in the framework.

- When there are multiple penalty parameters, performing cross validation in a discretised grid of the parameters becomes infeasible.

- It seems a shame to discard all those trained models in cross-validation – can't they be combined in some manner and used to make a better predictor?

Example 60 (Finding a good regularisation parameter)**.** In fig(13.5), we fit the function $a \sin(wx)$ to data, learning the parameters a and w. The unregularised solution fig(13.5a) badly overfits the data, and has a high validation error. To encourage a smoother solution, a regularisation term $E_{reg} = w^2$ is used. The validation error based on several different values of the regularisation parameter λ was computed, finding that $\lambda = 0.5$ gave a low validation error. The resulting fit to novel data, fig(13.5b) is reasonable.

13.2.4 Bayesian decision approach

An alternative to using the empirical distribution is to fit a model $p(c, x|\theta)$ to the train data $\mathcal{D}$. Given this model, the decision function $c(x)$ is automatically determined from the maximal expected utility (or minimal risk), with respect to this model, as in equation (13.2.5), in which the unknown $p(c^{true}|x)$ is replaced with $p(c|x, \theta)$. This approach therefore divorces learning the parameters θ of $p(c, x|\theta)$ from the utility (or loss).

Benefits of the Bayesian decision approach

- This is a conceptually 'clean' approach, in which one tries ones best to model the environment, independent of the subsequent decision process. In this case learning the environment is separated from the ultimate affect this will have on the expected utility.
- The ultimate decision c^* for a novel input x^* can be a highly complex function of x^* due to the maximisation operation.

Drawbacks of the Bayesian decision approach

- If the environment model $p(c, x|\theta)$ is poor, the prediction c^* could be highly inaccurate since modelling the environment is divorced from prediction.
- To avoid fully divorcing the learning of the model $p(c, x|\theta)$ from its effect on decisions, in practice one often includes regularisation terms in the environment model $p(c, x|\theta)$ which are set by validation based on an empirical utility.

There are two main approaches to fitting $p(c, x|\theta)$ to data $\mathcal{D}$. We could parameterise the joint distribution using

$$p(c, x|\theta) = p(c|x, \theta_{c|x})p(x|\theta_x) \qquad \textit{discriminative approach} \tag{13.2.21}$$

or

$$p(c, x|\theta) = p(x|c, \theta_{x|c})p(c|\theta_c) \qquad \textit{generative approach} \tag{13.2.22}$$

We'll consider these two approaches below in the context of trying to make a system that can distinguish between a male and female face. We have a database of face images in which each image is represented as a real-valued vector $\mathbf{x}^n, n = 1, \ldots, N$), along with a label $c^n \in \{0, 1\}$ stating if the image is male or female.

Generative approach $p(\mathbf{x}, c|\theta) = p(\mathbf{x}|c, \theta_{x|c})p(c|\theta_c)$

For simplicity we use Maximum Likelihood training for the parameters θ. Assuming the data $\mathcal{D}$ is i.i.d., we have a log likelihood

$$\log p(\mathcal{D}|\theta) = \sum_n \log p(\mathbf{x}^n|c^n, \theta_{x|c}) + \sum_n \log p(c^n|\theta_c) \tag{13.2.23}$$

As we see the dependence on $\theta_{x|c}$ occurs only in the first term, and θ_c only occurs in the second. This means that learning the optimal parameters is equivalent to isolating the data for the male-class and fitting a model $p(\mathbf{x}|c = \mathsf{male}, \theta_{x|\mathsf{male}})$. We similarly isolate the female data and fit a separate model $p(\mathbf{x}|c = \mathsf{female}, \theta_{x|\mathsf{female}})$. The class distribution $p(c|\theta_c)$ can be easily set according to the ratio of males/females in the set of training data.

To make a classification of a new image $\mathbf{x}^*$ as either male or female, we may use

$$p(c = \mathsf{male}|\mathbf{x}^*) = \frac{p(\mathbf{x}^*, c = \mathsf{male}|\theta_{x|\mathsf{male}})}{p(\mathbf{x}^*, c = \mathsf{male}|\theta_{x|\mathsf{male}}) + p(\mathbf{x}^*, c = \mathsf{female}|\theta_{x|\mathsf{female}})} \tag{13.2.24}$$

Based on zero-one loss, if this probability is greater than 0.5 we classify $\mathbf{x}^*$ as male, otherwise female. More generally, we may use this probability as part of a decision process, as in equation (13.2.5).

Advantages Prior information about the structure of the data is often most naturally specified through a generative model $p(x|c)$. For example, for male faces, we would expect to see heavier eyebrows, a squarer jaw, *etc.*

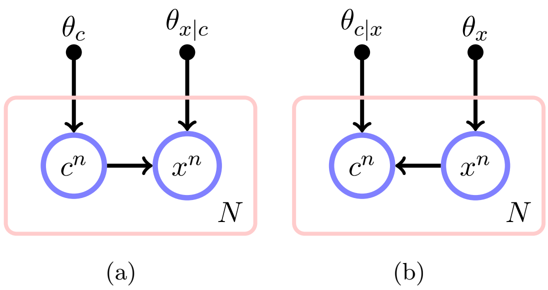


Figure 13.7: Two generic strategies for probabilistic classification. **(a)**: Class dependent generative model of x. After learning parameters, classification is obtained by making x evidential and inferring $p(c|x)$. **(b)**: A discriminative classification method $p(c|x)$.

Disadvantages The generative approach does not directly target the classification model $p(c|x)$ since the goal of generative training is rather to model $p(x|c)$. If the data x is complex, finding a suitable generative data model $p(x|c)$ is a difficult task. On the other hand it might be that making a model of $p(c|x)$ is simpler, particularly if the decision boundary between the classes has a simple form, even if the data distribution of each class is complex, see fig(13.8). Furthermore, since each generative model is separately trained for each class, there is no competition amongst the models to explain the data.

Discriminative approach $p(c, \mathbf{x}) = p(c|\mathbf{x}, \theta_{c|x})p(\mathbf{x}|\theta_x)$

Assuming i.i.d. data, the log likelihood is

$$\sum_n \log p(\mathcal{D}|\theta) = \sum_n \log p(c^n|\mathbf{x}^n, \theta_{c|x}) + \sum_n \log p(\mathbf{x}^n|\theta_x) \tag{13.2.25}$$

The parameters are isolated in the two terms so that Maximum Likelihood training is equivalent to finding the parameters of $\theta_{c|x}$ that will best predict the class c for a given training input x. The parameters θ_x for modelling the data occur separately in the second term above, and setting them can therefore be treated as a separate unsupervised learning problem. This approach therefore isolates modelling the *decision boundary* from modelling the input distribution, see fig(13.8).

Classification of a new point $\mathbf{x}^*$ is based on

$$p(c|\mathbf{x}, \theta^{opt}_{c|x}) \tag{13.2.26}$$

As for the generative case, this approach still learns a joint distribution $p(c, x) = p(c|x)p(x)$ which can be used as part of a decision process if required.

Advantages The discriminative approach directly addresses making an accurate classifier based on $p(c|x)$, modelling the decision boundary, as opposed to the class conditional data distribution in the generative approach. Whilst the data from each class may be distributed in a complex way, it could be that the decision boundary between them is relatively easy to model.

Disadvantages Discriminative approaches are usually trained as 'black-box' classifiers, with little prior knowledge built in as to how data for a given class might look. Domain knowledge is often more easily expressed using the generative framework.

Hybrid generative-discriminative approaches

One could use a generative description, $p(x|c)$, building in prior information, and use this to form a joint distribution $p(x, c)$, from which a discriminative model $p(c|x)$ may be formed, using Bayes' rule. Specifically, we can use

$$p(c|x, \theta) = \frac{p(x|c, \theta_{x|c})p(c|\theta_c)}{\sum_c p(x|c, \theta_{x|c})p(c|\theta_c)} \tag{13.2.27}$$

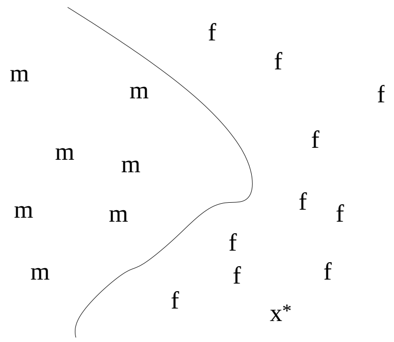


Figure 13.8: Each point represents a high dimensional vector with an associated class label, either male or female. The point $\mathbf{x}^*$ is a new point for which we would like to predict whether this should be male or female. In the generative approach, a male model $p(\mathbf{x}|\mathsf{male})$ generates data similar to the 'm' points. Similarly, the female model $p(\mathbf{x}|\mathsf{female})$ generates points that are similar to the 'f' points above. We then use Bayes' rule to calculate the probability $p(\mathsf{male}|\mathbf{x}^*)$ using the two fitted models, as given in the text. In the discriminative approach, we directly make a model of $p(\mathsf{male}|\mathbf{x}^*)$, which cares less about how the points 'm' or 'f' are distributed, but more about describing the boundary which can separate the two classes, as given by the line.

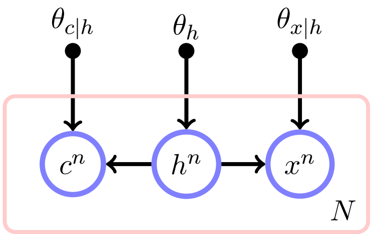


Figure 13.9: A strategy for semi-supervised learning. When c^n is missing, the term $p(c^n|h^n)$ is absent. The large amount of training data helps the model learn a good lower dimension/compressed representation h of the data x. Fitting then a classification model $p(c|h)$ using this lower dimensional representation may be much easier than fitting a model directly from the complex data to the class, $p(c|x)$.

and use a separate model for $p(x|\theta_x)$. Subsequently the parameters $\theta = \left(\theta_{x|c}, \theta_c\right)$, for this hybrid model can be found by maximising the probability of being in the correct class. This approach would appear to leverage the advantages of both the discriminative and generative frameworks since we can more readily incorporate domain knowledge in the generative model $p(x|c, \theta_{x|c})$ yet train this in a discriminative way. This approach is rarely taken in practice since the resulting functional form of the likelihood depends in a complex manner on the parameters. In this case no separation occurs (as was previously the case for the generative and discriminative approaches).

13.2.5 Learning lower-dimensional representations in semi-supervised learning

One way to exploit a large amount of unlabelled training data to improve classification modelling is to try to find a lower dimensional representation h of the data x. Based on this, the mapping from h to x may be rather simpler to learn than a mapping from x to c directly. To do so we can form the likelihood using, see fig(13.9),

$$p(\mathcal{C}, \mathcal{X}, \mathcal{H}|\theta) = \prod_n \left\{p(c^n|h^n, \theta_{c|h})\right\}^{\mathbb{I}[c^n \neq \emptyset]} p(x^n|h^n, \theta_{x|h}) p(h|\theta_h) \tag{13.2.28}$$

and then set any parameters for example by using Maximum Likelihood

$$\theta^{opt} = \underset{\theta}{\text{argmax}} \sum_{\mathcal{H}} p(\mathcal{C}, \mathcal{X}, \mathcal{H}|\theta) \tag{13.2.29}$$

13.2.6 Features and preprocessing

It is often the case that when attempting to make a predictive model, transforming the raw input x into a form that more directly captures the relevant label information can greatly improve performance. For example, in the male-female classification case, it might be that building a classifier directly in terms of the elements of the face vector $\mathbf{x}$ is difficult. However, using 'features' which contain geometric information such as the distance between eyes, width of mouth, *etc.* may make finding a classifier easier. In practice data is often preprocessed to remove noise, centre an image *etc.*

13.3 Bayes versus Empirical decisions

The empirical risk and Bayesian approaches are at the extremes of the philosophical spectrum. In the empirical risk approach one makes a seemingly over-simplistic data generating assumption. However decision function parameters are set based on the task of making decisions. On the other hand, the Bayesian approach attempts to learn $p(c,x)$ without regard to its ultimate use as part of a larger decision process. What 'objective' criterion can we use to learn $p(c,x)$, particularly if we are only interested in classification with a low test-risk? The following example is intended to recapitulate the two generic Bayes and empirical risk approaches we've been considering.

Example 61 (The two generic decision strategies)**.** Consider a situation in which, based on patient information x we need to take a decision d as whether or not to operate. The utility of operating $u(d,c)$ depends on whether or not the patient has cancer. For example

$$\begin{array}{ll} u(\mathsf{operate},\mathsf{cancer}) = 100 & u(\mathsf{operate},\mathsf{benign}) = 30 \\ u(\mathsf{don't\ operate},\mathsf{cancer}) = 0 & u(\mathsf{don't\ operate},\mathsf{benign}) = 70 \end{array} \tag{13.3.1}$$

We have independent true assessments of whether or not a patient had cancer, giving rise to a set of historical records $\mathcal{D} = \{(x^n, c^n), n = 1, \ldots, N\}$. Faced with a new patient x, we need to make a decision whether or not to operate.

In the Bayesian decision approach one would first make a model $p(c|x,\mathcal{D})$ (for example Logistic regression). Using this model the decision is given by that which maximises the expected utility

$$d = \underset{d}{\operatorname{argmax}}\ p(\mathsf{cancer}|x,\mathcal{D})u(d,\mathsf{cancer}) + p(\mathsf{benign}|x,\mathcal{D})u(d,\mathsf{benign}) \tag{13.3.2}$$

In this approach learning the model $p(c|x,\mathcal{D})$ is divorced from the ultimate use of the model in the decision making process. An advantage of this approach is that, from the viewpoint of expected utility, it is optimal – provided the model $p(c|x,\mathcal{D})$ is 'correct'. Unfortunately, this is rarely the case. Given the limited model resources it might make sense to focus on ensuring the prediction of cancer is correct since this has a more significant effect on the utility. However, formally, this is not possible in this framework.

The alternative empirical utility approach recognises that the task can be stated as to translate patient information x into an operation decision d. To do so one could parameterise this as $d(x) = f(x|\theta)$ and then learn θ under maximising the empirical utility

$$u(\theta) = \sum_n u(f(x^n|\theta), c^n) \tag{13.3.3}$$

For example, if $\mathbf{x}$ is a vector representing the patient information and $\boldsymbol{\theta}$ the parameter, we might use a linear decision function such as

$$f(\mathbf{x}|\boldsymbol{\theta}) = \begin{cases} \boldsymbol{\theta}^\mathsf{T}\mathbf{x} \geq 0 & d = \mathsf{operate} \\ \boldsymbol{\theta}^\mathsf{T}\mathbf{x} < 0 & d = \mathsf{don't\ operate} \end{cases} \tag{13.3.4}$$

The advantage of this approach is that the parameters of the decision are directly related to the utility of making the decision. A disadvantage is that we cannot easily incorporate domain knowledge into the decision function. It may be that we have a good model of $p(c|x)$ and would wish to make use of this.

Both approaches are heavily used in practice and which is to be preferred depends very much on the problem. Whilst the Bayesian approach appears formally optimal, it is prone to model mis-specification. A pragmatic alternative Bayesian approach is to fit a parameterised distribution $p(c,x|\lambda)$ to the data $\mathcal{D}$, where λ penalises 'complexity' of the fitted distribution, setting λ using validation on the risk. This has the potential advantage of allowing one to incorporate sensible prior information about $p(c,x)$ whilst assessing competing models in the light of their actual predictive risk. Similarly, for the empirical risk approach, one can modify the extreme empirical distribution assumption by using a more plausible model $p(x,c)$ of the data.

13.4 Representing Data

The numeric encoding of data can have a significant effect on performance and an understanding of the options for representing data is therefore of considerable importance.

13.4.1 Categorical

For categorical (or nominal) data, the observed value belongs to one of a number of classes, with no intrinsic ordering of the classes. An example of a categorical variable would be the description of the type of job that someone does, *e.g.* healthcare, education, financial services, transport, homeworker, unemployed, engineering *etc.*

One way to transform this data into numerical values would be to use 1-of-m encoding. Here's an example: There are 4 kinds of jobs: soldier, sailor, tinker, spy. A soldier is represented as (1,0,0,0), a sailer as (0,1,0,0), a tinker as (0,0,1,0) and a spy as (0,0,0,1). In this encoding the distance between the vectors representing two different professions is constant. It is clear that 1-of-m encoding induces dependencies in the profession attributes since if one of the profession attributes is 1, the others must be zero.

13.4.2 Ordinal

An ordinal variable consists of categories with an ordering or ranking of the categories, *e.g.* cold, cool, warm, hot. In this case, to preserve the ordering we could perhaps use -1 for cold, 0 for cool, +1 for warm and +2 for hot. This choice is somewhat arbitrary, and one should bear in mind that results will generally be dependent on the numerical coding used.

13.4.3 Numerical

Numerical data takes on values that are real numbers, *e.g.* a temperature measured by a thermometer, or the salary that someone earns.

13.5 Bayesian Hypothesis testing for Outcome Analysis

How can we assess if two classifiers are performing differently? For techniques which are based on Bayesian classifiers $p(c, \theta|\mathcal{D}, M)$ there will always be, in principle, a direct way to estimate the suitability of the model M by computing $p(M|\mathcal{D})$. We consider here the less fortunate situation where the only information presumed available is the test performance of the two classifiers.

To outline the basic issue, let's consider two classifiers A and B which predict the class of 55 test examples. Classifier A makes 20 errors, and 35 correct classifications, whereas classifier B makes 23 errors and 32 correct classifications. Is classifier A better than classifier B? Our lack of confidence in pronouncing that A is better than B results from the small number of test examples. On the other hand if classifier A makes 200 errors and 350 correct classifications, whilst classifier B makes 230 errors and 320 correct classifications, intuitively, we would be more confident that classifier A is better than classifier B.

Perhaps the most practically relevant question from a Machine Learning perspective is the probability that classifier A outperforms classifier B, given the available test information. Whilst this question can be addressed using a Bayesian procedure, section(13.5.5), we first focus on a simpler question, namely whether classifier A and B are the same[16].

13.5.1 Outcome analysis

The treatment in this section refers to outcomes and quantifies if data is likely to come from the same multinomial distribution. In the main we will apply this to assessing if two classifiers are essentially performing the same, although one should bear in mind that the method applies more generally to assessing if outcomes are likely to have been generated from the same or different underlying processes.

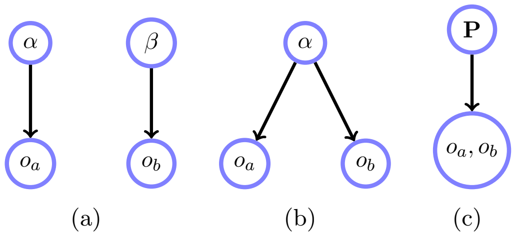


Figure 13.10: **(a)**: H_{diff} : Corresponds to the outcomes for the two classifiers being independently generated. **(b)**: H_{same}: both outcomes are generated from the same distribution. **(c)**: H_{dep} : the outcomes are dependent ('correlated').

Consider a situation where two classifiers A and B have been tested on some data, so that we have, for each example in the test set, an outcome pair

$$(o_a(n), o_b(n)), n = 1, \ldots, N \tag{13.5.1}$$

where N is the number of test data points, and $o_a \in \{1, \ldots, Q\}$ (and similarly for o_b). That is, there are Q possible types of outcomes that can occur.

For example, for binary classification we will typically have the four cases

$$\text{dom}(o) = \{\mathsf{TruePositive}, \mathsf{FalsePositive}, \mathsf{TrueNegative}, \mathsf{FalseNegative}\} \tag{13.5.2}$$

If the classifier predicts class $c \in \{\mathsf{true}, \mathsf{false}\}$ and the truth is class $t \in \{\mathsf{true}, \mathsf{false}\}$ these are defined as

$$\begin{array}{lll} \mathsf{TruePositive} & c = \mathsf{true} & t = \mathsf{true} \\ \mathsf{FalsePositive} & c = \mathsf{true} & t = \mathsf{false} \\ \mathsf{TrueNegative} & c = \mathsf{false} & t = \mathsf{false} \\ \mathsf{FalseNegative} & c = \mathsf{false} & t = \mathsf{true} \end{array} \tag{13.5.3}$$

We call $\mathbf{o}_a = \{o_a(n), n = 1, \ldots, N\}$, the outcomes for classifier A, and similarly for $\mathbf{o}_b = \{o_b(n), n = 1, \ldots, N\}$ for classifier B. To be specific we have two hypotheses we wish to test:

1. H_{diff} : $\mathbf{o}_a$ and $\mathbf{o}_b$ are from different categorical distributions.
2. H_{same} : $\mathbf{o}_a$ and $\mathbf{o}_b$ are from the same categorical distribution.

In both cases we will use categorical models $p(\mathbf{o}_c = q|\boldsymbol{\gamma}, H) = \gamma_q^c$, with unknown parameters $\boldsymbol{\alpha}$ (hypothesis 2 will correspond to using the same parameters $\boldsymbol{\gamma}_a = \boldsymbol{\gamma}_b$ for both classifiers, and hypothesis 1 to using different parameters, as we will discuss below). In the Bayesian framework, we want to find how likely it is that a model/hypothesis is responsible for generating the data. For any hypothesis H calculate

$$p(H|\mathbf{o}_a, \mathbf{o}_b) = \frac{p(\mathbf{o}_a, \mathbf{o}_b|H)p(H)}{p(\mathbf{o}_a, \mathbf{o}_b)} \tag{13.5.4}$$

where $p(H)$ is our prior belief that H is the correct hypothesis. Note that the normalising constant $p(\mathbf{o}_a, \mathbf{o}_b)$ does not depend on the hypothesis.

Under all hypotheses we will make the *independence of trials assumption*

$$p(\mathbf{o}_a, \mathbf{o}_b|H) = \prod_{n=1}^{N} p(o_a(n), o_b(n)|H). \tag{13.5.5}$$

To make further progress we need to state what the specific hypotheses mean.

13.5.2 H_{diff} : Model Likelihood

We now use the above assumptions to compute the Hypothesis likelihood:

$$p(H_{\mathsf{diff}}|\mathbf{o}_a, \mathbf{o}_b) = \frac{p(\mathbf{o}_a, \mathbf{o}_b|H_{\mathsf{diff}})p(H_{\mathsf{diff}})}{p(\mathbf{o}_a, \mathbf{o}_b)} \tag{13.5.6}$$

The outcome model for classifier A is specified using parameters, $\boldsymbol{\alpha}$, giving $p(\mathbf{o}_a|\boldsymbol{\alpha}, H_{\mathsf{diff}})$, and similarly we use β for classifier B. The finite amount of data means that we are uncertain as to these parameter values, and therefore the joint term in the numerator above is

$$p(\mathbf{o}_a, \mathbf{o}_b)p(H_{\mathsf{diff}}|\mathbf{o}_a, \mathbf{o}_b) = \int p(\mathbf{o}_a, \mathbf{o}_b|\boldsymbol{\alpha}, \boldsymbol{\beta}, H_{\mathsf{diff}})p(\boldsymbol{\alpha}, \boldsymbol{\beta}|H_{\mathsf{diff}})p(H_{\mathsf{diff}})d\boldsymbol{\alpha}d\boldsymbol{\beta} \tag{13.5.7}$$

$$= p(H_{\mathsf{diff}}) \int p(\mathbf{o}_a|\boldsymbol{\alpha}, H_{\mathsf{diff}})p(\boldsymbol{\alpha}|H_{\mathsf{diff}})d\boldsymbol{\alpha} \int p(\mathbf{o}_b|\boldsymbol{\beta}, H_{\mathsf{diff}})p(\boldsymbol{\beta}|H_{\mathsf{diff}})d\boldsymbol{\beta} \tag{13.5.8}$$

where we assumed

$$p(\boldsymbol{\alpha}, \boldsymbol{\beta}|H_{\mathsf{diff}}) = p(\boldsymbol{\alpha}|H_{\mathsf{diff}})p(\boldsymbol{\beta}|H_{\mathsf{diff}}) \quad \text{and} \quad p(\mathbf{o}_a, \mathbf{o}_b|\boldsymbol{\alpha}, \boldsymbol{\beta}, H_{\mathsf{diff}}) = p(\mathbf{o}_a|\boldsymbol{\alpha}, H_{\mathsf{diff}})p(\mathbf{o}_b|\boldsymbol{\beta}, H_{\mathsf{diff}}) \tag{13.5.9}$$

Note that one might expect there to be a specific constraint that the two models A and B are different. However since the models are assumed independent and each has parameters sampled from an effectively infinite set ($\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are continuous), the probability that sampled parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ of the two models are the same is zero.

Since we are dealing with categorical distributions, it is convenient to use the Dirichlet prior, which is conjugate to the categorical distribution:

$$p(\boldsymbol{\alpha}|H_{\mathsf{diff}}) = \frac{1}{Z(\mathbf{u})} \prod_q \alpha_q^{u_q - 1}, \qquad Z(\mathbf{u}) = \frac{\prod_{q=1}^{Q} \Gamma(u_q)}{\Gamma\left(\sum_{q=1}^{Q} u_q\right)} \tag{13.5.10}$$

The prior hyperparameter $\mathbf{u}$ controls how strongly the mass of the distribution is pushed to the corners of the simplex, see fig(8.5). Setting $u_q = 1$ for all q corresponds to a uniform prior. The likelihood of $\mathbf{o}_a$ is given by

$$\int p(\mathbf{o}_a|\boldsymbol{\alpha}, H_{\mathsf{diff}})p(\boldsymbol{\alpha}|H_{\mathsf{diff}})d\boldsymbol{\alpha} = \int \prod_q \alpha_q^{\sharp_q^a} \frac{1}{Z(\mathbf{u})} \prod_q \alpha_q^{u_q - 1} d\boldsymbol{\alpha} = \frac{Z(\mathbf{u} + \sharp^a)}{Z(\mathbf{u})} \tag{13.5.11}$$

where $\sharp^a$ is a vector with components $\sharp_q^a$ being the number of times that variable a is is state q in the data. Hence

$$p(H_{\mathsf{diff}}|\mathbf{o}_a, \mathbf{o}_b) = p(H_{\mathsf{diff}}) \frac{Z(\mathbf{u} + \sharp^a)}{Z(\mathbf{u})} \frac{Z(\mathbf{u} + \sharp^b)}{Z(\mathbf{u})} \tag{13.5.12}$$

where $Z(\mathbf{u})$ is given by equation (13.5.10).

13.5.3 H_{same} : Model Likelihood

In H_{same}, the hypothesis is that the outcomes for the two classifiers are generated from the *same* categorical distribution. Hence

$$p(\mathbf{o}_a, \mathbf{o}_b)p(H_{\mathsf{same}}|\mathbf{o}_a, \mathbf{o}_b) = p(H_{\mathsf{same}}) \int p(\mathbf{o}_a|\boldsymbol{\alpha}, H_{\mathsf{same}})p(\mathbf{o}_b|\boldsymbol{\alpha}, H_{\mathsf{same}})p(\boldsymbol{\alpha}|H_{\mathsf{same}})d\boldsymbol{\alpha} \tag{13.5.13}$$

$$= p(H_{\mathsf{same}}) \frac{Z(\mathbf{u} + \sharp^a + \sharp^b)}{Z(\mathbf{u})} \tag{13.5.14}$$

Bayes' factor

If we assume that we have no prior preference for either hypothesis ($p(H_{\mathsf{diff}}) = p(H_{\mathsf{same}})$), then

$$\frac{p(H_{\mathsf{diff}}|\mathbf{o}_a, \mathbf{o}_b)}{p(H_{\mathsf{same}}|\mathbf{o}_a, \mathbf{o}_b)} = \frac{Z(\mathbf{u} + \sharp^a)Z(\mathbf{u} + \sharp^b)}{Z(\mathbf{u})Z(\mathbf{u} + \sharp^a + \sharp^b)} \tag{13.5.15}$$

This is the evidence to suggest that the data were generated by two different categorical distributions.

Example 62. Two people classify the expression of each image into happy, sad or normal, using states $1, 2, 3$ respectively. Each column of the data below represents an image classed by the two people (person 1 is the top row and person 2 the second row). Are the two people essentially in agreement?

1	3	1	3	1	1	3	2	2	3	1	1	1	1	1	1	1	1	1	2
1	3	1	2	2	3	3	3	2	3	3	2	2	2	2	1	2	1	3	2

To help answer this question, we perform a H_{diff} versus H_{same} test. From this data, the count vector for person 1 is $[13, 3, 4]$ and for person 2, $[4, 9, 7]$. Based on a flat prior for the categorical distribution and assuming no prior preference for either hypothesis, we have the Bayes' factor

$$\frac{p(\text{persons 1 and 2 classify differently})}{p(\text{persons 1 and 2 classify the same})} = \frac{Z([14, 4, 5])Z([5, 10, 8])}{Z([1, 1, 1])Z([18, 13, 12])} = 12.87 \tag{13.5.16}$$

where the Z function is given in equation (13.5.10). This is strong evidence the two people are classifying the images differently.

Below we discuss some further examples for the H_{diff} versus H_{same} test. As above, the only quantities we need for this test are the vector counts from the data. Let's assume that there are three kinds of outcomes, $Q = 3$. For example $\mathrm{dom}(o) = \{\mathsf{good}, \mathsf{bad}, \mathsf{ugly}\}$ are our set of outcomes and we want to test if two classifiers are essentially producing the same outcome distributions, or different.

Example 63 (H_{diff} versus H_{same})**.**

- We have the two outcome counts $\sharp^a = [39, 26, 35]$ and $\sharp^b = [63, 12, 25]$. Then, the Bayes' factor equation (13.5.15) is 20.7 – strong evidence in favour of the two classifiers being different.
- Alternatively, consider the two outcome counts $\sharp^a = [52, 20, 28]$ and $\sharp^b = [44, 14, 42]$. Then, the Bayes' factor equation (13.5.15) is 0.38 – weak evidence against the two classifiers being different.
- As a final example, consider counts $\sharp^a = [459, 191, 350]$ and $\sharp^b = [465, 206, 329]$. This gives a Bayes' factor equation (13.5.15) of 0.008 – strong evidence that the two classifiers are statistically the same.

In all cases the results are consistent with the model in fact used to generate the count data – the two outcomes for A and B were indeed from different categorical distributions. The more test data we have, the more confident we are in our statements.

13.5.4 Dependent outcome analysis

Here we consider the (perhaps more common) case that outcomes are *dependent*. For example, it is often the case that if classifier A works well, then classifier B will also work well. Thus we want to evaluate the hypothesis:

$$H_{\mathsf{dep}}: \text{ the outcomes that the two classifiers make are dependent} \tag{13.5.17}$$

To do so we assume a categorical distribution over the joint states:

$$p(o_a(n), o_b(n)|\mathbf{P}, H_{\mathsf{dep}}) \tag{13.5.18}$$

Here $\mathbf{P}$ is a $Q \times Q$ matrix of probabilities:

$$[P]_{ij} = p(o_a = i, o_b = j) \tag{13.5.19}$$

so $[P]_{ij}$ is the probability that A makes outcome i, and B makes outcome j. Then,

$$p(\mathbf{o}|H_{\mathsf{dep}}) = \int p(\mathbf{o}, \mathbf{P}|H_{\mathsf{dep}})d\mathbf{P} = \int p(\mathbf{o}|\mathbf{P}, H_{\mathsf{dep}})p(\mathbf{P}|H_{\mathsf{dep}})d\mathbf{P}$$

where, for convenience, we write $\mathbf{o} = (\mathbf{o}_a, \mathbf{o}_b)$. Assuming a Dirichlet prior on $\mathbf{P}$, with hyperparameters $\mathbf{U}$, we have

$$p(\mathbf{o})p(H_{\mathsf{dep}}|\mathbf{o}) = p(H_{\mathsf{dep}})\frac{Z(vec\,(\mathbf{U} + \sharp))}{Z(vec(\mathbf{U}))} \tag{13.5.20}$$

where $vec(\mathbf{D})$ is a vector formed from concatenating the rows of the matrix $\mathbf{D}$. Here $\sharp$ is the count matrix, with $[\sharp]_{ij}$ equal to the number of times that joint outcome $(o_a = i, o_b = j)$ occurred in the N datapoints. As before, we can then use this in a Bayes factor calculation. For the uniform prior, $[U]_{ij} = 1, \forall i, j$.

Testing for dependencies in the outcomes: H_{dep} versus H_{diff}

To test whether or not the outcomes of the classifiers are dependent H_{dep} against the hypothesis that they are independent H_{diff} we may use, assuming $p(H_{\mathsf{diff}}) = p(H_{\mathsf{dep}})$,

$$\frac{p(H_{\mathsf{diff}}|\mathbf{o})}{p(H_{\mathsf{dep}}|\mathbf{o})} = \frac{Z(\mathbf{u} + \sharp^a)}{Z(\mathbf{u})}\frac{Z(\mathbf{u} + \sharp^b)}{Z(\mathbf{u})}\frac{Z(vec(\mathbf{U}))}{Z(vec\,(\mathbf{U} + \sharp))} \tag{13.5.21}$$

Example 64 (H_{dep} versus H_{diff}).

- Consider the outcome count matrix $\sharp$

$$\begin{pmatrix} 98 & 7 & 93 \\ 168 & 13 & 163 \\ 245 & 12 & 201 \end{pmatrix} \tag{13.5.22}$$

  so that $\sharp^a = [511, 32, 457]$, and $\sharp^b = [198, 344, 458]$. Then

$$\frac{p(H_{\mathsf{diff}}|\mathbf{o})}{p(H_{\mathsf{dep}}|\mathbf{o})} = 3020 \tag{13.5.23}$$

  - strong evidence that the classifiers perform independently.

- Consider the outcome count matrix $\sharp$

$$\begin{pmatrix} 82 & 120 & 83 \\ 107 & 162 & 4 \\ 170 & 203 & 70 \end{pmatrix} \tag{13.5.24}$$

  so that $\sharp^a = [359, 485, 156]$, and $\sharp^b = [284, 273, 443]$. Then

$$\frac{p(H_{\mathsf{diff}}|\mathbf{o})}{p(H_{\mathsf{dep}}|\mathbf{o})} = 2 \times 10^{-18} \tag{13.5.25}$$

  - strong evidence that the classifiers perform dependently.

These results are in fact consistent with the way the data was generated in each case.

Testing for dependencies in the outcomes: H_{dep} versus H_{same}

In practice, it is reasonable to believe that dependencies are quite likely in the outcomes that classifiers. For example two classifiers will often do well on ‘easy’ test examples, and badly on ‘difficult’ examples.

Are these dependencies strong enough to make us believe that the outcomes are coming from the *same* process? In this sense, we want to test

$$\frac{p(H_{\mathsf{same}}|\mathbf{o})}{p(H_{\mathsf{dep}}|\mathbf{o})} = \frac{Z(\mathbf{u}+\sharp^a+\sharp^b)}{Z(\mathbf{u})}\frac{Z(vec(\mathbf{U}))}{Z(vec(\mathbf{U}+\sharp))} \tag{13.5.26}$$

Example 65 (H_{dep} versus H_{same})**.**

- Consider an experiment which gives the test outcome count matrix $\sharp$

$$\begin{pmatrix} 105 & 42 & 172 \\ 42 & 45 & 29 \\ 192 & 203 & 170 \end{pmatrix} \tag{13.5.27}$$

  so that $\sharp^a = [339, 290, 371]$, and $\sharp^b = [319, 116, 565]$. Then

$$\frac{p(H_{\mathsf{same}}|\mathbf{o})}{p(H_{\mathsf{dep}}|\mathbf{o})} = 4.5 \times 10^{-38} \tag{13.5.28}$$

  – strong evidence that the classifiers are performing differently.

- Consider an experiment which gives the test outcome count matrix $\sharp$

$$\begin{pmatrix} 15 & 8 & 10 \\ 5 & 4 & 8 \\ 13 & 12 & 25 \end{pmatrix} \tag{13.5.29}$$

  so that $\sharp^a = [33, 24, 43]$, and $\sharp^b = [33, 17, 50]$. Then

$$\frac{p(H_{\mathsf{same}}|\mathbf{o})}{p(H_{\mathsf{dep}}|\mathbf{o})} = 42 \tag{13.5.30}$$

  – strong evidence that the classifiers are performing the same.

These results are in fact consistent with the way the data was generated.

13.5.5 Is classifier A better than B?

We return to the question with which we began this outcome analysis. Given the common scenario of observing a number of errors for classifier A on a test set and a number for B, can we say which classifier is better? This corresponds to the special case of binary classes $Q = 2$ with $\mathrm{dom}(e) = \{\mathsf{correct}, \mathsf{incorrect}\}$. Under the H_{diff} for this special case it makes sense to use a Beta distribution (which corresponds to the Dirichlet when $Q = 2$). Then for θ_a being the probability that classifier A generates a correct label we have

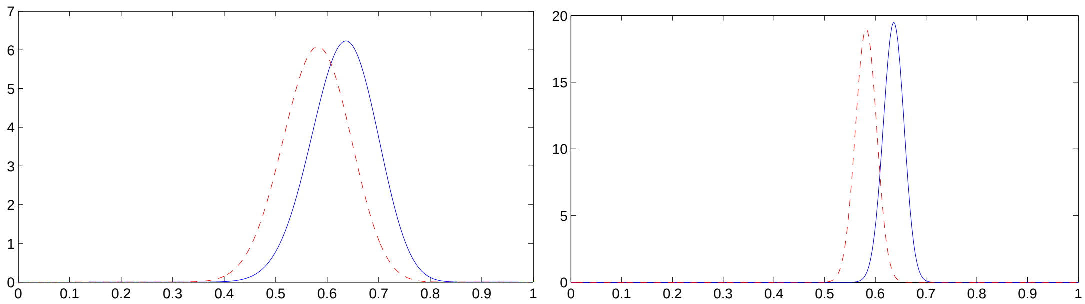
$$p(\mathbf{o}_A|\theta_a) = \theta_a^{\sharp^a_{\mathsf{correct}}}(1-\theta_a)^{\sharp^a_{\mathsf{incorrect}}} \tag{13.5.31}$$

Similarly

$$p(\mathbf{o}_B|\theta_b) = \theta_b^{\sharp^b_{\mathsf{correct}}}(1-\theta_b)^{\sharp^b_{\mathsf{incorrect}}} \tag{13.5.32}$$

We assume independent identical Beta distribution priors

$$p(\theta_a) = B(\theta_a|u_1, u_2), \qquad p(\theta_b) = B(\theta_b|u_1, u_2) \tag{13.5.33}$$



(a) (b)

Figure 13.11: Two classifiers A and B and their posterior distributions of the probability that they classify correctly (using a uniform Beta prior). **(a)**: For A with 35 correct and 20 incorrect labels, $B(x|1+35, 1+20)$ (solid curve); B with 32 correct 23 incorrect $B(y|1+32, 1+23)$ (dashed curve). $p(x > y) = 0.719$ **(b)**: For A with 350 correct and 200 incorrect labels (solid curve), $B(x|1+350, 1+200)$; B with 320 correct 230 incorrect $B(y|1+320, 1+230)$ (dashed curve), $p(x > y) = 0.968$. As the amount of data increases the overlap between the distributions decreases and the certainty that one classifier is better than the other correspondingly increases.

where a flat prior corresponds to using the hyperparameter setting $u_1 = u_2 = 1$. The posterior distributions for θ_a and θ_b are independent:

$$p(\theta_a|\mathbf{o}_A) = B\left(\theta_a|\sharp^a_{\text{correct}} + u_1, \sharp^a_{\text{incorrect}} + u_2\right), \quad p(\theta_b|\mathbf{o}_B) = B\left(\theta_b|\sharp^b_{\text{correct}} + u_1, \sharp^b_{\text{incorrect}} + u_2\right) \tag{13.5.34}$$

The question of whether A is better than B can then be addressed by computing

$$p(\theta_a > \theta_b|\mathbf{o}_A, \mathbf{o}_B) = \frac{1}{B(a,b)B(c,d)} \int_0^1 x^{a-1}(1-x)^{b-1} \int_x^1 y^{c-1}(1-y)^{d-1}\,dy dx \tag{13.5.35}$$

where

$$a = \sharp^a_{\text{correct}} + u_1, \qquad b = \sharp^a_{\text{incorrect}} + u_2, \qquad c = \sharp^b_{\text{correct}} + u_1, \qquad d = \sharp^b_{\text{incorrect}} + u_2 \tag{13.5.36}$$

Example 66. Classifier A makes 20 errors, and 35 correct classifications, whereas classifier B makes 23 errors and 32 correct classifications. Using a flat prior this gives

$$p(\theta_a > \theta_b|\mathbf{o}_A, \mathbf{o}_B) = \texttt{betaXbiggerY(1+35,1+20,1+32,1+23)} = 0.719 \tag{13.5.37}$$

On the other hand if classifier A makes 200 errors and 350 correct classifications, whilst classifier B makes 230 errors and 320 correct classifications, we have

$$p(\theta_a > \theta_b|\mathbf{o}_A, \mathbf{o}_B) = \texttt{betaXbiggerY(1+350,1+200,1+320,1+230)} = 0.968 \tag{13.5.38}$$

This demonstrates the intuitive effect that even though the proportion of correct/incorrect classifications doesn't change for the two scenarios, as we have more data our confidence in determining the better classifier increases.

13.6 Code

`demoBayesErrorAnalysis.m`: Demo for Bayesian Error Analysis
`betaXbiggerY.m`: $p(x > y)$ for $x \sim B(x|a, b)$, $y \sim B(y|c, d)$

13.7 Notes

A general introduction to machine learning is given in [198]. An excellent reference for Bayesian decision theory is [33]. Approaches based on empirical risk are discussed in [279].

13.8 Exercises

Exercise 150. *Given the distributions* $p(x|class1) = \mathcal{N}\left(x|\mu_1, \sigma_1^2\right)$ *and* $p(x|class2) = \mathcal{N}\left(x|\mu_2, \sigma_2^2\right)$*, with corresponding prior occurrence of classes* p_1 *and* p_2 $(p_1+p_2 = 1)$*, calculate the decision boundary explicitly as a function of* $\mu_1, \mu_2, \sigma_1^2, \sigma_2^2, p_1, p_2$*. How many solutions are there to the decision boundary, and are they all reasonable?*

Exercise 151. *Suppose that instead of using the Bayes' decision rule to choose class* k *if* $p(\mathcal{C}_k|\mathbf{x}) > p(\mathcal{C}_j|\mathbf{x})$ *for all* $j \neq k$*, we use a randomized decision rule, choosing class* j *with probability* $Q(\mathcal{C}_j|\mathbf{x})$*. Calculate the error for this decision rule, and show that the error is minimized by using Bayes' decision rule.*

Exercise 152. *WowCo.com is a new startup prediction company. After years of failures, they eventually find a neural network with a trillion hidden units that achieves zero test error on every learning problem posted on the internet up last week. Each learning problem included a training and test set. Proud of their achievement, they market their product aggressively with the claim that it 'predicts perfectly on all known problems'. Would you buy this product? Justify your answer.*

Exercise 153. *Three people classify images into 1 of three categories. Each column in the table below represents the classifications of each image, with the top row being the class from person 1, the middle from person 2 and the bottom from person 3.*

1	3	1	3	1	1	3	2	2	3	1	1	1	1	1	1	1	1	1	2
1	3	1	2	2	3	3	3	2	3	3	2	2	2	2	1	2	1	3	2
1	2	1	1	1	3	2	2	2	3	1	2	1	2	1	1	2	3	3	2

Assuming no prior preference amongst hypotheses and a uniform prior on counts, compute

$$\frac{p(\text{persons 1, 2 and 3 classify differently})}{p(\text{persons 1, 2 and 3 classify the same})} \tag{13.8.1}$$

Exercise 154 (Better than random guessing?)**.** *Consider a classifier that makes* R *correct classifications and* W *wrong classifications. Is the classifier better than random guessing? Let* $\mathcal{D}$ *be the fact that there are* R *right and* W *wrong answers. Assume also that the classifications are i.i.d.*

1. *Show that under the hypothesis the data is generated purely at random,*

$$p(\mathcal{D}|\mathcal{H}_{random}) = 0.5^{R+W} \tag{13.8.2}$$

2. *Define* θ *to be the probability that the classifier makes an error. Then*

$$p(\mathcal{D}|\theta) = \theta^R (1-\theta)^W \tag{13.8.3}$$

Then consider

$$p(\mathcal{D}|\mathcal{H}_{non\ random}) = \int_\theta p(\mathcal{D}|\theta)p(\theta) \tag{13.8.4}$$

Show that for a Beta prior, $p(\theta) = B(\theta|a, b)$

$$p(\mathcal{D}|\mathcal{H}_{non\ random}) = \frac{B(R+a, W+b)}{B(a, b)} \tag{13.8.5}$$

where $B(a, b)$ *is the beta-function.*

3. *Considering the random and non-random Hypotheses as* a priori *equally likely, show that*

$$p(\mathcal{H}_{random}|\mathcal{D}) = \frac{0.5^{R+W}}{0.5^{R+W} + \frac{B(R+a,W+b)}{B(a,b)}} \tag{13.8.6}$$

4. *For a flat prior $a = b = 1$ compute the probability that for 10 correct and 12 incorrect classifications, the data is from a purely random distribution (according to equation (13.8.6)). Repeat this for 100 correct and 120 incorrect classifications.*

5. *Show that the standard deviation in the number of errors of a random classifier is $0.5\sqrt{R+W}$ and relate this to the above computation.*

Exercise 155. *For a prediction model $\tilde{p}(y|x)$ and true data generating distribution $p(x, y)$, we define the accuracy as*

$$A = \int_{x,y} p(x, y)\tilde{p}(y|x) \tag{13.8.7}$$

1. *By defining*

$$\hat{p}(x, y) \equiv \frac{p(x, y)\tilde{p}(y|x)}{A} \tag{13.8.8}$$

and considering

$$KL(q(x, y)|\hat{p}(x, y)) \geq 0 \tag{13.8.9}$$

show that for any distribution $q(x, y)$,

$$\log A \geq -KL(q(x, y)|p(x, y)) + \langle \log \tilde{p}(y|x)\rangle_{q(x,y)} \tag{13.8.10}$$

2. *You are given a set of training data $\mathcal{D} = \{x^n, y^n, n = 1, \ldots, N\}$. By taking $q(x, y)$ to be the empirical distribution*

$$q(x, y) = \frac{1}{N}\sum_{n=1}^{n} \delta(x, x^n)\,\delta(y, y^n) \tag{13.8.11}$$

show that

$$\log A \geq -KL(q(x, y)|p(x, y)) + \frac{1}{N}\sum_{n=1}^{n} \log \tilde{p}(y^n|x^n) \tag{13.8.12}$$

This shows that the prediction accuracy is lower bounded by the training accuracy and the 'gap' between the empirical distribution and the unknown true data generating mechanism. In theories such as PAC Bayes, one may bound this gap, resulting in a bound on the predictive accuracy. According to this naive bound, the best thing to do to increase the prediction accuracy is to increase the training accuracy (since the first Kullback-Leibler term is independent of the predictor). As N increases, the first term Kullback-Leibler term becomes small, and minimising the training error is justifiable.

Assuming that the training data are drawn from a distribution $p(y|x)$ which is deterministic, show that

$$\log A \geq -KL(q(x)|p(x)) + \frac{1}{N}\sum_{n=1}^{n} \log \tilde{p}(y^n|x^n) \tag{13.8.13}$$

and hence that, provided the training data is correctly predicted, $(q(y^n|x^n) = p(y^n|x^n)$, the accuracy can be related to the empirical input distribution and true input distribution by

$$A \geq e^{-KL(q(x)|p(x))} \tag{13.8.14}$$

CHAPTER 14

Nearest Neighbour Classification

14.1 Do as your neighbour does

Successful prediction typically relies on smoothness in the data – if the class label can change arbitrarily as we move a small amount in the input space, the problem is essentially random and no algorithm will generalise well. Machine Learning researchers therefore construct appropriate measures of smoothness for the problem they have at hand. Nearest neighbour methods are a good starting point since they readily encode basic smoothness intuitions and are easy to program, forming a useful baseline method.

In a classification problem each input vector $\mathbf{x}$ has a corresponding class label, $c^n \in \{1, \ldots, C\}$. Given a dataset of N such training examples, $\mathcal{D} = \{\mathbf{x}^n, c^n\}, n = 1, \ldots, N$, and a novel $\mathbf{x}$, we aim to return the correct class $c(\mathbf{x})$. A simple, but often effective, strategy for this supervised learning problem can be stated as: for novel $\mathbf{x}$, find the nearest input in the training set and use the class of this nearest input, algorithm(13).

For vectors $\mathbf{x}$ and $\mathbf{x}'$ representing two different datapoints, we measure 'nearness' using a *dissimilarity function* $d(\mathbf{x}, \mathbf{x}')$. A common dissimilarity is the *squared Euclidean distance*

$$d(\mathbf{x}, \mathbf{x}') = (\mathbf{x} - \mathbf{x}')^\mathsf{T}(\mathbf{x} - \mathbf{x}') \tag{14.1.1}$$

which can be more conveniently written $(\mathbf{x} - \mathbf{x}')^2$. Based on the squared Euclidean distance, the decision boundary is determined by the lines which are the perpendicular bisectors of the closest training points with different training labels, see fig(14.1). This is called a *Voronoi tessellation*.

The nearest neighbour algorithm is simple and intuitive. There are, however, some issues:

- How should we measure the distance between points? Whilst the Euclidean square distance is popular, this may not always be appropriate. A fundamental limitation of the Euclidean distance is that it does not take into account how the data is distributed. For example if the length scales of $\mathbf{x}$ vary greatly the largest length scale will dominate the squared distance, with potentially useful class-specific information in other components of $\mathbf{x}$ lost. The *Mahalanobis distance*

  $$d(\mathbf{x}, \mathbf{x}') = \left(\mathbf{x} - \mathbf{x}'\right)^\mathsf{T} \mathbf{\Sigma}^{-1} \left(\mathbf{x} - \mathbf{x}'\right) \tag{14.1.2}$$

  where $\mathbf{\Sigma}$ is the covariance matrix of the inputs (from all classes) can overcome some of these problems since it rescales all length scales to be essentially equal.

- The whole dataset needs to be stored to make a classification. This can be addressed by a method called data editing in which datapoints which have little or no affect on the decision boundary are removed from the training dataset.

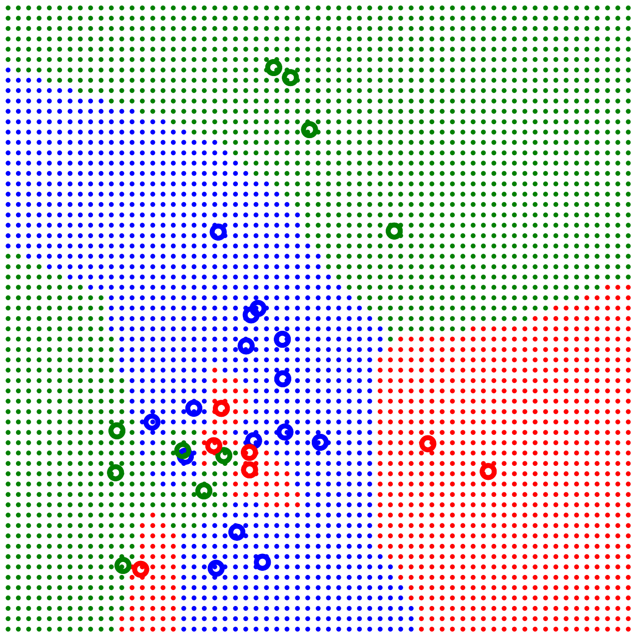


Figure 14.1: In nearest neighbour classification a new vector is assigned the label of the nearest vector in the training set. Here there are three classes, with training points given by the circles, along with their class. The dots indicate the class of the nearest training vector. The decision boundary is piecewise linear with each segment corresponding to the perpendicular bisector between two datapoints belonging to different classes, giving rise to a Voronoi tessellation of the input space.

Algorithm 13 Nearest neighbour algorithm to classify a new vector $\mathbf{x}$, given a set of training data $\mathcal{D} = \{(\mathbf{x}^n, c^n), n = 1, \ldots, N\}$:

1: Calculate the dissimilarity of the test point $\mathbf{x}$ to each of the stored points, $d^n = d(\mathbf{x}, \mathbf{x}^n)$, $n = 1, \ldots, N$.
2: Find the training point $\mathbf{x}^{n^*}$ which is nearest to $\mathbf{x}$:

$$n^* = \underset{n}{\operatorname{argmin}}\ d(\mathbf{x}, \mathbf{x}^n)$$

3: Assign the class label $c(\mathbf{x}) = c^{n^*}$.
4: In the case that there are two or more 'equidistant' neighbours with different class labels, the most numerous class is chosen. If there is no one single most numerous class, we use the K-nearest-neighbours case described in the next section.

- Each distance calculation can be expensive if the datapoints are high dimensional. Principal Components Analysis, see chapter(15), is one way to address this and replaces $\mathbf{x}^n$ with a low dimensional projection $\mathbf{p}$. The Euclidean distance of two datapoints $\left(\mathbf{x}^a - \mathbf{x}^b\right)^2$ is then approximately given by $\left(\mathbf{p}^a - \mathbf{p}^b\right)^2$, see section(15.2.4). This is both faster to compute and can also improve classification accuracy since only the large scale characteristics of the data are retained in the PCA projections.

- It is not clear how to deal with missing data or incorporate prior beliefs and domain knowledge.

- For large databases, computing the nearest neighbour of a novel point $\mathbf{x}^*$ can be very time-consuming since $\mathbf{x}^*$ needs to be compared to each of the training points. Depending on the geometry of the training points, finding the nearest neighbour can accelerated by examining the values of each of the components x_i of $\mathbf{x}$ in turn. Such an axis-aligned space-split is called a *KD-tree*[200] and can reduce the possible set of candidate nearest neighbours in the training set to the novel $\mathbf{x}^*$, particularly in low dimensions.

14.2 K-Nearest Neighbours

Basing the classification on only the single nearest neighbour can lead to inaccuracies. If your neighbour is simply mistaken (has an incorrect training class label), or is not a particularly representative example of his class, then these situations will typically result in an incorrect classification. By including more than the single nearest neighbour, we hope to make a more robust classifier with a smoother decision boundary (less swayed by single neighbour opinions). For datapoints which are somewhat anomalous compared with

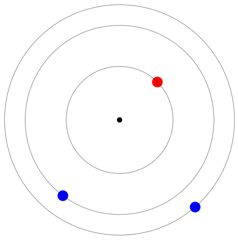


Figure 14.2: In K-nearest neighbours, we centre a hypersphere around the point we wish to classify (here the central dot). The inner circle corresponds to the nearest neighbour. However, using the 3 nearest neighbours, we find that there are two 'blue' classes and one 'red' – and we would therefore class the point as 'blue' class. In the case of a tie, one can extend K until the tie is broken.

neighbours from the same class, their influence will be outvoted.

If we assume the Euclidean distance as the dissimilarity measure, the K-Nearest Neighbour algorithm considers a hypersphere centred on the test point $\mathbf{x}$. We increase the radius r until the hypersphere contains exactly K points in the training data. The class label $c(\mathbf{x})$ is then given by the most numerous class within the hypersphere.

Choosing K

Whilst there is some sense in making $K > 1$, there is certainly little sense in making $K = N$ (N being the number of training points). For K very large, all classifications will become the same – simply assign each novel $\mathbf{x}$ to the most numerous class in the training data. This suggests that there is an 'optimal' intermediate setting of K which gives the best generalisation performance. This can be determined using cross-validation, as described in section(13.2.3).

Example 67 (Handwritten Digit Example)**.** Consider two classes of handwritten digits, zeros and ones. Each digit contains $28 \times 28 = 784$ pixels. The training data consists of 300 zeros, and 300 ones, a subset of which are plotted in fig(14.3a,b). To test the performance of the nearest neighbour method (based on Euclidean distance) we use an independent test set containing a further 600 digits. The nearest neighbour method, applied to this data, correctly predicts the class label of all 600 test points. The reason for the high success rate is that examples of zeros and ones are sufficiently different that they can be easily distinguished.

A more difficult task is to distinguish between ones and sevens. We repeat the above experiment, now using 300 training examples of ones, and 300 training examples of sevens, fig(14.3b,c). Again, 600 new test examples (containing 300 ones and 300 sevens) were used to assess the performance. This time, 18 errors are found using nearest neighbour classification – a 3% error rate for this two class problem. The 18 test points on which the nearest neighbour method makes errors are plotted in fig(14.4). If we use $K = 3$ nearest neighbours, the classification error reduces to 14 – a slight improvement. As an aside, the best Machine Learning methods classify real world digits (over all 10 classes) to an error of less than 1% (`yann.lecun.com/exdb/mnist`) – better than the performance of an 'average' human.

14.3 A Probabilistic Interpretation of Nearest Neighbours

Consider the situation where we have (for simplicity) data from two classes – class 0 and class 1. We make the following mixture model for data from class 0:

$$p(\mathbf{x}|c=0) = \frac{1}{N_0} \sum_{n \in \text{ class } 0} \mathcal{N}\left(\mathbf{x}|\mathbf{x}^n, \sigma^2\mathbf{I}\right) = \frac{1}{N_0} \frac{1}{(2\pi\sigma^2)^{D/2}} \sum_{n \in \text{ class } 0} e^{-(\mathbf{x}-\mathbf{x}^n)^2/(2\sigma^2)} \tag{14.3.1}$$

where D is the dimension of a datapoint $\mathbf{x}$ and N_0 are the number of training datapoints of class 0, and σ^2 is the variance. This is a *Parzen estimator*, which models the data distribution as a uniform weighted sum of distributions centred on the training points, fig(14.5).

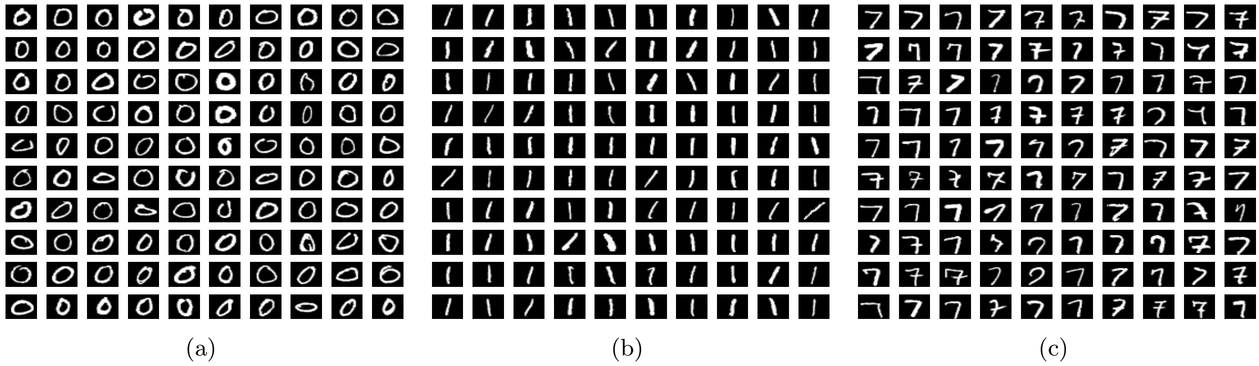


Figure 14.3: Some of the training examples of the digit zero and (a), one (b) and seven (c). There are 300 training examples of each of these three digit classes.

Figure 14.4: '1' versus '7' classification using the NN method. (Top) The 18 out of 600 test examples that are incorrectly classified; (Bottom) the nearest neighbours in the training set corresponding to each test-point above.

Similarly, for data from class 1:

$$p(\mathbf{x}|c=1) = \frac{1}{N_1} \sum_{n \in \text{ class } 1} \mathcal{N}\left(\mathbf{x}|\mathbf{x}^n, \sigma^2\mathbf{I}\right) = \frac{1}{N_1} \frac{1}{(2\pi\sigma^2)^{D/2}} \sum_{n \in \text{ class } 1} e^{-(\mathbf{x}-\mathbf{x}^n)^2/(2\sigma^2)} \tag{14.3.2}$$

To classify a new datapoint $\mathbf{x}^*$, we use Bayes' rule

$$p(c=0|\mathbf{x}^*) = \frac{p(\mathbf{x}^*|c=0)p(c=0)}{p(\mathbf{x}^*|c=0)p(c=0) + p(\mathbf{x}^*|c=1)p(c=1)} \tag{14.3.3}$$

The Maximum Likelihood setting of $p(c=0)$ is $N_0/(N_0+N_1)$, and $p(c=1) = N_1/(N_0+N_1)$. An analogous expression to equation (14.3.3) holds for $p(c=1|\mathbf{x}^*)$. To see which class is most likely we may use the ratio

$$\frac{p(c=0|\mathbf{x}^*)}{p(c=1|\mathbf{x}^*)} = \frac{p(\mathbf{x}^*|c=0)p(c=0)}{p(\mathbf{x}^*|c=1)p(c=1)} \tag{14.3.4}$$

If this ratio is greater than one, we classify $\mathbf{x}^*$ as 0, otherwise 1.

Equation(14.3.4) is a complicated function of $\mathbf{x}^*$. However, if σ^2 is very small, the numerator, which is a sum of exponential terms, will be dominated by that term for which datapoint $\mathbf{x}^{n_0}$ in class 0 is closest to the point $\mathbf{x}^*$. Similarly, the denominator will be dominated by that datapoint $\mathbf{x}^{n_1}$ in class 1 which is closest to $\mathbf{x}^*$. In this case, therefore,

$$\frac{p(c=0|\mathbf{x}^*)}{p(c=1|\mathbf{x}^*)} \approx \frac{e^{-(\mathbf{x}^*-\mathbf{x}^{n_0})^2/(2\sigma^2)}p(c=0)/N_0}{e^{-(\mathbf{x}^*-\mathbf{x}^{n_1})^2/(2\sigma^2)}p(c=1)/N_1} = \frac{e^{-(\mathbf{x}^*-\mathbf{x}^{n_0})^2/(2\sigma^2)}}{e^{-(\mathbf{x}^*-\mathbf{x}^{n_1})^2/(2\sigma^2)}} \tag{14.3.5}$$

Taking the limit $\sigma^2 \to 0$, with certainty we classify $\mathbf{x}^*$ as class 0 if $\mathbf{x}^*$ has a point in the class 0 data which is closer than the closest point in the class 1 data. The nearest (single) neighbour method is therefore recovered as the limiting case of a probabilistic generative model, see fig(14.5).

The motivation of using K nearest neighbours is to produce a result that is robust against unrepresentative nearest neighbours. To ensure a similar kind of robustness in the probabilistic interpretation, we may use a finite value $\sigma^2 > 0$. This smoothes the extreme probabilities of classification and means that more points (not just the nearest) will have an effective contribution in equation (14.3.4). The extension to more than

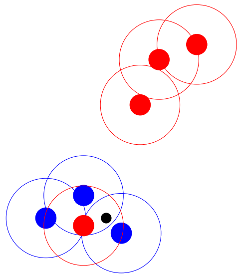


Figure 14.5: A probabilistic interpretation of nearest neighbours. For each class we use a mixture of Gaussians to model the data from that class $p(\mathbf{x}|c)$, placing at each training point an isotropic Gaussian of width σ^2. The width of each Gaussian is represented by the circle. In the limit $\sigma^2 \to 0$ a novel point (black) is assigned the class of its nearest neighbour. For finite $\sigma^2 > 0$ the influence of non-nearest neighbours has an effect, resulting in a soft version of nearest neighbours.

two classes is straightforward, requiring a class conditional generative model for each class.

To go beyond nearest neighbour methods, we can relax the assumption of using a Parzen estimator, and use a richer generative model. We will examine such cases in some detail in later chapters, in particular chapter(20).

14.3.1 When your nearest neighbour is far away

For a novel input $\mathbf{x}^*$ that is far from all training points, Nearest Neighbours, and its soft probabilistic variant will confidently classify $\mathbf{x}^*$ as belonging to the class of the nearest training point. This is arguably opposite to what we would like, namely that the classification should tend to the prior probabilities of the class based on the number of training data per class. A way to avoid this problem is, for each class, to include a fictitious mixture component at the mean of all the data with large variance, equal for each class. For novel inputs close to the training data, this extra fictitious datapoint will have no appreciable effect. However, as we move away from the high density regions of the training data, this additional fictitious component will dominate. Since the distance from $\mathbf{x}^*$ to each fictitious class point is the same, in the limit that $\mathbf{x}^*$ is far from the training data, the effect is that no class information from the position of $\mathbf{x}^*$ occurs. See section(20.3.3) for an example.

14.4 Code

`nearNeigh.m`: K Nearest Neighbour

14.4.1 Utility Routines

`majority.m`: Find the majority entry in each column of a matrix

14.4.2 Demonstration

`demoNearNeigh.m`: K Nearest Neighbour Demo

14.5 Exercises

Exercise 156. *The file* `NNdata.mat` *contains training and test data for the handwritten digits 5 and 9. Using leave one out cross-validation, find the optimal K in K-nearest neighours, and use this to compute the classification accuracy of the method on the test data.*

Exercise 157. *Write a routine* `SoftNearNeigh(xtrain,xtest,trainlabels,sigma)` *to implement soft nearest neighbours, analogous to* `nearNeigh.m`*. Here* `sigma` *is the variance σ^2 in equation (14.3.1). As above, the file* `NNdata.mat` *contains training and test data for the handwritten digits 5 and 9. Using leave one out cross-validation, find the optimal σ^2 and use this to compute the classification accuracy of the method on the test data. Hint: you may have numerical difficulty with this method. To avoid this, consider using the logarithm, and how to numerically compute $\log\left(e^a + e^b\right)$ for large (negative) a and b. See also* `logsumexp.m`*.*

Exercise 158. *In the text we suggested the use of the Mahalanobis distance*

$$d(\mathbf{x}, \mathbf{y}) = (\mathbf{x} - \mathbf{y})^{\mathsf{T}} \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \mathbf{y})$$

as a way to improve on the Euclidean distance, with $\boldsymbol{\Sigma}$ *the covariance matrix of the combined data from both classes. Consider a modification based on using a mixture model*

$$p(\mathbf{x}|c = 0) = \frac{1}{N_0} \sum_{n \in \text{class } 0} \mathcal{N}\left(\mathbf{x}|\mathbf{x}^n, \boldsymbol{\Sigma}_0\right)$$

and

$$p(\mathbf{x}|c = 1) = \frac{1}{N_1} \sum_{n \in \text{class } 1} \mathcal{N}\left(\mathbf{x}|\mathbf{x}^n, \boldsymbol{\Sigma}_1\right)$$

1. *Explain how the soft Nearest Neighbours algorithm can deal with the issue that the distribution of data from the different classes can be very different.*
2. *For the case* $\boldsymbol{\Sigma}_0 = \gamma^2 \boldsymbol{\Sigma}'_0$ *and* $\boldsymbol{\Sigma}_1 = \gamma^2 \boldsymbol{\Sigma}'_1$ *and* γ^2 *small, derive a simple expression that approximates*

$$\log \left(\frac{p(c = 0|\mathbf{x}^*)}{p(c = 1|\mathbf{x}^*)} \right)$$

Exercise 159. *The editor at YoMan! (a 'mens' magazine) has just had a great idea. Based on the success of a recent national poll to test IQ, she decides to make a 'Beauty Quotient' (BQ) test. She collects as many images of male faces as she can, taking care to make sure that all the images are scaled to roughly the same size and under the same lighting conditions. She then gives each male face a BQ score from 0 ('Severely Aesthetically Challenged') to 100 ('Generously Aesthetically Gifted'). Thus, for each image* $\mathbf{x}$*, there is an associated value* b *in the range 0 to 100. In total she collects* N *images and associated scores,* $\{(\mathbf{x}^n, b^n), n = 1, \ldots, N\}$*, where each image is represented by a* D*-dimensional real-valued vector* $\mathbf{x}$*. One morning, she bounces into your office and tells you the good news : it is your task to make a test for the male nation to determine their Beauty Quotient. The idea, she explains, is that a man can send online an image of their face* $\mathbf{x}^*$*, to YoMan! and will 'instantly' receive an automatic BQ response* b^**.*

1. *As a first step, you decide to use the* K *nearest neighbour method (KNN) to assign a BQ score* b^* *to a novel test image* $\mathbf{x}^*$*.*

   Describe how to determine the optimal number of neighbours K *to use.*

2. *Your line manager is pleased with your algorithm but is disappointed that it does not provide any simple explanation of Beauty that she can present in a future version of YoMan! magazine.*

   To address this, you decide to make a model based on linear regression. That is

$$b = \mathbf{w}^{\mathsf{T}} \mathbf{x} \tag{14.5.1}$$

   where $\mathbf{w}$ *is a parameter (weight) vector chosen to minimise*

$$E(\mathbf{w}) = \sum_n \left(b^n - \mathbf{w}^{\mathsf{T}} \mathbf{x}^n \right)^2$$

   (a) *After training (finding a suitable* $\mathbf{w}$*), how can YoMan! explain to its readership in a simple way what facial features are important for determining one's BQ?*

   (b) *Describe fully and mathematically a method to train this linear regression model. Your explanation must be detailed enough so that a programmer can directly implement it.*

   (c) *Discuss any implications of the situation* $D > N$*.*

   (d) *Discuss any advantages/disadvantages of using the linear regression model compared with using the KNN approach.*

CHAPTER 15

Unsupervised Linear Dimension Reduction

15.1 High-Dimensional Spaces – Low Dimensional Manifolds

In Machine Learning problems data is often high dimensional – images, bag-of-word descriptions, gene-expresssions *etc.* In such cases we cannot expect the training data to densely populate the space, meaning that there will be large parts in which little is known about the data. For the hand-written digits from chapter(14), the data is 784 dimensional. For binary valued pixels the possible number of images that could ever exist is $2^{784} \approx 10^{236}$. Nevertheless, we would expect that only a handful of examples of a digit should be sufficient (for a human) to understand how to recognise a 7. Digit-like images must therefore occupy a highly constrained subspace of the 784 dimensions and we expect only a small number of directions to be relevant for describing the data to a reasonable accuracy. Whilst the data vectors may be very high dimensional, they will therefore typically lie close to a much lower dimensional 'manifold' (informally, a two-dimensional manifold corresponds to a warped sheet of paper embedded in a high dimensional space), meaning that the distribution of the data is heavily constrained.

Here we concentrate on linear dimension reduction techniques for which there exist computationally efficient approaches. In this approach a high dimensional datapoint $\mathbf{x}$ is 'projected down' to a lower dimensional vector $\mathbf{y}$ by

$$\mathbf{y} = \mathbf{F}\mathbf{x} + \mathsf{const}. \tag{15.1.1}$$

where the non-square matrix $\mathbf{F}$ has dimensions $\dim(\mathbf{y}) \times \dim(\mathbf{x})$, with $\dim(\mathbf{y}) < \dim(\mathbf{x})$. The methods in this chapter are largely 'non-probabilistic', although many have natural probabilistic interpretations. For example, PCA is closely related to Factor Analysis, described in chapter(21).

15.2 Principal Components Analysis

If data lies close to a hyperplane, as in fig(15.1) we can accurately approximate each data point by using vectors that span the hyperplane alone. Effectively, we are trying to discover a low dimensional co-ordinate system in which we can approximately represent the data. We express the approximation for datapoint $\mathbf{x}^n$ as

$$\mathbf{x}^n \approx \mathbf{c} + \sum_{i=1}^{M} y_i^n \mathbf{b}^i \equiv \tilde{\mathbf{x}}^n \tag{15.2.1}$$

Here the vector $\mathbf{c}$ is a constant and defines a point in the hyperplane and the $\mathbf{b}^i$ define vectors in the hyperplane (also known as 'principal component coefficients' or 'loadings'). The y_i^n are the low dimensional co-ordinates of the data. Equation(15.2.1) expresses how to find the reconstruction $\tilde{\mathbf{x}}^n$ given the lower dimensional representation $\mathbf{y}^n$ (which has components $y_i^n, i = 1, \ldots, M$). For a data space of dimension

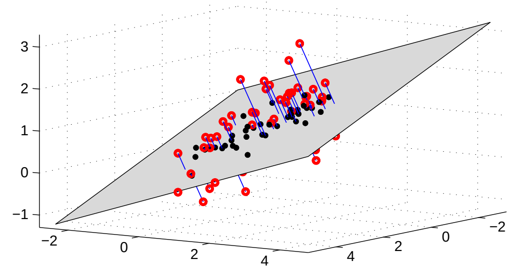


Figure 15.1: In linear dimension reduction a hyperplane is fitted such that the average distance between datapoints (red rings) and their projections onto the plane (black dots) is minimal.

$\dim(\mathbf{x}) = D$, we hope to accurately describe the data using only a small number $M \ll D$ of co-ordinates $\mathbf{y}$.

To determine the best lower dimensional representation it is convenient to use the square distance error between $\mathbf{x}$ and its reconstruction $\tilde{\mathbf{x}}$:

$$E(\mathbf{B}, \mathbf{Y}, \mathbf{c}) = \sum_{n=1}^{N} \sum_{i=1}^{D} [x_i^n - \tilde{x}_i^n]^2 \tag{15.2.2}$$

It is straightforward to show that the optimal bias $\mathbf{c}$ is given by the mean of the data $\sum_n \mathbf{x}^n/N$. We therefore assume that the data has been centred (has zero mean $\sum_n \mathbf{x}^n = \mathbf{0}$), so that we can set $\mathbf{c}$ to zero, and concentrate on finding the optimal basis $\mathbf{B}$ below.

15.2.1 Deriving the optimal linear reconstruction

To find the best basis vectors $\mathbf{B} = \left[\mathbf{b}^1 \ldots, \mathbf{b}^M\right]$ (defining $[B]_{i,j} = b_i^j$) and corresponding low dimensional coordinates $\mathbf{Y} = \left[\mathbf{y}^1, \ldots, \mathbf{y}^N\right]$, we may minimize the sum of squared differences between each vector $\mathbf{x}$ and its reconstruction $\tilde{\mathbf{x}}$:

$$E(\mathbf{B}, \mathbf{Y}) = \sum_{n=1}^{N} \sum_{i=1}^{D} \left[x_i^n - \sum_{j=1}^{M} y_j^n b_i^j \right]^2 = \text{trace}\left((\mathbf{X} - \mathbf{B}\mathbf{Y})^\mathsf{T} (\mathbf{X} - \mathbf{B}\mathbf{Y}) \right) \tag{15.2.3}$$

where $\mathbf{X} = \left[\mathbf{x}^1, \ldots, \mathbf{x}^N\right]$.

Consider a transformation $\mathbf{Q}$ of the basis $\mathbf{B}$ so that $\tilde{\mathbf{B}} \equiv \mathbf{B}\mathbf{Q}$ is an orthonormal matrix, $\tilde{\mathbf{B}}^\mathsf{T}\tilde{\mathbf{B}} = \mathbf{I}$. Since $\mathbf{Q}$ is invertible, we may write $\mathbf{B}\mathbf{Y} = \tilde{\mathbf{B}}\mathbf{Q}^{-1}\mathbf{Y} \equiv \tilde{\mathbf{B}}\tilde{\mathbf{Y}}$, which is of then same form as $\mathbf{B}\mathbf{Y}$, albeit with an orthonormality constraint on $\tilde{\mathbf{B}}$. Hence, without loss of generality, we may consider equation (15.2.3) under the orthonormality constraint $\mathbf{B}^\mathsf{T}\mathbf{B} = \mathbf{I}$, namely that the basis vectors are mutually orthogonal and of unit length.

By differentiating equation (15.2.3) with respect to y_k^n we obtain (using the orthonormality constraint)

$$-\frac{1}{2}\frac{\partial}{\partial y_k^n} E(\mathbf{B}, \mathbf{Y}) = \sum_i \left[x_i^n - \sum_j y_j^n b_i^j \right] b_i^k = \sum_i x_i^n b_i^k - \sum_j y_j^n \underbrace{\sum_i b_i^j b_i^k}_{\delta_{jk}} = \sum_i x_i^n b_i^k - y_k^n$$

The squared error $E(\mathbf{B}, \mathbf{Y})$ therefore has zero derivative when

$$y_k^n = \sum_i b_i^k x_i^n \tag{15.2.4}$$

We now substitute this solution into equation (15.2.3) to write the squared error only as a function of $\mathbf{B}$. Using

$$\sum_j y_j^n b_i^j = \sum_{j,k} b_i^j b_k^j x_k^n = \sum_{j,k} B_{i,j} B_{k,j} x_k^n = [\mathbf{B}\mathbf{B}^\mathsf{T}\mathbf{x}^n]_i \tag{15.2.5}$$

The objective $E(\mathbf{B})$ becomes

$$E(\mathbf{B}) = \sum_n \left(\left(\mathbf{I} - \mathbf{B}\mathbf{B}^\mathsf{T}\right)\mathbf{x}^n\right)^2 \tag{15.2.6}$$

Since $\left(\mathbf{I} - \mathbf{B}\mathbf{B}^\mathsf{T}\right)^2 = \mathbf{I} - \mathbf{B}\mathbf{B}^\mathsf{T}$, (using $\mathbf{B}^\mathsf{T}\mathbf{B} = \mathbf{I}$)

$$E(\mathbf{B}) = \sum_n (\mathbf{x}^n)^\mathsf{T} \left(\mathbf{I} - \mathbf{B}\mathbf{B}^\mathsf{T}\right)\mathbf{x}^n = \text{trace}\left(\sum_n (\mathbf{x}^n)(\mathbf{x}^n)^\mathsf{T}\left(\mathbf{I} - \mathbf{B}\mathbf{B}^\mathsf{T}\right)\right) \tag{15.2.7}$$

Hence the objective becomes

$$E(\mathbf{B}) = (N-1)\left[\text{trace}\left(\mathbf{S}\right) - \text{trace}\left(\mathbf{S}\mathbf{B}\mathbf{B}^\mathsf{T}\right)\right] \tag{15.2.8}$$

where $\mathbf{S}$ is the sample covariance matrix of the data[1]

$$\mathbf{m} = \frac{1}{N}\sum_{n=1}^N \mathbf{x}^n, \quad \mathbf{S} = \frac{1}{N-1}\sum_{n=1}^N (\mathbf{x}^n - \mathbf{m})(\mathbf{x}^n - \mathbf{m})^\mathsf{T} \tag{15.2.9}$$

To minimise equation (15.2.8) under the constraint $\mathbf{B}^\mathsf{T}\mathbf{B} = \mathbf{I}$ we use a set of Lagrange multipliers $\mathbf{L}$, so that the objective is to minimize

$$-\text{trace}\left(\mathbf{S}\mathbf{B}\mathbf{B}^\mathsf{T}\right) + \text{trace}\left(\mathbf{L}\left(\mathbf{B}^\mathsf{T}\mathbf{B} - \mathbf{I}\right)\right) \tag{15.2.10}$$

(neglecting the constant prefactor $N-1$). Since the constraint is symmetric, we can assume that $\mathbf{L}$ is also symmetric. Differentiating with respect to $\mathbf{B}$ and equating to zero we obtain that at the optimum

$$\mathbf{S}\mathbf{B} = \mathbf{B}\mathbf{L} \tag{15.2.11}$$

This is a form of eigen-equation so that a solution is given by taking $\mathbf{L}$ to be diagonal and $\mathbf{B}$ as the matrix whose columns are the corresponding eigenvectors of $\mathbf{S}$. In this case, $\text{trace}\left(\mathbf{S}\mathbf{B}\mathbf{B}^\mathsf{T}\right) = \text{trace}\left(\mathbf{L}\right)$, which is the sum of the eigenvalues corresponding to the eigenvectors forming $\mathbf{B}$. Since we wish to minimise $E(\mathbf{B})$, we take the eigenvectors with largest corresponding eigenvalues.

If we order the eigenvalues $\lambda_1 \geq \lambda_2, \ldots$, the squared error is given by, from equation (15.2.8)

$$\frac{1}{N-1}E(\mathbf{B}) = \text{trace}\left(\mathbf{S}\right) - \text{trace}\left(\mathbf{L}\right) = \sum_{i=1}^D \lambda_i - \sum_{i=1}^M \lambda_i = \sum_{i=M+1}^D \lambda_i \tag{15.2.12}$$

Whilst the solution to this eigen-problem is unique, this only serves to define the solution subspace since one may rotate and scale $\mathbf{B}$ and $\mathbf{Y}$ such that the value of the squared loss is exactly the same. The justification for choosing the non-rotated eigen solution is given by the additional requirement that the principal components corresponds to directions of maximal variance, as explained in section(15.2.2).

[1]Here we use the unbiased sample covariance, simply because this is standard in the literature. If we were to replace this with the sample covariance as defined in chapter(8), the only change required is to replace $N-1$ by N throughout, which has no effect on the form of the solutions found by PCA.

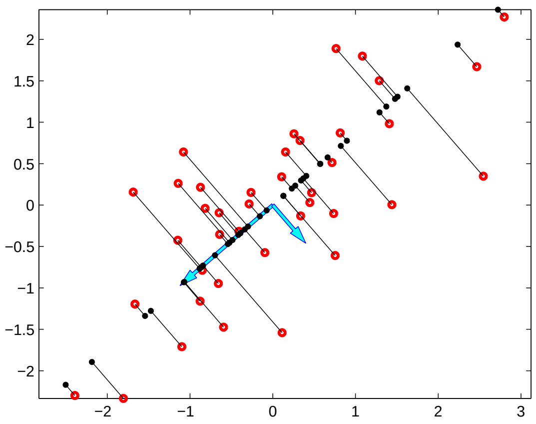


Figure 15.2: Projection of two dimensional data using one dimensional PCA. Plotted are the original datapoints $\mathbf{x}$ (larger rings) and their reconstructions $\tilde{\mathbf{x}}$ (small dots) using 1 dimensional PCA. The lines represent the orthogonal projection of the original datapoint onto the first eigenvector. The arrows are the two eigenvectors scaled by the square root of their corresponding eigenvalues. The data has been centred to have zero mean. For each 'high dimensional' datapoint $\mathbf{x}$, the 'low dimensional' representation $\mathbf{y}$ is given in this case by the distance (possibly negative) from the origin along the first eigenvector direction to the corresponding orthogonal projection point.

15.2.2 Maximum Variance Criterion

We search first for the single direction $\mathbf{b}$ such that, when the data is projected onto this direction, the variance of this projection is maximal amongst all possible such projections. Using equation (15.2.4) for a single vector $\mathbf{b}$ we have

$$y^n = \sum_i b_i x_i^n \tag{15.2.13}$$

The projection of a datapoint onto a direction $\mathbf{b}$ is $\mathbf{b}^\mathsf{T}\mathbf{x}^n$ for a unit length vector $\mathbf{b}$. Hence the sum of squared projections is

$$\sum_n \left(\mathbf{b}^\mathsf{T}\mathbf{x}^n\right)^2 = \mathbf{b}^\mathsf{T}\left[\sum_n \mathbf{x}^n \left(\mathbf{x}^n\right)^\mathsf{T}\right]\mathbf{b} = (N-1)\mathbf{b}^\mathsf{T}\mathbf{S}\mathbf{b} = \lambda(N-1) \tag{15.2.14}$$

which, ignoring constants, is simply the negative of equation (15.2.8) for a single retained eigenvector $\mathbf{b}$ (with $\mathbf{S}\mathbf{b} = \lambda\mathbf{b}$). Hence the optimal single $\mathbf{b}$ which maximises the projection variance is given by the eigenvector corresponding to the largest eigenvalue of $\mathbf{S}$. Under the criterion that the next optimal direction should be orthonormal to the first, one can readily show that $\mathbf{b}^{(2)}$ is given by the second largest eigenvector, and so on. This explains why, despite the squared loss equation (15.2.8) being invariant with respect to arbitrary rotation (and scaling) of the basis vectors, the ones given by the eigen-decomposition have the additional property that they correspond to directions of maximal variance. These maximal variance directions found by PCA are called the *principal directions*.

15.2.3 PCA algorithm

The routine for PCA is presented in algorithm(14). In the notation of $\mathbf{y} = \mathbf{F}\mathbf{x}$, the projection matrix $\mathbf{F}$ corresponds to $\mathbf{E}^\mathsf{T}$. Similarly for the reconstruction equation (15.2.1), the coordinate $\mathbf{y}^n$ corresponds to $\mathbf{E}^\mathsf{T}\mathbf{x}^n$ and $\mathbf{b}^i$ corresponds to $\mathbf{e}^i$. The PCA reconstructions are orthogonal projections of the data onto the subspace spanned by the eigenvectors corresponding to the M largest eigenvalues of the covariance matrix, see fig(15.2).

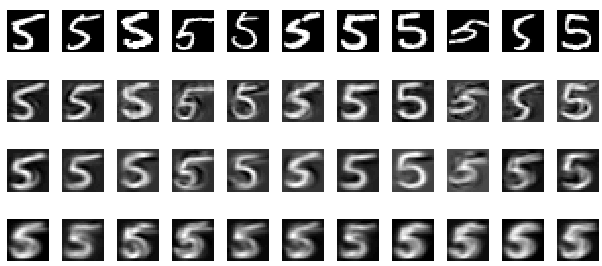


Figure 15.3: Top row : a selection of the digit 5 taken from the database of 892 examples. Plotted beneath each digit is the reconstruction using 100, 30 and 5 eigenvectors (from top to bottom). Note how the reconstructions for fewer eigenvectors express less variability from each other, and resemble more a mean 5 digit.

Algorithm 14 Principal Components Analysis to form an M-dimensional approximation of a dataset $\{\mathbf{x}^n, n = 1, \ldots, N\}$, with $\dim x^n = D$.

1: Find the $D \times 1$ sample mean vector and $D \times D$ covariance matrix

$$\mathbf{m} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}^n, \quad \mathbf{S} = \frac{1}{N-1}\sum_{n=1}^{N}(\mathbf{x}^n - \mathbf{m})(\mathbf{x}^n - \mathbf{m})^{\mathsf{T}}$$

2: Find the eigenvectors $\mathbf{e}^1, \ldots, \mathbf{e}^M$ of the covariance matrix $\mathbf{S}$, sorted so that the eigenvalue of $\mathbf{e}^i$ is larger than $\mathbf{e}^j$ for $i < j$. Form the matrix $\mathbf{E} = [\mathbf{e}^1, \ldots, \mathbf{e}^M]$.

3: The lower dimensional representation of each data point $\mathbf{x}^n$ is given by

$$\mathbf{y}^n = \mathbf{E}^{\mathsf{T}}(\mathbf{x}^n - \mathbf{m}) \tag{15.2.15}$$

4: The approximate reconstruction of the original datapoint $\mathbf{x}^n$ is

$$\mathbf{x}^n \approx \mathbf{m} + \mathbf{E}\mathbf{y}^n \tag{15.2.16}$$

5: The total squared error over all the training data made by the approximation is

$$\sum_{n=1}^{N}(\mathbf{x}^n - \tilde{\mathbf{x}}^n)^2 = (N-1)\sum_{j=M+1}^{D}\lambda_j \tag{15.2.17}$$

where $\lambda_{M+1} \ldots \lambda_N$ are the eigenvalues discarded in the projection.

Example 68 (Reducing the dimension of digits)**.** We have 892 examples of handwritten 5's, where each image consists of 28×28 real-values pixels, see fig(15.3). Each image matrix is stacked to form a 784 dimensional vector, giving a 784×892 dimensional data matrix $\mathbf{X}$. The covariance matrix of this data has eigenvalue spectrum as plotted in fig(15.4), where we plot only the 100 largest eigenvalues. Note how after around 40 components, the mean squared reconstruction error is small, indicating that the data indeed lie close to a 40 dimensional hyperplane. The eigenvalues are computed using `pca.m`.

The reconstructions using different numbers of eigenvectors (100, 30 and 5) are plotted in fig(15.3). Note how using only a small number of eigenvectors, the reconstruction more closely resembles the mean image.

Example 69 (Eigenfaces)**.** In fig(15.5) we present example images for which we wish to find a lower dimensional representation. Using PCA the first 49 'eigenfaces' are presented along with reconstructions of the original data using these eigenfaces, see fig(15.6).

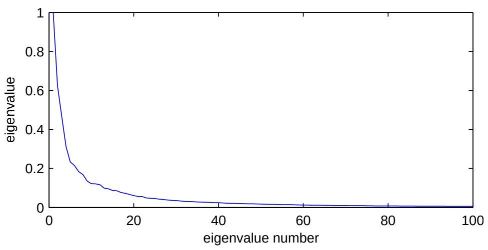
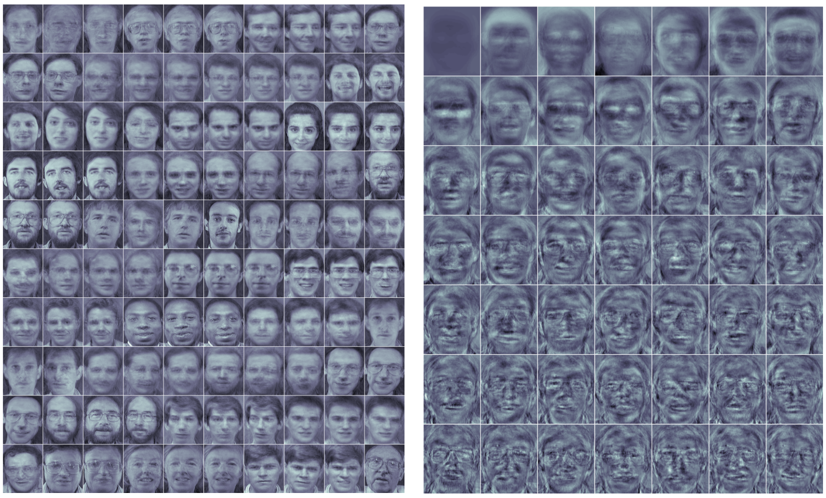


Figure 15.4: For the digits data consisting of 892 examples of the digit 5, each image being represented by a 784 dimensional vector. Plotted as the largest 100 eigenvalues (scaled so that the largest eigenvalue is 1) of the sample covariance matrix.

Figure 15.5: 100 of the 120 training images (40 people, with 3 images of each person). Each image consists of $92 \times 112 = 10304$ greyscale pixels. The training data is scaled so that, represented as an image, the components of each image sum to 1. The average value of each pixel across all images is 9.70×10^{-5}. This is a subset of the 400 images in the full Olivetti Research Face Database.



(a) (b)

Figure 15.6: **(a)**: SVD reconstruction of the images in fig(15.5) using a combination of the 49 eigen-images. **(b)**: The eigen-images are found using SVD of the fig(15.5) and taking the 49 eigenvectors with largest eigenvalue. The images corresponding to the largest eigenvalues are contained in the first row, and the next 7 in the row below, *etc.* The root mean square reconstruction error is 1.121×10^{-5}, a small improvement over PLSA (see fig(15.14)).

15.2.4 PCA and Nearest Neighbours

For high-dimensional data computing the squared Euclidean distance between vectors can be expensive, and also sensitive to noise. It is therefore often useful to project the data to form a lower dimensional representation first. For example, in making a classifier to distinguish between the digit 1 and the digit 7, example(67), we can form a lower dimensional representation first by ignoring the class label (to make a dataset of 1200 training points). Each of the training points $\mathbf{x}^n$ is then projected to a lower dimensional PCA representation $\mathbf{y}^n$. Subsequently, any distance calculations $(\mathbf{x}^a - \mathbf{x}^b)^2$ are replaced by $(\mathbf{y}^a - \mathbf{y}^b)^2$. To justify this, consider

$$\begin{aligned}(\mathbf{x}^a - \mathbf{x}^b)^\mathsf{T}(\mathbf{x}^a - \mathbf{x}^b) &\approx (\mathbf{E}\mathbf{y}^a + \mathbf{m} - \mathbf{E}\mathbf{y}^b - \mathbf{m})^\mathsf{T}(\mathbf{E}\mathbf{y}^a + \mathbf{m} - \mathbf{E}\mathbf{y}^b - \mathbf{m}) \\ &= (\mathbf{y}^a - \mathbf{y}^b)^\mathsf{T}\mathbf{E}^\mathsf{T}\mathbf{E}(\mathbf{y}^a - \mathbf{y}^b) \\ &= (\mathbf{y}^a - \mathbf{y}^b)^\mathsf{T}(\mathbf{y}^a - \mathbf{y}^b) \end{aligned} \tag{15.2.18}$$

where the last equality is due to the orthonormality of eigenvectors : $\mathbf{E}^\mathsf{T}\mathbf{E} = \mathbf{I}$.

Using 19 principal components (see example(70) why this number was chosen) and the nearest neighbour rule to classify ones and sevens gave a test-set error of 14 in 600 examples, compared to 18 from the standard method on the non-projected data. How can it be that the classification performance has improved? A plausible explanation is that the new PCA representation of the data is more robust since only the large scale change directions in the space are retained, with low variance directions discarded.

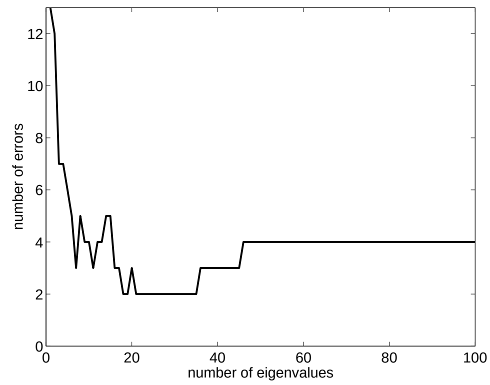


Figure 15.7: Finding the optimal PCA dimension to use for classifying hand-written digits using nearest neighbours. 400 training examples are used, and the validation error plotted on 200 further examples. Based on the validation error, we see that a dimension of 19 is reasonable.

Example 70 (Finding the best PCA dimension)**.** There are 600 examples of the digit 1 and 600 examples of the digit 7. We will use half the data for training and the other half for testing. The 600 training examples were further split into a training set of 400 examples, and a separate validation set of 200 examples. PCA was used to reduce the dimensionality of the inputs, and then nearest neighbours used to classify the 200 validation examples. Different reduced dimensions were in vestigated and, based on the validation results, 19 was selected as the optimal number of PCA components retained, see fig(15.7). The independent test error on 600 independent examples using 19 dimensions is 14.

15.2.5 Comments on PCA

The 'intrinsic' dimension of data

How many dimensions should the linear subspace have? From equation (15.2.12), the reconstruction error is proportional to the sum of the discarded eigenvalues. If we plot the eigenvalue spectrum (the set of eigenvalues ordered by decreasing value), we might hope to see a few large values and many small values. If the data does lie close to an M dimensional hyperplane, we would see M large eigenvalues with the rest being very small. This would give an indication of the number of degrees of freedom in the data, or the intrinsic dimensionality. Directions corresponding to the small eigenvalues are then interpreted as 'noise'.

Non-linear dimension reduction

In PCA we are presupposing that the data lies close to a hyperplane. Is this really a good description? More generally, we would expect data to lie on low dimensional curved manifolds. Also, data is often clustered – examples of handwritten '4's look similar to each other and form a cluster, separate from the '8's cluster. Nevertheless, since linear dimension reduction is computationally relatively straightforward, this is one of the most common dimensionality reduction techniques.

15.3 High Dimensional Data

The computational complexity of computing an eigen-decomposition of a $D \times D$ matrix is $O\left(D^3\right)$. You might be wondering therefore how it is possible to perform PCA on high dimensional data. For example, if we have 500 images each of $1000 \times 1000 = 10^6$ pixels, the covariance matrix will be $10^6 \times 10^6$ dimensional. It would appear a significant computational challenge to compute the eigen-decomposition of this matrix. In this case, however, since there are only 500 such vectors, the number of non-zero eigenvalues cannot exceed 500. One can exploit this fact to bound the complexity by $O\left(\min(D, N)^3\right)$, as described below.

15.3.1 Eigen-decomposition for $N < D$

First note that for zero mean data, the sample covariance matrix can be expressed as

$$[\mathbf{S}]_{ij} = \frac{1}{N-1}\sum_{n=1}^{N} x_i^n x_j^n \tag{15.3.1}$$

In matrix notation this can be written

$$\mathbf{S} = \frac{1}{N-1}\mathbf{X}\mathbf{X}^{\mathsf{T}} \tag{15.3.2}$$

where the $D \times N$ matrix $\mathbf{X}$ contains all the data vectors:

$$\mathbf{X} = \left[\mathbf{x}^1, \ldots, \mathbf{x}^N\right] \tag{15.3.3}$$

Since the eigenvectors of a matrix $\mathbf{M}$ are equal to those of $\gamma\mathbf{M}$ for scalar γ, one can consider more simply the eigenvectors of $\mathbf{X}\mathbf{X}^{\mathsf{T}}$. Writing the $D \times N$ matrix of eigenvectors as $\mathbf{E}$ (this is a non-square thin matrix since there will be fewer eigenvalues than data dimensions) and the eigenvalues as an $N \times N$ diagonal matrix $\mathbf{\Lambda}$, the eigen-decomposition of the covariance $\mathbf{S}$ is

$$\mathbf{X}\mathbf{X}^{\mathsf{T}}\mathbf{E} = \mathbf{E}\mathbf{\Lambda} \Rightarrow \mathbf{X}^{\mathsf{T}}\mathbf{X}\mathbf{X}^{\mathsf{T}}\mathbf{E} = \mathbf{X}^{\mathsf{T}}\mathbf{E}\mathbf{\Lambda} \Rightarrow \mathbf{X}^{\mathsf{T}}\mathbf{X}\tilde{\mathbf{E}} = \tilde{\mathbf{E}}\mathbf{\Lambda} \tag{15.3.4}$$

where we defined $\tilde{\mathbf{E}} = \mathbf{X}^{\mathsf{T}}\mathbf{E}$. The final expression above represents the eigenvector equation for $\mathbf{X}^{\mathsf{T}}\mathbf{X}$. This is a matrix of dimensions $N \times N$ so that calculating the eigen-decomposition takes $O\left(N^3\right)$ operations, compared with $O\left(D^3\right)$ operations in the original high-dimensional space . We then can therefore calculate the eigenvectors $\tilde{\mathbf{E}}$ and eigenvalues $\mathbf{\Lambda}$ of this matrix more easily. Once found, we use the fact that the eigenvalues of $\mathbf{S}$ are given by the diagonal entries of $\mathbf{\Lambda}$ and the eigenvectors by

$$\mathbf{E} = \mathbf{X}\tilde{\mathbf{E}}\mathbf{\Lambda}^{-1} \tag{15.3.5}$$

15.3.2 PCA via Singular value decomposition

An alternative to using an eigen-decomposition routine to find the PCA solution is to make use of the *Singular Value Decomposition* (SVD) of an $D \times N$ dimensional matrix $\mathbf{X}$. This is given by

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^{\mathsf{T}} \tag{15.3.6}$$

where $\mathbf{U}^{\mathsf{T}}\mathbf{U} = \mathbf{I}_D$ and $\mathbf{V}^{\mathsf{T}}\mathbf{V} = \mathbf{I}_N$ and $\mathbf{D}$ is a diagonal matrix of the (positive) singular values. We assume that the decomposition has ordered the singular values so that the upper left diagonal element of $\mathbf{D}$ contains the largest singular value. The matrix $\mathbf{X}\mathbf{X}^{\mathsf{T}}$ can then be written as

$$\mathbf{X}\mathbf{X}^{\mathsf{T}} = \mathbf{U}\mathbf{D}\mathbf{V}^{\mathsf{T}}\mathbf{V}\mathbf{D}\mathbf{U}^{\mathsf{T}} = \mathbf{U}\mathbf{D}^2\mathbf{U}^{\mathsf{T}} \tag{15.3.7}$$

Since $\mathbf{U}\mathbf{D}^2\mathbf{U}^{\mathsf{T}}$ is in the form of an eigen-decomposition, the PCA solution is equivalently given by performing the SVD decomposition of $\mathbf{X}$, for which the eigenvectors are then given by $\mathbf{U}$, and corresponding eigenvalues by the square of the singular values.

Equation(15.3.6) shows that PCA is a form of matrix decomposition method:

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^{\mathsf{T}} \approx \mathbf{U}_M\mathbf{D}_M\mathbf{V}_M^{\mathsf{T}} \tag{15.3.8}$$

where $\mathbf{U}_M$, $\mathbf{D}_M$, $\mathbf{V}_M$ correspond to taking only the first M singular values of the full matrices.

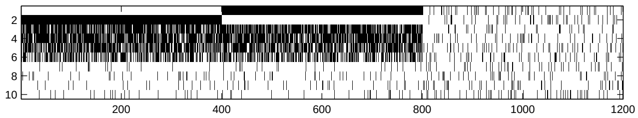


Figure 15.8: Document data for a dictionary containing 10 words and 1200 documents. Black indicates that a word was present in a document. The data consists of two 'similar' topics (differing only in their usage of the first two words) and a random background topic.

15.4 Latent Semantic Analysis

In the document analysis literature PCA is also called Latent Semantic Analysis and is concerned with analysing a set of N documents, where each document d^n, $n = 1, \ldots, N$ is represented by a vector

$$\mathbf{x}^n = (x_1^n, \ldots, x_D^n) \tag{15.4.1}$$

of word occurrences. For example the first element x_1^n might count how many times the word 'cat' appears in document n, x_2^n the number of occurrences of 'dog', *etc.* This *bag of words* bag of words description[2] is formed by first choosing a dictionary of D words. The vector element x_i^n is the (possibly normalised) number of occurrences of the word i in the document n. Typically D will be large, of the order 10^6, and $\mathbf{x}$ will be very sparse since any document contains only a small fraction of the available words in the dictionary. Given a set of documents $\mathcal{D}$, the aim in LSA is to form a lower dimensional representation of each document. The whole document database is represented by the so-called *term-document matrix*

$$\mathbf{X} = \left[\mathbf{x}^1, \ldots, \mathbf{x}^N\right] \tag{15.4.2}$$

which has dimension $D \times N$, see for example fig(15.8). In this small example the term-document matrix is 'short and fat', whereas in practice most often the matrix will be 'tall and thin'.

Example 71 (Latent Topic)**.** We have a small dictionary containing the words *influenza, flu, headache, nose, temperature, bed, cat, tree, car, foot*. The database contains a large number of articles that discuss ailments, and articles which seem to talk about the effects of influenza, in addition to some background documents that are not specific to ailments. Some of the more formal documents exclusively use the term *influenza*, whereas the other more 'tabloid' documents use the informal term *flu*. Each document is represented by a simple bag-of-words style description, namely a 10-dimensional vector in which element i of that vector is set to 1 if word i occurs in the document, and 0 otherwise. The data is represented in fig(15.8). The data is generated using the artificial mechanism described in `demoLSI.m`.

The result of using PCA on this data is represented in fig(15.9) where we plot the eigenvectors, scaled by their eigenvalue. The first eigenvector groups all the 'influenza' words together, and the second deals with the different usage of the terms *influenza* and *flu*.

Rescaling

In LSA it is common to scale the transformation so that the projected vectors have approximately unit covariance (assuming centred data). Using

$$\mathbf{y} = \sqrt{N-1}\mathbf{D}_M^{-1}\mathbf{U}_M^{\mathsf{T}}\mathbf{x} \tag{15.4.3}$$

the covariance of the projections is obtained from

$$\frac{1}{N-1}\sum_n \mathbf{y}^n \left(\mathbf{y}^n\right)^{\mathsf{T}} = \mathbf{D}_M^{-1}\mathbf{U}_M^{\mathsf{T}} \underbrace{\sum_n \mathbf{x}^n \left(\mathbf{x}^n\right)^{\mathsf{T}}}_{\mathbf{X}\mathbf{X}^{\mathsf{T}}} \mathbf{U}_M\mathbf{D}_M^{-1} = \mathbf{D}_M^{-1}\mathbf{U}_M^{\mathsf{T}}\mathbf{U}\mathbf{D}^2\mathbf{U}^{\mathsf{T}}\mathbf{U}_M\mathbf{D}_M^{-1} \approx \mathbf{I}$$

[2] More generally one can consider term-counts, in which terms can can be single words, or sets of words, or even sub-words.

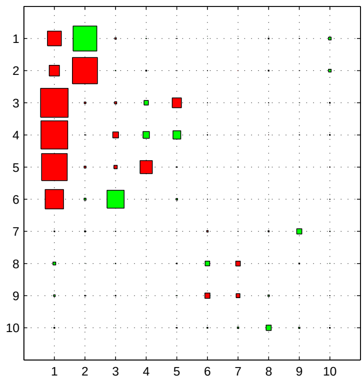


Figure 15.9: Hinton Diagram of the eigenvector matrix $\mathbf{E}$ where each eigenvector column is scaled by the corresponding eigenvalue. Red indicates positive and green negative (the area of each square corresponds to the magnitude), showing that there are only a few large eigenvalues. Note that the overall sign of any eigenvector is irrelevant. The first eigenvector corresponds to a topic in which the words *influenza, flu, headache, nose, temperature, bed* are prevalent. The second eigenvector denotes that there is negative correlation between the occurrence of *influenza* and *flu*.

Given $\mathbf{y}$, the approximate reconstruction $\tilde{\mathbf{x}}$ is

$$\tilde{\mathbf{x}} = \frac{1}{\sqrt{N-1}}\mathbf{U}_M\mathbf{D}_M\mathbf{y} \tag{15.4.4}$$

The Euclidean distance between two points $\mathbf{x}^a$ and $\mathbf{x}^b$ is then approximately

$$d\left(\tilde{\mathbf{x}}^a, \tilde{\mathbf{x}}^b\right) = \frac{1}{N-1}\left(\mathbf{y}^a - \mathbf{y}^b\right)^{\mathsf{T}}\mathbf{D}_M\mathbf{U}_M^{\mathsf{T}}\mathbf{U}_M\mathbf{D}_M\left(\mathbf{y}^a - \mathbf{y}^b\right) \approx \frac{1}{N-1}\left(\mathbf{y}^a - \mathbf{y}^b\right)^{\mathsf{T}}\mathbf{D}_M^2\left(\mathbf{y}^a - \mathbf{y}^b\right)$$

It is common to ignore the $\mathbf{D}_M^2$ term (and $1/(N-1)$ factor), and consider a measure of dissimilarity in the projected space just to be the Euclidean distance between the $\mathbf{y}$ vectors.

15.4.1 LSA for information retrieval

Consider a large collection of documents from the web, creating a database $\mathcal{D}$. Our interest it to find the most similar document to a specified query document. Using a bag-of-words style representation for document n, $\mathbf{x}^n$, and similarly for the query document, $\mathbf{x}^*$ we address this task by first defining a measure of dissimilarity between documents, for example

$$d(\mathbf{x}^n, \mathbf{x}^m) = (\mathbf{x}^n - \mathbf{x}^m)^{\mathsf{T}}(\mathbf{x}^n - \mathbf{x}^m) \tag{15.4.5}$$

One then searches for the document that minimises this dissimilarity:

$$n_{opt} = \underset{n}{\operatorname{argmin}}\ d(\mathbf{x}^n, \mathbf{x}^*) \tag{15.4.6}$$

and returns document $\mathbf{x}^{n_{opt}}$ as the result of the search query. A difficulty with this approach is that the bag-of-words representation will have mostly zeros (*i.e.* be very sparse). Hence differences may be due to 'noise' rather than any real similarity between the query and database document. LSA alleviates this problem by using a lower dimensional representation $\mathbf{y}$ of the high-dimensional $\mathbf{x}$. The $\mathbf{y}$ capture the main variations in the data and are less sensitive to random uncorrelated noise. Using the dissimilarity defined in terms of the lower dimensional $\mathbf{y}$ is therefore more robust and likely to retrieve more useful documents.

The squared difference between two documents can also be written

$$\left(\mathbf{x} - \mathbf{x}'\right)^{\mathsf{T}}\left(\mathbf{x} - \mathbf{x}'\right) = \mathbf{x}^{\mathsf{T}}\mathbf{x} + \mathbf{x}'^{\mathsf{T}}\mathbf{x}' - 2\mathbf{x}^{\mathsf{T}}\mathbf{x}' \tag{15.4.7}$$

If, as is commonly done, the bag-of-words representations are scaled to have unit length,

$$\hat{\mathbf{x}} = \frac{\mathbf{x}}{\sqrt{\mathbf{x}^{\mathsf{T}}\mathbf{x}}} \tag{15.4.8}$$

so that $\hat{\mathbf{x}}^{\mathsf{T}}\hat{\mathbf{x}} = 1$, the distance is

$$\left(\hat{\mathbf{x}} - \hat{\mathbf{x}}'\right)^{\mathsf{T}}\left(\hat{\mathbf{x}} - \hat{\mathbf{x}}'\right) = 2\left(1 - \hat{\mathbf{x}}^{\mathsf{T}}\hat{\mathbf{x}}'\right) \tag{15.4.9}$$

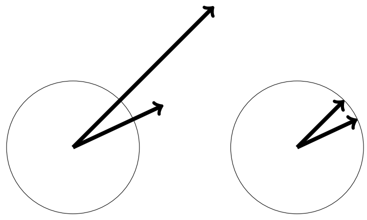


Figure 15.10: **(a)**: Two bag-of-word vectors. The Euclidean distance between the two is large. **(b)**: Normalised vectors. The Euclidean distance is now related directly to the angle between the vectors. In this case two documents which have the same relative frequency of words will both have the same dissimilarly, even though the number of occurrences of the words is different.

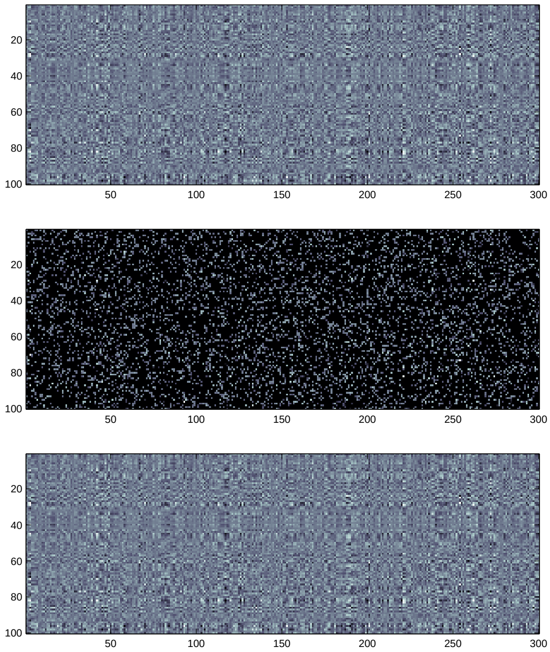


Figure 15.11: Top: original data matrix $\mathbf{X}$. Black is missing, white present. The data is constructed from a set of only 5 basis vectors. Middle : $\mathbf{X}$ with missing data (80% sparsity). Bottom : reconstruction found using `svdm.m`, SVD for missing data. This problem is essentially easy since, despite there being many missing elements, the data is indeed constructed from a model for which SVD is appropriate. Such techniques have application in collaborative filtering and recommender systems where one wishes to 'fill in' missing values in a matrix.

and one may equivalently consider the *cosine similarity*

$$s(\hat{\mathbf{x}}, \hat{\mathbf{x}}') = \hat{\mathbf{x}}^{\mathsf{T}}\hat{\mathbf{x}}' = \cos(\theta) \tag{15.4.10}$$

where θ is the angle between the unit vectors $\hat{\mathbf{x}}$ and $\hat{\mathbf{x}'}$, fig(15.10).

PCA is arguably suboptimal for document analysis since we would expect the presence of a latent topic to contribute only positive counts to the data. A related version of PCA in which the decomposition is constrained to have positive elements is called PLSA, and discussed in section(15.6).

Example 72. Continuing the Influenza example, someone who uploads a query document which uses the term 'flu' might also be interested in documents about 'influenza'. However, the search query term 'flu' does not contain the word 'influenza', so how can one retrieve such documents? Since the first component using PCA (LSA) groups all 'influenza' terms together, if we use only the first component of the representation $\mathbf{y}$ to compare documents, this will retrieve documents independent of whether the term 'flu' or 'influenza' is used.

15.5 PCA with missing data

When values of the data matrix $\mathbf{X}$ are missing, the standard PCA algorithm as described cannot be implemented. Unfortunately, there is no 'quick fix' PCA solution when some of the x_i^n are missing and more complex numerical procedures need to invoked. A naive approach in this case is to require the squared reconstruction error to be small only for the existing elements of $\mathbf{X}$. That is

$$E(\mathbf{B},\mathbf{Y}) = \sum_{n=1}^{N}\sum_{i=1}^{D}\gamma_i^n\left[x_i^n - \sum_j y_j^n b_i^j\right]^2 \tag{15.5.1}$$

where $\gamma_i^n = 1$ if the i^{th} entry of the n^{th} vector is available, and is zero otherwise. Differentiating, as before, we find that the optimal weights satisfy (assuming $\mathbf{B}^\mathsf{T}\mathbf{B} = \mathbf{I}$),

$$y_n^k = \sum_i \gamma_i^n x_i^n b_i^k \tag{15.5.2}$$

One then substitutes this expression into the squared error, and minimises the error with respect to $\mathbf{B}$ under the orthonormality constraint. An alternative iterative optimisation procedure is as follows: First select a random $D \times M$ matrix $\hat{\mathbf{B}}$. Then iterate until convergence the following two steps:

Optimize Y for fixed B

$$E(\hat{\mathbf{B}},\mathbf{Y}) = \sum_{n=1}^{N}\sum_{i=1}^{D}\gamma_i^n\left[x_i^n - \sum_j y_j^n \hat{b}_i^j\right]^2 \tag{15.5.3}$$

For fixed $\hat{\mathbf{B}}$ the above $E(\hat{\mathbf{B}},\mathbf{Y})$ is a quadratic function of the matrix $\mathbf{Y}$, which can be optimised directly. By differentiating and equating to zero, one obtains the fixed point condition

$$\sum_i \gamma_i^n\left(x_i^n - \sum_l y_l^n \hat{b}_i^l\right)\hat{b}_i^k = 0 \tag{15.5.4}$$

Defining

$$\left[\mathbf{w}^{(n)}\right]_l = y_n^l, \qquad \left[\mathbf{M}^{(n)}\right]_{kl} = \sum_i \hat{b}_i^l \hat{b}_i^k \gamma_i^n, \qquad [\mathbf{c}^n]_k = \sum_i \gamma_i^n x_i^n \hat{b}_i^k, \tag{15.5.5}$$

in matrix notation, we then have a set of linear systems:

$$\mathbf{c}^{(n)} = \mathbf{M}^{(n)}\mathbf{y}^{(n)}, \qquad n = 1,\ldots,N \tag{15.5.6}$$

One may solve each linear system using Gaussian elimination (one can avoid explicit matrix inversion by using the \ operator in MATLAB). It can be that one or more of the above linear systems is underdetermined– this can occur when there are less observed values in the n^{th} data column of $\mathbf{X}$ than there are components M. In this case one may use the pseudo-inverse to provide a minimal length solution.

Optimize B for fixed Y

One now freezes $\hat{\mathbf{Y}}$ and considers the function

$$E(\mathbf{B},\hat{\mathbf{Y}}) = \sum_{n=1}^{N}\sum_{i=1}^{D}\gamma_i^n\left[x_i^n - \sum_j \hat{y}_j^n b_i^j\right]^2 \tag{15.5.7}$$

For fixed $\hat{\mathbf{Y}}$ the above expression is quadratic in the matrix $\mathbf{B}$, which can again be optimised using linear algebra. This corresponds to solving a set of linear systems for the i^{th} row of $\mathbf{B}$:

$$\mathbf{m}^{(i)} = \mathbf{F}^{(i)}\mathbf{b}^{(i)} \tag{15.5.8}$$

where

$$\left[\mathbf{m}^{(i)}\right]_k = \sum_n \gamma_i^n x_i^n \hat{y}_k^n, \qquad \left[\mathbf{F}^{(i)}\right]_{kj} = \sum_n \gamma_i^n \hat{y}_j^n \hat{y}_k^n \tag{15.5.9}$$

Mathematically, this is $\mathbf{b}^{(i)} = \mathbf{F}^{(i)^{-1}}\mathbf{m}^{(i)}$.

In this manner one is guaranteed to iteratively decrease the value of the squared error loss until a minimum is reached. This technique is implemented in `svdm.m`. Note that efficient techniques based on updating the solution as a new column of $\mathbf{X}$ arrives one at a time ('online' updating) are available, see for example [49].

15.5.1 Finding the principal directions

For the missing data case the basis $\mathbf{B}$ found using the above technique is based only on minimising the squared reconstruction error and therefore does not necessarily satisfy the maximal variance (or principal directions) criterion, namely that the columns of $\mathbf{B}$ point along the eigen-directions. For a given $\mathbf{B}$, $\mathbf{Y}$ with approximate decomposition $\mathbf{X} \approx \mathbf{BY}$ we can return a new orthonormal basis $\mathbf{U}$ by performing SVD on the completed data, $\mathbf{BY} = \mathbf{USV}^\mathsf{T}$ to return an orthonormal basis $\mathbf{B} \to \mathbf{U}$. In general, however, this is potentially computationally expensive. If only the principal directions are required, an alternative is to explicitly transform the solution $\mathbf{B}$ using an invertible matrix $\mathbf{Q}$:

$$\mathbf{X} \approx \mathbf{BQQ}^{-1}\mathbf{Y} \tag{15.5.10}$$

Calling the new basis $\tilde{\mathbf{B}} = \mathbf{BQ}$, for a solution to be aligned with the principal directions, we need

$$\tilde{\mathbf{B}}^\mathsf{T}\tilde{\mathbf{B}} = \mathbf{I} \tag{15.5.11}$$

In other words

$$\mathbf{Q}^\mathsf{T}\mathbf{B}^\mathsf{T}\mathbf{BQ} = \mathbf{I} \tag{15.5.12}$$

Forming the SVD of $\mathbf{B}$,

$$\mathbf{B} = \mathbf{UDV}^\mathsf{T} \tag{15.5.13}$$

and substituting in equation (15.5.12), we have the requirement

$$\mathbf{Q}^\mathsf{T}\mathbf{VDU}^\mathsf{T}\mathbf{UDV}^\mathsf{T}\mathbf{Q} = \mathbf{I} \tag{15.5.14}$$

Since $\mathbf{U}^\mathsf{T}\mathbf{U} = \mathbf{I}$ and $\mathbf{V}^\mathsf{T}\mathbf{V} = \mathbf{VV}^\mathsf{T} = \mathbf{I}$, we can use

$$\mathbf{Q} = \mathbf{V}^\mathsf{T}\mathbf{D}^{-1} \tag{15.5.15}$$

Hence, given a solution $\mathbf{B}$, we can find the principal directions from the SVD of $\mathbf{B}$ using

$$\tilde{\mathbf{B}} = \mathbf{U}^\mathsf{T}\mathbf{D}^{-1}\mathbf{B} \tag{15.5.16}$$

If the $D \times M$ matrix $\mathbf{B}$ is non-square $M < D$, then the matrix $\mathbf{D}$ will be non-square and non-invertible. To make the above well defined, one may append $\mathbf{D}$ with the columns of the identity:

$$\mathbf{D}' = [\mathbf{D}, \mathbf{I}_{M+1}, \ldots, \mathbf{I}_D] \tag{15.5.17}$$

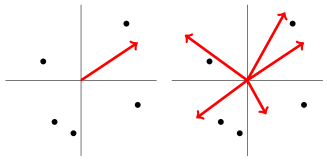
where $\mathbf{I}_K$ is the K^{th} column of the identity matrix, and use $\mathbf{D}'$ in place of $\mathbf{D}$ above.

15.5.2 Collaborative Filtering using PCA with missing data

A database contains a set of vectors, each describing the film ratings for a user in the database. The i^{th} entry in the vector $\mathbf{x}$ specifies the rating the user gives to the i^{th} film. The matrix $\mathbf{X} = \left[\mathbf{x}^1, \ldots, \mathbf{x}^N\right]$ for all the N users, has many missing values since any single user will only have given a rating for a small selection of the possible D films. In a practical example one might have $D = 10,000$ films and $N = 1,000,000$ users. For any user n the task is to predict reasonable values for the missing entries of their rating vector $\mathbf{x}^n$, thereby providing a suggestion as to which films they might like to view. Viewed as a missing data problem, one can fit $\mathbf{B}$ and $\mathbf{Y}$ using `svdm.m` as above. Given $\mathbf{B}$ and $\mathbf{Y}$ we can form a reconstruction on all the entries of $\mathbf{X}$, by using

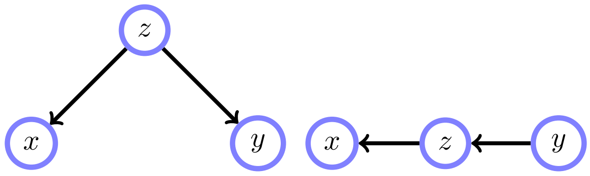
$$\tilde{\mathbf{X}} = \mathbf{BY} \tag{15.5.18}$$

giving therefore a prediction for the missing values.



(a) (b)

Figure 15.12: **(a)**: Under-complete representation. There are too few basis vectors to represent the datapoints. **(b)**: Over-complete representation. There are too many basis vectors to form a unique representation of a datapoint in terms of a linear combination of the basis vectors.



(a) (b)

Figure 15.13: **(a)**: Joint PLSA. **(b)**: Conditional PLSA. Whilst written as a graphical model, some care is required in the interpretation, see text.

15.6 Matrix decomposition methods

Given a data matrix $\mathbf{X}$ for which each column represents a datapoint, an approximate matrix decomposition is of the form $\mathbf{X} \approx \mathbf{BY}$ into a basis matrix $\mathbf{B}$ and weight (or coordinate) matrix $\mathbf{Y}$. Symbolically, matrix decompositions are of the form

$$\underbrace{\left(\quad X : \text{Data} \quad \right)}_{D\times N} \approx \underbrace{\left(\quad B : \text{Basis} \quad \right)}_{D\times M} \underbrace{\left(\quad Y : \text{Weights/Components} \quad \right)}_{M\times N} \tag{15.6.1}$$

In this section we will consider some common matrix decomposition methods.

Under-complete decompositions

When $M < D$, there are fewer basis vectors than dimensions, fig(15.12a). The matrix $\mathbf{B}$ is then called 'tall' or 'thin'. In this case the matrix $\mathbf{Y}$ forms a lower dimensional approximate representation of the data $\mathbf{X}$, PCA being a classic example.

Over-complete decompositions

For $M > D$ the basis is over-complete, there being more basis vectors than dimensions, fig(15.12b). In such cases additional constraints are placed on either the basis or components. For example, one might require that only a small number of the large number of available basis vectors is used to form the representation for any given $\mathbf{x}$. Such sparse-representations are common in theoretical neurobiology where issues of energy efficiency, rapidity of processing and robustness are of interest[212, 152, 249].

15.6.1 Probabilistic Latent Semantic Analysis

Consider two objects, x and y, where $\text{dom}(x) = \{1, \ldots, I\}$ and $\text{dom}(y) = \{1, \ldots, J\}$. We have a count matrix with elements C_{ij} which describes the number of times that $x = i, y = j$ was observed. We can transform this count matrix into a 'frequency' matrix p with elements

$$p(x = i, y = j) = \frac{C_{ij}}{\sum_{ij} C_{ij}} \tag{15.6.2}$$

Our interest is to find a decomposition of this frequency matrix of the form in fig(15.13a)

$$\underbrace{p(x = i, y = j)}_{X_{ij}} \approx \sum_k \underbrace{\tilde{p}(x = i|z = k)}_{B_{ik}} \underbrace{\tilde{p}(y = j|z = k)\tilde{p}(z = k)}_{Y_{kj}} \equiv \tilde{p}(x = i, y = j) \tag{15.6.3}$$

which is a form of matrix decomposition into basis $\mathbf{B}$ and coordinates $\mathbf{Y}$. This has the interpretation of discovering latent topics z that describe the joint behaviour of x and y.

An EM style training algorithm

In order to find the approximate decomposition we first need a measure of difference between the matrix with elements p_{ij} and the approximation with elements $\tilde{p}_{ij}$. Since all elements are bounded between 0 and 1 and sum to 1, we may interpret p as a joint probability and $\tilde{p}$ as an approximation to this. For probabilities, a useful measure of discrepancy is the Kullback-Leibler divergence

$$\mathrm{KL}(p|\tilde{p}) = \langle \log p \rangle_p - \langle \log \tilde{p} \rangle_p \tag{15.6.4}$$

Since p is fixed, minimising the Kullback-Leibler divergence with respect to the approximation $\tilde{p}$ is equivalent to maximising the 'likelihood' term $\langle \log \tilde{p} \rangle_p$. This is

$$\sum_{x,y} p(x, y) \log \tilde{p}(x, y) \tag{15.6.5}$$

It's convenient to derive an EM style algorithm to learn $\tilde{p}(x|z)$, $\tilde{p}(y|z)$ and $\tilde{p}(z)$. To do this, consider

$$\mathrm{KL}(q(z|x, y)|\tilde{p}(z|x, y)) = \sum_z q(z|x, y) \log q(z|x, y) - \sum_z q(z|x, y) \log \tilde{p}(z|x, y) \geq 0 \tag{15.6.6}$$

where $\sum_z$ implies summation over all states of the variable z. Rearranging, this gives the bound,

$$\log \tilde{p}(x, y) \geq -\sum_z q(z|x, y) \log q(z|x, y) + \sum_z q(z|x, y) \log \tilde{p}(z, x, y) \tag{15.6.7}$$

Plugging this into the 'likelihood' term above, we have the bound

$$\begin{aligned}\sum_{x,y} p(x, y) \log \tilde{p}(x, y) \geq &-\sum_{x,y} p(x, y) \sum_z q(z|x, y) \log q(z|x, y) \\ &+ \sum_{x,y} p(x, y) \sum_z q(z|x, y) \left[\log \tilde{p}(x|z) + \log \tilde{p}(y|z) + \log \tilde{p}(z)\right]\end{aligned} \tag{15.6.8}$$

M-step

For fixed $\tilde{p}(x|z), \tilde{p}(y|z)$, the contribution to the bound from $\tilde{p}(z)$ is

$$\sum_{x,y} p(x, y) \sum_z q(z|x, y) \log \tilde{p}(z) \tag{15.6.9}$$

It is straightforward to see that the optimal setting of $\tilde{p}(z)$ is

$$\tilde{p}(z) = \sum_{x,y} q(z|x, y)p(x, y) \tag{15.6.10}$$

since equation (15.6.9) is, up to a constant, $\mathrm{KL}\left(\sum_{x,y} q(z|x, y)p(x, y)|\tilde{p}(z)\right)$. Similarly, for fixed $\tilde{p}(y|z), \tilde{p}(z)$, the contribution to the bound from $\tilde{p}(x|z)$ is

$$\sum_{x,y} p(x, y) \sum_z q(z|x, y) \log \tilde{p}(y|z) \tag{15.6.11}$$

Therefore, optimally

$$\tilde{p}(x|z) \propto \sum_y p(x, y)q(z|x, y) \tag{15.6.12}$$

and similarly,

$$\tilde{p}(y|z) \propto \sum_x p(x, y)q(z|x, y) \tag{15.6.13}$$

(a) (b)

Figure 15.14: (a) Conditional PLSA reconstruction of the images in fig(15.5) using a positive (convex) combination of the 49 positive base images in (b). The root mean square reconstruction error is 1.391×10^{-5}. The base images tend to be more 'localised' than the corresponding eigen-images fig(15.6b). Here one sees local structure such as foreheads, chins, *etc.*

E-step

The optimal setting for the q distribution at each iteration is

$$q(z|x,y) = \tilde{p}(z|x,y) \tag{15.6.14}$$

which is fixed throughout the M-step.

The procedure is given in algorithm(15) and a demonstration is in `demoPLSA.m`. The 'likelihood' equation (15.6.5) is guaranteed to increase (and the Kullback-Leibler divergence equation (15.6.4) decrease) under iterating between the E and M-steps, since the method is analogous to an EM procedure Generalisations, such as using simpler q distributions, (corresponding to generalised EM procedures) are immediate based on modifying the above derivation.

A related probabilistic model

For variables x, y, z with z hidden and $\mathrm{dom}(x) = \{1, \ldots, I\}, \mathrm{dom}(y) = \{1, \ldots, J\}, \mathrm{dom}(z) = \{1, \ldots, K\}$, consider a distribution

$$\tilde{p}(x,y,z) = \tilde{p}(x|z)\tilde{p}(y|z)\tilde{p}(z) \tag{15.6.15}$$

and data $\mathcal{D} = \{(x^n, y^n), n = 1, \ldots, N\}$. Assuming the data are i.i.d. draws from equation (15.6.15) the log likelihood is

$$\log \tilde{p}(\mathcal{D}) = \sum_{n=1}^{N} \log \tilde{p}(x^n, y^n) \tag{15.6.16}$$

where

$$\tilde{p}(x^n, y^n) = \sum_{z^n} \tilde{p}(x^n|z^n)\tilde{p}(y^n|z^n)\tilde{p}(z^n) \tag{15.6.17}$$

If x^n, y^n are sampled from a distribution $p(x,y)$ then, in the limit of an infinite number of samples, $N \to \infty$, equation (15.6.16) becomes

$$\log \tilde{p}(\mathcal{D}) = \langle \log \tilde{p}(x,y) \rangle_{p(x,y)} \tag{15.6.18}$$

which is equation (15.6.5). From this viewpoint, even though we started out with a set of samples, in the limit, only the distribution of the observed data, $p(x,y)$ is relevant. The generic code for the finite sample case, trained with EM is given in `demoMultinomialpXYgZ.m`. See also exercise(167).

A fully probabilistic interpretation of PLSA can be made via Poisson processes[55]. A related probabilistic model is Latent Dirichlet Allocation, which is described in section(20.6.1).

Algorithm 15 PLSA: Given a frequency matrix $p(x = i, y = j)$, return a decomposition $\sum_k \tilde{p}(x = i|z = k)\tilde{p}(y = j|z = k)\tilde{p}(z = k)$. See `plsa.m`

1: Initialise $\tilde{p}(z), \tilde{p}(x|z), \tilde{p}(y|z)$.
2: **while** Not Converged **do**
3: Set $q(z|x, y) = \tilde{p}(z|x, y)$ ▷ E-step
4: Set $\tilde{p}(x|z) \propto \sum_y p(x, y)q(z|x, y)$ ▷ M-Steps
5: Set $\tilde{p}(y|z) \propto \sum_x p(x, y)q(z|x, y)$
6: **end while**
7: Set $\tilde{p}(z) = \sum_{x,y} p(x, y)q(z|x, y)$

Algorithm 16 Conditional PLSA: Given a frequency matrix $p(x = i|y = j)$, return a decomposition $\sum_k \tilde{p}(x = i|z = k)\tilde{p}(z = k|y = j)$. See `plsaCond.m`

1: Initialise $\tilde{p}(x|z), \tilde{p}(z|y)$.
2: **while** Not Converged **do**
3: Set $q(z|x, y) = \tilde{p}(z|x, y)$ ▷ E-step
4: Set $\tilde{p}(x|z) \propto \sum_y p(x|y)q(z|x, y)$ ▷ M-Steps
5: Set $\tilde{p}(z|y) \propto \sum_x p(x|y)q(z|x, y)$
6: **end while**

Conditional PLSA

In some cases it is more natural to consider a conditional frequency matrix

$$p(x = i|y = j) \tag{15.6.19}$$

and seek an approximate decomposition

$$\underbrace{p(x = i|y = j)}_{X_{ij}} \approx \sum_k \underbrace{\tilde{p}(x = i|z = k)}_{B_{ik}} \underbrace{\tilde{p}(z = k|y = j)}_{Y_{kj}} \tag{15.6.20}$$

as depicted in fig(15.13b). Deriving an EM style algorithm for this is straightforward (see exercise(166)), and is presented in algorithm(16), being equivalent to the non-negative matrix factorisation algorithm of [168].

Example 73 (Discovering the basis)**.** A set of images is give in fig(15.15a). These were created by first defining 4 base images fig(15.15b). Each base image is positive and scaled so that the sum of the pixels is unity, $\sum_i p(x = i|z = k) = 1$, where $k = 1, \ldots, 4$ and x indexes the pixels, see fig(15.15). We then sum each of these images using a randomly chosen positive set of 4 weights (under the constraint that the weights sum to 1) to generate a training image with elements $p(x = i|y = j)$ and j indexes the training image. This is repeated 144 times to form the full training set, fig(15.15a). The task is, given only the training set images, to reconstruct the basis from which the images were formed. We assume that we know the correct number of base images, namely 4. The results of using conditional PLSA on this task are presented in fig(15.15c) and using SVD in fig(15.15d). In this case PLSA finds the correct 'natural' basis, corresponding to the way the images were generated. The eigenbasis is just as good in terms of being able to represent any of the training images, but in this case does not correspond to the constraints under which the data was generated

15.6.2 Extensions and variations

Non-negative Matrix Factorisation

Non-negative Matrix factorisation (NMF) considers a decomposition in which both the basis and weight matrices have non-negative entries. An early example of this work is as a form of constrained Factor

(a) (b) (c) (d)

Figure 15.15: **(a)**: Training data, consisting of a positive (convex) combination of the base images. **(b)**: The chosen base images from which the training data is derived. **(c)**: Basis learned using conditional PLSA on the training data. This is virtually indistinguishable from the true basis. **(d)**: Eigenbasis (sometimes called 'eigenfaces').

Analysis[214]. Closely related works are [168] which is a generalisation of PLSA (since there is no requirement that the basis or components sum to unity). In all cases EM-style training algorithms exist, although their convergence can be slow. A natural relaxation is when only one of the factors in the decomposition is constrained to be non-negative. We will encounter similar models in the discussion on Independent Component Analysis, section(21.6).

Gradient based training

EM style algorithms are easy to derive and implement but can exhibit poor convergence. Gradient based methods to simultaneously optimize with respect to the basis and the components have been developed, but require a parameterisation that ensures positivity of the solutions[214].

Array decompositions

It is straightforward to extend the method to the decomposition of multidimensional arrays, based also on more than one basis. For example

$$p(s,t,u) \approx \sum_{v,w} \tilde{p}(s,t,u|v,w)\tilde{p}(v,w) = \sum_{v,w} \tilde{p}(s,t|u,v)\tilde{p}(u|w)\tilde{p}(v)\tilde{p}(w) \tag{15.6.21}$$

Such extensions require only additional bookkeeping.

15.6.3 Applications of PLSA/NMF

Physical models

Non-negative decompositions can arise naturally in certain physical situations. For example, in acoustics, positive amounts of energy combine linearly from different signal sources to form the observed signal. Let's imagine that two kinds of signals are present in an acoustic signal, say a piano and a singer. Using NMF one can learn two separate bases for these cases, and then reconstruct a given signal using only one of the bases. This means that one could potentially remove the singer from a recording, leaving only the piano. See also [282] for a more standard probabilistic model in acoustics. This would be analogous to reconstructing the images in fig(15.15a) using say only one of the learned basis images, see example(73).

Modelling citations

We have a collection of research documents which cite other documents. For example, document 1 might cite documents 3, 2, 10, *etc.* Given only the list of citations for each document, can we identify key research papers and the communities that cite them? Note that this is not the same question as finding the most cited documents – rather we want to identify documents with communities and find their relevance for a

factor 1	(Reinforcement Learning)
0.0108	Learning to predict by the methods of temporal differences. Sutton.
0.0066	Neuronlike adaptive elements that can solve difficult learning control problems. Barto et al.
0.0065	Practical Issues in Temporal Difference Learning. Tesauro.
factor 2	(Rule Learning)
0.0038	Explanation-based generalization: a unifying view. Mitchell et al.
0.0037	Learning internal representations by error propagation. Rumelhart et al.
0.0036	Explanation-Based Learning: An Alternative View. DeJong et al.
factor 3	(Neural Networks)
0.0120	Learning internal representations by error propagation. Rumelhart et al.
0.0061	Neural networks and the bias-variance dilemma. Geman et al.
0.0049	The Cascade-Correlation learning architecture. Fahlman et al.
factor 4	(Theory)
0.0093	Classification and Regression Trees. Breiman et al.
0.0066	Learnability and the Vapnik-Chervonenkis dimension. Blumer et al.
0.0055	Learning Quickly when Irrelevant Attributes Abound. Littlestone.
factor 5	(Probabilistic Reasoning)
0.0118	Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Pearl.
0.0094	Maximum likelihood from incomplete data via the em algorithm. Dempster et al.
0.0056	Local computations with probabilities on graphical structures. Lauritzen et al.
factor 6	(Genetic Algorithms)
0.0157	Genetic Algorithms in Search, Optimization, and Machine Learning. Goldberg.
0.0132	Adaptation in Natural and Artificial Systems. Holland.
0.0096	Genetic Programming: On the Programming of Computers by Means of Natural Selection. Koza.
factor 7	(Logic)
0.0063	Efficient induction of logic programs. Muggleton et al.
0.0054	Learning logical definitions from relations. Quinlan.
0.0033	Inductive Logic Programming Techniques and Applications. Lavrac et al.

Table 15.1: Highest ranked documents according to $p(c|z)$. The factor topic labels are manual assignments based on similarities to the Cora topics. Reproduced from [62].

community.

We use the variable $d \in \{1, \ldots, D\}$ to index documents and $c \in \{1, \ldots, D\}$ to index citations (both d and c have the same domain, namely the index of a research article). If document $d = i$ cites article $c = j$ then we set the entry of the matrix $C_{ij} = 1$. If there is no citation, C_{ij} is set to zero. We can form a 'distribution' over documents and citations using

$$p(d = i, c = j) = \frac{C_{ij}}{\sum_{ij} C_{ij}} \tag{15.6.22}$$

Example 74 (Modelling citations)**.** The Cora corpus (`www.cs.umass.edu/~mccallum`) contains an archive of around 30,000 computer science research papers. From this archive the authors in [62] extracted the papers in the machine learning category, consisting of 4220 documents and 38,372 citations. Using these the distribution equation (15.6.22) was formed. The documents have additionally been categorised by hand into 7 topics: *Case-based reasoning, Genetic Algorithms, Neural Networks, Probabilistic methods, Reinforcement Learning, Rule Learning and Theory*.

In [62] the joint PLSA method is fitted to the data using $z = 7$ topics. From the trained model the expression $p(c = j|z = k)$ defines how authoritative paper j is according to community $z = k$. The results are presented in table(15.1) and show how the method discovers intuitively meaningful topics.

Modelling the web

Consider a collection of websites, indexed by i. If website j points to website i, one sets $C_{ij} = 1$ giving a directed graph of website-to-website links. Since a website will discuss usually only of a small number of 'topics' we might be able to explain why there is a link between two websites using a PLSA decomposition. These algorithms have proved useful for internet search for example to determine the latent topics of websites and identify the most authoritative websites. See [63] for a discussion.

15.7 Kernel PCA

Kernel PCA is a non-linear extension of PCA designed to discover non-linear manifolds. Here we only briefly describe the approach and refer the reader to [239] for details. In kernel PCA, we replace each $\mathbf{x}$ by a 'feature' vector $\tilde{\mathbf{x}} \equiv \boldsymbol{\phi}(\mathbf{x})$. Note that the use of $\tilde{\mathbf{x}}$ here does not have the interpretation we used before as the approximate reconstruction. The *feature map* $\boldsymbol{\phi}$ takes a vector $\mathbf{x}$ and produces a higher dimensional vector $\tilde{\mathbf{x}}$. For example we could map a two dimensional vector $\mathbf{x} = [x_1, x_2]^\mathsf{T}$ using

$$\boldsymbol{\phi}(\mathbf{x}) = \left[x_1, x_2, x_1^2, x_2^2, x_1 x_2, x_1^3, \ldots\right]^\mathsf{T} \tag{15.7.1}$$

The idea is then to perform PCA on these higher dimensional feature vectors, subsequently mapping back the eigenvectors to the original space $\mathbf{x}$. The main challenge is to write this without explicitly computing PCA in the potentially very high dimensional feature vector space. As a reminder, in standard PCA, for zero mean data, one forms an eigen-decomposition of the sample matrix[3]

$$\mathbf{S} = \frac{1}{N}\tilde{\mathbf{X}}\tilde{\mathbf{X}}^\mathsf{T} \tag{15.7.2}$$

For simplicity, we concentrate here on finding the first principal component $\tilde{\mathbf{e}}$ which satisfies

$$\tilde{\mathbf{X}}\tilde{\mathbf{X}}^\mathsf{T}\tilde{\mathbf{e}} = \lambda'\tilde{\mathbf{e}} \tag{15.7.3}$$

for corresponding eigenvalue λ (writing $\lambda' = N\lambda$). The 'dual' representation is obtained by pre-multiplying by $\tilde{\mathbf{X}}^\mathsf{T}$, so that in terms of $\tilde{\mathbf{f}} \equiv \tilde{\mathbf{X}}^\mathsf{T}\tilde{\mathbf{e}}$, the standard PCA eigen-problem reduces to solving:

$$\tilde{\mathbf{X}}^\mathsf{T}\tilde{\mathbf{X}}\tilde{\mathbf{f}} = \lambda'\tilde{\mathbf{f}} \tag{15.7.4}$$

The feature eigenvector $\tilde{\mathbf{e}}$ is then recovered using

$$\tilde{\mathbf{X}}\tilde{\mathbf{f}} = \lambda'\tilde{\mathbf{e}} \tag{15.7.5}$$

We note that matrix $\tilde{\mathbf{X}}^\mathsf{T}\tilde{\mathbf{X}}$ has elements

$$\left[\tilde{\mathbf{X}}^\mathsf{T}\tilde{\mathbf{X}}\right]_{mn} = \boldsymbol{\phi}(\mathbf{x}^m)^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}^n) \tag{15.7.6}$$

and recognise this as the scalar product between vectors. This means that the matrix is positive (semi) definite and we may equivalently use a positive definite kernel, see section(19.3),

$$\left[\tilde{\mathbf{X}}^\mathsf{T}\tilde{\mathbf{X}}\right]_{mn} = k(\mathbf{x}^m, \mathbf{x}^n) = K_{mn} \tag{15.7.7}$$

Then equation (15.7.4) can be written as

$$\mathbf{K}\tilde{\mathbf{f}} = \lambda'\tilde{\mathbf{f}} \tag{15.7.8}$$

One then solves this eigen-equation to find the N dimensional principal dual feature vector $\tilde{\mathbf{f}}$. The projection of the feature $\tilde{\mathbf{x}}$ is given by

$$y = \tilde{\mathbf{x}}^\mathsf{T}\tilde{\mathbf{e}} = \frac{1}{\lambda}\tilde{\mathbf{x}}^\mathsf{T}\tilde{\mathbf{X}}\tilde{\mathbf{f}} \tag{15.7.9}$$

More generally, for a larger number of components, the i^{th} kernel PCA projection y_i can be expressed in terms of the kernel directly as

$$y_i = \frac{1}{N\lambda^i}\sum_{n=1}^{N} k(\mathbf{x}, \mathbf{x}^n)\tilde{f}_n^i \tag{15.7.10}$$

[3]We use the normalisation N as opposed to $N-1$ just for notational convenience – in practice, there is little difference.

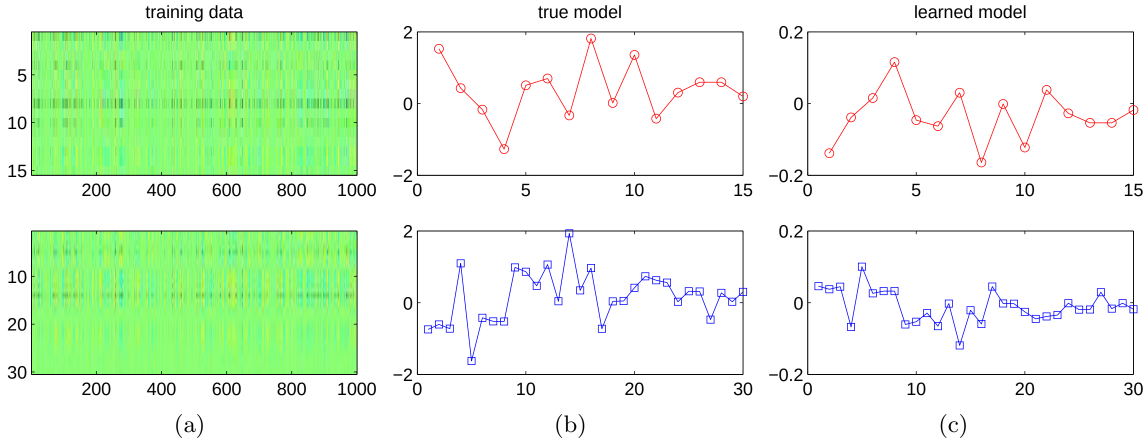


Figure 15.16: Canonical Correlation Analysis. **(a)**: Training data. The top panel contains the $\mathbf{X}$ matrix of 1000, 15 dimensional points, and the bottom the corresponding 30 dimensional $\mathbf{Y}$ matrix. **(b)**: The data in (a) was produced using $\mathbf{X} = \mathbf{Ah}, \mathbf{Y} = \mathbf{Bh}$ where $\mathbf{A}$ is a 15×1 matrix, and $\mathbf{B}$ is a 30×1 matrix. **(c)**: Matrices $\mathbf{A}$ and $\mathbf{B}$ learned by CCA. Note that they are close to the true $\mathbf{A}$ and $\mathbf{B}$ up to rescaling and sign changes. See `demoCCA.m`.

where i is the eigenvalue label.

The above derivation implicitly assumed zero mean features $\tilde{\mathbf{x}}$. Even if the original data $\mathbf{x}$ is zero mean, due to the non-linear mapping, the features may not be zero mean. To correct for this one may show that the only modification required is to replace the matrix $\mathbf{K}$ in equation (15.7.8) above with

$$K'_{mn} = k(\mathbf{x}^m, \mathbf{x}^n) - \frac{1}{N}\sum_{d=1}^{N} k(\mathbf{x}^d, \mathbf{x}^n) - \frac{1}{N}\sum_{d=1}^{N} k(\mathbf{x}^m, \mathbf{x}^d) + \frac{1}{N^2}\sum_{d=1,d'=1}^{N} k(\mathbf{x}^{d'}, \mathbf{x}^d) \tag{15.7.11}$$

Finding the reconstructions

The above gives a procedure for finding the KPCA projection $\mathbf{y}$. However, in many cases we would also like to have an approximate reconstruction using the lower dimensional $\mathbf{y}$. This is not straightforward since the mapping from $\mathbf{y}$ to $\mathbf{x}$ is in general highly non-linear. Here we outline a procedure for achieving this.

First we find the reconstruction $\tilde{\mathbf{x}}^*$ of the feature space $\tilde{\mathbf{x}}$. Now

$$\tilde{\mathbf{x}}^* = \sum_i y_i \tilde{\mathbf{e}}^i = \sum_i y_i \frac{1}{\lambda_i} \sum_n \tilde{f}_i^n \boldsymbol{\phi}(\mathbf{x}^n) \tag{15.7.12}$$

Given $\tilde{\mathbf{x}}^*$ we try to find that point $\mathbf{x}'$ in the original data space that maps to $\tilde{\mathbf{x}}^*$. This can be found by minimising

$$E(\mathbf{x}') = \left(\boldsymbol{\phi}(\mathbf{x}') - \tilde{\mathbf{x}}^*\right)^2 \tag{15.7.13}$$

Up to negligable constants this is

$$E(\mathbf{x}') = k(\mathbf{x}', \mathbf{x}') - 2\sum_i \frac{y_i}{\lambda_i} \sum_n \tilde{f}_i^n k(\mathbf{x}^n, \mathbf{x}') \tag{15.7.14}$$

One then finds $\mathbf{x}'$ by minimising $E(\mathbf{x}')$ numerically.

NEED DEMO.

15.8 Canonical Correlation Analysis

Consider $\mathbf{x}$ and $\mathbf{y}$ which have dimensions $\dim(\mathbf{x})$ and $\dim(\mathbf{y})$ respectively. For example $\mathbf{x}$ might represent a segment of video and $\mathbf{y}$ the corresponding audio. Given then a collection $(\mathbf{x}^n, \mathbf{y}^n), n = 1, \ldots, N$, an interesting challenge is to identify which parts of the audio and video files are strongly correlated. One might expect, for example, that the mouth region of the video is strongly correlated with the audio.

One way to achieve this is to project each $\mathbf{x}$ and $\mathbf{y}$ to one dimension using $\mathbf{a}^\mathsf{T}\mathbf{x}$ and $\mathbf{b}^\mathsf{T}\mathbf{y}$ such that the correlation between the projections is maximal. The unnormalised correlation between the projections $\mathbf{a}^\mathsf{T}\mathbf{x}$ and $\mathbf{b}^\mathsf{T}\mathbf{y}$ is

$$\sum_n \mathbf{a}^\mathsf{T}\mathbf{x}^n\mathbf{b}^\mathsf{T}\mathbf{y}^n = \mathbf{a}^\mathsf{T}\left[\sum_n \mathbf{x}^n\mathbf{y}^{n\mathsf{T}}\right]\mathbf{b} \tag{15.8.1}$$

and the normalised correlation is

$$\frac{\mathbf{a}^\mathsf{T}\mathbf{S}_{xy}\mathbf{b}}{\sqrt{\mathbf{a}^\mathsf{T}\mathbf{S}_{xx}\mathbf{a}}\sqrt{\mathbf{b}^\mathsf{T}\mathbf{S}_{yy}\mathbf{b}}} \tag{15.8.2}$$

where $\mathbf{S}_{xy}$ is the sample $\mathbf{x}, \mathbf{y}$ cross correlation matrix. When the joint covariance of the stacked vectors $\mathbf{z}^n = [\mathbf{x}^n, \mathbf{y}^n]$ is considered $\mathbf{S}_{xx}, \mathbf{S}_{xy}, \mathbf{S}_{yx}, \mathbf{S}_{yy}$ are the blocks of the joint covariance matrix.

Since equation (15.8.2) is invariant with respect to length scaling of $\mathbf{a}$ and also $\mathbf{b}$, we can consider the equivalent objective

$$E(\mathbf{a}, \mathbf{b}) = \mathbf{a}^\mathsf{T}\mathbf{S}_{xy}\mathbf{b} \tag{15.8.3}$$

subject to $\mathbf{a}^\mathsf{T}\mathbf{S}_{xx}\mathbf{a} = 1$ and $\mathbf{b}^\mathsf{T}\mathbf{S}_{yy}\mathbf{b} = 1$. To find the optimal projections $\mathbf{a}$, $\mathbf{b}$, under the constraints, we use the Lagrangian,

$$\mathcal{L}(\mathbf{a}, \mathbf{b}, \lambda_a, \lambda_b) \equiv \mathbf{a}^\mathsf{T}\mathbf{S}_{xy}\mathbf{b} + \frac{\lambda_a}{2}\left(1 - \mathbf{a}^\mathsf{T}\mathbf{S}_{xx}\mathbf{a}\right) + \frac{\lambda_b}{2}\left(1 - \mathbf{b}^\mathsf{T}\mathbf{S}_{yy}\mathbf{b}\right) \tag{15.8.4}$$

from which we obtain the zero derivative criteria

$$\mathbf{S}_{xy}\mathbf{b} = \lambda_a\mathbf{S}_{xx}\mathbf{a}, \qquad \mathbf{S}_{yx}\mathbf{a} = \lambda_b\mathbf{S}_{yy}\mathbf{b} \tag{15.8.5}$$

Hence

$$\mathbf{a}^\mathsf{T}\mathbf{S}_{xy}\mathbf{b} = \lambda_a\mathbf{a}^\mathsf{T}\mathbf{S}_{xx}\mathbf{a} = \lambda_a, \qquad \mathbf{b}^\mathsf{T}\mathbf{S}_{yx}\mathbf{a} = \lambda_b\mathbf{b}^\mathsf{T}\mathbf{S}_{yy}\mathbf{b} = \lambda_b \tag{15.8.6}$$

Since $\mathbf{a}^\mathsf{T}\mathbf{S}_{xy}\mathbf{b} = \mathbf{b}^\mathsf{T}\mathbf{S}_{yx}\mathbf{a}$ we must have $\lambda_a = \lambda_b = \lambda$ at the optimum. If we assume that $\mathbf{S}_{yy}$ is invertible,

$$\mathbf{b} = \frac{1}{\lambda}\mathbf{S}_{yy}^{-1}\mathbf{S}_{yx}\mathbf{a} \tag{15.8.7}$$

Using this to eliminate $\mathbf{b}$ in equation (15.8.5) we have

$$\mathbf{S}_{xy}\mathbf{S}_{yy}^{-1}\mathbf{S}_{yx}\mathbf{a} = \lambda^2\mathbf{S}_{xx}\mathbf{a} \tag{15.8.8}$$

which is a generalised eigen-problem. Assuming that $\mathbf{S}_{xx}$ is invertible we can equivalently write

$$\mathbf{S}_{xx}^{-1}\mathbf{S}_{xy}\mathbf{S}_{yy}^{-1}\mathbf{S}_{yx}\mathbf{a} = \lambda^2\mathbf{a} \tag{15.8.9}$$

which is a standard eigen-problem (albeit with λ^2 as the eigenvalue). Once this is solved we can find $\mathbf{b}$ using equation (15.8.7).

15.8.1 SVD formulation

It is straightforward to show that we can find $\mathbf{a}$ by first computing the SVD of

$$\mathbf{S}_{xx}^{-\frac{1}{2}}\mathbf{S}_{xy}\mathbf{S}_{yy}^{-\frac{1}{2}} \tag{15.8.10}$$

in the form $\mathbf{UDV}^{\mathsf{T}}$ and extracting the maximal singular vector $\mathbf{u}_1$ of $\mathbf{U}$ (the first column on $\mathbf{U}$). Then $\mathbf{a}$ is optimally $\mathbf{S}_{xx}^{-\frac{1}{2}}\mathbf{u}_1$, and similarly, $\mathbf{b}$ is optimally $\mathbf{S}_{yy}^{-\frac{1}{2}}\mathbf{v}_1$, where $\mathbf{v}_1$ is the first column of $\mathbf{V}$. In this way, the extension to finding M multiple directions $\mathbf{A} = \left[\mathbf{a}^1, \ldots, \mathbf{a}^M\right]$ and $\mathbf{B} = \left[\mathbf{b}^1, \ldots, \mathbf{b}^M\right]$ is clear – one takes the corresponding first M singular values accordingly. Doing so maximises the criterion

$$\frac{\text{trace}\left(\mathbf{A}^{\mathsf{T}}\mathbf{S}_{xy}\mathbf{B}\right)}{\sqrt{\text{trace}\left(\mathbf{A}^{\mathsf{T}}\mathbf{S}_{xx}\mathbf{A}\right)}\sqrt{\text{trace}\left(\mathbf{B}^{\mathsf{T}}\mathbf{S}_{yy}\mathbf{B}\right)}} \tag{15.8.11}$$

This approach is taken in `cca.m` – see fig(15.16) for a demonstration. One can also show that CCA corresponds to the probabilistic Factor Analysis model under a block restriction on the form of the factor loadings, see section(21.2.1).

CCA and related kernel extensions have been applied in Machine Learning contexts, for example to model the correlation between images and text in order to improve image retrieval from text queries, see [124].

15.9 Notes

PCA is also known as the Karhunen-Loève decomposition, particularly in the engineering literature.

15.10 Code

`pca.m`: Principal Components Analysis
`demoLSI.m`: Demo of Latent Semantic Indexing/Analysis
`svdm.m`: Singular Value Decomposition with missing data
`demoSVDmissing.m`: Demo SVD with missing data

`plsa.m`: Probabilistic Latent Semantic Analysis
`plsaCond.m`: Conditional Probabilistic Latent Semantic Analysis
`demoPLSA.m`: Demo of PLSA
`demoMultnomialpXYgZ.m`: Demo of 'finite sample' PLSA

`cca.m`: Canonical Correlation Analysis (CCA)
`demoCCA.m`: Demo of Canonical Correlation Analysis

15.11 Exercises

Exercise 160. *Consider a dataset in two dimensions where the data lies on the circumference of a circle of unit radius. What would be the effect of using PCA on this dataset, in which we attempt to reduce the dimensionality to 1? Suggest an alternative one dimensional representation of the data.*

Exercise 161. *Consider two vectors $\mathbf{x}^a$ and $\mathbf{x}^b$ and their corresponding PCA approximations $\mathbf{c}+\sum_{i=1}^M a_i\mathbf{e}^i$ and $\mathbf{c}+\sum_{i=1}^M b_i\mathbf{e}^i$, where the eigenvectors $\mathbf{e}^i, i = 1, \ldots M$ are mutually orthogonal and have unit length. The eigenvector $\mathbf{e}^i$ has corresponding eigenvalue λ^i. Approximate $(\mathbf{x}^a - \mathbf{x}^b)^2$ by using the PCA representations of the data, and show that this is equal to $(\mathbf{a} - \mathbf{b})^2$.*

Exercise 162. *Show how the solution for $\mathbf{a}$ to the CCA problem in equation (15.8.8) can be transformed into the form expressed by equation (15.8.10), as claimed in the text.*

Exercise 163. *Let* $\mathbf{S}$ *be the covariance matrix of the data. The Mahalanobis distance between* $\mathbf{x}^a$ *and* $\mathbf{x}^b$ *is defined as*

$$\left(\mathbf{x}^a - \mathbf{x}^b\right)^{\mathsf{T}} \mathbf{S}^{-1} \left(\mathbf{x}^a - \mathbf{x}^b\right). \tag{15.11.1}$$

Explain how to approximate this distance using M*-dimensional PCA approximations.*

Exercise 164 (PCA with external inputs)**.** *In some applications, one may suspect that certain external variables* $\mathbf{v}$ *have a strong influence on how the data* $\mathbf{x}$ *is distributed. For example, if* $\mathbf{x}$ *represents an image, it might be that we know the lighting condition* $\mathbf{v}$ *under which the image was made – this will have a large effect on the image. It would make sense therefore to include the known lighting condition in forming a lower dimensional representation of the image. Note that we don't want to form a lower dimensional representation of the joint* $\mathbf{x}, \mathbf{v}$*, rather we want to form a lower dimensional representation of* $\mathbf{x}$ *alone, bearing in mind that some of the variability observed may be due to* $\mathbf{v}$*.*
We therefore assume an approximation

$$\mathbf{x}^n \approx \sum_j y_j^n \mathbf{b}^j + \sum_k v_k^n \mathbf{c}^k \tag{15.11.2}$$

where the coefficients y_i^n*,* $i = 1, \ldots N$*,* $n = 1, \ldots N$ *and basis vectors* $\mathbf{b}^j$*,* $j = 1, \ldots J$ *and* $\mathbf{c}^k$*,* $k = 1, \ldots K$ *are to be determined. The external inputs* $\mathbf{v}^1, \ldots, \mathbf{v}^N$ *are given. The sum squared error loss between the* $\mathbf{x}^n$ *and their linear reconstruction equation (15.11.2) is*

$$E = \sum_{n,i} \left(x_i^n - \sum_j y_j^n b_i^j - \sum_k v_k^n c_i^k \right)^2 \tag{15.11.3}$$

Find the parameters that minimise E*.*

Exercise 165. *Consider the following 3-dimensional datapoints:*

$$(1.3, 1.6, 2.8)(4.3, -1.4, 5.8)(-0.6, 3.7, 0.7)(-0.4, 3.2, 5.8)(3.3, -0.4, 4.3)(-0.4, 3.1, 0.9) \tag{15.11.4}$$

Perform Principal Components Analysis by:

1. *Calculating the mean,* $\mathbf{c}$*, of the data.*
2. *Calculating the covariance matrix* $S = \frac{1}{6} \sum_{n=1}^{6} \mathbf{x}^n (\mathbf{x}^n)^{\mathsf{T}} - \mathbf{c}\mathbf{c}^{\mathsf{T}}$ *of the data.*
3. *Finding the eigenvalues and eigenvectors* $\mathbf{e}^i$ *of the covariance matrix.*

You should find that only two eigenvalues are large, and therefore that the data can be well represented using two components only. Let $\mathbf{e}^1$ *and* $\mathbf{e}^2$ *be the two eigenvectors with largest eigenvalues.*

1. *Calculate the two dimensional representation of each datapoint* $(\mathbf{e}^1 \cdot (\mathbf{x}^n - \mathbf{c}), \mathbf{e}^2 \cdot (\mathbf{x}^n - \mathbf{c})), n = 1, \ldots, 6$*.*
2. *Calculate the reconstruction of each datapoint* $\mathbf{c} + ({\mathbf{e}^1}^{\mathsf{T}}(\mathbf{x}^n - \mathbf{c}))\mathbf{e}^1 + ({\mathbf{e}^2}^{\mathsf{T}}(\mathbf{x}^n - \mathbf{c}))\mathbf{e}^2$*,* $n = 1, \ldots, 6$*.*

Exercise 166. *Consider a 'conditional frequency matrix'*

$$p(x = i|y = j) \tag{15.11.5}$$

Show how to derive an EM style algorithm for an approximate decomposition of this matrix in the form

$$p(x = i|y = j) \approx \sum_k \tilde{p}(x = i|z = k)\tilde{p}(z = k|y = j) \tag{15.11.6}$$

where $k = 1, \ldots, Z$*,* $i = 1, \ldots, X$*,* $j = 1, \ldots, Y$*.*

Exercise 167. *For the multinomial model* $\tilde{p}(x, y, z)$ *described in equation (15.6.15), derive explicitly the EM algorithm and implement this in* MATLAB*. For randomly chosen values for the conditional probabilities, draw 10000 samples from this model for* $X = 5, Y = 5, Z = 4$ *and compute from this the matrix with elements*

$$p_{ij} = \frac{\sharp\,(x = i, y = j)}{\sum_{i=1}^{X} \sum_{j=1}^{Y} \sharp\,(x = i, y = j)} \tag{15.11.7}$$

Now run PLSA (use `plsa.m`*) with the settings* $X = 5, Y = 5, Z = 4$ *to learn and compare your results with those obtained from the finite sample model equation (15.6.15).*

CHAPTER 16

Supervised Linear Dimension Reduction

16.1 Supervised Linear Projections

In chapter(15) we discussed dimension reduction using an unsupervised procedure. In cases where class information is available, and our ultimate interest is to reduce dimensionality for improved classification, it makes sense to use the available class information in forming the projections. Exploiting the class label information to improve the projection is a form of *supervised* dimension reduction. Let's consider data from two different classes. For class 1, we have a set of data N_1 datapoints,

$$\mathcal{X}_1 = \left\{ \mathbf{x}_1^1, \ldots, \mathbf{x}_1^{N_1} \right\} \tag{16.1.1}$$

and similarly for class 2, we have a set of N_2 datapoints

$$\mathcal{X}_2 = \left\{ \mathbf{x}_2^1, \ldots, \mathbf{x}_2^{N_2} \right\} \tag{16.1.2}$$

Our interest is then to find a linear projection,

$$\mathbf{y} = \mathbf{W}^\mathsf{T}\mathbf{x} \tag{16.1.3}$$

where $\dim \mathbf{W} = D \times L$, $L < D$, such that for datapoints $\mathbf{x}^i, \mathbf{x}^j$ in the same class, the distance between their projections $\mathbf{y}^i, \mathbf{y}^j$ should be small. Conversely, for datapoints in different classes, the distance between their projections should be large. This may be useful for classification purposes since for a novel point $\mathbf{x}^*$, if its projection

$$\mathbf{y}^* = \mathbf{W}^\mathsf{T}\mathbf{x}^* \tag{16.1.4}$$

is close to class 1 projected data, we would expect $\mathbf{x}^*$ to belong to class 1. In forming the supervised projection, only the class discriminative parts of the data are retained, so that the procedure can be considered a form of supervised feature extraction.

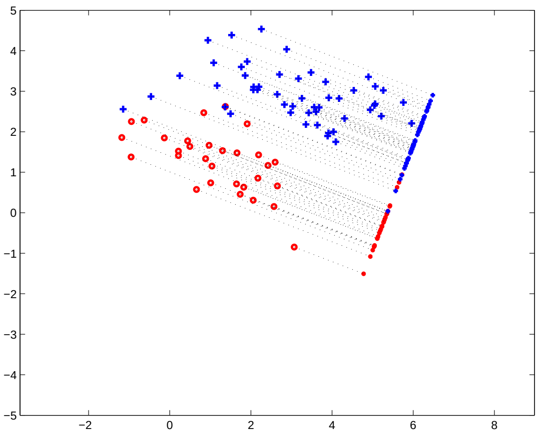
16.2 Fisher's Linear Discriminant

We restrict attention to binary class data. Also, for simplicity, we project the data down to one dimension. The canonical variates algorithm of section(16.3) deals with the generalisations.

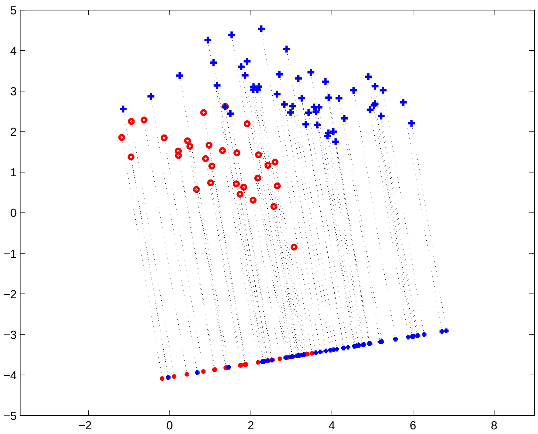
Gaussian assumption

We model the data from each class with a Gaussian. That is

$$p(\mathbf{x}_1) = \mathcal{N}(\mathbf{x}_1 | \mathbf{m}_1, \mathbf{S}_1), \qquad p(\mathbf{x}_2) = \mathcal{N}(\mathbf{x}_2 | \mathbf{m}_2, \mathbf{S}_2) \tag{16.2.1}$$



(a)



(b)

Figure 16.1: The large crosses represent data from class 1, and the large circles from class 2. Their projections onto 1 dimension are represented by their small counterparts. **(a)**: Fisher's Linear Discriminant Analysis. Here there is little class overlap in the projections. **(b)**: Unsupervised dimension reduction using Principal Components Analysis for comparison. There is considerable class overlap in the projection. In both (a) and (b) the one dimensional projection is the distance along the line, measured from an arbitrary chosen fixed point on the line.

where $\mathbf{m}_1$ is the sample mean of class 1 data, and $\mathbf{S}_1$ the sample covariance; similarly for class 2. The projections of the points from the two classes are then given by

$$y_1^n = \mathbf{w}^\mathsf{T}\mathbf{x}_1^n, \qquad y_2^n = \mathbf{w}^\mathsf{T}\mathbf{x}_2^n \tag{16.2.2}$$

Because the projections are linear, the projected distributions are also Gaussian,

$$p(y_1) = \mathcal{N}\left(y_1|\mu_1, \sigma_1^2\right), \qquad \mu_1 = \mathbf{w}^\mathsf{T}\mathbf{m}_1, \qquad \sigma_1^2 = \mathbf{w}^\mathsf{T}\mathbf{S}_1\mathbf{w} \tag{16.2.3}$$

$$p(y_2) = \mathcal{N}\left(y_2|\mu_2, \sigma_2^2\right), \qquad \mu_2 = \mathbf{w}^\mathsf{T}\mathbf{m}_2, \qquad \sigma_2^2 = \mathbf{w}^\mathsf{T}\mathbf{S}_2\mathbf{w} \tag{16.2.4}$$

We search for a projection $\mathbf{w}$ such that the projected distributions have minimal overlap. This can be achieved if the projected Gaussian means are maximally separated, $(\mu_1 - \mu_2)^2$ is large. However, if the variances σ_1^2, σ_2^2 are also large, there could be a large overlap still in the classes. A useful objective function therefore is

$$\frac{(\mu_1 - \mu_2)^2}{\pi_1\sigma_1^2 + \pi_2\sigma_2^2} \tag{16.2.5}$$

where π_i represents the fraction of the dataset in class i. In terms of the projection $\mathbf{w}$, the objective equation (16.2.5) is

$$F(\mathbf{w}) = \frac{\mathbf{w}^\mathsf{T}(\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^\mathsf{T}\mathbf{w}}{\mathbf{w}^\mathsf{T}(\pi_1\mathbf{S}_1 + \pi_2\mathbf{S}_2)\mathbf{w}} = \frac{\mathbf{w}^\mathsf{T}\mathbf{A}\mathbf{w}}{\mathbf{w}^\mathsf{T}\mathbf{B}\mathbf{w}} \tag{16.2.6}$$

where

$$\mathbf{A} = (\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^\mathsf{T}, \qquad \mathbf{B} = \pi_1\mathbf{S}_1 + \pi_2\mathbf{S}_2 \tag{16.2.7}$$

The optimal $\mathbf{w}$ can be found by differentiating equation (16.2.6) with respect to $\mathbf{w}$. This gives

$$\frac{\partial}{\partial\mathbf{w}}\frac{\mathbf{w}^\mathsf{T}\mathbf{A}\mathbf{w}}{\mathbf{w}^\mathsf{T}\mathbf{B}\mathbf{w}} = \frac{2}{(\mathbf{w}\mathbf{B}\mathbf{w})^2}\left[\left(\mathbf{w}^\mathsf{T}\mathbf{B}\mathbf{w}\right)\mathbf{A}\mathbf{w} - \left(\mathbf{w}^\mathsf{T}\mathbf{A}\mathbf{w}\right)\mathbf{B}\mathbf{w}\right] \tag{16.2.8}$$

and therefore the zero derivative requirement is

$$\left(\mathbf{w}^\mathsf{T}\mathbf{B}\mathbf{w}\right)\mathbf{A}\mathbf{w} = \left(\mathbf{w}^\mathsf{T}\mathbf{A}\mathbf{w}\right)\mathbf{B}\mathbf{w} \tag{16.2.9}$$

Multiplying by the inverse of **B** we have

$$\mathbf{B}^{-1}(\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^\mathsf{T}\mathbf{w} = \frac{\mathbf{w}^\mathsf{T}\mathbf{A}\mathbf{w}}{\mathbf{w}^\mathsf{T}\mathbf{B}\mathbf{w}}\mathbf{w} \tag{16.2.10}$$

This means that the optimal projection is explicitly given by

$$\mathbf{w} \propto \mathbf{B}^{-1}(\mathbf{m}_1 - \mathbf{m}_2) \tag{16.2.11}$$

Although the proportionality factor depends on $\mathbf{w}$, we may take it to be constant since the objective function $F(\mathbf{w})$ of equation (16.2.6) is invariant to rescaling of $\mathbf{w}$. We may therefore take

$$\mathbf{w} = k\mathbf{B}^{-1}(\mathbf{m}_1 - \mathbf{m}_2) \tag{16.2.12}$$

It is common to rescale $\mathbf{w}$ to have unit length, $\mathbf{w}^\mathsf{T}\mathbf{w} = 1$, such that

$$k = \frac{1}{\sqrt{(\mathbf{m}_1 - \mathbf{m}_2)^\mathsf{T}\mathbf{B}^{-2}(\mathbf{m}_1 - \mathbf{m}_2)}} \tag{16.2.13}$$

An illustration of the method is given in fig(16.1), which demonstrates how supervised dimension reduction can produce lower dimensional representations more suitable for subsequent classification than an unsupervised method such as PCA.

One can also arrive at the equation (16.2.12) from a different starting objective. By treating the projection as a regression problem $y = \mathbf{w}^\mathsf{T}\mathbf{x} + b$ in which the outputs y are defined as y_1 and y_2 for classes 1 and class 2 respectively, one may show that, for suitably chosen y_1 and y_2, the solution using a least squares criterion is given by equation (16.2.12) [82, 42]. This also suggests a way to regularise LDA, see exercise(170). Kernel extensions of LDA are possible, see for example [77, 245].

When the naive method breaks down

The above derivation relied on the existence of the inverse of **B**. In practice, however, **B** may not be invertible, and the above procedure requires modification. A case where **B** is not invertible is when there are fewer datapoints $N_1 + N_2$ than dimensions D. Another case is when there are elements of the input vectors that never vary. For example, in the hand-written digits case, the pixels at the corner edges are actually always zero. Let's call this corner pixel z. The matrix **B** will then have a zero entry for $[B]_{z,z}$ (indeed the whole z^{th} row and column will be zero) so that for any vector

$$\mathbf{w} = (0, 0, \ldots, w_z, 0, 0, \ldots, 0) \Rightarrow \mathbf{w}^\mathsf{T}\mathbf{B}\mathbf{w} = 0 \tag{16.2.14}$$

This shows that the denominator of Fisher's objective can become zero, and the objective ill defined. We will deal with these issues section(16.3.1).

16.3 Canonical Variates

Canonical Variates generalises Fisher's method to projections in more than one dimension and more than two classes. The projection of any point is given by

$$\mathbf{y} = \mathbf{W}^\mathsf{T}\mathbf{x} \tag{16.3.1}$$

where $\mathbf{W}$ is a $D \times L$ matrix. Assuming that the data $\mathbf{x}$ from class c is Gaussian distributed,

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\mathbf{m}_c, \mathbf{S}_c) \tag{16.3.2}$$

the projections $\mathbf{y}$ are also Gaussian

$$p(\mathbf{y}) = \mathcal{N}\left(\mathbf{y}|\mathbf{W}^\mathsf{T}\mathbf{m}_c, \mathbf{W}^\mathsf{T}\mathbf{S}_c\mathbf{W}\right) \tag{16.3.3}$$

To extend to more than two classes, we define the following matrices:

Between class Scatter Find $\mathbf{m}$ the mean of the whole dataset and $\mathbf{m}_c$, the mean of the each class c. Form

$$\mathbf{A} \equiv \sum_{c=1}^{C} N_c \left(\mathbf{m}_c - \mathbf{m}\right) \left(\mathbf{m}_c - \mathbf{m}\right)^{\mathsf{T}} \tag{16.3.4}$$

where N_c is the number of datapoints in class c, $c = 1, \ldots, C$.

Within class Scatter For each class c form a covariance matrix $\mathbf{S}_c$ and mean $\mathbf{m}_c$. Define

$$\mathbf{B} \equiv \sum_{c=1}^{C} N_c \mathbf{S}_c \tag{16.3.5}$$

This naturally gives rise to a *Raleigh quotient* objective

$$F(\mathbf{W}) \equiv \frac{\text{trace}\left(\mathbf{W}^{\mathsf{T}}\mathbf{A}\mathbf{W}\right)}{\text{trace}\left(\mathbf{W}^{\mathsf{T}}\mathbf{B}\mathbf{W}\right)} \tag{16.3.6}$$

Assuming $\mathbf{B}$ is invertible (see section(16.3.1) otherwise), we can define the Cholesky factor $\tilde{\mathbf{B}}$, with

$$\tilde{\mathbf{B}}^{\mathsf{T}}\tilde{\mathbf{B}} = \mathbf{B} \tag{16.3.7}$$

Then defining

$$\tilde{\mathbf{W}} = \tilde{\mathbf{B}}\mathbf{W} \Rightarrow \mathbf{W} = \tilde{\mathbf{B}}^{-1}\tilde{\mathbf{W}} \tag{16.3.8}$$

the objective can be written in terms of $\tilde{\mathbf{W}}$:

$$F(\tilde{\mathbf{W}}) \equiv \frac{\text{trace}\left(\tilde{\mathbf{W}}^{\mathsf{T}}\tilde{\mathbf{B}}^{-\mathsf{T}}\mathbf{A}\tilde{\mathbf{B}}^{-1}\tilde{\mathbf{W}}\right)}{\text{trace}\left(\tilde{\mathbf{W}}^{\mathsf{T}}\tilde{\mathbf{W}}\right)} \tag{16.3.9}$$

If we assume an orthonormality constraint on $\tilde{\mathbf{W}}$, then we equivalently require the maximisation of

$$F(\tilde{\mathbf{W}}) \equiv \text{trace}\left(\tilde{\mathbf{W}}^{\mathsf{T}}\mathbf{C}\tilde{\mathbf{W}}\right), \text{ subject to } \tilde{\mathbf{W}}^{\mathsf{T}}\tilde{\mathbf{W}} = \mathbf{I} \tag{16.3.10}$$

where

$$\mathbf{C} \equiv \tilde{\mathbf{B}}^{-\mathsf{T}}\mathbf{A}\tilde{\mathbf{B}}^{-1} \tag{16.3.11}$$

Since $\mathbf{C}$ is symmetric and positive semi-definite, it has a real eigen-decomposition

$$\mathbf{C} = \mathbf{E}\mathbf{\Lambda}\mathbf{E}^{\mathsf{T}} \tag{16.3.12}$$

where $\mathbf{\Lambda} = \text{diag}\left(\lambda_1, \lambda_2, \ldots, \lambda_D\right)$ is diagonal with non-negative entries containing the eigenvalues, sorted by decreasing order, $\lambda_1 \geq \lambda_2, \ldots$ and $\mathbf{E}^{\mathsf{T}}\mathbf{E} = \mathbf{I}$. Hence

$$F(\tilde{\mathbf{W}}) = \text{trace}\left(\tilde{\mathbf{W}}^{\mathsf{T}}\mathbf{E}\mathbf{\Lambda}\mathbf{E}^{\mathsf{T}}\tilde{\mathbf{W}}\right) \tag{16.3.13}$$

By setting $\tilde{\mathbf{W}} = [\mathbf{e}_1, \ldots, \mathbf{e}_L]$, where $\mathbf{e}_l$ is the l^{th} eigenvector, the objective becomes the sum of the first L eigenvalues. This setting maximises the objective function since forming $\tilde{\mathbf{W}}$ from any other columns of $\mathbf{E}$ would give a lower sum. We then return

$$\mathbf{W} = \tilde{\mathbf{B}}^{-1}\tilde{\mathbf{W}} \tag{16.3.14}$$

as the projection matrix. The procedure is outlined in algorithm(17). Note that since $\mathbf{A}$ has rank C, there can be no more than $C - 1$ non-zero eigenvalues and corresponding directions.

Algorithm 17 Canonical Variates

1: Compute the between and within class scatter matrices $\mathbf{A}$, equation (16.3.4) and $\mathbf{B}$, equation (16.3.5).
2: Compute the Cholesky factor $\tilde{\mathbf{B}}$ of $\mathbf{B}$.
3: Compute the L principal eigenvectors $[\mathbf{e}_1, \ldots, \mathbf{e}_L]$ of $\tilde{\mathbf{B}}^{-\mathsf{T}}\mathbf{A}\tilde{\mathbf{B}}^{-1}$.
4: $\tilde{\mathbf{W}} = [\mathbf{e}_1, \ldots, \mathbf{e}_L]$
5: Return $\mathbf{W} = \tilde{\mathbf{B}}^{-1}\tilde{\mathbf{W}}$ as the projection matrix.

16.3.1 Dealing with the Nullspace

The above derivation of Canonical Variates (and also Fisher's LDA) requires the invertibility of the matrix $\mathbf{B}$. However, as we discussed in section(16.2) one may encounter situations where $\mathbf{B}$ is not invertible. A solution is to require that $\mathbf{W}$ lies only in the subspace spanned by the data (that is there can be no contribution from the nullspace). To do this we first concatenate the training data from all classes into one large matrix $\mathbf{X}$. A basis for $\mathbf{X}$ can be found using, for example, the thin-SVD technique which returns an orthonormal basis $\mathbf{Q}$. We then require the solution $\mathbf{W}$ to be expressed in this basis:

$$\mathbf{W} = \mathbf{Q}\mathbf{W}' \tag{16.3.15}$$

for some matrix $\mathbf{W}'$. Substituting this in the Canonical Variates objective equation (16.3.6), we obtain

$$F(\mathbf{W}') \equiv \frac{\text{trace}\left(\mathbf{W}'^{\mathsf{T}}\mathbf{Q}^{\mathsf{T}}\mathbf{A}\mathbf{Q}\mathbf{W}'\right)}{\text{trace}\left(\mathbf{W}'^{\mathsf{T}}\mathbf{Q}^{\mathsf{T}}\mathbf{B}\mathbf{Q}\mathbf{W}'\right)} \tag{16.3.16}$$

This is of the same form as the standard quotient, equation (16.3.6), on replacing the between-scatter $\mathbf{A}$ with

$$\mathbf{A}' \equiv \mathbf{Q}^{\mathsf{T}}\mathbf{A}\mathbf{Q} \tag{16.3.17}$$

and the within-scatter $\mathbf{B}$ with

$$\mathbf{B}' \equiv \mathbf{Q}^{\mathsf{T}}\mathbf{B}\mathbf{Q} \tag{16.3.18}$$

In this case $\mathbf{B}'$ is guaranteed invertible, and one may carry out Canonical Variates, as in section(16.3) above. This will return a matrix $\mathbf{W}'$. We then return

$$\mathbf{W} = \mathbf{Q}\mathbf{W}' \tag{16.3.19}$$

See also `CanonVar.m`.

Example 75 (Using canonical variates on the Digits Data)**.** We apply canonical variates to project the digit data onto two dimensions, see fig(16.3). There are 800 examples of a three, 800 examples of a five and 800 examples of a seven. Thus, overall, there are 2400 examples lying in a 784 (28×28 pixels) dimensional space. Note how the canonical variates projected data onto two dimensions has very little class overlap, see fig(16.3a). In comparison the projections formed from PCA, which discards the class information, displays a high degree of class overlap. The different scales of the canonical variates and PCA projections

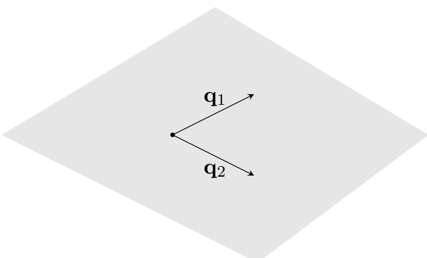
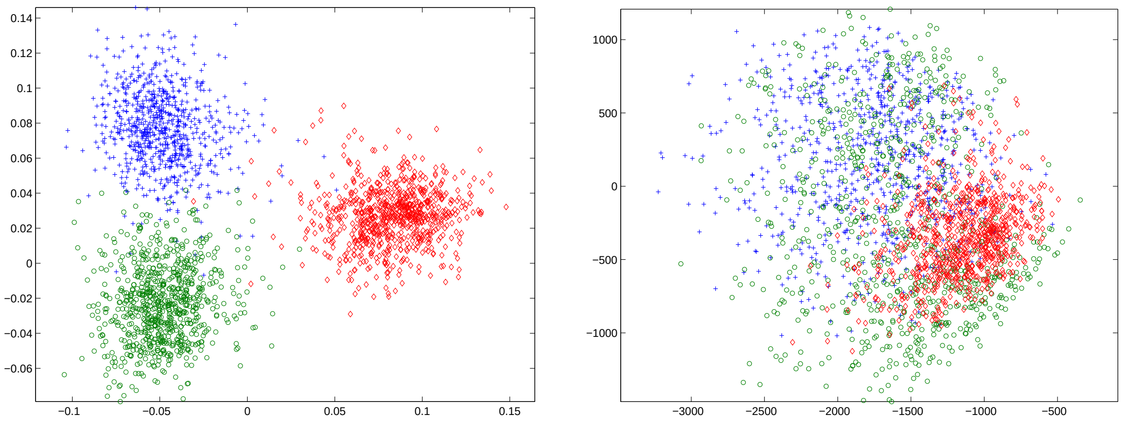


Figure 16.2: Each three dimensional datapoint lies in a two-dimensional plane, meaning that the matrix $\mathbf{B}$ is not full rank, and therefore not invertible. A solution is given by finding vectors $\mathbf{q}_1, \mathbf{q}_2$ that span the plane, and expressing the Canonical Variates solution in terms of these vectors alone.



(a) (b)

Figure 16.3: **(a)**: Canonical Variates projection of examples of handwritten digits 3('+'), 5('o') and 7(diamond). There are 800 examples from each digit class. Plotted are the projections down to 2 dimensions. **(b)**: PCA projections for comparison.

is due to the different constraints on the projection matrices $\mathbf{W}$. In PCA $\mathbf{W}$ is unitary; in canonical variates $\mathbf{W}^\mathsf{T}\mathbf{B}\mathbf{W} = \mathbf{I}$, meaning that $\mathbf{W}$ will scale with the inverse square root of the largest eigenvalues of the within class scatter matrix. Since the canonical variates objective is independent of linear scaling, $\mathbf{W}$ can be rescaled with an arbitrary scalar prefactor $\gamma\mathbf{W}$, as desired.

16.4 Using non-Gaussian data distributions

The applicability of canonical variates depends on our assumption that a Gaussian is a good description of the data. Clearly, if the data is multimodal, using a single Gaussian to model the data in each class is a poor assumption. This may result in projections with a large class overlap. In principle, there is no conceptual difficulty in using more complex distributions, with say more general criteria such as Kullback-Leibler divergence between projected distributions used as the objective. However, such criteria typically result in difficult optimisation problems. Canonical variates is popular due to its simplicity and lack of local optima issues in constructing the projection.

16.5 Code

`CanonVar.m`: Canonical Variates
`demoCanonVarDigits.m`: Demo for Canonical Variates

16.6 Exercises

Exercise 168. *What happens to Fisher's Linear Discriminant if there are less datapoints than dimensions?*

Exercise 169. *Modify* `demoCanonVarDigits.m` *to project and visualise the digits data in 3 dimensions.*

Exercise 170. *Consider* N_1 *class 1 datapoints* $\mathbf{x}_{n_1}, n_1 = 1, \ldots, N_1$ *and class 2 datapoints* $\mathbf{x}_{n_2}, n_2 = 1, \ldots, N_2$. *We will make a linear predictor for the data,*

$$y = \mathbf{w}^\mathsf{T}\mathbf{x} + b \tag{16.6.1}$$

with the aim to predict value y_1 for data from class 1 and y_2 for data from class two. A measure of the fit is given by

$$E(\mathbf{w}, b|y_1, y_2) = \sum_{n_1=1}^{N_1} \left(y_1 - \mathbf{w}^T\mathbf{x}_{n_1} - b\right)^2 + \sum_{n_2=1}^{N_2} \left(y_2 - \mathbf{w}^T\mathbf{x}_{n_2} - b\right)^2 \tag{16.6.2}$$

Show that by setting $y_1 = (N_1 + N_2)/N_1$ and $y_2 = (N_1 + N_2)/N_2$ the $\mathbf{w}$ which minimises E corresponds to Fisher's LDA solution. Hint: first show that the two zero derivative conditions are

$$\sum_{n_1} \left(y_1 - b - \mathbf{w}^T\mathbf{x}_{n_1}\right) + \sum_{n_2} \left(y_2 - b - \mathbf{w}^T\mathbf{x}_{n_2}\right) = 0 \tag{16.6.3}$$

and

$$\sum_{n_1} \left(y_1 - b - \mathbf{w}^T\mathbf{x}_{n_1}\right) \mathbf{x}_{n_1}^T + \sum_{n_2} \left(y_2 - b - \mathbf{w}^T\mathbf{x}_{n_2}\right) \mathbf{x}_{n_2}^T = 0 \tag{16.6.4}$$

which can be reduced to the single equation

$$N\left(\mathbf{m}_1 - \mathbf{m}_2\right) = \left(N\mathbf{B} + \frac{N_1 N_2}{N}\left(\mathbf{m}_1 - \mathbf{m}_2\right)\left(\mathbf{m}_1 - \mathbf{m}_2\right)^T\right)\mathbf{w} \tag{16.6.5}$$

where $\mathbf{B}$ is as defined for LDA in the text, equation (16.2.7).

Note that this suggests a way to regularise LDA, namely by adding on a term $\lambda\mathbf{w}^T\mathbf{w}$ to $E(\mathbf{w}, b|y_1, y_2)$. This can be absorbed into redefining equation (16.3.5) as

$$\mathbf{B}' = \mathbf{B} + \lambda\mathbf{I} \tag{16.6.6}$$

In other words, one can increase the covariance $\mathbf{B}$ by an additive amount $\lambda\mathbf{I}$. The optimal regularising constant λ may be set by cross-validation. More generally one can consider the use of a regularising matrix $\lambda\mathbf{R}$, where $\mathbf{R}$ is positive definite.

Exercise 171. *Consider the digit data of 892 fives* `digit5.mat` *and 1028 sevens* `digit7.mat`*. Make a training set which consists of the first 500 examples from each digit class. Use Canonical Variates to first project the data down to 50 dimensions and compute the Nearest Neighbour performance on the remaining digits. Compare the classification accuracy to using Nearest Neighbours the projections from PCA using 50 components.*

Exercise 172. *Consider an objective function of the form*

$$F(w) \equiv \frac{A(w)}{B(w)} \tag{16.6.7}$$

where $A(w)$ and $B(w)$ are positive functions, and our task is to maximise $F(w)$ with respect to w. It may be that this objective does not have a simple algebraic solution, even though $A(w)$ and $B(w)$ are simple functions.

We can consider an alternative objective, namely

$$J(w, \lambda) = A(w) - \lambda B(w) \tag{16.6.8}$$

where λ is a constant scalar. Choose an initial point w^{old} at random and set

$$\lambda^{old} \equiv A(w^{old})/B(w^{old}) \tag{16.6.9}$$

In that case $J(w^{old}, \lambda^{old}) = 0$. Now choose a w such that

$$J(w, \lambda^{old}) = A(w) - \lambda^{old}B(w) \geq 0 \tag{16.6.10}$$

This is certainly possible since $J(w^{old}, \lambda^{old}) = 0$. If we can find a w such that $J(w, \lambda^{old}) > 0$, then

$$A(w) - \lambda^{old}B(w) > 0 \tag{16.6.11}$$

Show that for such a w, $F(w) > F(w^{old})$, and suggest an iterative optimisation procedure for objective functions of the form $F(w)$.

CHAPTER 17

Linear Models

17.1 Introduction: Fitting a straight line

Given training data $\{(x^n, y^n), n = 1, \ldots, N\}$, for scalar input x^n and scalar output y^n, a linear regression fit is

$$y(x) = a + bx \tag{17.1.1}$$

To determine the best parameters a, b, we use a measure of the discrepancy between the observed outputs and the linear regression fit such as the sum squared training error. This is also called *ordinary least squares* and minimises the average vertical projection of the points y to fitted line:

$$E(a, b) = \sum_{n=1}^{N} [y^n - y(x^n)]^2 = \sum_{n=1}^{N} (y^n - a - bx^n)^2 \tag{17.1.2}$$

Our task is to find the parameters a and b that minimise $E(a, b)$. Differentiating with respect to a and b we obtain

$$\frac{\partial}{\partial a} E(a, b) = -2 \sum_{n=1}^{N} (y^n - a - bx^n), \qquad \frac{\partial}{\partial b} E(a, b) = -2 \sum_{n=1}^{N} (y^n - a - bx^n) x^n \tag{17.1.3}$$

Dividing by N and equating to zero, the optimal parameters are given from the solution to the two linear equations

$$\langle y \rangle - a - b \langle x \rangle = 0, \qquad \langle xy \rangle - a \langle x \rangle - b \left\langle x^2 \right\rangle = 0 \tag{17.1.4}$$

where we used the notation $\langle f(x, y) \rangle$ to denote $\frac{1}{N} \sum_{n=1}^{N} f(x^n, y^n)$. We can readily solve the equations(17.1.4) to determine a and b:

$$a = \langle y \rangle - b \langle x \rangle \tag{17.1.5}$$

$$b \left\langle x^2 \right\rangle = \langle yx \rangle - \langle x \rangle (\langle y \rangle - b \langle x \rangle) \Rightarrow b \left[\left\langle x^2 \right\rangle - \langle x \rangle^2 \right] = \langle xy \rangle - \langle x \rangle \langle y \rangle \tag{17.1.6}$$

Hence

$$b = \frac{\langle xy \rangle - \langle x \rangle \langle y \rangle}{\langle x^2 \rangle - \langle x \rangle^2} \tag{17.1.7}$$

and a is found by substituting this value for b into equation (17.1.5).

In contrast to ordinary least squares regression, PCA from chapter(15) minimises the orthogonal projection of y to the line and is known as *orthogonal least squares* – see example(76).

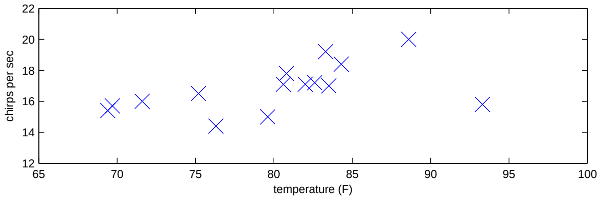


Figure 17.1: Data from crickets – the number of chirps per second, versus the temperature in Fahrenheit.

Example 76. Consider the data in fig(17.1), in which we plot the number of chirps c per second for crickets, versus the temperature t in degrees Fahrenheit. A biologist believes that there is a simple relation between the number of chirps and the temperature of the form

$$c = a + bt \tag{17.1.8}$$

where she needs to determine the parameters a and b. For the cricket data, the fit is plotted in fig(17.2a). For comparison we plot the fit from the PCA, fig(17.2b), which minimises the sum of the squared orthogonal projections from the data to the line. In this case there is little numerical difference between the two fits.

17.2 Linear parameter models for regression

We can generalise on the idea of fitting straight lines to fitting linear functions of vector inputs. For a dataset $\{(\mathbf{x}^n, y^n), n = 1, \ldots, N\}$, a linear parameter regression model (LPM) is defined by[1]

$$y(\mathbf{x}) = \mathbf{w}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}) \tag{17.2.1}$$

where $\phi(\mathbf{x})$ is a vector valued function of the input vector $\mathbf{x}$. For example, in the case of a straight line fit, with a scalar input and output, section(17.1), we have

$$\boldsymbol{\phi}(x) = (1, x)^\mathsf{T}, \qquad \mathbf{w} = (a, b)^\mathsf{T}, \tag{17.2.2}$$

We define the train error as the sum of squared differences between the observed outputs and the predictions under the linear model:

$$E(\mathbf{w}) = \sum_{n=1}^{N}(y^n - \mathbf{w}^\mathsf{T}\boldsymbol{\phi}^n)^2, \qquad \text{where } \boldsymbol{\phi}^n \equiv \boldsymbol{\phi}(\mathbf{x}^n) \tag{17.2.3}$$

We now wish to determine the parameter vector $\mathbf{w}$ that minimises $E(\mathbf{w})$. Writing out the error in terms of the components of $\mathbf{w}$,

$$E(\mathbf{w}) = \sum_{n=1}^{N}(y^n - \sum_i w_i\phi_i^n)(y^n - \sum_j w_j\phi_j^n) \tag{17.2.4}$$

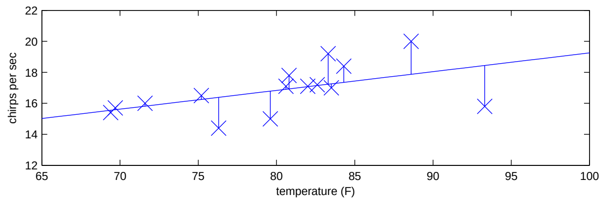
Differentiating with respect to w_k, and equating to zero gives

$$\sum_{n=1}^{N} y^n\phi_k^n = \sum_i w_i \sum_n \phi_i^n\phi_k^n \tag{17.2.5}$$

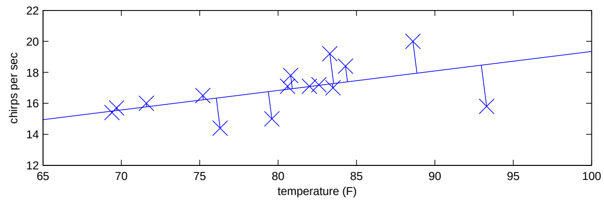
or, in matrix notation,

$$\sum_{n=1}^{N} y^n\boldsymbol{\phi}^n = \sum_{n=1}^{N}\boldsymbol{\phi}^n(\boldsymbol{\phi}^n)^\mathsf{T}\mathbf{w} \tag{17.2.6}$$

[1]Note that the model is linear in the parameter $\mathbf{w}$ – not necessarily linear in $\mathbf{x}$.



(a)



(b)

Figure 17.2: **(a)**: Straight line regression fit to the cricket data. **(b)**: PCA fit to the data. In regression we minimize the *residuals* – the vertical distances from datapoints to the line. In PCA the fit minimizes the orthogonal projections to the line. In this case, there is little difference in the fitted lines. Both go through the mean of the data; the linear regression fit has slope 0.121 and the PCA fit has slope 0.126.

These are called the *normal equations*, for which the solution is

$$\mathbf{w} = \left(\sum_{n=1}^{N} \boldsymbol{\phi}^n (\boldsymbol{\phi}^n)^\mathsf{T}\right)^{-1} \sum_{n=1}^{N} y^n \boldsymbol{\phi}^n \tag{17.2.7}$$

Although we write the solution using matrix inversion, in practice one finds the numerical solution using Gaussian elimination[114] since this is faster and more numerically stable.

Example 77. A cubic polynomial fit

A cubic polynomial is given by

$$y(x) = w_1 + w_2 x + w_3 x^2 + w_4 x^3 \tag{17.2.8}$$

As a LPM, this can be expressed using

$$\boldsymbol{\phi}(x) = \left(1, x, x^2, x^3\right)^\mathsf{T} \tag{17.2.9}$$

The ordinary least squares solution has the form given in equation (17.2.17). The fitted cubic polynomial is plotted in fig(17.3). See also `demoCubicPoly.m`.

Example 78 (Predicting return)**.** In fig(17.4) we present fitting an LPM with vector inputs $\mathbf{x}$ to a scalar output y. The vector $\mathbf{x}$ represents factors that are believed to affect the stock price of a company, with the stock price return given by the scalar y. A hedge fund manager believes that the returns may be linearly related to the factors:

$$y_t = \sum_{i=1}^{5} w_i x_{it} \tag{17.2.10}$$

and wishes to fit the parameters $\mathbf{w}$ in order to use the model to predict future stock returns. This is straightforward using ordinary least squares, this being simply an LPM with a linear ϕ function. See

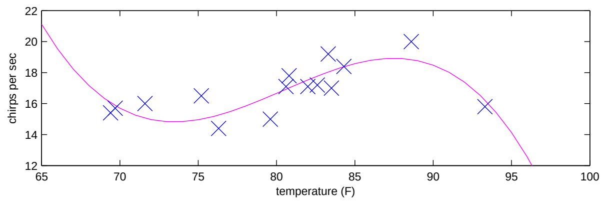


Figure 17.3: Cubic polynomial fit to the cricket data.

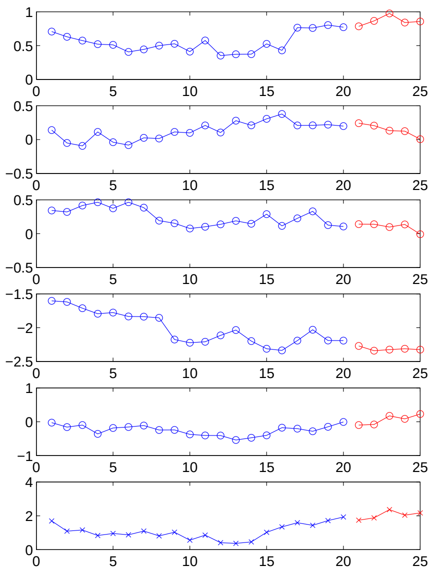


Figure 17.4: Predicting stock return using a linear LPM. The top five panels present the inputs $x_1, \ldots, x_5$ for 20 train days (blue) and 5 test days (red). The corresponding train output (stock return) y for each day is given in the bottom panel. The predictions $y_{21}, \ldots, y_{25}$ are the predictions based on $y_t = \sum_i w_i x_{it}$ with $\mathbf{w}$ trained using ordinary least squares. With a regularisation term $0.01\mathbf{w}^\mathsf{T}\mathbf{w}$, the OLS learned $\mathbf{w}$ is $[1.42, 0.62, 0.27, -0.26, 1.54]$. Despite the simplicity of these models, their application in the finance industry is widespread, with significant investment made on collating factors $\mathbf{x}$ that may be indicative of future return. See `demoLPMhedge.m`.

fig(17.4) for an example. Such models also form the basis for more complex models in finance, see for example [192].

17.2.1 Vector outputs

It is straightforward to generalise the above framework to vector outputs $\mathbf{y}$. Using a separate weight vector $\mathbf{w}_i$ for each output component y_i, we have

$$y_i(\mathbf{x}) = \mathbf{w}_i^\mathsf{T} \boldsymbol{\phi}(\mathbf{x}) \tag{17.2.11}$$

The mathematics follows similarly to before, and we may define a training error per output as

$$E(\mathbf{w}) = \sum_i E(\mathbf{w}_i) = \sum_i \sum_n \left(y_i^n - \mathbf{w}_i^\mathsf{T} \boldsymbol{\phi}^n \right)^2 \tag{17.2.12}$$

Since the training error decomposes into individual terms, one for each output, the weights for each output can be trained separately. In other words, the problem decomposes into a set of independent scalar output problems. In case the parameters $\mathbf{w}$ are tied or shared amongst the outputs, the training is still straightforward since the objective function remains linear in the parameters, and this is left as an exercise for the interested reader.

17.2.2 Regularisation

For most purposes, our interest is not just to find the function that best fits the training data but one that that will generalise well. To control the complexity of the fitted function we may add an extra 'regularising' (or 'penalty') term to the training error to penalise rapid changes in the output. For example a regularising term that can be added to equation (17.2.3) is

$$\lambda \sum_{n=1}^{N} \sum_{n'=1}^{N} e^{-\gamma\left(\mathbf{x}^n - \mathbf{x}^{n'}\right)^2} \left[y(\mathbf{x}^n) - y(\mathbf{x}^{n'}) \right]^2 \tag{17.2.13}$$

The factor $\left[y(\mathbf{x}^n) - y(\mathbf{x}^{n'}) \right]^2$ penalises large differences in the outputs corresponding to two inputs. The factor $e^{-\gamma\left(\mathbf{x}^n - \mathbf{x}^{n'}\right)^2}$ has the effect of weighting more heavily terms for which two input vectors $\mathbf{x}^n$ and $\mathbf{x}^{n'}$ are close together; γ is a fixed length-scale parameter and λ determines the overall strength of the regularising term.

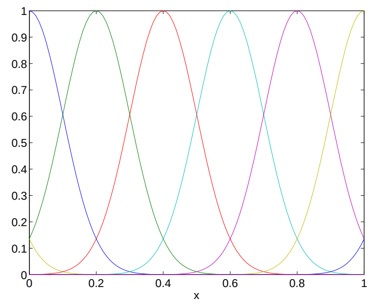


Figure 17.5: A set of fixed-width ($\alpha = 1$) radial basis functions, $e^{-\frac{1}{2}\left(x-m^i\right)^2}$, with the centres m^i evenly spaced. By taking a linear combination of these functions we can form a flexible function class.

Since $y = \mathbf{w}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x})$, expression (17.2.13) can be written as

$$\mathbf{w}^\mathsf{T}\mathbf{R}\mathbf{w} \tag{17.2.14}$$

where

$$\mathbf{R} \equiv \lambda \sum_{n=1}^{N}\sum_{n'=1}^{N} e^{-\gamma\left(\mathbf{x}^n-\mathbf{x}^{n'}\right)^2}\left(\boldsymbol{\phi}^n-\boldsymbol{\phi}^{n'}\right)\left(\boldsymbol{\phi}^n-\boldsymbol{\phi}^{n'}\right)^\mathsf{T} \tag{17.2.15}$$

The regularised train error is then

$$E'(\mathbf{w}) = \sum_{n=1}^{N}(y^n - \mathbf{w}^\mathsf{T}\boldsymbol{\phi}^n)^2 + \mathbf{w}^\mathsf{T}\mathbf{R}\mathbf{w} \tag{17.2.16}$$

By differentiating the regularised training error and equating to zero, we find the optimal $\mathbf{w}$ is given by

$$\mathbf{w} = \left(\sum_n \boldsymbol{\phi}^n(\boldsymbol{\phi}^n)^\mathsf{T} + \mathbf{R}\right)^{-1}\sum_{n=1}^{N} y^n\boldsymbol{\phi}^n \tag{17.2.17}$$

In practice it is common to use a regulariser that penalises the sum square length of the weights

$$\lambda\mathbf{w}^\mathsf{T}\mathbf{w} = \lambda\sum_i w_i^2 \tag{17.2.18}$$

which corresponds to setting $\mathbf{R} = \lambda\mathbf{I}$. Regularising pararameters such as λ, γ may be determined using a validation set, section(13.2.3).

17.2.3 Radial Basis Functions

A popular LPM is given by the non-linear function $\phi(\mathbf{x})$ with components

$$\phi_i(\mathbf{x}) = \exp\left(-\frac{1}{2\alpha^2}\left(\mathbf{x}-\mathbf{m}^i\right)^2\right) \tag{17.2.19}$$

These basis functions are bump shaped, with the center of the bump i being given by $\mathbf{m}^i$ and the width by α. An example is given in fig(17.5) in which several RBFs are plotted with different centres. In LPM regression, we can then use a linear combination of these 'bumps' to fit the data.

One can apply the same approach using vector inputs. For vector $\mathbf{x}$ and centre $\mathbf{m}$, the radial basis function depends on the *distance* between $\mathbf{x}$ and the centre $\mathbf{m}$, giving a 'bump' in input space, fig(17.8).

Example 79 (Setting α)**.** Consider fitting the data in fig(17.6) using 16 radial basis functions uniformly spread over the input space, with width parameter α and regularising term $\lambda\mathbf{w}^\mathsf{T}\mathbf{w}$. The generalisation performance on the test data depends heavily on the width and regularising parameter λ. In order to find reasonable values for these parameters we may use a validation set. For simplicity we set the regularisation parameter to $\lambda = 0.0001$ and use the validation set to determine a suitable α. In fig(17.7) we plot the validation error as a function of α. Based on this graph, we can find the best value of α; that which minimises the validation error. The predictions are also given in fig(17.6).

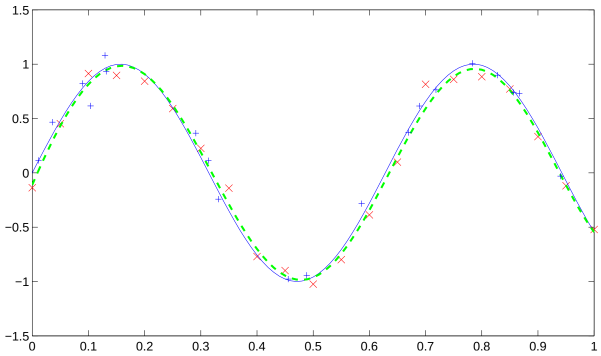


Figure 17.6: The $\times$ are the training points, and the $+$ are the validation points. The solid line is the correct underlying function $\sin(10x)$ which is corrupted with a small amount of additive noise to form the train data. The dashed line is the best predictor based on the validation set.

A curse of dimensionality

If the data has non-trivial behaviour over some region in x, then we need to cover the region of x space fairly densely with 'bump' type functions. In the above case, we used 16 basis functions for this one dimensional space. In 2 dimensions if we wish to cover each dimension to the same discretisation level, we would need $16^2 = 256$ basis functions. Similarly, for 10 dimensions we would need $16^{10} \approx 10^{12}$ functions. To fit such an LPM would require solving a linear system in more than 10^{12} variables. This explosion in the number of basis functions with the input dimension is a 'curse of dimensionality'.

A possible remedy is to make the basis functions very broad so that each covers more of the high dimensional space. However, this will mean a lack of flexibility of the fitted function since it is constrained to be smooth. Another approach is to place basis functions centred on the training input points and add some more basis functions randomly placed close to the training inputs. The rational behind this is that when we come to do prediction, we will most likely see novel x that are close to the training points – we do not need to make 'accurate' predictions over all the space. A further approach is to make the positions of the basis functions adaptive, allowing them to be moved around in the space to minimise the error. This approach motivates the neural network models[41]. An alternative is to reexpress the problem of fitting an LPM by reparameterising the problem, as discussed below.

17.3 The Dual Representation and Kernels

Consider a set of training data with inputs, $\mathcal{X} = \{\mathbf{x}^n, n = 1, \ldots, N\}$ and corresponding outputs $y^n, n = 1, \ldots, N$. For an LPM of the form

$$f(\mathbf{x}) = \mathbf{w}^\mathsf{T}\mathbf{x} \tag{17.3.1}$$

our interest is to find the 'best fit' parameters $\mathbf{w}$. We assume that we have found an optimal parameter $\mathbf{w}_*$. The nullspace of $\mathcal{X}$ are those $\mathbf{x}^\perp$ which are orthogonal to all the inputs in $\mathcal{X}$. That is,

$$\left(\mathbf{x}^\perp\right)^\mathsf{T}\mathbf{x}^n = 0, \tag{17.3.2}$$

for all n. If we then consider the vector $\mathbf{w}_*$ with an additional component in the direction orthogonal to the space spanned by $\mathcal{X}$,

$$\left(\mathbf{w}_* + \mathbf{x}^\perp\right)^\mathsf{T}\mathbf{x}^n = \mathbf{w}_*^\mathsf{T}\mathbf{x}^n \tag{17.3.3}$$

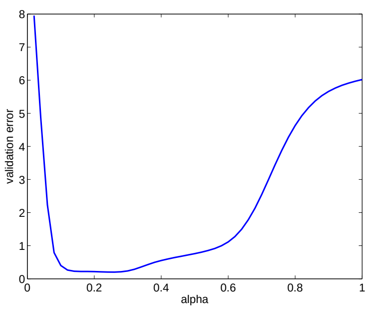
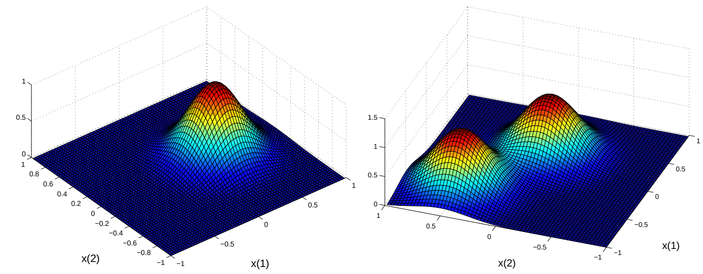


Figure 17.7: The validation error as a function of the basis function width for the validation data in fig(17.6) and RBFs in fig(17.5). Based on the validation error, the optimal setting of the basis function width parameter is $\alpha = 0.25$.



(a) (b)

Figure 17.8: **(a)**: The output of an RBF function $\exp(-\frac{1}{2}\left(\mathbf{x}-\mathbf{m}^1\right)^2/\alpha^2)$. Here $\mathbf{m}^1 = (0, 0.3)^T$ and $\alpha = 0.25$. **(b)**: The combined output for two RBFs with $\mathbf{m}^1$ as above and $\mathbf{m}^2 = (0.5, -0.5)^T$.

This means that adding a contribution to $\mathbf{w}_*$ outside of the space spanned by $\mathcal{X}$, has no effect on the predictions on the train data. If the training criterion depends only on how well the LPM predicts the train data, there is therefore no need to consider contributions to $\mathbf{w}$ from outside of $\mathcal{X}$. That is, without loss of generality we may consider the representation

$$\mathbf{w} = \sum_{n=1}^{N} a_n \mathbf{x}^n \tag{17.3.4}$$

The parameters $\mathbf{a} = (a_1, \ldots, a_N)$ are called the *dual parameters*. We can then write the output of the LPM directly in terms of the dual parameters,

$$\mathbf{w}^\mathsf{T}\mathbf{x}^n = \sum_{m=1}^{N} a_m \left(\mathbf{x}^m\right)^\mathsf{T} \mathbf{x}^n \tag{17.3.5}$$

More generally, for a vector function $\boldsymbol{\phi}(\mathbf{x})$, the solution will lie in the space spanned by $\boldsymbol{\phi}(\mathbf{x}^1), \ldots, \boldsymbol{\phi}(\mathbf{x}^N)$,

$$\mathbf{w} = \sum_{n=1}^{N} a_n \boldsymbol{\phi}\left(\mathbf{x}^n\right) \tag{17.3.6}$$

and we may write

$$\mathbf{w}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}^n) = \sum_{m=1}^{N} a_m \boldsymbol{\phi}\left(\mathbf{x}^m\right)^\mathsf{T} \boldsymbol{\phi}\left(\mathbf{x}^n\right) = \sum_{m=1}^{N} a_m K\left(\mathbf{x}^m, \mathbf{x}^n\right) \tag{17.3.7}$$

where we have defined a *kernel* function

$$K\left(\mathbf{x}^m, \mathbf{x}^n\right) \equiv \boldsymbol{\phi}\left(\mathbf{x}^m\right)^\mathsf{T} \boldsymbol{\phi}\left(\mathbf{x}^n\right) \equiv \left[\mathbf{K}\right]_{m,n} \tag{17.3.8}$$

In matrix form, the output of the LPM on a training input $\mathbf{x}$ is then

$$\mathbf{w}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}^n) = \left[\mathbf{K}\mathbf{a}\right]_n = \mathbf{a}^\mathsf{T}\mathbf{k}^n \tag{17.3.9}$$

where $\mathbf{k}^n$ is the n^{th} column of the *Gram matrix* $\mathbf{K}$.

17.3.1 Regression in the dual-space

For ordinary least squares regression, using equation (17.3.9), we have a train error

$$E(\mathbf{a}) = \sum_{n=1}^{N} \left(y^n - \mathbf{a}^\mathsf{T}\mathbf{k}^n\right)^2 \tag{17.3.10}$$

Equation(17.3.10) is analogous to the standard regression equation (17.2.3) on interchanging $\mathbf{a}$ for $\mathbf{w}$ and $\mathbf{k}^n$ for $\boldsymbol{\phi}(\mathbf{x}^n)$. Similarly, the regularisation term can be expressed as

$$\mathbf{w}^\mathsf{T}\mathbf{w} = \sum_{n,m=1}^{N} a_n a_m \boldsymbol{\phi}\left(\mathbf{x}^n\right) \boldsymbol{\phi}\left(\mathbf{x}^m\right) = \mathbf{a}^\mathsf{T}\mathbf{K}\mathbf{a} \tag{17.3.11}$$

By direct analogy the optimal solution for $\mathbf{a}$ is therefore

$$\mathbf{a} = \left(\sum_{n=1}^{N} \mathbf{k}^n \left(\mathbf{k}^n\right)^{\mathsf{T}} + \lambda \mathbf{K}\right)^{-1} \sum_{n=1}^{N} y^n \mathbf{k}^n \tag{17.3.12}$$

We can express the above solution more conveniently by considering

$$\mathbf{a} = \left(\sum_{n=1}^{N} \mathbf{K}^{-1}\mathbf{k}^n \left(\mathbf{k}^n\right)^{\mathsf{T}} + \lambda \mathbf{I}\right)^{-1} \sum_{n=1}^{N} y^n \mathbf{K}^{-1}\mathbf{k}^n \tag{17.3.13}$$

Since $\mathbf{k}^n$ is the n^{th} column of $\mathbf{K}$ then $\mathbf{K}^{-1}\mathbf{k}^n$ is the n^{th} column of the identity matrix. With a little thought, we can rewrite equation (17.3.13) more simply as

$$\mathbf{a} = \left(\mathbf{K} + \lambda \mathbf{I}\right)^{-1} \mathbf{y} \tag{17.3.14}$$

where $\mathbf{y}$ is the vector with components formed from the training inputs $y^1, \ldots, y^N$.

Using this, the prediction for a new input $\mathbf{x}^*$ is given by

$$y(\mathbf{x}^*) = \mathbf{k}_*^{\mathsf{T}} \left(\mathbf{K} + \lambda \mathbf{I}\right)^{-1} \mathbf{y} \tag{17.3.15}$$

where the vector $\mathbf{k}_*$ has components

$$\left[\mathbf{k}_*\right]_m = K\left(\mathbf{x}^*, \mathbf{x}^m\right) \tag{17.3.16}$$

This dual space solution shows that predictions can be expressed purely in terms of the kernel $K\left(\mathbf{x}, \mathbf{x}'\right)$. This means that we may dispense with defining the vector functions $\boldsymbol{\phi}(\mathbf{x})$ and define a kernel function directly. This approach is also used in Gaussian Processes, chapter(19) and enables us to use effectively very large (even infinite) dimensional vectors $\boldsymbol{\phi}$ without ever explicitly needing to compute them. Note that the Gram matrix $\mathbf{K}$ has dimension $N \times N$, which means that the computational complexity of performing the matrix inversion in equation (17.3.16) is $O\left(N^3\right)$. For moderate to large N (greater than 5000), this will be prohibitively expensive, and numerical approximations are required. This is in contrast to the computational complexity of solving the normal equations in the original weight space viewpoint is $O\left(\dim\left(\boldsymbol{\phi}\right)^3\right)$.

To an extent, the dual parameterisation helps us with the curse of dimensionality since the complexity of learning in the dual parameterisation scales cubically with the number of training points – not cubically with the dimension of the $\boldsymbol{\phi}$ vector.

17.3.2 Positive definite kernels (covariance functions)

The kernel $K\left(\mathbf{x}, \mathbf{x}'\right)$ in (17.3.8) was defined as the scalar product between two vectors $\boldsymbol{\phi}(\mathbf{x})$ and $\boldsymbol{\phi}(\mathbf{x}')$. For any set of points $\mathbf{x}^1, \ldots, \mathbf{x}^M$, the resulting matrix

$$\left[\mathbf{K}\right]_{m,n} = \boldsymbol{\phi}(\mathbf{x}^m)^{\mathsf{T}}\boldsymbol{\phi}(\mathbf{x}^n) = \sum_i \phi_i^m \phi_i^n \tag{17.3.17}$$

is positive semi-definite since for any $\mathbf{z}$

$$\mathbf{z}^{\mathsf{T}}\mathbf{K}\mathbf{z} = \sum_{m,n} z_m K_{mn} z_n = \sum_i \left(\sum_m z_m \phi_i^m\right)\left(\sum_n z_n \phi_i^n\right) = \sum_i \left(\sum_m z_m \phi_i^m\right)^2 \geq 0 \tag{17.3.18}$$

Instead of specifying high-dimensional $\boldsymbol{\phi}(\mathbf{x})$ vectors, we may instead specify a function $K(\mathbf{x}, \mathbf{x}')$ that produces a positive definite matrix $\mathbf{K}$ for any inputs $\mathbf{x}, \mathbf{x}'$. Such a function is called a covariance function, or a positive kernel. For example a popular choice is

$$e^{-\lambda|x-x'|^{\nu}}, 0 < \nu \leq 2, \lambda \geq 0 \tag{17.3.19}$$

For $\nu = 2$ this is commonly called the *squared exponential* kernel. For $\nu = 1$ this is known as the *Ornstein-Uhlenbeck* kernel. Covariance functions are discussed in more detail in section(19.3).

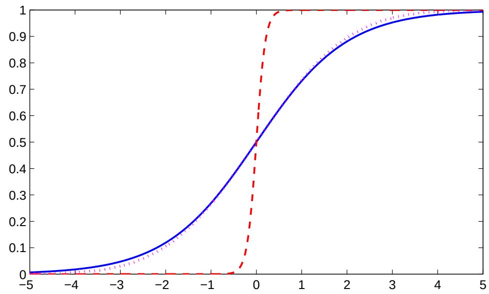


Figure 17.9: The logistic sigmoid function $\sigma_\beta(x) = 1/(1+e^{-\beta x})$. The parameter β determines the steepness of the sigmoid. The full (blue) line is for $\beta = 1$ and the dashed (red) for $\beta = 10$. As $\beta \to \infty$, the logistic sigmoid tends to a Heaviside step function. The dotted curve (magenta) is the error function (probit) $0.5\,(1 + \text{erf}(\lambda x))$ for $\lambda = \sqrt{\pi}/4$, which closely matches the standard logistic sigmoid with $\beta = 1$.

17.4 Linear Parameter Models for Classification

In a binary classification problem we are given some training data, $\mathcal{D} = \{(\mathbf{x}^n, c^n), n = 1 \ldots N\}$, where the targets $c \in \{0, 1\}$. Inspired by the LPM regression model, we can assign the probability that a novel input $\mathbf{x}$ belongs to class 1 using

$$p(c = 1|\mathbf{x}) = f(\mathbf{x}^\mathsf{T}\mathbf{w}) \tag{17.4.1}$$

where $0 \le f(x) \le 1$. In the statistics literature, $f(x)$ is termed a mean function – the inverse function $f^{-1}(x)$ is the link function.

Two popular choices for the function $f(x)$ are the and probit functions. The *logit* is given by

$$f(x) = \frac{e^x}{1 + e^x} = \frac{1}{1 + e^{-x}} \tag{17.4.2}$$

which is also called the *logistic sigmoid* and written $\sigma(x)$, fig(17.9). The scaled version is defined as

$$\sigma_\beta(x) = \sigma(\beta x) \tag{17.4.3}$$

A closely related model is *probit regression* which uses in place of the logistic sigmoid the error function the cumulative distribution of the standard normal distribution

$$f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{1}{2}t^2} dt = \frac{1}{2}\left(1 + \text{erf}(x)\right) \tag{17.4.4}$$

This can also be written in terms of the standard *error function*,

$$\text{erf}(x) \equiv \frac{2}{\sqrt{\pi}} \int_{0}^{x} e^{-t^2} dt \tag{17.4.5}$$

The shape of the probit and logistic functions are similar under rescaling, see fig(17.9). We focus below on the logit function. Similar derivations carry over in a straightforward manner to any monotonic mean function.

17.4.1 Logistic Regression

Logistic regression corresponds to the model

$$p(c = 1|\mathbf{x}) = \sigma(b + \mathbf{x}^\mathsf{T}\mathbf{w}) \tag{17.4.6}$$

where b is a scalar, and $\mathbf{w}$ is a vector. As the argument $b + \mathbf{x}^\mathsf{T}\mathbf{w}$ of the sigmoid function increases, the probability $\mathbf{x}$ belongs to class 1 increases.

The decision boundary

The decision boundary is defined as that set of $\mathbf{x}$ for which $p(c = 1|\mathbf{x}) = p(c = 0|\mathbf{x}) = 0.5$. This is given by the hyperplane

$$b + \mathbf{x}^\mathsf{T}\mathbf{w} = 0 \tag{17.4.7}$$

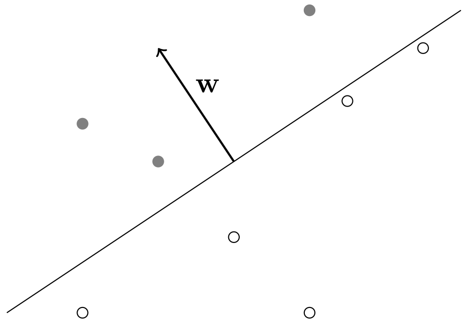


Figure 17.10: The decision boundary $p(c = 1|\mathbf{x}) = 0.5$ (solid line). For two dimensional data, the decision boundary is a line. If all the training data for class 1 (filled circles) lie on one side of the line, and for class 0 (open circles) on the other, the data is said to be *linearly separable*.

On the side of the hyperplane for which $b + \mathbf{x}^\mathsf{T}\mathbf{w} > 0$, inputs $\mathbf{x}$ are classified as 1's, and on the other side they are classified as 0's. The 'bias' parameter b simply shifts the decision boundary by a constant amount. The orientation of the decision boundary is determined by $\mathbf{w}$, the normal to the hyperplane, see fig(17.10).

To clarify the geometric interpretation, let $\mathbf{x}$ be a point on the decision boundary and consider a new point $\mathbf{x}^* = \mathbf{x} + \mathbf{w}^\perp$, where $\mathbf{w}^\perp$ is a vector perpendicular to $\mathbf{w}$, so that $\mathbf{w}^\mathsf{T}\mathbf{w}^\perp = 0$. Then

$$b + \mathbf{w}^\mathsf{T}\mathbf{x}^* = b + \mathbf{w}^\mathsf{T}\left(\mathbf{x} + \mathbf{w}^\perp\right) = b + \mathbf{w}^\mathsf{T}\mathbf{x} + \mathbf{w}^\mathsf{T}\mathbf{w}^\perp = b + \mathbf{w}^\mathsf{T}\mathbf{x} = 0 \tag{17.4.8}$$

Thus if $\mathbf{x}$ is on the decision boundary, so is $\mathbf{x}$ plus any vector perpendicular to $\mathbf{w}$. In D dimensions, the space of vectors that are perpendicular to $\mathbf{w}$ occupy a $D - 1$ dimensional hyperplane. For example, if the data is two dimensional, the decision boundary is a one dimensional hyperplane, a line, as depicted in fig(17.10).

Linear Separability and linear independence

Definition 90 (Linear separability)**.** If all the training data for class 1 lies on one side of a hyperplane, and for class 0 on the other, the data is said to be linearly separable.

For D dimensional data, provided there are no more than D training points, then these are linearly separable provided they are linearly independent. To see this, let $c^n = +1$ if $\mathbf{x}^n$ is in class 1, and $c^n = -1$ if $\mathbf{x}^n$ is in class 0. For the data to be linearly separable we require

$$\mathbf{w}^\mathsf{T}\mathbf{x}^n + b = \epsilon c^n, \qquad n = 1, \ldots, N \tag{17.4.9}$$

where ϵ is an arbitrarily small positive constant. The above equations state that each input is just the correct side of the decision boundary. If there are $N = D$ datapoints, the above can be written in matrix form as

$$\mathbf{X}\mathbf{w} + \mathbf{b} = \mathbf{c} \tag{17.4.10}$$

where $\mathbf{X}$ is a square matrix whose n^{th} column contains $\mathbf{x}^n$. Provided that $\mathbf{X}$ is invertible the solution is

$$\mathbf{w} = \mathbf{X}^{-1}(\mathbf{c} - \mathbf{b}) \tag{17.4.11}$$

The bias $\mathbf{b}$ can be set arbitrarily. This shows that provided the $\mathbf{x}^n$ are linearly independent, we can always find a hyperplane that linearly separates the data. Provided the data are not-collinear (all occupying the same $D - 1$ dimensional subspace) the bias can be used to improve this to enabling $D + 1$ arbitrarily labelled points to be linearly separated in D dimensions.

A dataset that is not linearly separable is given by the following four training points and class labels

$$\{([0, 0], 0), ([0, 1], 1), ([1, 0], 1), ([1, 1], 0)\} \tag{17.4.12}$$

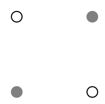


Figure 17.11: The XOR problem. This is not linearly separable.

This data represents the XOR function, and is plotted in fig(17.11). This function is not linearly separable since no straight line has all inputs from one class on one side and the other class on the other.

Classifying data which is not linearly separable can only be achieved using a non-linear decision boundary. It might be that data is non-linearly separable in the original data space. However, by mapping to a higher dimension using a non-linear vector function, we generate a set of non-linearly dependent high-dimensional vectors, which can then be separated using a high-dimensional hyperplane. We will discuss this in section(17.5).

The Perceptron

The perceptron assigns $\mathbf{x}$ to class 1 if $b + \mathbf{w}^\mathsf{T}\mathbf{x} \geq 0$, and to class 0 otherwise. That is

$$p(c = 1|\mathbf{x}) = \theta(b + \mathbf{x}^\mathsf{T}\mathbf{w}) \tag{17.4.13}$$

where the step function is defined as

$$\theta(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases} \tag{17.4.14}$$

If we consider the logistic regression model

$$p(c = 1|\mathbf{x}) = \sigma_\beta \left(b + \mathbf{x}^\mathsf{T}\mathbf{w}\right) \tag{17.4.15}$$

and take the limit $\beta \to \infty$, we have the perceptron like classifier

$$p(c = 1|\mathbf{x}) = \begin{cases} 1 & b + \mathbf{x}^\mathsf{T}\mathbf{w} > 0 \\ 0.5 & b + \mathbf{x}^\mathsf{T}\mathbf{w} = 0 \\ 0 & b + \mathbf{x}^\mathsf{T}\mathbf{w} < 0 \end{cases} \tag{17.4.16}$$

The only difference between this 'probabilistic perceptron' and the standard perceptron is in the technical definition of the value of the step function at 0. The perceptron may therefore essentially be viewed as a limiting case of logistic regression.

17.4.2 Maximum likelihood training

Given a data set $\mathcal{D}$, how can we learn the weights to obtain good classification? Probabilistically, if we assume that each data point has been drawn independently from the same distribution that generates the data (the standard i.i.d. assumption), the likelihood of the observed data is (writing explicitly the conditional dependence on the parameters $b, \mathbf{w}$)

$$p(\mathcal{D}|b, \mathbf{w}) = \prod_{n=1}^{N} p(c^n|\mathbf{x}^n, b, \mathbf{w})p(\mathbf{x}^n) = \prod_{n=1}^{N} p(c = 1|\mathbf{x}^n, b, \mathbf{w})^{c^n} \left(1 - p(c = 1|\mathbf{x}^n, b, \mathbf{w})\right)^{1-c^n} p(\mathbf{x}^n) \tag{17.4.17}$$

where we have used the fact that $c^n \in \{0, 1\}$. For this discriminative model, we do not model the input distribution $p(\mathbf{x})$ so that we may equivalently consider the log likelihood of the output class variables conditioned on the training inputs: For logistic regression this gives

$$L(\mathbf{w}, b) = \sum_{n=1}^{N} c^n \log \sigma(b + \mathbf{w}^\mathsf{T}\mathbf{x}^n) + (1 - c^n) \log \left(1 - \sigma(b + \mathbf{w}^\mathsf{T}\mathbf{x}^n)\right) \tag{17.4.18}$$

Gradient Ascent

There is no closed form solution to the maximisation of $L(\mathbf{w}, b)$ which needs to be carried out numerically. One of the simplest methods is gradient ascent for which the gradient is given by

$$\nabla_{\mathbf{w}} L = \sum_{n=1}^{N} (c^n - \sigma(\mathbf{w}^\mathsf{T}\mathbf{x}^n + b))\mathbf{x}^n \tag{17.4.19}$$

Here we made use of the derivative relation

$$d\sigma(x)/dx = \sigma(x)(1 - \sigma(x)) \tag{17.4.20}$$

for the logistic sigmoid. The derivative with respect to the bias is

$$\frac{dL}{db} = \sum_{n=1}^{N} (c^n - \sigma(\mathbf{w}^\mathsf{T}\mathbf{x}^n + b)) \tag{17.4.21}$$

The gradient ascent procedure then corresponds to updating the weights and bias using

$$\mathbf{w}^{new} = \mathbf{w} + \eta \nabla_{\mathbf{w}} L, \qquad b^{new} = b + \eta \frac{dL}{db} \tag{17.4.22}$$

where η, the *learning rate* is a scalar chosen small enough to ensure convergence. The application of the above rule will lead to a gradual increase in the log likelihood.

Batch Training

Writing the updates (17.4.22) explicitly gives

$$\mathbf{w}^{new} = \mathbf{w} + \eta \sum_{n=1}^{N} (c^n - \sigma(\mathbf{w}^\mathsf{T}\mathbf{x}^n + b))\mathbf{x}^n, \qquad b^{new} = b + \eta \sum_{n=1}^{N} (c^n - \sigma(\mathbf{w}^\mathsf{T}\mathbf{x}^n + b)) \tag{17.4.23}$$

This is called a *batch update* since the parameters $\mathbf{w}$ and b are updated only after passing through the whole (batch) of training data. This batch version 'converges' in all cases since the error surface is bowl shaped (see below). For linearly separable data, however, the optimal setting is for the weights to become infinitely large, since then the logistic sigmoids will saturate to 1 or 0 (see below). A stopping criterion based on either minimal changes to the log likelihood or the parameters is therefore required to halt the optimisation routine. For non-linearly separable data, the likelihood has a maximum at finite $\mathbf{w}$ so the algorithm converges. However, the predictions will be less certain, reflected in a broad confidence interval – see fig(17.12).

In batch training, the zero gradient criterion is

$$\sum_{n=1}^{N} (c^n - \sigma(\mathbf{w}^\mathsf{T}\mathbf{x}^n + b))\mathbf{x}^n = \mathbf{0} \tag{17.4.24}$$

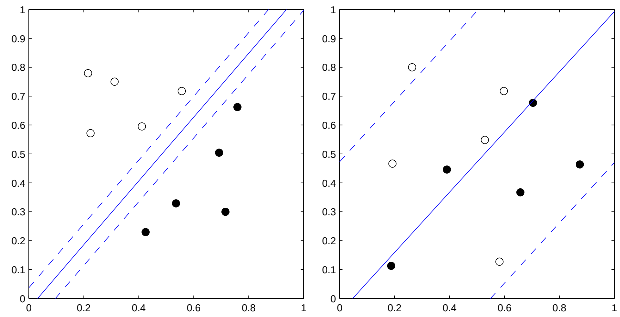
In the case that the inputs $\mathbf{x}^n, n = 1, \ldots, N$ are linearly independent, we immediately have the requirement that for a zero gradient, $c^n = \sigma\left(\mathbf{w}^\mathsf{T}\mathbf{x}^n + b\right)$, meaning that the weights must tend to infinity for this condition to hold.

For linearly separable data, we can also show that the weights must become infinite at convergence. Taking the scalar product of equation (17.4.24) with $\mathbf{w}$, we have the zero gradient requirement

$$\sum_{n=1}^{N} (c^n - \sigma^n)\, \mathbf{w}^\mathsf{T}\mathbf{x}^n = 0 \tag{17.4.25}$$

where $\sigma^n \equiv \sigma\left(\mathbf{w}^\mathsf{T}\mathbf{x}^n + b\right)$. For simplicity we assume $b = 0$. For linearly separable data we have

$$\mathbf{w}^\mathsf{T}\mathbf{x}^n \begin{cases} > 0 & c = 1 \\ < 0 & c = 0 \end{cases} \tag{17.4.26}$$



(a) (b)

Figure 17.12: The decision boundary $p(c = 1|\mathbf{x}) = 0.5$ (solid line) and confidence boundaries $p(c = 1|\mathbf{x}) = 0.9$ and $p(c = 1|\mathbf{x}) = 0.1$ and 10000 iterations of batch gradient ascent with $\eta = 0.1$. **(a)**: Linearly separable data. **(b)**: Non-linearly separable data. Note how the confidence interval remains broad, see `demoLogReg.m`.

Then, using the fact that $0 \leq \sigma^n \leq 1$, we have

$$(c^n - \sigma^n)\,\mathbf{w}^\mathsf{T}\mathbf{x}^n \begin{cases} \geq 0 & c = 1 \\ \geq 0 & c = 0 \end{cases} \tag{17.4.27}$$

Each term $(c^n - \sigma^n)\,\mathbf{w}^\mathsf{T}\mathbf{x}^n$ is non-negative and the zero gradient condition requires the sum of these terms to be zero. This can only happen if all the terms are zero, implying that $c^n = \sigma^n$, requiring the sigmoid to saturate, for which the weights must be infinite.

Online Training

In practice it is common to update the parameters after each training example has been considered:

$$\mathbf{w}^{new} = \mathbf{w} + \frac{\eta}{N}(c^n - \sigma(\mathbf{w}^\mathsf{T}\mathbf{x}^n + b))\mathbf{x}^n, \qquad b^{new} = b + \frac{\eta}{N}(c^n - \sigma(\mathbf{w}^\mathsf{T}\mathbf{x}^n + b)) \tag{17.4.28}$$

An advantage of online training is that the dataset does not need to be stored since only the performance on the current input is required. Provided that the data is linearly separable, the above online procedure converges (provided η is not too large). However, if the data is not linearly separable, the online version will not converge since the opposing class labels will continually pull the weights one way and then the other as each conflicting example is used to form an update. For the limiting case of the perceptron (replacing $\sigma(x)$ with $\theta(x)$) and linearly separable data, online updating converges in a finite number of steps[210, 41], but does not converge for non-linearly separable data.

Geometry of the error surface

The Hessian of the log likelihood $L(\mathbf{w})$ is the matrix with elements[2]

$$H_{ij} \equiv \frac{\partial^2 L}{\partial w_i w_j} = -\sum_n x_i^n x_j^n \sigma^n(1 - \sigma^n) \tag{17.4.29}$$

This is negative (semi) definite since for any $\mathbf{z}$

$$\sum_{ij} z_i H_{ij} z_j = -\sum_{i,j,n} z_i x_i^n z_j x_j^n \sigma^n(1 - \sigma^n) \leq -\left(\sum_{i,n} z_i x_i^n\right)^2 \leq 0 \tag{17.4.30}$$

This means that the error surface is concave (an upside down bowl) and gradient ascent is guaranteed to converge to the optimal solution, provided the learning rate η is small enough.

Example 80 (Classifying Handwritten Digits)**.** We apply logistic regression to the 600 handwritten digits of example(67), in which there are 300 ones and 300 sevens in the train data. Using gradient ascent training with a suitably chosen stopping criterion, the number of errors made on the 600 test points is 12, compared with 14 errors using Nearest Neighbour methods. See fig(17.13) for a visualisation of the learned $\mathbf{w}$.

[2]For simplicity we ignore the bias b. This can readily be dealt with by extending $\mathbf{x}$ to a $D+1$ dimensional vector $\hat{\mathbf{x}}$ with a 1 in the $D+1$ component. Then for a $D+1$ dimensional $\hat{\mathbf{w}} = (\mathbf{w}, w_{D+1})$, we have $\hat{\mathbf{w}}^\mathsf{T}\hat{\mathbf{x}} = \mathbf{w}^\mathsf{T}\mathbf{x} + w_{D+1}$.

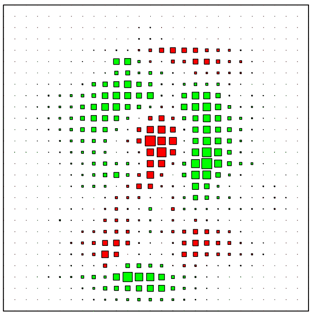


Figure 17.13: Logistic regression for classifying hand written digits 1 and 7. Displayed is a Hinton diagram of the 784 learned weight vector $\mathbf{w}$, plotted as a 28×28 image for visual interpretation. Green are positive and an input $\mathbf{x}$ with a (positive) value in this component will tend to increase the probability that the input is classed as a 7. Similarly, inputs with positive contributions in the red regions tend to increase the probability as being classed as a 1 digit. Note that the elements of each input $\mathbf{x}$ are either positive or zero.

17.4.3 Beyond first order gradient ascent

Since the surface has a single optimum, a Newton update

$$\mathbf{w}^{new} = \mathbf{w}^{old} + \eta \mathbf{H}^{-1}\mathbf{w}^{old} \tag{17.4.31}$$

where $\mathbf{H}$ is the Hessian matrix as above and $0 < \eta < 1$, will typically converge much faster than gradient ascent. For large scale problems, the inversion of the Hessian is computationally demanding and limited memory BFGS or conjugate gradient methods may be considered as more practical alternatives, see section(A.5).

17.4.4 Avoiding overconfident classification

Provided the data is linearly separable the weights will continue to increase and the classifications will become extreme. This is undesirable since the classifications will be over-confident. This can be prevented by adding a penalty term to the objective function

$$L'(\mathbf{w}, b) = L(\mathbf{w}, b) - \alpha \mathbf{w}^\mathsf{T}\mathbf{w}. \tag{17.4.32}$$

The scalar constant $\alpha > 0$ encourages smaller values of $\mathbf{w}$ (remember that we wish to maximise the log likelihood). An appropriate value for α can be determined using validation data.

17.4.5 Multiple Classes

For more than two classes, one may use the *softmax function*

$$p(c = i|\mathbf{x}) = \frac{e^{\mathbf{w}_i^\mathsf{T}\mathbf{x}+b_i}}{\sum_{j=1}^{C} e^{\mathbf{w}_j^\mathsf{T}\mathbf{x}+b_j}} \tag{17.4.33}$$

where C is the number of classes. When $C = 2$ this reduced to the logistic sigmoid. One can show that the likelihood for this case is also concave, see exercise(175) and [294].

17.5 The Kernel Trick for Classification

A drawback of logistic regression as described above is the simplicity of the decision surface – a hyperplane. One way to extend the method to more complex non-linear decision boundaries is to consider mapping the inputs $\mathbf{x}$ in a non-linear way to $\boldsymbol{\phi}(\mathbf{x})$:

$$p(c = 1|\mathbf{x}) = \sigma\left(\mathbf{w}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}) + b\right) \tag{17.5.1}$$

For example, the one-dimensional input x could get mapped to a two dimensional vector $(x^2, \sin(x))$. Mapping into a higher dimensional space makes it easier to find a separating hyperplane since any set of points that are linearly independent can be linearly separated provided we have as many dimensions as datapoints.

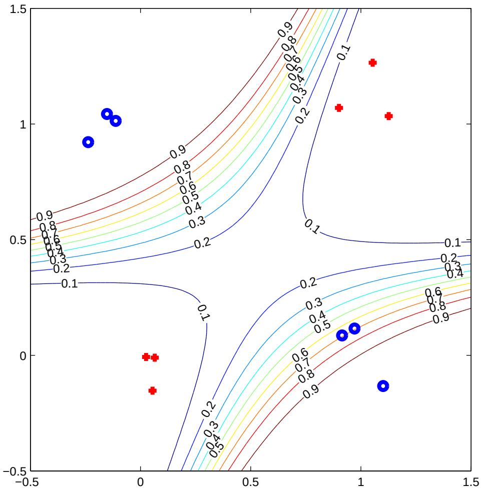


Figure 17.14: Logistic regression $p(c = 1|\mathbf{x}) = \sigma(\mathbf{w}^{\mathsf{T}}\boldsymbol{\phi}(\mathbf{x}))$ using a quadratic function $\boldsymbol{\phi}(\mathbf{x}) = (1, x_1, x_2, x_1^2, x_2^2, x_1 x_2)^{\mathsf{T}}$. 1000 iterations of gradient ascent training were performed with a learning rate $\eta = 0.1$. Plotted are the datapoints for the two classes (red cross) and (blue circle) and the equal probability contours. The decision boundary is the 0.5-probability contour. See `demoLogRegNonLinear.m`.

For the Maximum Likelihood criterion, we can use exactly the same algorithm as before on replacing $\mathbf{x}$ with $\boldsymbol{\phi}(\mathbf{x})$. See fig(17.14) for a demonstration using a quadratic function.

Since only the scalar product between the $\boldsymbol{\phi}$ vectors plays a role the dual representation may be used in which we assume the weight can be expressed in the form

$$\mathbf{w} = \sum_n \alpha_n \boldsymbol{\phi}^n \tag{17.5.2}$$

We then subsequently find a solution in terms of the dual parameters α_n. This is potentially advantageous since there may be less training points than dimensions of $\boldsymbol{\phi}$. The classifier depends only on scalar products which can be written in terms of a positive definite kernel,

$$p(c = 1|\mathbf{x}) = \sigma\left(\sum_n a_n K(\mathbf{x}, \mathbf{x}^n)\right) \tag{17.5.3}$$

For convenience, we can write the above as

$$p(c = 1|\mathbf{x}) = \sigma\left(\mathbf{a}^{\mathsf{T}}\mathbf{k}(\mathbf{x})\right) \tag{17.5.4}$$

where the N dimensional vector $\mathbf{k}(\mathbf{x})$ has elements $[\mathbf{k}(\mathbf{x})]_n = K(\mathbf{x}, \mathbf{x}^n)$. Then the above is of exactly the same form as the original specification of logistic regression, namely as a function of a linear combination of vectors. Hence the same training algorithm to maximise the likelihood can be employed, simply on replacing $\mathbf{x}^n$ with $k(\mathbf{x}^n)$. The details are left to the interested reader, and follow closely the treatment of Gaussian Processes for classification, section(19.6).

17.6 Support Vector Machines

Like kernel logistic regression, SVMs are a form of kernel linear classifier. However, the SVM uses an objective which more explicitly encourages good generalisation performance. SVMs do not fit comfortably within a probabilistic framework and as such we describe them here only briefly, referring the reader to the wealth of excellent literature on this topic[3]. The description here is inspired largely by [70].

17.6.1 Maximum margin linear classifier

In the SVM literature it is common to use +1 and −1 to denote the two classes. For a hyperplane defined by weight $\mathbf{w}$ and bias b, a linear discriminant is given by

$$\mathbf{w}^{\mathsf{T}}\mathbf{x} + b \geq 0 \qquad \text{class } +1 \tag{17.6.1}$$

$$\mathbf{w}^{\mathsf{T}}\mathbf{x} + b < 0 \qquad \text{class -1} \tag{17.6.2}$$

[3] `http://www.support-vector.net`

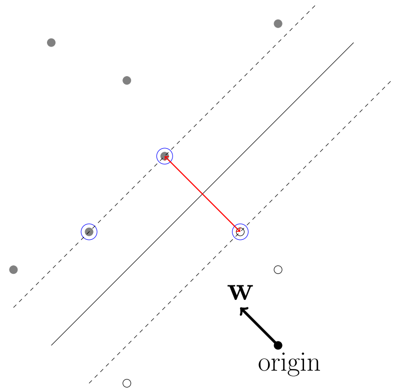


Figure 17.15: SVM classification of data from two classes (open circles and filled circles). The decision boundary $\mathbf{w}^\mathsf{T}\mathbf{x} + b = 0$ (solid line). For linearly separable data the maximum margin hyperplane is equidistant from the closest opposite class points. These support vectors are highlighted in blue and the margin in red. The distance of the decision boundary from the origin is $-b/\sqrt{\mathbf{w}^\mathsf{T}\mathbf{w}}$, and the distance of a general point $\mathbf{x}$ from the origin along the direction $\mathbf{w}$ is $\mathbf{x}^\mathsf{T}\mathbf{w}/\sqrt{\mathbf{w}^\mathsf{T}\mathbf{w}}$.

For a point $\mathbf{x}$ that is close to the decision boundary at $\mathbf{w}^\mathsf{T}\mathbf{x} + b = 0$, a small change in $\mathbf{x}$ can lead to a change in classification. To make the classifier more robust we therefore impose that for the training data at least, the decision boundary should be separated from the data by some finite margin (assuming in the first instance that the data is linearly separable):

$$\mathbf{w}^\mathsf{T}\mathbf{x} + b \geq \epsilon^2 \qquad \text{class } +1 \tag{17.6.3}$$

$$\mathbf{w}^\mathsf{T}\mathbf{x} + b < -\epsilon^2 \qquad \text{class -1} \tag{17.6.4}$$

Since $\mathbf{w}$, b and ϵ^2 can all be rescaled arbitrary, we need to fix the scale of the above to break this invariance. It is convenient to set $\epsilon = 1$ so that a point $\mathbf{x}_+$ from class +1 that is closest to the decision boundary satisfies

$$\mathbf{w}^\mathsf{T}\mathbf{x}_+ + b = 1 \tag{17.6.5}$$

and a point $\mathbf{x}_-$ from class -1 that is closest to the decision boundary satisfies

$$\mathbf{w}^\mathsf{T}\mathbf{x}_- + b = -1 \tag{17.6.6}$$

From vector algebra, fig(17.15), the distance from the origin along the direction $\mathbf{w}$ to a point $\mathbf{x}$ is given by

$$\frac{\mathbf{w}^\mathsf{T}\mathbf{x}}{\sqrt{\mathbf{w}^\mathsf{T}\mathbf{w}}} \tag{17.6.7}$$

The *margin* between the hyperplanes for the two classes is then the difference between the two distances along the direction $\mathbf{w}$ which is

$$\frac{\mathbf{w}^\mathsf{T}}{\sqrt{\mathbf{w}^\mathsf{T}\mathbf{w}}}(\mathbf{x}_+ - \mathbf{x}_-) = \frac{2}{\sqrt{\mathbf{w}^\mathsf{T}\mathbf{w}}} \tag{17.6.8}$$

To set the distance between the two hyperplanes to be maximal, we need to minimise the length $\mathbf{w}^\mathsf{T}\mathbf{w}$. Given that for each $\mathbf{x}^n$ we have a corresponding label $y^n \in \{+1, -1\}$, in order to classify the training labels correctly and maximise the margin, the optimisation problem is therefore equivalent to:

$$\text{minimise } \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} \qquad \text{subject to } y^n\left(\mathbf{w}^\mathsf{T}\mathbf{x}^n + b\right) \geq 1, \qquad n = 1, \ldots, N \tag{17.6.9}$$

This is a *quadratic programming* problem. Note that the factor 0.5 is just for convenience.

To account for potentially mislabelled training points (or for data that is not linearly separable), we relax the exact classification constraint and use instead

$$y^n\left(\mathbf{w}^\mathsf{T}\mathbf{x}^n + b\right) \geq 1 - \xi^n \tag{17.6.10}$$

where $\xi^n \geq 0$. Here each ξ^n measures how far $\mathbf{x}^n$ is from the correct margin. For $0 < \xi^n < 1$ datapoint $\mathbf{x}^n$ is on the correct side of the decision boundary. However for $\xi^n > 1$, the datapoint is assigned the opposite class to its training label. Ideally we want to limit the size of these 'violations' ξ^n. Here we briefly describe two standard approaches.

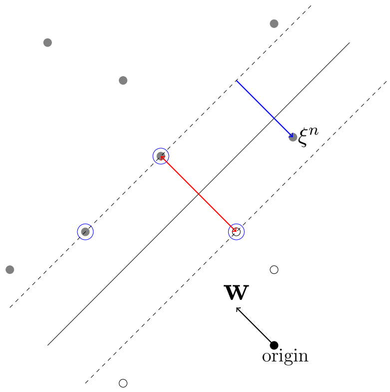


Figure 17.16: Slack Margin. The term ξ^n measures how far a variable is from the wrong side of the margin for its class. If $\xi^n > 1$ then the point will be misclassified – treated as an outlier.

2-Norm soft-margin

The 2-norm soft-margin objective is

$$\text{minimise } \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} + \frac{C}{2}\sum_n (\xi^n)^2 \qquad \text{subject to} \quad y^n\left(\mathbf{w}^\mathsf{T}\mathbf{x}^n + b\right) \geq 1 - \xi^n, \quad n = 1, \ldots, N \tag{17.6.11}$$

where C controls the number of mislabellings of the training data. The constant C needs to be determined empirically using a validation set. The optimisation problem expressed by (17.6.11) can be formulated using the Lagrangian

$$L(\mathbf{w}, b, \xi, \alpha) = \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} + \frac{C}{2}\sum_n (\xi^n)^2 - \sum_n \alpha^n \left[y^n\left(\mathbf{w}^\mathsf{T}\mathbf{x}^n + b\right) - 1 + \xi^n\right], \qquad \alpha^n \geq 0, \xi^n \geq 0 \tag{17.6.12}$$

which is to be minimised with respect to $\mathbf{x}, b, \xi$ and maximised with respect to α.

For points $\mathbf{x}^n$ on the 'correct' side of the decision boundary $y^n\left(\mathbf{w}^\mathsf{T}\mathbf{x}^n + b\right) - 1 + \xi^n > 0$ so that maximising L with respect to α requires the corresponding α^n to be set to zero. Only training points that are *support vectors* lying on the decision boundary will have non-zero α^n.

Differentiating the Lagrangian and equating to zero, we have the conditions

$$\frac{\partial}{\partial w_i} L(\mathbf{w}, b, \xi, \alpha) = w_i - \sum_n \alpha^n y^n x_i^n = 0 \tag{17.6.13}$$

$$\frac{\partial}{\partial b} L(\mathbf{w}, b, \xi, \alpha) = -\sum_n \alpha^n y^n = 0 \tag{17.6.14}$$

$$\frac{\partial}{\partial \xi^n} L(\mathbf{w}, b, \xi, \alpha) = C\xi^n - \alpha^n = 0 \tag{17.6.15}$$

From this we see that the solution for $\mathbf{w}$ is given by

$$\mathbf{w} = \sum_n \alpha^n y^n \mathbf{x}^n \tag{17.6.16}$$

Since only the support vectors have non-zero α^n, the solution for $\mathbf{w}$ will typically depend on only a small number of the training data. Using these conditions and substituting back into the original problem, the objective is equivalent to minimising

$$L(\alpha) = \sum_n \alpha^n - \frac{1}{2}\sum_{n,m} y^n y^m \alpha^n \alpha^m \left(\mathbf{x}^n\right)^\mathsf{T}\mathbf{x}^m - \frac{1}{2C}\sum_n (\alpha^n)^2$$

$$\text{subject to} \qquad \sum_n y^n \alpha^n = 0, \qquad \alpha^n \geq 0 \tag{17.6.17}$$

If we define

$$K(\mathbf{x}^n, \mathbf{x}^m) = \left(\mathbf{x}^n\right)^\mathsf{T}\mathbf{x}^m \tag{17.6.18}$$

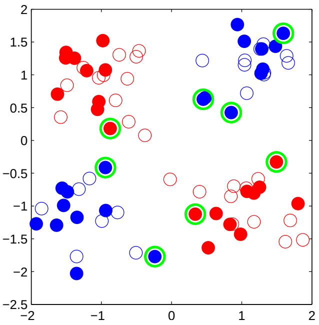


Figure 17.17: SVM training. The solid red and solid blue circles represent training data from different classes. The support vectors are highlighted in green. For the unfilled test points, the class assigned to them by the SVM is given by the colour. See `demoSVM.m`

The optimisation problem is

$$\begin{aligned} &\text{maximize} && \sum_n \alpha^n - \frac{1}{2}\sum_{n,m} y^n y^m \alpha^n \alpha^m \left(K(\mathbf{x}^n, \mathbf{x}^m) + \frac{1}{C}\delta_{n,m}\right) \\ &\text{subject to} && \sum_n y^n \alpha^n = 0, \qquad \alpha^n \geq 0 \end{aligned} \tag{17.6.19}$$

Optimising this objective is discussed in section(17.6.3).

1-norm soft-margin (box constraint)

In the 1-norm soft-margin version, one uses a 1-norm penalty

$$C\sum_n \xi^n \tag{17.6.20}$$

to give the optimisation problem:

$$\text{minimise } \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} + C\sum_n \xi^n \qquad \text{subject to } y^n\left(\mathbf{w}^\mathsf{T}\mathbf{x}^n + b\right) \geq 1 - \xi^n, \xi^n \geq 0, \qquad n = 1, \ldots, N \tag{17.6.21}$$

where C is an empirically determined penalty factor that controls the number of mislabellings of the training data. To reformulate the optimisation problem we use the Lagrangian

$$L(\mathbf{w}, b, \xi) = \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} + C\sum_n \xi^n - \sum_n \alpha^n \left[y^n\left(\mathbf{w}^\mathsf{T}\mathbf{x}^n + b\right) - 1 + \xi^n\right] - \sum_n r^n \xi^n, \qquad \alpha^n \geq 0, \xi^n \geq 0, r^n \geq 0 \tag{17.6.22}$$

The variables r^n are introduced in order to give a non-trivial solution (otherwise $\alpha^n = C$). Following a similar argument as for the 2-norm case, by differentiating the Lagrangian and equating to zero, we arrive at the optimisation problem

$$\begin{aligned} &\text{maximize} && \sum_n \alpha^n - \frac{1}{2}\sum_{n,m} y^n y^m \alpha^n \alpha^m K(\mathbf{x}^n, \mathbf{x}^m) \\ &\text{subject to} && \sum_n y^n \alpha^n = 0, \qquad 0 \leq \alpha^n \leq C \end{aligned} \tag{17.6.23}$$

which is closely related to the 2-norm problem except that we now have the box-constraint $0 \leq \alpha^n \leq C$.

17.6.2 Using Kernels

The final objectives (17.6.19) and (17.6.17) depend on the inputs $\mathbf{x}^n$ only via the scalar product $(\mathbf{x}^n)^\mathsf{T}\mathbf{x}^n$. If we map $\mathbf{x}$ to a vector function of $\mathbf{x}$, then we can write

$$K(\mathbf{x}^n, \mathbf{x}^m) = \boldsymbol{\phi}(\mathbf{x}^n)^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}^m) \tag{17.6.24}$$

This means that we can use any positive (semi) definite kernel K and make a non-linear classifier. See section(19.3).

17.6.3 Performing the optimisation

Both of the above soft-margin SVM optimisations problems (17.6.19) and (17.6.17) are quadratic programs for which the exact computational cost scales as $O\left(N^3\right)$. Whilst these can be solved with general purpose routines, specifically tailored routines that exploit the structure of the problem are preferred in practice. Of particular practical interest are 'chunking' techniques that optimise over a subset of the α. In the limit of updating only two components of α, this can be achieved analytically, resulting in the Sequential Minimal Optimisation algorithm[221], whose practical performance is typically $O\left(N^2\right)$ or better. A variant of this algorithm [90] is provided in `SVMtrain.m`.

Once the optimal solution α_* is found the decision function for a new point $\mathbf{x}$ is

$$\sum_n \alpha_*^n y^n K(\mathbf{x}^n, \mathbf{x}) + b_* \begin{cases} > 0 & \text{assign to class 1} \\ < 0 & \text{assign to class -1} \end{cases} \tag{17.6.25}$$

The optimal b_* is determined using the maximum margin condition, equations(17.6.5,17.6.6):

$$b_* = \frac{1}{2}\left[\min_{y^n=1} \mathbf{w}_*^{\mathsf{T}}\mathbf{x}^n - \max_{y^n=-1} \mathbf{w}_*^{\mathsf{T}}\mathbf{x}^n\right] \tag{17.6.26}$$

17.6.4 Probabilistic interpretation

Kernelised logistic-regression has some of the characteristics of the SVM but does not express the large margin requirement. Also the sparse data usage of the SVM is similar to that of the Relevance Vector Machine we discuss in section(18.2.3). However, a probabilistic model whose MAP assignment matches exactly the SVM is hampered by the normalisation requirement for a probability distribution. Whilst, arguably, no fully satisfactory direct match between the SVM and a related probabilistic model has been achieved, approximate matches have been obtained[252].

17.7 Soft zero-one loss for Outlier robustness

Both the support vector machine and logistic regression are potentially mislead by outliers. For the SVM, a mislabelled datapoint that is far from the correct side of the decision boundary would require a large slack ξ. However, since exactly such large ξ are discouraged, it is unlikely that the SVM would admit such a solution. For logistic regression, the probability of generating a mislabelled point far from the correct side of the decision boundary is so exponentially small that this will never happen in practice. This means that the model trained with Maximum Likelihood will never present such a solution. In both cases therefore mislabelled points (or outliers) have a significant impact on the location of the decision boundary.

A robust technique to deal with outliers is to use the zero-one loss in which a mislabeled point contributes only a relatively small loss. Soft variants of this are obtained by using the objective

$$\sum_{n=1}^{N}\left[\sigma_\beta(b + \mathbf{w}^{\mathsf{T}}x^n) - c^n\right]^2 + \lambda \mathbf{w}^{\mathsf{T}}\mathbf{w} \tag{17.7.1}$$

which is to be minimised with respect to $\mathbf{w}$ and b. For $\beta \to \infty$ the first term above tends to the zero-one loss. The second term represents a penalty on the length of $\mathbf{w}$ and prevents overfitting. Kernel extensions of this soft zero-one loss are straightforward.

Unfortunately, the objective (17.7.1) is highly non-convex and finding the optimal $\mathbf{w}, b$ is computationally difficult. A simple-minded scheme is to fix all components of $\mathbf{w}$ except one, w_i and then perform a numerical one-dimensional optimisation over this single parameter w_i. At the next step, another parameter w_j is chosen, and the procedure repeated until convergence. As usual, λ can be set using validation. The practical difficulties of minimising non-convex high-dimensional objective functions means that these approaches are rarely used in practice. A discussion of practical attempts in this area is given in [283].

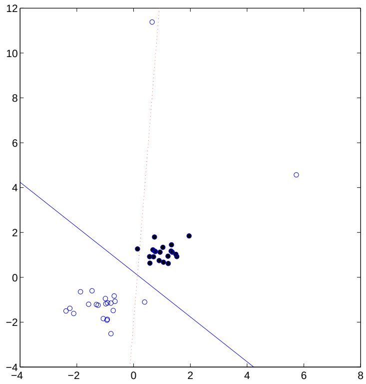


Figure 17.18: Soft zero-one loss decision boundary (solid line) versus logistic regression (dotted line). The number of mis-classified training points using the soft zero-one loss is 2, compared to 3 for logistic regression. The penalty $\lambda = 0.01$ was used for the soft-loss, with $\beta = 10$. For logistic regression, no penalty term was used. The outliers have a significant impact on the decision boundary for logistic regression, whilst the soft zero-one loss essentially gives up on the outliers and fits a large margin classifier between the remaining points. See `demoSoftLoss.m`.

An illustration of the difference between logistic regression and this soft zero-one loss is given in fig(17.18), which demonstrates how logistic regression is influenced by the mass of the data points, whereas the zero-one loss attempts to mimimise the number of mis-classifications whilst maintaining a large margin.

17.8 Notes

The perceptron has a long history in artificial intelligence and machine learning. Rosenblatt discussed the perceptron as a model for human learning, arguing that its distributive nature (the input-output 'patterns' are stored in the weight vector) is closely related to the kind of information storage believed to be present in biological systems[230]. To deal with non-linear decision boundaries, the main thrust of research in the ensuing neural network community was on the use of multilayered structures in which the outputs of perceptrons are used as the inputs to other perceptrons, resulting in potentially highly non-linear discriminant functions. This line of research was largely inspired by analogies to biological information processing in which layered structures are prevalent. Such multilayered artificial neural networks are fascinating and, once trained, are extremely fast in forming their decisions. However, reliably training these systems is a highly complex task and probabilistic generalisations in which priors are placed on the parameters lead to computational difficulties. Whilst perhaps less inspiring from a biological viewpoint, the alternative route of using the kernel trick to boost the power of a linear classifier has the advantage of ease of training and generalisation to probabilistic variants. More recently, however, there has been a resurgence of interest in the multilayer systems, with new heuristics aimed at improving the difficulties in training, see for example [134].

17.9 Code

`demoCubicPoly.m`: Demo of Fitting a Cubic Polynomial
`demoLogReg.m`: Demo Logistic Regression
`LogReg.m`: Logistic Regression Gradient Ascent Training
`demoLogRegNonLinear.m`: Demo of Logistic regression with a non-linear $\phi(x)$
`SVMtrain.m`: SVM training using the SMO algorithm
`demoSVM.m`: SVM demo
`demoSoftLoss.m`: softloss demo
`softloss.m`: softloss function

17.10 Exercises

Exercise 173.

1. *Give an example of a two-dimensional dataset for which the data are linearly separable, but not linearly independent.*
2. *Can you find a dataset which is linearly independent but not linearly separable?*

Exercise 174. *Show that for both Ordinary and Orthogonal Least Squares regression fits to data* $\{(x^n, y^n), n = 1,$ *the fitted lines go through the point* $\sum_{n=1}^{N}(x^n, y^n)/N$.

Exercise 175. *Consider the softmax function for classifying an input vector* $\mathbf{x}$ *into one of* $c = 1, \ldots, C$ *classes using*

$$p(c|\mathbf{x}) = \frac{e^{\mathbf{w}_c^T \mathbf{x}}}{\sum_{c'=1}^{C} e^{\mathbf{w}_{c'}^T \mathbf{x}}} \tag{17.10.1}$$

A set of input-class examples is given by $\mathcal{D} = \{(\mathbf{x}^n, c^n), n = 1, \ldots, N\}$.

1. *Write down the log-likelihood* L *of the classes conditional on the inputs, assuming that the data is i.i.d.*

2. *Compute the Hessian with elements*

$$H_{ij} = \frac{\partial^2 L(\mathcal{D})}{\partial w_i w_j} \tag{17.10.2}$$

where $\mathbf{w}$ *is is the stacked vector*

$$\mathbf{w} = \left(\mathbf{w}_1^T, \ldots, \mathbf{w}_C^T\right)^T \tag{17.10.3}$$

and show that the Hessian is positive semi-definite, that is $\mathbf{z}^T \mathbf{H} \mathbf{z} \geq 0$ *for any* $\mathbf{z}$.

Exercise 176. *Derive from equation (17.6.11) the dual optimisation problem equation (17.6.17).*

Exercise 177. *A datapoint* $\mathbf{x}$ *is projected to a lower dimensional vector* $\acute{\mathbf{x}}$ *using*

$$\acute{\mathbf{x}} = \mathbf{M}\mathbf{x} \tag{17.10.4}$$

where $\mathbf{M}$ *is a fat matrix. For a set of data* $\mathbf{x}^n, n = 1, \ldots, N$ *and corresponding binary class labels* $y^n \in \{0, 1\}$, *using logistic regression on the projected datapoints* $\acute{\mathbf{x}}^n$ *corresponds to a form of constrained logistic regression in the original higher dimensional space* $\mathbf{x}$. *Explain if it is reasonable to use an algorithm such as PCA to first reduce the data dimensionality before using logistic regression.*

Exercise 178. *The logistic sigmoid function is defined as* $\sigma(x) = e^x/(1+e^x)$. *What is the inverse function,* $\sigma^{-1}(x)$?

Exercise 179. *Given a dataset* $\mathcal{D} = \{(\mathbf{x}^n, c^n), n = 1, \ldots, N\}$, *where* $c^n \in \{0, 1\}$, *logistic regression uses the model* $p(c = 1|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + b)$. *Assuming the data is drawn independently and identically, show that the derivative of the log likelihood* L *with respect to* $\mathbf{w}$ *is*

$$\nabla_{\mathbf{w}} L = \sum_{n=1}^{N} \left(c^n - \sigma\left(\mathbf{w}^T \mathbf{x}^n + b\right)\right) \mathbf{x}^n \tag{17.10.5}$$

Exercise 180. *Consider a dataset* $\mathcal{D} = \{(\mathbf{x}^n, c^n), n = 1, \ldots, N\}$, *where* $c^n \in \{0, 1\}$, *and* $\mathbf{x}$ *is a* D *dimensional vector.*

1. *Show that if the training data is linearly separable with the hyperplane* $\mathbf{w}^T\mathbf{x} + b$, *the data is also separable with the hyperplane* $\tilde{\mathbf{w}}^T\mathbf{x} + \tilde{b}$, *where* $\tilde{\mathbf{w}} = \lambda\mathbf{w}$, $\tilde{b} = \lambda b$ *for any scalar* $\lambda > 0$.

2. *What consequence does the above result have for maximum likelihood training of linearly separable data?*

Exercise 181. *Consider a dataset* $\mathcal{D} = \{(\mathbf{x}^n, c^n), n = 1, \ldots, N\}$, *where* $c^n \in \{0, 1\}$, *and* $\mathbf{x}$ *is a* N *dimensional vector. (Hence we have* N *datapoints in a* N *dimensional space). In the text we showed that we can find a hyperplane (parameterised by* $(\mathbf{w}, b)$*) that linearly separates this data we need, for each datapoint* $\mathbf{x}^n$, $\mathbf{w}^T\mathbf{x}^n + b = \epsilon^n$ *where* $\epsilon^n > 0$ *for* $c^n = 1$ *and* $\epsilon^n < 0$ *for* $c^n = 0$. *Comment on the relation between maximum likelihood training and the algorithm suggested above.*

Exercise 182. *Given training data* $\mathcal{D} = \{(x^n, y^n), n = 1, \ldots, N\}$, *you decide to fit a regression model* $y = mx + c$ *to this data. Derive an expression for* m *and* c *in terms of* $\mathcal{D}$ *using the minimum sum squared error criterion.*

Exercise 183. *Given training data* $\mathcal{D} = \{(\mathbf{x}^n, c^n), n = 1, \ldots, N\}$, $c^n \in \{0, 1\}$, *where* $\mathbf{x}$ *are vector inputs, a discriminative model is*

$$p(c = 1|\mathbf{x}) = \sigma(b_0 + v_1 g(\mathbf{w}_1^{\mathsf{T}}\mathbf{x} + b_1) + v_2 g(\mathbf{w}_2^{\mathsf{T}}\mathbf{x} + b_2)) \tag{17.10.6}$$

where $g(x) = \exp(-0.5x^2)$. *and* $\sigma(x) = e^x/(1 + e^x)$ *(this is a* neural network *with a single hidden layer and two hidden units).*

1. *Write down the log likelihood for the class conditioned on the inputs, based on the usual i.i.d. assumption.*

2. *Calculate the derivatives of the log likelihood as a function of the network parameters,* $\mathbf{w}_1, \mathbf{w}_2, b_1, b_2, \mathbf{v}, b0$

3. *Comment on the relationship between this model and logistic regression.*

4. *Comment on the decision boundary of this model.*

CHAPTER 18

Bayesian Linear Models

18.1 Regression with additive Gaussian noise

The models in chapter(17) were trained under Maximum Likelihood, and do not deal with the issue that, from a probabilistic perspective, parameter estimates are inherently uncertain due to the limited available training data. Here we treat this case by using a probabilistic extension of the 'primal' parameter representation. The 'dual' representation (which leads to kernel treatments) is most appropriately dealt with under the class of Gaussian Process models, and is deferred to chapter(19).

Regression refers to inferring a mapping on the basis of observed data $\mathcal{D} = \{(\mathbf{x}^n, y^n), n = 1, \ldots N\}$, where $(\mathbf{x}^n, y^n)$ represents an input-output pair. Most of our discussion will relate only to scalar outputs (and vector inputs $\mathbf{x}$). The extension to the vector output case $\mathbf{y}$ is straightforward. We assume that each (clean) output is generated from a model $f(\mathbf{x}; \mathbf{w})$ where the parameters $\mathbf{w}$ of the function f are unknown. An observed output y^n is generated by the addition of noise η to the clean model output,

$$y(\mathbf{x}) = f(\mathbf{x}; \mathbf{w}) + \eta \tag{18.1.1}$$

If the noise is Gaussian distributed, $\eta \sim \mathcal{N}(\eta | 0, \sigma^2)$, the model generates an output y for input $\mathbf{x}$ with probability

$$p(y|\mathbf{w}, \mathbf{x}) = \mathcal{N}(y | f(\mathbf{x}; \mathbf{w}), \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}[y - f(\mathbf{x};\mathbf{w})]^2} \tag{18.1.2}$$

If we assume that each data input-output pair is generated identically and independently, the likelihood the model generates the data $\mathcal{D} = \{\mathcal{D}_x, \mathcal{D}_y\}$ is

$$p(\mathcal{D}|\mathbf{w}) = \prod_{n=1}^{N} p(y^n|\mathbf{w}, \mathbf{x}^n) p(\mathbf{x}^n) \tag{18.1.3}$$

We may use a prior weight distribution $p(\mathbf{w})$ to quantify our a priori belief in the suitability each parameter setting. The posterior weight distribution is then given by

$$p(\mathbf{w}|\mathcal{D}) \propto p(\mathcal{D}|\mathbf{w}) p(\mathbf{w}) \propto p(\mathcal{D}_y|\mathbf{w}, \mathcal{D}_x) p(\mathbf{w}) \tag{18.1.4}$$

Using the Gaussian noise assumption, and for convenience defining $\beta = 1/\sigma^2$, this gives

$$\log p(\mathbf{w}|\mathcal{D}) = -\frac{\beta}{2} \sum_{n=1}^{N} [y^n - f(\mathbf{x}^n; \mathbf{w})]^2 + \log p(\mathbf{w}) + \frac{N}{2} \log \beta + \text{const.} \tag{18.1.5}$$

Note the similarity between equation (18.1.5) and the regularised training error equation (17.2.16). In the probabilistic framework, we identify the choice of a sum square error with the assumption of additive Gaussian noise. Similarly, the regularising term is identified with $\log p(\mathbf{w})$.

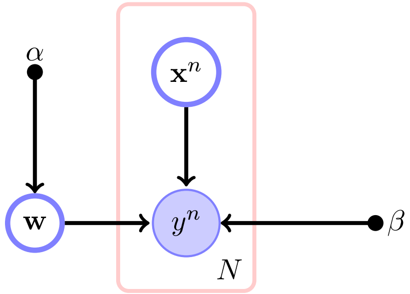


Figure 18.1: Belief Network representation of a Bayesian Model for regression under the i.i.d. data assumption. The hyperparameter α acts as a form of regulariser, controlling the flexibility of the prior on the weights $\mathbf{w}$. The hyperparameter β controls the level of noise on the observations.

18.1.1 Bayesian linear parameter models

Linear parameter models, as discussed in chapter(17) have the form

$$f(\mathbf{x};\mathbf{w}) = \sum_{i=1}^{B} w_i \phi_i(\mathbf{x}) \equiv \mathbf{w}^{\mathsf{T}} \boldsymbol{\phi}(\mathbf{x}) \tag{18.1.6}$$

where the parameters w_i are also called 'weights' and $\dim w = B$. Such models have a linear parameter dependence, but may represent a non-linear input-output mapping if the basis functions $\phi_i(\mathbf{x})$ are non-linear in $\mathbf{x}$.

Since the output is linearly dependent on $\mathbf{w}$, we can discourage extreme output values by penalising large weight values. A natural weight prior is thus

$$p(\mathbf{w}|\alpha) = \mathcal{N}\left(\mathbf{w}\middle|\mathbf{0}, \alpha^{-1}\mathbf{I}\right) = \left(\frac{\alpha}{2\pi}\right)^{\frac{B}{2}} e^{-\frac{\alpha}{2}\mathbf{w}^{\mathsf{T}}\mathbf{w}} \tag{18.1.7}$$

where the *precision* α is the inverse variance. If α is large, the total squared length of the weight vector $\mathbf{w}$ is encouraged to be small. Under the Gaussian noise assumption, the posterior distribution is

$$\log p(\mathbf{w}|\Gamma, \mathcal{D}) = -\frac{\beta}{2}\sum_{n=1}^{N}\left[y^n - \mathbf{w}^{\mathsf{T}}\boldsymbol{\phi}(\mathbf{x}^n)\right]^2 - \frac{\alpha}{2}\mathbf{w}^{\mathsf{T}}\mathbf{w} + \text{const.} \tag{18.1.8}$$

where $\Gamma = \{\alpha, \beta, \ldots\}$ represents the hyperparameter set. Parameters that determine the functions ϕ may also be included in the hyperparameter set.

Using the LPM in equation (18.1.5) with a Gaussian prior, equation (18.1.7), the weight posterior is Gaussian,

$$p(\mathbf{w}|\Gamma, \mathcal{D}) = \mathcal{N}\left(\mathbf{w}|\bar{\mathbf{w}}, \mathbf{S}_w\right) \tag{18.1.9}$$

where the covariance and mean are given by

$$\mathbf{S}_w = \left(\alpha\mathbf{I} + \beta\sum_{n=1}^{N}\boldsymbol{\phi}\left(\mathbf{x}^n\right)\boldsymbol{\phi}\left(\mathbf{x}^n\right)^{\mathsf{T}}\right)^{-1}, \qquad \bar{\mathbf{w}} = \beta\mathbf{S}_w\sum_{n=1}^{N} y^n \boldsymbol{\phi}\left(\mathbf{x}^n\right) \tag{18.1.10}$$

This result follows from completing the square, section(8.6.2) in equation (18.1.8). The mean prediction for an input $\mathbf{x}$ is the given by

$$\bar{f}(\mathbf{x}) \equiv \int f(\mathbf{x};\mathbf{w}) p(\mathbf{w}|\mathcal{D}, \Gamma) d\mathbf{w} = \bar{\mathbf{w}}^{\mathsf{T}}\boldsymbol{\phi}\left(\mathbf{x}\right). \tag{18.1.11}$$

Similarly, the variance of the underlying estimated clean function is

$$\text{var}(f(\mathbf{x})) = \left\langle\left[\mathbf{w}^{\mathsf{T}}\boldsymbol{\phi}(\mathbf{x})\right]^2\right\rangle - \bar{f}(\mathbf{x})^2 = \boldsymbol{\phi}^{\mathsf{T}}(\mathbf{x})\mathbf{S}_w\boldsymbol{\phi}(\mathbf{x}) \tag{18.1.12}$$

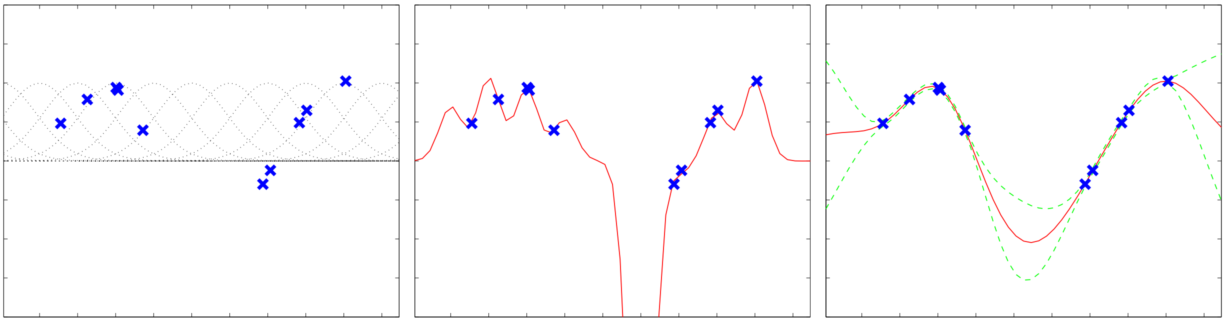


Figure 18.2: Along the horizontal axis we plot the input x and along the vertical axis the output t. **(a)**: The raw input-output training data. **(b)**: Prediction using regularised training and fixed hyperparameters. **(c)**: Prediction using ML-II optimised hyperparameters. Also plotted are standard error bars on the clean underlying function, $\sqrt{\text{var}(f(x))}$.

Under the Gaussian assumptions, the output variance $\text{var}(f(\mathbf{x}))$ depends only on the input variables and not on the training outputs y.Since the additive noise η is uncorrelated with the model outputs, the *predictive variance* is

$$\text{var}(y(\mathbf{x})) = \text{var}(f(\mathbf{x})) + \sigma^2 \tag{18.1.13}$$

and represents the variance of the 'noisy' output for an input $\mathbf{x}$.

Example 81. In fig(18.2b), we show the mean prediction on the data in fig(18.2a) using 15 Gaussian basis functions with width $\lambda = 0.03^2$ spread out evenly over the 1-dimensional input space. We set the other hyperparameters by hand to $\beta = 100$ and $\alpha = 1$. The prediction severely overfits the data, a result of a poor choice of hyperparameter settings.

18.1.2 Determining hyperparameters: ML-II

The hyperparameter posterior distribution is

$$p(\Gamma|\mathcal{D}) \propto p(\mathcal{D}|\Gamma)p(\Gamma) \tag{18.1.14}$$

A simple summarisation of the hyperparameter posterior is given by the MAP assignment

$$\Gamma^* = \underset{\Gamma}{\text{argmax}}\, p(\Gamma|\mathcal{D}) \tag{18.1.15}$$

If the prior belief about the hyperparameters is weak ($p(\Gamma) \approx \text{const.}$), we can estimate the optimal hyperparameters by optimising the *marginal likelihood*

$$p(\mathcal{D}|\Gamma) = \int p(\mathcal{D}|\Gamma, \mathbf{w})p(\mathbf{w}|\Gamma)d\mathbf{w} \tag{18.1.16}$$

This approach to setting hyperparameters is called 'ML-II' [33] or the *Evidence Procedure*[178].

In the case of Bayesian Linear Parameter models under Gaussian additive noise computing the marginal likelihood involves only Gaussian integration. A naive approach to deriving an expression for the marginal likelihood is to use

$$p(\mathcal{D}|\Gamma, \mathbf{w})p(\mathbf{w}) = \exp\left(-\frac{\beta}{2}\left[y^n - \mathbf{w}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}^n)\right]^2 - \frac{\alpha}{2}\mathbf{w}^\mathsf{T}\mathbf{w}\right)(2\pi\beta)^{N/2}(2\pi\alpha)^{B/2} \tag{18.1.17}$$

By collating terms in $\mathbf{w}$ (completing the square, section(8.6.2)), the above represents a Gaussian in $\mathbf{w}$ with additional factors. After integrating over this Gaussian we have

$$2\log p(\mathcal{D}|\Gamma) = -\beta\sum_{n=1}^{N}(y^n)^2 + \mathbf{d}^\mathsf{T}\mathbf{S}_w^{-1}\mathbf{d} - \log\det(\mathbf{S}_w) + B\log\alpha + N\log\beta - N\log(2\pi) \tag{18.1.18}$$

where

$$\mathbf{d} = \beta\sum_n \boldsymbol{\phi}(\mathbf{x}^n)y^n \tag{18.1.19}$$

See exercise(185) for an alternative expression.

18.1.3 Prediction

How would the mean predictor be calculated if we were to include the hyperparameters Γ as part of a hierarchical model? Formally, this becomes

$$\bar{f}(\mathbf{x}) = \int f(\mathbf{x};\mathbf{w})p(\mathbf{w},\Gamma|\mathcal{D})d\mathbf{w}d\Gamma = \int\left\{\int f(\mathbf{x};\mathbf{w})p(\mathbf{w}|\Gamma,\mathcal{D})d\mathbf{w}\right\}p(\Gamma|\mathcal{D})d\Gamma \tag{18.1.20}$$

The term in curly brackets is the mean predictor for fixed hyperparameters. Equation(18.1.20) weights each mean predictor by the posterior probability of the hyperparameter $p(\Gamma|\mathcal{D})$. This is a general recipe for combining model predictions, where each model is weighted by its posterior probability. However, computing the integral over the hyperparameter posterior is numerically challenging and approximations are usually required. Provided the hyperparameters are well determined by the data, we may instead approximate the above hyperparameter integral by finding the MAP hyperparameters and use

$$\bar{f}(\mathbf{x}) \approx \int f(\mathbf{x};\mathbf{w})p(\mathbf{w}|\Gamma^*,\mathcal{D})d\mathbf{w} \tag{18.1.21}$$

Example 82. Using the hyperparameters α, β, λ that optimise expression (18.1.18) gives the results in fig(18.2c) where we plot both the mean predictions and standard predictive error bars. This demonstrates that an acceptable setting for the hyperparameters can be obtained by maximising the marginal likelihood.

18.1.4 Learning the hyperparameters using EM

We can set hyperparameters such as α and β by maximising the marginal likelihood equation (18.1.16). A convenient computational procedure to achieve this is to interpret the $\mathbf{w}$ as latent variables and apply the EM algorithm, section(11.2). In this case the energy term is

$$E \equiv \left\langle \log p(\mathcal{D}|\mathbf{w},\Gamma)p(\mathbf{w}|\Gamma)\right\rangle_{p(\mathbf{w}|\mathcal{D},\Gamma^{old})} \tag{18.1.22}$$

According to the general EM procedure we need to maximise the energy term. For a hyperparameter Γ we the derivative of the energy is given by

$$\frac{\partial}{\partial\Gamma}E \equiv \left\langle \frac{\partial}{\partial\Gamma}\log p(\mathcal{D}|\mathbf{w},\Gamma)p(\mathbf{w}|\Gamma)\right\rangle_{p(\mathbf{w}|\mathcal{D},\Gamma^{old})} \tag{18.1.23}$$

For the Bayesian LPM with Gaussian weight and noise distributions, we obtain

$$\frac{\partial}{\partial\beta}E = \frac{N}{2\beta} - \frac{1}{2}\sum_{n=1}^{N}\left\langle\left[y^n - \mathbf{w}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}^n)\right]^2\right\rangle_{p(\mathbf{w}|\Gamma^{old},\mathcal{D})} \tag{18.1.24}$$

$$= \frac{N}{2\beta} - \frac{1}{2}\sum_{n=1}^{N}\left[y^n - \bar{\mathbf{w}}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}^n)\right]^2 - \frac{1}{2}\text{trace}\left(\mathbf{S}_w\sum_{n=1}^{N}\boldsymbol{\phi}(\mathbf{x}^n)\boldsymbol{\phi}(\mathbf{x}^n)^\mathsf{T}\right) \tag{18.1.25}$$

Solving for the zero derivatives gives the M-step update

$$\frac{1}{\beta^{new}} = \frac{1}{N}\sum_{n=1}^{N}\left[y^n - \bar{\mathbf{w}}^{\mathsf{T}}\boldsymbol{\phi}(\mathbf{x}^n)\right]^2 + \text{trace}\left(\mathbf{S}_w\hat{\mathbf{S}}\right) \tag{18.1.26}$$

where $\hat{\mathbf{S}}$ is the empirical covariance of the basis-function vectors $\boldsymbol{\phi}(\mathbf{x}^n), n = 1, \ldots, N$.

Similarly, for α,

$$\frac{\partial}{\partial\alpha}E = \frac{B}{2\alpha} - \frac{1}{2}\left\langle \mathbf{w}^{\mathsf{T}}\mathbf{w}\right\rangle_{p(\mathbf{w}|\Gamma^{old},\mathcal{D})} = \frac{B}{2\alpha} - \frac{1}{2}\left(\text{trace}\left(\mathbf{S}_w\right) + \bar{\mathbf{w}}^{\mathsf{T}}\bar{\mathbf{w}}\right)$$

which gives the update

$$\frac{1}{\alpha^{new}} = \frac{1}{B}\left(\text{trace}\left(\mathbf{S}_w\right) + \bar{\mathbf{w}}^{\mathsf{T}}\bar{\mathbf{w}}\right) \tag{18.1.27}$$

where $\mathbf{S}_w$ and $\bar{\mathbf{w}}$ are given in equation (18.1.10). An alternative fixed point procedure that is often more rapidly convergent than EM is given in equation (18.2.21). Closed form updates for other hyperparameters, such as the width of the basis functions, are not available, and the corresponding energy term needs to be optimised numerically.

Example 83 (Learning the basis function widths)**.** In fig(18.3) we plot the training data for a regression problem using a Bayesian LPM. A set of 10 Radial Basis Functions are used,

$$\phi_i(x) = \exp\left(-0.5(x - m_i)^2/\lambda^2\right) \tag{18.1.28}$$

with m_i, $i = 1, \ldots, 10$ spread out evenly between -2 and 2. The hyperparameters α and β are learned by ML-II under EM updating. For a fixed width λ we then present the predictions, each time finding the optimal α and β for this width. The optimal joint α, β, λ hyperparameter setting is obtained as described in fig(18.4) which shows the marginal log likelihood for a range of widths.

Validation likelihood

The hyperparameters found by ML-II are those which are best at explaining the training data. In principle, this is different from those that are best for prediction and, in practice therefore, it is reasonable set hyperparameters also by validation techniques. One such method is to set hyperparameters by minimal prediction error on a validation set.

Another common technique is to set hyperparameters by their likelihood on a validation set $\{\mathcal{X}_{val}, \mathcal{Y}_{val}\} \equiv \{(\mathbf{x}^m_{val}, y^m_{val}), m = 1, \ldots, M\}$:

$$p(\mathcal{Y}_{val}|\Gamma, \mathcal{X}_{train}, \mathcal{Y}_{train}, \mathcal{X}_{val}) = \int_{\mathbf{w}} p(\mathcal{Y}_{val}|\mathbf{w}, \Gamma)p(\mathbf{w}|\Gamma, \mathcal{X}_{train}, \mathcal{Y}_{train}) \tag{18.1.29}$$

from which we obtain (see exercise(186))

$$\log p(\mathcal{Y}_{val}|\Gamma, \mathcal{D}_{train}, \mathcal{X}_{val}) = -\frac{1}{2}\log\det\left(2\pi\mathbf{C}_{val}\right) - \frac{1}{2}\left(\mathbf{y}_{val} - \boldsymbol{\Phi}_{val}\bar{w}\right)^{\mathsf{T}}\mathbf{C}^{-1}_{val}\left(\mathbf{y}_{val} - \boldsymbol{\Phi}_{val}\bar{w}\right) \tag{18.1.30}$$

where $\mathbf{y}_{val} = \left[y^1_{val}, \ldots, y^M_{val}\right]^{\mathsf{T}}$ and

$$\mathbf{C}_{val} \equiv \boldsymbol{\Phi}_{val}\mathbf{S}_w\boldsymbol{\Phi}^{\mathsf{T}}_{val} + \sigma^2\mathbf{I}_M \tag{18.1.31}$$

and the *design matrix*

$$\boldsymbol{\Phi}^{\mathsf{T}}_{val} = \left[\boldsymbol{\phi}(\mathbf{x}^1_{val}), \ldots, \boldsymbol{\phi}(\mathbf{x}^M_{val})\right] \tag{18.1.32}$$

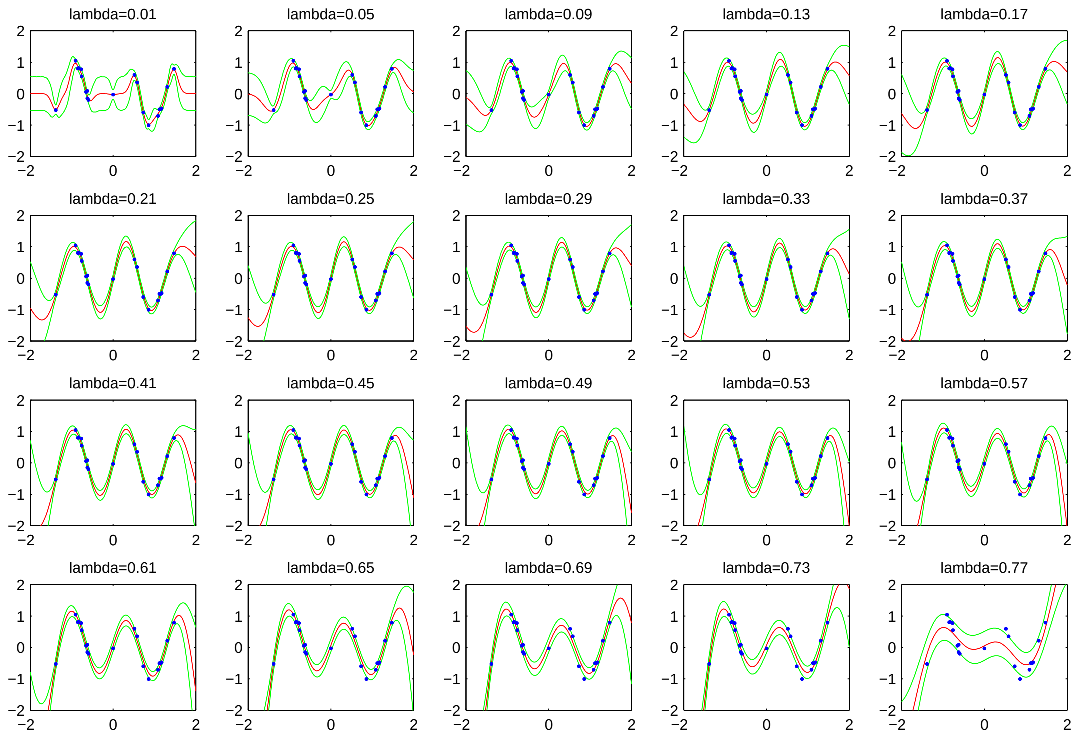


Figure 18.3: Predictions for an RBF for different widths λ. For each λ the optimal α, β are obtained by running the EM procedure to convergence and subsequently used to form the predictions. In each panel the dots represent the training points, with x along the horizontal axis and y along the vertical axis. Mean predictions are plotted, along with predictive error bars of one standard deviation. According to ML-II, the best model corresponds to $\lambda = 0.37$, see fig(18.4). The smaller values of λ overfit the data, giving rise to too 'rough' functions. The largest values of λ underfit, giving too 'smooth' functions. See `demoBayesLinReg.m`.

18.1.5 The relevance vector machine

The *Relevance Vector Machine* assumes that only a small number of components of the basis function vector are relevant in determining the solution for $\mathbf{w}$. If we consider the basic equation,

$$f(\mathbf{x}; \mathbf{w}) = \sum_{i=1}^{B} w_i \phi_i(\mathbf{x}) \equiv \mathbf{w}^\mathsf{T} \boldsymbol{\phi}(\mathbf{x}) \tag{18.1.33}$$

then it is often the case that some basis functions will be redundant in the sense that a linear combination of the other basis functions can reproduce the training outputs with insignificant loss in accuracy. To exploit this effect and seek a parsimonious solution we may use a more refined prior that encourages each w_i itself to be small:

$$p(\mathbf{w}|\boldsymbol{\alpha}) = \prod_i p(w_i|\alpha_i) \tag{18.1.34}$$

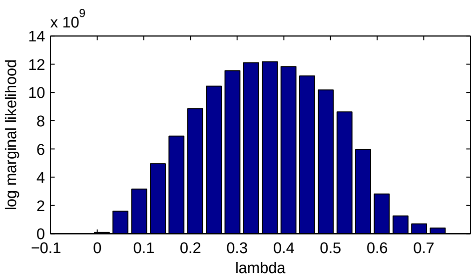


Figure 18.4: The log marginal likelihood $\log p(\mathcal{D}|\lambda, \alpha^*(\lambda), \beta^*(\lambda))$ having found the optimal values of the hyperparameters α and β using ML-II. These optimal values are dependent on λ. According to ML-II, the best model corresponds to $\lambda = 0.37$.

where the prior on each individual weight is given by

$$p(w_i|\alpha_i) = \mathcal{N}\left(w_i|0, \alpha_i^{-1}\right) = \left(\frac{\alpha_i}{2\pi}\right)^{\frac{1}{2}} e^{-\frac{\alpha_i}{2}w_i^2} \tag{18.1.35}$$

The modifications required to the description of section(18.1.1) are to replace $\mathbf{S}_w$ with

$$\mathbf{S}_w = \left(\text{diag}\left(\boldsymbol{\alpha}\right) + \beta \sum_{n=1}^{N} \boldsymbol{\phi}\left(\mathbf{x}^n\right) \boldsymbol{\phi}\left(\mathbf{x}^n\right)^{\mathsf{T}}\right)^{-1} \tag{18.1.36}$$

The marginal likelihood is then given by

$$2\log p(\mathcal{D}|\Gamma) = -\beta \sum_{n=1}^{N} (y^n)^2 + \mathbf{d}^{\mathsf{T}}\mathbf{S}_w^{-1}\mathbf{d} - \log\det\left(\mathbf{S}_w\right) + \sum_{i=1}^{B} \log\alpha_i + N\log\beta - N\log\left(2\pi\right) \tag{18.1.37}$$

The EM update for β is unchanged, and the EM update for each α_i is

$$\frac{1}{\alpha_i^{new}} = [\mathbf{S}_w]_{ii} + \bar{w}_i^2 \tag{18.1.38}$$

18.2 Classification

For the logistic regression model

$$p(c = 1|\mathbf{w}, \mathbf{x}) = \sigma\left(\sum_{i}^{B} w_i \phi_i(\mathbf{x})\right) \tag{18.2.1}$$

the Maximum Likelihood method returns only a single optimal $\mathbf{w}$. To deal with the inevitable uncertainty in estimating $\mathbf{w}$ we need to determine the posterior distribution of the weights $\mathbf{w}$. To do so we first define a prior on the weights $p(\mathbf{w})$ for which a convenient choice is a Gaussian:

$$p(\mathbf{w}|\alpha) = \mathcal{N}\left(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I}\right) = \frac{\alpha^{B/2}}{(2\pi)^{B/2}} e^{-\alpha\mathbf{w}^{\mathsf{T}}\mathbf{w}/2} \tag{18.2.2}$$

where α is the inverse variance (also called the precision). Given a dataset of input-class labels, $\mathcal{D} = \{(\mathbf{x}^n, c^n), n = 1, \ldots, N\}$, the parameter posterior is

$$p(\mathbf{w}|\alpha, \mathcal{D}) = \frac{p(\mathcal{D}|\mathbf{w}, \alpha)p(\mathbf{w}|\alpha)}{p(\mathcal{D}|\alpha)} = \frac{1}{p(\mathcal{D}|\alpha)} p(\mathbf{w}|\alpha) \prod_{n=1}^{N} p(c^n|\mathbf{x}^n, \mathbf{w}) \tag{18.2.3}$$

18.2.1 Laplace approximation

The weight posterior is given by

$$p(\mathbf{w}|\alpha, \mathcal{D}) \propto e^{-E(\mathbf{w})} \tag{18.2.4}$$

where

$$E(\mathbf{w}) = \frac{\alpha}{2}\mathbf{w}^{\mathsf{T}}\mathbf{w} - \sum_{n=1}^{N} \log\sigma\left(\mathbf{w}^{\mathsf{T}}\mathbf{h}^n\right), \qquad \mathbf{h}^n \equiv (2c^n - 1)\boldsymbol{\phi}^n \tag{18.2.5}$$

By approximating $E(\mathbf{w})$ by a quadratic function in $\mathbf{w}$, we obtain a Gaussian approximation $q(\mathbf{w}|\mathcal{D}, \alpha)$ to $p(\mathbf{w}|\mathcal{D}, \alpha)$. To do so we first find the minimum of $E(\mathbf{w})$. Differentiating, we obtain

$$\nabla E = \alpha\mathbf{w} - \sum_{n=1}^{N} (1 - \sigma^n)\mathbf{h}^n, \qquad \sigma^n \equiv \sigma\left(\mathbf{w}^{\mathsf{T}}\mathbf{h}^n\right) \tag{18.2.6}$$

It is convenient to use a Newton method to find the optimum. The Hessian matrix with elements

$$H_{ij} \equiv \frac{\partial^2}{\partial w_i \partial w_j} E(\mathbf{w}) \tag{18.2.7}$$

is given by

$$\mathbf{H} = \alpha \mathbf{I} + \underbrace{\sum_{n=1}^{N} \sigma^n (1 - \sigma^n) \boldsymbol{\phi}^n \left(\boldsymbol{\phi}^n\right)^\mathsf{T}}_{\mathbf{J}} \tag{18.2.8}$$

Note that the Hessian is positive semi-definite (see exercise(188)) so that the function $E(\mathbf{w})$ is convex (bowl shaped), and finding a minimum of $E(\mathbf{w})$ is numerically unproblematic. A Newton update then is

$$\mathbf{w}^{new} = \mathbf{w} - \mathbf{H}^{-1} \left(\nabla E\right) \tag{18.2.9}$$

The posterior approximation is given by

$$q(\mathbf{w}|\mathcal{D}, \alpha) = \mathcal{N}\left(\mathbf{w} | \mathbf{w}^*, \mathbf{S}_\mathbf{w}\right), \qquad \mathbf{S}_\mathbf{w} \equiv \mathbf{H}^{-1} \tag{18.2.10}$$

where $\mathbf{w}^*$ is the converged estimate of the minimum point of $E(\mathbf{w})$ and H is the Hessian of $E(\mathbf{w})$ at this point.

Approximating the marginal likelihood

To determine α using ML-II we need the marginal likelihood

$$p(\mathcal{D}|\alpha) = \int_\mathbf{w} p(\mathcal{D}|\mathbf{w}) p(\mathbf{w}|\alpha) = \int_\mathbf{w} \prod_{n=1}^{N} p(c^n|\mathbf{x}^n, \mathbf{w}) \left(\frac{\alpha}{2\pi}\right)^{B/2} e^{-\frac{\alpha}{2}\mathbf{w}^\mathsf{T}\mathbf{w}} \propto \int_\mathbf{w} e^{-E(\mathbf{w})} \tag{18.2.11}$$

For an optimum value $\mathbf{w}^*$, we approximate the marginal likelihood using (see section(28.2))

$$\log p(\mathcal{D}|\alpha) \approx L(\alpha) \equiv -\frac{\alpha}{2} (\mathbf{w}^*)^\mathsf{T} \mathbf{w}^* + \sum_n \log \sigma \left((\mathbf{w}^*)^\mathsf{T} \mathbf{h}^n \right) - \frac{1}{2} \log \det \left(\alpha \mathbf{I} + \mathbf{J}\right) + \frac{B}{2} \log \alpha \tag{18.2.12}$$

To find the optimal α, we search for the zero derivative of $\log p(\mathcal{D}|\alpha)$. There are two strategies we could adopt here : either (i) compute the derivative of $\log p(\mathcal{D}|\alpha)$ analytically and approximate the resulting expression or (ii) differentiate the approximate marginal likelihood $L(\alpha)$ directly. In fact the two procedures result in the same updates. We'll adopt approach (i) since this is arguably more intellectually appealing. See exercise(190) for a discussion of the alternative procedure.

From equation (18.2.11) we can use the general derivative identity to arrive at

$$\frac{\partial}{\partial \alpha} \log p(\mathcal{D}|\alpha) = \left\langle \frac{\partial}{\partial \alpha} \log p(\mathbf{w}|\alpha) \right\rangle_{p(\mathbf{w}|\alpha, \mathcal{D})} \tag{18.2.13}$$

Since

$$\log p(\mathbf{w}|\alpha) = -\frac{\alpha}{2} \mathbf{w}^\mathsf{T} \mathbf{w} + \frac{B}{2} \log \alpha + \mathsf{const.} \tag{18.2.14}$$

we obtain the following exact expression for the derivative of the marginal likelihood

$$\frac{\partial}{\partial \alpha} \log p(\mathcal{D}|\alpha) = \frac{1}{2} \left\langle -\mathbf{w}^\mathsf{T} \mathbf{w} + \frac{B}{\alpha} \right\rangle_{p(\mathbf{w}|\alpha, \mathcal{D})} \tag{18.2.15}$$

Setting the derivative to zero, an exact equation for the optimal α is

$$0 = -\left\langle \mathbf{w}^\mathsf{T} \mathbf{w} \right\rangle_{p(\mathbf{w}|\alpha, \mathcal{D})} + \frac{B}{\alpha} \tag{18.2.16}$$

Algorithm 18 Evidence Procedure for Bayesian Logistic Regression

Initialise $\mathbf{w}$ and α.
while Not Converged **do**
    Find optimal $\mathbf{w}^*$ by iterating equation (18.2.9), equation (18.2.8) to convergence. ▷ E-step
    Update α according to equation (18.2.19). ▷ M-Step
end while

One may now form a fixed point equation

$$\alpha^{new} = \frac{B}{\langle \mathbf{w}^\mathsf{T}\mathbf{w}\rangle_{p(\mathbf{w}|\alpha,\mathcal{D})}} \tag{18.2.17}$$

which is in fact a re-derivation of the EM procedure for this model. The averages in the above expression cannot be computed exactly and are replaced by averages with respect to the Laplace approximation of the posterior $q(\mathbf{w}|\alpha, \mathcal{D})$. Note that since we only have an approximation to the posterior, and therefore the mean and covariance statistics, we cannot guarantee that the likelihood will always increase.

To explicitly write the above update in terms of the Laplace approximation we use

$$\left\langle \mathbf{w}^\mathsf{T}\mathbf{w}\right\rangle = \text{trace}\left(\left\langle \mathbf{w}\mathbf{w}^\mathsf{T}\right\rangle - \langle\mathbf{w}\rangle\langle\mathbf{w}\rangle^\mathsf{T} + \langle\mathbf{w}\rangle\langle\mathbf{w}\rangle^\mathsf{T}\right) = \text{trace}\left(\mathbf{S}_\mathbf{w}\right) + \langle\mathbf{w}\rangle^\mathsf{T}\langle\mathbf{w}\rangle \tag{18.2.18}$$

$$\alpha^{new} = \frac{B}{\text{trace}\left(\mathbf{S}_\mathbf{w}\right) + \left(\mathbf{w}^*\right)^\mathsf{T}\mathbf{w}^*} \tag{18.2.19}$$

Using the fact that for a Gaussian the mean is the same as the mode, $\langle\mathbf{w}\rangle = \mathbf{w}^*$.

Gull-MacKay fixed point iteration

From equation (18.2.16) we have

$$0 = -\alpha\left\langle \mathbf{w}^\mathsf{T}\mathbf{w}\right\rangle_{p(\mathbf{w}|\alpha,\mathcal{D})} + B = -\alpha\mathbf{S}_\mathbf{w} - \alpha\left(\mathbf{w}^*\right)^\mathsf{T}\mathbf{w}^* + B \tag{18.2.20}$$

so that an alternative fixed point equation[121, 177] is

$$\alpha^{new} = \frac{B - \alpha\mathbf{S}_\mathbf{w}}{\left(\mathbf{w}^*\right)^\mathsf{T}\mathbf{w}^*} \tag{18.2.21}$$

18.2.2 Making predictions

Ultimately, our interest is to classify in novel situations, averaging over posterior weight uncertainty,

$$p(c = 1|\mathbf{x}, \alpha, \mathcal{D}) = \int_\mathbf{w} p(c = 1|\mathbf{x}, \mathbf{w})p(\mathbf{w}|\alpha, \mathcal{D}) \tag{18.2.22}$$

The B dimensional integrals over $\mathbf{w}$ cannot be computed analytically and numerical approximation is required. In this particular case the relative benign nature of the posterior (the log posterior is concave, see below) suggests a simple Laplace approximation may suffice (see [141] for a variational approximation). To make a class prediction for a novel input $\mathbf{x}$, we use

$$p(c = 1|\mathbf{x}, \mathcal{D}, \alpha^*) = \int p(c = 1|\mathbf{x}, \mathbf{w})p(\mathbf{w}|\mathcal{D}, \alpha^*)d\mathbf{w} = \int \sigma\left(\mathbf{x}^\mathsf{T}\mathbf{w}\right)p(\mathbf{w}|\mathcal{D}, \alpha^*)d\mathbf{w}$$

To compute the predictions it would appear that we need to carry out an integral in B dimensions. However, since the term $\sigma\left(\mathbf{x}^\mathsf{T}\mathbf{w}\right)$ depends on $\mathbf{w}$ via the scalar product $\mathbf{x}^\mathsf{T}\mathbf{w}$, we only require the integral over the one-dimensional projection (see exercise(189))

$$h \equiv \mathbf{x}^\mathsf{T}\mathbf{w} \tag{18.2.23}$$

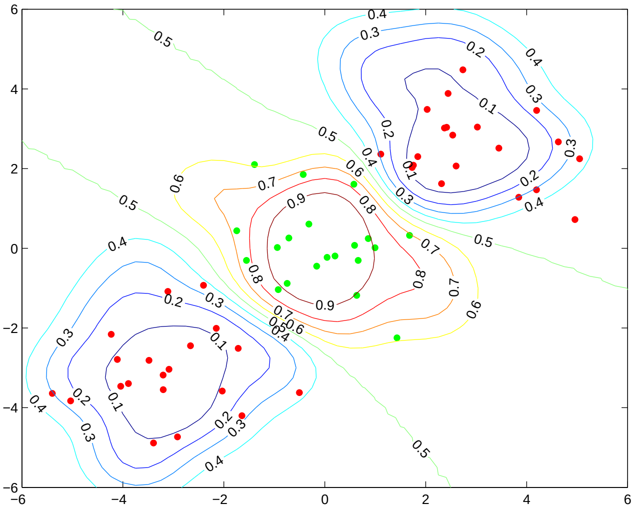


Figure 18.5: Bayesian Logistic Regression with the RBF $e^{-\lambda(\mathbf{x}-\mathbf{m})^2}$, placing basis functions centred on a subset of the training points. The green points are training data from class 1, and the red points are training data from class 0. The contours represent the probability of being in class 1. The optimal value of α found by the evidence procedure in this case is 0.45 (λ is set by hand to 2). See `demoBayesLogRegression.m`

so that

$$p(c=1|\mathbf{x},\mathcal{D},\alpha^*) = \int \sigma(h)\, p(h|\mathbf{x},\mathcal{D},\alpha^*)dh \tag{18.2.24}$$

Under the Laplace approximation, $\mathbf{w}$ is Gaussian,

$$p(\mathbf{w}|\mathcal{D}) = \mathcal{N}(\mathbf{w}|\boldsymbol{\mu},\boldsymbol{\Sigma}), \qquad \boldsymbol{\mu} = \mathbf{w}^*, \boldsymbol{\Sigma} = (\mathbf{H}(\mathbf{w}^*))^{-1} \tag{18.2.25}$$

Since h is a projection of $\mathbf{w}$, h is also Gaussian distributed

$$p(h|\mathbf{x},\mathcal{D},\alpha^*) = \mathcal{N}\left(h|\mathbf{x}^\mathsf{T}\mathbf{w}^*, \mathbf{x}^\mathsf{T}\boldsymbol{\Sigma}\mathbf{x}\right) \tag{18.2.26}$$

Predictions are then made by numerically evaluating the one-dimensional integral over the Gaussian distribution in h, equation (18.2.24).

Approximating the Gaussian average of a logistic sigmoid

Predictions under a Gaussian posterior approximation require the computation of

$$I \equiv \langle\sigma(x)\rangle_{\mathcal{N}(x|\mu,\sigma^2)} \tag{18.2.27}$$

where $\mu = \mathbf{x}^\mathsf{T}\boldsymbol{\mu}$, $\sigma^2 = \mathbf{x}^\mathsf{T}\boldsymbol{\Sigma}\mathbf{x}$. Gaussian quadrature is an obvious numerical candidate. An alternative is to replace the logistic sigmoid by a suitably transformed erf function[178], the reason being that the Gaussian average of an erf function is another erf function. Using a single erf, an approximation is[1]

$$\sigma(x) \approx \frac{1}{2}\left(1+\mathrm{erf}(\lambda x)\right) \tag{18.2.28}$$

These two functions agree at $-\infty, 0, \infty$. A reasonable criterion is that the derivatives of these two should agree at $x=0$ since then they have locally the same slope around the origin and have globally similar shape. Using $\sigma(0)=0.5$ and that the derivative is $\sigma(0)(1-\sigma(0))$, this requires

$$\frac{1}{4} = \frac{\lambda}{\sqrt{\pi}} \Rightarrow \lambda = \frac{\sqrt{\pi}}{4} \tag{18.2.29}$$

A more accurate approximation can be found by considering

$$\sigma(x) \approx \sum_i \frac{u_i}{2}\left(1+\mathrm{erf}(\lambda_i x)\right) \tag{18.2.30}$$

[1]Note that the definition of the erf function used here is taken to be consistent with MATLAB, namely that $\mathrm{erf}(x) \equiv \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt$. Other authors define it to be the cumulative density function of a standard Gaussian, $\frac{2}{\sqrt{\pi}}\int_{-\infty}^x e^{-\frac{1}{2}t^2}dt$.

where $\sum_i u_i = 1$. Suitable values for u_i and λ_i are given in `logsigapp.m` which uses a linear combination of 11 erf functions to approximate the logistic sigmoid. To compute the approximate average of $\sigma(x)$ over a Gaussian, one may then make use of the result

$$\frac{1}{\sqrt{2\pi}}\int_{\infty}^{\infty} e^{-\frac{x^2}{2}}\mathrm{erf}\left(\frac{1}{\sqrt{2}}(cx+d)\right)dx = \mathrm{erf}\left(\frac{d}{\sqrt{2+2c^2}}\right) \tag{18.2.31}$$

Consider

$$\frac{1}{\sqrt{2\pi\sigma^2}}\int_{\infty}^{\infty} e^{-\frac{(x-\mu)^2}{2\sigma^2}}\mathrm{erf}(\lambda x)\,dx = \frac{1}{\sqrt{2\pi}}\int_{\infty}^{\infty} e^{-\frac{x^2}{2}}\mathrm{erf}(\lambda(\sigma x+\mu))\,dx \tag{18.2.32}$$

Hence

$$\langle\sigma(x)\rangle_{\mathcal{N}(x|\mu,\sigma)} \approx \frac{1}{2}+\frac{1}{2}\sum_i u_i \mathrm{erf}\left(\frac{\lambda_i\mu}{\sqrt{1+2\lambda_i^2\sigma^2}}\right) \tag{18.2.33}$$

Further approximate statistics can be obtained using the results derived in [22].

18.2.3 Relevance vector machine for classification

In adopting the RVM prior to classification, we encourage individual weights to be small

$$p(\mathbf{w}|\boldsymbol{\alpha}) = \prod_i p(w_i|\alpha_i) \tag{18.2.34}$$

where

$$p(w_i|\alpha_i) = \mathcal{N}\left(w_i \middle| 0, \frac{1}{\alpha_i}\right) \tag{18.2.35}$$

The only alterations in the previous evidence procedure are

$$[\nabla E]_i = \alpha_i w_i - \sum_n (1-\sigma^n)h_i^n, \qquad \mathbf{H} = \mathrm{diag}(\boldsymbol{\alpha}) + \mathbf{J} \tag{18.2.36}$$

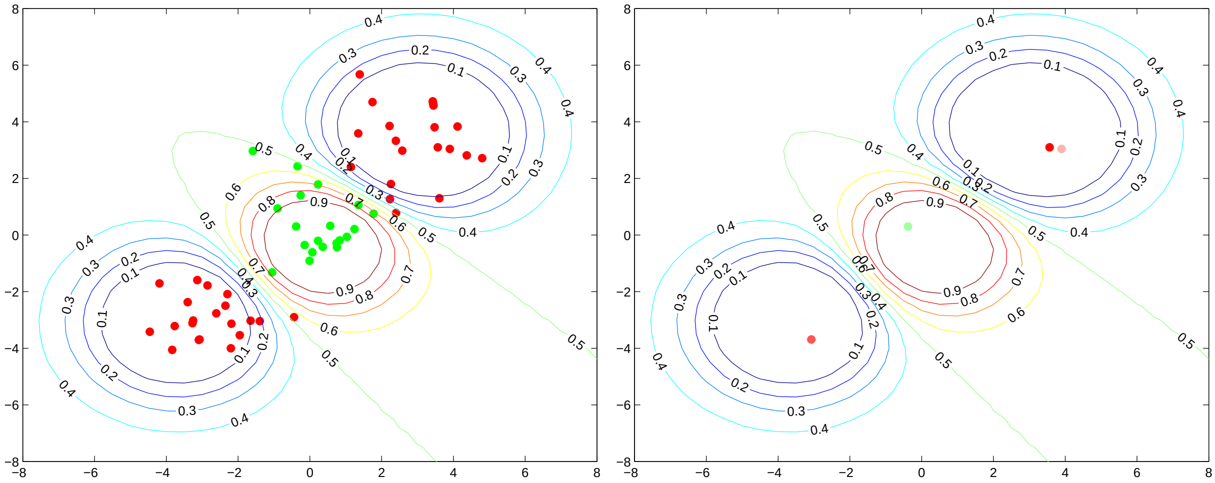
These are used in the Newton update formula as before. The EM update equation for the α's is given by

$$\alpha_i^{new} = \frac{1}{(w_i^*)^2 + \Sigma_{ii}} \tag{18.2.37}$$

where $\boldsymbol{\Sigma} = \mathbf{H}^{-1}$. Similarly, the Gull-MacKay update is given by

$$\alpha_i^{new} = \frac{1-\alpha_i[\mathbf{S_w}]_{ii}}{(w_i^*)^2} \tag{18.2.38}$$

Running this procedure, one typically finds that many of the α's tend to infinity and the corresponding weights may be effectively pruned. Those remaining tend to be rather in the centres of mass of a bunch of datapoints of the same class. Contrast this with the situation in SVMs, where the retained datapoints tend to be on the decision boundaries. The number of training points retained by the RVM tends to be very small – smaller indeed that the number retained in the SVM framework. Whilst the RVM does not support large margins, and hence may be a less robust classifier, it does retain the advantages of a probabilistic framework[273]. A potential critique of the RVM, coupled with an ML-II procedure for learning the α_i is that it is overly aggressive in terms of pruning. Indeed, as one may verify running `demoBayesLogRegRVM.m` it is common to find an instance of a problem for which there exists a set of α_i such that the training data can be classified perfectly. However, after using ML-II, so many α_i are set to zero that the training data can no longer be classified perfectly.



(a) (b)

Figure 18.6: Classification using the RVM with RBF $e^{-\lambda(\mathbf{x}-\mathbf{m})^2}$, placing a basis function on a subset of the training data points. The green points are training data from class 1, and the red points are training data from class 0. The contours represent the probability of being in class 1. **(a)**: Training points. **(b)**: The training points weighted by their relevance value $1/\alpha_n$. Nearly all the points have a value so small that they effectively vanish. See `demoBayesLogRegRVM.m`

18.2.4 Variational approximation

Here we consider an alternative to the Laplace approximation based on fitting a Gaussian using the Kullback-Leibler divergence. For fixed α, the marginal likelihood is given by the integral

$$p(\mathcal{D}) = \int p(\mathbf{w}) \prod_n \sigma\left(s_n \mathbf{w}^\mathsf{T} \mathbf{x}^n\right) \tag{18.2.39}$$

where $s_n = 2c_n - 1$. We may use a Gaussian approximation $q(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{m}, \mathbf{S})$

$$\mathrm{KL}(q(\mathbf{w})|p(\mathbf{w}|\mathcal{D})) = \langle \log q(\mathbf{w}) \rangle_{q(\mathbf{w})} - \langle \log p(\mathbf{w}|\mathcal{D}) \rangle_{q(\mathbf{w})} \geq 0 \tag{18.2.40}$$

which gives the marginal likelihood bound

$$\log p(\mathcal{D}) \geq - \langle \log q(\mathbf{w}) \rangle_{q(\mathbf{w})} + \langle \log p(\mathbf{w}, \mathcal{D}) \rangle_{q(\mathbf{w})} \tag{18.2.41}$$

The entropy term is

$$H \equiv - \langle \log q(\mathbf{w}) \rangle_{q(\mathbf{w})} = \frac{1}{2} \log \det (2\pi \mathbf{S}) + \frac{D}{2} \tag{18.2.42}$$

The energy term is

$$E = \langle \log p(\mathbf{w}, \mathcal{D}) \rangle_{q(\mathbf{w})} = \langle \log p(\mathbf{w}) \rangle_{q(\mathbf{w})} + \sum_n \left\langle \log \sigma\left(s_n \mathbf{w}^\mathsf{T} \mathbf{x}^n\right) \right\rangle_{q(\mathbf{w})} \tag{18.2.43}$$

$$\langle \log p(\mathbf{w}) \rangle_{q(\mathbf{w})} = -\frac{1}{2} \mathrm{trace}\left(\boldsymbol{\Sigma}_w^{-1} \left\langle \mathbf{w}\mathbf{w}^\mathsf{T} \right\rangle\right) - \frac{1}{2} \log \det (2\pi \boldsymbol{\Sigma}_w) \tag{18.2.44}$$

$$= -\frac{1}{2} \left(\mathrm{trace}\left(\boldsymbol{\Sigma}_w^{-1} \mathbf{S}\right) + \mathbf{m}^\mathsf{T} \boldsymbol{\Sigma}_w^{-1} \mathbf{m} + \log \det (2\pi \boldsymbol{\Sigma}_w) \right) \tag{18.2.45}$$

Finding the best Gaussian

To find the best Gaussian in the minimal Kullback-Leibler sense we may differentiate the bound and use the corresponding gradient as part of a standard optimisation routine. The derivative of the entropy is

$$\frac{\partial}{\partial S_{ij}} H = \frac{1}{2} \left[\mathbf{S}^{-1}\right]_{ij} \tag{18.2.46}$$

The derivative of the energy with respect to $\mathbf{S}$ is $-\frac{1}{2}\mathbf{\Sigma}_w^{-1}$. And with respect to $\mathbf{m}$ is $-\mathbf{\Sigma}_w^{-1}\mathbf{m}$. Writing

$$I^n \equiv \left\langle \log \sigma \left(s_n \mathbf{w}^\mathsf{T} \mathbf{x}^n \right) \right\rangle_{q(\mathbf{w})} \tag{18.2.47}$$

we can express this as

$$I^n = \langle \log \sigma(a) \rangle_{\mathcal{N}(a|\mu_n,\sigma_n^2)} = \langle \log \sigma \left(\mu_n + z\sigma_n \right) \rangle_{\mathcal{N}(z|0,1)} \tag{18.2.48}$$

where

$$\mu_n = s_n \left(\mathbf{x}^n \right)^\mathsf{T} \mathbf{m}, \qquad \sigma_n^2 = \left(\mathbf{x}^n \right)^\mathsf{T} \mathbf{S} \left(\mathbf{x}^n \right) \tag{18.2.49}$$

Then

$$\frac{\partial}{\partial m_i} I^n = s_n x_i^n \left\langle \log \sigma' \left(\mu_n + z\sigma_n \right) \right\rangle_{\mathcal{N}(z|0,1)} = s_n x_i^n \left(1 - \langle \sigma \left(\mu_n + z\sigma_n \right) \rangle_{\mathcal{N}(z|0,1)} \right) \tag{18.2.50}$$

and

$$\frac{\partial}{\partial S_{ij}} I^n = \frac{x_i^n x_j^n}{2\sigma_n} \left\langle z \log \sigma' \left(\mu_n + z\sigma_n \right) \right\rangle_{\mathcal{N}(z|0,1)} = -\frac{x_i^n x_j^n}{2\sigma_n} \langle z\sigma \left(\mu_n + z\sigma_n \right) \rangle_{\mathcal{N}(z|0,1)} \tag{18.2.51}$$

and $\log \sigma' = 1 - \sigma$. Putting this all together, we have

$$\log p(\mathcal{D}) \geq B(\mathbf{m}, \mathbf{S}) \tag{18.2.52}$$

with

$$\frac{\partial}{\partial m_i} B = -\left[\mathbf{\Sigma}_W^{-1} \mathbf{m} \right]_i + \sum_n \frac{\partial}{\partial m_i} I^n \tag{18.2.53}$$

and

$$\frac{\partial}{\partial S_{ij}} B = -\left[\frac{1}{2} \mathbf{\Sigma}_w^{-1} \right]_{ij} + \frac{1}{2} \left[\mathbf{S}^{-1} \right]_{ij} + \sum_n \frac{\partial}{\partial S_{ij}} I^n \tag{18.2.54}$$

Similarly the Hessian for $\mathbf{m}$ is given by

$$\frac{\partial^2}{\partial m_i \partial m_j} B = -\left[\mathbf{\Sigma}_W^{-1} \right]_{i,j} + \sum_n \frac{\partial^2}{\partial m_i \partial m_j} I^n \tag{18.2.55}$$

with

$$\frac{\partial^2}{\partial m_i \partial m_j} I^n = -x_i^n x_j^n \left\langle \sigma \left(\mu_n + z\sigma_n \right) \left(1 - \sigma \left(\mu_n + z\sigma_n \right) \right) \right\rangle_{\mathcal{N}(z|0,1)} \tag{18.2.56}$$

Comparison with local variational method

TO BE COMPLETED. See Bishop's book for a description of Jaakkola and Jordan's local variational approach. The above global method is theoretically a tighter bound than the local method. To show this one can see the 'local' method as in fact a global KL Gaussian approximation with an additional bound on the logistic sigmoid function. The second bound is not required in the global method, resulting in a tighter bound.

We note in passing that alternative variational procedures can be formed, such as factorised approximate distributions, potentially in a rotated co-ordinate system.

18.2.5 Multi-class case

We briefly mention that the multi-class case can be treated by using the softmax function under a one-of-m class coding scheme. The class probabilities are

$$p(c = m|y) = \frac{e^{y_m}}{\sum_{m'} e^{y_{m'}}} \tag{18.2.57}$$

which automatically enforces the constraint $\sum_m p(c = m) = 1$. Naively it would appear that for C classes, the cost of the Laplace approximation scales as $O\left(C^3 N^3\right)$. However, one may show by careful implementation that the cost is only $O\left(CN^3\right)$, analogous to the cost savings possible in the Gaussian Process for classification model [294, 226].

18.3 Notes

18.4 Code

`demoBayesLinReg.m`: Demo of Bayesian Linear Regression
`BayesLinReg.m`: Bayesian Linear Regression

`demoBayesLogRegRVM.m`: Demo of Bayesian Logistic Regression (RVM)
`BayesLogRegressionRVM.m`: Bayesian Logistic Regression (RVM)

`avsigmaGauss.m`: Approximation of the Gaussian average of a logistic sigmoid
`logsigapp.m`: Approximation of the Logistic Sigmoid using mixture of erfs

18.5 Exercises

Exercise 184. *The exercise concerns Bayesian regression.*

1. *Show that for*

$$f = \mathbf{w}^\mathsf{T}\mathbf{x} \tag{18.5.1}$$

and $p(\mathbf{w}) \sim \mathcal{N}(\mathbf{w}|\mathbf{0}, \mathbf{\Sigma})$, that $p(f|\mathbf{x})$ is Gaussian distributed. Furthermore, find the mean and covariance of this Gaussian.

2. *Consider a target point $t = f + \epsilon$, where $\epsilon \sim \mathcal{N}\left(\epsilon|0, \sigma^2\right)$. What is $p(f|t, \mathbf{x})$?*

Exercise 185. *A Bayesian Linear Parameter regression model is given by*

$$y^n = \mathbf{w}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}^n) + \eta^n \tag{18.5.2}$$

In vector notation $\mathbf{y} = \left(y^1, \ldots, y^N\right)$ this can be written

$$\mathbf{y} = \mathbf{\Phi}\mathbf{w} + \boldsymbol{\eta} \tag{18.5.3}$$

with $\mathbf{\Phi}^\mathsf{T} = \left[\boldsymbol{\phi}(\mathbf{x}^1), \ldots, \boldsymbol{\phi}(\mathbf{x}^N)\right]$ and $\boldsymbol{\eta}$ is a zero mean Gaussian distributed vector with covariance $\beta^{-1}\mathbf{I}$. An expression for the marginal likelihood of a dataset is given in equation (18.1.18). A more compact expression can be obtained by considering

$$p(y^1, \ldots, y^N|\mathbf{x}^1, \ldots, \mathbf{x}^N, \Gamma) \tag{18.5.4}$$

Since y^n is linearly related to $\mathbf{x}^n$ through $\mathbf{w}$, Then $\mathbf{y}$ is Gaussian distributed with mean

$$\langle \mathbf{y} \rangle = \mathbf{\Phi} \langle \mathbf{w} \rangle = \mathbf{0} \tag{18.5.5}$$

and covariance matrix

$$\left\langle \mathbf{y}\mathbf{y}^\mathsf{T} \right\rangle - \langle \mathbf{y} \rangle \langle \mathbf{y} \rangle^\mathsf{T} = \left\langle (\mathbf{\Phi}\mathbf{w} + \boldsymbol{\eta})(\mathbf{\Phi}\mathbf{w} + \boldsymbol{\eta})^\mathsf{T} \right\rangle \tag{18.5.6}$$

For $p(\mathbf{w}) = \mathcal{N}\left(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I}\right)$:

1. *Show that the covariance matrix can be expressed as*

$$\mathbf{C} = \frac{1}{\beta}\mathbf{I} + \frac{1}{\alpha}\mathbf{\Phi}\mathbf{\Phi}^{\mathsf{T}} \tag{18.5.7}$$

2. *Hence show that the log marginal likelihood can we written as*

$$\log p(y^1, \ldots, y^N|\mathbf{x}^1, \ldots, \mathbf{x}^N, \Gamma) = -\frac{1}{2}\log\det(2\pi\mathbf{C}) - \frac{1}{2}\mathbf{y}^{\mathsf{T}}\mathbf{C}^{-1}\mathbf{y} \tag{18.5.8}$$

Exercise 186. *Using exercise(185) as a basis, derive expression (18.1.30) for the log likelihood on a validation set.*

Exercise 187. *Show that*

$$\frac{\partial}{\partial\theta}\left\langle \log p(x|\theta)\right\rangle_{p(x|\theta^0)}|_{\theta=\theta^0} = 0 \tag{18.5.9}$$

Exercise 188. *Consider the function $E(\mathbf{w})$ as defined in equation (18.2.5).*

1. *Compute the Hessian matrix which has elements,*

$$H_{ij} \equiv \frac{\partial^2}{\partial w_i \partial w_j}E(\mathbf{w}) = \alpha\delta_{ij} + \sum_{n=1}^{N}\sigma^n\left(1-\sigma^n\right)\boldsymbol{\phi}^n\left(\boldsymbol{\phi}^n\right)^{\mathsf{T}} \tag{18.5.10}$$

2. *Show that the Hessian is positive semi-definite*

Exercise 189. *Show that for any function $f(\cdot)$,*

$$\int f(\mathbf{x}^{\mathsf{T}}\mathbf{w})p(\mathbf{w})d\mathbf{w} = \int f(h)p(h)dh \tag{18.5.11}$$

where $p(h)$ is the distribution of the scalar $\mathbf{x}^{\mathsf{T}}\mathbf{w}$. The significance of this result is that any high-dimensional integral of the above form can be reduced to a one-dimensional integral over the distribution of the 'field' h[22].

Exercise 190. *This exercise concerns Bayesian Logistic Regression. Our interest to derive a formula for the optimal regularisation parameter α based on the Laplace approximation to the marginal log-likelihood given by*

$$\log p(\mathcal{D}|\alpha) \approx L(\alpha) \equiv -\frac{\alpha}{2}(\mathbf{w})^{\mathsf{T}}\mathbf{w} + \sum_n \log\sigma\left((\mathbf{w})^{\mathsf{T}}\mathbf{h}^n\right) - \frac{1}{2}\log\det(\alpha\mathbf{I}+\mathbf{J}) + \frac{B}{2}\log\alpha \tag{18.5.12}$$

The Laplace procedure finds first an optimal $\mathbf{w}^*$ that maximises $\alpha\mathbf{w}^{\mathsf{T}}\mathbf{w}/2 - \sum_n \log\sigma\left(\mathbf{w}^{\mathsf{T}}\mathbf{h}^n\right)$, as in equation (18.2.5) which will depend on the setting of α. Formally, therefore, in finding the α that optimises $L(\alpha)$ we should make use of the total derivative formula

$$\frac{dL}{d\alpha} = \frac{\partial L}{\partial\alpha} + \sum_i \frac{\partial L}{\partial w_i}\frac{\partial w_i}{\partial\alpha} \tag{18.5.13}$$

However, when evaluated at $\mathbf{w} = \mathbf{w}^*$, $\frac{\partial L}{\partial \mathbf{w}} = \mathbf{0}$. This means that in order to compute the derivative with respect to α, we only need consider the terms with an explicit α dependence. Show that by differentiating w.r.t. *α, and using*

$$\partial\log\det(\mathbf{M}) = \text{trace}\left(\mathbf{M}^{-1}\partial\mathbf{M}\right) \tag{18.5.14}$$

and setting to zero, the optimal α satisfies the fixed point equation

$$\alpha^{new} = \frac{N}{(\mathbf{w}^*)^{\mathsf{T}}\mathbf{w}^* + \text{trace}\left((\alpha\mathbf{I}+\mathbf{J})^{-1}\right)} \tag{18.5.15}$$

CHAPTER 19

Gaussian Processes

19.1 Non-parametric prediction

Gaussian Processes are flexible Bayesian models that fit well within the probabilistic modelling framework. In developing GPs it is useful to first step back and see what information we need to form a predictor. Given a set of training data

$$\mathcal{D} = \{(x^n, y^n), n = 1, \ldots, N\} = \mathcal{X} \cup \mathcal{Y} \tag{19.1.1}$$

where x^n is the input for datapoint n and y^n the corresponding output (a continuous variable in the regression case and a discrete variable in the classification case), our aim is to make a prediction y^* for a new input x^*. In the discriminative framework no model of the inputs x is assumed and only the outputs are modelled, conditioned on the inputs. Given a joint model

$$p(y^1, \ldots, y^N, y^*|x^1, \ldots, x^N, x^*) = p(\mathcal{Y}, y^*|\mathcal{X}, x^*) \tag{19.1.2}$$

we subsequently use conditioning to form a predictor $p(y^*|x^*, \mathcal{D})$. In (19.1.2) there is no requirement that a parametric model $p(y^n|x^n, \theta)$ lies behind the data – provided we have a joint model of all outputs conditioned on the inputs, in principle, predictions can be made. In previous chapters we've made much use of the i.i.d. assumption that each datapoint is independently sampled from the same generating distribution. In this context, this might appear to suggest the assumption

$$p(y^1, \ldots, y^N, y^*|x^1, \ldots, x^N, x^*) = p(y^*|\mathcal{X}, x^*) \prod_n p(y^n|\mathcal{X}, x^*) \tag{19.1.3}$$

However, this is clearly of little use since the predictive conditional is simply $p(y^*|\mathcal{D}, x^*) = p(y^*|\mathcal{X}, x^*)$ meaning the predictions make no use of the training outputs. For a non-trivial predictor we therefore need to specify a joint non-factorised distribution over outputs.

19.1.1 From parametric to non-parametric

If we revisit our i.i.d. assumptions for parametric models, we used a parameter θ to make a model of the input-output distribution $p(y|x, \theta)$. For a parametric model predictions are formed using

$$p(y^*|x^*, \mathcal{D}) \propto p(y^*, x^*, \mathcal{D}) = \int_\theta p(y^*, \mathcal{Y}, x^*, \mathcal{X}, \theta) \propto \int_\theta p(y^*, \mathcal{Y}|\theta, x^*, \mathcal{X}) p(\theta|\cancel{x^*, \mathcal{X}}) \tag{19.1.4}$$

Under the assumption that, given θ, the data is i.i.d., we obtain

$$p(y^*|x^*, \mathcal{D}) \propto \int_\theta p(y^*|x^*, \theta) p(\theta) \prod_n p(y^n|\theta, x^n) \propto \int_\theta p(y^*|x^*, \theta) p(\theta|\mathcal{D}) \tag{19.1.5}$$

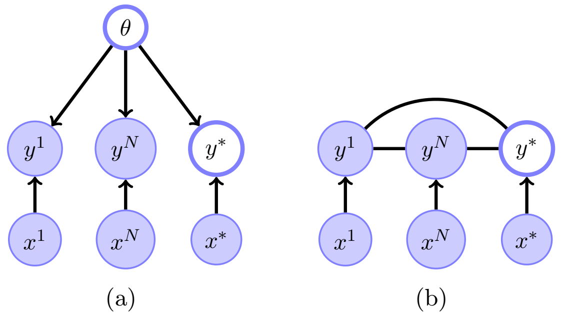


Figure 19.1: **(a)**: A parametric model for prediction assuming i.i.d. data. **(b)**: The form of the model after integrating out the parameters θ. Our non-parametric model will have this structure.

where

$$p(\theta|\mathcal{D}) \propto p(\theta) \prod_n p(y^n|\theta, x^n) \tag{19.1.6}$$

After integrating over the parameters θ, the joint data distribution is given by

$$p(y^*, \mathcal{Y}|x^*, \mathcal{X}) = \int_\theta p(y^*|x^*, \theta)p(\theta) \prod_n p(y^n|\theta, x^n) \tag{19.1.7}$$

which does not in general factorise into individual datapoint terms, see fig(19.1). The idea of a non-parametric approach is to specify the form of these dependencies without reference to an explicit parametric model. One route towards a non-parametric model is to start with a parametric model and integrate out the parameters. In order to make this tractable, we use a simple linear parameter predictor with a Gaussian parameter prior. For regression this leads to closed form expressions, although the classification case will require numerical approximation techniques.

19.1.2 From Bayesian linear models to Gaussian processes

To develop the GP, we briefly revisit the Bayesian linear parameter model of section(18.1.1). For parameters $\mathbf{w}$ and basis functions $\phi_i(x)$ the output is given by (assuming zero output noise)

$$y = \sum_i w_i \phi_i(x) \tag{19.1.8}$$

Assuming a Gaussian weight prior

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \Sigma_\mathbf{w}) \tag{19.1.9}$$

and since y is linear in $\mathbf{w}$, the joint output $y^1, \ldots, y^N$ is Gaussian distributed. If we stack all the $y^1, \ldots, y^N$ into a vector $\mathbf{y}$, then

$$\mathbf{y} = \Phi \mathbf{w} \tag{19.1.10}$$

where $\Phi = \left[\boldsymbol{\phi}(x^1), \ldots, \boldsymbol{\phi}(x^N)\right]^\mathsf{T}$ is the design matrix. A Gaussian prior on $\mathbf{w}$ induces a Gaussian on the joint $\mathbf{y}$ with mean

$$\langle \mathbf{y} \rangle = \Phi \langle \mathbf{w} \rangle_{p(\mathbf{w})} = \mathbf{0} \tag{19.1.11}$$

and covariance

$$\left\langle \mathbf{y}\mathbf{y}^\mathsf{T} \right\rangle = \Phi \left\langle \mathbf{w}\mathbf{w}^\mathsf{T} \right\rangle_{p(\mathbf{w})} \Phi^\mathsf{T} = \Phi \Sigma_\mathbf{w} \Phi^\mathsf{T} = \left(\Phi \Sigma_\mathbf{w}^{\frac{1}{2}}\right)\left(\Phi \Sigma_\mathbf{w}^{\frac{1}{2}}\right)^\mathsf{T} \tag{19.1.12}$$

From this we see that the $\Sigma_\mathbf{w}$ can be absorbed into Φ using its Cholesky decomposition. In other words, without loss of generality we may assume $\Sigma_\mathbf{w} = \mathbf{I}$. After integrating out the weights, the Bayesian linear regression model induces a Gaussian distribution on any set of outputs $\mathbf{y}$ as

$$p(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{K}) \tag{19.1.13}$$

where the covariance matrix $\mathbf{K}$ depends on the training inputs alone via

$$[\mathbf{K}]_{n,n'} = \boldsymbol{\phi}(x^n)^\mathsf{T}\boldsymbol{\phi}(x^{n'}), \qquad n, n' = 1, \ldots, N \tag{19.1.14}$$

Since the matrix $\mathbf{K}$ is formed as the scalar product of vectors, it is by construction positive (semi) definite, as we saw in section(17.3.2). After integrating out the weights, the only thing the model directly depends on is the covariance matrix $\mathbf{K}$. In a Gaussian process we directly specify the covariance joint output covariance $\mathbf{K}$ as a function of the inputs $\mathbf{x}$. Specifically we need to define the n, n' element of the covariance matrix for any two inputs x^n and $x^{n'}$. This will be achieved using a *covariance function* $k(x^n, x^{n'})$

$$[\mathbf{K}]_{n,n'} = k(x^n, x^{n'}) \tag{19.1.15}$$

The required form of the function $k(x^n, x^{n'})$ is very special – when applied to create the elements of the matrix $\mathbf{K}$ it must produce a positive definite matrix. We discuss in detail how to create such functions in section(19.3). As we saw above, one explicit straightforward construction is to form the covariance function from the scalar product of the basis vector $\boldsymbol{\phi}(x^n)$ and $\boldsymbol{\phi}(x^{n'})$. For finite-dimensional $\boldsymbol{\phi}$ this is known as a *finite dimensional Gaussian Process*. Given any covariance function, we can always find a corresponding basis vector representation – that is, for any GP, we can always relate this back to a parametric Bayesian LPM. However, for many commonly used covariance functions, the basis functions corresponds to infinite dimensional vectors. It is in such cases that the advantages of using the GP framework are particularly evident since we would not be able to compute efficiently with the corresponding infinite dimensional parametric model.

19.1.3 A prior on functions

The nature of many machine learning applications is such that the knowledge about the true underlying mechanism behind the data generation process is limited. Instead one relies on generic 'smoothness' assumptions of the form that for two inputs x and x' that are close, the corresponding outputs y and y' should be similar. Many generic techniques in machine learning can be viewed as different characterisations of smoothness. An advantage of the GP framework in this respect is that the mathematical smoothness properties of the functions are well understood, giving confidence in the procedure.

For a given covariance matrix $\mathbf{K}$, equation (19.1.13) specifies a distributions on functions[1] in the following sense: we specify a set of input points $\mathbf{x} = (x^1, \ldots, x^N)$ and a $N \times N$ covariance matrix $\mathbf{K}$. Then we draw a vector $\mathbf{y}$ from the Gaussian defined by equation (19.1.13). We can then plot the sampled 'function' at the finite set of points $(x^n, y^n), n = 1, \ldots, N$. What kind of functions does a GP correspond to? Consider two scalar inputs, x^i and x^j separated by a distance $|x^i - x^j|$. The corresponding sampled outputs y^i and y^j fluctuate as different functions are drawn. For a covariance function that has a high value for $|x^i - x^j|$ small, we expect y^i and y^j to be very similar since they are highly correlated. Conversely, for a covariance function that has low value for a given separation $|x^i - x^j|$, we expect y^i and y^j to be effectively independent[2]. In general, we would expect the correlation between y_i and y_j to decrease the further apart x_i and x_j are.

In fig(19.2a) we show three sample functions drawn from a Squared Exponential covariance function defined over 500 points uniformly spaced from -2 to 3. Each sampled function looks reasonably smooth, a characteristic which is related to the choice of the SE covariance function. For the Ornstein Uhlenbeck covariance function in fig(19.2c), the sampled functions look locally rough, see section(19.5.1).

The zero mean assumption implies that if we were to draw a large number of such 'functions', the mean across these functions at a given point x would tend to zero. Similarly, for any two points x and x' if we compute the sample covariance between the corresponding y and y' for all such sampled functions, this will tend to the covariance value $k(x, x')$. The zero-mean assumption can be easily relaxed by defining a

[1]The term 'function' is potentially confusing since we do not have an explicit functional form for the input output-mapping. For any finite set of inputs $x^1, \ldots, x^N$ we the values for the function are given by the outputs at those points $y^1, \ldots, y^N$.

[2]For periodic functions, however, we would expect high correlation at separating distances corresponding to the period of the function.

mean function $m(x)$ to give $p(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mathbf{y}|\mathbf{m}, \mathbf{K})$. In many practical situations one typically deals with 'detrended' data in which such mean trends have been already removed. For this reason much of the development of GPs in the machine learning literature is for the zero mean case.

19.2 Gaussian Process prediction

For a dataset $\mathcal{D}$ and novel input x^*, a zero mean GP makes a Gaussian model of the joint outputs $y^1, \ldots, y^N, y^*$ given the joint inputs $x^1, \ldots, x^N, x^*$. For convenience we write this as

$$p(\mathbf{y}, y^*|\mathbf{x}, x^*) = \mathcal{N}\left(\mathbf{y}, y^*|\mathbf{0}_{N+1}, \mathbf{K}^+\right) \tag{19.2.1}$$

where $\mathbf{0}_{N+1}$ is a $N+1$ dimensional zero-vector. The covariance matrix $\mathbf{K}^+$ is a block matrix with elements

$$\mathbf{K}^+ \equiv \begin{bmatrix} \mathbf{K}_{\mathbf{x},\mathbf{x}} & \mathbf{K}_{\mathbf{x},x^*} \\ \mathbf{K}_{x^*,\mathbf{x}} & \mathbf{K}_{x^*,x^*} \end{bmatrix}$$

where $\mathbf{K}_{\mathbf{x},\mathbf{x}}$ is the covariance matrix of the training inputs $\mathbf{x} = \left\{x^1, \ldots, x^N\right\}$ – that is

$$\left[\mathbf{K}_{\mathbf{x},\mathbf{x}}\right]_{n,n'} \equiv k(x^n, x^{n'}), \quad n, n' = 1, \ldots, N \tag{19.2.2}$$

The $N \times 1$ vector $\mathbf{K}_{\mathbf{x},x^*}$ has elements

$$\left[\mathbf{K}_{\mathbf{x},x^*}\right]_{n,*} \equiv k(x^n, x^*) \quad n = 1, \ldots, N \tag{19.2.3}$$

$\mathbf{K}_{x^*,\mathbf{x}}$ is the transpose of the above vector. The scalar covariance is given by

$$\mathbf{K}_{x^*,x^*} \equiv k(x^*, x^*) \tag{19.2.4}$$

The predictive distribution $p(y^*|\mathbf{y}, \mathcal{D})$ is obtained by Gaussian conditioning using the results in definition(78), giving a Gaussian distribution

$$p(y^*|x^*, \mathcal{D}) = \mathcal{N}\left(y^*\middle|\mathbf{K}_{x^*,\mathbf{x}}\mathbf{K}_{\mathbf{x},\mathbf{x}}^{-1}\mathbf{y}, \mathbf{K}_{x^*,x^*} - \mathbf{K}_{x^*,\mathbf{x}}\mathbf{K}_{\mathbf{x},\mathbf{x}}^{-1}\mathbf{K}_{\mathbf{x},x^*}\right) \tag{19.2.5}$$

For fixed hyperparameters, GP regression is an exact method and there are no issues with local minima during training. Furthermore, GPs are attractive since they automatically model uncertainty in the predictions. However, the computational complexity for making a prediction is $O\left(N^3\right)$ due to the requirement of performing the matrix inversion (or solving the corresponding linear system by Gaussian elimination). This can be prohibitively expensive for large datasets and a large body of research on efficient approximations exists. A discussion of these techniques is beyond the scope of this book, and the reader is referred to [226].

19.2.1 Regression with noisy training outputs

To prevent overfitting to noisy data it is useful to assume that a training output y^n is the result of some clean process f^n corrupted by additive Gaussian noise,

$$y^n = f^n + \epsilon^n, \quad \text{where} \quad \epsilon^n \sim \mathcal{N}\left(\epsilon^n|0, \sigma^2\right) \tag{19.2.6}$$

In this case our interest is to predict the clean signal f^* for a novel input x^*. Then the distribution $p(\mathbf{y}, f^*|\mathbf{x}, x^*)$ is a zero mean Gaussian with block covariance matrix

$$\begin{pmatrix} \mathbf{K}_{\mathbf{x},\mathbf{x}} + \sigma^2\mathbf{I} & \mathbf{K}_{\mathbf{x},x^*} \\ \mathbf{K}_{x^*,\mathbf{x}} & \mathbf{K}_{x^*,x^*} \end{pmatrix} \tag{19.2.7}$$

so that $\mathbf{K}_{\mathbf{x},\mathbf{x}}$ is replaced by $\mathbf{K}_{\mathbf{x},\mathbf{x}} + \sigma^2\mathbf{I}$ in forming the prediction, equation (19.2.5).

Example 84. Training data from a one-dimensional input x and one dimensional output y are plotted in fig(19.2b), along with the mean regression function fit, fig(19.2d), based on two different covariance functions. Note how the smoothness of the prior translates into smoothness of the prediction. The smoothness of the function space prior is a consequence of the choice of covariance function. Naively, we can partially understand this by the behaviour of the covariance function at the origin, section(19.5.1). See `demoGPreg.m`

The marginal likelihood and hyperparameter learning

For a set of N one-dimensional training inputs represented by the $N \times 1$ dimensional vector $\mathbf{y}$ and a covariance matrix $\mathbf{K}$ defined on the inputs $x^1, \ldots, x^N$, the log marginal likelihood can be computed exactly using

$$\log p(\mathbf{y}|\mathbf{x}) = -\frac{1}{2}\mathbf{y}^{\mathsf{T}}\mathbf{K}^{-1}\mathbf{y} - \frac{1}{2}\log\det(2\pi\mathbf{K}) \tag{19.2.8}$$

One can learn any free (hyper)parameters of the covariance function by maximising the marginal likelihood. For example, for a squared exponential covariance function with (hyper) parameters:

$$k(x, x') = v_0 \exp\left\{-\frac{1}{2}\lambda\left(x - x'\right)^2\right\} \tag{19.2.9}$$

The λ parameter in equation (19.2.12) specifies the appropriate length-scale of the inputs, and v_0 the variance of the function. Whilst equation (19.2.8) is a closed form exact expression for the log marginal likelihood, the dependence on the hyperparameters is typically complex so that no closed form expression for the Maximum Likelihood optimum exists, and one needs to use a numerical optimisation technique such as conjugate gradients.

Vector inputs

For regression with vector inputs and scalar outputs we need to define a covariance as a function of the two vectors, $k(\mathbf{x}, \mathbf{x}')$. Using the multiplicative property of covariance functions, definition(93), a simple way to do this is to define

$$k(\mathbf{x}, \mathbf{x}') = \prod_i k(x_i, x_i') \tag{19.2.10}$$

For example, for the squared exponential covariance function this gives

$$k(\mathbf{x}, \mathbf{x}') = e^{-(\mathbf{x}-\mathbf{x}')^2} \tag{19.2.11}$$

though 'correlated' forms are possible as well, see exercise(199).

Note that we can generalise the above using (hyper) parameters:

$$k(\mathbf{x}, \mathbf{x}') = v_0 \exp\left\{-\frac{1}{2}\sum_{l=1}^{D}\lambda_l\left(x_l - x_l'\right)^2\right\} \tag{19.2.12}$$

where x_l is the lth component of $\mathbf{x}$ and $\theta = (v_0, \lambda_1, \ldots, \lambda_D)$ are the hyperparameters. The λ_l parameters in equation (19.2.12) allow a different length scale on each input dimension. The hyperparameters can be learned by numerically maximising the marginal likelihood. For irrelevant inputs, the corresponding λ_l will become small, and the model will ignore the l^{th} input dimension. This is closely related to Automatic Relevance Determination [178].

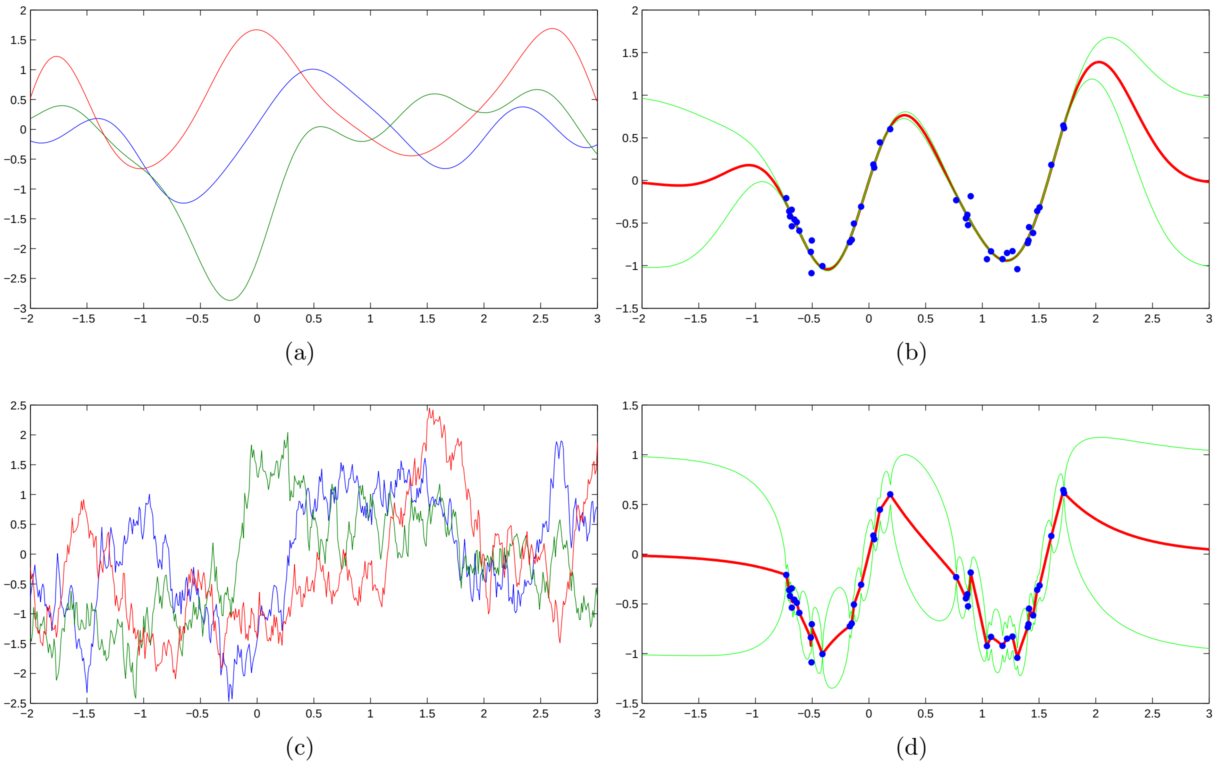


Figure 19.2: The input space from -2 to 3 is split evenly into 1000 points $x^1, \ldots, x^{1000}$. **(a)**: Three samples from a GP prior with Squared Exponential (SE) covariance function, $\lambda = 2$. The 1000×1000 covariance matrix $\mathbf{K}$ is defined using the SE kernel, from which the samples are drawn using `mvrandn(zeros(1000,1),K,3)`. **(b)**: Prediction based on training points. Plotted is the posterior predicted function based on the SE covariance. The central line is the mean prediction, with standard errors bars on either side. The log marginal likelihood is ≈ 70. **(c)**: Three samples from the Ornstein-Uhlenbeck GP prior with $\lambda = 2$. **(d)**: Posterior prediction for the OU covariance. The log marginal likelihood is ≈ 3, meaning that the SE covariance is much more heavily supported by the data than the rougher OU process.

19.3 Covariance Functions

Definition 91 (Covariance function (kernel))**.** Given any collection of points $x^1, \ldots, x^M$, a covariance function $c(x^i, x^j)$ defines the elements of a $M \times M$ matrix

$$[\mathbf{C}]_{i,j} = c(x^i, x^j)$$

such that $\mathbf{C}$ is positive semidefinite.

19.3.1 Making new covariance functions from old

The following rules (which can all be proved directly) generate new covariance functions from existing covariance functions k_1, k_2 [179],[226].

Definition 92 (Sum)**.**

$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') \tag{19.3.1}$$

Definition 93 (Product).

$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}') \tag{19.3.2}$$

Definition 94 (Product Spaces). For $\mathbf{z} = \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix}$,

$$k(\mathbf{z}, \mathbf{z}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{y}, \mathbf{y}') \tag{19.3.3}$$

and

$$k(\mathbf{z}, \mathbf{z}') = k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{y}, \mathbf{y}') \tag{19.3.4}$$

Definition 95 (Vertical Rescaling).

$$k(\mathbf{x}, \mathbf{x}') = a(\mathbf{x})k_1(\mathbf{x}, \mathbf{x}')a(\mathbf{x}') \tag{19.3.5}$$

for any function $a(\mathbf{x})$.

Definition 96 (Warping and Embedding).

$$k(\mathbf{x}, \mathbf{x}') = k_1\left(\mathbf{u}(\mathbf{x}), \mathbf{u}(\mathbf{x}')\right) \tag{19.3.6}$$

for any mapping $\mathbf{x} \to \mathbf{u}(\mathbf{x})$, where the mapping $\mathbf{u}(\mathbf{x})$ has arbitrary dimension.

19.4 Popular Covariance Functions

A small collection of covariance functions commonly used in Machine Learning is given below. We refer the reader to [226] and [104] for further popular covariance functions.

19.4.1 Stationary covariance functions

Definition 97 (Stationary Kernel). A kernel $k(x, x')$ is stationary if the kernel depends only on the separation $\mathbf{x} - \mathbf{x}'$. That is

$$k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x} - \mathbf{x}') \tag{19.4.1}$$

Following the notation in [226], for a stationary covariance function we may write

$$k(\mathbf{d}) \tag{19.4.2}$$

where $\mathbf{d} = \mathbf{x} - \mathbf{x}'$. This means that for functions drawn from the GP, on average, the functions depend only on the distance between inputs and not on the absolute position of an input. In other words, the functions are on average translation invariant.

For *isotropic covariance functions*, the covariance is defined as a function of the distance $k(|\mathbf{d}|)$.

Definition 98 (Squared Exponential)**.**

$$k(\mathbf{d}) = e^{-|\mathbf{d}|^2} \tag{19.4.3}$$

The Squared Exponential is one of the most common covariance functions. There are many ways to show that this is a covariance function. An elementary technique is to observe that

$$e^{-\frac{1}{2}\left(\mathbf{x}^n-\mathbf{x}^{n'}\right)^{\mathsf{T}}\left(\mathbf{x}^n-\mathbf{x}^{n'}\right)} = e^{-\frac{1}{2}|\mathbf{x}^n|^2}e^{-\frac{1}{2}|\mathbf{x}^{n'}|^2}e^{(\mathbf{x}^n)^{\mathsf{T}}\mathbf{x}^{n'}} \tag{19.4.4}$$

The first two factors form a kernel of the form $\phi(\mathbf{x}^n)\phi(\mathbf{x}^{n'})$. In the final term $k_1(\mathbf{x}^n, \mathbf{x}^{n'}) = (\mathbf{x}^n)^{\mathsf{T}}\mathbf{x}^{n'}$ is the linear kernel. Taking the exponential and writing the power series expansion of the exponential, we have

$$e^{k_1(\mathbf{x}^n,\mathbf{x}^{n'})} = \sum_{i=1}^{\infty}\frac{1}{i!}k_1^i(\mathbf{x}^n, \mathbf{x}^{n'}) \tag{19.4.5}$$

this can be expressed as a series of integer powers of k_1, with positive coefficients. By the product (with itself) and sum rules above, this is therefore a kernel as well. We then use the fact that equation (19.4.4) is the product of two kernels, and hence also a kernel.

Definition 99 (γ-Exponential)**.**

$$k(\mathbf{d}) = e^{-|\mathbf{d}|^{\gamma}}, \qquad 0 < \gamma \leq 2 \tag{19.4.6}$$

When $\gamma = 2$ we have the squared exponential covariance function. When $\gamma = 1$ this is the *Ornstein-Uhlenbeck* covariance function.

Definition 100 (Matérn)**.**

$$k(\mathbf{d}) = |\mathbf{d}|^{\nu}K_{\nu}\left(|\mathbf{d}|\right) \tag{19.4.7}$$

where K_ν is a modified Bessel function, $\nu > 0$.

Definition 101 (Rational Quadratic)**.**

$$k(\mathbf{d}) = \left(1 + |\mathbf{d}|^2\right)^{-\alpha}, \qquad \alpha > 0 \tag{19.4.8}$$

Definition 102 (Periodic)**.** For 1-dimensional x and x' a stationary (and isotropic) covariance function can be obtained by first mapping x to the two dimensional vector $\mathbf{u}(x) = (\cos(x), \sin(x))$ and then using the SE covariance $e^{-(\mathbf{u}(x)-\mathbf{u}(x'))^2}$

$$k(x - x') = e^{-\lambda \sin^2(\omega(x-x'))}, \qquad \lambda > 0 \tag{19.4.9}$$

See [179] and [226].

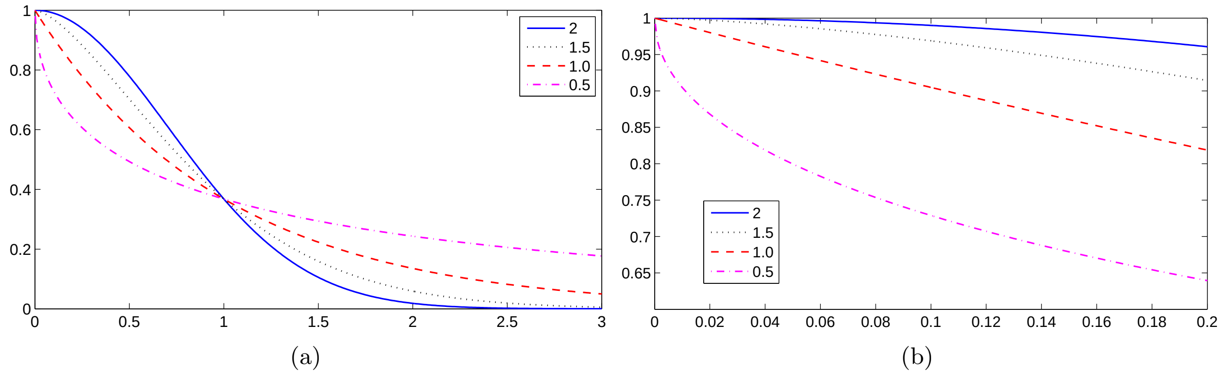


Figure 19.3: **(a)**: Plots of the Gamma-Exponential covariance $e^{-|x|^\gamma}$ versus x. The case $\gamma = 2$ corresponds to the SE covariance function. The drop in the covariance is much more rapid as a function of the separation x for small γ, suggesting that the functions corresponding to smaller γ will be locally rough (though possess relatively higher long range correlation). **(b)**: As for (a) but zoomed in towards the origin. For the SE case, $\gamma = 2$, the derivative of the covariance function is zero, whereas the OU covariance $\gamma = 1$ has a first order contribution to the drop in the covariance, suggesting that locally OU sampled functions will be much rougher than SE functions.

19.4.2 Non-stationary Covariance Functions

Definition 103 (Linear)**.**

$$k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\mathsf{T}\mathbf{x}' \tag{19.4.10}$$

Definition 104 (Neural Network)**.**

$$k(\mathbf{x}, \mathbf{x}') = \arcsin\left(\frac{2\mathbf{x}^\mathsf{T}\mathbf{x}'}{\sqrt{1 + 2\mathbf{x}^\mathsf{T}\mathbf{x}}\sqrt{1 + 2\mathbf{x}'^\mathsf{T}\mathbf{x}'}}\right) \tag{19.4.11}$$

The functions defined by this covariance always go through the origin. To shift this, one may use the embedding $\mathbf{x} \to (1, \mathbf{x})$ where the 1 has the effect of a 'bias' from the origin. To change the scale of the bias and and non-bias contributions one may use additional parameters $\mathbf{x} \to (b, \lambda\mathbf{x})$. The NN covariance function can be derived as a limiting case of a neural network with infinite hidden units[293], and making use of exact integral results in [21].

Definition 105 (Gibbs)**.**

$$k(\mathbf{x}, \mathbf{x}') = \prod_i \left(\frac{r_i(\mathbf{x})r_i(\mathbf{x}')}{r_i^2(\mathbf{x}) + r_i^2(\mathbf{x}')}\right)^{\frac{1}{2}} e^{-\frac{(x_i - x_i')^2}{r_i^2(\mathbf{x}) + r_i^2(\mathbf{x}')}} \tag{19.4.12}$$

for functions $r_i(\mathbf{x}) > 0$.

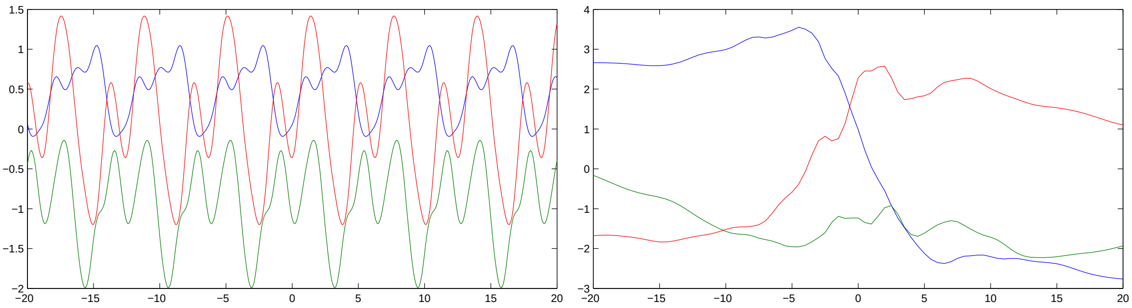


Figure 19.4: Samples from a GP prior for 500 x points uniformly placed from -20 to 20. **(a)**: Sampled from the periodic covariance function $\exp -2\sin^2(0.5(x-x'))$. **(b)**: Neural Network covariance function with bias $b = 5$ and $\lambda = 1$.

19.5 Analysis of Covariance Functions

19.5.1 Smoothness of the functions

We examine local smoothness for a translation invariant kernel $k(x, x') = k(x - x')$. For two one-dimensional points x and x', separated by a small amount $\delta \ll 1$, $x' = x + \delta$, the covariance between the outputs y and y' is, by Taylor expansion,

$$k(x, x') \approx k(0) + \delta \frac{dk}{dx}|_{x=0} + O\left(\delta^2\right) \tag{19.5.1}$$

so that the change in the covariance at the local level is dominated by the first derivative of the covariance function. For the SE covariance $k(x) = e^{-x^2}$,

$$\frac{dk}{dx} = -2xe^{-x^2} \tag{19.5.2}$$

which is zero at $x = 0$. This means that for the SE covariance function, the first order change in the covariance is zero, and only higher order δ^2 terms contribute.

For the Ornstein-Uhlenbeck covariance, $k(x) = e^{-|x|}$, the right derivative at the origin is

$$\lim_{\delta \to 0} \frac{k(\delta) - k(0)}{\delta} = \lim_{\delta \to 0} \frac{e^{-\delta} - 1}{\delta} = -1 \tag{19.5.3}$$

where this result is obtained using L'Hôpital's rule. Hence for the OU covariance function, there is a first order negative change in the covariance; at the local level, this decrease in the covariance is therefore much more rapid than for the SE covariance, see fig(19.3). Since low covariance implies low dependence (in Gaussian distributions), locally the functions generated from the OU process are rough, whereas they are smooth in the SE case. A more formal treatment for the stationary case can be obtained by examining the eigenvalue-frequency plot of the covariance function (spectral density), section(19.5.3). For rough functions the density of eigenvalues for high frequency components is higher than for smooth functions.

19.5.2 Mercer kernels

Consider the function

$$k(x, x') = \boldsymbol{\phi}(x)^\mathsf{T}\boldsymbol{\phi}(x') = \sum_{s=1}^{B} \phi_s(x)\phi_s(x') \tag{19.5.4}$$

where $\boldsymbol{\phi}(x)$ is a vector with component functions $\phi_1(x), \phi_2(x), \ldots, \phi_B(x)$. Then for a set of points $x^1, \ldots, x^P$, we construct the matrix $\mathbf{K}$ with elements

$$[\mathbf{K}]_{i,j} = k(x^i, x^j) = \sum_{s=1}^{B} \phi_s(x^i)\phi_s(x^j) \tag{19.5.5}$$

We claim that the matrix $\mathbf{K}$ so constructed is positive semidefinite and hence a valid covariance matrix. Recalling that a matrix is positive semi-definite if for any non zero vector $\mathbf{z}$, $\mathbf{z}^\mathsf{T}\mathbf{K}\mathbf{z} \geq 0$. Using the definition of $\mathbf{K}$ above we have

$$\mathbf{z}^\mathsf{T}\mathbf{K}\mathbf{z} = \sum_{i,j=1}^{P} z_i K_{i,j} z_j = \sum_{s=1}^{B} \underbrace{\left[\sum_{i=1}^{P} z_i \phi_s(x^i)\right]}_{\gamma_s} \underbrace{\left[\sum_{j=1}^{P} \phi_s(x^j) z_j\right]}_{\gamma_s} = \sum_{s=1}^{B} \gamma_s^2 \geq 0 \tag{19.5.6}$$

Hence any function of the form equation (19.5.4) is a covariance function. Mercer's Theorem states that any covariance function must be representable in the form equation (19.5.4) albeit for a possibly infinite dimensional vector $\boldsymbol{\phi}(x)$. See [226] for details.

We can generalise the Mercer kernel to complex functions $\boldsymbol{\phi}(x)$ using

$$k(x, x') = \boldsymbol{\phi}(x)^\mathsf{T} \boldsymbol{\phi}^\dagger(x') \tag{19.5.7}$$

where † represents the complex conjugate. Then the matrix $\mathbf{K}$ formed from inputs x^i, $i = 1, \ldots, P$ is positive (semi) definite since for any real vector $\mathbf{z}$,

$$\mathbf{z}^\mathsf{T}\mathbf{K}\mathbf{z} = \sum_{s=1}^{B} \underbrace{\left[\sum_{i=1}^{P} z_i \phi_s(x^i)\right]}_{\gamma_s} \underbrace{\left[\sum_{j=1}^{P} \phi_s^\dagger(x^j) z_j\right]}_{\gamma_s^\dagger} = \sum_{s=1}^{B} |\gamma_s|^2 \geq 0 \tag{19.5.8}$$

where we made use of the general result for a complex variable $xx^\dagger = |x|^2$. A further generalisation is to write

$$k(x, x') = \int f(s) \phi(x, s) \phi^\dagger(x', s) ds \tag{19.5.9}$$

for $f(s) \geq 0$, and scalar complex functions $\phi(x, s)$. Then replacing summations with integration (and assuming we can interchange the sum over the components of $\mathbf{z}$ with the integral over s), we obtain

$$\mathbf{z}^\mathsf{T}\mathbf{K}\mathbf{z} = \int f(s) \underbrace{\left[\sum_{i=1}^{P} z_i \phi(x^i, s)\right]}_{\gamma(s)} \underbrace{\left[\sum_{j=1}^{P} \phi^\dagger(x^j, s) z_j\right]}_{\gamma^\dagger(s)} ds = \int f(s) |\gamma(s)|^2 ds \geq 0 \tag{19.5.10}$$

Spectral Decomposition

Equation(19.5.9) is a generalisation of the spectral decomposition of a kernel $k(x, x')$ since if we write $f(s)$ as a sum of Dirac delta functions,

$$f(s) = \sum_{k=1}^{\infty} \lambda_k \delta(s - k) \tag{19.5.11}$$

and using $\boldsymbol{\phi}(x, k) = \psi_k(x)$, for an eigenfunction $\psi_k(x)$ indexed by k with eigenvalue λ_k, we obtain the spectral decomposition

$$k(x, x') = \sum_{k=1}^{\infty} \lambda_k \psi_k(x) \psi_k(x') \tag{19.5.12}$$

If we all the eigenvalues of a kernel are non-negative, the kernel is a covariance function.

Consider for example the following function

$$k(x, x') = e^{-(x-x')^2} \tag{19.5.13}$$

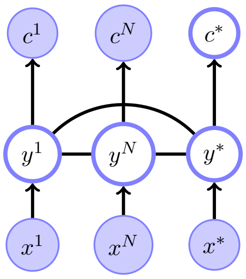


Figure 19.5: GP classification. The GP induces a Gaussian distribution on the latent activations $y_1, \ldots, y^N, y^*$, given the observed values of $c^1, \ldots, c^N$. The classification of the new input x^* is then given via the correlation induced by the training points on the latent activation y^*.

We claim that this is a covariance function. This is indeed a valid covariance function in the sense that for any set of points $x^1, \ldots, x^d$, the $d \times d$ matrix formed with elements $k(x^d, x^{d'})$ is positive definite, as discussed after definition(98). The solution given to exercise(193) shows that there do indeed exist real-valued vectors such that one can represent

$$k(x, x') = \boldsymbol{\phi}(x)^{\mathsf{T}} \boldsymbol{\phi}(x') \tag{19.5.14}$$

where the vectors are infinite dimensional. This demonstrates the generalisation of the finite-dimensional 'weight space' viewpoint of a GP to the potentially implicit infinite dimensional representation.

19.5.3 Fourier analysis for stationary kernels

For a function $g(x)$ with Fourier transform $\tilde{g}(s)$, we may use the inverse Fourier transform to write

$$g(x) = \frac{1}{2\pi} \int \tilde{g}(s) e^{-\mathrm{i}xs} ds \tag{19.5.15}$$

where $\mathrm{i} \equiv \sqrt{-1}$. For a stationary kernel $k(x)$ with Fourier transform $\tilde{k}(s)$, we can therefore write

$$k(x - x') = \frac{1}{2\pi} \int \tilde{k}(s) e^{-\mathrm{i}(x-x')s} ds = \frac{1}{2\pi} \int \tilde{k}(s) e^{-\mathrm{i}xs} e^{\mathrm{i}x's} ds \tag{19.5.16}$$

which is of the same form as equation (19.5.9) where the Fourier transform $\tilde{k}(s)$ is identified with $f(s)$ and $\phi(x, s) = e^{-\mathrm{i}sx}$. Hence, provided the Fourier transform $\tilde{k}(s)$ is positive, the translation invariant kernel $k(x - x')$ is a covariance function. Bochner's Theorem[226] asserts the converse that any translation invariant covariance function must have such a Fourier representation.

Application to the squared exponential Kernel

For the translation invariant squared exponential kernel, $k(x) = e^{-\frac{1}{2}x^2}$, its Fourier transform is

$$\tilde{k}(s) = \int_{-\infty}^{\infty} e^{-\frac{1}{2}x^2 + \mathrm{i}sx} dx = e^{-\frac{s^2}{2}} \int_{-\infty}^{\infty} e^{-\frac{1}{2}(x + \mathrm{i}s)^2} dx = \sqrt{2\pi} e^{-\frac{s^2}{2}} \tag{19.5.17}$$

Hence the Fourier transform of the SE kernel is a Gaussian. Since this is positive the SE kernel is a covariance function.

19.6 Gaussian Processes for Classification

Adapting the GP framework to classification requires replacing the Gaussian regression term $p(y|x)$ with a corresponding classification term $p(c|x)$ for a discrete label c. To do so we will use the GP to define a latent continuous space which will then be mapped to a class probability using

$$p(c|x) = \int p(c|y, \cancel{x}) p(y|x) dy = \int p(c|y) p(y|x) dy \tag{19.6.1}$$

Given training data inputs $\mathcal{X} = \{x^1, \ldots, x^N\}$, corresponding class labels $\mathcal{C} = \{c^1, \ldots, c^N\}$, and a novel input x^*, then

$$p(c^*|x^*, \mathcal{C}, \mathcal{X}) = \int p(c^*|y^*) p(y^*|\mathcal{X}, \mathcal{C}) dy^* \tag{19.6.2}$$

where

$$
\begin{aligned}
p(y^*|\mathcal{X},\mathcal{C}) &\propto p(y^*,\mathcal{C}|\mathcal{X}) \\
&= \int p(y^*,\mathcal{Y},\mathcal{C}|\mathcal{X},x^*)d\mathcal{Y} \\
&= \int p(\mathcal{C}|\mathcal{Y})p(y^*,\mathcal{Y}|\mathcal{X},x^*)d\mathcal{Y} \\
&= \int \underbrace{\left\{\prod_{n=1}^{N} p(c^n|y^n)\right\}}_{\text{class mapping}} \underbrace{p(y^1,\ldots,y^N,y^*|x^1,\ldots,x^N,x^*)}_{\text{Gaussian Process}} dy^1,\ldots,dy^N
\end{aligned}
\tag{19.6.3}
$$

The graphical structure of this model is depicted in fig(19.5.) The posterior latent y^* is then formed from the standard regression term from the Gaussian Process, multiplied by a set of non-Gaussian maps from the latent activations to the class probabilities. We can reformulate the prediction problem more conveniently as follows:

$$
p(y^*,\mathcal{Y}|x^*,\mathcal{X},\mathcal{C}) \propto p(y^*,\mathcal{Y},\mathcal{C}|x^*,\mathcal{X}) \propto p(y^*|\mathcal{Y},x^*,\mathcal{X})p(\mathcal{Y}|\mathcal{C},\mathcal{X})
\tag{19.6.4}
$$

where

$$
p(\mathcal{Y}|\mathcal{C},\mathcal{X}) \propto \left\{\prod_{n=1}^{N} p(c^n|y^n)\right\} p(y^1,\ldots,y^N|x^1,\ldots,x^N)
\tag{19.6.5}
$$

In equation (19.6.4) the term $p(y^*|\mathcal{Y},x^*,\mathcal{X})$ does not contain any class label information and is simply a conditional Gaussian. The advantage of the above description is that we can therefore form an approximation to $p(\mathcal{Y}|\mathcal{C},\mathcal{X})$ and then reuse this approximation in the prediction for many different x^* without needing to rerun the approximation[294, 226].

19.6.1 Binary Classification

For the binary class case we will use the convention that $c \in \{1,0\}$. We therefore need to specify $p(c=1|y)$ for a real valued activation y. A convenient choice is the logistic transfer function[3]

$$
\sigma(x) = \frac{1}{1+e^{-x}}
\tag{19.6.6}
$$

Then

$$
p(c|y) = \sigma\left((2c-1)\,y\right)
\tag{19.6.7}
$$

is a valid distribution since $\sigma(-x) = 1-\sigma(x)$, ensuring that the sum over the class states is 1. A difficulty is that the non-linear class mapping term makes the computation of the posterior distribution equation (19.6.3) difficult since the integrals over $y^1,\ldots,y^N$ cannot be carried out analytically. There are many approximate techniques one could apply in this case, including variational methods analogous to that described in section(18.2.4). Below we describe the straightforward Laplace method, leaving the more sophisticated methods for further reading[226].

19.6.2 Laplace's approximation

In the Laplace method we approximate the non-Gaussian distribution (19.6.5) by a Gaussian[4] $q(\mathcal{Y}|\mathcal{C},\mathcal{X})$,

$$
p(\mathcal{Y}|\mathcal{C},\mathcal{X}) \approx q(\mathcal{Y}|\mathcal{C},\mathcal{X})
\tag{19.6.8}
$$

[3] We will also refer to this as 'the sigmoid function'. More strictly a sigmoid function refers to any 's-shaped' function (from the Greek for 's').

[4] Some authors use the term Laplace approximation solely for approximating an integral. Here we use the term to refer to a Gaussian approximation of a non-Gaussian distribution.

Predictions can be formed from the joint Gaussian

$$p(y^*, \mathcal{Y}|x^*, \mathcal{X}, \mathcal{C}) \approx p(y^*|\mathcal{Y}, x^*, \mathcal{X})q(\mathcal{Y}|\mathcal{C}, \mathcal{X}) \tag{19.6.9}$$

For compactness we define the class label vector, and outputs

$$\mathbf{c} = \left(c^1, \ldots, c^N\right), \qquad \mathbf{y} = \left(y^1, \ldots, y^N\right) \tag{19.6.10}$$

and notationally drop the (ever present) conditioning on the inputs x. Also for convenience, we define

$$\boldsymbol{\sigma} = \left(\sigma(y^1), \ldots, \sigma(y^N)\right) \tag{19.6.11}$$

Finding the mode

The Laplace approximation, section(28.2), corresponds to a second order expansion around the mode of the distribution. Our task is therefore to find the maximum of

$$p(\mathbf{y}|\mathbf{c}) \propto p(\mathbf{y}, \mathbf{c}) = e^{\Psi(\mathbf{y})} \tag{19.6.12}$$

where

$$\Psi(\mathbf{y}) = \mathbf{c}^\mathsf{T}\mathbf{y} - \sum_{n=1}^{N} \log(1 + e^{y_n}) - \frac{1}{2}\mathbf{y}^\mathsf{T}\mathbf{K}_{\mathbf{x},\mathbf{x}}^{-1}\mathbf{y} - \frac{1}{2}\log\det\left(\mathbf{K}_{\mathbf{x},\mathbf{x}}\right) - \frac{N}{2}\log 2\pi \tag{19.6.13}$$

The maximum needs to be found numerically, and it is convenient to use the Newton method [117, 294, 226].

$$\mathbf{y}^{new} = \mathbf{y} - \left(\nabla\nabla\Psi\right)^{-1}\nabla\Psi \tag{19.6.14}$$

Differentiating equation 19.6.13 with respect to $\mathbf{y}$ we obtain the gradient and Hessian

$$\nabla\Psi = (\mathbf{c} - \boldsymbol{\sigma}) - \mathbf{K}_{\mathbf{x},\mathbf{x}}^{-1}\mathbf{y} \tag{19.6.15}$$

$$\nabla\nabla\Psi = -\mathbf{K}_{\mathbf{x},\mathbf{x}}^{-1} - \mathbf{D} \tag{19.6.16}$$

where the 'noise' matrix is given by

$$\mathbf{D} = \mathrm{diag}\left(\sigma_1(1-\sigma_1), \ldots, \sigma_N(1-\sigma_N)\right) \tag{19.6.17}$$

Using these expressions in the Newton update, (19.6.14) gives

$$\mathbf{y}^{new} = \mathbf{y} + \left(\mathbf{K}_{\mathbf{x},\mathbf{x}}^{-1} + \mathbf{D}\right)^{-1}\left(\mathbf{c} - \boldsymbol{\sigma} - \mathbf{K}_{\mathbf{x},\mathbf{x}}^{-1}\mathbf{y}\right) \tag{19.6.18}$$

To avoid unnecessary inversions, one may rewrite this in the form

$$\mathbf{y}^{new} = \mathbf{K}_{\mathbf{x},\mathbf{x}}\left(\mathbf{I} + \mathbf{D}\mathbf{K}_{\mathbf{x},\mathbf{x}}\right)^{-1}\left(\mathbf{D}\mathbf{y} + (\mathbf{c} - \boldsymbol{\sigma})\right) \tag{19.6.19}$$

Making predictions

Given a converged solution $\tilde{\mathbf{y}}$ we have found a Gaussian approximation

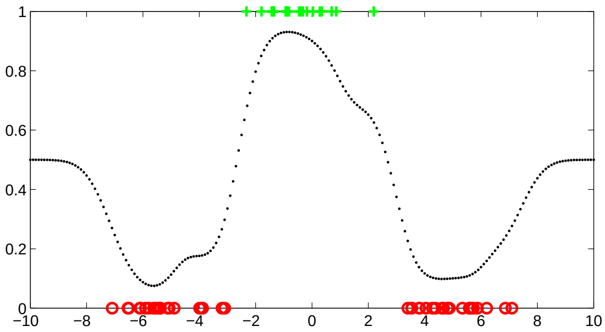
$$q(\mathbf{y}|\mathcal{X}, x^*, \mathcal{C}) = \mathcal{N}\left(\mathbf{y}\middle|\tilde{\mathbf{y}}, \left(\mathbf{K}_{\mathbf{x},\mathbf{x}}^{-1} + \mathbf{D}\right)^{-1}\right) \tag{19.6.20}$$

We now have Gaussians for $p(y^*|\mathbf{y})$ and $q(\mathbf{y}|\mathcal{X}, x^*, \mathcal{C})$ in equation (19.6.9). Predictions are then made using

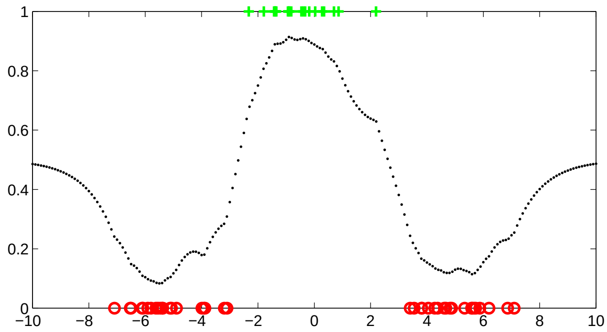
$$p(y^*|x^*, \mathcal{X}, \mathcal{C}) = \int p(y^*|x^*, \mathcal{X}, \mathbf{y})q(\mathbf{y}|\mathcal{X}, x^*, \mathcal{C})d\mathbf{y} \tag{19.6.21}$$

where, by conditioning, section(8.6.1),

$$p(y^*|\mathcal{Y}, x^*, \mathcal{X}) = \mathcal{N}\left(y^*\middle|\mathbf{K}_{x^*,\mathbf{x}}\mathbf{K}_{\mathbf{x},\mathbf{x}}^{-1}\mathbf{y}, \mathbf{K}_{x^*,x^*} - \mathbf{K}_{x^*,\mathbf{x}}\mathbf{K}_{\mathbf{x},\mathbf{x}}^{-1}\mathbf{K}_{\mathbf{x},x^*}\right) \tag{19.6.22}$$



(a)



(b)

Figure 19.6: Gaussian Process classification. The x-axis are the inputs, and the class is the y-axis. Green points are training points from class 1 and red from class 0. The dots are the predictions $p(c = 1|x^*)$ for a rand of points x^*. **(a)**: Square exponential covariance ($\gamma = 2$). **(b)**: OU covariance ($\gamma = 1$). See `demoGPclass1D.m`.

We can also write this as a linear system

$$y^* = \mathbf{K}_{x^*,\mathbf{x}}\mathbf{K}_{\mathbf{x},\mathbf{x}}^{-1}\mathbf{y} + \eta \tag{19.6.23}$$

where $\eta \sim \mathcal{N}\left(\eta|0, \mathbf{K}_{x^*,x^*} - \mathbf{K}_{x^*,\mathbf{x}}\mathbf{K}_{\mathbf{x},\mathbf{x}}^{-1}\mathbf{K}_{\mathbf{x},x^*}\right)$. Using equation (19.6.23) and equation (19.6.20) and averaging over $\mathbf{y}$ and the noise η, we obtain

$$\langle y^*|x^*, \mathcal{X}, \mathcal{C}\rangle \approx \mathbf{K}_{x^*,\mathbf{x}}\mathbf{K}_{\mathbf{x},\mathbf{x}}^{-1}\tilde{\mathbf{y}} = \mathbf{K}_{x^*,\mathbf{x}}\left(\mathbf{c} - \boldsymbol{\sigma}\left(\tilde{\mathbf{y}}\right)\right) \tag{19.6.24}$$

Similarly, the variance of the latent prediction is

$$\text{var}(y^*|x^*, \mathcal{X}, \mathcal{C}) \approx \mathbf{K}_{x^*,\mathbf{x}}\mathbf{K}_{\mathbf{x},\mathbf{x}}^{-1}\left(\mathbf{K}_{\mathbf{x},\mathbf{x}}^{-1} + \mathbf{D}\right)^{-1}\mathbf{K}_{\mathbf{x},\mathbf{x}}^{-1}\mathbf{K}_{\mathbf{x},x^*} + \mathbf{K}_{x^*,x^*} - \mathbf{K}_{x^*,\mathbf{x}}\mathbf{K}_{\mathbf{x},\mathbf{x}}^{-1}\mathbf{K}_{\mathbf{x},x^*} \tag{19.6.25}$$

$$= \mathbf{K}_{x^*,x^*} - \mathbf{K}_{x^*,\mathbf{x}}\left(\mathbf{K}_{\mathbf{x},\mathbf{x}} + \mathbf{D}^{-1}\right)^{-1}\mathbf{K}_{\mathbf{x},x^*} \tag{19.6.26}$$

where the last line is obtained using the Matrix Inversion Lemma, definition(**??**).

The class prediction for a new input x^* is then given by

$$p(c^* = 1|x^*, \mathcal{X}, \mathcal{C}) \approx \langle\sigma(y^*)\rangle_{\mathcal{N}(y^*|\langle y^*\rangle, \text{var}(y^*))} \tag{19.6.27}$$

In order to calculate the Gaussian integral over the logistic sigmoid function, we use an approximation of the sigmoid function based on the error function erf(x), see `avsigmaGauss.m`.

Example 85. An example of binary classification is given in fig(19.6) in which one-dimensional input training data with binary class labels is plotted along with the class probability predictions on a range of input points. In both cases the covariance function is of the form $2e^{|x_i - x_j|^\gamma} + 0.001\delta_{ij}$. The square exponential covariance produces a smoother class prediction than the Ornstein-Uhlenbeck covariance function. See `demoGPclass1D.m` and `demoGPclass.m`.

Marginal likelihood

The marginal likelihood is given by

$$p(\mathcal{C}|\mathcal{X}) = \int_{\mathcal{Y}} p(\mathcal{C}|\mathcal{Y})p(\mathcal{Y}|\mathcal{X}) = \int_{\mathbf{y}} e^{\Psi(\mathbf{y})} \tag{19.6.28}$$

Under the Laplace approximation, the marginal likelihood is approximated by

$$p(\mathcal{C}|\mathcal{X}) \approx \int_{\mathbf{y}} e^{\Psi(\tilde{\mathbf{y}})}e^{-\frac{1}{2}(\mathbf{y}-\tilde{\mathbf{y}})^{\mathsf{T}}\mathbf{A}(\mathbf{y}-\tilde{\mathbf{y}})} \tag{19.6.29}$$

where $\mathbf{A} = -\nabla\nabla\Psi$. Integrating, over $\mathbf{y}$ gives

$$\log p(\mathcal{C}|\mathcal{X}) \approx \log q(\mathcal{C}|\mathcal{X}) \tag{19.6.30}$$

where

$$\log q(\mathcal{C}|\mathcal{X}) = \Psi(\tilde{\mathbf{y}}) - \frac{1}{2}\log\det(2\pi\mathbf{A}) \tag{19.6.31}$$

$$= \Psi(\tilde{\mathbf{y}}) - \frac{1}{2}\log\det\left(\mathbf{K}_{\mathbf{x},\mathbf{x}}^{-1} + \mathbf{D}\right) + \frac{N}{2}\log 2\pi \tag{19.6.32}$$

$$= \mathbf{c}^{\mathsf{T}}\tilde{\mathbf{y}} - \sum_{n=1}^{N}\log(1+e^{\tilde{y}_n}) - \frac{1}{2}\tilde{\mathbf{y}}^{\mathsf{T}}\mathbf{K}_{\mathbf{x},\mathbf{x}}^{-1}\tilde{\mathbf{y}} - \frac{1}{2}\log\det(\mathbf{I} + \mathbf{K}_{\mathbf{x},\mathbf{x}}\mathbf{D}) \tag{19.6.33}$$

where $\tilde{\mathbf{y}}$ is the converged iterate of equation 19.6.18. One can also simplify the above using that at convergence $\mathbf{K}_{\mathbf{x},\mathbf{x}}^{-1}\tilde{\mathbf{y}} = \mathbf{c} - \boldsymbol{\sigma}(\mathbf{y})$.

The approximate marginal likelihood can be used to assess hyperparameters θ of the kernel. A little care is required in computing derivatives of the approximate marginal likelihood since the optimum $\tilde{\mathbf{y}}$ depends on θ. We use the total derivative formula [25]

$$\frac{d}{d\theta}\log q(\mathcal{C}|\mathcal{X}) = \frac{\partial}{\partial\theta}\log q(\mathcal{C}|\mathcal{X}) + \sum_i \frac{\partial}{\partial\tilde{y}_i}\log q(\mathcal{C}|\mathcal{X})\frac{d}{d\theta}\tilde{y}_i \tag{19.6.34}$$

$$\frac{\partial}{\partial\theta}\log q(\mathcal{C}|\mathcal{X}) = -\frac{1}{2}\frac{\partial}{\partial\theta}\left[\mathbf{y}^{\mathsf{T}}\mathbf{K}_{\mathbf{x},\mathbf{x}}^{-1}\mathbf{y} + \log\det(\mathbf{I} + \mathbf{K}_{\mathbf{x},\mathbf{x}}\mathbf{D})\right] \tag{19.6.35}$$

which can be evaluated using the standard results for the derivative of a matrix determinant and inverse. Since the derivative of Ψ is zero at $\tilde{\mathbf{y}}$, and noting that $\mathbf{D}$ depends explicitly on $\tilde{\mathbf{y}}$,

$$\frac{\partial}{\partial\tilde{y}_i}\log q(\mathcal{C}|\mathcal{X}) = -\frac{1}{2}\frac{\partial}{\partial\tilde{y}_i}\log\det(\mathbf{I} + \mathbf{K}_{\mathbf{x},\mathbf{x}}\mathbf{D}) \tag{19.6.36}$$

The implicit derivative is obtained from using the fact that at convergence

$$\tilde{\mathbf{y}} = \mathbf{K}_{\mathbf{x},\mathbf{x}}(\mathbf{c} - \boldsymbol{\sigma}(\mathbf{y})) \tag{19.6.37}$$

to give

$$\frac{d}{d\theta}\tilde{\mathbf{y}} = (\mathbf{I} + \mathbf{K}_{\mathbf{x},\mathbf{x}}\mathbf{D})^{-1}\frac{\partial}{\partial\theta}\mathbf{K}_{\mathbf{x},\mathbf{x}}(\mathbf{c} - \boldsymbol{\sigma}) \tag{19.6.38}$$

These results are substituted into equation (19.6.34) to find an explicit expression for the derivative.

19.6.3 Multiple classes

The extension of the preceding framework to multiple classes is essentially straightforward, using the softmax function

$$p(c = m|y) = \frac{e^{y_m}}{\sum_{m'} e^{y_{m'}}} \tag{19.6.39}$$

which automatically enforces the constraint $\sum_m p(c = m) = 1$. Naively it would appear that the for C classes, the cost of implementing the Laplace approximation for the multiclass case scales as $O\left(C^3N^3\right)$. However, one may show by careful implementation that the cost is only $O\left(CN^3\right)$, and we refer the reader to [294, 226] for details.

19.7 Further Reading

Gaussian Processes have been heavily developed within the machine learning community over recent years and efficient approximations for both regression and classification remains an open research topic. We direct the interested reader to [242] and [226] for further discussion.

19.8 Code

`GPreg.m`: Gaussian Process Regression
`demoGPreg.m`: Demo GP regression
`covfnGE.m`: Gamma-Exponential Covariance function
`GPclass.m`: Gaussian Process Classification
`demoGPclass.m`: Demo Gaussian Process Classification

19.9 Exercises

Exercise 191. *Show that the sample covariance matrix with elements* $S_{ij} = \sum_{n=1}^{N} x_i^n x_j^n / N - \bar{x}_i \bar{x}_j$, *where* $\bar{x}_i = \sum_{n=1}^{N} x_i^n / N$, *is positive (semi) definite.*

Exercise 192. *Show that*

$$k(x - x') = e^{-|\sin(x - x')|} \tag{19.9.1}$$

is a covariance function.

Exercise 193. *Consider the function*

$$f(x_i, x_j) = e^{-\frac{1}{2}(x_i - x_j)^2} \tag{19.9.2}$$

for one dimensional inputs x_i. *Show that*

$$f(x_i, x_j) = e^{-\frac{1}{2}x_i^2} e^{x_i x_j} e^{-\frac{1}{2}x_j^2} \tag{19.9.3}$$

By Taylor expanding the central term, show that $e^{-\frac{1}{2}(x_i - x_j)^2}$ *is a kernel and find an explicit representation for the kernel* $f(x_i, x_j)$ *as the scalar product of two infinite dimensional feature vectors.*

Exercise 194. *Show that for a covariance function* $k_1(\mathbf{x}, \mathbf{x}')$ *then*

$$k(\mathbf{x}, \mathbf{x}') = f\left(k_1\left(\mathbf{x}, \mathbf{x}'\right)\right) \tag{19.9.4}$$

is also a covariance function for any polynomial $f(x)$ *with positive coefficients. Show therefore that* $e^{k_1(\mathbf{x}, \mathbf{x}')}$ *and* $\tan(k_1(\mathbf{x}, \mathbf{x}'))$ *are covariance functions.*

Exercise 195 (String kernel)**.** *Let* x *and* x' *be two strings of characters and* $\phi_s(x)$ *be the number of times that substring* s *appears in string* x. *Then*

$$k(x, x') = \sum_s w_s \phi_s(x) \phi_s(x') \tag{19.9.5}$$

is a (string kernel) covariance function, provided the weight of each substring w_s *is positive.*

1. *Given a collection of strings about* politics *and another collection about* sport, *explain how to form a GP classifier using a string kernel.*

2. *Explain how the weights* w_s *can be adjusted to improve the fit of the classifier to the data and give an explicit formula for the derivative with respect to* w_s *of the log marginal likelihood under the Laplace approximation.*

Exercise 196 (Vector regression)**.** *Consider predicting a vector output* $\mathbf{y}$ *given training data* $\mathcal{X} \cup \mathcal{Y} = \{\mathbf{x}^n, \mathbf{y}^n, n = 1, \ldots, n\}$. *To make a GP predictor*

$$p(\mathbf{y}^* | \mathbf{x}^*, \mathcal{X}, \mathcal{Y}) \tag{19.9.6}$$

we need a Gaussian model

$$p(\mathbf{y}^1, \ldots, \mathbf{y}^N, \mathbf{y}^* | \mathbf{x}^1, \ldots, \mathbf{x}^n, \mathbf{x}^*) \tag{19.9.7}$$

A GP requires then a specification of the covariance $c(y_i^m, y_j^n|\mathbf{x}^n, \mathbf{x}^m)$ of the components of the outputs for two different input vectors. Show that under the dimension independence assumption

$$c(y_i^m, y_j^n|\mathbf{x}^n, \mathbf{x}^m) = c_i(y_i^m, y_i^n|\mathbf{x}^n, \mathbf{x}^m)\delta_{ij} \tag{19.9.8}$$

where $c_i(y_i^m, y_i^n|\mathbf{x}^n, \mathbf{x}^m)$ is a covariance function for the i^{th} dimension, that separate GP predictors can be constructed independently, one for each output dimension i.

Exercise 197. *Consider the Markov update of a linear dynamical system, section(24.1),*

$$\mathbf{x}_t = \mathbf{A}\mathbf{x}_{t-1} + \boldsymbol{\eta}_t, \qquad t \geq 2 \tag{19.9.9}$$

where $\mathbf{A}$ is a given matrix and $\boldsymbol{\eta}_t$ is zero mean Gaussian noise with covariance $\left\langle \eta_{i,t}\eta_{j,t'} \right\rangle = \sigma^2\delta_{ij}\delta_{t,t'}$. Also, $p(\mathbf{x}_1) = \mathcal{N}(\mathbf{x}_1|\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

1. *Show that $\mathbf{x}_1, \ldots, \mathbf{x}_t$ is Gaussian distributed.*

2. *Show that the covariance matrix of $\mathbf{x}_1, \ldots, \mathbf{x}_t$ has elements*

$$\left\langle \mathbf{x}_{t'}\mathbf{x}_t^{\mathsf{T}} \right\rangle = \begin{cases} \mathbf{A}^{t'-t}\boldsymbol{\Sigma} & t \neq t' \\ \mathbf{A}^t\boldsymbol{\Sigma}\left(\mathbf{A}^t\right)^{\mathsf{T}} & t = t' \end{cases} \tag{19.9.10}$$

   and explain why a linear dynamical system is a (constrained) Gaussian Process.

3. *Consider*

$$\mathbf{y}_t = \mathbf{B}\mathbf{x}_t + \boldsymbol{\epsilon}_t \tag{19.9.11}$$

   where $\boldsymbol{\epsilon}_t$ is zero mean Gaussian noise with covariance $\left\langle \epsilon_{i,t}\epsilon_{j,t'} \right\rangle = \nu^2\delta_{ij}\delta_{t,t'}$. The vectors $\boldsymbol{\epsilon}$ are uncorrelated with the vectors $\boldsymbol{\eta}$. Show that the sequence of vectors $\mathbf{y}_1, \ldots, \mathbf{y}_t$ is a Gaussian Process with a suitably defined covariance function.

Exercise 198. *A form of independent components analysis, section(21.6), of a one-dimensional signal $y_1, \ldots, y_T$ is obtained from the joint model*

$$p(y_{1:T}, x^1_{1:T}, x^2_{1:T}|\mathbf{w}) = \left\{\prod_t p(y_t|x_t^1, x_t^2, \mathbf{w})\right\}\mathcal{N}\left(x^1_{1:T}|\mathbf{0}, \boldsymbol{\Sigma}\right)\mathcal{N}\left(x^2_{1:T}|\mathbf{0}, \boldsymbol{\Sigma}\right) \tag{19.9.12}$$

with

$$p(y_t|x_t^1, x_t^2, \mathbf{w}) = \mathcal{N}\left(y_t|w^1x_t^1 + w^2x_t^2, \nu^2\right) \tag{19.9.13}$$

where ν^2 is a given noise variance. The signal $y_{1:T}$ can be viewed as the linear combination of two independent Gaussian Processes. The covariance matrices of the two processes have elements from a stationary kernel,

$$\Sigma_{t,t'} = e^{-(t-t')^2} \tag{19.9.14}$$

1. *Write down an EM algorithm for learning the mixing parameters w^1, w^2 given an observation sequence $y_{1:T}$.*

2. *Consider an extension of the above model to the case of two outputs:*

$$p(y^1_{1:T}, y^2_{1:T}, x^1_{1:T}, x^2_{1:T}|\mathbf{W}) = \prod_{i=1}^2\left\{\prod_t p(y_t^i|x_t^1, x_t^2, \mathbf{w}_i)\right\}\mathcal{N}\left(x^1_{1:T}|\mathbf{0}, \boldsymbol{\Sigma}\right)\mathcal{N}\left(x^2_{1:T}|\mathbf{0}, \boldsymbol{\Sigma}\right) \tag{19.9.15}$$

   with

$$p(y_t^1|x_t^1, x_t^2, \mathbf{W}) = \mathcal{N}\left(y_t^1|w_{11}x_t^1 + w_{12}x_t^2, \nu^2\right) \tag{19.9.16}$$
$$p(y_t^2|x_t^1, x_t^2, \mathbf{W}) = \mathcal{N}\left(y_t^1|w_{21}x_t^1 + w_{22}x_t^2, \nu^2\right) \tag{19.9.17}$$

   Show that for $T > 1$ the likelihood

$$p(y^1_{1:T}, y^2_{1:T}|\mathbf{W}) \tag{19.9.18}$$

   is not invariant with respect to an orthogonal rotation $\mathbf{W}' = \mathbf{W}\mathbf{R}$, with $\mathbf{R}\mathbf{R}^{\mathsf{T}} = \mathbf{I}$, and explain the significance of this with respect to identifying independent components.

Exercise 199. *For a covariance function*

$$k_1(\mathbf{x}, \mathbf{x}') = f(\left(\mathbf{x} - \mathbf{x}'\right)^{\mathsf{T}} \left(\mathbf{x} - \mathbf{x}'\right)) \tag{19.9.19}$$

show that

$$k_2(\mathbf{x}, \mathbf{x}') = f(\left(\mathbf{x} - \mathbf{x}'\right)^{\mathsf{T}} \mathbf{A} \left(\mathbf{x} - \mathbf{x}'\right)) \tag{19.9.20}$$

is also a valid covariance function for a positive definite symmetric matrix $\mathbf{A}$.

CHAPTER 20

Mixture Models

20.1 Density estimation using mixtures

A mixture model is one in which a set of component models is combined to produce a richer model:

$$p(v) = \sum_{h=1}^{H} p(v|h)p(h) \tag{20.1.1}$$

Here the variable v denotes the 'visible' or 'observable' variables, and $h = 1, \ldots, H$ indexes each component model $p(v|h)$, and its weight $p(h)$. Mixture models have natural application in clustering data, where h indexes the cluster. This interpretation can be gained from considering how to generate a sample datapoint v from the model equation (20.1.1). First we sample a cluster h from $p(h)$, and then draw a visible state v from $p(v|h)$.

For a set of data $v^1, \ldots, v^N$, assuming i.i.d. data, a mixture model is of the form, fig(20.1),

$$p(v^1, \ldots, v^N|\theta) = \prod_{n=1}^{N} \sum_{h^n} p(v^n|h^n, \theta_{v|h})p(h^n|\theta_h) \tag{20.1.2}$$

Clustering is achieved by inference of

$$\underset{h^1,\ldots,h^N}{\text{argmax}}\ p(h^1, \ldots, h^N|v^1, \ldots, v^N) \tag{20.1.3}$$

which, thanks to the factorised form of the distribution is equivalent to computing $\underset{h^n}{\text{argmax}}\ p(h^n|v^n)$ for each datapoint. In this way we can cluster many kinds of data for which a 'distance' measure in the sense of K-means is not directly apparent.

The optimal parameters $\theta_{v|h}, \theta_h$ of a mixture model can be set by Maximum Likelihood,

$$\theta_{opt} = \underset{\theta}{\text{argmax}}\ p(v^1, \ldots, v^N|\theta) \tag{20.1.4}$$

Numerically, this can be achieved using an optimisation procedure such as gradient based approaches. Alternatively, by treating the component indices as latent variables, one may also apply the EM algorithm, as described in the following section, which in many classical cases produces simple update formulae.

Example 86. The data in fig(20.2) naturally has two clusters and can be modelled with a mixture of two two-dimensional Gaussians, each Gaussian describing one of the clusters. Here there is a clear visual interpretation of the meaning of 'cluster', with the mixture model placing two datapoints in the same cluster if they are both likely to be generated by the same model component.

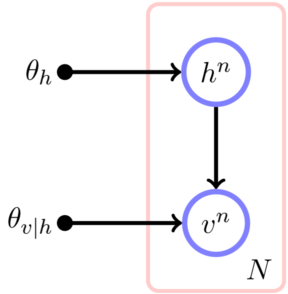


Figure 20.1: A mixture model has a trivial graphical representation as a DAG with a single hidden node, which can be in and one of H states, $i = 1 \ldots H$. Maximum likelihood learning assuming i.i.d. means that there is a parameter.

20.2 Expectation Maximisation for Mixture Models

By treating the index h as a missing variable, mixture models can be trained using the EM algorithm, section(11.2). There are two sets of parameters – those of each component model $p(v|h, \theta_{v|h})$ and the mixture weights $p(h|\theta_h)$. According to the general approach for i.i.d. data of section(11.2), we need to consider the energy term:

$$E(\theta) = \sum_{n=1}^{N} \langle \log p(v^n, h|\theta)\rangle_{p^{old}(h|v^n)} \tag{20.2.1}$$

$$= \sum_{n=1}^{N} \left\langle \log p(v^n|h, \theta_{v|h})\right\rangle_{p^{old}(h|v^n)} + \sum_{n=1}^{N} \langle \log p(h|\theta_h)\rangle_{p^{old}(h|v^n)} \tag{20.2.2}$$

where

$$p^{old}(h|v^n) \propto p(v^n|h, \theta^{old}_{v|h})p(h|\theta^{old}_h) \tag{20.2.3}$$

and maximise (20.2.2) with respect to the parameters $\theta_{v|h}$, θ_h.

20.2.1 Unconstrained discrete tables

Here we consider training a Belief Network $p(v|h)p(h)$ in which the tables are unconstrained. This is a special case of the more general framework discussed in section(11.2).

M-step: $p(h)$

If no constraint is placed on $p(h|\theta_h)$ we may write the parameters as simply $p(h)$, with the understanding that $0 \le p(h) \le 1$ and $\sum_h p(h) = 1$. Isolating the dependence of equation (20.2.2) on $p(h)$ we obtain

$$\sum_{n=1}^{N} \langle \log p(h)\rangle_{p^{old}(h|v^n)} = \sum_h \log p(h) \sum_{n=1}^{N} p^{old}(h|v^n) \tag{20.2.4}$$

It is standard to treat the maximisation problem using Lagrange multipliers, see exercise(204). Here we take an alternative approach based on recognising the similarity of the above to a form of Kullback-Leibler divergence. First we define the distribution

$$\tilde{p}(h) \equiv \frac{\sum_{n=1}^{N} p^{old}(h|v^n)}{\sum_h \sum_{n=1}^{N} p^{old}(h|v^n)} \tag{20.2.5}$$

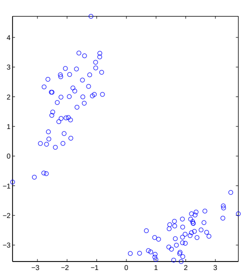


Figure 20.2: Two dimensional data which displays clusters. In this case a Gaussian mixture model $1/2\mathcal{N}(\mathbf{x}|\mathbf{m}_1, \mathbf{C}_1) + 1/2\mathcal{N}(\mathbf{x}|\mathbf{m}_2, \mathbf{C}_2)$ would fit the data well for suitable means $\mathbf{m}_1, \mathbf{m}_2$ and covariances $\mathbf{C}_1, \mathbf{C}_2$.

Then maximising equation (20.2.4) is equivalent to maximising

$$\langle \log p(h) \rangle_{\tilde{p}(h)} \tag{20.2.6}$$

since the two expressions are related by the constant factor $\sum_h \sum_{n=1}^N p^{old}(h|v^n)$. By subtracting the constant term $\langle \log \tilde{p}(h) \rangle_{\tilde{p}(h)}$ from equation (20.2.6), we obtain the negative Kullback-Leibler divergence $\mathrm{KL}(\tilde{p}|p)$. This means that the optimal $p(h)$ is that distribution which minimises the Kullback-Leibler divergence. Optimally, therefore $\tilde{p}(h) = p(h)$, so that

$$p^{new}(h) = \frac{\sum_{n=1}^N p^{old}(h|v^n)}{\sum_h \sum_{n=1}^N p^{old}(h|v^n)} = \frac{1}{N} \sum_{n=1}^N p^{old}(h|v^n) \tag{20.2.7}$$

M-step : $p(v|h)$

The dependence of equation (20.2.2) on $p(v|h)$ is

$$\sum_{n=1}^N \left\langle \log p\left(v^n|h, \theta_{v|h}\right) \right\rangle_{p^{old}(h|v^n)} \tag{20.2.8}$$

If the distributions $p\left(v|h, \theta_{v|h}\right)$ are not constrained, we can apply a similar Kullback-Leibler method, as we did in section(11.2). For didactical purposes only, here we demonstrate the more standard Lagrange procedure. We need to ensure that $p(v|h)$ is a distribution for each of the mixture states $h = 1, \ldots, H$. This can be achieved using a set of Lagrange multipliers, giving the Lagrangian:

$$\mathcal{L} \equiv \sum_v \sum_{n=1}^N \sum_{h=1}^H \mathbb{I}\left[v^n = v\right] p^{old}(h|v^n) \log p\left(v|h\right) + \sum_{h=1}^H \lambda(h) \left(1 - \sum_{v=1}^V p(v|h)\right) \tag{20.2.9}$$

Here we assumed a discrete random variable v. The case of continuous a continuous v is analogous on replacing summation over the states of v with integration. Differentiating with respect to $p(v = i|h = j)$ and equating to zero, $p^{new} = \underset{p}{\operatorname{argmax}}\, \mathcal{L}$ is given by

$$p^{new}(v = i|h = j) \propto \sum_{n=1}^N \mathbb{I}\left[v^n = i\right] p^{old}(v = i|h = j) \tag{20.2.10}$$

which, using the normalisation requirement, gives

$$p^{new}(v = i|h = j) = \frac{\sum_{n=1}^N \mathbb{I}\left[v^n = i\right] p^{old}(h = j|v = i)}{\sum_{i=1}^V \sum_{n=1}^N \mathbb{I}\left[v^n = i\right] p^{old}(h = j|v = i)} \tag{20.2.11}$$

E-step

According to the general EM procedure, section(11.2), optimally we set $p^{new}(h|v^n) = p(h|v^n)$:

$$p^{new}(h|v^n) = \frac{p(v^n|h)p(h)}{\sum_h p(v^n|h)p(h)} \tag{20.2.12}$$

Equations (20.2.7,20.2.11,20.2.12) are repeated until convergence. The initialisation of the tables and mixture probabilities can severely affect the quality of the solution found since the likelihood often has local optima. If random initialisations are used, it is recommended to record the value of the likelihood itself, to see which converged parameters have the higher likelihood. The solution with the highest likelihood is to be preferred.

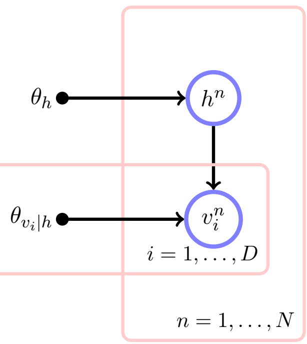


Figure 20.3: Mixture of a product of Bernoulli distributions. In a Bayesian Treatment, a parameter prior is used. In the text we simply set the parameters using Maximum Likelihood.

20.2.2 Mixture of product of Bernoulli distributions

As an example application of mixture models for clustering, we consider a simple model of binary attribute vectors. The mixture of Bernoulli products[1] model is given by

$$p(v_1, \ldots, v_D) = \sum_{h=1}^{H} p(h) \prod_{i=1}^{D} p(v_i|h) \tag{20.2.13}$$

where each term $p(v_i|h)$ is a Bernoulli distribution – that is, there are two states $v_i \in \{0, 1\}$. The model is depicted in fig(20.3) and has parameters $p(h)$ and $p(v_i = 1|h)$, $h = 1 \ldots, H$.

EM training

To train the model under Maximum Likelihood it is convenient to use the EM algorithm which, as usual, may be derived by writing down the energy:

$$\sum_n \langle \log p(v^n, h) \rangle_{p^{old}(h|v^n)} = \sum_n \sum_i \langle \log p(v_i^n|h) \rangle_{p^{old}(h|v^n)} + \sum_n \langle \log p(h) \rangle_{p^{old}(h|v^n)} \tag{20.2.14}$$

and then performing the maximisation over the table entries. From our general results we may immediately jump to the updates. The M-step is given by

$$\begin{aligned} p^{new}(v_i = 1|h = j) &= \frac{\sum_n \mathbb{I}[v_i^n = 1]\, p^{old}(h = j|v^n)}{\sum_n \mathbb{I}[v_i^n = 1]\, p^{old}(h = j|v^n) + \sum_n \mathbb{I}[v_i^n = 0]\, p^{old}(h = j|v^n)} \\ p^{new}(h = j) &= \frac{\sum_n p^{old}(h = j|v^n)}{\sum_{h'} \sum_n p^{old}(h'|v^n)} \end{aligned} \tag{20.2.15}$$

and the E-step by

$$p^{old}(h = j|v^n) \propto p(h = j) \prod_{i=1}^{D} p(v_i^n|h = j) \tag{20.2.16}$$

Equations (20.2.15,20.2.16) are iterated until convergence.

If an attribute i is missing for datapoint n, one needs to sum over the states of the corresponding v_i^n. The effect of performing the summation for this model is simply to remove the corresponding factor $p(v_i^n|h)$ from the algorithm, see exercise(200).

Initialisation

The EM algorithm can be very sensitive to initial conditions. Consider the following initialisation: $p(v_i = 1|h = j) = 0.5$, with $p(h)$ set arbitrarily. This means that at the first iteration, $p^{old}(h = j|v^n) = p(h = j)$. The subsequent M-step updates are

$$p^{new}(h) = p(h), \qquad p^{new}(v_i|h = j) = p^{new}(v_i|h = j') \tag{20.2.17}$$

[1] This is similar to the Naive Bayes classifier in which the class labels are always hidden.

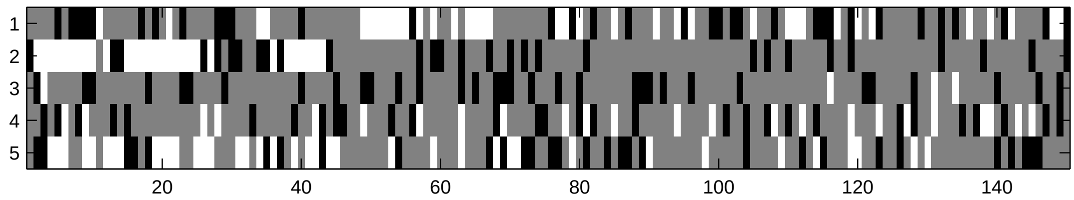


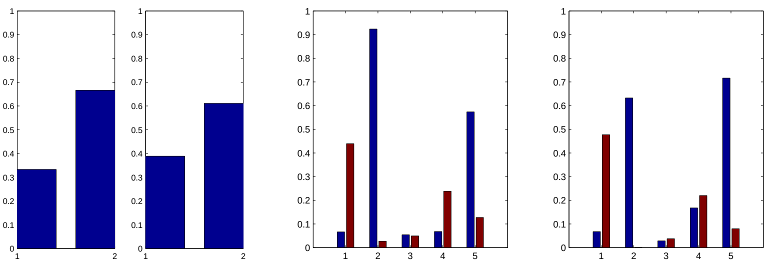
Figure 20.4: Data from questionnaire responses. 150 people were each asked 5 questions, with 'yes' (white) and 'no' (gray) answers. Black denotes that the absence of a response (missing data). This training data was generated by two component Binomial mixture. Missing data was simulated by randomly removing values from the dataset.

for any j, j'. This means that the parameters $p(v|h)$ immediately become independent of h and the model is numerically trapped in a symmetric solution. It makes sense, therefore, to initialise the parameters in a non-symmetric fashion.

Example 87 (Questionnaire)**.** A company sends out a questionnaire to each customer, containing a set of D 'yes/no' questions. The binary responses of a customer are stored in a vector $\mathbf{v} = (v_1, \ldots, v_D)^\mathsf{T}$. In total N customers send back their questionnaires, $\mathbf{v}^1, \ldots, \mathbf{v}^N$, and the company wishes to perform an analysis to find what kinds of customers it has. The company assumes there are H essential types of customer for which the profile of responses is defined by only the customer type.

Data from a questionnaire containing 5 questions, with 150 respondents is presented in fig(20.4). The data has a large number of missing values. We assume there are $H = 2$ kinds of respondents and attempt to assign each respondent into one of the two clusters. Running the EM algorithm on this data, with random initial values for the tables, produces the results in fig(20.5). Based on assigning the each datapoint v^n to the cluster with maximal posterior probability $h^n = \arg\max_h p(h|v^n)$, given a trained model $p(v|h)p(h)$, the model assigns 86% of the data to the correct cluster (which is known in this simulated case). See fig(20.5) and `MIXprodBern.m`.

Example 88 (Handwritten digits)**.** A collection of 5000 handwritten digits is given, which we wish to cluster into 20 groups, fig(20.6). Each digits is a $28 \times 28 = 784$ dimensional binary vector. Using a mixture of Bernoulli products, trained with 50 iterations of EM (with a random perturbation of the mean of the data used as initialisation), the clusters are presented in fig(20.6). As we see, the method roughly captures natural clusters in the data – for example, there are two kinds of 1, one slightly more slanted than the other, similarly, two kinds of 4, *etc.*



(a) (b)

Figure 20.5: EM learning of a mixture of Bernoulli products. **(a)**: True $p(h)$ (left) and learned $p(h)$ (right) for $h = 1, 2$. **(b)**: True $p(v|h)$ (left) and learned $p(v|h)$ (right) for $v = 1, \ldots, 5$. Each column pair corresponds to $p(v_i|h = 1)$ (red) and $p(v_i|h = 2)$ (blue) with $i = 1, \ldots, 5$. The learned probabilities are reasonably close to the true values.

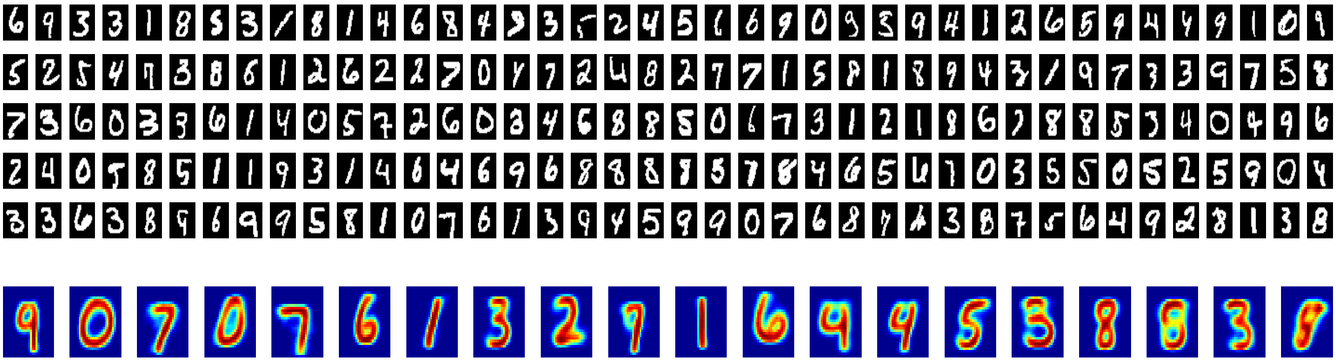


Figure 20.6: Top: a selection of 200 of the 5000 handwritten digits in the training set. Bottom: the trained cluster outputs $p(v_i = 1|h)$ for $h = 1, \ldots, 20$ mixtures. See `demoMixBernoulliDigits.m`.

20.3 The Gaussian Mixture Model

Gaussians are particularly convenient continuous mixture components since they constitute 'bumps' of probability mass, aiding an intuitive interpretation of the model. As a reminder, a D dimensional Gaussian distribution for a continuous variable $\mathbf{x}$ is

$$p(\mathbf{x}|\mathbf{m}, \mathbf{S}) = \frac{1}{\sqrt{\det(2\pi\mathbf{S})}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\mathbf{m})^{\mathsf{T}} \mathbf{S}^{-1}(\mathbf{x}-\mathbf{m})\right\} \tag{20.3.1}$$

where $\mathbf{m}$ is the mean and $\mathbf{S}$ is the covariance matrix. A mixture of Gaussians is then

$$p(\mathbf{x}) = \sum_{i=1}^{H} p(\mathbf{x}|\mathbf{m}_i, \mathbf{S}_i)p(h=i) \tag{20.3.2}$$

For a set of data $\mathcal{X} = \left\{\mathbf{x}^1, \ldots, \mathbf{x}^N\right\}$ and under the usual i.i.d. assumption, the log likelihood is

$$\log p(\mathcal{X}|\theta) = \sum_{n=1}^{N} \log \sum_{i=1}^{H} p(h=i) \frac{1}{\sqrt{\det(2\pi\mathbf{S}_i)}} \exp\left\{-\frac{1}{2}(\mathbf{x}^n-\mathbf{m}_i)^{\mathsf{T}} \mathbf{S}_i^{-1}(\mathbf{x}^n-\mathbf{m}_i)\right\} \tag{20.3.3}$$

where the parameters are $\theta = \{\mathbf{m}_i, \mathbf{S}_i, p(h=i), i = 1, \ldots, H\}$. The optimal parameters θ can be set using Maximum Likelihood, bearing in mind the constraint that the $\mathbf{S}_i$ must be symmetric positive definite matrices, in addition to $0 \leq p(h=i) \leq 1$, $\sum_i p(h=i) = 1$. Gradient based optimisation approaches are feasible under a parameterisation of the $\mathbf{S}_i$ (*e.g.* Cholesky decomposition) and $p(h=i)$ (*e.g.* softmax) that enforce the constraints. An alternative is the EM approach which in this case is particularly convenient since it automatically provides parameter updates that ensure these constraints.

20.3.1 EM algorithm

The energy term is

$$\sum_{n=1}^{N} \langle \log p(\mathbf{x}^n, h)\rangle_{p^{old}(h|\mathbf{x}^n)} = \sum_{n=1}^{N} \langle \log \left[p(\mathbf{x}^n|h)p(h)\right]\rangle_{p^{old}(h|\mathbf{x}^n)} \tag{20.3.4}$$

Plugging in the definition of the Gaussian components, we have

$$\sum_{i=1}^{H} p^{old}(i|\mathbf{x}^n) \sum_{n=1}^{N} \left\{-\frac{1}{2}(\mathbf{x}^n-\mathbf{m}_i)^{\mathsf{T}} \mathbf{S}_i^{-1}(\mathbf{x}^n-\mathbf{m}_i) - \frac{1}{2}\log\det(2\pi\mathbf{S}_i) + \log p(h=i)\right\} \tag{20.3.5}$$

The M-step requires the maximisation of the above with respect to $\mathbf{m}_i, \mathbf{S}_i, p(h=i)$.

M-step : optimal $\mathbf{m}_i$

Maximising equation (20.3.5) with respect to $\mathbf{m}_i$ is equivalent to minimising

$$\sum_{n=1}^{N}\sum_{i=1}^{H} p^{old}(i|\mathbf{x}^n)\left(\mathbf{x}^n - \mathbf{m}_i\right)^{\mathsf{T}} \mathbf{S}_i^{-1}\left(\mathbf{x}^n - \mathbf{m}_i\right) \tag{20.3.6}$$

Differentiating with respect to $\mathbf{m}_i$ and equating to zero we have

$$2\sum_{n=1}^{N} p^{old}(i|\mathbf{x}^n)\mathbf{S}_i^{-1}\left(\mathbf{x}^n - \mathbf{m}_i\right) = \mathbf{0} \tag{20.3.7}$$

Hence, optimally,

$$\mathbf{m}_i = \frac{\sum_{n=1}^{N} p^{old}(i|\mathbf{x}^n)\mathbf{x}^n}{\sum_{n=1}^{N} p^{old}(i|\mathbf{x}^n)} \tag{20.3.8}$$

By defining the membership distribution

$$p^{old}(n|i) \equiv \frac{p^{old}(i|\mathbf{x}^n)}{\sum_{n=1}^{N} p^{old}(i|\mathbf{x}^n)} \tag{20.3.9}$$

which quantifies the membership of datapoints to cluster i, we can write equation (20.3.8) more compactly as

$$\mathbf{m}_i = \sum_{n=1}^{N} p^{old}(n|i)\mathbf{x}^n \tag{20.3.10}$$

M-step : optimal $\mathbf{S}_i$

Optimising equation (20.3.5) with respect to $\mathbf{S}_i$ is equivalent to minimising

$$\sum_{n=1}^{N}\left\langle (\Delta_i^n)^{\mathsf{T}}\mathbf{S}_i^{-1}\Delta_i^n - \log\det\left(\mathbf{S}_i^{-1}\right)\right\rangle_{p^{old}(i|\mathbf{x}^n)} \tag{20.3.11}$$

where $\Delta_i^n \equiv \mathbf{x}^n - \mathbf{m}_i$. To aid the matrix calculus, we isolate the dependency on $\mathbf{S}_i$ to give

$$\text{trace}\left(\mathbf{S}_i^{-1}\sum_{n=1}^{N} p^{old}(i|\mathbf{x}^n)\Delta_i^n\left(\Delta_i^n\right)^{\mathsf{T}}\right) - \log\det\left(\mathbf{S}_i^{-1}\right)\sum_{n=1}^{N} p^{old}(i|\mathbf{x}^n) \tag{20.3.12}$$

Differentiating with respect to $\mathbf{S}_i^{-1}$ and equating to zero, we obtain

$$\sum_{n=1}^{N} p^{old}(i|\mathbf{x}^n)\Delta_i^n\left(\Delta_i^n\right)^{\mathsf{T}} - \mathbf{S}_i\sum_{n=1}^{N} p^{old}(i|\mathbf{x}^n) = \mathbf{0} \tag{20.3.13}$$

Using $p^{old}(n|i)$, the optimal $\mathbf{S}_i$ is given by

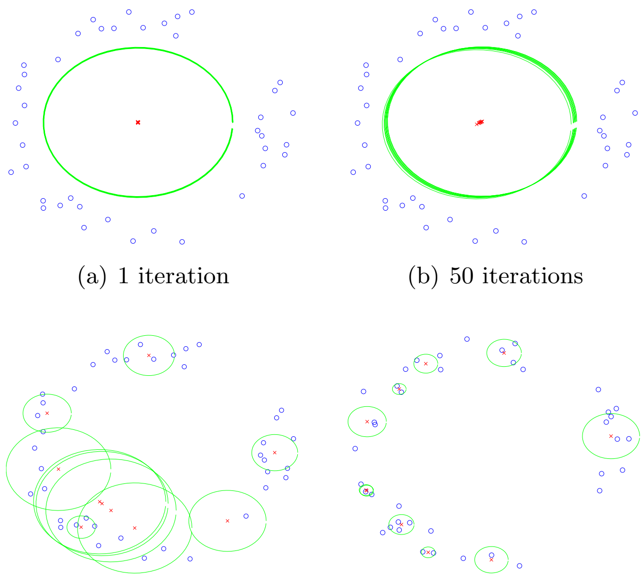
$$\mathbf{S}_i = \sum_{n=1}^{N} p^{old}(n|i)\left(\mathbf{x}^n - \mathbf{m}_i\right)\left(\mathbf{x}^n - \mathbf{m}_i\right)^{\mathsf{T}} \tag{20.3.14}$$

This ensures that $\mathbf{S}_i$ is symmetric positive semi-definite. A special case is to constrain the covariances $\mathbf{S}_i$ to be diagonal for which the update is, see exercise(202),

$$\mathbf{S}_i = \sum_{n=1}^{N} p^{old}(n|i)\text{diag}\left(\left(\mathbf{x}^n - \mathbf{m}_i\right)\left(\mathbf{x}^n - \mathbf{m}_i\right)^{\mathsf{T}}\right) \tag{20.3.15}$$

where above $\text{diag}(\mathbf{M})$ means forming a new matrix from the matrix $\mathbf{M}$ with zero entries except for the diagonal entries of $\mathbf{M}$. A more extreme case is that of *isotropic* Gaussians $\mathbf{S}_i = \sigma_i^2\mathbf{I}$. The reader may show that the optimal update for σ_i^2 in this case is given by taking the average of the diagonal entries of the diagonally constrained covariance update,

$$\sigma_i^2 = \frac{1}{\dim \mathbf{x}}\sum_{k=1}^{\dim \mathbf{x}}\sum_{n=1}^{N} p^{old}(n|i)\left(\mathbf{x}_k^n - \mathbf{m}_{i,k}\right)^2 \tag{20.3.16}$$



(a) 1 iteration (b) 50 iterations (c) 125 iterations (d) 150 iterations

Figure 20.7: Training a mixture of 10 isotropic Gaussians **(a)**: If we start with large variances for the Gaussians, even after one iteration, the Gaussians are centred close to the mean of the data. **(b)**: The Gaussians begin to separate **(c)**: One by one, the Gaussians move towards appropriate parts of the data **(d)**: The final converged solution. The Gaussians are constrained to have variances greater than a set amount. See `demoGMMem.m`.

M-step : optimal mixture coefficients

If no constraint is placed on the weights, the update follows the general formula given in equation (20.2.7),

$$p(h=i) = \frac{1}{N}\sum_{n=1}^{N} p^{old}(i|\mathbf{x}^n) \tag{20.3.17}$$

E-step

$$p^{old}(i|\mathbf{x}^n) \propto p^{old}(\mathbf{x}^n|i)p(h=i) \tag{20.3.18}$$

Explicitly, this is given by the *responsibility*

$$p^{old}(i|\mathbf{x}^n) = \frac{p(h=i)\exp\left\{-\frac{1}{2}(\mathbf{x}^n-\mathbf{m}_i)^{\mathsf{T}}\,\mathbf{S}_i^{-1}\,(\mathbf{x}^n-\mathbf{m}_i)\right\}}{\sum_{i'} p(h=i')\exp\left\{-\frac{1}{2}(\mathbf{x}^n-\mathbf{m}_{i'})^{\mathsf{T}}\,\mathbf{S}_{i'}^{-1}\,(\mathbf{x}^n-\mathbf{m}_{i'})\right\}} \tag{20.3.19}$$

The above equations (20.3.8,20.3.14,20.3.17,20.3.19) are iterated until convergence.

The performance of EM for Gaussian mixtures can be strongly dependent on the initialisation, which we discuss below. In addition, constraints on the covariance matrix are required in order to find sensible solutions.

20.3.2 Practical issues

Infinite troubles

A difficulty arises with using Maximum Likelihood to fit a Gaussian mixture model. Consider placing a component $p(\mathbf{x}|\mathbf{m}_i, \mathbf{S}_i)$ with mean $\mathbf{m}_i$ set to one of the datapoints $\mathbf{m}_i = \mathbf{x}^n$. The contribution from that Gaussian will be

$$p(\mathbf{x}^n|\mathbf{m}_i, \mathbf{S}_i) = \frac{1}{\sqrt{\det(2\pi\mathbf{S}_i)}} e^{-\frac{1}{2}(\mathbf{x}^n-\mathbf{x}^n)^{\mathsf{T}}\mathbf{S_i}^{-1}(\mathbf{x}^n-\mathbf{x}^n)} = \frac{1}{\sqrt{\det(2\pi\mathbf{S}_i)}} \tag{20.3.20}$$

In the limit that the 'width' of the covariance goes to zero (the eigenvalues of $\mathbf{S}_i$ tend to zero), this probability density becomes infinite. This means that one can obtain a Maximum Likelihood solution by placing zero-width Gaussians on a selection of the datapoints, resulting in an infinite likelihood. This is clearly undesirable and arises because, in this case, the Maximum Likelihood solution does not constrain the parameters in a sensible way. Note that this is not related to the EM algorithm, but a property of the

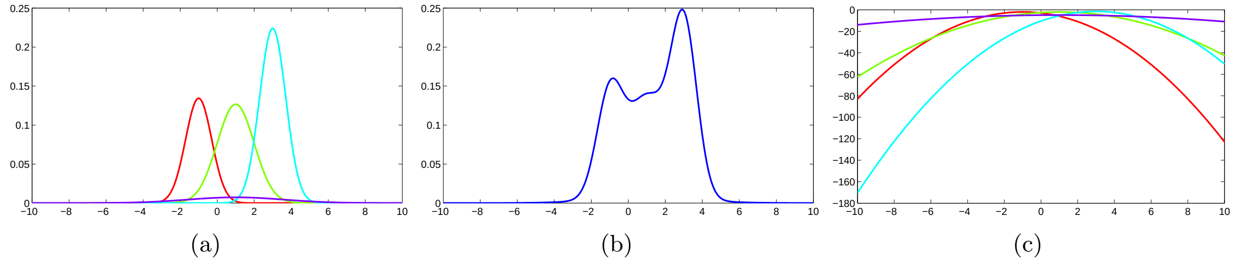


Figure 20.8: **(a)**: A Gaussian mixture model with $H = 4$ components. There is a component (purple) with large variance and small weight that has little effect on the distribution close to where the other three components have appreciable mass. As we move further away this additional component gains in influence. **(b)**: The GMM probability density function from (a). **(c)**: Plotted on a log scale, the influence of each Gaussian far from the origin becomes clearer.

Maximum Likelihood method itself. All computational methods which aim to fit unconstrained mixtures of Gaussians using Maximum Likelihood therefore succeed in finding 'reasonable' solutions merely by getting trapped in serendipitous local maxima.

A remedy is to include an additional constraint on the width of the Gaussians, ensuring that they cannot become too small. One approach is to monitor the eigenvalues of each covariance matrix and if an update would result in a new eigenvalue smaller than a desired threshold, the update is rejected. In `GMMem.m` we use a similar approach in which we constrain the determinant (the product of the eigenvalues) of the covariances to be greater than a desired specified minimum value.
One can view the formal failure of Maximum Likelihood in the case of Gaussian mixtures as a result of an inappropriate prior. The Maximum Likelihood is equivalent to MAP in which a flat prior is placed on each matrix $\mathbf{S}_i$. This is unreasonable since the matrices are required to be positive definite and of non-vanishing width. A Bayesian solution to this problem is possible, placing a priori on covariance matrices. The natural prior in this case is the Wishart Distribution, or a Gamma distribution in the case of a diagonal covariance.

Initialisation

A useful intialisation strategy is to set the covariances to be diagonal with large variances. This gives the components a chance to 'sense' where data lies. An illustration of the performance of the algorithm is given in fig(20.7).

Symmetry Breaking

If the covariances are initialised to large values, the EM algorithm appears to make little progress in the first iterations as each component jostles with the others to try to explain the data. Eventually one Gaussian component breaks away and takes responsibility for explaining the data in its vicinity, see fig(20.7).

The origin of this jostling is an inherent symmetry in the solution – it makes no difference to the likelihood if we relabel what the components are called. This permutation symmetry causes initial confusion as to which model should explain which parts of the data. Eventually, this symmetry is broken, and a local solution is found.

The symmetries can severely handicap EM in fitting a large number of models in the mixture since the number of permutations increases dramatically with the number of components. A heuristic is to begin with a small number of components, say two, for which symmetry breaking is less problematic. Once a local broken solution has been found, more models are included into the mixture, initialised close to the currently found solutions. In this way, a hierarchical breaking scheme is envisaged. Another popular method for initialisation is to center the means to those found by the K-means algorithm – however, this itself requires a heuristic initialisation.

20.3.3 Classification using Gaussian mixture models

Consider data drawn from two classes, $c \in \{1, 2\}$. We can fit a GMM $p(\mathbf{x}|c = 1, \mathcal{X}_1)$ to the data $\mathcal{X}_1$ from class 1, and another GMM $p(\mathbf{x}|c = 2, \mathcal{X}_2)$ to the data $\mathcal{X}_2$ from class 2. This would give rise to two class-conditional GMMs,

$$p(\mathbf{x}|c, \mathcal{X}_c) = \sum_{i=1}^{H} p(i|c)\mathcal{N}\left(\mathbf{x}|\mathbf{m}_i^c, \mathbf{S}_i^c\right) \tag{20.3.21}$$

For a novel point $\mathbf{x}^*$, the posterior class probability is

$$p(c|\mathbf{x}^*, \mathcal{X}) \propto p(\mathbf{x}^*|c, \mathcal{X}_c)p(c) \tag{20.3.22}$$

where $p(c)$ is the prior class probability. The Maximum Likelihood setting is that $p(c)$ is proportional to the number of training points in class c.

Consider a testpoint $\mathbf{x}^*$ a long way from the training data for both classes. For such a point, the probability that either of the two class models generated the data is very low. However, one will be much lower than the other (since Gaussians drop exponentially quickly), meaning that the posterior probability will be confidently close to 1 for that class which has a component closest to $\mathbf{x}^*$. This is an unfortunate property since we would end up confidently predicting the class of novel data that is not similar to anything we've seen before. We would prefer the opposite effect that for novel data far from the training data, the classification confidence drops and all classes become equally likely.

A remedy for this situation is to include an additional component in the Gaussian mixture for each class that is very broad. We collect the input data from all classes into a dataset $\mathcal{X}$, and let $\mathbf{m}$ be the mean of all this data and $\mathbf{S}$ the covariance. Then for the model of each class c data we include an additional Gaussian (dropping the notational dependency on $\mathcal{X}$)

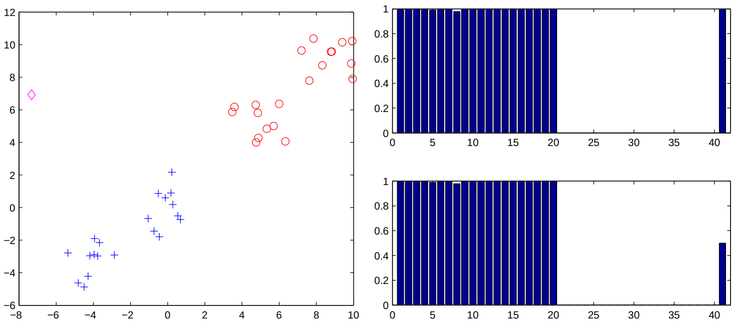
$$p(\mathbf{x}|c) = \sum_{i=1}^{H} \tilde{p}_i^c \mathcal{N}\left(\mathbf{x}|\mathbf{m}_i^c, \mathbf{S}_i^c\right) + \tilde{p}_{H+1}^c \mathcal{N}\left(\mathbf{x}|\mathbf{m}, \lambda\mathbf{S}\right) \tag{20.3.23}$$

where

$$\tilde{p}_i^c \propto \begin{cases} p_i^c & i \leq H \\ \delta & i = H + 1 \end{cases} \tag{20.3.24}$$

where δ is a small positive value and λ inflates the covariance (we take $\delta = 0.0001$ and $\lambda = 10$ in `demoGMMclass.m`). The effect of the additional component on the training likelihood is negligible since it has small weight and large variance compared to the other components, see fig(20.8). However, as we move away from the region where the first H components have appreciable mass, the additional component gains in influence since it has a higher variance. If we include the same additional component in the GMM for each class c then the influence of this additional component will be the same for each class, dominating as we move far from the influence of the other components. For a point far from the training data the likelihood will be roughly equal for each class since in this region the additional broad component dominates with equal measure. The posterior distribution will then tend to the prior class probability $p(c)$, mitigating the deleterious effect of a single GMM dominating when a testpoint is far from the training data.

Example 89. The data in fig(20.9a) has a cluster structure for each class. Based on fitting a GMM to each of the classes, a test point (diamond) far from the training data is confidently classified as belonging to class 1. This is an undesired effect since we would prefer that points far from the training data are not classified with any certainty. Including an additional large variance Gaussian component for each class, however, has little effect on the class probabilities of the training data, yet has the effect of making the class probability for the test point maximally uncertain, fig(20.9b).



(a) (b)

Figure 20.9: Class conditional GMM training and classification. **(a)**: Data from two different classes. We fit a GMM with two components to the data from each class. The (magenta) diamond is a test point far from the training data we will classify. **(b)**: Upper subpanel are the class probabilities $p(c = 1|n)$ for the 40 training points, and the 41^{st} point, being the test point. The lower subpanel are the class probabilities but including the additional large variance Gaussian term. See `demoGMMclass.m`.

Algorithm 19 K-means

1: Initialise the centres $\mathbf{m}_i$, $i = 1, \ldots, K$.
2: **while** not converged **do**
3: For each centre i, find all the $\mathbf{x}^n$ for which i is the nearest (in Euclidean sense) centre.
4: Call this set of points $\mathcal{N}_i$. Let N_i be the number of datapoints in set $\mathcal{N}_i$.
5: Update the means

$$\mathbf{m}_i^{new} = \frac{1}{N_i} \sum_{n \in \mathcal{N}_i} \mathbf{x}^n$$

6: **end while**

20.3.4 The Parzen estimator

The Parzen density estimator is formed by placing a 'bump of mass', $\rho(\mathbf{x}|\mathbf{x}^n)$, on each datapoint,

$$p(\mathbf{x}) = \frac{1}{N} \sum_{n=1}^{N} \rho(\mathbf{x}|\mathbf{x}^n) \tag{20.3.25}$$

A popular choice is (for a D dimensional $\mathbf{x}$)

$$\rho(\mathbf{x}|\mathbf{x}^n) = \mathcal{N}\left(\mathbf{x}|\mathbf{x}^n, \sigma^2 \mathbf{I}_D\right) \tag{20.3.26}$$

giving the mixture of Gaussians

$$p(\mathbf{x}) = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{(2\pi\sigma^2)^{D/2}} e^{-\frac{1}{2\sigma^2}(\mathbf{x}-\mathbf{x}^n)^2} \tag{20.3.27}$$

There is no 'training' required for a Parzen estimator – only the positions of the N datapoints need storing. Whilst the Parzen technique is a reasonable and cheap way to form a density estimator, it does not enable us to form any simpler description of the data. In particular, we cannot perform clustering since there is no lower number of clusters assumed to underly the data generating process. This is in contrast to GMMs trained using Maximum Likelihood on a fixed number $H \leq N$ of components.

20.3.5 K-Means

Consider a mixture of K isotropic Gaussians in which each covariance is constrained to be equal to $\sigma^2 \mathbf{I}$,

$$p(\mathbf{x}) = \sum_{i=1}^{K} p_i \mathcal{N}\left(\mathbf{x}|\mathbf{m}_i, \sigma^2 \mathbf{I}\right) \tag{20.3.28}$$

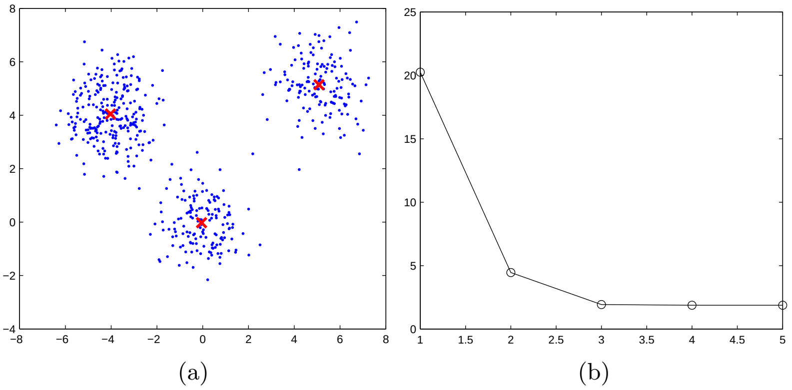


Figure 20.10: **(a)**: 550 datapoints clustered using K-means with 3 components. The means are given by the red crosses. **(b)**: Evolution of the mean square distance to nearest centre with iterations of the algorithm. The means were initialised to close to the overall mean of the data. See `demoKmeans.m`.

Whilst the EM algorithm breaks down if a Gaussian component is allowed to set $\mathbf{m}_i$ equal to a datapoint with $\sigma^2 \to 0$, by constraining all components to have the same variance σ^2, the algorithm has a well defined limit as $\sigma^2 \to 0$. The reader may show (exercise(203)) that in this case the membership distribution equation (20.3.9) becomes deterministic

$$p^{old}(n|i) \propto \begin{cases} 1 & \text{if } \mathbf{m}_i \text{ is closest to } \mathbf{x}^n \\ 0 & \text{otherwise} \end{cases} \tag{20.3.29}$$

In this limit the EM update (20.3.10) for the mean $\mathbf{m}_i$ is given by taking the average of the points closest to $\mathbf{m}_i$. This limiting and constrained GMM reduces to the so-called K-means algorithm, algorithm(19). Despite its simplicity the K-means algorithm converges quickly and often gives a reasonable clustering, provided the centres are initialised reasonably. See fig(20.10).

K-means is often used as a simple form of data compression. Rather than sending the datapoint $\mathbf{x}^n$, one sends instead the index of the centre to which it is associated. This is called *vector quantisation* and is a form of lossy compression. To improve the quality, more information can be transmitted such as an approximation of the difference between $\mathbf{x}$ and the corresponding mean $\mathbf{m}$, which can be used to improve the reconstruction of the compressed datapoint.

20.3.6 Bayesian mixture models

Bayesian extensions include placing priors on the parameters of each model in the mixture, and also on the component distribution. In most cases this will give rise to the marginal likelihood being an intractable integral and learning needs to be carried out with a form of approximate inference. Sampling techniques are popular in this respect[95]. See [106, 66] for an approximate variational treatment focussed on Bayesian Gaussian mixture models.

20.3.7 Semi-Supervised learning

In some cases we may know to which mixture component (or 'class') certain datapoints belong. Given this information we want to fit a mixture model with a specified number of components H and parameters θ. We write (v_*^m, h_*^m), $m = 1, \ldots, M$ for the M known datapoints and corresponding classes, and (v^n, h^n), $n = 1, \ldots, N$ for the remaining datapoints whose classes h^n are unknown. We aim then to maximise the likelihood

$$p(v_*^{1:M}, v^{1:N}|h_*^{1:M}, \theta) = \prod_m p(v_*^m|h_*^m, \theta) \prod_n \sum_{h^n} p(v^n|h^n, \theta)p(h^n) \tag{20.3.30}$$

If we were to lump all the datapoints together, this is essentially equivalent to the standard unsupervised case, expect that some of the h are fixed into known states. The only effect on the EM algorithm is therefore in the terms $p^{old}(h|v)$ which are delta functions in the known state, exercise(206).

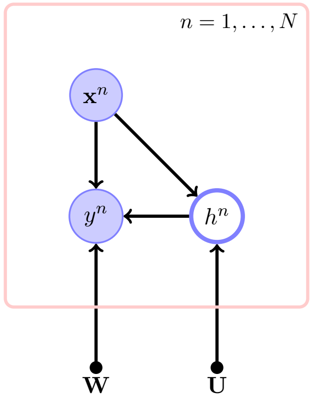


Figure 20.11: Mixture of experts model. The prediction of the output y^n (real or continuous) given the input $\mathbf{x}^n$ averages over individual experts $p(y^n|\mathbf{x}^n, \mathbf{w}_{h^n})$. The expert h^n is selected by the gating mechanism with probability $p(h^n|\mathbf{x}^n, \mathbf{U})$, so that some experts will be more able to predict inputs $\mathbf{x}$ in 'their' part of the space. The parameters $\mathbf{W}, \mathbf{U}$ can be learned by Maximum Likelihood after marginalising over the hidden expert variables.

20.4 Mixture of Experts

The mixture of experts model[147] is related to discriminative training of an output y distribution conditioned on an input $\mathbf{x}$. This can be used in either the regression of classification contexts and has the general form, see fig(20.11)

$$p(y|\mathbf{x}, \mathbf{W}, \mathbf{U}) = \sum_{h=1}^{H} p(y|\mathbf{x}, \mathbf{w}_h) p(h|\mathbf{x}, \mathbf{U}) \tag{20.4.1}$$

Here h indexes the mixture component. Unlike a standard mixture model, the component distribution $p(h|\mathbf{x}, \mathbf{U})$ is dependent on the input $\mathbf{x}$. This so-called gating distribution is conventionally taken to be of the softmax form

$$p(h|\mathbf{x}, \mathbf{U}) = \frac{e^{\mathbf{u}_h^\mathsf{T}\mathbf{x}}}{\sum_h e^{\mathbf{u}_h^\mathsf{T}\mathbf{x}}} \tag{20.4.2}$$

The idea is that we have a set of H predictive models (experts), $p(y|\mathbf{x}, \mathbf{w}_h)$, each with a different parameter $\mathbf{w}_h$, $h = 1, \ldots, H$. How suitable model h is for predicting with the current input $\mathbf{x}$ is determined by the alignment of input $\mathbf{x}$ with the weight vector $\mathbf{u}_h$. In this way the input $\mathbf{x}$ is softly assigned to the appropriate experts. Training this model requires learning the expert parameters $\mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_H]$ and gating parameters $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_H]$.

Maximum Likelihood training can be achieved using a form of EM. We will not derive the EM algorithm for the mixture of experts model in full, merely pointing the direction along which the derivation would continue. For a single datapoint $\mathbf{x}$, the EM energy term is

$$\langle \log p(y|\mathbf{x}, \mathbf{w}_h) p(h|\mathbf{x}, \mathbf{U}) \rangle_{p(h|\mathbf{x}, \mathbf{W}^{old}, \mathbf{U}^{old})} \tag{20.4.3}$$

For regression a simple choice is

$$p(y|\mathbf{x}, \mathbf{w}_h) = \mathcal{N}\left(y|\mathbf{x}^\mathsf{T}\mathbf{w}_h, \sigma^2\right) \tag{20.4.4}$$

and for (binary) classification

$$p(y = 1|\mathbf{x}, \mathbf{w}_h) = \sigma(\mathbf{x}^\mathsf{T}\mathbf{w}_h) \tag{20.4.5}$$

In both cases computing the derivatives of the energy with respect to the parameters $\mathbf{W}$ is straightforward, so that an EM algorithm is readily available. An alternative to EM is to compute the gradient of the likelihood directly using the standard approach discussed in section(11.7).

A Bayesian treatment is to consider

$$p(y|\mathbf{x}) = \int_{\mathbf{W},\mathbf{U}} \sum_h p(y|\mathbf{x}, \mathbf{w}_h) p(h|\mathbf{x}, \mathbf{u}) p(\mathbf{W}) p(\mathbf{U}) \tag{20.4.6}$$

where it is conventional to assume $p(\mathbf{W}) = \prod_h p(\mathbf{w}_h)$, $p(\mathbf{U}) = \prod_h p(\mathbf{u}_h)$. The integrals are intractable and approximations are required. See [287] for a variational treatment for regression and [43] for a variational treatment of classification. An extension to Bayesian model selection in which the number of experts is estimated is considered in [142].

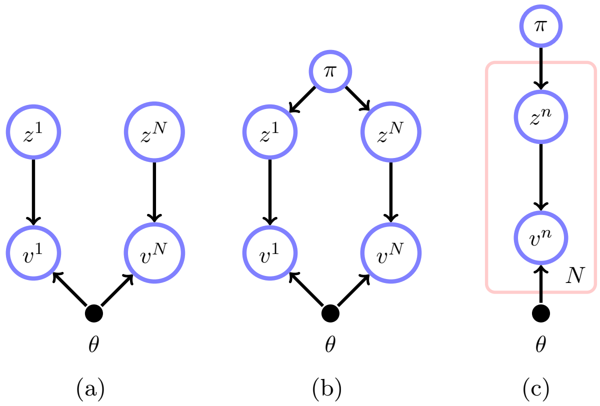


Figure 20.12: **(a)**: A generic mixture model for data $v^{1:N}$. Each z^n indicates the cluster of each datapoint. θ is a set of parameters and $z^n = k$ selects parameter θ^k for datapoint v^n. **(b)**: For a potentially large number of clusters one way to control complexity is to constrain the joint indicator distribution. **(c)**: Plate notation of (b).

20.5 Indicator models

In the indicator approach we specify a distribution over the cluster assignments. For consistency with the literature we use an 'indicator' z, as opposed to a hidden variable h, although they play the same role. A clustering model with parameters θ on the component models and joint indicator prior $p(z^{1:N}$ takes the form

$$p(v^{1:N}|\theta) = \sum_{z^{1:N}} p(v^{1:N}|z^{1:N}, \theta)p(z^{1:N}) \tag{20.5.1}$$

Since z^n are indicates cluster membership,

$$p(v^{1:N}|\theta) = \sum_{z^{1:N}} p(z^{1:N}) \prod_{n=1}^{N} p(v^n|z^n, \theta) \tag{20.5.2}$$

Below we discuss the role of different indicator models $p(z^{1:N})$ in clustering.

20.5.1 Joint indicator approach: factorised prior

Assuming independence of indicators,

$$p(z^{1:N}) = \prod_{n=1}^{N} p(z^n), \qquad z^n \in \{1, \dots, K\} \tag{20.5.3}$$

we obtain from equation (20.5.2)

$$p(v^{1:N}|\theta) = \sum_{z^{1:N}} \prod_{n=1}^{N} p(v^n|z^n, \theta)p(z^n) = \prod_{n=1}^{N} \sum_{z^n} p(v^n|z^n, \theta)p(z^n) \tag{20.5.4}$$

which recovers the standard mixture model equation (20.1.2). As we discuss below, more sophisticated joint indicator priors can be used to explicitly control the complexity of the indicator assignments and open the path to essentially 'infinite dimensional' models.

20.5.2 Joint indicator approach : Polya prior

For a large number of available clusters $K \gg 1$, using a factorised joint indicator distribution could potentially lead to overfitting with little or no meaningful clustering. One way to control the effective number of components that are used is via a parameter π that regulates the complexity,

$$p(z^{1:N}) = \int_{\pi} \left\{ \prod_n p(z^n|\pi) \right\} p(\pi) \tag{20.5.5}$$

where $p(z|\pi)$ is a categorical distribution,

$$p(z^n = k|\pi) = \pi_k \tag{20.5.6}$$

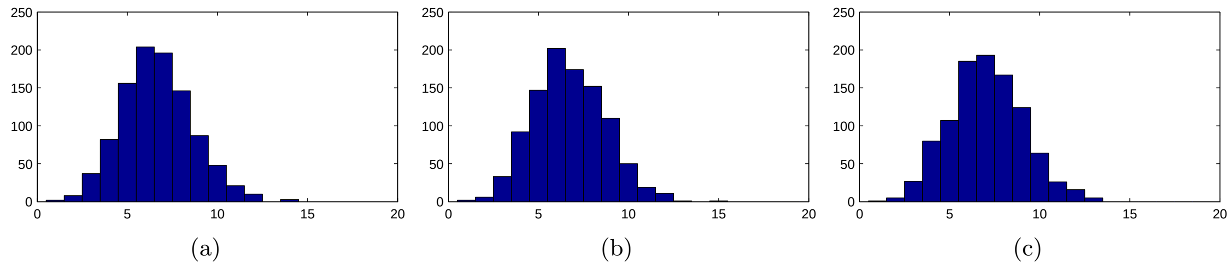


Figure 20.13: The number of unique clusters U when indicators are sampled from a Polya distribution equation (20.5.8), with $\alpha = 2$, and $N = 50$ datapoints. **(a)**: $K = 50$, **(b)**: $K = 100$, **(c)**: $K = 1000$. Even though the number of available clusters K is larger than the number of datapoints, the effective number of used clusters remains constrained. See `demoPolya.m`.

A convenient choice for $p(\pi)$ is the Dirichlet distribution (since this is conjugate to the categorical distribution),

$$p(\pi) = \text{Dirichlet}\left(\pi|\alpha\right) \propto \prod_{k=1}^{K} \pi_k^{\alpha/K-1} \tag{20.5.7}$$

The integral over π can be performed analytically to give a Polya distribution:

$$p(z^{1:N}) = \frac{\Gamma(\alpha)}{\Gamma(N+\alpha)} \prod_{k=1}^{K} \frac{\Gamma(N_k + \alpha/K)}{\Gamma(\alpha/K)}, \qquad N_k \equiv \sum_n \mathbb{I}\left[z^n = k\right] \tag{20.5.8}$$

The number of unique clusters used is then given by $U = \sum_k \mathbb{I}\left[N_k > 0\right]$. The distribution over likely cluster numbers is controlled by the parameter α, fig(20.13). The scaling α/K ensures a sensible limit as $K \to \infty$, see fig(20.13), in which limit the models are known as *Dirichlet process mixture models*. This approach means that we do not need to explicitly constrain the number of possible components K since the number of active components U remains limited even for very large K.

Clustering is achieved by considering $\underset{z^{1:N}}{\text{argmax}}\, p(z^{1:N}|v^{1:N})$. In practice it is common to consider

$$\underset{z^n}{\text{argmax}}\, p(z^n|v^{1:N}) \tag{20.5.9}$$

Unfortunately, posterior inference of $p(z^n|v^{1:N})$ for this class of models is formally computationally intractable and approximate inference techniques are required. A detailed discussion of these techniques is beyond the scope of this book, and we refer the reader to [162] for a deterministic (variational) approach and [204] for a discussion of sampling approaches.

20.6 Mixed Membership models

Unlike standard mixture models in which each object is assumed to have been generated from a single cluster, in mixed membership models an object may be a member of more than one group. Latent Dirichlet Allocation in an example of such a mixed membership model, and is one of a number of models developed in recent years, [4, 89].

20.6.1 Latent Dirichlet Allocation

So far we've considered clustering in the sense that each observation is assumed to have been generated from a single cluster. In contrast, Latent Dirichlet Allocation[44] and related methods are generative *mixed membership* models in which each datapoint may belong to more than a single cluster. A typical application is to identify topic clusters in a collection of documents. A single document contains a sequence of words, for example

$$v = (\mathsf{the}, \mathsf{cat}, \mathsf{sat}, \mathsf{on}, \mathsf{the}, \mathsf{cat}, \mathsf{mat}) \tag{20.6.1}$$

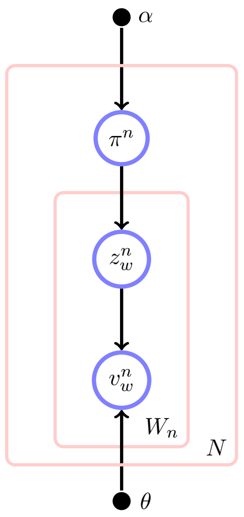


Figure 20.14: Latent Dirichlet Allocation. For document n we first sample a distribution of topics for that document π^n. Then for each word position w in the document we sample a topic z_w^n from the topic distribution. Given the topic we then sample a word from the word distribution of that topic. The parameters of the model are the word distributions for each topic θ, and the parameters of the topic distribution α.

If each word in the available dictionary is assigned to a unique state (say $\mathsf{dog} = 1, \mathsf{tree} = 2, \mathsf{cat} = 3, \ldots$), we can represent then the n^{th} document as a vector

$$v^n = \left(v_1^n, \ldots, v_{W_n}^n\right), \qquad v_i^n \in \{1, \ldots, D\} \tag{20.6.2}$$

where W_n is the number of words in the n^{th} document. The number of words W_n in each document can vary although the overall dictionary from which they came is fixed.

The aim is to find common topics in documents, assuming that any document could potentially contain more than one topic. It is useful to think first of an underlying generative model of words, including latent topics (which we will later integrate out). We first sample a probability distribution (histogram) that represents the topics likely to occur for this document. Then, for each word-position in the document, sample a topic and subsequently a word from the distribution of words for that topic.

Mathematically, for document n and the w^{th} word-position in the document, v_w^n, we use $z_w^n \in \{1, \ldots, K\}$ to indicate which of the K possible topics that word belongs. For each topic k, one then has a categorical distribution over all the words $i = 1, \ldots, D$, in the dictionary:

$$p(v_w^n = i | z_w^n = k, \theta) = \theta_{i|k} \tag{20.6.3}$$

The 'animal' has high probability to emit animal-like words, *etc.* For each document n we have a distribution of topics π^n with $\sum_{k=1}^{K} \pi_k^n = 1$ which gives a latent description of the document in terms of its topic membership. For example, document n (which discusses issues related to wildlife conservation) might have a topic distribution with high mass on the latent 'animals' and 'environment', topics. Note that the topics are indeed latent – the name 'animal' would be given post-hoc based on the kinds of words that the latent topic would generate, $\theta_{i|k}$. As in section(20.5.2), to control complexity one may use a Dirichlet prior to limit the number of topics active in any particular document:

$$p(\pi^n | \boldsymbol{\alpha}) = \text{Dirichlet}\left(\pi^n | \boldsymbol{\alpha}\right) \tag{20.6.4}$$

where $\boldsymbol{\alpha}$ is a vector of length the number of topics.

A generative model for sampling a document v^n is:

1. Choose $\pi^n \sim \text{Dirichlet}\left(\pi^n | \boldsymbol{\alpha}\right)$
2. For each of word position v_w^n, $w = 1, \ldots, W_n$:
   (a) Choose a topic $z_w^n \sim p\left(z_w^n | \pi^n\right)$
   (b) Choose a word $v_w^n \sim p\left(v_w^n | \theta_{\cdot | z_w^n}\right)$

Arts	Budgets	Children	Education
new	million	children	school
film	tax	women	students
show	program	people	schools
music	budget	child	education
movie	billion	years	teachers
play	federal	families	high
musical	year	work	public
best	spending	parents	teacher
actor	new	says	bennett
first	state	family	manigat
york	plan	welfare	namphy
opera	money	men	state
theater	programs	percent	president
actress	government	care	elementary
love	congress	life	haiti

(a)

The William Randolph Hearst Foundation will give $ 1.25 million to Lincoln Center, Metropolitan Opera Co., New York Philharmonic and Juilliard School. Our board felt that we had a real opportunity to make a mark on the future of the performing arts with these grants an act every bit as important as our traditional areas of support in health, medical research, education and the social services, Hearst Foundation President Randolph A. Hearst said Monday in announcing the grants. Lincoln Centers share will be $200,000 for its new building, which will house young artists and provide new public facilities. The Metropolitan Opera Co. and New York Philharmonic will receive $400,000 each. The Juilliard School, where music and the performing arts are taught, will get $250,000. The Hearst Foundation, a leading supporter of the Lincoln Center Consolidated Corporate Fund, will make its usual annual $100,000 donation, too.

(b)

Figure 20.15: **(a)**: A subset of the latent topics discovered by LDA and the high probability words associated with each topic. Each column represents a topic, with the topic name such as 'art' assigned by hand after viewing the most likely words corresponding to the topic. **(b)**: A document from the training data in which the words are coloured according to the most likely latent topic. This demonstrates the mixed-membership nature of the model, assigning the datapoint (document in this case) to several clusters (topics). Reproduced from [44].

Training the LDA model corresponds to learning the parameters $\boldsymbol{\alpha}$, which relates to the number of topics, and θ, which describes the distribution of words within each topic. Unfortunately, finding the requisite marginals for learning from the posterior is formally computationally intractable. Efficient approximate inference for this class of models is a topic of research interest and both variational and sampling approaches have been developed[44, 270, 222].

There are close similarities between LDA and PLSA[110], section(15.6.1), both of which describe a document in terms of a distribution over latent topics. LDA is a probabilistic model for which issues such as setting hyperparameters can be addressed using Maximum Likelihood. PLSA on the other hand is essentially a matrix decomposition technique (such as PCA). Issues such as hyperparameters setting for PLSA are therefore addressed using validation data. Whilst PLSA is a description only of the training data, LDA is a generative data model and can in principle be used to synthesise new documents.

Example 90. An illustration of the use of LDA is given in fig(20.15)[44]. The documents are taken from the TREC Associated Press corpus containing 16,333 newswire articles with 23,075 unique terms. After removing a standard list of *stop words* (frequent words such as 'the','a' *etc.* that would otherwise dominate the statistics), the EM algorithm (with variational approximate inference) was used to find the Dirichlet and conditional categorical parameters for a 100-topic LDA model. The top words from four resulting categorical distributions $\theta_{i|k}$ are illustrated fig(20.15a). These distributions capture some of the underlying topics in the corpus. An example document from the corpus is presented along with the words coloured by the most probable latent topic they correspond to.

20.6.2 Graph based representations of data

Mixed membership models are used in a variety of contexts and are distinguished also by the form of data available. Here we focus on analysing a representation of the interactions amongst a collection of objects; in particular, the data has been processed such that all the information of interest is characterised by an interaction matrix. For graph based representations of data, two objects are similar if they are neighbours on a graph representing the data objects. In the field of social-networks, for example, each individual is represented as a node in a graph, with a link between two nodes if the individuals are friends. Given a graph one might wish to identify communities of closely linked friends. Interpreted as a social network, in fig(20.16a), individual 3 is a member of his work group $(1, 2, 3)$ and also the poker group $(3, 4, 5)$. These

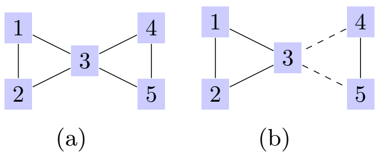


Figure 20.16: (a) The social network of a set of 5 individuals, represented as an undirected graph. Here individual 3 belongs to the group $(1, 2, 3)$ and also $(3, 4, 5)$. (b) By contrast, in Graph Partitioning, one breaks the graph into roughly equally sized disjoint partitions such that each node is a member of only a single partition, with a minimal number of edges between partitions.

two groups of individuals are otherwise disjoint. Discovering such groupings contrasts with *graph partitioning* in which each node is assigned to only one of a set of subgraphs, fig(20.16b), for which a typical criterion is that each subgraph should be roughly of the same size and there are few connections between the subgraphs[153].

Another example is that nodes in the graph represent products and a link between nodes i and j indicates that customers who by product i frequently also buy product j. The aim is to decompose the graph into groups, each corresponding to products that are commonly co-bought by customers[113].

A growing area of application of graph based representations is in bioinformatics in which nodes represent genes, and a link between them representing that the two genes have similar activity profiles. The task is then to identify groups of similarly behaving genes[5].

20.6.3 Dyadic data

Consider two kinds of objects, for example, films and customers. Each film is indexed by $f = 1, \ldots, F$ and each user by $u = 1, \ldots, U$. The interaction of user u with film f can be described by the element of a matrix M_{uf} representing the rating a user gives to a film. A dyadic dataset consists of such a matrix and the aim is to decompose this matrix to explain the ratings by finding types of films and types of user.

Another example is to consider a collection of documents, summarised by an interaction matrix in which $M_{w,d}$ is 1 if word w appears in document d and zero otherwise. This matrix can be represented as a bipartite graph, as in fig(20.17a). The upper nodes represent documents, and the lower nodes words, with a link between them if that word occurs in that document. One might then seek assignments of documents to groups or latent 'topics' to succinctly explain the link structure of the bipartite graph via a small number of latent nodes, as schematically depicted in fig(20.17b). One may view this as a form of matrix factorisation[135, 186]

$$M_{wd} \approx \sum_t U_{wt} V^{\mathsf{T}}_{td} \tag{20.6.5}$$

where t indexes the topics and the feature matrices $\mathbf{U}$ and $\mathbf{V}$ control the word-to-topic mapping and the topic-to-document mapping[2]. More generally, we can write a distribution

$$p(\mathbf{M}|\mathbf{U}, \mathbf{V}) \tag{20.6.6}$$

where $\mathbf{U}$ and $\mathbf{V}$ are feature matrices. In [186], real-valued data is modelled using

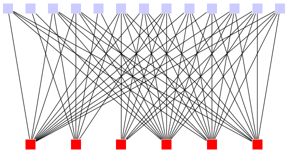
$$p(\mathbf{M}|\mathbf{U}, \mathbf{W}, \mathbf{V}) = \mathcal{N}\left(\mathbf{M}|\mathbf{U}\mathbf{W}\mathbf{V}^{\mathsf{T}}, \sigma^2\mathbf{I}\right) \tag{20.6.7}$$

where $\mathbf{U}$ and $\mathbf{V}$ are assumed binary and the real-valued $\mathbf{W}$ is a topic-interaction matrix. In this viewpoint learning then consists of inferring these matrices, given the dyadic observation matrix $\mathbf{M}$. Assuming factorised priors, the posterior over the matrices is

$$p(\mathbf{U}, \mathbf{W}, \mathbf{V}|\mathbf{M}) \propto p(\mathbf{M}|\mathbf{U}, \mathbf{W}, \mathbf{V})p(\mathbf{U})p(\mathbf{W})p(\mathbf{V}) \tag{20.6.8}$$

In [186] a convenient choice is a Gaussian prior distribution for $\mathbf{W}$, with the feature matrices $\mathbf{U}$ and $\mathbf{V}$ sampled from Beta-Bernoulli priors. The resulting posterior distribution is formally computationally intractable, and in [186] this is addressed using a sampling approximation.

[2]This is different to latent Dirichlet allocation which has a probabilistic interpretation of first generating a topic and then a word, conditional on the chosen topic. Here the interaction between document-topic matrix $\mathbf{V}$ and word-topic matrix $\mathbf{U}$ is non-probabilistic. Additional clarification required.


(a)

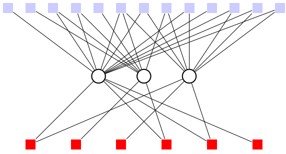

(b)

Figure 20.17: Graphical representation of dyadic data. **(a)**: There are 6 documents and 13 words. A link represents that a particular word-document pair occurs in the dataset. **(b)**: A latent decomposition of (a) using 3 'topics'. A topic corresponds to a collection of words, and each document a collection of topics. The open nodes indicate latent variables.

20.6.4 Monadic data

In monadic data there is only one type of object and the interaction between the objects is represented by a square interaction matrix. For example one might have a matrix with elements $A_{ij} = 1$ if proteins i and j can bind to each other and 0 otherwise. A graphical depiction of the interaction matrix is given by a graph in which an edge represents an interaction, for example fig(20.18). One then seeks a decomposition that explains this link structure in a parsimonious way using latent variables. Graphically this means that we seek a bipartite representation of the original graph fig(20.18) as given in fig(20.19a).

The perhaps more standard concept of statistical clustering is to assign each object to only one of a number of clusters – it cannot be a member of more than one cluster itself. For example, a gene might be assigned to a single gene-cluster such that all genes in a cluster have a similar microarray expression profile[132]. Graphically this would restrict the degree of each visible node in fig(20.19a) to 1. For a mixed membership model no such restriction is imposed. In the following section we discuss a particular mixed membership model and highlight potential applications. The method is based on clique decompositions of graphs and as such we require a short digression into clique-based graph representations.

20.6.5 Cliques and adjacency matrices for monadic binary data

A symmetric adjacency matrix has elements $A_{ij} \in \{0, 1\}$, with a 1 indicating a link between nodes i and j. For the graph in fig(20.18), the adjacency matrix is

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \tag{20.6.9}$$

where we include self connections on the diagonal. Given $\mathbf{A}$, our aim is to find a 'simpler' description that reveals the underlying cluster structure, such as $(1, 2, 3)$ and $(2, 3, 4)$ in fig(20.18). Given the undirected graph in fig(20.18), the incidence matrix $\mathbf{F}_{inc}$ is an alternative description of the adjacency structure[80]. Given the V nodes in the graph, we construct $\mathbf{F}_{inc}$ as follows: For each link $i \sim j$ in the graph, form a column of the matrix $\mathbf{F}_{inc}$ with zero entries except for a 1 in the i^{th} and j^{th} row. The column ordering is arbitrary. For example, for the graph in fig(20.18) an incidence matrix is

$$\mathbf{F}_{inc} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix} \tag{20.6.10}$$

The incidence matrix has the property that the adjacency structure of the original graph is given by the product of the incidence matrix with itself. The diagonal entries contain the degree (number of links) of

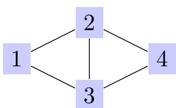


Figure 20.18: The minimal clique cover is $(1, 2, 3), (2, 3, 4)$.

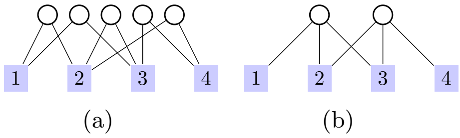


Figure 20.19: Bipartite representations of the decompositions of fig(20.18). Shaded nodes represent observed variables, and open nodes latent variables. (a) Incidence, (b) Minimal Clique decomposition.

each node. For our example, this gives

$$\mathbf{F}_{inc}\mathbf{F}_{inc}^{\mathsf{T}} = \begin{pmatrix} 2 & 1 & 1 & 0 \\ 1 & 3 & 1 & 1 \\ 1 & 1 & 3 & 1 \\ 0 & 1 & 1 & 2 \end{pmatrix} \tag{20.6.11}$$

so that

$$\mathbf{A} = H\left(\mathbf{F}_{inc}\mathbf{F}_{inc}^{\mathsf{T}}\right) \tag{20.6.12}$$

Here $H(\cdot)$ is the element-wise Heaviside step function, $[H(\mathbf{M})]_{ij} = 1$ if $M_{ij} > 0$ and is 0 otherwise. A useful viewpoint of the incidence matrix is that it identifies two-cliques in the graph (here we are using the term 'clique' in the non-maximal sense). There are five 2-cliques in fig(20.18), and each column of $\mathbf{F}_{inc}$ specifies which elements are in each 2-clique. Graphically we can depict this incidence decomposition as a bipartite graph, as in fig(20.19a) where the open nodes represent the five 2-cliques. The incidence matrix can be generalised to describe larger cliques. Consider the following matrix as a decomposition for fig(20.18), and its outer-product:

$$\mathbf{F} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 1 \end{pmatrix}, \quad \mathbf{F}\mathbf{F}^{\mathsf{T}} = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 2 & 2 & 1 \\ 1 & 2 & 2 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \tag{20.6.13}$$

The interpretation is that $\mathbf{F}$ represents a decomposition into two 3-cliques. As in the incidence matrix, each column represents a clique, and the rows containing a '1' express which elements are in the clique defined by that column. This decomposition can be represented as the bipartite graph of fig(20.19b). For the graph of fig(20.18), both $\mathbf{F}_{inc}$ and $\mathbf{F}$ satisfy

$$\mathbf{A} = H\left(\mathbf{F}\mathbf{F}^{\mathsf{T}}\right) = H\left(\mathbf{F}_{inc}\mathbf{F}_{inc}^{\mathsf{T}}\right) \tag{20.6.14}$$

One can view equation (20.6.14) as a form of binary matrix factoristation of the binary square (symmetric) matrix $\mathbf{A}$ into non-square binary matrices. For our clustering purposes, the decomposition using $\mathbf{F}$ is to be preferred to the incidence decomposition since $\mathbf{F}$ decomposes the graph into a smaller number of larger cliques. A formal specification of the problem of finding a minimum number of maximal fully-connected subsets is the computational problem `MIN CLIQUE COVER`[100, 248]. Indeed, $\mathbf{F}$ solves `MIN CLIQUE COVER` for fig(20.16b).

Definition 106 (Clique matrix)**.** Given an adjacency matrix $[A]_{ij}, i, j = 1, \ldots, V$ $(A_{ii} = 1)$, a clique matrix[3] $\mathbf{F}$ has elements $\mathbf{F}_{i,c} \in \{0, 1\}, i = 1, \ldots, V, c = 1, \ldots C$ such that $\mathbf{A} = H(\mathbf{F}\mathbf{F}^{\mathsf{T}})$. Diagonal elements $\left[\mathbf{F}\mathbf{F}^{\mathsf{T}}\right]_{ii}$ express the number of cliques/columns that node i occurs in. Off-diagonal elements $\left[\mathbf{F}\mathbf{F}^{\mathsf{T}}\right]_{ij}$ contain the number of cliques/columns that nodes i and j jointly inhabit [18].

Whilst finding a clique decomposition $\mathbf{F}$ is easy (use the incidence matrix for example), finding a clique decomposition with the minimal number of columns, *i.e.* solving `MIN CLIQUE COVER`, is NP-Hard[100, 10].

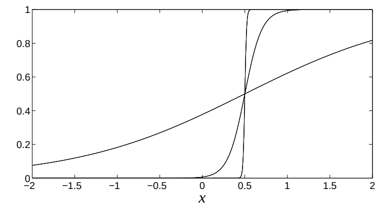


Figure 20.20: The function $\sigma(x) \equiv \left(1+e^{\beta(0.5-x)}\right)^{-1}$ for $\beta = 1, 10, 100$. As β increases, this sigmoid function tends to a step function.

A generative model of adjacency matrices

Solving `MIN CLIQUE COVER` is a computationally hard problem and approximations are in general unavoidable. Below we relax the strict clique requirement and assume that provided that only a small number of links in an 'almost clique' are missing, this may be considered a sufficiently well-connected group of nodes to form a cluster.

Given an adjacency matrix $\mathbf{A}$ and a prior on clique matrices $\mathbf{F}$, our interest is the posterior

$$p(\mathbf{F}|\mathbf{A}) \propto p(\mathbf{A}|\mathbf{F})p(\mathbf{F}) \tag{20.6.15}$$

We first concentrate on the generative term $p(\mathbf{A}|\mathbf{F})$. To find 'well-connected' clusters, we relax the constraint that the decomposition is in the form of cliques in the original graph and view the absence of links as statistical fluctuations away from a perfect clique. Given a $V \times C$ matrix $\mathbf{F}$, we desire that the higher the overlap between rows[4] $\mathbf{f}_i$ and $\mathbf{f}_j$ is, the greater the probability of a link between i and j. This may be achieved using, for example,

$$p(A_{ij} = 1|\mathbf{F}) = \sigma\left(\mathbf{f}_i\mathbf{f}_j^{\mathsf{T}}\right) \tag{20.6.16}$$

with

$$\sigma(x) \equiv \left(1+e^{\beta(0.5-x)}\right)^{-1} \tag{20.6.17}$$

where β controls the steepness of the function, see fig(20.20). The 0.5 shift in equation (20.6.17) ensures that σ approximates the step-function since the argument of σ is an integer. Under equation (20.6.16), if $\mathbf{f}_i$ and $\mathbf{f}_j$ have at least one '1' in the same position, $\mathbf{f}_i\mathbf{f}_j^{\mathsf{T}} - 0.5 > 0$ and $p(A_{ij} = 1)$ is high. Absent links contribute $p(A_{ij} = 0|\mathbf{F}) = 1 - p(A_{ij} = 1|\mathbf{F})$. The parameter β controls how strictly $\sigma(\mathbf{F}\mathbf{F}^{\mathsf{T}})$ matches $\mathbf{A}$; for large β, very little flexibility is allowed and only cliques will be identified. For small β, subsets that would be cliques if it were not for a small number of missing links, are clustered together. The setting of β is user and problem dependent.

Assuming each element of the adjacency matrix is sampled independently from the generating process, the joint probability of observing $\mathbf{A}$ is (neglecting its diagonal elements),

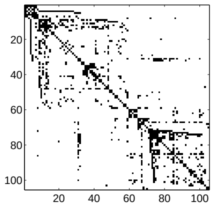
$$p(\mathbf{A}|\mathbf{F}) = \prod_{i\sim j}\sigma\left(\mathbf{f}_i\mathbf{f}_j^{\mathsf{T}}\right)\prod_{i\not\sim j}\left(1-\sigma\left(\mathbf{f}_i\mathbf{f}_j^{\mathsf{T}}\right)\right) \tag{20.6.18}$$

The ultimate quantity of interest is the posterior distribution of clique structure, given the known adjacency structure which, according to Bayes' rule is given by,

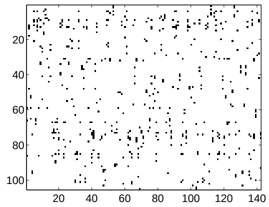
$$p(\mathbf{F}|\mathbf{A}) \propto p(\mathbf{A}|\mathbf{F})p(\mathbf{F}) \tag{20.6.19}$$

where $p(\mathbf{F})$ is a prior over clique matrices.

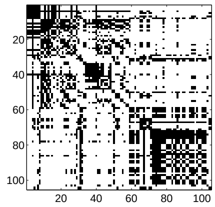
[4]We use lower indices $\mathbf{f}_i$ to denote the i^{th} row of $\mathbf{F}$.



(a)



(b)



(c)

Figure 20.21: **(a)**: Adjacency matrix of 105 Political Books (black=1). **(b)**: Clique matrix: 521 non-zero entries. **(c)**: Adjacency reconstruction using an Approximate Clique Matrix with 10 cliques – see also fig(20.22) and `demoCliqueDecomp.m`.

Clique matrix prior $p(\mathbf{F})$

To bias the contributions to the adjacency matrix $\mathbf{A}$ to occur from a small number of columns of $\mathbf{F}$, we first reparameterise $\mathbf{F}$ as

$$\mathbf{F} = \left(\alpha_1 \mathbf{f}^1, \ldots, \alpha_{C_{max}} \mathbf{f}^{C_{max}}\right) \tag{20.6.20}$$

where $\alpha_c \in \{0, 1\}$ play the role of indicators and $\mathbf{f}^c$ is the vector of column c of $\mathbf{F}$. C_{max} is an assumed maximal number of clusters. Ideally, we would like to find a likely solution $\mathbf{F}$ with a low number of indicators $\alpha_1, \ldots, \alpha_{C_{max}}$ in state 1. To achieve this we define a prior distribution on the binary hypercube $\alpha = (\alpha_1, \ldots, \alpha_{C_{max}})$,

$$p(\boldsymbol{\alpha}|\nu) = \prod_c \nu^{\alpha_c} (1-\nu)^{1-\alpha_c} \tag{20.6.21}$$

To encourage a small number of $\alpha' s$ to be used, we use a Beta prior $p(\nu)$ with suitable parameters. This gives rise to a Beta-Bernoulli distribution

$$p(\boldsymbol{\alpha}) = \int_\nu p(\alpha|\nu)p(\nu) = \frac{B(a+N, b+C_{max}-N)}{B(a,b)} \tag{20.6.22}$$

where $B(a, b)$ is the beta function and $N = \sum_{c=1}^{C_{max}} \alpha_c$ is the number of indicators in state 1. To encourage that only a small number of components should be active, we set $a = 1, b = 3$. The distribution (20.6.22) is on the vertices of the binary hypercube $\{0, 1\}^{C_{max}}$ with a bias towards vertices close to the origin $(0, \ldots, 0)$. Through equation (20.6.20), the prior on $\boldsymbol{\alpha}$ induces a prior on $\mathbf{F}$. The resulting distribution $p(\mathbf{F}, \boldsymbol{\alpha}|\mathbf{A}) \propto p(\mathbf{F}|\boldsymbol{\alpha})p(\boldsymbol{\alpha})$ is formally intractable in [18] this is addressed using a variational technique. See `cliquedecomp.c` and `cliquedecomp.m`.

Clique matrices play a natural role in the parameterisation of positive definite matrices. A discussion of this topic is beyond the scope of this book and we refer the interested reader to [18].MAKE AN EXERCISE FOR THIS

Example 91 (Political Books Clustering)**.** The data consists of 105 books on US politics sold by the online bookseller Amazon. The adjacency matrix with element $A_{ij} = 1$ fig(20.21a), represents frequent co-purchasing of books i and j (Valdis Krebs, www.orgnet.com). Additionally, books are labelled 'liberal', 'neutral', or 'conservative' according to the judgement of a politically astute reader (`www-personal.umich.edu/~mejn/netdata/`). The interest is to assign books to clusters, using $\mathbf{A}$ alone, and then see if these clusters correspond in some way to the ascribed political leanings of each book. Note that the information here is minimal – all that is known to the clustering algorithm is which books were co-bought (matrix $\mathbf{A}$); no other information on the content or title of the books are exploited by the algorithm. With an initial $C_{max} = 200$ cliques, Beta parameters $a = 1, b = 3$ and steepness $\beta = 10$, the posterior contains 142 cliques fig(20.21b), giving a perfect reconstruction of the adjacency $\mathbf{A}$. For comparison, the incidence matrix has 441 2-cliques. However, this clique matrix is too large to provide a compact interpretation of the data – indeed there are more clusters than books. To cluster the data more aggressively, we fix $C_{max} = 10$ and re-run the algorithm. This results only in an approximate clique decomposition, $\mathbf{A} \approx H(\mathbf{F}\mathbf{F}^\mathsf{T})$, as plotted in fig(20.21c). The resulting 105×10 approximate clique matrix

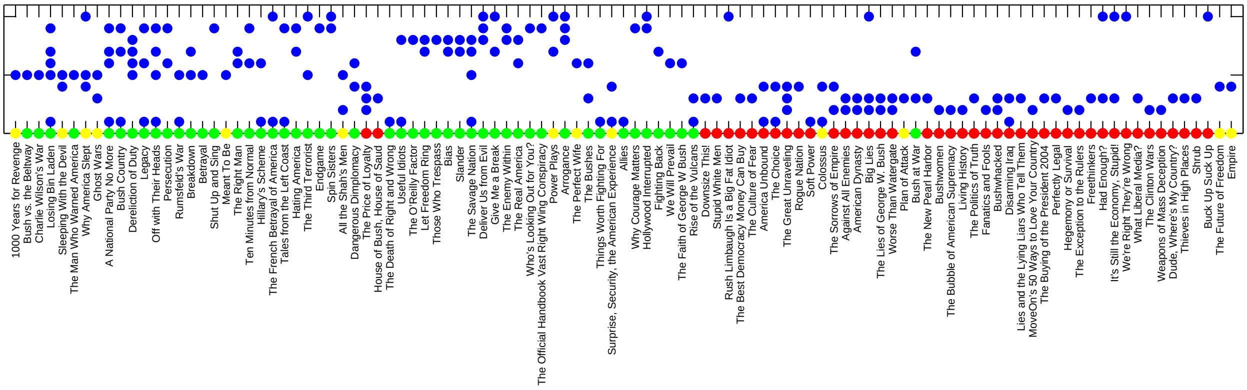


Figure 20.22: Political Books. 105 × 10 dimensional clique matrix broken into 3 groups by a politically astute reader. A black square indicates $q(\mathbf{f}_{i,c}) > 0.5$. Liberal books (red), Conservative books (green), Neutral books(yellow). By inspection, cliques 5,6,7,8,9 largely correspond to 'conservative' books.

is plotted in fig(20.22) and demonstrates how individual books are present in more than one cluster. Interestingly, the clusters found only on the basis of the adjacency matrix have some correspondence with the ascribed political leanings of each book; cliques $5, 6, 7, 8, 9$ correspond to largely 'conservative' books. Most books belong to more than a single clique/cluster, suggesting that they are not single topic books, consistent with the assumption of a mixed membership model.

20.7 Further reading

The literature on mixture modelling is extensive, and a good overview and entrance to the literature is contained in [185].

20.8 Code

`MIXprodBern.m`: EM training of a Mixture of product Bernoulli distributions
`demoMixBernoulli.m`: Demo of a Mixture of product Bernoulli distributions

`GMMem.m`: EM training of a mixture of Gaussians
`GMMloglik.m`: GMM log likelihood
`demoGMMem.m`: Demo of a EM for mixture of Gaussians
`demoGMMclass.m`: Demo GMM for classification

`Kmeans.m`: K-means
`demoKmeans.m`: Demo of K-means

`demoPolya.m`: Demo of the number of active clusters from a Polya distribution
`dirrnd.m`: Dirichlet random distribution generator

`cliquedecomp.m`: Clique Matrix decomposition
`cliquedecomp.c`: Clique Matrix decomposition (C-code)
`DemoCliqueDecomp.m`: Demo clique matrix decomposition

20.9 Exercises

Exercise 200. *Consider a mixture of factorised models*

$$p(v) = \sum_h p(h) \prod_i p(v_i|h) \tag{20.9.1}$$

For assumed i.i.d. data $v^n, n = 1, \ldots, N$, some observation components may be missing so that, for example the third component of the fifth datapoint, v_3^5 is unknown. Show that Maximum Likelihood training on the observed data corresponds to ignoring components v_i^n that are missing.

Exercise 201. *Consider data points generated from two different classes. Class 1 has the distribution $p(x|c = 1) \sim \mathcal{N}(x|m_1, \sigma^2)$ and class 2 has the distribution $p(x|c = 2) \sim \mathcal{N}(x|m_2, \sigma^2)$. The prior probabilities of each class are $p(c = 1) = p(c = 2) = 1/2$. Show that the posterior probability $p(c = 1|x)$ is of the form*

$$p(c = 1|x) = \frac{1}{1 + \exp -(ax + b)} \tag{20.9.2}$$

and determine a and b in terms of m_1, m_2 and σ^2.

Exercise 202. *Derive the optimal EM update for fitting a mixture of Gaussians under the constraint that the covariances are diagonal.*

Exercise 203. *Consider a mixture of K isotropic Gaussians, each with the same covariance, $\mathbf{S}_i = \sigma^2\mathbf{I}$. In the limit $\sigma^2 \to 0$ show that the EM algorithm tends to the K-means clustering algorithm.*

Exercise 204. *Consider the term*

$$\sum_{n=1}^{N} \langle \log p(h) \rangle_{p^{old}(h|v^n)} \tag{20.9.3}$$

We wish to optimise the above with respect to the distribution $p(h)$. This can be achieved by defining the Lagrangian

$$L = \sum_{n=1}^{N} \langle \log p(h) \rangle_{p^{old}(h|v^n)} + \lambda \left(1 - \sum_h p(h)\right) \tag{20.9.4}$$

By differentiating the Lagrangian with respect to $p(h)$ and using the normalisation constraint $\sum_h p(h) = 1$, show that, optimally

$$p(h) = \frac{1}{N} \sum_{n=1}^{N} p^{old}(h|v^n) \tag{20.9.5}$$

Exercise 205. *We showed that fitting an unconstrained mixture of Gaussians using Maximum Likelihood is problematic since, by placing one of the Gaussians over a datapoints and letting the covariance determinant go to zero, we obtain an infinite likelihood. In contrast, when fitting a single Gaussian $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ to i.i.d. data $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^N$ show that the Maximum Likelihood optimum for $\boldsymbol{\Sigma}$ has non-zero determinant, and that the optimal likelihood remains finite.*

Exercise 206. *Modify* `GMMem.m` *suitably so that it can deal with the semi-supervised scenario in which the class (mixture component h) of some of the observations v is known.*

CHAPTER 21

Latent Linear Models

21.1 Factor Analysis

In chapter(15) we discussed Principal Components Analysis which forms lower dimensional representations of data based on assuming that the data lies close to a hyperplane. Here we describe a related probabilistic model. The potential benefit is that given a probabilistic description, uncertainty is incorporated into the model and extensions to Bayesian methods can be envisaged. Any probabilistic model may also be used as a component of a larger more complex model, such as a mixture model, enabling natural generalisations.

We use $\mathbf{v}$ to describe a data vector to emphasise that this is a visible (observable) quantity. The dataset is then given by a set of vectors,

$$\mathcal{V} = \left\{\mathbf{v}^1, \ldots, \mathbf{v}^N\right\} \tag{21.1.1}$$

where $\dim(\mathbf{v}) = D$. Our interest is to find a lower dimensional probabilistic description of this data. If data lies close to a H-dimensional hyperplane we may accurately approximate each datapoint by a low H-dimensional coordinate system. In general, datapoints will not lie exactly on the hyperplane and we model this discrepancy with Gaussian noise. Mathematically, the FA model generates an observation $\mathbf{v}$ according to

$$\mathbf{v} = \mathbf{F}\mathbf{h} + \mathbf{c} + \boldsymbol{\epsilon} \tag{21.1.2}$$

where the noise $\boldsymbol{\epsilon}$ is Gaussian distributed,

$$\boldsymbol{\epsilon} \sim \mathcal{N}\left(\boldsymbol{\epsilon} | \mathbf{0}, \boldsymbol{\Psi}\right) \tag{21.1.3}$$

The constant bias $\mathbf{c}$ sets the origin of the coordinate system. The *factor loading* matrix $\mathbf{F}$ plays a similar role as the basis matrix in PCA, see section(15.2). Similarly, the hidden coordinates $\mathbf{h}$ plays the role of the components we used in section(15.2).

The difference between PCA and Factor Analysis is in the choice of $\boldsymbol{\Psi}$:

Probabilistic PCA

$$\boldsymbol{\Psi} = \sigma^2\mathbf{I} \tag{21.1.4}$$

Factor Analysis

$$\boldsymbol{\Psi} = \mathrm{diag}\left(\psi_1, \ldots, \psi_D\right) \tag{21.1.5}$$

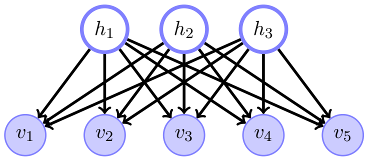


Figure 21.1: Factor Analysis. The visible vector variable $\mathbf{v}$ is related to the vector hidden variable $\mathbf{h}$ by a linear mapping, with independent additive Gaussian noise on each visible variable. The prior on the hidden variable may be taken to be an isotropic Gaussian, thus being independent across its components.

A probabilistic description

From equation (21.1.2) and equation (21.1.3), given $\mathbf{h}$, the data is Gaussian distributed with mean $\mathbf{Fh}+\mathbf{c}$ and covariance $\mathbf{\Psi}$

$$p(\mathbf{v}|\mathbf{h}) = \mathcal{N}(\mathbf{v}|\mathbf{Fh}+\mathbf{c}, \mathbf{\Psi}) \propto e^{-\frac{1}{2}(\mathbf{v}-\mathbf{Fh}-\mathbf{c})^\mathsf{T}\mathbf{\Psi}^{-1}(\mathbf{v}-\mathbf{Fh}-\mathbf{c})} \tag{21.1.6}$$

To complete the model, we need to specify the hidden distribution $p(\mathbf{h})$. A convenient choice is a Gaussian

$$p(\mathbf{h}) = \mathcal{N}(\mathbf{h}|\mathbf{0}, \mathbf{I}) \propto e^{-\mathbf{h}^\mathsf{T}\mathbf{h}/2} \tag{21.1.7}$$

Under this prior the coordinates $\mathbf{h}$ will be preferentially concentrated around values close to $\mathbf{0}$. If we were to sample from such a $p(\mathbf{h})$ and then draw a value for $\mathbf{v}$ using $p(\mathbf{v}|\mathbf{h})$, the sampled $\mathbf{v}$ vectors would produce a saucer or 'pancake' of points in the $\mathbf{v}$ space. Note that a correlated Gaussian prior $p(\mathbf{h}) = \mathcal{N}(\mathbf{h}|\mathbf{0}, \mathbf{\Sigma}_H)$ adds nothing to the complexity of the model since this can be absorbed into transforming the factors $\mathbf{F}$, exercise(209). Since $\mathbf{v}$ is linearly related to $\mathbf{h}$ through equation (21.1.2) and both $\boldsymbol{\epsilon}$ and $\mathbf{h}$ are Gaussian, then $\mathbf{v}$ is Gaussian distributed $p(\mathbf{v})$:

$$p(\mathbf{v}) = \int p(\mathbf{v}|\mathbf{h})\, p(\mathbf{h})\, d\mathbf{h} = \mathcal{N}\left(\mathbf{v}\middle|\mathbf{c}, \mathbf{F}\mathbf{F}^\mathsf{T} + \mathbf{\Psi}\right) \tag{21.1.8}$$

Invariance of the likelihood under Factor rotation

Since the matrix $\mathbf{F}$ only appears in the final model $p(\mathbf{v})$ through $\mathbf{F}\mathbf{F}^\mathsf{T} + \mathbf{\Psi}$, the likelihood is unchanged if we rotate $\mathbf{F}$ using $\mathbf{FR}$, with $\mathbf{R}\mathbf{R}^\mathsf{T} = \mathbf{I}$:

$$\mathbf{FR}(\mathbf{FR})^\mathsf{T} + \mathbf{\Psi} = \mathbf{F}\mathbf{R}\mathbf{R}^\mathsf{T}\mathbf{F}^\mathsf{T} + \mathbf{\Psi} = \mathbf{F}\mathbf{F}^\mathsf{T} + \mathbf{\Psi} \tag{21.1.9}$$

The solution space for $\mathbf{F}$ is therefore not unique – we can arbitrarily rotate the matrix $\mathbf{F}$ and produce an equally likely model of the data. Some care is therefore required when interpreting the entries of $\mathbf{F}$. *Varimax* is a technique to provide a 'more interpretable' $\mathbf{F}$ using a suitable rotation matrix $\mathbf{R}$. The aim is to produce a matrix $\mathbf{F}'$ for which each column has only a small number of large values. This is results in a non-linear optimisation problem and needs to be solved numerically. See [182] for details.

21.1.1 Finding the optimal bias

For a set of data, $\mathcal{V}$ and using the usual i.i.d. assumption, the log likelihood is

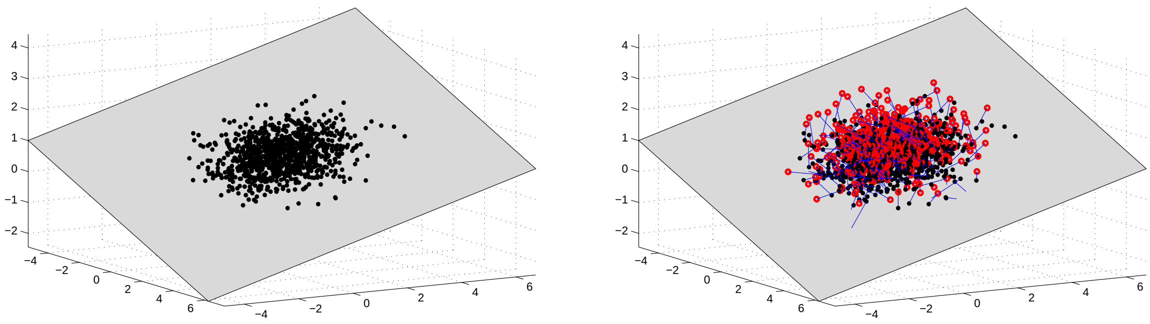
$$\log p(\mathcal{V}|\mathbf{F}, \mathbf{\Psi}) = \sum_{n=1}^{N} \log p(\mathbf{v}^n) = -\frac{1}{2}\sum_{n} (\mathbf{v}^n - \mathbf{c})^\mathsf{T} \mathbf{\Sigma}_D^{-1} (\mathbf{v}^n - \mathbf{c}) - \frac{N}{2} \log\det(2\pi\mathbf{\Sigma}_D) \tag{21.1.10}$$

where

$$\mathbf{\Sigma}_D \equiv \mathbf{F}\mathbf{F}^\mathsf{T} + \mathbf{\Psi} \tag{21.1.11}$$

Differentiating equation (21.1.10) with respect to $\mathbf{c}$ and equating to zero, we arrive at the Maximum Likelihood optimal setting that the 'bias' $\mathbf{c}$ is the mean of the data,

$$\mathbf{c} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{v}^n \equiv \bar{\mathbf{v}} \tag{21.1.12}$$



(a) (b)

Figure 21.2: Factor Analysis: 1000 points generated from the model. **(a)**: 1000 latent two-dimensional points $\mathbf{h}^n$ sampled from $\mathcal{N}(\mathbf{h}|\mathbf{0},\mathbf{I})$. These are transformed to a point on the three-dimensional plane by $\mathbf{x}_0^n = \mathbf{c} + \mathbf{F}\mathbf{h}^n$. The covariance of $\mathbf{x}_0$ is degenerate, with covariance matrix $\mathbf{F}\mathbf{F}^\mathsf{T}$. **(b)**: For each point $\mathbf{x}_0^n$ on the plane a random noise vector is drawn from $\mathcal{N}(\boldsymbol{\epsilon}|\mathbf{0},\boldsymbol{\Psi})$ and added to the in-plane vector to form a sample $\mathbf{x}^n$, plotted in red. The distribution of points forms a 'pancake' in space. Points 'underneath' the plane are not shown.

We will use this setting throughout. With this, the log likelihood can be written

$$\log p(\mathcal{V}|\mathbf{F},\boldsymbol{\Psi}) = -\frac{N}{2}\left(\text{trace}\left(\boldsymbol{\Sigma}_D^{-1}\mathbf{S}\right) + \log\det\left(2\pi\boldsymbol{\Sigma}_D\right)\right) \tag{21.1.13}$$

where $\mathbf{S}$ is the sample covariance matrix

$$\mathbf{S} = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{v}-\bar{\mathbf{v}})(\mathbf{v}-\bar{\mathbf{v}})^\mathsf{T} \tag{21.1.14}$$

21.2 Factor Analysis : Maximum Likelihood

We now specialise to the assumption that $\boldsymbol{\Psi} = \text{diag}(\psi_1,\ldots,\psi_D)$. We will consider two well-known methods for learning the factor loadings $\mathbf{F}$: a 'direct' approach[1], section(21.2.1) and an EM approach, section(21.2.2).

21.2.1 Direct likelihood optimisation

Optimal F for fixed Ψ

To find the maximum likelihood setting of $\mathbf{F}$ we differentiate the log likelihood equation (21.1.10) with respect to $\mathbf{F}$ and equate to zero. This gives

$$\mathbf{0} = -\text{trace}\left(\boldsymbol{\Sigma}_D^{-1}\partial_\mathbf{F}\boldsymbol{\Sigma}\right) + \text{trace}\left(\boldsymbol{\Sigma}_D^{-1}(\partial_\mathbf{F}\boldsymbol{\Sigma}_D)\boldsymbol{\Sigma}_D^{-1}\mathbf{S}\right) \tag{21.2.1}$$

Using $\partial_\mathbf{F}(\mathbf{F}\mathbf{F}^\mathsf{T}) = \mathbf{F}(\partial_\mathbf{F}\mathbf{F}^\mathsf{T}) + (\partial_\mathbf{F}\mathbf{F})\mathbf{F}^\mathsf{T}$, a stationary point is given when

$$\boldsymbol{\Sigma}_D^{-1}\mathbf{F} = \boldsymbol{\Sigma}_D^{-1}\mathbf{S}\boldsymbol{\Sigma}_D^{-1}\mathbf{F} \tag{21.2.2}$$

Since $\boldsymbol{\Sigma}_D$ is invertible

$$\mathbf{F} = \mathbf{S}\boldsymbol{\Sigma}_D^{-1}\mathbf{F} \tag{21.2.3}$$

Using the definition of $\boldsymbol{\Sigma}_D$, equation (21.1.11), one can rewrite $\boldsymbol{\Sigma}_D^{-1}\mathbf{F}$ as (see exercise(210))

$$\boldsymbol{\Sigma}_D^{-1}\mathbf{F} = \boldsymbol{\Psi}^{-1}\mathbf{F}\left(\mathbf{I} + \mathbf{F}^\mathsf{T}\boldsymbol{\Psi}^{-1}\mathbf{F}\right)^{-1} \tag{21.2.4}$$

[1] The presentation here follows closely that of [298].

Plugging this into the zero derivative condition, equation (21.2.3) can be rearranged to give

$$\mathbf{F}\left(\mathbf{I}+\mathbf{F}^{\mathsf{T}}\mathbf{\Psi}^{-1}\mathbf{F}\right)=\mathbf{S}\mathbf{\Psi}^{-1}\mathbf{F} \tag{21.2.5}$$

Using the reparameterisations

$$\tilde{\mathbf{F}}\equiv\mathbf{\Psi}^{-\frac{1}{2}}\mathbf{F}, \qquad \tilde{\mathbf{S}}=\mathbf{\Psi}^{-\frac{1}{2}}\mathbf{S}\mathbf{\Psi}^{-\frac{1}{2}} \tag{21.2.6}$$

equation (21.2.5) can be written in the 'isotropic' form

$$\tilde{\mathbf{F}}\left(\mathbf{I}+\tilde{\mathbf{F}}^{\mathsf{T}}\tilde{\mathbf{F}}\right)=\tilde{\mathbf{S}}\tilde{\mathbf{F}} \tag{21.2.7}$$

We assume that the factor matrix has a thin SVD decomposition

$$\tilde{\mathbf{F}}=\mathbf{U}_H\mathbf{L}\mathbf{V}^{\mathsf{T}} \tag{21.2.8}$$

where $\dim\mathbf{U}=D\times H$, $\dim\mathbf{L}=H\times H$, $\dim\mathbf{V}=H\times H$ and

$$\mathbf{U}_H^{\mathsf{T}}\mathbf{U}_H=\mathbf{I}_H, \qquad \mathbf{V}^{\mathsf{T}}\mathbf{V}=\mathbf{I}_H \tag{21.2.9}$$

and $\mathbf{L}=\mathrm{diag}\left(l_1,\ldots,l_H\right)$ are the singular values of $\tilde{\mathbf{F}}$. Plugging this assumption into equation (21.2.7) we obtain

$$\mathbf{U}_H\mathbf{L}\mathbf{V}^{\mathsf{T}}\left(\mathbf{I}_H+\mathbf{V}\mathbf{L}^2\mathbf{V}^{\mathsf{T}}\right)=\tilde{\mathbf{S}}\mathbf{U}_H\mathbf{L}\mathbf{V}^{\mathsf{T}} \tag{21.2.10}$$

which gives

$$\mathbf{U}_H\left(\mathbf{I}_H+\mathbf{L}^2\right)=\tilde{\mathbf{S}}\mathbf{U}_H, \qquad \mathbf{L}^2=\mathrm{diag}\left(l_1^2,\ldots,l_H^2\right) \tag{21.2.11}$$

Equation(21.2.11) is then an eigen-equation for $\mathbf{U}_H$. We can relate the form of the solution to the eigen-decomposition of $\tilde{\mathbf{S}}$

$$\tilde{\mathbf{S}}=\mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathsf{T}}, \qquad \mathbf{U}=\left[\mathbf{U}_H|\mathbf{U}_r\right] \tag{21.2.12}$$

where $\mathbf{U}_r$ are arbitrary additional columns chosen to complete $\mathbf{U}_H$ to form an orthogonal $\mathbf{U}$, $\mathbf{U}\mathbf{U}^{\mathsf{T}}=\mathbf{I}$. Using $\mathbf{\Lambda}=\mathrm{diag}\left(\lambda_1,\ldots,\lambda_D\right)$, equation (21.2.11) stipulates $1+l_i^2=\lambda_i$, or $l_i=\sqrt{\lambda_i-1}$. Given the solution for $\tilde{\mathbf{F}}$, the solution for $\mathbf{F}$ is found from equation (21.2.6). To determine the optimal λ_i we write the log likelihood in terms of the λ_i as follows. Using the new parameterisation,

$$\mathbf{\Sigma}_D=\mathbf{\Psi}^{\frac{1}{2}}\left(\tilde{\mathbf{F}}\tilde{\mathbf{F}}^{\mathsf{T}}+\mathbf{I}\right)\mathbf{\Psi}^{\frac{1}{2}} \tag{21.2.13}$$

and $\mathbf{S}=\mathbf{\Psi}^{\frac{1}{2}}\tilde{\mathbf{S}}\mathbf{\Psi}^{\frac{1}{2}}$, we have

$$\mathrm{trace}\left(\mathbf{\Sigma}_D^{-1}\mathbf{S}\right)=\mathrm{trace}\left(\left(\tilde{\mathbf{F}}\tilde{\mathbf{F}}^{\mathsf{T}}+\mathbf{I}_D\right)^{-1}\tilde{\mathbf{S}}\right) \tag{21.2.14}$$

The log likelihood equation (21.1.10) in this new parameterisation is

$$-\frac{2}{N}\log p(\mathcal{V}|\mathbf{F},\mathbf{\Psi})=\mathrm{trace}\left(\left(\mathbf{I}_D+\tilde{\mathbf{F}}\tilde{\mathbf{F}}^{\mathsf{T}}\right)^{-1}\tilde{\mathbf{S}}\right)+\log\det\left(\mathbf{I}_D+\tilde{\mathbf{F}}\tilde{\mathbf{F}}^{\mathsf{T}}\right)+\log\det\left(2\pi\mathbf{\Psi}\right) \tag{21.2.15}$$

Note that, using $\lambda_i=1+l_i^2$,

$$\mathbf{I}_D+\tilde{\mathbf{F}}\tilde{\mathbf{F}}^{\mathsf{T}}=\mathbf{I}_D+\mathbf{U}_H\mathbf{L}^2\mathbf{U}_H^{\mathsf{T}}=\mathbf{U}\mathrm{diag}\left(\lambda_1,\ldots,\lambda_H,1,\ldots,1\right)\mathbf{U}^{\mathsf{T}} \tag{21.2.16}$$

So that

$$\mathrm{trace}\left(\left(\mathbf{I}+\tilde{\mathbf{F}}\tilde{\mathbf{F}}^{\mathsf{T}}\right)^{-1}\tilde{\mathbf{S}}\right)=\sum_i\frac{\lambda_i}{\lambda_i'}, \qquad \lambda_i'=\begin{cases}\lambda_i & i\leq H\\ 1 & i>H\end{cases} \tag{21.2.17}$$

Similarly

$$\log\det\left(\mathbf{I}_D + \tilde{\mathbf{F}}\tilde{\mathbf{F}}^{\mathsf{T}}\right) = \sum_{i=1}^{H}\log\lambda_i \tag{21.2.18}$$

Using this we can write the log likelihood as a function of the eigenvalues (for fixed $\mathbf{\Psi}$) as

$$-\frac{2}{N}\log p(\mathcal{V}|\mathbf{F},\mathbf{\Psi}) = \sum_{i=1}^{H}\log\lambda_i + H + \sum_{i=H+1}^{D}\lambda_i + \log\det(2\pi\mathbf{\Psi}) \tag{21.2.19}$$

To maximise the likelihood we need to minimise the right hand side of the above. Since $\log\lambda < \lambda$ we should place the largest H eigenvalues in the $\sum_i \log\lambda_i$ term. A solution for fixed $\mathbf{\Psi}$ is therefore

$$\mathbf{F} = \mathbf{\Psi}^{\frac{1}{2}}\mathbf{U}_H\left(\mathbf{\Lambda}_H - \mathbf{I}_H\right)^{\frac{1}{2}}\mathbf{R} \tag{21.2.20}$$

where

$$\mathbf{\Lambda}_H \equiv \text{diag}\left(\lambda_1,\ldots,\lambda_H\right) \tag{21.2.21}$$

are the H largest eigenvalues of $\mathbf{\Psi}^{-\frac{1}{2}}\mathbf{S}\mathbf{\Psi}^{-\frac{1}{2}}$, with $\mathbf{U}_H$ being the matrix of the corresponding eigenvectors. $\mathbf{R}$ is an arbitrary orthogonal (rotation) matrix.

Alternatively, rather than finding the eigen-decomposition of $\mathbf{\Psi}^{-\frac{1}{2}}\mathbf{S}\mathbf{\Psi}^{-\frac{1}{2}}$ we can avoid forming the covariance matrix by considering the thin SVD decomposition of

$$\tilde{\mathbf{X}} = \frac{1}{\sqrt{N}}\mathbf{\Psi}^{-\frac{1}{2}}\mathbf{X} \tag{21.2.22}$$

where the data matrix is

$$\mathbf{X} \equiv \left[\mathbf{x}^1,\ldots,\mathbf{x}^N\right] \tag{21.2.23}$$

Given a thin decomposition

$$\tilde{\mathbf{X}} = \mathbf{U}_H\tilde{\mathbf{\Lambda}}\mathbf{V}^{\mathsf{T}} \tag{21.2.24}$$

we obtain the eigenvalues $\lambda_i = \tilde{\Lambda}_{ii}^2$. For $D > N$ this is convenient since the computational complexity using this SVD method is $O\left(\min(H,N)^3\right)$. When the matrix $\mathbf{X}$ is too large to store in memory, online SVD methods are available[49].

Finding the optimal $\mathbf{\Psi}$

The zero derivative of the log likelihood occurs when

$$\mathbf{\Psi} = \text{diag}\left(\mathbf{S} - \mathbf{F}\mathbf{F}^{\mathsf{T}}\right) \tag{21.2.25}$$

where $\mathbf{F}$ is given by equation (21.2.20). There is no closed form solution to equations(21.2.25, 21.2.20). A simple iterative scheme is to first guess values for the diagonal entries of $\mathbf{\Psi}$ and then find the optimal $\mathbf{F}$ using equation (21.2.20). Subsequently $\mathbf{\Psi}$ is updated using

$$\mathbf{\Psi}^{new} = \text{diag}\left(\mathbf{S} - \mathbf{F}\mathbf{F}^{\mathsf{T}}\right) \tag{21.2.26}$$

Updating $\mathbf{F}$ using equation (21.2.20) and $\mathbf{\Psi}$ using equation (21.2.26) is iterated until convergence.

Alternative schemes for updating the noise matrix $\mathbf{\Psi}$ can improve convergence considerably; for example updating only a single component of $\mathbf{\Psi}$ with the rest fixed can be achieved using a closed form expression[298].

21.2.2 Expectation Maximisation

A popular way to train Factor Analysis in Machine Learning is to use EM.

M-step

Here we assume that the bias $\mathbf{c}$ has been optimally set to the data mean $\bar{\mathbf{v}}$. As usual, we need to consider the energy which, neglecting constants, is

$$E(\mathbf{F}, \mathbf{\Psi}) = -\sum_{n=1}^{N} \left\langle \frac{1}{2} (\mathbf{d}^n - \mathbf{F}\mathbf{h})^\mathsf{T} \mathbf{\Psi}^{-1} (\mathbf{d}^n - \mathbf{F}\mathbf{h}) \right\rangle_{q(\mathbf{h}|\mathbf{v}^n)} - \frac{N}{2} \log\det(\mathbf{\Psi}) \tag{21.2.27}$$

where $\mathbf{d}^n \equiv \mathbf{v}^n - \bar{\mathbf{v}}$. The optimal variational distribution $q(\mathbf{h}|\mathbf{v}^n)$ is determined by the E-step below.

Maximising $E(\mathbf{F}, \mathbf{\Psi})$ with respect to $\mathbf{F}$ gives

$$\mathbf{F}^{new} = \mathbf{A}\mathbf{H}^{-1} \tag{21.2.28}$$

where

$$\mathbf{A} \equiv \frac{1}{N} \sum_n \mathbf{d}^n \langle \mathbf{h} \rangle^\mathsf{T}_{q(\mathbf{h}|\mathbf{v}^n)}, \qquad \mathbf{H} \equiv \frac{1}{N} \sum_n \left\langle \mathbf{h}\mathbf{h}^\mathsf{T} \right\rangle_{q(\mathbf{h}|\mathbf{v}^n)} \tag{21.2.29}$$

Finally

$$\mathbf{\Psi}^{new} = \frac{1}{N} \sum_n \text{diag}\left(\left\langle (\mathbf{d}^n - \mathbf{F}\mathbf{h})(\mathbf{d}^n - \mathbf{F}\mathbf{h})^\mathsf{T} \right\rangle_{q(\mathbf{h}|\mathbf{v}^n)} \right) = \text{diag}\left(\frac{1}{N} \sum_n \mathbf{d}^n (\mathbf{d}^n)^\mathsf{T} - 2\mathbf{F}\mathbf{A}^\mathsf{T} + \mathbf{F}\mathbf{H}\mathbf{F}^\mathsf{T} \right) \tag{21.2.30}$$

E-step

The above recursions depend on the statistics $\langle \mathbf{h} \rangle_{q(\mathbf{h}|\mathbf{v}^n)}$ and $\left\langle \mathbf{h}\mathbf{h}^\mathsf{T} \right\rangle_{q(\mathbf{h}|\mathbf{v}^n)}$. Using the EM optimal choice for the E-step we have

$$q(\mathbf{h}|\mathbf{v}^n) \propto p(\mathbf{v}^n|\mathbf{h}) p(\mathbf{h}) = \mathcal{N}(\mathbf{h}|\mathbf{m}^n, \mathbf{\Sigma}) \tag{21.2.31}$$

with

$$\mathbf{m}^n = \langle \mathbf{h} \rangle_{q(\mathbf{h}|\mathbf{v}^n)} = \left(\mathbf{I} + \mathbf{F}^\mathsf{T} \mathbf{\Psi}^{-1} \mathbf{F} \right)^{-1} \mathbf{F}^\mathsf{T} \mathbf{\Psi}^{-1} \mathbf{d}^n, \qquad \mathbf{\Sigma} = \left(\mathbf{I} + \mathbf{F}^\mathsf{T} \mathbf{\Psi}^{-1} \mathbf{F} \right)^{-1} \tag{21.2.32}$$

Using these results we can express the statistics in equation (21.2.29) as

$$\mathbf{H} = \mathbf{\Sigma} + \frac{1}{N} \sum_n \mathbf{m}^n (\mathbf{m}^n)^\mathsf{T} \tag{21.2.33}$$

Equations (21.2.28,21.2.30,21.2.32) are iterated till convergence. As for any EM algorithm, the likelihood equation (21.1.10) (under the diagonal constraint on $\mathbf{\Psi}$) increases at each iteration.

Convergence using this EM technique can be slower than that of the direct eigen-approach of section(21.2.1) and most 'commercial' implementations avoid EM for this reason. Provided however that a reasonable initialisation is used, the performance of the two training algorithms can be similar. A useful initialisation is to use PCA and then set $\mathbf{F}$ to the principal directions.

Mixtures of FA

An advantage of probabilistic models is that they may be used as components in more complex models, such as mixtures of FA and related models, [272]. Training can then be achieved using EM or gradient based approaches. Bayesian extensions are clearly of interest, although typically intractable and need to be addressed using approximate methods, for example [95, 175, 106].

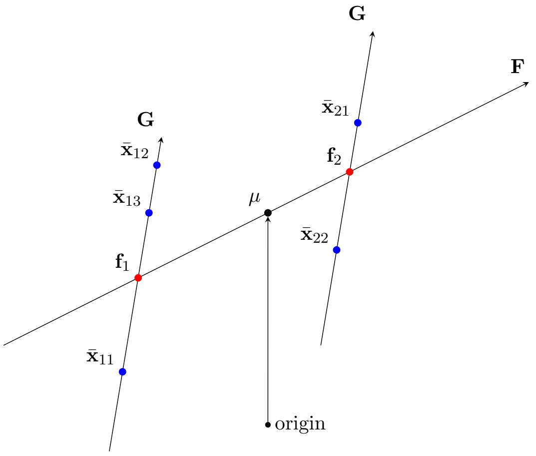


Figure 21.3: Latent Identity Model. The mean $\boldsymbol{\mu}$ represents the mean of the faces. The subspace $\mathbf{F}$ represents the directions of variation of different faces so that $\mathbf{f}_1 = \boldsymbol{\mu} + \mathbf{F}\mathbf{h}_1$ is a mean face for individual 1, and similarly for $\mathbf{f}_2 = \boldsymbol{\mu} + \mathbf{F}\mathbf{h}_2$. The subspace G denotes the directions of variability for any individual face, caused by pose, lighting *etc.* This variability is assumed the same for each person. A particular mean face is then given by the mean face of the person plus pose/illumination variation, for example $\bar{\mathbf{x}}_{12} = \mathbf{f}_1 + \mathbf{G}\mathbf{w}_{12}$. A sample face is then given by a mean face $\bar{\mathbf{x}}_{ij}$ plus Gaussian noise from $\mathcal{N}(\boldsymbol{\epsilon}_{ij}|\mathbf{0}, \boldsymbol{\Sigma})$.

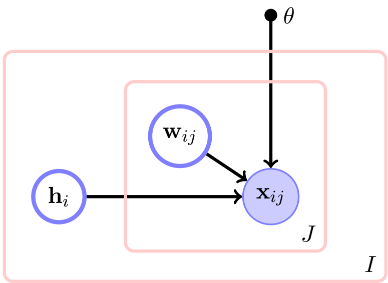


Figure 21.4: The j^{th} image of the i^{th} person, $\mathbf{x}_{ij}$, is modelled using a linear latent model with parameters θ.

21.3 Interlude: Modelling faces

Factor Analysis has widespread application in statistics and machine learning. As an inventive application of FA, highlighting the probabilistic nature of the model, we describe a face modelling technique that has as its heart a latent linear model[224]. Consider a gallery of face images $\mathcal{X} = \{\mathbf{x}_{ij}, i = 1, \ldots, I; j = 1, \ldots, J\}$ so that the vector $\mathbf{x}_{ij}$ represents the j^{th} image of the i^{th} person. As a latent linear model of faces we consider

$$\mathbf{x}_{ij} = \boldsymbol{\mu} + \mathbf{F}\mathbf{h}_i + \mathbf{G}\mathbf{w}_{ij} + \boldsymbol{\epsilon}_{ij} \tag{21.3.1}$$

Here $\mathbf{F}$ ($\dim \mathbf{F} = D \times F$) is used to model variability between people, and $\mathbf{G}$ ($\dim \mathbf{G} = D \times G$) models variability related to pose, illumination *etc.* within the different images of each person. The contribution

$$\mathbf{f}_i \equiv \boldsymbol{\mu} + \mathbf{F}\mathbf{h}_i \tag{21.3.2}$$

accounts for variability between different people, being constant for individual i. For fixed i, the contribution

$$\mathbf{G}\mathbf{w}_{ij} + \boldsymbol{\epsilon}_{ij} \tag{21.3.3}$$

accounts for the variability over the images of person i, explaining why two images of the same person do not look identical. See fig(21.3) for a graphical representation.

As a probabilistic linear latent variable model, we have for an image $\mathbf{x}_{ij}$:

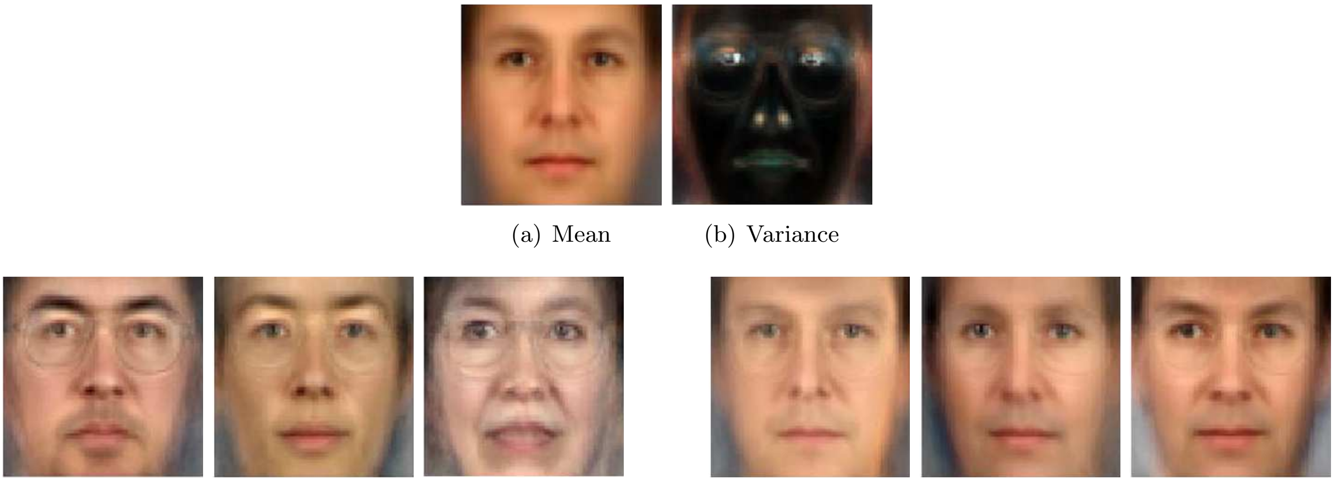
$$p(\mathbf{x}_{ij}|\mathbf{h}_i, \mathbf{w}_{ij}, \theta) = \mathcal{N}(\mathbf{x}_{ij}|\boldsymbol{\mu} + \mathbf{F}\mathbf{h}_i + \mathbf{G}\mathbf{w}_{ij}, \boldsymbol{\Sigma}) \tag{21.3.4}$$

$$p(\mathbf{h}_i) = \mathcal{N}(\mathbf{h}_i|\mathbf{0}, \mathbf{I}), \qquad p(\mathbf{w}_{ij}) = \mathcal{N}(\mathbf{w}_{ij}|\mathbf{0}, \mathbf{I}) \tag{21.3.5}$$

The parameters are $\theta = \{\mathbf{F}, \mathbf{G}, \boldsymbol{\mu}, \boldsymbol{\Sigma}\}$.

For the collection of images, assuming i.i.d. data,

$$p(\mathcal{X}, \mathbf{w}, \mathbf{h}|\theta) = \prod_{i=1}^{I} \left\{ \prod_{j=1}^{J} p(\mathbf{x}_{ij}|\mathbf{h}_i, \mathbf{w}_{ij}, \theta) p(\mathbf{w}_{ij}) \right\} p(\mathbf{h}_i) \tag{21.3.6}$$



(a) Mean (b) Variance

(c) (d) (e) (f) (g) (h)

Figure 21.5: Latent Identity Model of face images. Each image is represented by a $70 \times 70 \times 3$ vector (the 3 comes from the RGB colour coding). There are $I = 195$ individuals in the database and $J = 4$ images per person. **(a)**: Mean of the data. **(b)**: Per pixel standard deviation – black is low, white is high. **(c,d,e)**: Three directions from the between-individual subspace $\mathbf{F}$. **(f,g,h)**: Three samples from the model with $\mathbf{h}$ fixed and drawing randomly from $\mathbf{w}$ in the within-individual subspace $\mathbf{G}$. Reproduced from [224].

for which the graphical model is depicted in fig(21.4). The task of learning is then to maximise the likelihood

$$p(\mathcal{X}|\theta) = \int_{\mathbf{w},\mathbf{h}} p(\mathcal{X}, \mathbf{w}, \mathbf{h}|\theta) \tag{21.3.7}$$

This model can be seem as a constrained version of Factor Analysis by using stacked vectors (here for only a single individual, $I = 1$)

$$\begin{pmatrix} \mathbf{x}_{11} \\ \mathbf{x}_{12} \\ \vdots \\ \mathbf{x}_{1J} \end{pmatrix} = \begin{pmatrix} \boldsymbol{\mu} \\ \boldsymbol{\mu} \\ \vdots \\ \boldsymbol{\mu} \end{pmatrix} + \begin{pmatrix} \mathbf{F} & \mathbf{G} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{F} & \mathbf{0} & \mathbf{G} & \dots & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{F} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{G} \end{pmatrix} \begin{pmatrix} \mathbf{h}_1 \\ \mathbf{w}_{11} \\ \mathbf{w}_{12} \\ \vdots \\ \mathbf{w}_{1J} \end{pmatrix} + \begin{pmatrix} \boldsymbol{\epsilon}_{11} \\ \boldsymbol{\epsilon}_{12} \\ \vdots \\ \boldsymbol{\epsilon}_{1J} \end{pmatrix} \tag{21.3.8}$$

The generalisation to multiple individuals $I > 1$ is straightforward. The model can be trained using either a constrained form of the direct method, or EM as described in [224]. Example images from the trained model are presented in fig(21.5).

Recognition

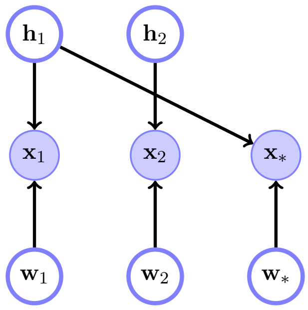
In closed set face recognition a new 'probe' face $\mathbf{x}_*$ is to be matched to a person n in the gallery of training faces. In model $\mathcal{M}_n$ the n^{th} gallery face is forced to share its latent identity variable $\mathbf{h}_n$ with the test face, indicating that these faces belong to the same person[2]. Assuming a single exemplar per person ($J = 1$),

$$p(\mathbf{x}_1, \dots, \mathbf{x}_I, \mathbf{x}_*|\mathcal{M}_n) = p(\mathbf{x}_n, \mathbf{x}_*) \prod_{i=1, i\neq n}^{I} p(\mathbf{x}_i) \tag{21.3.9}$$

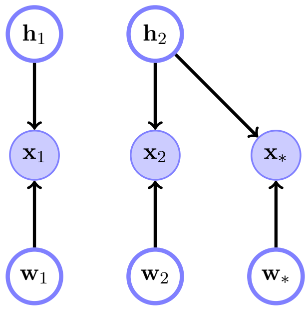
Bayes' rule then gives the posterior class assignment

$$p(\mathcal{M}_n|\mathbf{x}_1, \dots, \mathbf{x}_I, \mathbf{x}_*) \propto p(\mathbf{x}_1, \dots, \mathbf{x}_I, \mathbf{x}_*|\mathcal{M}_n) p(\mathcal{M}_n) \tag{21.3.10}$$

[2]This is analogous to Bayesian outcome analysis in section(13.5) in which the hypotheses assume that either the errors were generated from the same or a different model.



(a) $\mathcal{M}_1$



(b) $\mathcal{M}_2$

Figure 21.6: Face recognition model (depicted only for a single examplar per person, $J = 1$). **(a)**: In model $\mathcal{M}_1$ the test image (or 'probe') $\mathbf{x}_*$ is assumed to be from person 1, albeit with a different pose/illumination. **(b)**: For model $\mathcal{M}_2$ the test image is assumed to be from person 2. One calculates $p(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_*|\mathcal{M}_1)$ and $p(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_*|\mathcal{M}_2)$ and then uses Bayes' rule to infer which person the test image $\mathbf{x}_*$ most likely belongs.

For a uniform prior, the term $p(\mathcal{M}_n)$ is constant and can be neglected. The quantities we require above are given by

$$p(\mathbf{x}_n) = \int_{\mathbf{h}_n \mathbf{w}_n} p(\mathbf{x}_n, \mathbf{h}_n, \mathbf{w}_n), \qquad p(\mathbf{x}_*) = \int_{\mathbf{h}_* \mathbf{w}_*} p(\mathbf{x}_*, \mathbf{h}_*, \mathbf{w}_*) \tag{21.3.11}$$

$$p(\mathbf{x}_n, \mathbf{x}_*) = \int_{\mathbf{h}_n \mathbf{w}_n, \mathbf{w}_*} p(\mathbf{x}_n, \mathbf{x}_*, \mathbf{h}_n, \mathbf{w}_n, \mathbf{w}_*) \tag{21.3.12}$$

where $p(\mathbf{x}, \mathbf{h}, \mathbf{w})$ is obtained from equation (21.3.6) (where we assume the parameters θ are fixed, having been learned using Maximum Likelihood). Note that in equation (21.3.12) we do not introduce $\mathbf{h}_*$ since both $\mathbf{x}_n$ and $\mathbf{x}_*$ are assumed to be generated from the same person with the latent identity $\mathbf{h}_n$. These marginal probabilities are straightforward to derive since they are marginals of Gaussians.

In practice, the best results are obtained using a between-individual subspace dimension F and within-individual subspace dimension G both equal to 128. This model has performance competitive with the state-of-the-art[224]. A benefit of the probabilistic model is that the extension to mixtures of this model is essentially straightforward, which boosts performance further[224]. Related models can also be used for the 'open set' face recognition problem in which the probe face may or may not belong to one one of the individuals in the database[224].

21.4 Probabilistic Principal Components Analysis

PPCA corresponds to Factor Analysis under the restriction $\mathbf{\Psi} = \sigma^2 \mathbf{I}_D$. Plugging this assumption into the direct optimisation solution equation (21.2.20) gives

$$\mathbf{F} = \sigma \mathbf{U}_H \left(\mathbf{\Lambda}_H - \mathbf{I}_H\right)^{\frac{1}{2}} \mathbf{R} \tag{21.4.1}$$

where the eigenvalues (diagonal entries of $\mathbf{\Lambda}_H$) and corresponding eigenvectors (columns of $\mathbf{U}_H$) are the largest eigenvalues of $\sigma^{-2}\mathbf{S}$. Since the eigenvalues of $\sigma^{-2}\mathbf{S}$ are those of $\mathbf{S}$ simply scaled by σ^{-2} (and the eigenvectors are unchanged), we can equivalently write

$$\mathbf{F} = \mathbf{U}_H \left(\mathbf{\Lambda}_H - \sigma^2 \mathbf{I}_H\right)^{\frac{1}{2}} \mathbf{R} \tag{21.4.2}$$

where $\mathbf{R}$ is an arbitrary orthogonal matrix, $\mathbf{R}^\mathsf{T}\mathbf{R} = \mathbf{I}$ and $\mathbf{U}_H$, $\mathbf{\Lambda}_H$ are the eigenvectors and corresponding eigenvalues of the sample covariance $\mathbf{S}$. Classical PCA is recovered in the limit $\sigma^2 \to 0$. Note that for a full correspondence with PCA, one needs to set $\mathbf{R} = \mathbf{I}$, which points $\mathbf{F}$ along the principal directions.

Optimal σ^2

A particular convenience of PPCA is that the optimal noise σ^2 can be found immediately. We order the eigenvalues of $\mathbf{S}$ so that $\lambda_1 \geq \lambda_2, ... \geq \lambda_D$. In equation (21.2.19) an expression for the log likelihood is

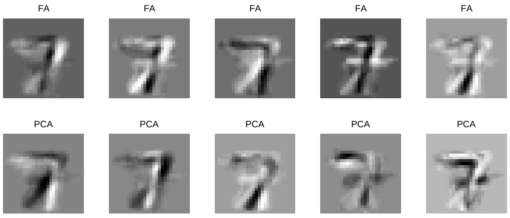


Figure 21.7: For a 5 hidden unit model, here are plotted the results of training PPCA and FA on 100 examples of the handwritten digit seven. The top row contains the 5 Factor Analysis factors and the bottom row the 5 largest eigenvectors from PPCA are plotted.

given in which the eigenvalues are those $\sigma^{-2}\mathbf{S}$. On replacing λ_i with λ_i/σ^2 we can therefore write an explicit expression for the log likelihood in terms of σ^2 and the eigenvalues of $\mathbf{A}$,

$$L(\sigma^2) = -\frac{N}{2}\left(D\log(2\pi) + \sum_{i=1}^{H}\log\lambda_i + \frac{1}{\sigma^2}\sum_{i=H+1}^{D}\lambda_i + (D-H)\log\sigma^2 + H\right) \tag{21.4.3}$$

By differentiating $L(\sigma^2)$ and equating to zero, the Maximum Likelihood optimal setting for σ^2 is

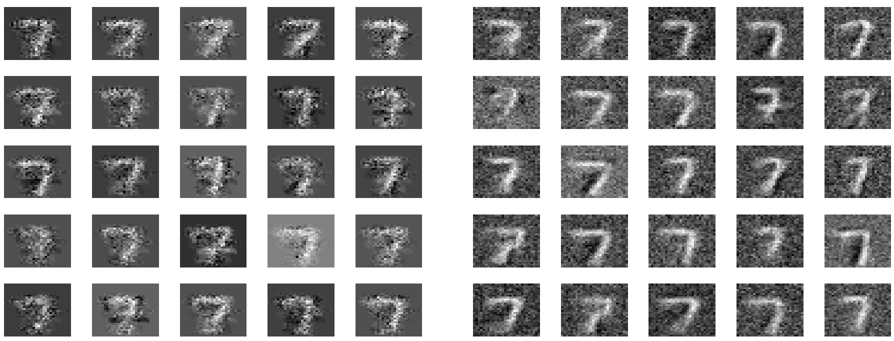
$$\sigma^2 = \frac{1}{D-H}\sum_{j=H+1}^{D}\lambda_j \tag{21.4.4}$$

In summary PPCA is obtained by taking the principal eigenvalues and corresponding eigenvectors of the sample covariance matrix $\mathbf{S}$, and setting the variance by equation (21.4.4). The single-shot training nature of PPCA makes it an attractive algorithm and also gives a useful initialisation for $\mathbf{\Psi}$ in Factor Analysis.

Example 92 (A Comparision of FA and PPCA)**.** We trained both PPCA and FA to model handwritten digits of the number 7. From a database of 100 such images, we fitted both PPCA and FA (100 iterations of EM in each case) using 5 hidden units. The learned factors for these models are in fig(21.7). To get a feeling for how well each of these models the data, we drew 25 samples from each model, as given in fig(21.8a). Compared with PPCA, in FA the individual noise on each visible variable enables a cleaner representation of the regions of zero sample variance.

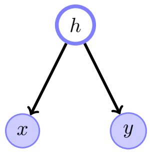
21.5 Canonical Correlation Analysis and Factor Analysis

Here we outline how CCA, as discussed in section(15.8) is related to a constrained form of FA. As a brief reminder, CCA considers two spaces X and Y. X might represent an audio sequence of a person speaking and Y the corresponding video sequence of the face of the person speaking. The two streams of data are dependent, since we would expect the parts around the mouth region to be correlated with the speech signal. The aim in CCA is to find a low dimensional representation that explains the correlation between the X and Y spaces.



(a) Factor Analysis (b) PPCA

Figure 21.8: **(a)**: 25 samples from the learned FA model. Note how the noise variance depends on the pixel, being zero for pixels on the boundary of the image. **(b)**: 25 samples from the learned PPCA model.



Canonical Correlation Analysis. CCA corresponds to the latent variable model in which a common latent variable generates both the observed x and y variables. This is therefore a formed of constrained Factor Analysis.

A model that achieves a similar effect to CCA, is to use a latent factor h to underlie the data in both the X and Y spaces. That is

$$p(\mathbf{x}, \mathbf{y}) = \int p(\mathbf{x}|h)p(\mathbf{y}|h)p(h)dh \tag{21.5.1}$$

where

$$p(\mathbf{x}|h) = \mathcal{N}(\mathbf{x}|h\mathbf{a}, \mathbf{\Psi}_x), \qquad p(\mathbf{y}|h) = \mathcal{N}(\mathbf{y}|h\mathbf{b}, \mathbf{\Psi}_y), \qquad p(h) = \mathcal{N}(h|0, 1) \tag{21.5.2}$$

We can express equation (21.5.2) as a form of Factor Analysis by writing

$$\begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \begin{pmatrix} \mathbf{a} \\ \mathbf{b} \end{pmatrix} h + \begin{pmatrix} \boldsymbol{\epsilon}_x \\ \boldsymbol{\epsilon}_y \end{pmatrix}, \qquad \boldsymbol{\epsilon}_x \sim \mathcal{N}(\boldsymbol{\epsilon}_x|\mathbf{0}, \mathbf{\Psi}_x), \boldsymbol{\epsilon}_y \sim \mathcal{N}(\boldsymbol{\epsilon}_y|\mathbf{0}, \mathbf{\Psi}_y) \tag{21.5.3}$$

By using the stacked vectors

$$\mathbf{z} = \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix}, \qquad \mathbf{f} = \begin{pmatrix} \mathbf{a} \\ \mathbf{b} \end{pmatrix}, \tag{21.5.4}$$

and integrating out the latent variable h, we obtain

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{\Sigma}), \qquad \mathbf{\Sigma} = \mathbf{f}\mathbf{f}^{\mathsf{T}} + \mathbf{\Psi}, \qquad \mathbf{\Psi} = \begin{pmatrix} \mathbf{\Psi}_x & \mathbf{0} \\ \mathbf{0} & \mathbf{\Psi}_y \end{pmatrix} \tag{21.5.5}$$

From the FA results (21.2.5) the optimal $\mathbf{f}$ is given by

$$\mathbf{f}\left(1 + \mathbf{f}^{\mathsf{T}}\mathbf{\Psi}^{-1}\mathbf{f}\right) = \mathbf{S}\mathbf{\Psi}^{-1}\mathbf{f} \Rightarrow \mathbf{f} \propto \mathbf{S}\mathbf{\Psi}^{-1}\mathbf{f} \tag{21.5.6}$$

so that optimally $\mathbf{f}$ is given by the principal eigenvector of $\mathbf{S}\mathbf{\Psi}^{-1}$. By imposing $\mathbf{\Psi}_x = \sigma_x^2\mathbf{I}$, $\mathbf{\Psi}_y = \sigma_y^2\mathbf{I}$ the above equation can be expressed as the coupled equations

$$\mathbf{a} \propto \frac{1}{\sigma_x^2}\mathbf{S}_{xx}\mathbf{a} + \frac{1}{\sigma_y^2}\mathbf{S}_{xy}\mathbf{b} \tag{21.5.7}$$

$$\mathbf{b} \propto \frac{1}{\sigma_x^2}\mathbf{S}_{yx}\mathbf{a} + \frac{1}{\sigma_y^2}\mathbf{S}_{yy}\mathbf{b} \tag{21.5.8}$$

Eliminating $\mathbf{b}$ we have, for an arbitrary proportionality constant γ,

$$\left(\mathbf{I} - \frac{\gamma}{\sigma_x^2}\mathbf{S}_{xx}\right)\mathbf{a} = \frac{\gamma^2}{\sigma_x^2\sigma_y^2}\mathbf{S}_{xy}\left(\mathbf{I} - \frac{\gamma}{\sigma_y^2}\mathbf{S}_{yy}\right)^{-1}\mathbf{S}_{yx}\mathbf{a} \tag{21.5.9}$$

In the limit $\sigma_x^2, \sigma_y^2 \to 0$, this tends to the zero derivative condition equation (15.8.8) so that CCA can be seen as in fact a form of FA (see [12] for a more thorough correspondence). A benefit of viewing CCA in this manner is that extensions to using more than a single latent dimension H become clear, see exercise(208).

As we've indicated, CCA corresponds to training a form of FA by maximising the joint likelihood $p(\mathbf{x}, \mathbf{y}|\mathbf{w}, \mathbf{u})$. Training based on the maximising the conditional $p(\mathbf{y}|\mathbf{x}, \mathbf{w}, \mathbf{u})$ corresponds to a special case of a technique called *Partial Least Squares*, see for example [77]. This correspondence is left as an exercise for the interested reader.

Extending FA to kernel variants is not straightforward since under replacing $\mathbf{x}$ with a non-linear mapping $\boldsymbol{\phi}(\mathbf{x})$, normalising the expression $e^{-(\boldsymbol{\phi}(\mathbf{x}) - \mathbf{w}h)^2}$ is in general intractable.

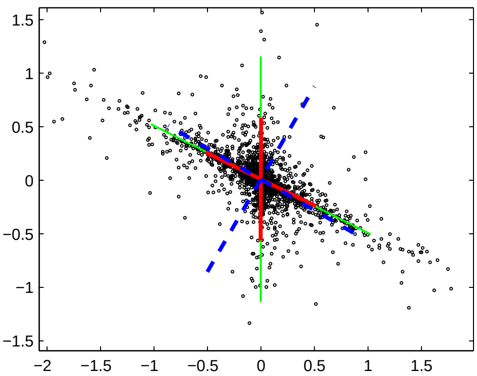


Figure 21.9: Latent data is sampled from the prior $p(x_i) \propto \exp(-5\sqrt{|x_i|})$ with the mixing matrix $\mathbf{A}$ shown in green to create the observed two dimensional vectors $\mathbf{y} = \mathbf{Ax}$. The red lines are the mixing matrix estimated by `ica.m` based on the observations. For comparison, PCA produces the blue (dashed) components. Note that the components have been scaled to improve visualisation. As expected, PCA finds the orthogonal directions of maximal variation. ICA however, correctly estimates the directions in which the components were independently generated. See `demoICA.m`.

21.6 Independent Components Analysis

Independent Components Analysis (*ICA*) is a linear decomposition of the data $\mathbf{v}$ in which the underlying latent variables $\mathbf{h}$ are independent[218, 137]. In other words, we seek a linear coordinate system in which the coordinates are independent. This can give rise to very different representations than PCA, see fig(21.9). Such independent coordinate systems arguably form a natural representation of the data. From a probabilistic viewpoint, we have

$$p(\mathbf{v}, \mathbf{h}|\mathbf{A}) = p(\mathbf{v}|\mathbf{h}, \mathbf{A}) \prod_i p(h_i) \tag{21.6.1}$$

In ICA it is common to assume that the observations are linearly related to the latent variables $\mathbf{h}$. For technical reasons, the most convenient practical choice is to use[3]

$$\mathbf{v} = \mathbf{Ah} \tag{21.6.2}$$

where $\mathbf{A}$ is a square *mixing matrix* so that the likelihood of an observation $\mathbf{v}$ is

$$p(\mathbf{v}) = \int p(\mathbf{v}|\mathbf{h}, \mathbf{A}) \prod_i p(h_i) d\mathbf{h} = \int \delta(\mathbf{v} - \mathbf{Ah}) \prod_i p(h_i) d\mathbf{h} = \frac{1}{|\det(\mathbf{A})|} \prod_i p([\mathbf{A}^{-1}\mathbf{v}]_i) \tag{21.6.3}$$

The underlying independence assumptions are then the same as for PPCA (in the limit of zero output noise). Below, however, we will choose a non-Gaussian prior $p(h_i)$.

For a given set of data $\mathcal{V} = (\mathbf{v}^1, \ldots, \mathbf{v}^N)$ and prior $p(h)$, our aim is to find $\mathbf{A}$. For i.i.d. data, the log likelihood is conveniently written in terms of $\mathbf{B} = \mathbf{A}^{-1}$,

$$L(\mathbf{B}) = N \log\det(\mathbf{B}) + \sum_n \sum_i \log p([\mathbf{Bv}^n]_i) \tag{21.6.4}$$

Note that for a Gaussian prior

$$p(h) \propto e^{-h^2} \tag{21.6.5}$$

the log likelihood becomes

$$L(\mathbf{B}) = N \log\det(\mathbf{B}) + \sum_n (\mathbf{v}^n)^\mathsf{T} \mathbf{B}^\mathsf{T}\mathbf{B}\mathbf{v}^n + \text{const.} \tag{21.6.6}$$

which is invariant with respect to an orthogonal rotation $\mathbf{B} \to \mathbf{RB}$, with $\mathbf{R}^\mathsf{T}\mathbf{R} = \mathbf{I}$. This means that for a Gaussian prior $p(h)$, we cannot estimate uniquely the mixing matrix. To break this rotational invariance we therefore need to use a non-Gaussian prior. Assuming we have a non-Gaussian prior $p(h)$, taking the derivative *w.r.t.* B_{ab} we obtain

$$\frac{\partial}{\partial B_{ab}} L = N A_{ba} + \sum_n \phi([\mathbf{Bv}]_a) v_b^n \tag{21.6.7}$$

[3]This treatment follows that presented in [180].

where

$$\phi(x) \equiv \frac{d}{dx} \log p(x) = \frac{1}{p(x)} \frac{d}{dx} p(x) \tag{21.6.8}$$

A simple gradient ascent learning rule for $\mathbf{B}$ is then

$$\mathbf{B}^{new} = \mathbf{B} + \eta \left(\mathbf{B}^{-\mathsf{T}} + \frac{1}{N} \sum_n \boldsymbol{\phi}(\mathbf{B}\mathbf{v}^n) \mathbf{v}^n \right) \tag{21.6.9}$$

An alternative 'natural gradient' algorithm[8, 180] that approximates a Newton update is given by multiplying the gradient by $\mathbf{B}^{\mathsf{T}}\mathbf{B}$ on the right to give the update

$$\mathbf{B}^{new} = \mathbf{B} + \eta \left(\mathbf{I} + \frac{1}{N} \sum_n \boldsymbol{\phi}(\mathbf{B}\mathbf{v}^n) \left(\mathbf{B}\mathbf{v}^n\right)^{\mathsf{T}} \right) \mathbf{B} \tag{21.6.10}$$

Here η is a learning rate which in the code `ica.m` we nominally set to 0.5.

A natural extension is to consider noise on the outputs, exercise(212), for which an EM algorithm is readily available. However, in the limit of low output noise, the EM formally fails, an effect which is related to the general discussion in section(11.4).

A popular alternative estimation method is FastICA[4] and can be related to an iterative Maximum Likelihood optimisation procedure. ICA can also be motivated from several alternative directions, including information theory[29]. We refer the reader to [137] for an in-depth discussion of ICA and related extensions.

21.7 Code

`FA.m`: Factor Analysis
`demoFA.m`: Demo of Factor Analysis
`ica.m`: Independent Components Analysis
`demoIca.m`: Demo ICA

21.8 Exercises

Exercise 207. *Factor analysis and scaling. Assume that a H-factor model holds for $\mathbf{x}$. Now consider the the transformation $\mathbf{y} = \mathbf{C}\mathbf{x}$, where $\mathbf{C}$ is a non-singular square diagonal matrix. Show that factor analysis is scale invariant,* i.e. *that the H-factor model also holds for $\mathbf{y}$, with the factor loadings appropriately scaled. How must the specific factors be scaled?*

Exercise 208. *For the constrained Factor Analysis model*

$$\mathbf{x} = \begin{pmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{B} \end{pmatrix} \mathbf{h} + \boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}\left(\boldsymbol{\epsilon} | \mathbf{0}, \text{diag}\left(\psi_1, \ldots, \psi_n\right)\right), \qquad \mathbf{h} \sim \mathcal{N}\left(\mathbf{h} | \mathbf{0}, \mathbf{I}\right) \tag{21.8.1}$$

derive a Maximum Likelihood EM algorithm for the matrices $\mathbf{A}$ and $\mathbf{B}$, assuming the datapoints $\mathbf{x}^1, \ldots, \mathbf{x}^N$ are i.i.d.

Exercise 209. *An apparent extension of FA analysis is to consider a correlated prior*

$$p(\mathbf{h}) = \mathcal{N}\left(\mathbf{h} | \mathbf{0}, \boldsymbol{\Sigma}_H\right) \tag{21.8.2}$$

Show that, provided no constraints are placed on the factor loading matrix $\mathbf{F}$, using a correlated prior $p(\mathbf{h})$ is an equivalent model to the original uncorrelated FA model.

[4] See `www.cis.hut.fi/projects/ica/fastica/`

Exercise 210. *Using the Woodbury identity and the definition of* $\mathbf{\Sigma}_D$ *in equation (21.2.2), show that one can rewrite* $\mathbf{\Sigma}_D{}^{-1}\mathbf{F}$ *as*

$$\mathbf{\Sigma}_D{}^{-1}\mathbf{F} = \mathbf{\Psi}^{-1}\mathbf{F}\left(\mathbf{I} + \mathbf{F}^{\mathsf{T}}\mathbf{\Psi}^{-1}\mathbf{F}\right)^{-1} \tag{21.8.3}$$

Exercise 211. *For the log likelihood function*

$$L(\sigma^2) = -\frac{N}{2}\left(D\log(2\pi) + \sum_{i=1}^{H}\log\lambda_i + \frac{1}{\sigma^2}\sum_{i=H+1}^{D}\lambda_i + (D-H)\log\sigma^2 + H\right) \tag{21.8.4}$$

Show $L(\sigma^2)$ *is maximal for*

$$\sigma^2 = \frac{1}{D-H}\sum_{j=H+1}^{D}\lambda_j \tag{21.8.5}$$

Exercise 212. *Consider an ICA model*

$$p(\mathbf{y}, \mathbf{x}|\mathbf{W}) = \prod_j p(y_j|\mathbf{x}, \mathbf{W})\prod_i p(x_i) \tag{21.8.6}$$

with

$$p(y_j|\mathbf{w}, \mathbf{W}) = \mathcal{N}\left(y_j\middle|\mathbf{w}_j^{\mathsf{T}}\mathbf{x}, \sigma^2\right) \tag{21.8.7}$$

1. *For the above model derive an EM algorithm for a set of i.i.d. data* $\mathbf{y}^1, \ldots, \mathbf{y}^N$ *and show that the required statistics for the M-step are* $\langle\mathbf{x}\rangle_{p(\mathbf{x}|\mathbf{y}^n,\mathbf{W})}$ *and* $\left\langle\mathbf{x}\mathbf{x}^{\mathsf{T}}\right\rangle_{p(\mathbf{x}|\mathbf{y}^n,\mathbf{W})}$.

2. *Show that for a non-Gaussian prior* $p(x_i)$*, the posterior*

$$p(\mathbf{x}|\mathbf{y}, \mathbf{W}) \tag{21.8.8}$$

   is non-factorised, non-Gaussian and generally intractable (its normalisation constant cannot be computed efficiently).

3. *Show that in the limit* $\sigma^2 \to 0$*, the EM algorithm fails.*

CHAPTER 22

Latent Ability Models

22.1 The Rasch Model

Consider an exam in which student s answers question q either correctly $x_{qs} = 1$ or incorrectly $x_{qs} = 0$. For a set of N students and Q questions, the performance of all students is given in the $Q \times N$ binary matrix $\mathbf{X}$. Based on this data alone we wish to evaluate the ability of each student. One approach is to define the ability a_s as the fraction of questions student s answered correctly. A more subtle analysis is to accept that some questions are more difficult than others so that a student who answered difficult questions should be awarded more highly than a student who answered the same number of easy questions. A priori we do not know which are the difficult questions and this needs to be estimated based on $\mathbf{X}$.

To account for inherent differences in question difficulty we may model the probability that a student s gets a question q correct based on the student's latent ability α_s and the latent difficulty of the question δ_q. A simple generative model of the response is

$$p(x_{qs} = 1|\boldsymbol{\alpha}, \boldsymbol{\delta}) = \sigma\left(\alpha_s - \delta_q\right) \tag{22.1.1}$$

where $\sigma\left(x\right) = 1/(1 + e^{-x})$. Under this model, the higher the latent ability is above the latent difficulty of the question, the more likely it is that the student will answer the question correctly.

22.1.1 Maximum Likelihood training

Making the i.i.d. assumption, the likelihood of the data $\mathbf{X}$ under this model is

$$p(\mathbf{X}|\boldsymbol{\alpha}, \boldsymbol{\delta}) = \prod_{s=1}^{S}\prod_{q=1}^{Q} \sigma\left(\alpha_s - \delta_q\right)^{x_{qs}}\left(1 - \sigma\left(\alpha_s - \delta_q\right)\right)^{1-x_{qs}} \tag{22.1.2}$$

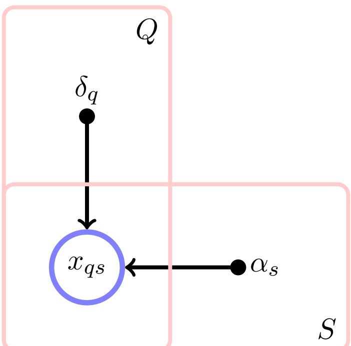


Figure 22.1: The Rasch model for analysing questions. Each element of the binary matrix $\mathbf{X}$, with $X_{qs} = 1$ if student s gets question q correct, is generated using the latent ability of the student α_s and the latent difficulty of the question δ_q.

The log likelihood is

$$L \equiv \log p(\mathbf{X}|\boldsymbol{\alpha}, \boldsymbol{\delta}) = \sum_{q,s} x_{qs} \log \sigma\left(\alpha_s - \delta_q\right) + (1 - x_{qs}) \log\left(1 - \sigma\left(\alpha_s - \delta_q\right)\right) \tag{22.1.3}$$

with derivatives

$$\frac{\partial L}{\partial \alpha_s} = \sum_{q=1}^{Q} \left(x_{qs} - \sigma\left(\alpha_s - \delta_q\right)\right), \qquad \frac{\partial L}{\partial \delta_q} = -\sum_{s=1}^{S} \left(x_{qs} - \sigma\left(\alpha_s - \delta_q\right)\right) \tag{22.1.4}$$

A simple way to learn the parameters is to use gradient ascent, see `demoRasch.m`, with extensions to Newton methods straightforward.

The generalisation to more than two responses can be achieved using a softmax-style function. More generally, the Rasch model is an example of an *item response theory*, a subject dealing with the analysis of questionnaires[91].

Missing data

Assuming the data is missing at random, missing data can be treated by computing the likelihood of only the observed elements of $\mathbf{X}$. In `rasch.m` missing data is assumed to be coded as `nan` so that the likelihood and gradients are straightforward to compute based on summing only over terms containing non `nan` entries.

Example 93. We plot an example of the use of the Rasch model in fig(22.2), estimating the latent abilities of 20 students based on a set of 50 questions. Based on using the number of questions each student answered correctly, the best students are (ranked from first) $8, 6, 1, 19, 4, 17, 20, 7, 15, 5, 12, 16, 2, 3, 18, 9, 11, 14, 10, 13$. Alternatively, ranking students according to the latent ability gives $8, 6, 19, 1, 20, 4, 17, 7, 15, 12, 5, 16, 2, 3, 18, 9, 11, 14, 10, 13$. This differs (only slightly in this case) from the number-correct ranking since the Rasch model takes into account the fact that some students answered difficult questions correctly. For example student 20 answered some difficult questions correctly.

22.1.2 Bayesian Rasch Models

The Rasch model will potentially overfit the data especially when there is only a small amount of data. A natural extension is to use a Bayesian technique, placing independent priors on the ability and question difficulty, so that the posterior ability and question difficulty is given by

$$p(\boldsymbol{\alpha}, \boldsymbol{\delta}|\mathbf{X}) \propto p(\mathbf{X}|\boldsymbol{\alpha}, \boldsymbol{\delta})p(\boldsymbol{\alpha})p(\boldsymbol{\delta}) \tag{22.1.5}$$

Natural priors are

$$p(\boldsymbol{\alpha}) = \prod_{s} \mathcal{N}\left(\alpha_s|0, \sigma^2\right), \qquad p(\boldsymbol{\delta}) = \prod_{q} \mathcal{N}\left(\delta_q|0, \tau^2\right) \tag{22.1.6}$$

where σ^2 and τ^2 are hyperparameters that can be learned by maximising $p(\mathbf{X}|\sigma^2, \tau^2)$.

Even in the case of using Gaussian priors, the posterior distribution $p(\boldsymbol{\alpha}, \boldsymbol{\delta}|\mathbf{X})$ is not of a standard form and approximations are required. In this case however, the posterior is log concave so that approximation methods based on variational or Laplace techniques are potentially adequate, or alternatively simple sampling approximations.

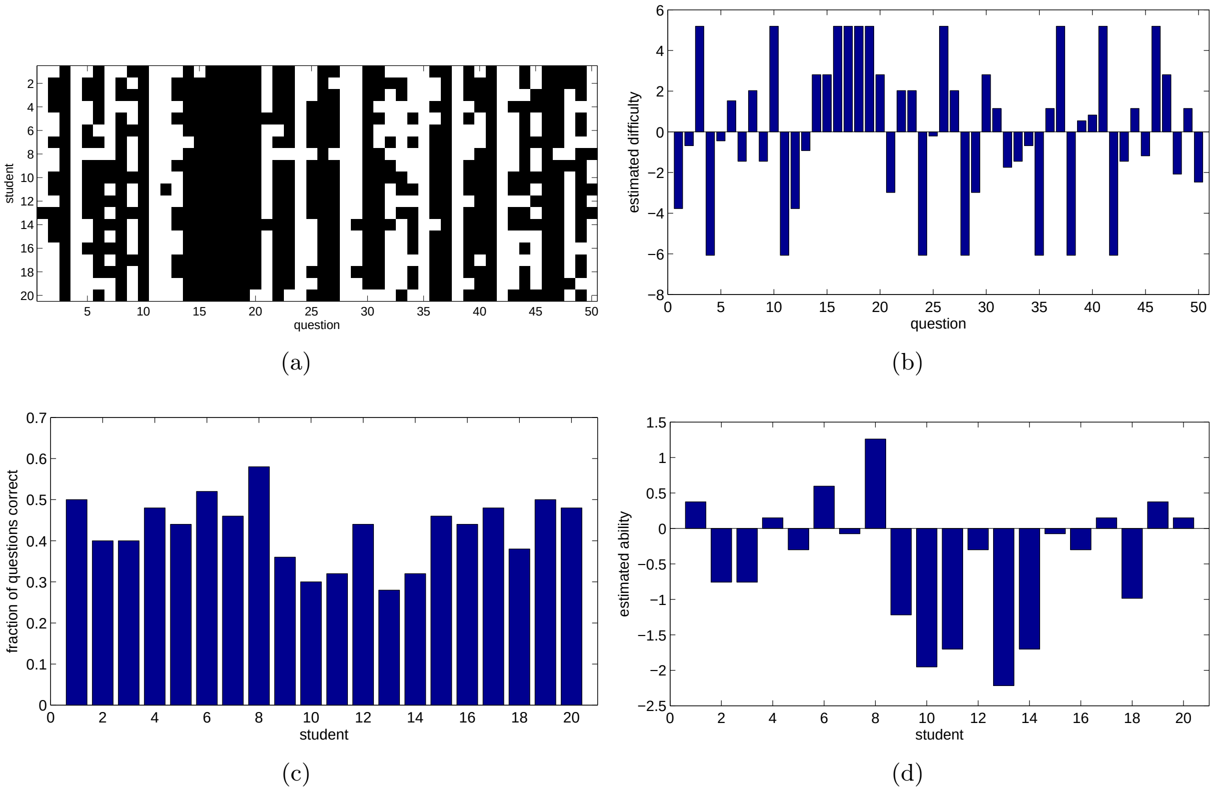


Figure 22.2: Rasch Model. **(a)**: The data of correct (white) and incorrect (black) answers. **(b)**: The estimated latent difficulty of each question. **(c)**: The fraction of questions each student answered correctly. **(d)**: The estimated latent ability.

22.2 Competition Models

22.2.1 Bradly-Terry-Luce model

The Bradly-Terry-Luce model assesses the ability of players based on one-on-one matches. Here we describe games in which only win/lose outcomes arise, leaving aside the minor complicating possibility of draws. For this win/lose scenario, the BTL model is a straightforward modification of the Rasch model so that for latent ability α_i of player i and latent ability α_j of player j, the probability that i beats j is given by

$$p(i \rhd j|\boldsymbol{\alpha}) = \sigma\left(\alpha_i - \alpha_j\right) \tag{22.2.1}$$

where $i \rhd j$ stands for player i beats player j. Based on a set of games data $\mathcal{X}$ with

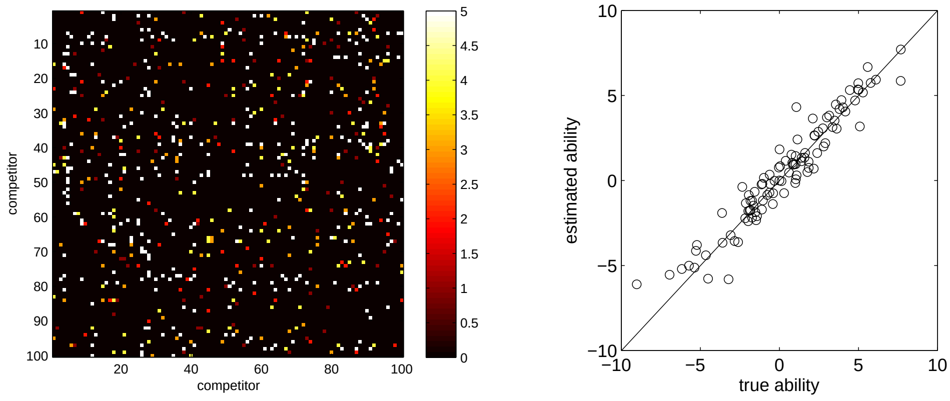
$$x_{ij}^n = \begin{cases} 1 & \text{if } i \rhd j \text{ in game } n \\ 0 & \text{otherwise} \end{cases} \tag{22.2.2}$$

the likelihood of the model is given by

$$p(\mathbf{X}|\boldsymbol{\alpha}) = \prod_n \prod_{ij} \left[\sigma\left(\alpha_i - \alpha_j\right)\right]^{x_{ij}^n} = \prod_{ij} \left[\sigma\left(\alpha_i - \alpha_j\right)\right]^{M_{ij}} \tag{22.2.3}$$

where $M_{ij} = \sum_n x_{ij}^n$ is the number of times player i beat player j. Training using Maximum Likelihood or a Bayesian technique can then proceed as for the Rasch model.

For the case of only two objects interacting, these models are called *pairwise comparison models*. Thurstone in the 1920's applied such models to a wide range of data, and the Bradley-Terry-Luce model is in fact a special case of his work[72].



(a) (b)

Figure 22.3: BTL model. **(a)**: The data $\mathbf{X}$ with X_{ij} being the number of times that competitor i beat competitor j. **(b)**: The true versus estimated ability. Even though the data is quite sparse, a reasonable estimate of the latent ability of each competitor is found.

Example 94. An example of the BTL model is given in fig(22.3) in which a matrix $\mathbf{M}$ containing the number of times that competitor i beat competitor j is given. The matrix entries $\mathbf{M}$ were drawn from a BTL model based on 'true abilities'. Using $\mathbf{M}$ alone the Maximum Likelihood estimate of these latent abilities is in close agreement with the true abilities.

22.2.2 Elo chess ranking

The Elo system [88] used in chess ranking is closely related to the BTL model above, though there is the added complication of the possibility of draws. In addition, the Elo system takes into account a measure of the variability in performance. For a given ability α_i, the actual performance π_i of player i in a game is given by

$$\pi_i = \alpha_i + \epsilon_i \tag{22.2.4}$$

where $\epsilon_i \sim \mathcal{N}\left(\epsilon_i | 0, \sigma^2\right)$. The variance σ^2 is fixed across all players and thus takes into account intrinsic variability in the performance. More formally the Elo model modifies the BTL model to give

$$p(\mathbf{X}|\boldsymbol{\alpha}) = \int_{\boldsymbol{\pi}} p(\mathbf{X}|\boldsymbol{\pi})p(\boldsymbol{\pi}|\boldsymbol{\alpha}), \qquad p(\boldsymbol{\pi}|\boldsymbol{\alpha}) = \mathcal{N}\left(\boldsymbol{\pi}|\boldsymbol{\alpha}, \sigma^2\mathbf{I}\right) \tag{22.2.5}$$

where $p(\mathbf{X}|\boldsymbol{\pi})$ is given by equation (22.2.3) on replacing $\boldsymbol{\alpha}$ with $\boldsymbol{\pi}$.

22.2.3 Glicko and TrueSkill

Glicko[111] and TrueSkill[128] are essentially Bayesian versions of the Elo model with the refinement that the latent ability is modelled, not by a single number, but by a Gaussian distribution

$$p(\alpha_i|\theta_i) = \mathcal{N}\left(\alpha_i | \mu_i, \sigma_i^2\right) \tag{22.2.6}$$

This can capture the fact that a player may be consistently reasonable (quite high μ_i and low σ_i^2) or an erratic genius (high μ_i but with large σ_i^2). The parameters of the model are then

$$\theta = \left\{\mu_i, \sigma_i^2, i = 1, \ldots, S\right\} \tag{22.2.7}$$

for a set of S players. The interaction model $p(\mathbf{X}|\boldsymbol{\alpha})$ is as for the win/lose Elo model, equation (22.2.1). The likelihood for the model given the parameters is

$$p(\mathbf{X}|\theta) = \int_{\boldsymbol{\alpha}} p(\mathbf{X}|\boldsymbol{\alpha})p(\boldsymbol{\alpha}|\theta) \tag{22.2.8}$$

This integral is formally intractable and numerical approximations are required. In this context Expectation Propagation has proven to be a useful technique[193].

The TrueSkill system is used for example to assess the abilities of players in online gaming, also taking into account the abilities of teams of individuals in tournaments. A temporal extension has recently been used to reevaluate the change in ability of chess players with time[71].

22.3 Code

`rasch.m`: Rasch model training
`demoRasch.m`: Demo for the Rasch model

22.4 Exercises

Exercise 213 (Bucking Bronco)**.** `bronco.mat` *contains information about a bucking bronco competition. There are 500 competitors and 20 bucking broncos. A competitor j attempts to stay on a bucking bronco i for a minute. If the competitor succeeds the entry X_{ij} is 1, otherwise 0. Each competitor gets to ride three bucking broncos only (the missing data is coded as* `nan`*). Having viewed all the 500 amateurs, desperate Dan enters the competition and bribes the organisers into letting him avoid having to ride the difficult broncos. Based on using a Rasch model, what are the top 10 most difficult broncos, in order of the most difficult first?*

Exercise 214 (BTL training)**.**

1. *Show that the log likelihood for the Bradly-Terry-Luce model is given by*
$$L(\alpha) = \sum_{ij} X_{ij} \log \sigma (\alpha_i - \alpha_j) \tag{22.4.1}$$
where X_{ij} is the number of times that player i beats player j in a set of games.
2. *Compute the gradient of $L(\alpha)$.*
3. *Compute the Hessian of the BTL model and verify that it is negative semidefinite.*

Exercise 215 (La Reine)**.**

1. *Program a simple gradient ascent routine to learn the latent abilities of competitors based on a series of win/lose outcomes.*
2. *In a modified form of Swiss cow 'fighting', a set of cows compete by pushing each other until submission. At the end of the competition one cow is deemed to be 'la reine'. Based on the data in* `BTL.mat` *(for which X_{ij} contains the number of times cow i beat cow j), fit a BTL model and return a ranked list of the top ten best fighting cows, 'la reine' first.*

Exercise 216. *An extension of the BTL model is to consider additional 'factors' that describe the state of the competitors when they play. For example, we have a set of S football teams, and a set of matrices $\mathbf{X}^1, \ldots, \mathbf{X}^N$, with $X^n_{ij} = 1$ if team i beat team j in match n. In addition we have for each match and team a vector of binary factors $f^n_i \in \{0, 1\}$ that describes the team. For example, for the team $i = 1$ (Madchester United), the factor $f_{1,1} = 1$ if Bozo is playing, 0 if not. It is suggested that the ability of team i in game n is measured by*

$$\alpha^n_i = d_i + \sum_{h=1}^{H} w_{h,i} f^n_{h,i} \tag{22.4.2}$$

where $f^n_{h,i} = 1$ if factor h is present in team i in game n. d_i is a default latent ability of the team which is assumed constant across all games. We have such a set of factors for each match, giving $f^n_{h,i}$.

1. *Using the above definition of the latent ability in the BTL model, our interest is to find the weights* $\mathbf{W}$ *and abilities* $\mathbf{d}$ *that best predict the ability of the team, given that we have a set of historical plays* $(\mathbf{X}^n, \mathbf{F}^n), n = 1, \ldots, N$*. Write down the likelihood for the BTL model as a function of the set of all team weights* $\mathbf{W}, \mathbf{d}$*.*

2. *Compute the gradient of the log likelihood of this model.*

3. *Explain how this model can be used to assess the importance of Bozo's contribution to Madchester United's ability.*

4. *Given a learned set of* $\mathbf{W}, \mathbf{d}$ *and the knowledge that Madchester United (team 1) will play Chelski (team 2) tomorrow explain how, given the list of factors* $\mathbf{f}$ *for Chelski (which includes issues such as who will be playing in the team), one can select the best Madchester United team to maximise the probability of winning the game.*

Part IV

Dynamical Models

CHAPTER 23

Discrete-state Markov Chains

23.1 Markov Models

Time-series are datasets for which the constituent datapoints can be naturally ordered. This order often corresponds to an underlying single physical dimension, typically time, though any other single dimension may be used. The time-series models we consider are probability models over a collection of random variables $v_1, \ldots, v_T$ with individual variables v_t indexed by a time index t. These indices are elements of the index set $\mathcal{T}$. For nonnegative indices, $\mathcal{T} = \mathbb{N}^+$, the model is a discrete-time process. Continuous-time processes, $\mathcal{T} = \mathbb{R}$, are natural in particular application domains, yet require additional notation and concepts. We therefore focus exclusively on discrete-time models. A probabilistic time-series model requires a specification of the joint distribution $p(v_1, \ldots, v_T)$. For the case in which the observed data v_t is discrete, the joint probability table for $p(v_1, \ldots, v_T)$ has exponentially many entries. Clearly we cannot expect to independently specify all the exponentially many entries and we are forced to make simplified models under which these entries can be parameterised in a lower dimensional manner. Such simplifications are at the heart of time-series modelling and we will discuss some classical models in the following sections.

Definition 107 (Time-Series Notation)**.**

$$x_{a:b} \equiv x_a, x_{a+1}, \ldots, x_b, \qquad \text{with } x_{a:b} = x_a \text{ for } b \leq a \tag{23.1.1}$$

For timeseries data $v_1, \ldots, v_T$, we need a model $p(v_1, \ldots, v_T)$ which we can write more compactly as $p(v_{1:T})$. For a predictive model it is meaningful to consider the decomposition

$$p(v_{1:T}) = \prod_{t=1}^{T} p(v_t|v_{1:t-1}) \tag{23.1.2}$$

with the convention $p(v_t|v_{1:t-1}) = p(v_1)$ for $t = 1$. It is often natural to assume that the influence of the immediate past is more relevant than the remote past and in Markov models only a limited number of previous observations are required to predict the present.

Definition 108 (Markov chain)**.** A Markov chain model defined on either discrete or continuous variables $v_{1:T}$ is one in which the following conditional independence assumption holds:

$$p(v_t|v_1, \ldots, v_{t-1}) = p(v_t|v_{t-L}, \ldots, v_{t-1}) \tag{23.1.3}$$

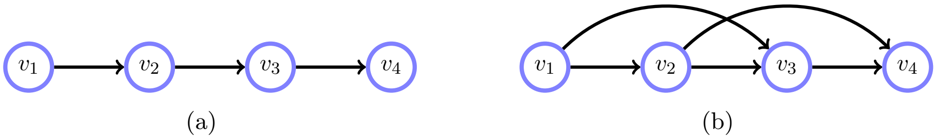


Figure 23.1: **(a)**: First order Markov Chain. **(b)**: Second order Markov Chain.

where $L \geq 1$ is the *order* of the Markov chain and $v_t = \emptyset$ for $t < 1$. For a first order Markov chain,

$$p(v_{1:T}) = p(v_1)p(v_2|v_1)p(v_3|v_2), \ldots, p(v_T|v_{T-1}) \tag{23.1.4}$$

For a *stationary Markov chain* the transitions $p(v_t = s'|v_{t-1} = s) = f(s', s)$ are time-independent. Otherwise the chain is called non-stationary, $p(v_t = s'|v_{t-1} = s) = f(s', s, t)$.

23.1.1 Equilibrium and stationary distribution of a Markov chain

The stationary distribution p_∞ of a Markov chain with transition matrix $\mathbf{M}$ is defined by the condition

$$p_\infty(i) = \sum_j \underbrace{p(x_t = i|x_{t-1} = j)}_{M_{ij}} p_\infty(j) \tag{23.1.5}$$

In matrix notation this can be written as the vector equation

$$\mathbf{p}_\infty = \mathbf{M}\mathbf{p}_\infty \tag{23.1.6}$$

so that the stationary distribution is proportional to the eigenvector with unit eigenvalue of the transition matrix. Note that a stationary distribution always exists, although it may not be unique. See exercise(217) and [120].

Given a state x_1, we can iteratively draw samples $x_2, \ldots, x_t$ from the Markov chain drawing a sample from $p(x_2|x_1 = \mathsf{x}_1)$, and then from $p(x_3|\mathsf{x}_2)$ *etc.* If we plot the histogram of sample states, what does this converge to as $t \to \infty$? As we repeatedly sample a new state from the chain, the distribution at time t for an initial distribution $\mathbf{p}_1(i) = \delta(i, \mathsf{x}_1)$ is

$$\mathbf{p}_t = \mathbf{M}^t \mathbf{p}_1 \tag{23.1.7}$$

If for $t \to \infty$, $\mathbf{p}_\infty$ is independent of the initial distribution $\mathbf{p}_1$, then $\mathbf{p}_\infty$ is the equilibrium distribution of the chain. See exercise(218) for an example of a Markov chain which does not have an equilibrium distribution.

Example 95 (*PageRank*)**.** Despite their apparent simplicity, Markov chains have been put to interesting use in information retrieval and search-engines. Define the matrix

$$A_{ij} = \begin{cases} 1 & \text{if website } j \text{ has a hyperlink to website } i \\ 0 & \text{otherwise} \end{cases} \tag{23.1.8}$$

From this we can define a Markov transition matrix with elements

$$M_{ij} = \frac{A_{ij}}{\sum_i A_{ij}} \tag{23.1.9}$$

The equilibrium distribution of this Markov Chain has the following interpretation : If we jump at random from website to website, following the links, the equilibrium distribution $p_\infty(i)$ is the relative number of times we will visit website i. This has a natural interpretation as the 'importance' of website i; if a website is isolated in the web, it will be visited infrequently by random hopping; if a website is linked by many others it will be visited more frequently.

A crude *search engine* works then as follows. For each website i a list of words associated with that website is collected. After doing this for all websites, one can make an 'inverse' list of which websites contain word w. When a user searches for term w, the list of websites that contain word w is then returned, ranked according to their importance (as defined by the equilibrium distribution above). This is a crude summary as how early search engines worked, `infolab.stanford.edu/~backrub/google.html`.

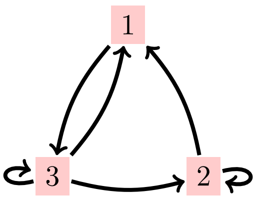


Figure 23.2: A state transition diagram for a three state Markov chain. Note that a state transition diagram is not a graphical model – it simply graphically displays the non-zero entries of the transition matrix $p(i|j)$. The absence of link from j to i indicates that $p(i|j) = 0$.

23.1.2 Fitting Markov models

Given a sequence $v_{1:T}$, fitting a stationary Markov chain by Maximum Likelihood corresponds to setting the transitions by counting the number of observed (first-order) transitions in the sequence:

$$p(v_\tau = i|v_{\tau-1} = j) \propto \sum_{t=2}^{T} \mathbb{I}\left[v_t = i, v_{t-1} = j\right] \tag{23.1.10}$$

To show this, for convenience we write $p(v_\tau = i|v_{\tau-1} = j) \equiv \theta i|j$, so that the likelihood is

$$p(v_{1:T}|\theta) = \prod_{t=2}^{T} \prod_{ij} \theta_{i|j}^{\mathbb{I}[v_t=i,v_{t-1}=j]} \tag{23.1.11}$$

Taking logs and adding the Lagrange constraint for the normalisation,

$$L(\theta) = \sum_{t=2}^{T} \sum_{ij} \mathbb{I}\left[v_t = i, v_{t-1} = j\right] \log \theta_{i|j} + \sum_j \lambda_j \left(1 - \sum_i \theta_{i|j}\right) \tag{23.1.12}$$

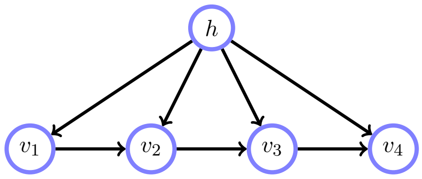
Differentiating with respect to $\theta_{i|j}$, we immediately arrive at the intuitive setting, (23.1.10). For a set of timeseries, $v^n_{1:T_n}, n = 1, \ldots, N$, the transition is given by counting all transitions across time and datapoints. The Maximum Likelihood setting for the initial first timestep distribution is $p(v_1 = i) \propto \sum_n \mathbb{I}\left[v^n_1 = i\right]$.

Bayesian fitting

A convenient choice for a prior on the transition is $p(\theta) = \prod_j p(\theta_{\cdot|j})$ where each conditional transition is itself a Dirichlet distribution with hyperparameters $\mathbf{u}_j$, $p(\theta_{\cdot|j}) = \text{Dirichlet}\left(\theta_{\cdot|j}|\mathbf{u}_j\right)$, since this is conjugate to the categorical transition and

$$p(\theta|v_{1:T}) \propto p(v_{1:T}|\theta)p(\theta) = \prod_t \prod_{ij} \theta_{i|j}^{\mathbb{I}[v_t=i,v_{t-1}=j]} \theta_{i|j}^{u_{ij}-1} = \prod_j \text{Dirichlet}\left(\theta_{\cdot|j}|\hat{\mathbf{u}}_j\right) \tag{23.1.13}$$

where $\hat{\mathbf{u}}_j = \sum_{t=2}^{T} \mathbb{I}\left[v_{t-1} = i, v_t = j\right]$, being the number of $i \to j$ transitions in the dataset.



(a)

Figure 23.3: Mixture of first order Markov chains. The discrete hidden variable $\mathrm{dom}(h) = \{1,\ldots,H\}$ indexes the Markov chain $p(v_t|v_{t-1},h)$. Such models can be useful as simple sequence clustering tools.

23.1.3 Mixture of Markov models

Given a set of sequences $\mathcal{V} = \{v^n_{1:T}, n = 1,\ldots,N\}$, how might we cluster them? To keep the notation less cluttered, we assume that all sequences are of the same length T with the extension to differing lengths being straightforward. One simple approach is to fit a mixture of Markov models. Assuming the data is i.i.d., $p(\mathcal{V}) = \prod_n p(v^n_{1:T})$, we define a mixture model for a single datapoint $v_{1:T}$. Here we assume each component model is first order Markov

$$p(v_{1:T}) = \sum_{h=1}^{H} p(h)p(v_{1:T}|h) = \sum_{h=1}^{H} p(h) \prod_{t=1}^{T} p(v_t|v_{t-1},h) \tag{23.1.14}$$

The graphical model is depicted in fig(23.3). Clustering can then be finding the Maximum Likelihood parameters $p(h)$, $p(v_t|v_{t-1},h)$ and subsequently assign the clusters according to $p(h|v^n_{1:T}$. Below we discuss the application of the EM algorithm to this model to learn the Maximum Likelihood parameters.

EM algorithm

Under the i.i.d. data assumption, the log likelihood is

$$\log p(\mathcal{V}) = \sum_{n=1}^{N} \log \sum_{h=1}^{H} p(h) \prod_{t=1}^{T} p(v^n_t|v^n_{t-1},h) \tag{23.1.15}$$

For the M-step, our task is to maximise the energy

$$E = \sum_{n=1}^{N} \left\langle \log p(v^n_{1:T},h)\right\rangle_{p^{old}(h|v^n_{1:T})} = \sum_{n=1}^{N} \left\{ \left\langle \log p(h)\right\rangle_{p^{old}(h|v^n_{1:T})} + \sum_{t=1}^{T} \left\langle \log p(v_t|v_{t-1},h)\right\rangle_{p^{old}(h|v^n_{1:T})} \right\}$$

The contribution to the energy from the parameter $p(h)$ is

$$\sum_{n=1}^{N} \left\langle \log p(h)\right\rangle_{p^{old}(h|v^n_{1:T})} \tag{23.1.16}$$

By defining

$$\hat{p}^{old}(h) \propto \sum_{n=1}^{N} p^{old}(h|v^n_{1:T}) \tag{23.1.17}$$

one can view maximising (23.1.16) as equivalent to minimising

$$\mathrm{KL}\left(\hat{p}^{old}(h)|p(h)\right) \tag{23.1.18}$$

so that the optimal choice from the M-step is to set $p^{new} = \hat{p}^{old}$, namely

$$p^{new}(h) \propto \sum_{n=1}^{N} p^{old}(h|v^n_{1:T}) \tag{23.1.19}$$

For those less comfortable with this argument, a direct maximisation including a Lagrange term to ensure normalisation of $p(h)$ can be used to derive the same result.

Similarly, the M-step for $p(v_t|v_{t-1}, h)$ is

$$p^{new}(v_t = i|v_{t-1} = j, h = k) \propto \sum_{n=1}^{N} p^{old}(h = k|v_{1:T}^n) \sum_{t=2}^{T} \mathbb{I}[v_t^n = i] \, \mathbb{I}\left[v_{t-1}^n = j\right] \tag{23.1.20}$$

The initial term $p(v_1|h)$ is updated using

$$p^{new}(v_1 = i|h = k) \propto \sum_{n=1}^{N} p^{old}(h = k|v_{1:T}^n)\mathbb{I}[v_1^n = i] \tag{23.1.21}$$

The E-step sets

$$p^{old}(h|v_{1:T}^n) \propto p(h)p(v_{1:T}^n|h) = p(h) \prod_{t=1}^{T} p(v_t^n|v_{t-1}^n, h) \tag{23.1.22}$$

For long sequences, explicitly computing the product of many terms may lead to numerical underflow issues. In practice it is therefore best to work with logs,

$$\log p^{old}(h|v_{1:T}^n) = \log p(h) + \sum_{t=1}^{T} \log p(v_t^n|v_{t-1}^n, h) + \mathsf{const.} \tag{23.1.23}$$

In this way any large constants common to all h can be removed and the distribution may be computed accurately. See `mixMarkov.m`.

Example 96 (Gene Clustering)**.** Consider the 20 fictitious gene sequences below presented in an arbitrarily chosen order. Each sequence consists of 20 symbols from the set $\{A, C, G, T\}$. The task is to try to cluster these sequences into two groups.

CATAGGCATTCTATGTGCTG	CCAGTTACGGACGCCGAAAG	TGGAACCTTAAAAAAAAAAA	GTCTCCTGCCCTCTCTGAAC	
GTGCCTGGACCTGAAAAGCC	CGGCCGCGCCTCCGGGAACG	AAAGTGCTCTGAAAACTCAC	ACATGAACTACATAGTATAA	
GTTGGTCAGCACACGGACTG	CCTCCCCTCCCCTTTCCTGC	CACTACGGCTACCTGGGCAA	CGGTCCGTCCGAGGCACTC	(23.1.24)
TAAGTGTCCTCTGCTCCTAA	CACCATCACCCTTGCTAAGG	AAAGAACTCCCCTCCCTGCC	CAAATGCCTCACGCGTCTCA	
GCCAAGCAGGGTCTCAACTT	CATGGACTGCTCCACAAAGG	AAAAAAACGAAAAACCTAAG	GCGTAAAAAAAGTCCTGGGT	

A simple approach is to assume that the sequences are generated from a two-component $H = 2$ mixture of Markov Models and train the model using maximum likelihood. The likelihood has local optima so that the procedure needs to be run several times and the solution with the highest likelihood chosen. One can then assign each of the sequences by examining $p(h|v_{1:T}^n)$. If this posterior probability is greater than 0.5, we assign it to class 1, otherwise to class 2. Using this procedure, we find the following clusters:

CATAGGCATTCTATGTGCTG	TGGAACCTTAAAAAAAAAAA	
CCAGTTACGGACGCCGAAAG	GTCTCCTGCCCTCTCTGAAC	
CGGCCGCGCCTCCGGGAACG	GTGCCTGGACCTGAAAAGCC	
ACATGAACTACATAGTATAA	AAAGTGCTCTGAAAACTCAC	
GTTGGTCAGCACACGGACTG	CCTCCCCTCCCCTTTCCTGC	
CACTACGGCTACCTGGGCAA	TAAGTGTCCTCTGCTCCTAA	(23.1.25)
CGGTCCGTCCGAGGCACTCG	AAAGAACTCCCCTCCCTGCC	
CACCATCACCCTTGCTAAGG	AAAAAAACGAAAAACCTAAG	
CAAATGCCTCACGCGTCTCA	GCGTAAAAAAAGTCCTGGGT	
GCCAAGCAGGGTCTCAACTT		
CATGGACTGCTCCACAAAGG		

where sequences in the first column are assigned to cluster 1, and sequences in the second column to cluster 2. In this case the data in (23.1.24) was in fact generated by a two-component Markov mixture, and the posterior assignment (23.1.25) is in agreement with the actual clusters used in the generation of the data. See `demoMixMarkov.m`

23.2 Hidden Markov Models

The Hidden Markov Model (HMM) defines a Markov chain on hidden (or 'latent') variables. The observed (or 'visible') variables are dependent on the hidden variables through

$$p(h_{1:T}, v_{1:T}) = p(v_1|h_1)p(h_1) \prod_{t=2}^{T} p(v_t|h_t)p(h_t|h_{t-1}) \tag{23.2.1}$$

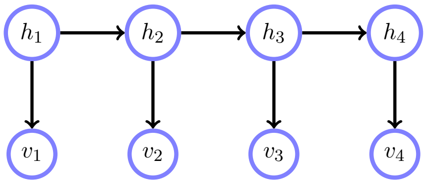


Figure 23.4: A first order hidden Markov model with 'hidden' variables $\text{dom}(h_t) = \{1, \ldots, H\}$, $t = 1 : T$. The 'visible' variables v_t can be either discrete or continuous.

for which the graphical model is depicted in fig(23.4). For a stationary HMM the transition $p(h_t|h_{t-1})$ and emission $p(v_t|h_t)$ distributions are constant through time.

Definition 109 (Transition Distribution)**.** For a stationary HMM the transition distribution $p(h_{t+1}|h_t)$ is defined by the $H \times H$ transition matrix

$$A_{i',i} = p(h_{t+1} = i'|h_t = i) \tag{23.2.2}$$

and an initial distribution

$$a_i = p(h_1 = i). \tag{23.2.3}$$

Definition 110 (Emission Distribution)**.** For a stationary HMM and emission distribution $p(v_t|h_t)$ with discrete states $v_t \in \{1, \ldots, V\}$, we defined a $V \times H$ emission matrix

$$B_{i,j} = p(v_t = i|h_t = j) \tag{23.2.4}$$

For continuous outputs, h_t selects on of H possible output distribution $p(v_t|h_t)$.

In the engineering and machine learning communities, the term HMM typically refers to the case of discrete variables h_t. In statistics the term HMM often refers to any model with the independence structure in equation (23.2.1), regardless of the form of the variables h_t (see for example [53]). A subset of the many applications of HMMs is given in section(23.5).

23.2.1 The classical inference problems

Filtering	(Inferring the present)	$p(h_t|v_{1:t})$	
Prediction	(Inferring the future)	$p(h_t|v_{1:s})$	$t > s$
Smoothing	(Inferring the past)	$p(h_t|v_{1:u})$	$t < u$
Likelihood		$p(v_{1:T})$	
Most likely Hidden path	(Viterbi alignment)	$\underset{h_{1:T}}{\text{argmax}}\ p(h_{1:T}|v_{1:T})$	

The most likely hidden path problem is termed *Viterbi alignment* in the engineering literature. All these classical inference problems are straightforward in the case of discrete variables h since the distribution is singly-connected and any standard inference method can be adopted for these problems. The factor graph and junction trees for the first order HMM are given in fig(23.5). In both cases, after suitable setting of the factors and clique potentials, filtering corresponds to passing messages from left to right and upwards; smoothing corresponds to a valid schedule of message passing/absorption both forwards and backwards along all edges.

For the basic first order HMM it is straightforward to derive appropriate recursions directly. This is both instructive but also helps in constructing compact and numerically stable algorithms.

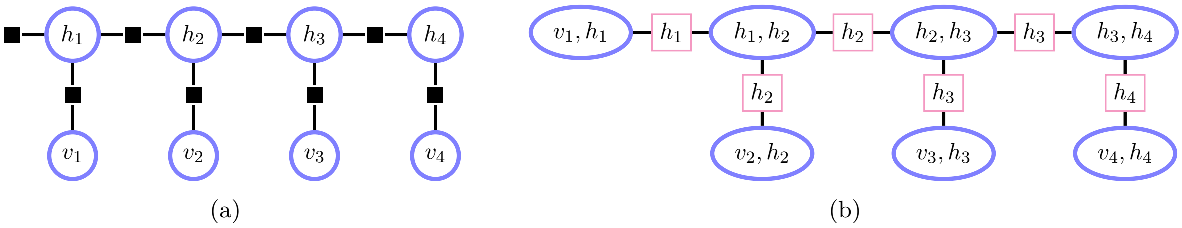


Figure 23.5: **(a)**: Factor Graph for the first order HMM of fig(23.4). **(b)**: Junction Tree for fig(23.4).

23.2.2 Filtering $p(h_t|v_{1:t})$

It is useful to first compute the joint marginal $p(h_t, v_{1:t})$ since the likelihood of the sequence can be obtained from this expression. The conditional marginal $p(h_t|v_{1:t})$ is obtained once $p(h_t, v_{1:t})$ has been found. A recursion for $p(h_t, v_{1:t})$ is obtained by considering:

$$p(h_t, v_{1:t}) = \sum_{h_{t-1}} p(h_t, h_{t-1}, v_{1:t-1}, v_t) \tag{23.2.5}$$

$$= \sum_{h_{t-1}} p(v_t|\cancel{v_{1:t-1}}, h_t, \cancel{h_{t-1}})p(h_t|\cancel{v_{1:t-1}}, h_{t-1})p(v_{1:t-1}, h_{t-1}) \tag{23.2.6}$$

$$= \sum_{h_{t-1}} p(v_t|h_t)p(h_t|h_{t-1})p(h_{t-1}, v_{1:t-1}) \tag{23.2.7}$$

The cancellations follow from the conditional independence assumptions of the model. Hence if we define

$$\alpha(h_t) = p(h_t, v_{1:t}) \tag{23.2.8}$$

the above gives the *α-recursion*

$$\alpha(h_t) = \sum_{h_{t-1}} p(v_t|h_t)p(h_t|h_{t-1})\alpha(h_{t-1}), \qquad t > 1 \tag{23.2.9}$$

with

$$\alpha(h_1) = p(h_1, v_1) = p(v_1|h_1)p(h_1) \tag{23.2.10}$$

Since each α is smaller than 1, and the recursion involves multiplication by terms less than 1, the α's can become very small. To avoid numerical problems it is therefore advisable to work with $\log \alpha(h_t)$, see `HMMforward.m`.

Normalisation gives the *filtered posterior*

$$p(h_t|v_{1:t}) \propto \alpha(h_t) \tag{23.2.11}$$

If we only require the filtered posterior we are free to rescale the α's as we wish. In this case an alternative to working with $\log \alpha$ messages is to work with normalised α messages so that the sum of the components is always 1.

We can write equation (23.2.7) above directly in terms of the recursion of the filtered distribution

$$p(h_t|v_{1:t}) \propto \sum_{h_{t-1}} p(v_t|h_t)p(h_t|h_{t-1})p(h_{t-1}|v_{1:t-1}) \qquad t > 1 \tag{23.2.12}$$

The term $p(h_{t-1}|v_{1:t-1})$ has the effect of removing all nodes in the graph before time $t-1$ and replacing their influence by a modified distribution on h_t. The effect of a new observation and latent transition can be thought of as likelihood terms so that one may interpret $p(h_{t-1}|v_{1:t-1})$ as a prior term, and $p(v_t|h_t)p(h_t|h_{t-1})$ as a likelihood, giving rise to the joint posterior $p(h_t, h_{t-1}|v_{1:t})$ under Bayesian updating. At the next timestep the previous posterior becomes the new prior, and a new posterior is computed.

23.2.3 Parallel smoothing $p(h_t|v_{1:T})$

There are two main approaches to computing $p(h_t|v_{1:T})$. Perhaps the most common in the HMM literature is the parallel method which is equivalent to message passing on Factor Graphs. In this one separates the smoothed posterior into contributions from the past and future:

$$p(h_t, v_{1:T}) = p(h_t, v_{1:t}, v_{t+1:T}) = \underbrace{p(h_t, v_{1:t})}_{past}\underbrace{p(v_{t+1:T}|h_t, v_{1:t})}_{future} = \alpha(h_t)\beta(h_t) \tag{23.2.13}$$

The term $\alpha(h_t)$ is obtained from the 'forward' α recursion, (23.2.9). The term $\beta(h_t)$ may be obtained using a 'backward' β recursion as we show below. The forward and backward recursions are independent and may therefore be run in parallel, with their results combined to obtain the smoothed posterior. This approach is also sometimes termed the *two-filter smoother*.

The β recursion

$$p(v_{t:T}|h_{t-1}) = \sum_{h_t} p(v_t, v_{t+1:T}, h_t|h_{t-1}) \tag{23.2.14}$$

$$= \sum_{h_t} p(v_t|\cancel{v_{t+1:T}}, h_t, \cancel{h_{t-1}})p(v_{t+1:T}, h_t|h_{t-1}) \tag{23.2.15}$$

$$= \sum_{h_t} p(v_t|h_t)p(v_{t+1:T}|h_t, \cancel{h_{t-1}})p(h_t|h_{t-1}) \tag{23.2.16}$$

Defining

$$\beta(h_t) \equiv p(v_{t+1:T}|h_t) \tag{23.2.17}$$

equation (23.2.16) above gives the *β-recursion*

$$\beta(h_{t-1}) = \sum_{h_t} p(v_t|h_t)p(h_t|h_{t-1})\beta(h_t), \qquad 2 \le t \le T \tag{23.2.18}$$

with $\beta(h_T) = 1$. As for the forward pass, working in log space is recommended to avoid numerical difficulties[1]. The smoothed posterior is then given by

$$p(h_t, v_{1:T}) = \alpha(h_t)\beta(h_t), \qquad t = 1, \ldots, T \tag{23.2.19}$$

and

$$p(h_t|v_{1:T}) \equiv \gamma(h_t) = \frac{\alpha(h_t)\beta(h_t)}{\sum_{h_t} \alpha(h_t)\beta(h_t)} \tag{23.2.20}$$

Together the $\alpha - \beta$ recursions are called the *Forward-Backward* algorithm in the Engineering literature.

23.2.4 Sequential smoothing

An alternative to the parallel method is to form a recursion directly for the smoothed posterior. This can be achieved by recognising that conditioning on the present makes the future redundant:

$$p(h_t|v_{1:T}) = \sum_{h_{t+1}} p(h_t, h_{t+1}|v_{1:T}) = \sum_{h_{t+1}} p(h_t|h_{t+1}, v_{1:t}, \cancel{v_{t+1:T}})p(h_{t+1}|v_{1:T}) \tag{23.2.21}$$

This gives a recursion for $\gamma(h_t) \equiv p(h_t|v_{1:T})$:

$$\gamma(h_t) = \sum_{h_{t+1}} p(h_t|h_{t+1}, v_{1:t})\gamma(h_{t+1}) \tag{23.2.22}$$

[1] If one only desires posterior distributions, one can also perform local normalisation at each stage since only the relative magnitude of the components of β are of importance.

with $\gamma(h_T) \propto \alpha(h_T)$. The term $p(h_t|h_{t+1}, v_{1:t})$ may be computed based on the filtered results $p(h_t|v_{1:t})$ using

$$p(h_t|h_{t+1}, v_{1:t}) \propto p(h_{t+1}, h_t|v_{1:t}) \propto p(h_{t+1}|h_t)p(h_t|v_{1:t}) \tag{23.2.23}$$

where the proportionality constant is found by normalisation. This is a form of *dynamics reversal*, as if we are reversing the direction of the hidden to hidden arrow in the HMM. This procedure, also termed the *Rauch-Tung-Striebel* smoother[2], is sequential since we need to first complete the α recursions, after which the γ recursion may begin. This is a so-called *correction smoother* since it 'corrects' the filtered result. Interestingly, the evidential states $v_{1:T}$ are not needed during the γ recursion.

The $\alpha - \beta$ and $\alpha - \gamma$ recursions are related through

$$\gamma(h_t) \propto \alpha(h_t)\beta(h_t) \tag{23.2.24}$$

Computing the pairwise marginal $p(h_t, h_{t-1}|v_{1:T})$

To implement the EM algorithm for learning, section(23.3.1), we require terms such as $p(h_t, h_{t-1}|v_{1:T})$. These can be obtained for example by message passing on a factor graph. If one used the junction tree approach to perform smoothed inference, the pairwise marginals are contained in the cliques, see fig(23.4b). Alternatively, an explicit recursion is as follows:

$$\begin{aligned} p(h_t, h_{t+1}|v_{1:T}) &\propto p(v_{1:t}, v_{t+1}, v_{t+2:T}, h_{t+1}, h_t) \\ &= p(v_{t+2:T}|\cancel{v_{1:t}, v_{t+1}, h_t}, h_{t+1})p(v_{1:t}, v_{t+1}, h_{t+1}, h_t) \\ &= p(v_{t+2:T}|h_{t+1})p(v_{t+1}|\cancel{v_{1:t}, h_t}, h_{t+1})p(v_{1:t}, h_{t+1}, h_t) \\ &= p(v_{t+2:T}|h_{t+1})p(v_{t+1}|h_{t+1})p(h_{t+1}|\cancel{v_{1:t}}, h_t)p(v_{1:t}, h_t) \end{aligned} \tag{23.2.25}$$

Rearranging, we therefore have

$$p(h_t, h_{t+1}|v_{1:T}) \propto \alpha(h_t)p(v_{t+1}|h_{t+1})p(h_{t+1}|, h_t)\beta(h_{t+1}) \tag{23.2.26}$$

See `HMMsmooth.m`.

The likelihood $p(v_{1:T})$

The likelihood of a sequence of observations can be computed from

$$p(v_{1:T}) = \sum_{h_T} p(h_T, v_{1:T}) = \sum_{h_T} \alpha(h_T) \tag{23.2.27}$$

If normalisation of the α messages has been employed, unless these normalisation terms have been stored, we cannot use the α messages directly to compute the likelihood. An alternative computation can be found by making use of the decomposition

$$p(v_{1:t}) = \prod_t p(v_t|v_{1:t-1}) \tag{23.2.28}$$

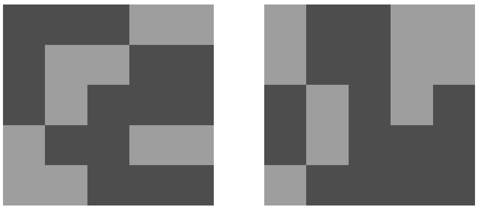
Each factor can be computed using

$$p(v_t|v_{1:t-1}) = \sum_{h_t} p(v_t, h_t|v_{1:t-1}) \tag{23.2.29}$$

$$= \sum_{h_t} p(v_t|h_t, \cancel{v_{1:t-1}})p(h_t|v_{1:t-1}) \tag{23.2.30}$$

$$= \sum_{h_t} p(v_t|h_t) \sum_{h_{t-1}} p(h_t|h_{t-1}, \cancel{v_{1:t-1}})p(h_{t-1}|v_{1:t-1}) \tag{23.2.31}$$

[2] It is most common to use this terminology for the continuous variable case, though we adopt it here also for the discrete variable case.



(a) 'Creaks' (b) 'Bumps'

Figure 23.6: Localising the burglar. The latent variable $h_t \in \{1, \ldots, 25\}$ denotes the positions, defined over the 5×5 grid of the ground floor of the house. **(a)**: A representation of the probability that the 'floor will creak' at each of the 25 positions, $p(v^{creak}|h)$. Light squares represent probability 0.9 and dark square 0.1. **(b)**: A representation of the probability $p(v^{bump}|h)$ that the burglar will bump into something in each of the 25 positions.

where the final term $p(h_{t-1}|v_{1:t-1})$ is the filtered result.

In both approaches the likelihood of an output sequence requires only a forward computation (filtering). If required, one can also compute the likelihood using, (23.2.19),

$$p(v_{1:T}) = \sum_{h_t} \alpha(h_t)\beta(h_t) \tag{23.2.32}$$

23.2.5 Most likely joint state

The most likely path $h_{1:T}$ of $p(h_{1:T}|v_{1:T})$ is the same as the most likely state (for fixed $v_{1:T}$) of

$$p(h_{1:T}, v_{1:T}) = \prod_t p(v_t|h_t)p(h_t|h_{t-1}) \tag{23.2.33}$$

The most likely path can be found using the max-product version of the factor graph or max-absorption on the Junction Tree. Alternatively, an explicit derivation can be obtained by considering:

$$\max_{h_T} \prod_{t=1}^{T} p(v_t|h_t)p(h_t|h_{t-1}) = \left\{ \prod_{t=1}^{T-1} p(v_t|h_t)p(h_t|h_{t-1}) \right\} \underbrace{\max_{h_T} p(v_T|h_T)p(h_T|h_{T-1})}_{\mu(h_{T-1})} \tag{23.2.34}$$

The message $\mu(h_{T-1})$ conveys information from the end of the chain to the penultimate timepoint. We can continue in this manner, defining the recursion

$$\mu(h_{t-1}) = \max_{h_t} p(v_t|h_t)p(h_t|h_{t-1})\mu(h_t), \qquad 2 \leq t \leq T \tag{23.2.35}$$

with $\mu(h_T) = 1$. This means that the effect of maximising over $h_2, \ldots, h_T$ is compressed into a message $\mu(h_1)$ so that the most likely state h_1^* is given by

$$h_1^* = \operatorname*{argmax}_{h_1} p(v_1|h_1)p(h_1)\mu(h_1) \tag{23.2.36}$$

Once computed, backtracking gives

$$h_t^* = \operatorname*{argmax}_{h_t} p(v_t|h_t)p(h_t|h_{t-1}^*)\mu(h_{t-1}) \tag{23.2.37}$$

This special case of the max-product algorithm is called the *Viterbi algorithm*.

Example 97 (A localisation example)**.** Consider the following situation: You're asleep upstairs in your house and hear some noises from downstairs. You realise that a burglar is on the ground floor and you attempt to understand where he his from listening to his movements. You mentally partition the ground floor into a 5×5 grid. For each grid position you know the probability that if someone is in that position the floorboard will creak, fig(23.6a). Similarly you know for each position the probability that someone will bump into something in the dark, fig(23.6b). The floorboard creaking and bumping into objects can occur independently. In addition you assume that the burglar will move only one grid square – forwards,

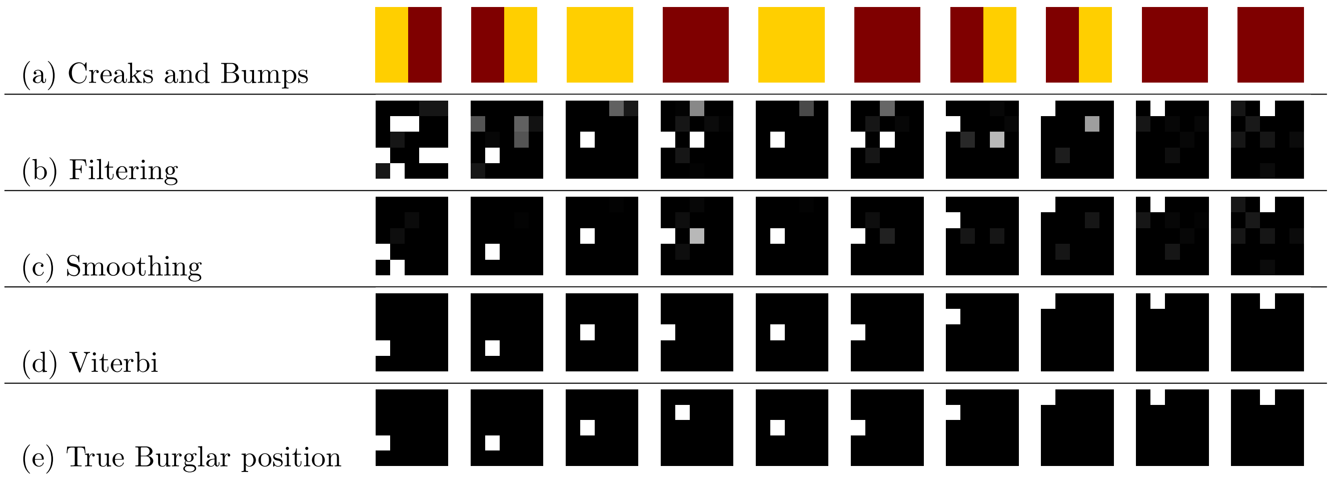


Figure 23.7: Localising the burglar through time for 10 time steps. **(a)**: Each panel represents the visible information $v_t = \left(v_t^{creak}, v_t^{bump}\right)$, where $v_t^{creak} = 1$ means that there was a 'creak in the floorboard' ($v_t^{creak} = 2$ otherwise) and $v_t^{bump} = 1$ meaning 'bumped into something' (and is in state 2 otherwise). There are 10 panels, one for each time $t = 1, \ldots, 10$. The left half of the panel represents v_t^1 and the right half v_t^2. The lighter colour represents the occurrence of a creak or bump, the darker colour the absence. **(b)**: The filtered distribution $p(h_t|v_{1:t})$ representing where we think the burglar is. **(c)**: The smoothed distribution $p(h_t|v_{1:10})$ so that we can figure out where we think the burglar went. **(d)**: The most likely (Viterbi) burglar path $\arg\max_{h_{1:10}} p(h_{1:10}|v_{1:10})$. **(e)**: The actual path of the burglar.

backwards, left or right in a single timestep. Based on a series of bump/no bump and creak/no creak information, fig(23.7a), you try to figure out based on your knowledge of the ground floor, where the burglar might be.

We can represent the scenario using a HMM where $h \in \{1, \ldots, 25\}$ denotes the grid square. In this case the visible variable has a factorised form. To use our standard code, however, we form a new visible variable with 4 states using

$$p(v|h) = p(v^{creak}|h)p(v^{bump}|h) \tag{23.2.38}$$

Based on the past information, our belief as to where the burglar might be is represented by the filtered distribution $p(h_t|v_{1:t})$, fig(23.7). After the burglar has left at $T = 10$, the police arrive and try to piece together where the burglar went, based on the sequence of creaks and bumps you provide. At any time t, the information as to where the burglar could have been is represented by the smoothed distribution $p(h_t|v_{1:10})$. The police's single best-guess for the sequence of burglar positions is provided by the most likely joint hidden state $\arg\max_{h_{1:10}} p(h_{1:10}|v_{1:10})$.

23.2.6 Self localisation and kidnapped robots

A robot has an internal grid-based map of its environment and for each location knows the likely sensor readings he would expect in that location. The robot is 'kidnapped' and placed somewhere in the environment. The robot then starts to move, gathering sensor information. Based on these readings $v_{1:t}$ and intended movements $m_{1:t}$, the robot attempts to figure out his location. Due to wheel slippage on the floor an intended action by the robot, such as 'move forwards', might not be successful. Given all the information the robot has, he would like to infer $p(h_t|v_{1:t}, m_{1:t})$.

This problem differs from the burglar scenario in that the robot now has knowledge of the intended movements he makes. This should give more information as to where he could be. One can view this as

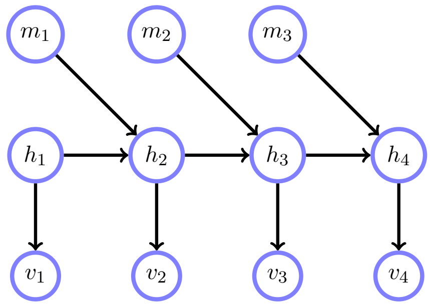


Figure 23.8: A model for robot self-localisation. At each time the robot makes an intended movement, m_t. As a generative model, knowing the intended movement m_t and the current grid position h_t, the robot has an idea of where he should be at the next time-step and what sensor reading v_{t+1} he would expect there. Based on only the sensor information $v_{1:T}$ and the intended movements $m_{1:T}$, the task is to infer a distribution over robot locations $p(h_{1:T}|m_{1:T}, v_{1:T})$.

extra 'visible' information, though it is more natural to think of this as additional input information. A model of this scenario is, see fig(23.8),

$$p(v_{1:T}, m_{1:T}, h_{1:T}) = \prod_{t=1}^{T} p(v_t|h_t)p(h_t|h_{t-1}, m_{t-1})p(m_t) \tag{23.2.39}$$

where for any time t, the visible variables $v_{1:t}$ are known, as are the intended movements $m_{1:t}$. The model expresses that the movements selected by the robot are random (hence no decision making in terms of where to go next). We assume that the robot has full knowledge of the conditional distributions defining the model (he knows the 'map' of his environment and all state transition and emission probabilities).

If our interest is only in localising the robot, since the inputs m are known, this model is in fact a form of time-dependent HMM:

$$p(v_{1:T}, h_{1:T}) = \prod_{t=1}^{T} p(v_t|h_t)p(h_t|h_{t-1}, t) \tag{23.2.40}$$

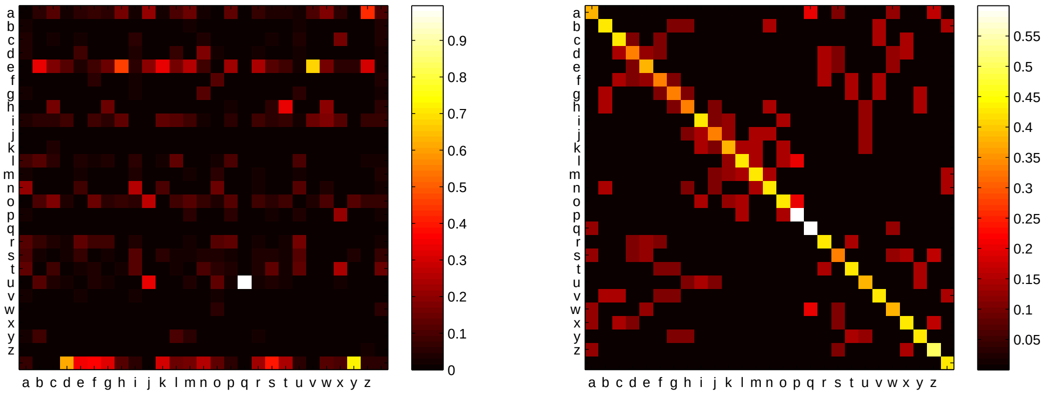
for a time-dependent transition $p(h_t|h_{t-1}, t)$. Any inference task required then follows the standard stationary HMM algorithms, albeit on replacing the time-independent transitions $p(h_t|h_{t-1})$ with the known time-dependent transitions.

In *self localisation and mapping* (SLAM) the robot does not know the map of his environment. This corresponds to having to learn the transition and emission distributions on-the-fly as he explores the environment.

Example 98 (Stubby fingers)**.** A with 'stubby fingers' typist has the tendency to hit either the correct key or a neighbouring key. For simplicity we assume that there are 27 keys: lower case a to lower case z and the space bar. To model this we use an emission distribution $B_{ij} = p(v = i|h = j)$ where $i = 1, \ldots, 27$, $j = 1, \ldots, 27$, as depicted in fig(23.9). A database of letter-to-next-letter frequencies (`www.data-compression.com/english.shtml`), yields the transition matrix $A_{ij} = p(h' = i|h = j)$ in English. For simplicity we assume that $p(h_1)$ is uniform. Also we assume that each intended key press results in a single press. Given a typed sequence `kezrninh` what is the most likely word that this corresponds to? By listing the 200 most likely hidden sequences and discarding those that are not in a standard English dictionary (`www.curlewcommunications.co.uk/wordlist.html`), the most likely word that was intended is `learning`. See `demoHMMbigram.m`.

23.3 Learning HMMs

Given a set of data $\mathcal{V} = \{\mathbf{v}^1, \ldots, \mathbf{v}^N\}$ of N sequences, where each sequence $\mathbf{v}^n = v^n_{1:T_n}$ of length T_n, we seek the HMM transition parameter $\mathbf{A}$, emission parameter $\mathbf{B}$, and initial parameter $\mathbf{a}$ most likely to have have generated $\mathcal{V}$. We make the i.i.d. assumption so that each sequence is independently generateed and



(a) (b)

Figure 23.9: **(a)**: The letter-to-letter transition matrix for English $p(h' = i|h = j)$. **(b)**: The letter emission matrix for a typist with 'stubby fingers' in which the key or its neighbours on the keyboard are likely to be hit.

assume that we know the number of hidden H states. For simplicity we concentrate here on the case of discrete visible variables, assuming also we know the number of states V.

23.3.1 EM algorithm

The application of EM to the HMM model is called the *Baum-Welch* algorithm.

M-step

Assuming i.i.d. data, according to the general EM approach, the M-step is given by maximising the 'energy':

$$\sum_{n=1}^{N} \langle \log p(v_1^n, v_2^n \ldots, v_T^n, h_1^n, h_2^n, \ldots h_T^n) \rangle_{p^{old}(\mathbf{h}^n|\mathbf{v}^n)} \tag{23.3.1}$$

with respect to the parameters $\mathbf{A}, \mathbf{B}, \mathbf{a}$; $\mathbf{h}^n$ denotes $h_{1:T_n}$. Using the form of the HMM, we obtain

$$\sum_{n=1}^{N} \left\{ \langle \log p(h_1) \rangle_{p^{old}(h_1|\mathbf{v}^n)} + \sum_{t=1}^{T_n-1} \langle \log p(h_{t+1}|h_t) \rangle_{p^{old}(h_t,h_{t+1}|\mathbf{v}^n)} + \sum_{t=1}^{T_n} \langle \log p(v_t^n|h_t) \rangle_{p^{old}(h_t|\mathbf{v}^n)} \right\} \tag{23.3.2}$$

where for compactness we dropped the sequence index from the h variables. To avoid potential confusion, we write $p^{new}(h_1 = i)$ to denote the (new) table entry for probability that the initial hidden variable is in state i. Optimising equation (23.3.2) with respect to $p(h_1)$, (and enforcing $p(h_1)$ to be a distribution) we obtain

$$a_i^{new} \equiv p^{new}(h_1 = i) = \frac{1}{N} \sum_{n=1}^{N} p^{old}(h_1 = i|\mathbf{v}^n) \tag{23.3.3}$$

which is the average number of times that the first hidden variable is in state i. Similarly,

$$A_{i',i}^{new} \equiv p^{new}(h_{t+1} = i'|h_t = i) \propto \sum_{n=1}^{N} \sum_{t=1}^{T_n-1} p^{old}(h_t = i, h_{t+1} = i'|\mathbf{v}^n) \tag{23.3.4}$$

which is the number of times that a transition from hidden state i to hidden state i' occurs, averaged over all times (since we assumed stationarity) and training sequences. The proportionality constant ensures the distribution is normalised,

$$A_{i',i}^{new} = \frac{\sum_{n=1}^{N} \sum_{t=1}^{T_n-1} p^{old}(h_t = i, h_{t+1} = i'|\mathbf{v}^n)}{\sum_{i'} \sum_{n=1}^{N} \sum_{t=1}^{T_n-1} p^{old}(h_t = i, h_{t+1} = i'|\mathbf{v}^n)} \tag{23.3.5}$$

Finally,

$$B^{new}_{j,i} \equiv p^{new}(v_t = j|h_t = i) \propto \sum_{n=1}^{N}\sum_{t=1}^{T_n} \mathbb{I}\left[v^n_t = j\right] p^{old}(h_t = i|\mathbf{v}^n) \tag{23.3.6}$$

which is the expected number of times that, for the observation being in state j, the hidden state is i. The proportionality is determined by the normalisation requirement.

E-step

In computing the M-step above the quantities $p^{old}(h_1 = i|\mathbf{v}^n)$, $p^{old}(h_t = i, h_{t+1} = i'|\mathbf{v}^n)$ and $p^{old}(h_t = i|\mathbf{v}^n)$ are obtained by inference using the techniques described in section(23.2.1).

Equations (23.3.3,23.3.5,23.3.6) are repeated until convergence. See `HMMem.m` and `demoHMMlearn.m`.

Parameter initialisation

The EM algorithm converges to a local maxima of the likelihood and, in general, there is no guarantee that the algorithm will find the global maximum. How best to initialise the parameters is a thorny issue, with a suitable initialisation of the emission distribution often being critical for success[225]. A practical strategy is to initialise the emission $p(v|h)$ based on first fitting a simpler non-temporal mixture model $\sum_h p(v|h)p(h)$ to the data.

Mixture emission

To make a richer emission model, one approach is use a mixture

$$p(v_t|h_t) = \sum_{k_t} p(v_t|k_t, h_t)p(k_t|h_t) \tag{23.3.7}$$

where k_t is a discrete summation variable. For inference on $h_{1:T}$, one simply replaces uses the mixture model emission $p(v_t|h_t)$. For learning, however, it is useful to consider the k_t as additional latent variables so that updates for each component of the emission model can be derived. To achieve this, consider the contribution to the energy from the emission:

$$E_v \equiv \sum_n \sum_{t=1}^{T} \left\langle \log p(v^n_t|h_t)\right\rangle_{q(h_t|v^n_{1:T})} \tag{23.3.8}$$

As it stands, the parameters of each component $p(v_t|k_t, h_t)$ are coupled in the above expression. One approach is to consider

$$\mathrm{KL}(q(k_t|h_t)|p(k_t|h_t, v_t)) \geq 0 \tag{23.3.9}$$

from which we immediately obtain the bound

$$\log p(v_t, h_t) \geq -\left\langle \log q(k_t|h_t)\right\rangle_{q(k_t|h_t)} + \left\langle \log p(v_t, k_t, h_t)\right\rangle_{q(k_t|h_t)} \tag{23.3.10}$$

and

$$\log p(v^n_t|h^n_t) \geq -\left\langle \log q(k_t|h^n_t)\right\rangle_{q(k_t|h^n_t)} + \left\langle \log p(v^n_t|k_t, h^n_t)\right\rangle_{q(k_t|h^n_t)} + \left\langle \log p(k_t|h^n_t)\right\rangle_{q(k_t|h^n_t)} \tag{23.3.11}$$

Using this in the energy contribution (23.3.8) we have the bound on the energy contribution

$$E_v \geq \sum_n \sum_{t=1}^{T} \left\langle -\left\langle \log q(k_t|h^n_t)\right\rangle_{q(k_t|h^n_t)} + \left\langle \log p(v^n_t|k_t, h^n_t)\right\rangle_{q(k_t|h^n_t)} + \left\langle \log p(k_t|h^n_t)\right\rangle_{q(k_t|h^n_t)} \right\rangle_{q(h^n_t|v^n_{1:T})} \tag{23.3.12}$$

We may now maximise this lower bound on the energy (instead of the energy itself). The contribution from each emission component $p(v = \mathsf{v}|h = \mathsf{h}, k = \mathsf{k})$ is

$$\sum_n \sum_{t=1}^{T} q(k_t = \mathsf{k}|h^n_t = \mathsf{h})q(h^n_t = \mathsf{h}|v^n_{1:T}) \log p(v^n_t|h = \mathsf{h}, k = \mathsf{k}) \tag{23.3.13}$$

The above can then be optimised (M-step) for fixed $q(k_t = \mathsf{k}|h_t^n = \mathsf{h})$, with these distributions updated using

$$q^{new}(k_t|h_t^n) \propto p(v^n|h_t^n, k_t)p(k_t|h_t) \tag{23.3.14}$$

The contribution to the energy bound from the mixture weights is given by

$$\log p(k = \mathsf{k}|h = \mathsf{h}) \sum_n \sum_{t=1}^{T} q(k_t = \mathsf{k}|h_t^n = \mathsf{h})q(h_t^n = \mathsf{h}|v_{1:T}^n) \tag{23.3.15}$$

so that the M-step update for the mixture weights is,

$$p(k = \mathsf{k}|h = \mathsf{h}) \propto \sum_n \sum_{t=1}^{T} q(k_t = \mathsf{k}|h_t^n = \mathsf{h})q(h_t^n = \mathsf{h}|v_{1:T}^n) \tag{23.3.16}$$

In this case the EM algorithm is composed of an 'emission' EM loop in which the transitions and $q(h_t^n = \mathsf{h}|v_{1:T}^n$ is fixed, during which the emissions $p(v|h, k)$ are learned, along with updating $q(k_t = \mathsf{k}|h_t^n = \mathsf{h})$. The 'transition' EM loop fixes the emission distribution $p(v_t|h_t)$ and learns the best transition $p(h_t|h_{t-1})$.

An alternative to the above derivation is to consider the k as hidden variables, and then use standard EM algorithm on the joint latent variables (h_t, k_t). The reader may show that the two approaches are equivalent.

23.3.2 Continuous observations

For a continuous vector observation $\mathbf{v}_t$, with $\dim \mathbf{v}_t = D$, we require a model $p(\mathbf{v}_t|h_t)$ mapping the discrete state h_t to a distribution over outputs. Using a continuous output does not change any of the standard inference message passing equations so that inference can be carried out for essentially arbitrarily complex emission distributions. Indeed, for inference alone, the normalisation of the emission $p(v|h)$ is not required. That is, filtering and smoothing can be carried out for $p(v|h) = \phi(v, h)/Z$ with unknown normalisation Z. The reason for this that we may normalise any marginal as desired, so that, for example

$$p(h_t|v_{1:t}) = \frac{1}{z}\alpha(h_t) \tag{23.3.17}$$

where z is a local normalisation constant. If the α messages are defined using $\phi(v, h)$ in place of the normalised $p(v|h) = \phi(v, h)/Z$, the effect of the missing Z can be absorbed into the unknown local normalisation z. For learning, however, the emission normalisation constant is required since this is a dependent on the parameters of the model.

23.3.3 The HMM-GMM

A common continuous observation mixture emission model component is a Gaussian

$$p(\mathbf{v}_t|k_t, h_t) = \mathcal{N}\left(\mathbf{v}_t|\boldsymbol{\mu}_{k_t,h_t}, \boldsymbol{\Sigma}_{k_t,h_t}\right) \tag{23.3.18}$$

so that k_t, h_t indexes the $K \times H$ mean vectors and covariance matrices. EM updates for these means and covariances are straightforward to derive from equation (23.3.12), see exercise(232). These models are common in tracking applications, in particular in speech recognition (usually under the constraint that the covariances are diagonal).

23.3.4 Discriminative training

HMMs can be used for supervised learning of sequences. That is, for each sequence $v_{1:T}^n$, we have a corresponding class label c^n. For example, we might associated a particular composer c with a sequence $v_{1:T}$ and wish to make a mode that will predict the composer for a novel music sequence. A generative approach to using HMMs for classification is to train a separate HMM for each class, $p(v_{1:T}|c)$ and subsequently use Bayes' rule to form the classification for a novel sequence $v^*_{1:T}$ using

$$p(c^*|v^*_{1:T}) = \frac{p(v^*_{1:T}|c^*)p(c^*)}{\sum_{c'=1}^{C} p(v^*_{1:T}|c')p(c')} \tag{23.3.19}$$

If the data is noisy and difficult to model, however, this generative approach may not work well since much of the expressive power of each model is used to model the complex data, rather than focussing on the decision boundary, which is ultimately of more interest.

In applications such as speech recognition, improvements in performance are often reported when the models are trained in a discriminative way. In discriminative training (see for example [148]) , one defines a new single discriminative model, formed from the C HMMs using

$$p(c|v_{1:T}) = \frac{p(v_{1:T}|c)p(c)}{\sum_{c'=1}^{C} p(v_{1:T}|c')p(c')} \tag{23.3.20}$$

and then maximises the likelihood of a set of observed classes and corresponding visible variables $v_{1:T}$. For a single data pair, $(c^n, v^n_{1:T})$,

$$\log p(c^n|v^n_{1:T}) = \underbrace{\log p(v^n_{1:T}|c)}_{\text{generative likelihood}} + \log p(c) - \log \sum_{c'=1}^{C} p(v^n_{1:T}|c')p(c') \tag{23.3.21}$$

The first term above represents the generative likelihood term, with the last term above accounting for the discrimination. Whilst deriving EM style updates is hampered by the discriminative terms, computing the gradient of the is straightforward using the technique described in section(11.7).

In some applications, a class label is available at each timepoint c_t, together with an observation v_t. Given a training sequence (or more generally a set of sequences) $v_{1:T}, c_{1:T}$ the aim is to find the optimal class sequence $c^*_{1:T}$ for a novel observation sequence $v^*_{1:T}$. One approach is to train a generative model

$$p(v_{1:T}, c_{1:T}) = \prod_t p(v_t|c_t)p(c_t|c_{t-1}) \tag{23.3.22}$$

although this approach may not be optimal in terms of class discrimination. A cheap surrogate is to train a discriminative classification model $\tilde{p}(c_t|v_t)$ separately. With this, one can form the emission (here written for continuous v_t)

$$p(v_t|c_t) = \frac{\tilde{p}(c_t|v_t)\tilde{p}(v_t)}{\int_{v_t} \tilde{p}(c_t|v_t)\tilde{p}(v_t)} \tag{23.3.23}$$

where $\tilde{p}(v_t)$ is user defined. Whilst computing the local normalisation $\int_{v_t} \tilde{p}(c_t|v_t)\tilde{p}(v_t)$ may appear problematic, if the only use of $p(v_t|c_t)$ is to find the optimal class sequence for a novel observation sequence $v^*_{1:T}$,

$$c^*_{1:T} = \underset{c_{1:T}}{\operatorname{argmax}}\ p(c_{1:T}|v^*_{1:T}) \tag{23.3.24}$$

then the local normalisations play no role since they are independent of c. Hence, during Viterbi decoding we may replace the term $p(v_t|h_t)$ with $\tilde{p}(c_t|v_t)$ without affecting the optimal sequence. Using a model in this way is a special case of the general hybrid procedure described in section(13.2.4). The approach is suboptimal since learning the classifier is divorced from learning the transition model. Nevertheless, this heuristic historically has support in the speech recognition community.

23.4 Related Models

23.4.1 Explicit duration model

For a HMM with transition $p(h_t = i|h_{t-1} = i) \equiv \gamma_i$, the probability that the dynamics stays in state i for τ timesteps is γ_i^τ, which decays exponentially with time. This is a general property of first order Markov

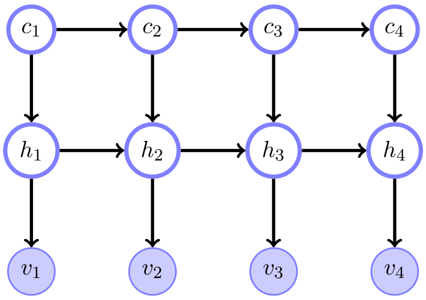


Figure 23.10: An explicit duration HMM. The counter variables c_t deterministically count down to zero. When they reach zero, a h transition is allowed, and the new value for c_t is sampled.

models of this form. In practice, however, we would often like to constrain the dynamics to remain in the same state for a minimum number of timesteps, or to have a specified duration distribution $p_{dur}(c_t)$. A simple way to enforce this is to use a latent counter variable c_t that at the beginning is initialised into the state sampled from $p_{dur}(c_t)$. Then at each timestep the counter decrements by 1, until it reaches 1, after which a new duration is sampled:

$$p(c_t|c_{t-1}) = \begin{cases} \delta(c_t, c_{t-1} - 1) & c_{t-1} > 1 \\ p_{dur}(c_t) & c_{t-1} = 1 \end{cases} \tag{23.4.1}$$

The state h_t can transition only when $c_t = 0$:

$$p(h_t|h_{t-1}, c_t) = \begin{cases} \delta(h_t, h_{t-1}) & c_t > 1 \\ p_{tran}(h_t|h_{t-1}) & c_t = 1 \end{cases} \tag{23.4.2}$$

In this way the joint latent variable (c_t, h_t) ensures that the duration can be sampled from an arbitrary duration distribution, see fig(23.10). Since $\dim(c_t, h_t) = D_{max}H$, naively the computational complexity of inference in this model scales as $O\left(TH^2D_{max}^2\right)$. However, when one runs the forward and backward equations, the deterministic nature of the transitions means that this can be reduced to $O\left(TH^2D_{max}\right)$[197]. See also exercise(233).

23.4.2 Hidden Semi-Markov models

In a standard HMM, for each state h_t we have a corresponding emission $p(v_t|h_t)$. Whilst we can extend this model by including additional past visible information, $p(v_t|h_t, v_{t-k:t-1})$, fundamentally the model remains Markovian in terms of the outputs v. In a hidden semi-Markov model, each state h_t emits a distribution $p(\mathcal{V}_t|h_t)$ on a segment of variables $\mathcal{V}_t$. The distribution within the segment is not necessarily Markovian, nor do the segments necessarily need to have equal length.

For example we could define a Gaussian process emission over scalars for a segment of length L_t:

$$p(\mathcal{V}_t|h_t = \mathsf{h}) = \mathcal{N}(\mathcal{V}_t|\mathbf{m}_t, \mathbf{S}_t) \tag{23.4.3}$$

where $\dim \mathbf{m}_t = L_t$ and $\dim \mathbf{S}_t = L_t \times L_t$.

Filtering

TO BE COMPLETED. A useful insight is note that we can always use the cascade representation

$$p(\mathcal{V}_t|h_t = \mathsf{h}) = \prod_{\tau=t-L_t}^{t} p(v_\tau|v_{t-L_t:\tau}, h_t = \mathsf{h}) \tag{23.4.4}$$

23.4.3 Input-Output HMM

The IOHMM[31] is a HMM augmented with additional input variables $x_{1:T}$, see fig(23.11). Each input can be continuous or discrete and modulates the transitions

$$p(v_{1:T}, h_{1:T}|x_{1:T}) = \prod_t p(v_t|h_t, x_t)p(h_t|h_{t-1}, x_t) \tag{23.4.5}$$

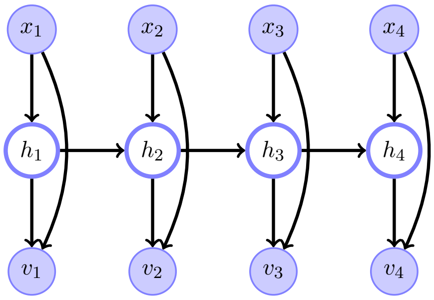


Figure 23.11: A first order input-output hidden Markov model. The input x and output v nodes are shaded to emphasise that their states are known during training. During testing, the inputs are known and the outputs are predicted.

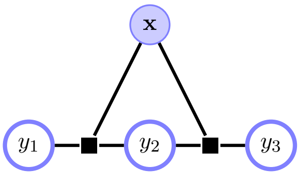


Figure 23.12: Linear Chain CRF. Since the input x is observed, the distribution is just a linear chain Factor Graph. The inference of pairwise marginals $p(y_t, y_{t-1}|x)$ is therefore straightforward using message passing.

The IOHMM may be used as a conditional predictor, where the outputs v_t represent the prediction at time t. (There are other ways to train this model, say by specifying a prediction only at the end of the sequence). In the case of continuous inputs and discrete outputs, the tables $p(v_t|h_t, x_t)$ and $p(h_t|h_{t-1}, x_t)$ are usually parameterised using a non-linear function, for example

$$p(v_t = y|h_t = h, x_t = x, \mathbf{w}) \propto e^{\mathbf{w}_{h,y}^{\mathsf{T}} x} \tag{23.4.6}$$

Inference then follows in a similar manner as for the standard HMM. The forward pass is given by

$$p(h_t|x_{1:t}, v_{1:t}) \propto \sum_{h_{t-1}} p(h_t, h_{t-1}, x_{1:t}, v_{1:t-1}, v_t) \tag{23.4.7}$$

$$= \sum_{h_{t-1}} p(v_t|v_{1:t-1}, x_{1:t}, h_t, h_{t-1}) p(h_t|v_{1:t-1}, x_{1:t}, h_{t-1}) p(v_{1:t-1}, x_{1:t}, h_{t-1}) \tag{23.4.8}$$

$$\propto p(v_t|x_t, h_t) \sum_{h_{t-1}} p(h_t|h_{t-1}, x_t) p(h_{t-1}|x_{1:t-1}, v_{1:t-1}) \tag{23.4.9}$$

The γ backward pass is

$$p(h_t|x_{1:T}, v_{1:T}) = \sum_{h_{t+1}} p(h_t, h_{t+1}|x_{1:t+1}, x_{t+2:T}, v_{1:T}) = \sum_{h_{t+1}} p(h_t|h_{t+1}, x_{1:t+1}, v_{1:t}) p(h_{t+1}|x_{1:T}, v_{1:T}) \tag{23.4.10}$$

We therefore need

$$p(h_t|h_{t+1}, x_{1:t+1}, v_{1:t}) = \frac{p(h_{t+1}, h_t|x_{1:t+1}, v_{1:t})}{p(h_{t+1}|x_{1:t+1}, v_{1:t})} = \frac{p(h_{t+1}|h_t, x_{t+1}) p(h_t|x_{1:t}, v_{1:t})}{\sum_{h_t} p(h_{t+1}|h_t, x_{t+1}) p(h_t|x_{1:t}, v_{1:t})} \tag{23.4.11}$$

The likelihood can be found as usual, either using an unnormalised α recursion, or using the cascade decomposition $p(v_{1:T}) = \prod_t p(v_t|v_{1:t-1})$.

Direction bias

Consider predicting the output distribution $p(v_t|x_{1:T})$ given both past and future input information $x_{1:T}$. Because the hidden states are unobserved, then $p(v_t|x_{1:T}) = p(v_t|x_{1:t})$. The prediction only uses past information and discards any future contextual information. Such 'direction bias' is identified in some sections of the literature (particularly in natural language modelling) as problematic and motivates the use of undirected models, such as conditional random fields.

23.4.4 Linear chain CRFs

Linear chain Conditional Random Fields (CRFs), are an extension of the unstructured CRFs we briefly discussed in section(9.4.6) and have application to modelling the distribution of a set of outputs $y_{1:T}$ given an input vector $\mathbf{x}$. For example, $\mathbf{x}$ might represent a sentence in English, and $y_{1:T}$ should represent the translation into French. Note that the vector $\mathbf{x}$ does not have to have dimension T. A first order linear chain CRF has the form

$$p(y_{1:T}|\mathbf{x}, \boldsymbol{\lambda}) = \frac{1}{Z(\mathbf{x}, \boldsymbol{\lambda})} \prod_{t=2}^{T} \phi_t(y_t, y_{t-1}, \mathbf{x}, \boldsymbol{\lambda}) \tag{23.4.12}$$

where $\boldsymbol{\lambda}$ are the free parameters of the potentials. In practice it is common to use potentials of the form

$$e^{\sum_{k=1}^{K} \lambda_k f_{k,t}(y_t, y_{t-1}, \mathbf{x})} \tag{23.4.13}$$

where $f_{k,t}(y_t, y_{t-1}, x)$ are 'features'. An example of such features was given in section(9.4.6). Given an input-output sequence, we can learn the optimal feature weights $\boldsymbol{\lambda}$. Faced with a new input $\mathbf{x}$, the most likely output sequence $y_{1:T}$ can be easily inferred (see below).

Given a set of input-output sequence pairs, $\mathbf{x}^n, y^n_{1:T}, n = 1, \ldots, N$ (assuming all sequenced have equal length T for simplicity), we can learn the parameters $\boldsymbol{\lambda}$ by Maximum Likelihood under the standard i.i.d. data assumption. The log likelihood is

$$L(\boldsymbol{\lambda}) = \sum_{t,n} \sum_{k} \lambda_k f_k(y^n_t, y^n_{t-1}, \mathbf{x}^n) - \sum_n \log Z(\mathbf{x}^n, \boldsymbol{\lambda}) \tag{23.4.14}$$

The reader may readily check that the log likelihood is concave so that the objective function has no local optima. The gradient is given by

$$\frac{\partial}{\partial \lambda_i} L = \sum_{n,t} \left(f_i(y^n_t, y^n_{t-1}, \mathbf{x}^n) - \langle f_i(y_t, y_{t-1}, \mathbf{x}^n) \rangle_{p(y_t, y_{t-1}|\mathbf{x}^n, \lambda)} \right) \tag{23.4.15}$$

Learning therefore requires inference of the marginal terms $p(y_t, y_{t-1}|\mathbf{x}, \boldsymbol{\lambda})$. Using

$$p(y|\mathbf{x}, \lambda) \propto \prod_{t=2}^{T} \phi_t(y_t, y_{t-1}), \qquad \phi_t(y_t, y_{t-1}) = e^{\sum_k \lambda_k f_k(y_t, y_{t-1}, \mathbf{x})} \tag{23.4.16}$$

we see this is a linear chain factor graph, see fig(23.12), for which inference of pairwise marginals is straightforward using message passing. One can therefore either use the standard factor graph message passing or derive an explicit algorithm, see exercise(227).

Finding the most likely output sequence for a novel input $\mathbf{x}^*$ is straightforward since

$$y^*_{1:T} = \underset{y_{1:T}}{\text{argmax}} \prod_t \phi_t(y_t, y_{t-1}, \mathbf{x}^*, \boldsymbol{\lambda}) \tag{23.4.17}$$

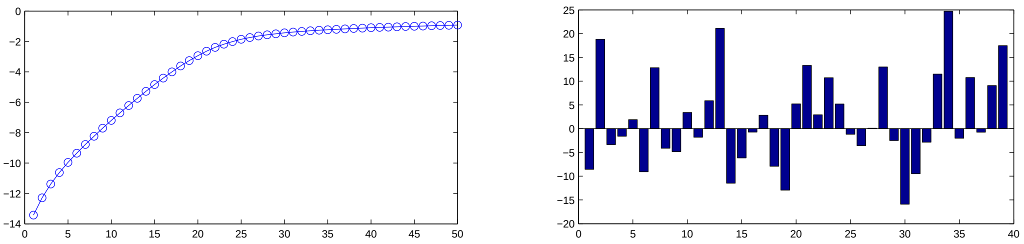
corresponds again to a simple linear chain, for which a max-product inference yields the required result, see also exercise(226).

In practice many hundreds of thousands of such functions are used ($K \gg 1$). This means that the storage of the Hessian is not feasible for Newton based training of a CRF and either limited memory methods or conjugate gradient techniques are used[285].

Example 99 (Linear Chain CRF)**.** As a model for the data in table(23.1), a linear CRF model has potentials $\phi(y_t, y_{t-1}, x_t) = \exp\left(\sum_i \lambda_i f_i(y_t, y_{t-1}, x_t)\right)$ where the binary feature functions are here simply defined by first mapping each of the $\dim x \times \dim y^2$ states to a unique integer $i(a, b, c)$ from 1 to $\dim x \times \dim y^2$

$$f_{i(a,b,c)}(y_t, y_{t-1}, x_t) = \mathbb{I}[y_t = a]\, \mathbb{I}[y_{t-1} = b]\, \mathbb{I}[x_t = c] \tag{23.4.18}$$

That is, each joint configuration of y_t, y_{t-1}, x_t is mapped to an index, and in this case the feature vector $\mathbf{f}$ will trivially have only a single non-zero entry. The evolution of the gradient ascent training algorithm is plotted in fig(23.13). In practice one would use richer feature functions defined to seek features of the input sequence $\mathbf{x}$ and also to produce a feature vector with more than one non-zero entry. See `demoLinearCRF.m`.



(a) (b)

Figure 23.13: Using a linear chain CRF to learn the sequences in table(23.1). **(a)**: The evolution of the log likelihood under gradient ascent. **(b)**: The learned parameter vector $\boldsymbol{\lambda}$ at the end of training.

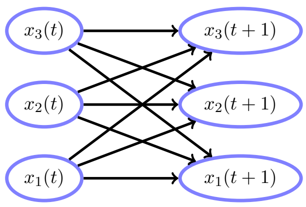


Figure 23.14: A Dynamic Bayesian Network. Possible transitions between variables at the same time-slice have not been shown.

23.4.5 Dynamic Bayesian networks

A DBN is defined as a Belief Network replicated through time,

$$p(\mathbf{x}_1, \ldots, \mathbf{x}_T) = \prod_{t=1}^{T} \prod_{i} p(x_i(t)|\mathbf{x}_{\backslash i}(t), \mathbf{x}(t-1)) \tag{23.4.19}$$

where $\mathbf{x}_{\backslash i}(t)$ denotes the set of variables at time t, expect for $x_i(t)$. At each time-step t there is a set of variables $x_i(t), i = 1, \ldots, X$, some of which may be observed. In a first order DBN, each variable $x_i(t)$ has parental variables taken from the set of variables in the previous time-slice, $\mathbf{x}_{t-1}$, or from the present time-slice. In most applications, the model is temporally homogeneous so that one may describe fully the distribution in terms of a two-time-slice model, fig(23.14). The generalisation to higher-order models is straightforward.

A *coupled HMM* is a special DBN that may be used to model coupled 'streams' of information, for example video and audio, see fig(23.15)[207].

23.5 Applications

23.5.1 Object Tracking

HMMs are used to track moving objects, based on an understanding of the dynamics of the object (encoded in the transition distribution) and an understanding of how an object with a known position would be observed (encoded in the emission distribution). Given an observed sequence, the hidden position can then be inferred. The burglar, example(10) is such a case in point, though HMMs are used to track people in videos, musical pitch *etc.*

7 3	4 1	7 3	2 1	3 2	4 3	5 3	7 1	3 2	5 1		
10 1	3 1	2 1									
9 2	8 3	3 3	9 3								
7 2	8 3	8 3	4 1	6 3	10 1	2 1	7 3	7 2	6 2	6 3	10 1
7 3	9 1	3 2	3 3	4 3	8 3	8 3					

Table 23.1: A subset of the 10 training input-output sequences. Each row contains an input x_t (upper entry) and output y_t (lower entry). There are 10 input states and 3 output states.

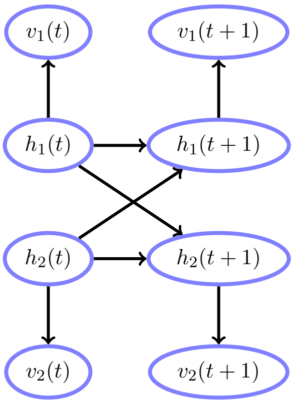


Figure 23.15: A Coupled HMM. For example the upper HMM might model speech, and the lower the corresponding video sequence. The upper hidden units then correspond to phonemes, and the lower to mouth positions; this model therefore captures the expected coupling between mouth positions and phonemes.

23.5.2 Automatic Speech Recognition

Most state-of-the-art speech recognition systems make heavy use of HMMs[296]. Roughly speaking, a continuous output vector $\mathbf{v}_t$ at time t, represents which frequencies are present in the speech signal in a small window around time t. These acoustic vectors are typically formed from taking a Discrete Fourier Transform of the speech signal over a small window around time t, with additional transformations to mimic human auditory processing, or related forms of linear coding of the observed acoustic waveform[129].

The corresponding discrete latent state h_t represents a phoneme – a basic unit of human speech (for which there are 44 for standard English). In traditional speech recognition, the setup is that a human linguist has painstakingly labelled the phoneme h_t for each time t and many different sequences. Given each acoustic vector $\mathbf{v}_t$ and an associated phoneme h_t, one may use maximum likelihood to fit a mixture of (usually isotropic) Gaussians $p(\mathbf{v}_t|h_t)$ to $\mathbf{v}_t$. This forms the emission distribution for a HMM.

Using the database of labelled phonemes, the phoneme transition $p(h_t|h_{t-1})$ can be learned (by simple counting) and forms the transition distribution for a HMM. (Note that in this case, since the 'hidden' variable h and observation v are known during training, training the HMM is straightforward and boils down to training the emission and transition distributions independently).

The idea is then that for a new sequence of 'acoustic' vectors $\mathbf{v}_{1:T}$ we can use the HMM to infer the most likely phoneme sequence through time, $\arg\max_{h_{1:T}} p(h_{1:T}|\mathbf{v}_{1:T})$, which takes into account the way that phonemes appear as acoustic vectors, and prior language constraints are encoded in the phoneme transitions.

- Most HMM models are trained on the assumption of 'clean' underlying speech. In practice noise corrupts the speech signal in a complex way, so that the resulting model is inappropriate, and performance degrades significantly.

- The fact that people speak at different speeds can be addressed using *time-warping* in which the latent phoneme remains in the same state for a number of timesteps.

- If the HMM is used to model a single word, it is natural to constrain the hidden state sequence to go 'forwards' through time, visiting a set of states in sequence (since the phoneme order for the word is known). In this case the structure of the transition matrices is upper triangular (or lower, depending on your definition), or even a banded triangular matrix. Such forward constraints describe a so-called *left-to-right transition matrix*.

23.5.3 BioInformatics

In the field of Bioinformatics HMMs have been widely applied to modelling genetic sequences. Multiple sequence alignment using forms of constrained HMMs have been particularly successful[160, 83].

23.5.4 Part-of-speech tagging

Consider the sentence below in which each word has been linguistically tagged

```
hospitality_NN is_BEZ an_AT excellent_JJ virtue_NN ,_,
but_CC not_XNOT when_WRB the_ATI guests_NNS have_HV
to_TO sleep_VB in_IN rows_NNS in_IN the_ATI cellar_NN !_!
```

The subscripts denote a linguistic tag, for example NN is the singular common noun tag, ATI is the article tag *etc.* Given a training set of such tagged sequences, the task is to tag a novel word sequence. One approach is to use h_t to be a tag, and v_t to be a word and fit a HMM to this data. For the training data, both the tags and words are observed so that Maximum Likelihood training of the transition and emission distribution is essentially trivial. Given a new sequence of words, the most likely tag sequence can be inferred using the Viterbi algorithm.

More recent part-of-speech taggers tend to use conditional random fields in which the input sequence $x_{1:T}$ is the sentence and the output sequence $y_{1:T}$ is the tag sequence. One possible parameterisation of for a linear chain CRF is to use a potential of the form $\phi(y_{t-1}, y_t)\phi(y_t, \mathbf{x})$ in which the first first factor encodes the grammatical structure of the language and the second the a priori likely tag y_t for an input sentence[163].

23.6 Code

`demoMixMarkov.m`: Demo for Mixture of Markov models
`mixMarkov.m`: Mixture of Markov models
`demoHMMinference.m`: Demo of HMM Inference
`HMMforward.m`: Forward α recursion
`HMMbackward.m`: Forward β recursion
`HMMgamma.m`: RTS γ 'correction' recursion
`HMMsmooth.m`: Single and Pairwise $\alpha - \beta$ smoothing
`HMMviterbi.m`: Most Likely State (Viterbi) algorithm
`demoHMMburglar.m`: Demo of Burglar Localisation
`demoHMMbigram.m`: demo of stubby fingers typing
`HMMem.m`: EM algorithm for HMM (Baum-Welch)
`demoHMMlearn.m`: demo of EM algorithm for HMM (Baum-Welch)
`demoLinearCRF.m`: demo of learning a linear chain CRF

The following linear chain CRF potential is particularly simple and in practice one would use a more complex one.
`linearCRFpotential.m`: Linear CRF potential
The following likelihood and gradient routines are valid for any linear CRF potential $\phi(y_{t-1}, y_t, \mathbf{x})$.
`linearCRFgrad.m`: Linear CRF gradient
`linearCRFloglik.m`: Linear CRF log likelihood

23.7 Exercises

Exercise 217. *Consider a stochastic matrix M_{ij} with $\sum_i M_{ij} = 1$. Consider an eigenvalue λ and eigenvector $\mathbf{e}$ such $\sum_j M_{ij} e_j = \lambda e_i$. By summing over i show that, provided $\sum_i e_i > 0$ then λ must be equal to 1.*

Exercise 218. *Consider the Markov chain with transition matrix*

$$\mathbf{M} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{23.7.1}$$

Show that this Markov chain does not have an equilibrium distribution and state a stationary distribution for this chain.

Exercise 219. *Consider a HMM with 3 states ($M = 3$) and 2 output symbols, with a left-to-right state transition matrix*

$$\mathbf{A} = \begin{pmatrix} 0.5 & 0.0 & 0.0 \\ 0.3 & 0.6 & 0.0 \\ 0.2 & 0.4 & 1.0 \end{pmatrix} \tag{23.7.2}$$

where $A_{ij} \equiv p(h(t+1) = i|h(t) = j)$, emission matrix $B_{ij} \equiv p(v(t) = i|h(t) = j)$

$$\mathbf{B} = \begin{pmatrix} 0.7 & 0.4 & 0.8 \\ 0.3 & 0.6 & 0.2 \end{pmatrix} \tag{23.7.3}$$

and initial state probability vector $\mathbf{a} = (0.9\ 0.1\ 0.0)^\mathsf{T}$. Given that the observed symbol sequence is $v_{1:3} = (0, 1, 1)$, compute

1. $p(v_{1:3})$
2. $p(h_1|v_{1:3})$.
3. *Find the most probable hidden state sequence* $\arg\max_{h_{1:3}} p(h_{1:3}|v_{1:3})$.

Exercise 220. *This exercise follows from example(98). Given the 27 long character string* `rgenmonleunosbpnntje vrancg` *typed with 'stubby fingers', what is the most likely correct English sentence intended? In the list of decoded sequences, what value is $\log p(h_{1:27}|v_{1:27})$ for this sequence? You will need to modify* `demoHMMbigram.m` *suitably.*

Exercise 221. *Show that if a transition probability $A_{ij} = p(h_t = i|h_{t-1} = j)$ in a HMM is set to initialised to zero during EM training, then it will remain at zero throughout training.*

Exercise 222. *Consider the problem : Find the most likely joint output sequence $v_{1:T}$ for a HMM. That is,*

$$\underset{v_{1:T}}{\text{argmax}}\ p(v_{1:T}) \tag{23.7.4}$$

where

$$p(h_{1:T}, v_{1:T}) = \prod_t p(v_t|h_t)p(h_t|h_{t-1}) \tag{23.7.5}$$

1. *Explain why a local message passing algorithm cannot, in general, be found for this problem and discuss the computational complexity of finding an exact solution.*
2. *Explain how to adapt the Expectation-Maximisation algorithm to form a recursive algorithm, with local message passing, to guarantee at each stage of the algorithm an improved joint output state.*

Exercise 223. *Explain how to train a HMM using EM, but with a constrained transition matrix. In particular, explain how to learn a transition matrix with an upper triangular structure.*

Exercise 224. *Write a program to fit a mixture of L^{th} order Markov models.*

Exercise 225.

1. *Run the demo* `demoMixMarkov.m` *which corresponds to the Gene Clustering example in the notes on Mixture of Markov models. Try to understand whether or not there are local minima in the clustering problem by running the routine more than once. If you get different results after doing several runs, which one should you prefer?*

2. *Using the correspondence $A = 1, C = 2, G = 3, T = 4$ define a 4×4 transition matrix p that produces sequences of the form*

$$A, C, G, T, A, C, G, T, A, C, G, T, A, C, G, T, \ldots \tag{23.7.6}$$

Now define a new transition matrix

```
pnew = 0.9*p + 0.1*ones(4)/4                                    (23.7.7)
```

Define a 4×4 transition matrix q that produces sequences of the form

$$T, G, C, A, T, G, C, A, T, G, C, A, T, G, C, A, \ldots \tag{23.7.8}$$

Now define a new transition matrix

```
qnew = 0.9*q + 0.1*ones(4)/4                                    (23.7.9)
```

Assume that the probability of being in the initial state of the Markov Chain $p(h_1)$ is constant for all four states A, C, G, T. What is the probability that the Markov Chain `pnew` *generated the sequence S given by*

$$S \equiv A, A, G, T, A, C, T, T, A, C, C, T, A, C, G, C \tag{23.7.10}$$

3. *Similarly what is the probability that S was generated by* `qnew`*? Does it make sense that S has a higher likelihood under pnew compared with qnew?*

4. *Using the function* `randgen.m`*, generate 100 sequences of length 16 from the Markov chain defined by* `pnew`*. Similarly, generate 100 sequences each of length 16 from the Markov chain defined by* `qnew`*.*

   Concatenate all these sequences into a cell array `v` *so that* `v`$\{1\}$ *contains the first sequence and* `v`$\{200\}$ *the last sequence.*

   Use `MixMarkov.m` *to learn the optimum Maximum Likelihood parameters that generated these sequences. Assume that there are $H = 2$ kinds of Markov Chain. The result returned in* `phgvn` *indicates the posterior probability that sequence* `n` *belongs to the two models. Do you agree with the solution found?*

5. *Take the sequence S as defined in equation (23.7.10). Define an emission distribution that has 4 output states such that*

$$p(v = i|h = j) = \begin{cases} 0.7 & i = j \\ 0.1 & i \neq j \end{cases} \tag{23.7.11}$$

Using this emission distribution and the transition given by `pnew` *defined in equation (23.7.7), adapt* `demoHMMinferenceSimple.m` *suitably to find the most likely hidden sequence $h^p_{1:16}$ that generated the observed sequence S.*

Repeat the above computation but for the transition `qnew` *to give $h^q_{1:16}$. Which hidden sequence – $h^p_{1:16}$ or $h^q_{1:16}$ is to be preferred? Justify your answer.*

Exercise 226. *Derive an algorithm that will find the most likely joint state*

$$\underset{h_1,\ldots,h_T}{\text{argmax}} \prod_{t=2}^{T} \phi_t(h_t, h_{t-1}) \tag{23.7.12}$$

for arbitrarily defined potentials $\phi_t(h_t, h_{t-1})$.

1. *First consider*

$$\max_{h_1,\ldots,h_T} \prod_{t=2}^{T} \phi_t(h_t, h_{t-1}) \tag{23.7.13}$$

Show that how the maximisation over h_T may be pushed inside the product and that the result of the maximisation can be interpreted as a message

$$\gamma_{T-1\leftarrow T}(h_{T-1}) \tag{23.7.14}$$

2. *Derive the recursion*

$$\gamma_{t-1\leftarrow t}(h_{t-1}) = \max_{h_t} \phi_t(h_t, h_{t-1})\gamma_{t\leftarrow t+1}(h_t) \tag{23.7.15}$$

3. *Explain how the above recursion enables the computation of*

$$\underset{h_1}{\text{argmax}} \prod_{t=2}^{T} \phi_t(h_t, h_{t-1}) \tag{23.7.16}$$

4. *Explain how once the most likely state for h_1 is computed, one may efficiently compute the remaining optimal states $h_2, \ldots, h_T$.*

Exercise 227. *Derive an algorithm that will compute pairwise marginals*

$$p(h_t, h_{t-1}) \tag{23.7.17}$$

from the joint distribution

$$p(h_{1:T}) \propto \prod_{t=2}^{T} \phi_t(h_t, h_{t-1}) \tag{23.7.18}$$

for arbitrarily defined potentials $\phi_t(h_t, h_{t-1})$.

1. *First consider*

$$\sum_{h_1,\ldots,h_T} \prod_{t=2}^{T} \phi_t(h_t, h_{t-1}) \tag{23.7.19}$$

Show that how the summation over h_1 may be pushed inside the product and that the result of the maximisation can be interpreted as a message

$$\alpha_{1\rightarrow 2}(h_2) = \sum_{h_1} \phi_2(h_1, h_2) \tag{23.7.20}$$

2. *Derive the recursion*

$$\alpha_{t-1\rightarrow t}(h_t) = \sum_{h_{t-1}} \phi_t(h_t, h_{t-1})\alpha_{t-2\rightarrow t-1}(h_{t-1}) \tag{23.7.21}$$

3. *Similarly, show that one can push the summation of h_T inside the product to define*

$$\beta_{T-1\leftarrow T}(h_{T-1}) = \sum_{h_T} \phi_T(h_T, h_{T-1}) \tag{23.7.22}$$

and that by pushing in h_{T-1} etc. *one can define messages*

$$\beta_{t\leftarrow t+1}(h_t) = \sum_{h_{t+1}} \phi_{t+1}(h_{t+1}, h_t)\beta_{t+1\leftarrow t+2}(h_{t+1}) \tag{23.7.23}$$

4. Show that

$$p(h_t, h_{t-1}) \propto \sum_{h_{t+1}} \alpha_{t-2\to t-1}(h_{t-1})\phi(h_t, h_{t-1})\beta_{t\leftarrow t+1}(h_t) \tag{23.7.24}$$

Exercise 228. *A second order HMM is defined as*

$$p^{HMM2}(h_1, \ldots, h_T, v_1, \ldots, v_T) = p(h_1)p(v_1|h_1)p(h_2|h_1)p(v_2|h_2)\prod_{t=3}^{T}[p(h_t|h_{t-1}, h_{t-2})p(v_t|h_t)] \tag{23.7.25}$$

Following a similar approach to the first order HMM, derive explicitly a message passing algorithm to compute the most likely joint state

$$\underset{h_1,\ldots,h_T}{\text{argmax}}\; p^{HMM2}(h_1, \ldots, h_T|v_1, \ldots, v_T) \tag{23.7.26}$$

Exercise 229. *Since the likelihood of the HMM can be computed using filtering only, in principle we do not need smoothing to maximise the likelihood (contrary to the EM approach). Explain how one could compute the gradient of the likelihood by the use of filtered information alone (*i.e. *using only a forward pass).*

Exercise 230. *Derive the EM updates for fitting a HMM with an emission distribution given by a mixture of multi-variate Gaussians.*

Exercise 231. *Consider the HMM defined on hidden variables $\mathcal{H} = \{h_1, \ldots, h_T\}$ and visible (observation) variables $\mathcal{V} = \{v_1, \ldots, v_T\}$*

$$p(\mathcal{V}, \mathcal{H}) = p(h_1)p(v_1|h_1)\prod_{t=2}^{T} p(h_t|h_{t-1})p(v_t|h_t) \tag{23.7.27}$$

Show that the posterior $p(\mathcal{H}|\mathcal{V})$ is a Markov chain

$$p(\mathcal{H}|\mathcal{V}) = \tilde{p}(h_1)\prod_{t=2}^{T} \tilde{p}(h_t|h_{t-1}) \tag{23.7.28}$$

where $\tilde{p}(h_t|h_{t-1})$ and $\tilde{p}(h_1)$ are suitably defined distributions.

Exercise 232. *For training a HMM with a Gaussian mixture emission (the HMM-GMM model) in section(23.3.3), derive the EM update formulae for the means and covariances:*

$$\boldsymbol{\mu}^{new}_{k,h} = \sum_{n=1}^{N}\sum_{t=1}^{T} \rho_{k,h}(t, n)\mathbf{v}^n_t \tag{23.7.29}$$

and

$$\boldsymbol{\mu}^{new}_{k,h} = \sum_{n=1}^{N}\sum_{t=1}^{T} \rho_{k,h}(t, n)\left(\mathbf{v}^n_t - \boldsymbol{\mu}_{k,h}\right)\left(\mathbf{v}^n_t - \boldsymbol{\mu}_{k,h}\right)^{\mathsf{T}} \tag{23.7.30}$$

where

$$\rho_{k,h}(t, n) = \frac{q(k_t = k|h^n_t = h)q(h^n_t = h|v^n_{1:T})}{\sum_n \sum_t q(k_t = k|h^n_t = h)q(h^n_t = h|v^n_{1:T})} \tag{23.7.31}$$

Exercise 233. *Consider the HMM duration model defined by equation (23.4.2) and equation (23.4.1) with emission distribution $p(v_t|h_t)$. Our interest is to derive a recursion for the filtered distribution*

$$\alpha_t(h_t, c_t) \equiv p(h_t, c_t, v_{1:t}) \tag{23.7.32}$$

1. *Show that :*

$$\alpha_t(h_t, c_t) = p(v_t|h_t) \sum_{h_{t-1}, c_{t-1}} p(h_t|h_{t-1}, c_t)p(c_t|c_{t-1})\alpha_{t-1}(h_{t-1}, c_{t-1}) \tag{23.7.33}$$

2. *Using this derive*

$$\frac{\alpha_t(h_t, c_t)}{p(v_t|h_t)} = \sum_{h_{t-1}} p(h_t|h_{t-1}, c)p(c_t|c_{t-1} = 1)\alpha_{t-1}(h_{t-1}, c_{t-1} = 1) + \sum_{h_{t-1}} p(h_t|h_{t-1}, c) \sum_{c_{t-1}=2}^{D_{max}} p(c|c_{t-1})\alpha_{t-1}(h_{t-1}, c_{t-1}) \tag{23.7.34}$$

3. *Show that the right hand side of the above can be written as*

$$\sum_{h_{t-1}} p(h_t|h_{t-1}, c_t = c)p(c_t = c|c_{t-1} = 1)\alpha_{t-1}(h_{t-1}, 1) + \mathbb{I}\left[c \neq D_{max}\right] \sum_{h_{t-1}} p(h_t|h_{t-1}, c)\alpha_{t-1}(h_{t-1}, c+1) \tag{23.7.35}$$

4. *Show that the recursion for α is then given by*

$$\alpha_t(h, 1) = p(v_t|h_t = h)p_{dur}(1) \sum_{h_{t-1}} p_{tran}(h|h_{t-1})\alpha_{t-1}(h_{t-1}, 1) + \mathbb{I}\left[D_{max} \neq 1\right] p(v_t|h_t = h) \sum_{h_{t-1}} p_{tran}(h_t|h_{t-1})\alpha_{t-1}(h_{t-1}, 2) \tag{23.7.36}$$

and for $c > 1$

$$\alpha_t(h, c) = p(v_t|h_t = h)\left\{p_{dur}(c)\alpha_{t-1}(h, 0) + \mathbb{I}\left[c \neq D_{max}\right]\alpha_{t-1}(h, c+1)\right\} \tag{23.7.37}$$

5. *Explain why the computational complexity of filtered inference in the duration model is $O\left(TH^2D_{max}\right)$.*
6. *Derive an efficient smoothing algorithm for this duration model.*

CHAPTER 24

Continuous-state Markov Models

24.1 Observed Linear Dynamical Systems

In many practical timeseries applications, the data is naturally continuous, particularly for models of the physical environment. Our interest here is to develop the theory for carrying out inference and learning for simple continuous-state Markov models. In contrast to discrete-state Markov models, chapter(23), continuous state distributions are not automatically closed under operations such as products and marginalisation. To make practical algorithms for which inference and learning can be carried efficiently, we therefore are heavily restricted in the form of the continuous transition $p(v_t|v_{t-1})$. A simple yet powerful class of such transitions are the linear dynamical systems, which we describe below.

A deterministic observed linear dynamical system[1] (OLDS) defines the temporal evolution of a vector $\mathbf{v}_t$ according to the discrete-time update equation

$$\mathbf{v}_{t+1} = \mathbf{A}_t \mathbf{v}_t \tag{24.1.1}$$

where $\mathbf{A}_t$ is the transition matrix at time t. For the case that $\mathbf{A}_t$ is invariant with t, the process is called stationary.

A motivation for studying OLDSs is that many equations that describe our physical world can be written as an OLDS. OLDSs are interesting since they are simple prediction models: if $\mathbf{v}_t$ describes the state of the environment today, $\mathbf{A}\mathbf{v}_t$ predicts the environment tomorrow. As such, these models, have widespread application in many branches of science, from engineering and physics to economics.

The OLDS equation (24.1.1) is deterministic so that if we specify $\mathbf{v}_1$, all future values $\mathbf{v}_2, \mathbf{v}_3, \ldots,$ are defined. Indeed, we have

$$\mathbf{v}_t = \mathbf{A}^{t-1}\mathbf{v}_1 = \mathbf{P}\mathbf{\Lambda}^{t-1}\mathbf{P}^{-1}\mathbf{v}_1 \tag{24.1.2}$$

where $\mathbf{\Lambda} = \text{diag}(\lambda_1, \ldots, \lambda_V)$, $V = \dim \mathbf{v}$, is the diagonal eigenvalue matrix, and $\mathbf{P}$ is the corresponding eigenvector matrix of $\mathbf{A}$. If $\lambda_i > 1$ then for large t, $\mathbf{v}_t$ will explode. On the other hand, if $\lambda_i < 1$, then λ_i^{t-1} will tend to zero. For stable systems we require therefore no eigenvalues of magnitude greater than 1 and only unit eigenvalues will contribute in long term. Note that the eigenvalues may be complex which corresponds to rotational behaviour, see exercise(234). In this case, for a stable system, we require the magnitude of all eigenvalues to be not greater than 1. More generally, we may consider noise in the $\mathbf{v}$ and define a stochastic OLDS.

[1] We use the terminology 'observed' LDS to differentiate from the more generate LDS state-space model. In some texts, however, the term LDS is applied to the models under discussion in this chapter.

Definition 111 (Observed Linear Dynamical System)**.**

$$\mathbf{v}_{t+1} = \mathbf{A}_t \mathbf{v}_t + \boldsymbol{\eta}_{t+1} \tag{24.1.3}$$

where $\boldsymbol{\eta}_{t+1}$ is a noise vector sampled from a Gaussian distribution,

$$\mathcal{N}\left(\boldsymbol{\eta}_{t+1} | \boldsymbol{\mu}_{t+1}, \boldsymbol{\Sigma}_{t+1}\right) \tag{24.1.4}$$

This is equivalent to a first order Markov model

$$p(\mathbf{v}_{t+1}|\mathbf{v}_t) = \mathcal{N}\left(\mathbf{v}_{t+1} | \mathbf{A}_t \mathbf{v}_t + \boldsymbol{\mu}_{t+1}, \boldsymbol{\Sigma}_{t+1}\right) \tag{24.1.5}$$

At $t = 1$ we have an initial distribution $p(\mathbf{v}_1) = \mathcal{N}\left(\mathbf{v}_1 | \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1\right)$. For $t > 1$ if the parameters are time-independent, $\boldsymbol{\mu}_t \equiv \boldsymbol{\mu}$, $\mathbf{A}_t \equiv \mathbf{A}$, $\boldsymbol{\Sigma}_t \equiv \boldsymbol{\Sigma}$, the process is called stationary.

24.1.1 Stationary distribution with noise

Consider the one-dimensional linear system

$$v_t = a v_{t-1} + \eta_t, \qquad \eta_t \sim \mathcal{N}\left(\eta_t | 0, \sigma_v^2\right) \tag{24.1.6}$$

If we start at some state v_1, and then for $t > 1$ keep sampling according to $v_t = a v_{t-1} + \eta_t$, does the distribution of the $v_t, t \gg 1$ tend to a steady, fixed distribution? Assuming that we can represent the distribution of v_{t-1} as a Gaussian with some mean μ_{t-1} and variance σ_{t-1}^2, $v_{t-1} \sim \mathcal{N}\left(v_{t-1} | \mu_{t-1}, \sigma_{t-1}^2\right)$, then

$$\langle v_t \rangle = a \langle v_{t-1} \rangle + \langle \eta_t \rangle \Rightarrow \mu_t = a \mu_{t-1} \tag{24.1.7}$$

$$\left\langle v_t^2 \right\rangle = \langle a v_{t-1} + \eta_t \rangle^2 = a^2 \left\langle v_{t-1}^2 \right\rangle + 2 \langle v_{t-1} \rangle \underbrace{\langle \eta_t \rangle}_{0} + \left\langle \eta_t^2 \right\rangle \Rightarrow \sigma_t^2 = a^2 \sigma_{t-1}^2 + \sigma_v^2 \tag{24.1.8}$$

so that

$$v_t \sim \mathcal{N}\left(v_t | a \mu_{t-1}, a^2 \sigma_{t-1}^2 + \sigma_v^2\right) \tag{24.1.9}$$

Assuming there is a fixed variance σ_∞^2 for the infinite time case, the stationary distribution is given by

$$\sigma_\infty^2 = a^2 \sigma_\infty^2 + \sigma_v^2, \qquad \Rightarrow \sigma_\infty^2 = \frac{\sigma_v^2}{1 - a^2} \tag{24.1.10}$$

Similarly, the mean is given by $\mu_\infty = a^\infty \mu_1$. If $a \geq 1$ the variance (and mean) increases indefinitely with t. For $a < 1$, the mean tends to zero yet the variance remains finite; even though the magnitude of v_{t-1} is decreased by a factor of a at each iteration, the additive noise on average boosts the magnitude so that it remains steady in the long run. More generally for a system updating a vector $\mathbf{v}_t$

$$\mathbf{v}_t = \mathbf{A} \mathbf{v}_{t-1} + \boldsymbol{\eta}_t \tag{24.1.11}$$

for the existence of a steady state we require that all eigenvalues of $\mathbf{A}$ must be ≤ 1.

24.2 Auto-regressive models

An important timeseries model that is closely related to the OLDS is the auto-regressive model. For the scalar case, and assuming stationarity, the model is

$$v_t = \sum_{l=1}^{L} a_l v_{t-l} + \eta_t, \qquad \eta_t \sim \mathcal{N}\left(\eta_t | \mu, \sigma^2\right) \tag{24.2.1}$$

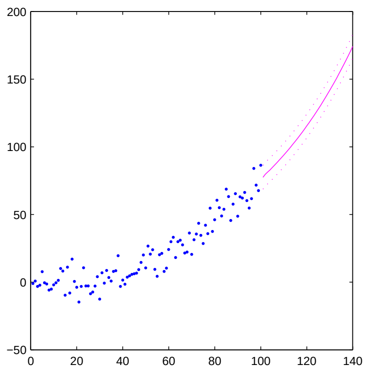


Figure 24.1: Fitting an order 3 AR model to the training points. The x axis represents time, and the y axis the value of the timeseries. The solid line is the mean prediction and the dashed lines ± one standard deviation. See `demoARtrain.m`

where $\mathbf{a}$ are called the AR coefficients and σ^2 is called the *innovation noise*. The model predicts the future based on a linear combination of the past L observations. As a Belief Network, the AR model can be written

$$p(v_{1:T}) = \prod_t p(v_t|v_{t-1}, \ldots, v_{t-L}) \tag{24.2.2}$$

with

$$p(v_t|v_{t-1}, \ldots, v_{t-L}) = \mathcal{N}\left(v_t \middle| \sum_{l=1}^{L} a_l v_{t-l}, \sigma^2\right) \tag{24.2.3}$$

AR models are heavily used in financial time-series prediction (see for example [269]), being able to capture simple trends in the data. Another common application area is in speech processing whereby for a one-dimensional speech signal partitioned into windows of length T, the AR coefficients best able to describe the signal in each window are used found[213]. These AR coefficients are then used to form a compressed representation of the signal, and transmitted for each window, rather than the original signal. The signal can then be reconstructed based on the AR coefficients[2].

24.2.1 Training an AR model

Maximum Likelihood training of the AR coefficients is straightforward based on

$$\log p(v_{1:T}) = \sum_{t=1}^{T} \log p(v_t|\hat{\mathbf{v}}_{t-1}) = -\frac{1}{2\sigma^2}\sum_{t=1}^{T}\left(v_t - \hat{\mathbf{v}}_{t-1}^{\mathsf{T}}\mathbf{a}\right)^2 - \frac{T}{2}\log(2\pi\sigma^2) \tag{24.2.4}$$

Differentiating *w.r.t.* $\mathbf{a}$ and equating to zero we arrive at

$$\sum_t \left(v_t - \hat{\mathbf{v}}_{t-1}^{\mathsf{T}}\mathbf{a}\right)\hat{\mathbf{v}}_{t-1} = 0 \tag{24.2.5}$$

so that optimally

$$\mathbf{a} = \left(\sum_t \hat{\mathbf{v}}_{t-1}\hat{\mathbf{v}}_{t-1}^{\mathsf{T}}\right)^{-1}\sum_t v_t \hat{\mathbf{v}}_{t-1} \tag{24.2.6}$$

These equations can be solved by Gaussian elimination. Similarly, optimally,

$$\sigma^2 = \frac{1}{T}\sum_{t=1}^{T}\left(v_t - \hat{\mathbf{v}}_{t-1}^{\mathsf{T}}\mathbf{a}\right)^2 \tag{24.2.7}$$

Above we assume that 'negative' timepoints are available in order to keep the notation simple. If times before the window over which we learn the coefficients are not available, a minor adjustment is required to start the summations from $t = L + 1$.

[2]Such a representation is used for example in telephones and known as a linear predictive vocoder, [254].

Example 100 (AR training)**.** In fig(24.1) we plot univariate timeseries data to which an AR model of order 3 was fitted. Given a value for $\mathbf{a}$, future predictions can be made using $v_{t+1} = \hat{\mathbf{v}}_t^{\mathsf{T}}\mathbf{a}$. As we see, the model is capable of capturing the local trend in the data.

24.2.2 AR model as an OLDS

We can write equation (24.2.1) as an OLDS using

$$\begin{pmatrix} v_t \\ v_{t-1} \\ \vdots \\ v_{t-L+1} \end{pmatrix} = \begin{pmatrix} a_1 & a_2 & \dots & a_L \\ 1 & 0 & \dots & 0 \\ \dots & 1 & \dots & 0 \\ 0 & \dots & 1 & 0 \end{pmatrix} \begin{pmatrix} v_{t-1} \\ v_{t-2} \\ \vdots \\ v_{t-L} \end{pmatrix} + \begin{pmatrix} \eta_t \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{24.2.8}$$

Introducing the vector of the L previous observations

$$\hat{\mathbf{v}}_{t-1} \equiv [v_{t-1}, v_{t-2}, \dots, v_{t-L}]^{\mathsf{T}} \tag{24.2.9}$$

equation (24.2.1) can be then be written as the OLDS

$$\hat{\mathbf{v}}_t = \mathbf{A}\hat{\mathbf{v}}_{t-1} + \boldsymbol{\eta}_t, \qquad \boldsymbol{\eta}_t \sim \mathcal{N}(\boldsymbol{\eta}_t|\mathbf{0}, \boldsymbol{\Sigma}) \tag{24.2.10}$$

where we define the block matrices

$$\mathbf{A} = \left(\begin{array}{c|c} a_{1:L-1} & a_L \\ \hline \mathbf{I} & \mathbf{0} \end{array}\right), \qquad \boldsymbol{\Sigma} = \left(\begin{array}{c|c} \sigma^2 & 0_{1,1:L-1} \\ \hline 0_{1:L-1,1} & 0_{1:L-1,1:L-1} \end{array}\right) \tag{24.2.11}$$

In this representation, the first component of the vector is updated according to the standard AR model, with the remaining components being copies of the previous values.

24.2.3 Non-stationary AR model

In most practical applications the AR coefficients are learned on the basis of a window of past data. An alternative is to view learning the AR coefficients as a problem in inference in a latent LDS, which is discussed in detail in section(24.3). If $\mathbf{a}_t$ are the latent AR coefficients, the term

$$v_t = \mathbf{a}_t^{\mathsf{T}}\hat{\mathbf{v}}_{t-1} + \eta_t, \qquad \eta_t \sim \mathcal{N}(\eta_t|0, \sigma^2) \tag{24.2.12}$$

can be viewed as the emission distribution of an latent LDS in which the hidden variable is $\mathbf{a}_t$ and the time-dependent emission matrix (vector in this case)is given by $\hat{\mathbf{v}}_{t-1}$. By placing a simple latent transition

$$\mathbf{a}_t = \mathbf{a}_{t-1} + \boldsymbol{\eta}_t^a, \qquad \boldsymbol{\eta}_t^a \sim \mathcal{N}(\eta_t^a|0, \sigma^2\mathbf{I}) \tag{24.2.13}$$

we encourage the AR coefficients to change slowly with time. This defines a model

$$p(v_{1:T}, \mathbf{a}_{1:T}) = \prod_t p(v_t|\mathbf{a}_t, \hat{\mathbf{v}}_{t-1})p(\mathbf{a}_t|\mathbf{a}_{t-1}) \tag{24.2.14}$$

Our interest is then in the conditional $p(\mathbf{a}_{1:T}|v_{1:T})$, from which we can compute the a-posteriori most likely sequence of AR coefficients. The above model is a form of non-stationary Latent Linear Dynamical system, as described in section(24.3), for which the emission matrix is given by $\hat{\mathbf{v}}_{t-1}$. Standard inference algorithms can then be applied to yield the time-varying AR coefficients, see `demoARlds.m`.

Definition 112 (Discrete Fourier Transform)**.** For a sequence $x_{0:N-1}$ the DFT $f_{0:N-1}$ is defined as

$$f_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N}kn}, \qquad k = 0, \dots, N-1 \tag{24.2.15}$$

f_k is a (complex) representation as to how much frequency k is present in the sequence $x_{0:N-1}$. The power of each component f_k is defined as the absolute length of the complex f_k.

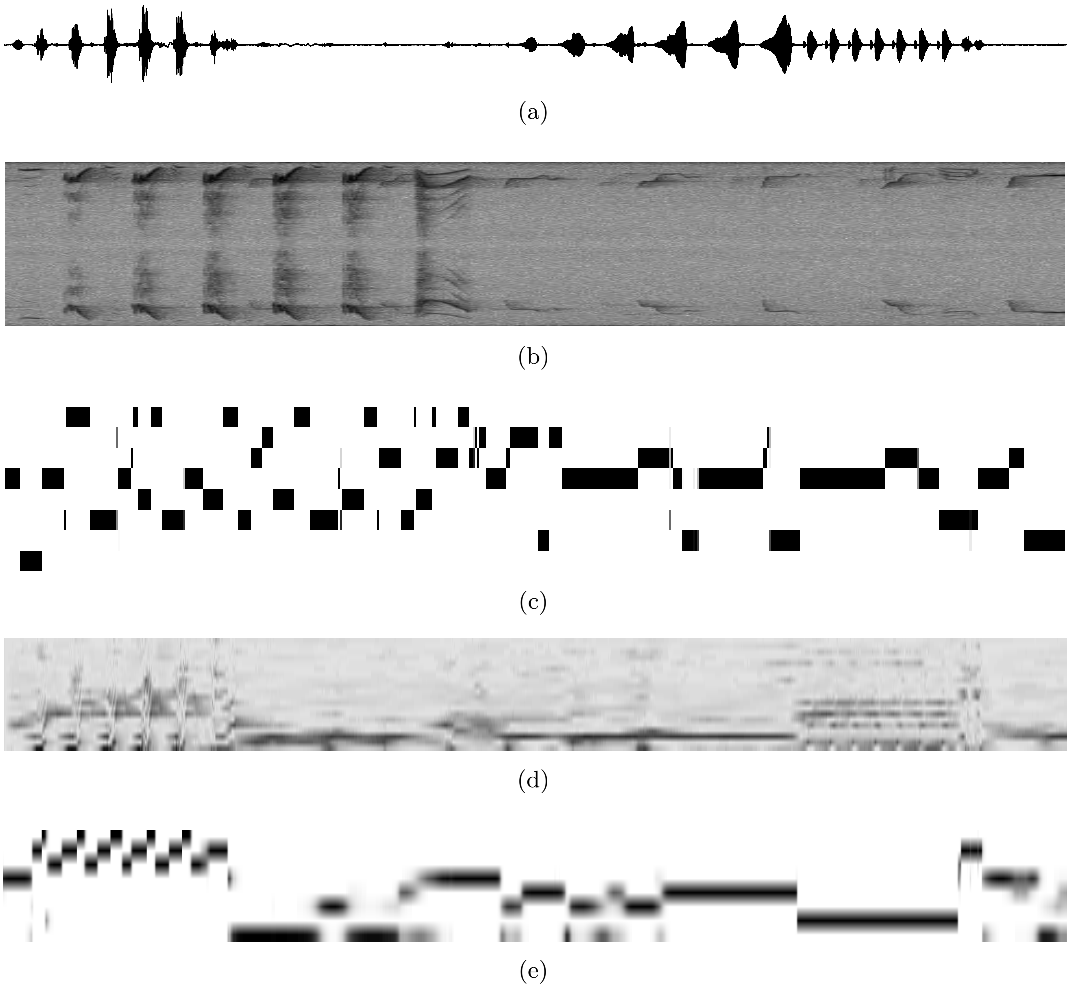


Figure 24.2: **(a)**: The raw recording of 5 seconds of a nightingale song (with additional background birdsong). **(b)**: Spectrogram of (a) up to 20,000 Hz. **(c)**: Clustering of the results in panel (b) using an 8 component Gaussian mixture model. **(d)**: The 20 AR coefficients learned using $\sigma_v^2 = 0.001$, $\sigma_h^2 = 0.001$, see `ARlds.m`. **(e)**: Clustering the results in panel (d) using a Gaussian mixture model with 8 components. The AR components group roughly according to the different song regimes.

Definition 113 (Spectrogram)**.** Given a timeseries $x_{1:T}$ the spectrogram at time t is a representation of the frequencies present in a window localised around t. For a window one computes the Discrete Fourier Transfor, from which we obtain a vector of log power in each frequency. The window is then moved (usually) one step forward and the DFT recomputed. Note that by taking the logarithm, small values in the original signal can translate to visibly appreciable values in the spectrogram.

Example 101 (Nightingale)**.** In fig(24.2a) we plot the raw acoustic recording for a 5 second fragment of a nightingale song[3]. The spectrogram is also plotted and gives an indication of which frequencies are present in the signal as a function of time. The nightingale song is very complicated but at least locally can be very repetitive. A crude way to find which segments repeat is to form a cluster analysis of the spectrogram. In fig(24.2c) we show the results of fitting a Gaussian mixture model, section(20.3), with 8 components, from which we see there is some repetition of components locally in time. An alternative representation of the signal is to use the time-varying AR coefficients, as plotted in fig(24.2d). These were found from embedding the AR model into a LDS whose latent variables are the AR coefficients, and performing smoothing. A GMM clustering with 8 components fig(24.2e) in this case produces a somewhat

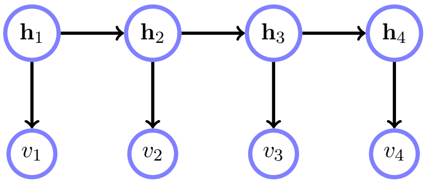


Figure 24.3: A (latent) LDS. Both hidden and visible variables are Gaussian Distributed.

clearer depiction of the different phases of the nightingale singing than that afforded by the spectrogram.

24.3 Latent Linear Dynamical Systems

The Latent LDS defines a stochastic linear dynamical system in a latent (or 'hidden') space on a sequence of vectors $\mathbf{h}_{1:T}$. The observations are taken as linear functions of these latent vectors. This model is also called a *linear Gaussian state space model*[4]. The model can also be considered a form of LDS on the joint variables $x_t = (v_t, h_t)$, with parts of the vector x_t missing. For this reason we will also refer to this model as a Linear Dynamical System (without the 'latent' prefix).

Definition 114 (Latent linear dynamical system)**.**

$$\begin{array}{lll} \mathbf{h}_t = \mathbf{A}_t \mathbf{h}_{t-1} + \boldsymbol{\eta}_t^h & \boldsymbol{\eta}_t^h \sim \mathcal{N}\left(\boldsymbol{\eta}_t^h \middle| \bar{\mathbf{h}}_t, \boldsymbol{\Sigma}_t^H\right) & \text{transition model} \\ \mathbf{v}_t = \mathbf{B}_t \mathbf{h}_t + \boldsymbol{\eta}_t^v & \boldsymbol{\eta}_t^v \sim \mathcal{N}\left(\boldsymbol{\eta}_t^v \middle| \bar{\mathbf{v}}_t, \boldsymbol{\Sigma}_t^V\right) & \text{emission model} \end{array} \tag{24.3.1}$$

where $\boldsymbol{\eta}_t^h$ and $\boldsymbol{\eta}_t^v$ are noise vectors. $\mathbf{A}_t$ is called the *transition matrix* and $\mathbf{B}_t$ the *emission matrix*. The terms $\bar{\mathbf{h}}_t$ and $\bar{\mathbf{v}}_t$ are the hidden and output bias respectively. The transition and emission models define a first order Markov model

$$p(\mathbf{h}_{1:T}, \mathbf{v}_{1:T}) = p(\mathbf{h}_1)p(\mathbf{v}_1|\mathbf{h}_1) \prod_{t=2}^{T} p(\mathbf{h}_t|\mathbf{h}_{t-1})p(\mathbf{v}_t|\mathbf{h}_t) \tag{24.3.2}$$

with the transitions and emissions defining local Gaussian distributions

$$p(\mathbf{h}_t|\mathbf{h}_{t-1}) = \mathcal{N}\left(\mathbf{h}_t \middle| \mathbf{A}_t \mathbf{h}_{t-1} + \bar{\mathbf{h}}_t, \boldsymbol{\Sigma}_t^H\right) \qquad p(\mathbf{h}_1) = \mathcal{N}\left(\mathbf{h}_1 | \boldsymbol{\mu}_\pi, \boldsymbol{\Sigma}_\pi\right) \tag{24.3.3}$$

$$p(\mathbf{v}_t|\mathbf{h}_t) = \mathcal{N}\left(\mathbf{v}_t \middle| \mathbf{B}_t \mathbf{h}_t + \bar{\mathbf{v}}_t, \boldsymbol{\Sigma}_t^V\right) \tag{24.3.4}$$

This (latent) LDS can be represented as a graphical model fig(24.3) with the extension to higher orders being intuitive. In the stationary case the parameters are time-independent.

Explicit expressions for the transition and emission distributions are given below for the stationary case with zero bias hidden and emissions. Each hidden variable is a multidimensional Gaussian distributed vector $\mathbf{h}_t$, with

$$p(\mathbf{h}_t|\mathbf{h}_{t-1}) = \frac{1}{\sqrt{|2\pi \boldsymbol{\Sigma}_H|}} \exp\left(-\frac{1}{2}\left(\mathbf{h}_t - \mathbf{A}\mathbf{h}_{t-1}\right)^\mathsf{T} \boldsymbol{\Sigma}_H^{-1} \left(\mathbf{h}_t - \mathbf{A}\mathbf{h}_{t-1}\right)\right) \tag{24.3.5}$$

which states that $\mathbf{h}_{t+1}$ has a mean equal to $\mathbf{A}\mathbf{h}_t$ and has Gaussian fluctuations described by the covariance matrix Σ_H. Similarly,

$$p(\mathbf{v}_t|\mathbf{h}_t) = \frac{1}{\sqrt{|2\pi \boldsymbol{\Sigma}_V|}} \exp\left(-\frac{1}{2}\left(\mathbf{v}_t - \mathbf{B}\mathbf{h}_t\right)^\mathsf{T} \boldsymbol{\Sigma}_V^{-1} \left(\mathbf{v}_t - \mathbf{B}\mathbf{h}_t\right)\right) \tag{24.3.6}$$

[4]These models are also often called Kalman Filters. We avoid this terminology here since the word 'filter' refers to a specific kind of inference and runs the risk of confusing a filtering algorithm with the model itself.

describes an output $\mathbf{v}_t$ with mean $\mathbf{B}\mathbf{h}_t$ and covariance $\mathbf{\Sigma}_V$. One may also include an external input observation $\mathbf{o}_t$ at each time, which will add $\mathbf{C}\mathbf{o}_t$ to the mean of the hidden variable and $\mathbf{D}\mathbf{o}_t$ to the mean of the output visible variable.

Example 102. Consider a dynamical system defined on two dimensional vectors $\mathbf{h}_t$:

$$\mathbf{h}_{t+1} = \mathbf{R}_\theta \mathbf{h}_t, \qquad \text{with } \mathbf{R}_\theta = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \tag{24.3.7}$$

$\mathbf{R}_\theta$ rotates the vector $\mathbf{h}_t$ through angle θ in one timestep. Under this LDS $\mathbf{h}$ will trace out points on a circle through time. By taking a scalar projection of $\mathbf{h}_t$, for example,

$$v_t = [\mathbf{h}_t]_1 = [1 \ \ 0]^\mathsf{T} \mathbf{h}_t, \tag{24.3.8}$$

the elements v_t, $t = 1, \ldots,$ describe a sinusoid through time. That sinusoids can be generated by LDSs is an important in signal processing in which one often wishes to find sinusoidal decompositions of signals. By using a block diagonal $\mathbf{R} = \text{blkdiag}(\mathbf{R}_{\theta_1}, \ldots, \mathbf{R}_{\theta_m})$ and taking a scalar projection of the extended $m \times 2$ dimensional $\mathbf{h}$ vector, one can construct a representation of a signal in terms of m sinusoidal components.

24.4 Inference

Given an observation sequence $v_{1:T}$ we wish to consider filtering and smoothing, as we did for the HMM, section(23.2.1). For the HMM, in deriving the various message passing recursions, we used only the independence structure encoded by the Belief Network. Since the LDS has the same independence structure as the HMM, we can use the same independence assumptions in deriving the updates for the LDS. However, in implementing them we need to deal with the issue that we now have continuous hidden variables, rather than discrete states. One approach to discretise continuous variables and subsequently use discrete messages as an approximation to continuous messages. In low-dimensional systems, this is feasible. However, in the LDS the fact that the distributions are Gaussian means that we can deal with continuous messages exactly. In translating the HMM message passing equations, we first replace summation with integration. For example, the filtering recursion (23.2.7) becomes

$$p(h_t|v_{1:t}) \propto \int_{h_{t-1}} p(v_t|h_t)p(h_t|h_{t-1})p(h_{t-1}|v_{1:t-1}), \qquad t > 1 \tag{24.4.1}$$

where we used the shorthand

$$\int_h f(h) \equiv \int_{-\infty}^{\infty} f(h)dh \tag{24.4.2}$$

Since the product of two Gaussians is another Gaussian, and the integral of a Gaussian is another Gaussian, the resulting $p(h_t|v_{1:t})$ is also Gaussian. This closure property of Gaussians means that we may represent $p(h_{t-1}|v_{1:t-1}) = \mathcal{N}(h_{t-1}|\mathbf{f}_{t-1}, \mathbf{F}_{t-1})$ with mean $\mathbf{f}_{t-1}$ and covariance $\mathbf{F}_{t-1}$. The effect of equation (24.4.1) is equivalent to updating the mean f_{t-1} and covariance F_{t-1} into a mean $\mathbf{f}_t$ and covariance $\mathbf{F}_t$ for $p(h_t|v_{1:t})$. Our task below is to find explicit algebraic formuale for these updates.

Numerical stability

Translating the message passing inference techniques we developed for the HMM into the LDS is largely straightforward. Indeed, one could simply run a standard sum-product algorithm (albeit for continuous variables), see `demoSumprodGaussCanonLDS.m`. However, the message updates involve the solution of linear equations for which numerical stability issues can arise. Routines based on a direct translation of a discrete message routine therefore need to be adopted with some care. In long timeseries, numerical instabilities can build up and may result in grossly inaccurate results, depending on the transition and emission distribution parameters and the method of implementing the message updates. For this reason specialised routines have been developed that are reasonably numerically stable under certain parameter regimes[280]. For example, for the HMM, in section(23.2.1), we discussed two alternative methods for smoothing, the parallel β approach, and the sequential γ approach. The β recursion is suitable when the emission and transition covariance entries are small, and the γ recursion usually preferable in the more standard case of small covariance values.

Analytical shortcuts

In deriving the inference recursions we need to frequently multiply and integrate Gaussians. Whilst in principle straightforward, this is tedious and wherever possible, it is useful to appeal to known shortcuts. For example, one can exploit the general result that the linear transform of a Gaussian random variable is another Gaussian distribution. Similarly it is convenient to make use of the conditioning formulae, as well as the dynamics reversal intuition. These results are described in section(8.6), and below were re-derive the most useful for our purposes here.

Consider a linear transformation of a Gaussian random variable:

$$\mathbf{y} = \mathbf{M}\mathbf{x} + \boldsymbol{\eta}, \qquad \boldsymbol{\eta} \sim \mathcal{N}(\boldsymbol{\eta}|\boldsymbol{\mu}, \boldsymbol{\Sigma}), \qquad \mathbf{x} \sim \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x) \tag{24.4.3}$$

where $\mathbf{x}$ and $\boldsymbol{\eta}$ are assumed to be generated from independent processes. What is the distribution of $\mathbf{y}$? One approach would be to write this formally as

$$p(\mathbf{y}) = \int \mathcal{N}(\mathbf{y}|\mathbf{M}\mathbf{x} + \boldsymbol{\mu}, \boldsymbol{\Sigma}) \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x)\, d\mathbf{x} \tag{24.4.4}$$

and carry out the integral (by completing the square). However, since a Gaussian variable under linear transformation is another Gaussian, we can take a shortcut and just find the mean and covariance of the transformed variable. Its mean is given by

$$\langle \mathbf{y} \rangle = \mathbf{M}\langle \mathbf{x} \rangle + \langle \boldsymbol{\eta} \rangle = \mathbf{M}\boldsymbol{\mu}_x + \boldsymbol{\mu} \tag{24.4.5}$$

To find the covariance, consider the displacement of a variable $\mathbf{h}$ from its mean, which we write as

$$\Delta\mathbf{h} \equiv \mathbf{h} - \langle \mathbf{h} \rangle \tag{24.4.6}$$

The covariance is, by definition, $\left\langle \Delta\mathbf{h}\Delta\mathbf{h}^{\mathsf{T}} \right\rangle$. For $\mathbf{y}$, the displacement is

$$\Delta\mathbf{y} = \mathbf{M}\Delta\mathbf{x} + \Delta\boldsymbol{\eta}, \tag{24.4.7}$$

So that the covariance is

$$\begin{aligned}
\left\langle \Delta\mathbf{y}\Delta\mathbf{y}^{\mathsf{T}} \right\rangle &= \left\langle (\mathbf{M}\Delta\mathbf{x} + \Delta\boldsymbol{\eta})(\mathbf{M}\Delta\mathbf{x} + \Delta\boldsymbol{\eta})^{\mathsf{T}} \right\rangle \\
&= \mathbf{M}\left\langle \Delta\mathbf{x}\Delta\mathbf{x}^{\mathsf{T}} \right\rangle \mathbf{M}^{\mathsf{T}} + \mathbf{M}\left\langle \Delta\mathbf{x}\Delta\boldsymbol{\eta}^{\mathsf{T}} \right\rangle + \left\langle \Delta\boldsymbol{\eta}\Delta\mathbf{x}^{\mathsf{T}} \right\rangle \mathbf{M}^{\mathsf{T}} + \left\langle \Delta\boldsymbol{\eta}\Delta\boldsymbol{\eta}^{\mathsf{T}} \right\rangle
\end{aligned}$$

Since the noises $\boldsymbol{\eta}$ and $\mathbf{x}$ are assumed independent, $\left\langle \Delta\boldsymbol{\eta}\Delta\mathbf{x}^{\mathsf{T}} \right\rangle = \mathbf{0}$ and we have

$$\boldsymbol{\Sigma}_y = \mathbf{M}\boldsymbol{\Sigma}_x\mathbf{M}^{\mathsf{T}} + \boldsymbol{\Sigma} \tag{24.4.8}$$

In deriving the filtering and smoothing recursions, we will make liberal use of these results.

24.4.1 Filtering

We represent the filtered distribution as a Gaussian with mean $\mathbf{f}_t$ and covariance $\mathbf{F}_t$,

$$p(\mathbf{h}_t|\mathbf{v}_{1:t}) \sim \mathcal{N}(\mathbf{h}_t|\mathbf{f}_t, \mathbf{F}_t) \tag{24.4.9}$$

This is called the *moment representation*. Our task is then to find a recursion for $\mathbf{f}_t, \mathbf{F}_t$ in terms of $\mathbf{f}_{t-1}$, $\mathbf{F}_{t-1}$. A convenient approach is to first find the joint distribution $p(\mathbf{h}_t, \mathbf{v}_t|\mathbf{v}_{1:t-1})$ and then condition on $\mathbf{v}_t$

Algorithm 20 LDS Forward Pass. Compute the filtered posteriors $p(\mathbf{h}_t|\mathbf{v}_{1:t}) \equiv \mathcal{N}(\mathbf{f}_t, \mathbf{F}_t)$ for a LDS with parameters $\theta_t = \left\{\mathbf{A}, \mathbf{B}, \mathbf{\Sigma}^h, \mathbf{\Sigma}^v, \bar{\mathbf{h}}, \bar{\mathbf{v}}\right\}_t$. The log-likelihood $L = \log p(\mathbf{v}_{1:T})$ is also returned.

```
{f_1, F_1, p_1} = LDSFORWARD(0, 0, v_1; θ_t)
F_0 ← 0, f_0 ← 0,
L ← log p_1
for t ← 2, T do
    {f_t, F_t, p_t} = LDSFORWARD(f_{t−1}, F_{t−1}, v_t; θ)
    L ← L + log p_t
end for
function LDSFORWARD(f, F, v; θ)
    μ_h ← Af + h̄,   μ_v ← Bμ_h + v̄                              ▷ Mean of p(h_t, v_t|v_{1:t−1})
    Σ_hh ← AFA^T + Σ^h,   Σ_vv ← BΣ_hh B^T + Σ^v,   Σ_vh ← BΣ_hh   ▷ Covariance of p(h_t, v_t|v_{1:t−1})
    f' ← μ_h + Σ_vh^T Σ_vv^{−1} (v − μ_v),   F' ← Σ_hh − Σ_vh^T Σ_vv^{−1} Σ_vh   ▷ Find p(h_t|v_{1:t}) by conditioning:
    p' ← exp(−½ (v − μ_v)^T Σ_vv^{−1} (v − μ_v)) / sqrt(det(2πΣ_vv))   ▷ Compute p(v_t|v_{1:t−1})
    return f', F', p'
end function
```

to find the distribution $p(\mathbf{h}_t|\mathbf{v}_{1:t})$. The term $p(\mathbf{h}_t, \mathbf{v}_t|\mathbf{v}_{1:t-1})$ is a Gaussian whose statistics can be found from the relations

$$\mathbf{v}_t = \mathbf{B}\mathbf{h}_t + \boldsymbol{\eta}_t^v, \qquad \mathbf{h}_t = \mathbf{A}\mathbf{h}_{t-1} + \boldsymbol{\eta}_t^h \tag{24.4.10}$$

Using the above, and assuming stationarity and zero biases, we readily find

$$\left\langle \Delta\mathbf{h}_t\Delta\mathbf{h}_t^\mathsf{T}|\mathbf{v}_{1:t-1}\right\rangle = \mathbf{A}\left\langle \Delta\mathbf{h}_{t-1}\Delta\mathbf{h}_{t-1}^\mathsf{T}|\mathbf{v}_{1:t-1}\right\rangle\mathbf{A}^\mathsf{T} + \mathbf{\Sigma}_H = \mathbf{A}\mathbf{F}_{t-1}\mathbf{A}^\mathsf{T} + \mathbf{\Sigma}_H \tag{24.4.11}$$

$$\left\langle \Delta\mathbf{v}_t\Delta\mathbf{h}_t^\mathsf{T}|\mathbf{v}_{1:t-1}\right\rangle = \mathbf{B}\left\langle \Delta\mathbf{h}_t\Delta\mathbf{h}_t^\mathsf{T}|\mathbf{v}_{1:t-1}\right\rangle = \mathbf{B}\left(\mathbf{A}\mathbf{F}_{t-1}\mathbf{A}^\mathsf{T} + \mathbf{\Sigma}_H\right) \tag{24.4.12}$$

$$\left\langle \Delta\mathbf{v}_t\Delta\mathbf{v}_t^\mathsf{T}|\mathbf{v}_{1:t-1}\right\rangle = \mathbf{B}\left\langle \Delta\mathbf{h}_t\Delta\mathbf{h}_t^\mathsf{T}|\mathbf{v}_{1:t-1}\right\rangle\mathbf{B}^\mathsf{T} + \mathbf{\Sigma}_V = \mathbf{B}\left(\mathbf{A}\mathbf{F}_{t-1}\mathbf{A}^\mathsf{T} + \mathbf{\Sigma}_H\right)\mathbf{B}^\mathsf{T} + \mathbf{\Sigma}_V \tag{24.4.13}$$

$$\langle\mathbf{v}_t|\mathbf{v}_{1:t-1}\rangle = \mathbf{B}\mathbf{A}\langle\mathbf{h}_{t-1}|\mathbf{v}_{1:t-1}\rangle, \qquad \langle\mathbf{h}_t|\mathbf{v}_{1:t-1}\rangle = \mathbf{A}\langle\mathbf{h}_{t-1}|\mathbf{v}_{1:t-1}\rangle \tag{24.4.14}$$

In the above, using our moment representation of the forward messages

$$\langle\mathbf{h}_{t-1}|\mathbf{v}_{1:t-1}\rangle \equiv \mathbf{f}_{t-1}, \qquad \left\langle \Delta\mathbf{h}_{t-1}\Delta\mathbf{h}_{t-1}^\mathsf{T}|\mathbf{v}_{1:t-1}\right\rangle \equiv \mathbf{F}_{t-1} \tag{24.4.15}$$

Then, using conditioning[5] $p(\mathbf{h}_t|\mathbf{v}_t, \mathbf{v}_{1:t-1})$ will have mean

$$\mathbf{f}_t \equiv \langle\mathbf{h}_t|\mathbf{v}_{1:t-1}\rangle + \left\langle \Delta\mathbf{h}_t\Delta\mathbf{v}_t^\mathsf{T}|\mathbf{v}_{1:t-1}\right\rangle\left\langle \Delta\mathbf{v}_t\Delta\mathbf{v}_t^\mathsf{T}|\mathbf{v}_{1:t-1}\right\rangle^{-1}(\mathbf{v}_t - \langle\mathbf{v}_t|\mathbf{v}_{1:t-1}\rangle) \tag{24.4.16}$$

and covariance

$$\mathbf{F}_t \equiv \left\langle \Delta\mathbf{h}_t\Delta\mathbf{h}_t^\mathsf{T}|\mathbf{v}_{1:t-1}\right\rangle - \left\langle \Delta\mathbf{h}_t\Delta\mathbf{v}_t^\mathsf{T}|\mathbf{v}_{1:t-1}\right\rangle\left\langle \Delta\mathbf{v}_t\Delta\mathbf{v}_t^\mathsf{T}|\mathbf{v}_{1:t-1}\right\rangle^{-1}\left\langle \Delta\mathbf{v}_t\Delta\mathbf{h}_t^\mathsf{T}|\mathbf{v}_{1:t-1}\right\rangle \tag{24.4.17}$$

Writing out the above explicitly we have for the mean:

$$\mathbf{f}_t = \mathbf{A}\mathbf{f}_{t-1} + \mathbf{P}\mathbf{B}^\mathsf{T}\left(\mathbf{B}\mathbf{P}\mathbf{B}^\mathsf{T} + \mathbf{\Sigma}_V\right)^{-1}(\mathbf{v}_t - \mathbf{B}\mathbf{A}\mathbf{f}_{t-1}) \tag{24.4.18}$$

and covariance

$$\mathbf{F}_t = \mathbf{P} + \mathbf{\Sigma}_H - \mathbf{P}\mathbf{B}^\mathsf{T}\left(\mathbf{B}\mathbf{P}\mathbf{B}^\mathsf{T} + \mathbf{\Sigma}_V\right)^{-1}\mathbf{B}\mathbf{P} \tag{24.4.19}$$

where

$$\mathbf{P} \equiv \mathbf{A}\mathbf{F}_{t-1}\mathbf{A}^\mathsf{T} + \mathbf{\Sigma}_H \tag{24.4.20}$$

[5] $p(\mathbf{x}|\mathbf{y})$ is a Gaussian with mean $\boldsymbol{\mu}_x + \mathbf{\Sigma}_{xy}\mathbf{\Sigma}_{yy}^{-1}(\mathbf{y} - \boldsymbol{\mu}_y)$ and covariance $\mathbf{\Sigma}_{xx} - \mathbf{\Sigma}_{xy}\mathbf{\Sigma}_{yy}^{-1}\mathbf{\Sigma}_{yx}$.

The filtering procedure is presented in algorithm(20) and a single update in `LDSforwardUpdate.m`.

One can write the covariance update as

$$\mathbf{F}_t = (\mathbf{I} - \mathbf{KB})\,\mathbf{P} \tag{24.4.21}$$

where we define the *Kalman gain* matrix

$$\mathbf{K} = \mathbf{P}\mathbf{B}^\mathsf{T}\left(\mathbf{\Sigma}_V + \mathbf{B}\mathbf{P}\mathbf{B}^\mathsf{T}\right)^{-1} \tag{24.4.22}$$

We present in algorithm(22) the recursion in standard engineering notation. See also `LDSsmooth.m`. The iteration is expected to be numerically stable when the noise covariances are small.

Symmetrising the updates

A potential numerical issue with the covariance update equation (24.4.21) is that it is the difference of two positive definite matrices. If there are numerical errors, the $\mathbf{F}_t$ may not be positive definite, nor symmetric.

Using the Woodbury identity, definition(132), equation (24.4.19) can be written more compactly as

$$\mathbf{F}_t = \left(\mathbf{P}^{-1} + \mathbf{B}^\mathsf{T}\mathbf{\Sigma}_V^{-1}\mathbf{B}\right)^{-1} \tag{24.4.23}$$

Numerically, however, this form is less useful since it involves two matrix inversions.

An alternative is to use the definition of $\mathbf{K}$, from which we can write

$$\mathbf{K}\mathbf{\Sigma}_V\mathbf{K}^\mathsf{T} = (\mathbf{I} - \mathbf{KB})\,\mathbf{P}\mathbf{B}^\mathsf{T}\mathbf{K}^\mathsf{T} \tag{24.4.24}$$

hence we arrive at *Joseph's symmetrized update*[101]

$$(\mathbf{I} - \mathbf{KB})\,\mathbf{P}\,(\mathbf{I} - \mathbf{KB})^\mathsf{T} + \mathbf{K}\mathbf{\Sigma}_V\mathbf{K}^\mathsf{T} \equiv (\mathbf{I} - \mathbf{KB})\left(\mathbf{P}\,(\mathbf{I} - \mathbf{KB})^\mathsf{T} + \mathbf{P}\mathbf{B}^\mathsf{T}\mathbf{K}^\mathsf{T}\right) \equiv (\mathbf{I} - \mathbf{KB})\,\mathbf{P} \tag{24.4.25}$$

The left hand side is the addition of two positive definite matrices so that the resulting update for the covariance is more numerically stable. A similar method can be used in the backward pass below. An alternative is to avoid using covariance matrices directly and use their square root as the parameter, deriving updates for these instead[223, 38].

24.4.2 Smoothing : Rauch-Tung-Striebel correction method

The smoothed posterior $p(\mathbf{h}_t|\mathbf{v}_{1:T})$ is necessarily a Gaussian since it is the conditional marginal of a larger Gaussian . By represent the posterior as a Gaussian with mean $\mathbf{g}_t$ and covariance $\mathbf{G}_t$,

$$p(\mathbf{h}_t|\mathbf{v}_{1:T}) \sim \mathcal{N}\left(\mathbf{h}_t|\mathbf{g}_t, \mathbf{G}_t\right) \tag{24.4.26}$$

we can form a recursion for $\mathbf{g}_t$ and $\mathbf{G}_t$ as follows:

$$p(\mathbf{h}_t|\mathbf{v}_{1:T}) = \int p(\mathbf{h}_t, \mathbf{h}_{t+1}|\mathbf{v}_{1:T}) \tag{24.4.27}$$

$$\propto \int_{\mathbf{h}_{t+1}} p(\mathbf{h}_t|\mathbf{v}_{1:T}, \mathbf{h}_{t+1})p(\mathbf{h}_{t+1}|\mathbf{v}_{1:T}) \propto \int_{\mathbf{h}_{t+1}} p(\mathbf{h}_t|\mathbf{v}_{1:t}, \mathbf{h}_{t+1})p(\mathbf{h}_{t+1}|\mathbf{v}_{1:T}) \tag{24.4.28}$$

The term $p(\mathbf{h}_t|\mathbf{v}_{1:t}, \mathbf{h}_{t+1})$ can be found by conditioning the joint distribution

$$p(\mathbf{h}_t, \mathbf{h}_{t+1}|\mathbf{v}_{1:t}) = p(\mathbf{h}_{t+1}|\mathbf{h}_t, \cancel{\mathbf{v}_{1:t}})p(\mathbf{h}_t|\mathbf{v}_{1:t}) \tag{24.4.29}$$

which is obtained in the usual manner by finding its mean and covariance: The term $p(\mathbf{h}_t|\mathbf{v}_{1:t})$ is a known Gaussian from filtering with mean $\mathbf{f}_t$ and covariance $\mathbf{F}_t$. Hence the joint distribution $p(\mathbf{h}_t, \mathbf{h}_{t+1}|\mathbf{v}_{1:t})$ has means

$$\langle \mathbf{h}_t|\mathbf{v}_{1:t}\rangle = \mathbf{f}_t, \qquad \langle \mathbf{h}_{t+1}|\mathbf{v}_{1:t}\rangle = \mathbf{A}\mathbf{f}_t \tag{24.4.30}$$

and covariance elements

$$\left\langle \Delta \mathbf{h}_t \Delta \mathbf{h}_t^{\mathsf{T}} | \mathbf{v}_{1:t} \right\rangle = \mathbf{F}_t, \quad \left\langle \Delta \mathbf{h}_t \Delta \mathbf{h}_{t+1}^{\mathsf{T}} | \mathbf{v}_{1:t} \right\rangle = \mathbf{F}_t \mathbf{A}^{\mathsf{T}}, \quad \left\langle \Delta \mathbf{h}_{t+1} \Delta \mathbf{h}_{t+1}^{\mathsf{T}} | \mathbf{v}_{1:t} \right\rangle = \mathbf{A} \mathbf{F}_t \mathbf{A}^{\mathsf{T}} + \mathbf{\Sigma}_H \tag{24.4.31}$$

To find the conditional distribution $p(\mathbf{h}_t|\mathbf{v}_{1:t}, \mathbf{h}_{t+1})$, we may use the conditioned Gaussian results, definition(78). It is particularly useful, however, to write the conditioning using the system reversal result, section(8.6.1), which interprets $p(\mathbf{h}_t|\mathbf{v}_{1:t}, \mathbf{h}_{t+1})$ as an equivalent linear system going backwards in time:idxlinear dynamical system!dynamics reversal

$$\mathbf{h}_t = \overleftarrow{\mathbf{A}}_t \mathbf{h}_{t+1} + \overleftarrow{\mathbf{m}}_t + \overleftarrow{\boldsymbol{\eta}}_t \tag{24.4.32}$$

where

$$\overleftarrow{\mathbf{A}}_t \equiv \left\langle \Delta \mathbf{h}_t \Delta \mathbf{h}_{t+1}^{\mathsf{T}} | \mathbf{v}_{1:t} \right\rangle \left\langle \Delta \mathbf{h}_{t+1} \Delta \mathbf{h}_{t+1}^{\mathsf{T}} | \mathbf{v}_{1:t} \right\rangle^{-1} \tag{24.4.33}$$

$$\overleftarrow{\mathbf{m}}_t \equiv \langle \mathbf{h}_t | \mathbf{v}_{1:t} \rangle - \left\langle \Delta \mathbf{h}_t \Delta \mathbf{h}_{t+1}^{\mathsf{T}} | \mathbf{v}_{1:t} \right\rangle \left\langle \Delta \mathbf{h}_{t+1} \Delta \mathbf{h}_{t+1}^{\mathsf{T}} | \mathbf{v}_{1:t} \right\rangle^{-1} \langle \mathbf{h}_{t+1} | \mathbf{v}_{1:t} \rangle \tag{24.4.34}$$

and $\overleftarrow{\boldsymbol{\eta}}_t \sim \mathcal{N}\left(\overleftarrow{\boldsymbol{\eta}}_t | \mathbf{0}, \overleftarrow{\mathbf{\Sigma}}_t\right)$, where

$$\overleftarrow{\mathbf{\Sigma}}_t \equiv \left\langle \Delta \mathbf{h}_t \Delta \mathbf{h}_t^{\mathsf{T}} | \mathbf{v}_{1:t} \right\rangle - \left\langle \Delta \mathbf{h}_t \Delta \mathbf{h}_{t+1}^{\mathsf{T}} | \mathbf{v}_{1:t} \right\rangle \left\langle \Delta \mathbf{h}_{t+1} \Delta \mathbf{h}_{t+1}^{\mathsf{T}} | \mathbf{v}_{1:t} \right\rangle^{-1} \left\langle \Delta \mathbf{h}_{t+1} \Delta \mathbf{h}_t^{\mathsf{T}} | \mathbf{v}_{1:t} \right\rangle \tag{24.4.35}$$

With the dynamics reversal, equation (24.4.32) and assuming that $\mathbf{h}_{t+1}$ is Gaussian distributed, it is then straightforward to work out the statistics of $p(\mathbf{h}_t|\mathbf{v}_{1:T})$. The mean is given by

$$\mathbf{g}_t \equiv \langle \mathbf{h}_t | \mathbf{v}_{1:T} \rangle = \overleftarrow{\mathbf{A}}_t \langle \mathbf{h}_{t+1} | \mathbf{v}_{1:T} \rangle + \overleftarrow{\mathbf{m}}_t \equiv \overleftarrow{\mathbf{A}}_t \mathbf{g}_{t+1} + \overleftarrow{\mathbf{m}}_t \tag{24.4.36}$$

and covariance

$$\mathbf{G}_t \equiv \left\langle \Delta \mathbf{h}_t \Delta \mathbf{h}_t^{\mathsf{T}} | \mathbf{v}_{1:T} \right\rangle = \overleftarrow{\mathbf{A}}_t \left\langle \Delta \mathbf{h}_{t+1} \Delta \mathbf{h}_{t+1}^{\mathsf{T}} | \mathbf{v}_{1:T} \right\rangle \overleftarrow{\mathbf{A}}_t^{\mathsf{T}} + \overleftarrow{\mathbf{\Sigma}}_t \equiv \overleftarrow{\mathbf{A}}_t \mathbf{G}_{t+1} \overleftarrow{\mathbf{A}}_t^{\mathsf{T}} + \overleftarrow{\mathbf{\Sigma}}_t \tag{24.4.37}$$

This procedure is equivalent to the Rauch-Tung-Striebel Kalman smoother[227]. This is called a 'correction' method since it takes the filtered estimate $p(\mathbf{h}_t|\mathbf{v}_{1:t})$ and 'corrects' it to form a smoothed estimate $p(\mathbf{h}_t|\mathbf{v}_{1:T})$. The procedure is outlined in algorithm(21) and is detailed in `LDSbackwardUpdate.m`. Together, the forward and backward updates are presented in algorithm(22) in standard form. See also `LDSsmooth.m`.

The cross moment

An advantage of the dynamics reversal interpretation given above is that the cross moment (which is required for learning) is immediately obtained from

$$\langle \Delta \mathbf{h}_t \Delta \mathbf{h}_{t+1} | \mathbf{v}_{1:T} \rangle = \overleftarrow{\mathbf{A}}_t \mathbf{G}_{t+1} \Rightarrow \left\langle \mathbf{h}_t \mathbf{h}_{t+1}^{\mathsf{T}} | \mathbf{v}_{1:T} \right\rangle = \overleftarrow{\mathbf{A}}_t \mathbf{G}_{t+1} + \mathbf{g}_t \mathbf{g}_{t+1}^{\mathsf{T}} \tag{24.4.38}$$

In more standard Engineering notation [119], this is, see algorithm(22),

$$\left\langle \mathbf{h}_t \mathbf{h}_{t+1}^{\mathsf{T}} | \mathbf{v}_{1:T}) \right\rangle = \overleftarrow{\mathbf{A}}_t \mathbf{P}_{t+1}^T + \hat{\mathbf{h}}_t^T \left(\hat{\mathbf{h}}_{t+1}^T \right)^{\mathsf{T}} \tag{24.4.39}$$

This is simpler than expressions more commonly found in the classical engineering literature[119]. See `LDSsmooth.m`.

Algorithm 21 LDS Backward Pass. Compute the smoothed posteriors $p(\mathbf{h}_t|\mathbf{v}_{1:T})$. This requires the filtered results from algorithm(20).

$\mathbf{G}_T \leftarrow \mathbf{F}_T$, $\mathbf{g}_T \leftarrow \mathbf{f}_T$
for $t \leftarrow T-1, 1$ **do**
  $\{\mathbf{g}_t, \mathbf{G}_t\} = \text{LDSBACKWARD}(\mathbf{g}_{t+1}, \mathbf{G}_{t+1}, \mathbf{f}_t, \mathbf{F}_t; \theta)$
end for
function LDSBACKWARD$(\mathbf{g}, \mathbf{G}, \mathbf{f}, \mathbf{F}; \theta)$
  $\boldsymbol{\mu}_h \leftarrow \mathbf{A}\mathbf{f} + \bar{\mathbf{h}}$, $\quad \boldsymbol{\Sigma}_{h'h'} \leftarrow \mathbf{A}\mathbf{F}\mathbf{A}^\mathsf{T} + \boldsymbol{\Sigma}^h$, $\quad \boldsymbol{\Sigma}_{h'h} \leftarrow \mathbf{A}\mathbf{F}$ ▷ Statistics of $p(\mathbf{h}_t, \mathbf{h}_{t+1}|\mathbf{v}_{1:t})$
  $\overleftarrow{\boldsymbol{\Sigma}} \leftarrow \mathbf{F} - \boldsymbol{\Sigma}_{h'h}^\mathsf{T}\boldsymbol{\Sigma}_{h'h'}^{-1}\boldsymbol{\Sigma}_{h'h}$, $\quad \overleftarrow{\mathbf{A}} \leftarrow \boldsymbol{\Sigma}_{h'h}^\mathsf{T}\boldsymbol{\Sigma}_{h'h'}^{-1}$, $\quad \overleftarrow{\mathbf{m}} \leftarrow \mathbf{f} - \overleftarrow{\mathbf{A}}\boldsymbol{\mu}_h$ ▷ Dynamics Reversal $p(\mathbf{h}_t|\mathbf{h}_{t+1}, \mathbf{v}_{1:t})$
  $\mathbf{g}' \leftarrow \overleftarrow{\mathbf{A}}\mathbf{g} + \overleftarrow{\mathbf{m}}$, $\quad \mathbf{G}' \leftarrow \overleftarrow{\mathbf{A}}\mathbf{G}\overleftarrow{\mathbf{A}}^\mathsf{T} + \overleftarrow{\boldsymbol{\Sigma}}$ ▷ Backward propagation
  return $\mathbf{g}', \mathbf{G}'$
end function

24.4.3 The likelihood

We can compute the likelihood using the decomposition

$$p(\mathbf{v}_{1:T}) = \prod_{t=1}^{T} p(\mathbf{v}_t|\mathbf{v}_{1:t-1}) \tag{24.4.40}$$

in which each conditional $p(\mathbf{v}_t|\mathbf{v}_{1:t-1})$ is a Gaussian in $\mathbf{v}_t$. It is straightforward to show that the term $p(\mathbf{v}_t|\mathbf{v}_{1:t-1})$ has mean and covariance

$$\begin{array}{lll} \boldsymbol{\mu}_1 \equiv \mathbf{B}\mu & \boldsymbol{\Sigma}_1 \equiv \mathbf{B}\boldsymbol{\Sigma}\mathbf{B}^\mathsf{T} + \boldsymbol{\Sigma}_V & t = 1 \\ \boldsymbol{\mu}_t \equiv \mathbf{B}\mathbf{A}\mathbf{f}_{t-1} & \boldsymbol{\Sigma}_t \equiv \mathbf{B}\left(\mathbf{A}\mathbf{F}_{t-1}\mathbf{A}^\mathsf{T} + \boldsymbol{\Sigma}_H\right)\mathbf{B}^\mathsf{T} + \boldsymbol{\Sigma}_V & t > 1 \end{array} \tag{24.4.41}$$

The log likelihood is then given by

$$\log p(\mathbf{v}_{1:T}) = -\frac{1}{2}\sum_{t=1}^{T}\left[(\mathbf{v}_t - \boldsymbol{\mu}_t)^\mathsf{T}\boldsymbol{\Sigma}_t^{-1}(\mathbf{v}_t - \boldsymbol{\mu}_t) + \log\det(2\pi\boldsymbol{\Sigma}_t)\right] \tag{24.4.42}$$

See also `LDSsmooth.m`.

24.4.4 Most likely state

For Gaussians, there is no difference between the most probable joint posterior state

$$\underset{\mathbf{h}_{1:T}}{\text{argmax}}\ p(\mathbf{h}_{1:T}|\mathbf{v}_{1:T}) \tag{24.4.43}$$

and the most probable marginal states

$$h_t = \underset{\mathbf{h}_{tT}}{\text{argmax}}\ p(\mathbf{h}_t|\mathbf{v}_{1:T}), \qquad t = 1, \ldots, T \tag{24.4.44}$$

since the mode of a Gaussian is equal to its mean. Hence, the most likely hidden state sequence is equivalent to the smoothed mean sequence.

24.4.5 Time independence and Riccati equations

Both the filtered $\mathbf{F}_t$ and smoothed $\mathbf{G}_t$ covariance recursions are independent of the observations $\mathbf{v}_{1:T}$, depending only on the parameters of the model. This is a general characteristic of linear Gaussian systems. Typically the covariance recursions converge quickly to values that are reasonably constant throughout the dynamics, with only appreciable differences at the boundaries $t = 1$ and $t = T$. In practice one often drops the time-dependence of the covariances and approximates them with a single time-independent covariance. This approximation is usually quite accurate provided we are not close to the ends of the series, and dramatically reduces storage requirements. The converged filtered $\mathbf{F}$, satisfies the recursion

$$\mathbf{F} = \mathbf{A}\mathbf{F}\mathbf{A}^\mathsf{T} + \boldsymbol{\Sigma}_H - \left(\mathbf{A}\mathbf{F}\mathbf{A}^\mathsf{T} + \boldsymbol{\Sigma}_H\right)\mathbf{B}^\mathsf{T}\left(\mathbf{B}\left(\mathbf{A}\mathbf{F}\mathbf{A}^\mathsf{T} + \boldsymbol{\Sigma}_H\right)\mathbf{B}^\mathsf{T} + \boldsymbol{\Sigma}_V\right)^{-1}\mathbf{B}\left(\mathbf{A}\mathbf{F}\mathbf{A}^\mathsf{T} + \boldsymbol{\Sigma}_H\right) \tag{24.4.45}$$

Algorithm 22 LDS: Forward and Backward Recursive Updates (written in standard notation). The filtered posterior $p(\mathbf{h}_t|\mathbf{v}_{1:t})$ is returned with means $\hat{\mathbf{h}}_t^t$ and covariances $\mathbf{P}_t^t$. The smoothed posterior $p(\mathbf{h}_t|\mathbf{v}_{1:T})$ means and covariances are $\hat{\mathbf{h}}_t^T$ and $\mathbf{P}_t^T$.

procedure FORWARD
  $\hat{\mathbf{h}}_1^0 \leftarrow \boldsymbol{\mu}$ ▷ Prior has mean $\boldsymbol{\mu}$ and cov $\boldsymbol{\Sigma}$
  $\mathbf{K} \leftarrow \boldsymbol{\Sigma}\mathbf{B}^\mathsf{T}(\mathbf{B}\boldsymbol{\Sigma}\mathbf{B}^\mathsf{T} + \boldsymbol{\Sigma}_V)^{-1}$
  $\mathbf{P}_1^1 \leftarrow (\mathbf{I} - \mathbf{K}\mathbf{B})\boldsymbol{\Sigma}$
  $\hat{\mathbf{h}}_1^1 \leftarrow \hat{\mathbf{h}}_1^0 + \mathbf{K}(\mathbf{v}_t - \mathbf{B}\hat{\mathbf{h}}_1^0)$
  for $t \leftarrow 2, T$ **do**
    $\mathbf{P}_t^{t-1} \leftarrow \mathbf{A}\mathbf{P}_{t-1}^{t-1}\mathbf{A}^\mathsf{T} + \boldsymbol{\Sigma}_H$
    $\hat{\mathbf{h}}_t^{t-1} \leftarrow \mathbf{A}\hat{\mathbf{h}}_{t-1}^{t-1}$
    $\mathbf{K} \leftarrow \mathbf{P}_t^{t-1}\mathbf{B}^\mathsf{T}(\mathbf{B}\mathbf{P}_t^{t-1}\mathbf{B}^\mathsf{T} + \boldsymbol{\Sigma}_V)^{-1}$ ▷ Kalman Gain
    $\mathbf{P}_t^t \leftarrow (\mathbf{I} - \mathbf{K}\mathbf{B})\mathbf{P}_t^{t-1}$
    $\hat{\mathbf{h}}_t^t \leftarrow \hat{\mathbf{h}}_t^{t-1} + \mathbf{K}(\mathbf{v}_t - \mathbf{B}\hat{\mathbf{h}}_t^{t-1})$
  end for
end procedure
procedure BACKWARD
  for $t \leftarrow T-1, 1$ **do**
    $\overleftarrow{\mathbf{A}}_t \leftarrow \mathbf{P}_t^t\mathbf{A}^\mathsf{T}(\mathbf{P}_{t+1}^t)^{-1}$
    $\mathbf{P}_t^T \leftarrow \mathbf{P}_t^t + \overleftarrow{\mathbf{A}}_t(\mathbf{P}_{t+1}^\mathsf{T} - \mathbf{P}_{t+1}^t)\overleftarrow{\mathbf{A}}_t^\mathsf{T}$
    $\hat{\mathbf{h}}_t^T \leftarrow \hat{\mathbf{h}}_t^t + \overleftarrow{\mathbf{A}}_t(\hat{\mathbf{h}}_{t+1}^\mathsf{T} - \mathbf{A}\hat{\mathbf{h}}_t^t)$
  end for
end procedure

which can be related to a form of *algebraic Riccati equation*. A simple technique to solve these equations is by recursion, beginning with setting the covariance to that of $p(\mathbf{h}_1)$, $\mathbf{F} = \boldsymbol{\Sigma}$. With this, a new $\mathbf{F}$ is found using the right hand side of (**??**), and recursed.

Alternatively, using the Woodbury identity, the converged covariance satisfies

$$\mathbf{F} = \left(\left(\mathbf{A}\mathbf{F}\mathbf{A}^\mathsf{T} + \boldsymbol{\Sigma}_H\right)^{-1} + \mathbf{B}^\mathsf{T}\boldsymbol{\Sigma}_V^{-1}\mathbf{B}\right)^{-1} \tag{24.4.46}$$

although this form is less numerically convenient in forming an iterative solver for $\mathbf{F}$ since it requires two matrix inversions.

Example 103 (Newtonian Trajectory Analysis)**.** A toy rocket with unknown mass and initial velocity is launched in the air. In addition, the constant accelerations from the rocket's propulsion system are unknown. It is known is that Newton's laws apply and an instrument can measure the vertical height and horizontal distance of the rocket at each time $x(t), y(t)$ from the origin. Based on noisy measurements of $x(t)$ and $y(t)$, our task is to infer the position of the rocket at each time.

Although this is perhaps most appropriately considered from the using continuous time dynamics, we will translate this into a discrete time approximation. Newton's law states that

$$\frac{d^2}{dt^2}x = \frac{f_x(t)}{m}, \qquad \frac{d^2}{dt^2}y = \frac{f_y(t)}{m} \tag{24.4.47}$$

where m is the mass of the object, and $f_x(t), f_y(t)$ are the horizontal and vertical forces respectively. Hence As they stand, these equations are not in a form directly usable in the LDS framework. There are several ways to rewrite them to make them suitable; a simple approach is to reparameterise time to use the variable $\tilde{t}$ such that $t \equiv \tilde{t}\Delta$, where $\tilde{t}$ is integer and Δ is a unit of time. The dynamics is then

$$x((\tilde{t}+1)\Delta) = x(\tilde{t}\Delta) + \Delta x'(\tilde{t}\Delta) \tag{24.4.48}$$

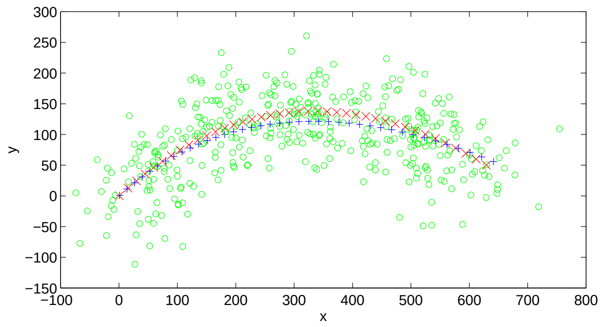


Figure 24.4: Estimate of the trajectory of a Newtonian ballistic object based on noisy observations (small circles). All time labels are known but omitted in the plot. The 'x' points are the true positions of the object, and the crosses '+' are the estimated smoothed mean positions $\langle x_t, y_t|\mathbf{v}_{1:T}\rangle$ of the object plotted every several time steps. See `demoLDStracking.m`

$$y((\tilde{t}+1)\Delta) = y(\tilde{t}\Delta) + \Delta y'(\tilde{t}\Delta) \tag{24.4.49}$$

where $y'(t) \equiv \frac{dy}{dt}$. We can write an update equation for the x' and y' as

$$x'((\tilde{t}+1)\Delta) = x'(\tilde{t}\Delta) + f_x\Delta/m, \qquad y'((\tilde{t}+1)\Delta) = y'(\tilde{t}\Delta) + f_y\Delta/m \tag{24.4.50}$$

These equations are then discrete time difference equations indexed by $\tilde{t}$. The instrument which measures $x(t)$ and $y(t)$ is not completely accurate. What is actually measured is $\hat{x}(t)$ and $\hat{y}(t)$, which are noisy versions of $x(t)$ and $y(t)$. For simplicity, we relabel $a_x(t) = f_x(t)/m(t)$, $a_y(t) = f_y(t)/m(t)$ – these accelerations will be assumed to be roughly constant, but unknown :

$$a_x((\tilde{t}+1)\Delta) = a_x(\tilde{t}\Delta) + \eta_x, \quad a_y((\tilde{t}+1)\Delta) = a_y(\tilde{t}\Delta) + \eta_y, \tag{24.4.51}$$

where η_x and η_y are small noise terms. The initial distributions for the accelerations are vague, using a zero mean Gaussian with large variance.

One way to describe the above approach is to consider $x(t)$, $y(t)$, $x'(t)$, $y'(t)$, $a_x(t)$ and $a_y(t)$ as hidden variables, giving rise to a $H = 6$ dimensional LDS, see `demoLDStracking.m`, with transition and emission matrices as below:

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 0 & 0 & \Delta & 0 \\ \Delta & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta \\ 0 & 0 & \Delta & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \qquad \mathbf{B} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \tag{24.4.52}$$

We place a large variance prior on their initial values, and attempt to infer the unknown trajectory. A demonstration for this is given in fig(24.4.5). The object trajectory can be accurately inferred despite the large amount of measurement noise.

24.5 Learning Linear Dynamical Systems

Whilst in many applications, particularly of underlying known physical processes, the paremeters of the LDS are known, in many machine learning tasks we need to learn the parameters of the LDS based on $\mathbf{v}_{1:T}$. For example in bioinformatics, it is sometimes assumed that the observed gene expression through time is driven by an unknown latent linear dynamical system[27]. For simplicity we assume that we know the dimensionality H of the LDS.

24.5.1 Identifiability issues

An important question is whether we can uniquely identify (learn) the parameters of an LDS. There are always trivial redundancies in the solution, obtained by permuting the hidden variables arbitrarily and flipping their signs. To show that there are potentially many more equivalent solutions, consider the following LDS

$$\mathbf{v}_t = \mathbf{B}\mathbf{h}_t + \boldsymbol{\eta}_t^v, \qquad \mathbf{h}_t = \mathbf{A}\mathbf{h}_{t-1} + \boldsymbol{\eta}_t^h \tag{24.5.1}$$

We now attempt to transform this original system to a new form which will produce exactly the same outputs $\mathbf{v}_{1:T}$. For an invertible matrix $\mathbf{R}$ we consider

$$\mathbf{R}\mathbf{h}_t = \mathbf{R}\mathbf{A}\mathbf{R}^{-1}\mathbf{R}\mathbf{h}_{t-1} + \mathbf{R}\boldsymbol{\eta}_t^h \tag{24.5.2}$$

which is representable as a new latent dynamics

$$\hat{\mathbf{h}}_t = \hat{\mathbf{A}}\hat{\mathbf{h}}_{t-1} + \hat{\boldsymbol{\eta}}_t^h \tag{24.5.3}$$

where $\hat{\mathbf{A}} \equiv \mathbf{R}\mathbf{A}\mathbf{R}^{-1}$, $\hat{\mathbf{h_t}} \equiv \mathbf{R}\mathbf{h}_t$, $\hat{\boldsymbol{\eta}}_t^h \equiv \mathbf{R}\boldsymbol{\eta}_t^h$. In addition, we can reexpress the outputs to be a function of the transformed $\mathbf{h}$:

$$\mathbf{v}_t = \mathbf{B}\mathbf{R}^{-1}\mathbf{R}\mathbf{h}_t + \boldsymbol{\eta}_t^v = \hat{\mathbf{B}}\hat{\mathbf{h}}_t + \boldsymbol{\eta}_t^v \tag{24.5.4}$$

Hence, provided we place no constraints on $\mathbf{A}$, $\mathbf{B}$ and $\boldsymbol{\Sigma}_H$ there exists an infinite space of solutions, $\hat{\mathbf{A}} = \mathbf{R}\mathbf{A}\,\mathbf{R}^{-1}$, $\hat{\mathbf{B}} = \mathbf{R}\mathbf{B}$, $\hat{\boldsymbol{\Sigma}}_H = \mathbf{R}\boldsymbol{\Sigma}_H\mathbf{R}^{\mathsf{T}}$, all with the same Maximum Likelihood value. This redundancy is broken under the constraint that, for example, $\boldsymbol{\Sigma}_H$ is diagonal.

24.5.2 EM algorithm

For simplicity, we assume we have a single sequence $\mathbf{v}_{1:T}$, for which we wish to fit a LDS using Maximum Likelihood. Since the LDS contains latent variables one approach is to use the EM algorithm. As usual, the M-step of the EM algorithm requires us to maximise the energy

$$\left\langle \log p(\mathbf{v}_{1:T}, \mathbf{h}_{1:T}) \right\rangle_{p^{old}(\mathbf{h}_{1:T}|\mathbf{v}_{1:T})} \tag{24.5.5}$$

with respect to the parameters $\mathbf{A}$, $\mathbf{B}$, $\mathbf{a}$, $\boldsymbol{\Sigma}$, $\boldsymbol{\Sigma}_V$, $\boldsymbol{\Sigma}_H$. Thanks to the form of the LDS the energy decomposes as

$$\left\langle \log p(\mathbf{h}_1) \right\rangle_{p^{old}(\mathbf{h}_1|\mathbf{v}_{1:T})} + \sum_{t=2}^{T} \left\langle \log p(\mathbf{h}_t|\mathbf{h}_{t-1}) \right\rangle_{p^{old}(\mathbf{h}_t,\mathbf{h}_{t-1}|\mathbf{v}_{1:T})} + \sum_{t=1}^{T} \left\langle \log p(\mathbf{v}_t|\mathbf{h}_t) \right\rangle_{p^{old}(\mathbf{h}_t|\mathbf{v}_{1:T})} \tag{24.5.6}$$

It is straightforward to derive that the M-step for the parameters is given by:

$$\boldsymbol{\Sigma}_V^{new} = \frac{1}{T}\sum_t \left(\mathbf{v}_t\mathbf{v}_t^{\mathsf{T}} - \mathbf{v}_t \left\langle \mathbf{h}_t \right\rangle^{\mathsf{T}} \mathbf{B}^{\mathsf{T}} - \mathbf{B}\left\langle \mathbf{h}_t \right\rangle \mathbf{v}_t^{\mathsf{T}} + \mathbf{B}\left\langle \mathbf{h}_t\mathbf{h}_t^{\mathsf{T}} \right\rangle \mathbf{B}^{\mathsf{T}} \right) \tag{24.5.7}$$

$$\boldsymbol{\Sigma}_H^{new} = \frac{1}{T-1}\sum_t \left(\left\langle \mathbf{h}_{t+1}\mathbf{h}_{t+1}^{\mathsf{T}} \right\rangle - \mathbf{A}\left\langle \mathbf{h}_t\mathbf{h}_{t+1}^{\mathsf{T}} \right\rangle - \left\langle \mathbf{h}_{t+1}\mathbf{h}_t^{\mathsf{T}} \right\rangle \mathbf{A}^{\mathsf{T}} + \mathbf{A}\left\langle \mathbf{h}_t\mathbf{h}_t^{\mathsf{T}} \right\rangle \mathbf{A}^{\mathsf{T}} \right) \tag{24.5.8}$$

$$\boldsymbol{\mu}^{new} = \left\langle \mathbf{h}_1 \right\rangle \tag{24.5.9}$$

$$\boldsymbol{\Sigma}^{new} = \left\langle \mathbf{h}_1\mathbf{h}_1^{\mathsf{T}} \right\rangle - \mu\mu^{\mathsf{T}} \tag{24.5.10}$$

$$\mathbf{A}^{new} = \sum_{t=1}^{T-1} \left\langle \mathbf{h}_{t+1}\mathbf{h}_t^{\mathsf{T}} \right\rangle \left(\sum_{t=1}^{T-1} \left\langle \mathbf{h}_t\mathbf{h}_t^{\mathsf{T}} \right\rangle \right)^{-1} \tag{24.5.11}$$

$$\mathbf{B}^{new} = \sum_{t=1}^{T} \mathbf{v}_t \left\langle \mathbf{h}_t \right\rangle^{\mathsf{T}} \left(\sum_{t=1}^{T} \left\langle \mathbf{h}_t\mathbf{h}_t^{\mathsf{T}} \right\rangle \right)^{-1} \tag{24.5.12}$$

If $\mathbf{B}$ is updated according to the above, the first equation can be simplified to

$$\boldsymbol{\Sigma}_V^{new} = \frac{1}{T}\sum_t \left(\mathbf{v}_t\mathbf{v}_t^{\mathsf{T}} - \mathbf{v}_t \left\langle \mathbf{h}_t \right\rangle^{\mathsf{T}} \mathbf{B}^{\mathsf{T}} \right) \tag{24.5.13}$$

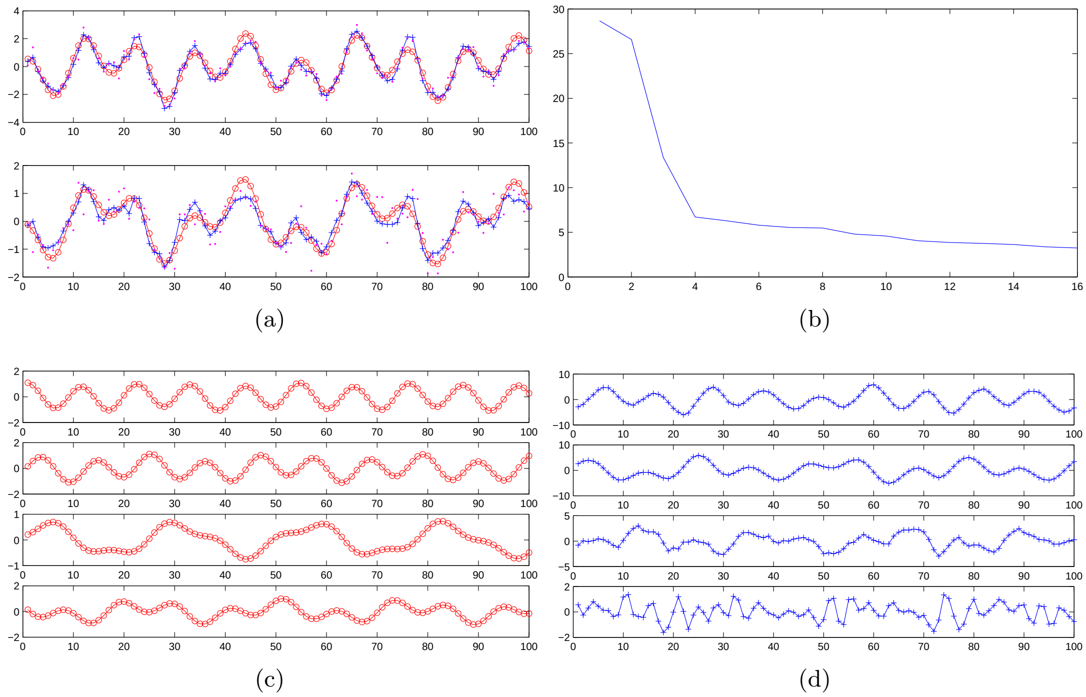


Figure 24.5: Learning the parameters of a noisy LDS with $V = 2$. $H = 4$. **(a)**: The small points are the visible data, circles the noise free visible observations, and the crosses the estimates of the noise free observations based on using the subspace method. **(b)**: Singular values. In a noise free case, only $H = 4$ singular values will be non-zero. The noise in this problem means that additional singular values appear. **(c)**: Hidden values used to generate the data. **(d)**: Hidden values estimated by the subspace method, formed from the principal 4-dimensional subspace.

Similarly, if $\mathbf{A}$ is updated according to EM algorithm, then the second equation can be simplified to

$$\mathbf{\Sigma}_H^{new} = \frac{1}{T-1}\sum_t \left(\left\langle \mathbf{h}_{t+1}\mathbf{h}_{t+1}^\mathsf{T}\right\rangle - \mathbf{A}\left\langle \mathbf{h}_t\mathbf{h}_{t+1}^\mathsf{T}\right\rangle\right) \tag{24.5.14}$$

In the above updates, the angled brackets $\langle\cdot\rangle$ denote expectation with respect to the smoothed posterior $p(\mathbf{h}_{1:T}|\mathbf{v}_{1:T})$. The statistics required therefore include smoothed means, covariances, and cross moments, all of which are available after filtering and smoothing. The extension to learning multiple timeseries is straightforward since the energy is simply summed over the data sequences.

The performance of the EM algorithm for the LDS often depends heavily on a suitable initialisation. If we remove the hidden to hidden links, the model is closely related to Factor Analysis (the LDS can be considered a temporal extension of Factor Analysis). One initialisation technique is therefore to learn the $\mathbf{B}$ matrix using Factor Analysis by treating the visible observations as temporally independent.

24.5.3 Subspace Methods

An alternative to EM and Maximum Likelihood training of an LDS is to use a subspace method[278, 246]. The chief benefit of these techniques is that they avoid the convergence difficulties of EM.

To motivate subspace techniques, consider a deterministic LDS

$$\mathbf{h}_t = \mathbf{A}\mathbf{h}_{t-1}, \qquad \mathbf{v}_t = \mathbf{B}\mathbf{h}_t \tag{24.5.15}$$

Under this assumption, $\mathbf{v}_t = \mathbf{B}\mathbf{h}_t = \mathbf{B}\mathbf{A}\mathbf{h}_{t-1}$ and, more generally, $\mathbf{v}_t = \mathbf{B}\mathbf{A}^t\mathbf{h}_1$. This means that a low dimensional system underlies all visible information since all points $\mathbf{A}^t\mathbf{h}_1$ lie in a H-dimensional subspace, which is then projected to form the visible variable. This suggests that some form of subspace identification technique will enable us to learn $\mathbf{A}$ and $\mathbf{B}$.

Given a set of observation vectors $\mathbf{v}_1, \ldots, \mathbf{v}_t$, consider the block *Hankel matrix* formed from stacking the vectors. For an order L matrix, this is a $VL \times T-L+1$ matrix. For example, for $T = 6$ and $L = 3$, this is

$$\mathbf{M} = \begin{pmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \mathbf{v}_3 & \mathbf{v}_4 \\ \mathbf{v}_2 & \mathbf{v}_3 & \mathbf{v}_4 & \mathbf{v}_5 \\ \mathbf{v}_3 & \mathbf{v}_4 & \mathbf{v}_5 & \mathbf{v}_6 \end{pmatrix} \tag{24.5.16}$$

If the $\mathbf{v}$ are generated from a (noise free) LDS, we can write

$$\mathbf{M} = \begin{pmatrix} \mathbf{B}\mathbf{h}_1 & \mathbf{B}\mathbf{h}_2 & \mathbf{B}\mathbf{h}_3 & \mathbf{B}\mathbf{h}_4 \\ \mathbf{B}\mathbf{A}\mathbf{h}_1 & \mathbf{B}\mathbf{A}\mathbf{h}_2 & \mathbf{B}\mathbf{A}\mathbf{h}_3 & \mathbf{B}\mathbf{A}\mathbf{h}_4 \\ \mathbf{B}\mathbf{A}^2\mathbf{h}_1 & \mathbf{B}\mathbf{A}^2\mathbf{h}_2 & \mathbf{B}\mathbf{A}^2\mathbf{h}_3 & \mathbf{B}\mathbf{A}^2\mathbf{h}_4 \end{pmatrix} = \begin{pmatrix} \mathbf{B} \\ \mathbf{B}\mathbf{A} \\ \mathbf{B}\mathbf{A}^2 \end{pmatrix} \begin{pmatrix} \mathbf{h}_1 & \mathbf{h}_2 & \mathbf{h}_3 & \mathbf{h}_4 \end{pmatrix} \tag{24.5.17}$$

We now find the SVD of $\mathbf{M}$,

$$\mathbf{M} = \hat{\mathbf{U}} \underbrace{\hat{\mathbf{S}}\hat{\mathbf{V}}^\mathsf{T}}_{\mathbf{W}} \tag{24.5.18}$$

where $\mathbf{W}$ is termed the *extended observability matrix*. The matrix $\hat{\mathbf{S}}$ will contain the singular values up to the dimension of the hidden variables H, with the remaining singular values 0. This means that the emission matrix $\mathbf{B}$ is contained in $\hat{\mathbf{U}}_{1:V,1:H}$.

The estimated hidden variables are then contained in the submatrix $\mathbf{W}_{1:H,1:T-L+1}$,

$$[\mathbf{h}_1 \quad \mathbf{h}_2 \quad \mathbf{h}_3 \quad \mathbf{h}_4] = \mathbf{W}_{1:H,1:T-L+1} \tag{24.5.19}$$

Based on the relation $\mathbf{h}_t = \mathbf{A}\mathbf{h}_{t-1}$ one can then find the best least squares estimate for $\mathbf{A}$ by minimising

$$\sum_{t=2}^{T} (\mathbf{h}_t - \mathbf{A}\mathbf{h}_{t-1})^2 \tag{24.5.20}$$

for which the optimal solution is

$$\mathbf{A} = [\mathbf{h}_2 \quad \mathbf{h}_3 \quad \ldots \quad \mathbf{h}_t] \, [\mathbf{h}_1 \quad \mathbf{h}_2 \quad \ldots \quad \mathbf{h}_{T-1}]^\dagger \tag{24.5.21}$$

where † denotes the pseudo inverse, see `LDSsubspace.m`. Estimates for the covariance matrices can also be obtained from the residual errors in fitting the block Hankel matrix ($\mathbf{\Sigma}_V$) and extended observability matrix ($\mathbf{\Sigma}_H$).

Whilst the above derivation holds for the noise free case one can apply this still in the case of non-zero noise and hope to gain an estimate for $\mathbf{A}$ and $\mathbf{B}$ that is correct in the mean. In addition to forming a solution in its own right, the subspace method forms a potentially useful way to initialise the EM algorithm.

Example 104 (Subspace fitting)**.** In fig(24.5) we present learning the parameters of an LDS using the subspace method. Whilst the visible variables are well modelled by this system, the model corresponding to the one that generated the data has not been correctly identified.

24.5.4 Structured LDSs

Many physical equations are local both in time (Markov) and space (a variable depends only on its neighbours). For example in weather models the atmosphere is partitioned into cells $h_i(t)$ each containing the pressure at that location. The equations describing how the pressure updates only depend on the pressure at the current and small number of neighbouring cells at the previous time $t-1$. If we use a linear model, and measure some aspects of the cells at each time, then the weather is describable by a LDS with a highly structured sparse transition matrix $\mathbf{A}$. In practice, the weather models are non-linear but local

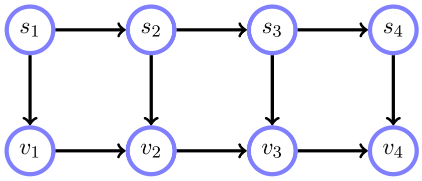


Figure 24.6: A first order switching AR model. In terms of inference, conditioned on $v_{1:T}$, this is a HMM.

linear approximations are often employed[251]. A similar situation arises in brain imaging in which voxels (local cubes of activity) depend only on their neighbours from the previous timestep[98].

Another application of structured LDSs is in temporal independent component analysis. This is defined as the discovery of a set of independent latent dynamical processes, from which the data is a projected observation. If each independent dynamical process can itself be described by a LDS, this gives rise to a structured LDS with a block diagonal transition matrix $\mathbf{A}$. Such models can be used to extract independent components under prior knowledge of the likely underlying frequencies in each of the temporal compoments[58]. See also exercise(198).

24.5.5 Bayesian LDSs

The extension to placing priors on the transition and emission parameters of the LDS leads in general to computational difficulties in computing the likelihood $p(\mathbf{v}_{1:T}) = \int_{\mathbf{A}} p(\mathbf{v}_{1:T}|\mathbf{A})p(\mathbf{A})$ since the dependency of the likelihood on the matrix $\mathbf{A}$ is a complicated function. A treatment of this case is beyond the scope of this book, although we briefly note that sampling methods[53, 95] are popular in this context. For the special case of Variational Bayes under Gaussian priors on the transition and emission matrices, deterministic approximations have been developed, and one can show that the variational inference equations are related directly to smoothing in a modified LDS[27, 23, 58].

24.6 Switching auto-regressive models

A natural extension of the AR model is to a switching AR model in which there are a set of transition matrices $\hat{\mathbf{A}}_h, h = 1, \ldots, H$. This means that one can use a different dynamic matrix $\hat{\mathbf{A}}_h$ at different times, with the switching between these dynamics modelled by the transition distribution $p(\mathbf{h}_t|\mathbf{h}_{t-1})$, see fig(24.6). To prevent too frequent switching, constraints are placed on the transition $p(\mathbf{h}_t|\mathbf{h}_{t-1})$. For a time-series of scalar values $v_{1:T}$ an L^{th} order switching AR model can be written as

$$v_t = \mathbf{a}(s_t)^\mathsf{T}\hat{\mathbf{v}}_{t-1} + \eta_t, \qquad \eta_t \sim \mathcal{N}\left(\eta_t|0, \sigma^2(s_t)\right) \tag{24.6.1}$$

where we now have a set of AR coefficients $\mathbf{a}(s_t)$, $s_t \in \{1, \ldots, S\}$. The discrete switch variables themselves have a Markov transition

$$p(s_{1:T}) = \prod_t p(s_t|s_{t-1}) \tag{24.6.2}$$

so that the full model is

$$p(v_{1:T}, s_{1:T}) = \prod_t p(v_t|v_{t-1}, \ldots, v_{t-L}, s_t)p(s_t|s_{t-1}) \tag{24.6.3}$$

Inference

Given an observed sequence $v_{1:T}$ and the set of AR coefficients $\mathbf{a}(s)$, $s = 1, \ldots, S$, inference is straightforward since this is a form of HMM. To make this more apparent we may write

$$p(v_{1:T}, s_{1:T}) = \prod_t \hat{p}(v_t|s_t)p(s_t|s_{t-1}), \qquad \text{where} \quad \hat{p}(v_t|s_t) = p(v_t|v_{t-1}, \ldots, v_{t-L}, s_t) \tag{24.6.4}$$

The emission distribution $\hat{p}(\mathbf{v}_t|s_t)$ corresponds to a time-dependent emission matrix, $\mathbf{B}_t \equiv \hat{\mathbf{v}}_{t-1}^\mathsf{T}$. The filtering recursion is then

$$\alpha(s_t) = \sum_{s_{t-1}} \hat{p}(v_t|s_t)p(s_t|s_{t-1})\alpha(s_{t-1}) \tag{24.6.5}$$

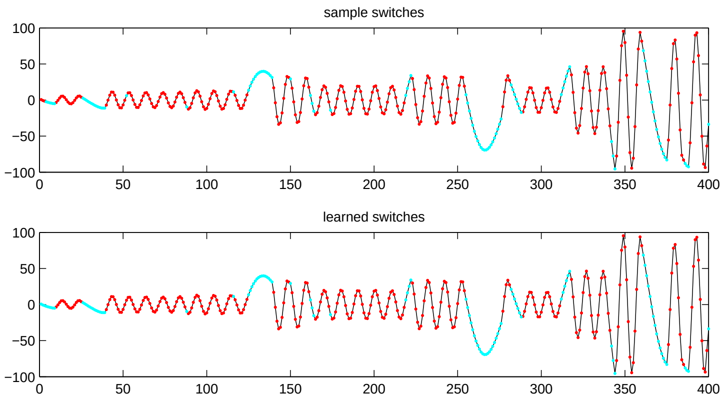


Figure 24.7: The upper plot shows the training data. The colour indicates which of the two AR models is active at that time. Whilst this information is plotted here, this is actually unknown to the learning algorithm, as are the coefficients $\mathbf{a}(s)$. We assume that the order $L = 2$ and number of switches $S = 2$ however is known. In the bottom plot we show the time series again coloured after training, and using then the most likely model at each timestep. See `demoSARlearn.m`.

Smoothing can be achieved using either with a backward β recursion, or using the γ recursion. In `demoSARinference.m` we use the β recursion and also present the factor graph setup for comparison.

Maximum Likelihood Learning using EM

To fit the set of AR coefficients and innovation variances, $\mathbf{a}(s), \sigma^2(s), s = 1, \ldots, S$, using Maximum Likelihood training for a set of data $v_{1:T}$, we may make use of the EM algorithm. Up to negligible constants, the energy is given by

$$E = \sum_t \langle \log p(v_t|\hat{\mathbf{v}}_{t-1}, \mathbf{a}(s_t)) \rangle_{p^{old}(s_t|v_{1:T})} + \sum_t \langle \log p(s_t|s_{t-1}) \rangle_{p^{old}(s_t, s_{t-1})} \tag{24.6.6}$$

which we need to maximise with respect to the parameters $\mathbf{a}$, σ^2. Using the definition of the emission and isolating the dependency on $\mathbf{a}$, we have

$$-2E = \sum_t \left\langle \frac{1}{\sigma^2(s_t)} \left(v_t - \hat{\mathbf{v}}_{t-1}^{\mathsf{T}} \mathbf{a}(s_t) \right)^2 + \log \sigma^2(s_t) \right\rangle_{p^{old}(s_t|v_{1:T})} + \text{const.} \tag{24.6.7}$$

On differentiating with respect to $\mathbf{a}(\mathsf{s})$ and equating to zero, the update rule for $\mathbf{a}(\mathsf{s})$ requires the solution of the linear equation

$$\sum_t p^{old}(s_t = \mathsf{s}|v_{1:T}) \frac{v_t \hat{\mathbf{v}}_{t-1}}{\sigma^2(\mathsf{s})} = \left[\sum_t p^{old}(s_t = \mathsf{s}|v_{1:T}) \frac{\hat{\mathbf{v}}_{t-1} \hat{\mathbf{v}}_{t-1}^{\mathsf{T}}}{\sigma^2(\mathsf{s})} \right] \mathbf{a}(\mathsf{s}) \tag{24.6.8}$$

Similarly one may show that updates that maximise the energy with respect to σ^2 are

$$\sigma^2(\mathsf{s}) = \frac{1}{\sum_{t'} p^{old}(s'_t = \mathsf{s}|v_{1:T})} \sum_t p^{old}(s_t = \mathsf{s}|v_{1:T}) \left[v_t - \hat{\mathbf{v}}_{t-1}^{\mathsf{T}} \mathbf{a}(s_t) \right]^2 \tag{24.6.9}$$

The update for $p(s_t|s_{t-1})$ follows the standard EM for HMM rule, equation (23.3.5), see `SARlearn.m`. Here we don't include an update for the prior $p(s_1)$ since there is insufficient information at the start of the sequence and assume $p(s_1)$ is flat.

In practice with high frequency data it is unlikely that a change in the switch variable is reasonable at each time t. A simple constraint there is to use a modified transition

$$\hat{p}(s_t|s_{t-1}) = \begin{cases} p(s_t|s_{t-1}) & \text{mod}\,(t, T_{skip}) = 0 \\ \delta\,(s_t - s_{t-1}) & \text{otherwise} \end{cases} \tag{24.6.10}$$

Example 105 (Learning a switching AR model)**.** In fig(24.7) we show training data and a corresponding fit using a SAR model. In this case the data was actually generated by an SAR model so that we know the

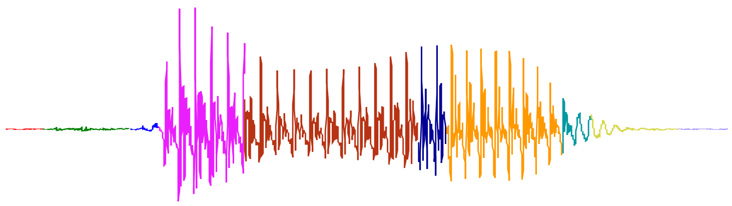


Figure 24.8: A spoken digit of the word 'four' modelled by a Switching Autoregressive model. The SAR was trained on many example sequences using $S = 10$ states with a left-to-right transition matrix. The colours indicate the states used at each time. The states found correspond to basic sound component models that when used in sequence generate realistic sounding waveforms.

ground truth as to which model generated which parts of the data. Based on the training data (assuming the labels s_t are unknown), a SAR model is fitted using EM. In this case the problem is straightforward so that a good estimate is obtained of both the sets of AR parameters and which switches were used at which time.

Example 106 (Modelling parts of speech)**.** An example application from a speech recognition problem is presented in fig(24.8) in which the AR models each correspond to basic parts of speech[190, 189].

24.7 Code

In the Linear Dynamical System code below only the simplest form of the recursions is given. No attempt has been made to ensure numerical stability.
`LDSforwardUpdate.m`: LDS forward
`LDSbackwardUpdate.m`: LDS backward
`LDSsmooth.m`: Linear Dynamical System : filtering and smoothing
`LDSforward.m`: Alternative LDS forward algorithm (see SLDS chapter)
`LDSbackward.m`: Alternative LDS backward algorithm (see SLDS chapter)
`demoSumprodGaussCanonLDS.m`: Sum-product algorithm for smoothed inference

`demoLDStracking.m`: Demo of tracking in a Newtonian system

`LDSsubspace.m`: Subspace Learning (Hankel matrix method)
`demoLDSsubspace.m`: Demo of Subspace Learning method

24.7.1 Autoregressive models

Note that in the code the autoregressive vector $\mathbf{a}$ has as its last entry the first AR coefficient (*i.e.* in reverse order to that presented in the text).
`ARtrain.m`: Learn AR coefficients (Gaussian Elimination)
`demoARtrain.m`: Demo of fitting an AR model to data
`ARlds.m`: Learn AR coefficients using a LDS
`demoARlds.m`: Demo of learning AR coefficients using an LDS
`demoSARinference.m`: Demo for inference in a Switching Autoregressive Model

In `SARlearn.m` a slight fudge is used since we do not deal fully with the case at the start where there is insufficient information to define the AR model. For long timeseries this will have a negligible effect, although it might lead to small decreases in the log likelihood under the EM algorithm.

`SARlearn.m`: Learning of a SAR using EM
`demoSARlearn.m`: Demo of SAR learning
`HMMforwardSAR.m`: Switching Autoregressive HMM forward pass
`HMMbackwardSAR.m`: Switching Autoregressive HMM backward pass

24.8 Exercises

Exercise 234. *Consider the two-dimension linear model*

$$\mathbf{h}_t = \mathbf{R}_\theta \mathbf{h}_{t-1} \tag{24.8.1}$$

where

$$\mathbf{R}_\theta = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \tag{24.8.2}$$

R_θ is rotation matrix which rotates the vector $\mathbf{h}_t$ through angle θ in one timestep.

1. *By writing*

$$\begin{pmatrix} x_t \\ y_t \end{pmatrix} = \begin{pmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{pmatrix} \begin{pmatrix} x_{t-1} \\ y_{t-1} \end{pmatrix} \tag{24.8.3}$$

eliminate y_t to write an equation for x_{t+1} in terms of x_t and x_{t-1}.

2. *Explain why the eigenvalues of a rotation matrix are (in general) imaginary.*

Exercise 235.

1. *Explain how to model a sinusoid, rotating with angular velocity ω using a two-dimensional LDS.*
2. *Explain how to model a sinusoid using an AR model.*
3. *Explain the relationship between the second order differential equation $\ddot{x} = -\lambda x$, which describes a Harmonic Oscillator, and the second order difference equation which approximates this differential equation. Is it possible to find a difference equation which exactly matches the solution of the differential equation at chosen points?*

Exercise 236. *Show that for any anti-symmetric matrix $\mathbf{M}$,*

$$\mathbf{M} = -\mathbf{M}^{\mathsf{T}} \tag{24.8.4}$$

the matrix exponential (in MATLAB *this is* `expm`*)*

$$\mathbf{A} = e^{\mathbf{M}} \tag{24.8.5}$$

is orthogonal, namely

$$\mathbf{A}^{\mathsf{T}}\mathbf{A} = \mathbf{I} \tag{24.8.6}$$

Explain how one may then construct random orthogonal matrices with some control over the angles of the complex eigenvalues. Discuss how this relates to the frequencies encountered in a LDS where $\mathbf{A}$ is the transition matrix.

Exercise 237. *Run the demo `demoLDStracking.m` which tracks a ballistic object using a Linear Dynamical system, see example(103). Modify `demoLDStracking.m` so that in addition to the x and y positions, the x speed is also observed. Compare and contrast the accuracy of the tracking with and without this extra information.*

Exercise 238. *Use* `load nightsong` *to load in a small segment of sound from a nightingale sampled at* 44100 *Hertz. If you have a windows machine you can play this using* `wavplay(x,44110)`. *Note that* `x` *contains two channels since this is a stereo recording.*

1. *Plot the original waveform using* `plot(x(:,1))`
2. *Download the program* `myspecgram.m` *from* `http://labrosa.ee.columbia.edu/matlab/sgram/myspecgram.m` *and plot the spectrogram (the short time windowed discrete Fourier transform) using*

   `y=myspecgram(x,1024,44100). imagesc(log(abs(y)))`
3. *The routine* `demoGMMem.m` *demonstrates fitting a mixture of Gaussians to data. The mixture assignment probabilities are contained in* `phgn`. *Write a routine to cluster the data* `v=log(abs(y))` *using 8 Gaussian components, and explain how one might segment the series* `x` *into different regions.*
4. *Examine the routine* `demoARlds.m` *which fits autoregressive coefficients using an interpretation as a Linear Dynamical System. Adapt the routine* `demoARlds.m` *to learn the AR coefficients of the data* `x`. *You will almost certainly need to subsample the data* `x` *– for example by taking every* 4^{th} *datapoint. With the learned AR coefficients (use the smoothed results) fit a Gaussian mixture with 8 components. Compare and contrast your results with those obtained from the Gaussian mixture model fit to the Spectrogram.*

Exercise 239. *Consider a supervised learning problem in which we make a linear model of the scaler output* y_t *based on vector input* $\mathbf{x}_t$*:*

$$y_t = \mathbf{w}_t^T \mathbf{x}_t + \boldsymbol{\eta}_t^y \tag{24.8.7}$$

where $\boldsymbol{\eta}_t^y$ *is zero mean Gaussian noise. Training data* $\mathcal{D} = \{(\mathbf{x}_t, y_t), t = 1, \ldots, T\}$ *is available.*

1. *For a stationary weight vector* $\mathbf{w}_t \equiv \mathbf{w}$, *explain how to find the single weight vector* $\mathbf{w}$ *and the noise variance* σ^2 *by Maximum Likelihood.*
2. *Extend the above model to include a transition*

   $$\mathbf{w}_t = \mathbf{w}_{t-1} + \boldsymbol{\eta}_t^w \tag{24.8.8}$$

   where $\boldsymbol{\eta}_t^w$ *is zero mean Gaussian noise with a given covariance* $\boldsymbol{\Sigma}$ *and for* $\mathbf{w}_1$ *having zero mean. Explain how to cast finding* $\langle \mathbf{w}_t | \mathcal{D} \rangle$ *as smoothing in a related Linear Dynamical System. Write a routine* `W = LinPredAR(X,Y,SigmaW,SigmaY)` *that takes an input data matrix* $\mathbf{X}$ *where each column contains an input, and vector* Y*;* `SigmaW` *is the additive weight noise and* `SigmaY` *is an assumed known stationary output noise. The returned* `W` *contains the smoothed mean weights.*

CHAPTER 25

Switching Linear Dynamical Systems

25.1 Introduction

The Linear Dynamical System (LDS), chapter(24) is a standard time-series model in which a latent linear process generates the observed time-series. Complex time-series which are not well described globally by a single LDS, may be divided into segments, each modelled by a potentially different LDS. Such models can handle situations in which the underlying model 'jumps' from one parameter setting to another.

For example a single LDS might well represent the normal flows in a chemical plant. However, if there is a break in a pipeline, the dynamics of the system changes from one set of linear flow equations to another. This scenario could be modelled by two sets of linear systems, each with different parameters, with a discrete latent variable at each time $s_t \in \{\mathsf{normal}, \mathsf{pipe\ broken}\}$, indicating which of the LDSs is most appropriate at the current time. This is called a Switching LDS and used in many disciplines, from econometrics to Machine Learning [13, 107, 171, 158, 157, 59, 56, 215, 299, 172].

25.2 The Switching LDS

At each time t, a switch variable $s_t \in 1, \ldots, S$ describes which of a set of LDSs is to be used. The observation (or 'visible') variable $\mathbf{v}_t \in \mathcal{R}^V$ is linearly related to the hidden state $\mathbf{h}_t \in \mathcal{R}^H$ by

$$\mathbf{v}_t = \mathbf{B}(s_t)\mathbf{h}_t + \boldsymbol{\eta}^v(s_t), \qquad \boldsymbol{\eta}^v(s_t) \sim \mathcal{N}\left(\boldsymbol{\eta}^v(s_t)\middle|\bar{\mathbf{v}}(s_t), \boldsymbol{\Sigma}^v(s_t)\right) \tag{25.2.1}$$

Here s_t describes which of the set of emission matrices $\mathbf{B}(1), \ldots, \mathbf{B}(S)$ is active at time t. The observation noise $\boldsymbol{\eta}^v(s_t)$ is drawn from a one of a set of Gaussians with different means $\bar{\mathbf{v}}(s_t)$ and covariance $\boldsymbol{\Sigma}(s_t)$. The transition dynamics of the continuous hidden state $\mathbf{h}_t$ is linear,

$$\mathbf{h}_t = \mathbf{A}(s_t)\mathbf{h}_{t-1} + \boldsymbol{\eta}^h(s_t), \qquad \boldsymbol{\eta}^h(s_t) \sim \mathcal{N}\left(\boldsymbol{\eta}^h(s_t)\middle|\bar{\mathbf{h}}(s_t), \boldsymbol{\Sigma}^h(s_t)\right) \tag{25.2.2}$$

and the switch variable s_t selects a single transition matrix from the available set $\mathbf{A}(1), \ldots, \mathbf{A}(S)$. The Gaussian transition noise $\boldsymbol{\eta}^h(s_t)$ also depends on the switch variables. The dynamics of s_t itself is Markovian, with transition $p(s_t|s_{t-1})$. For the more general 'augmented' aSLDS model the switch s_t is dependent on both the previous s_{t-1} and $\mathbf{h}_{t-1}$.

The probabilistic model defines a joint distribution(see fig(25.1))

$$p(\mathbf{v}_{1:T}, \mathbf{h}_{1:T}, s_{1:T}) = \prod_{t=1}^{T} p(\mathbf{v}_t|\mathbf{h}_t, s_t)p(\mathbf{h}_t|\mathbf{h}_{t-1}, s_t)p(s_t|\mathbf{h}_{t-1}, s_{t-1})$$

with

$$p(\mathbf{v}_t|\mathbf{h}_t, s_t) = \mathcal{N}\left(\mathbf{v}_t\middle|\bar{\mathbf{v}}(s_t) + \mathbf{B}(s_t)\mathbf{h}_t, \boldsymbol{\Sigma}^v(s_t)\right), \qquad p(\mathbf{h}_t|\mathbf{h}_{t-1}, s_t) = \mathcal{N}\left(\mathbf{h}_t\middle|\bar{\mathbf{h}}(s_t) + \mathbf{A}(s_t)\mathbf{h}_t, \boldsymbol{\Sigma}^h(s_t)\right)$$

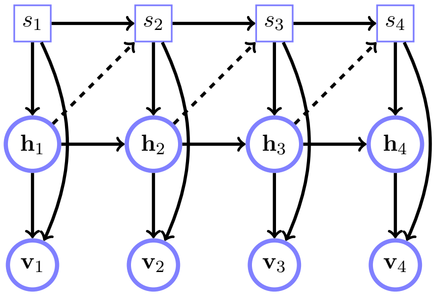


Figure 25.1: The independence structure of the aSLDS. Square nodes s_t denote discrete switch variables; $\mathbf{h}_t$ are continuous latent/hidden variables, and $\mathbf{v}_t$ continuous observed/visible variables. The discrete state s_t determines which Linear Dynamical system from a finite set of Linear Dynamical systems is operational at time t. In the SLDS links from h to s are not normally considered.

(25.2.3)

At time $t = 1$, $p(s_1|\mathbf{h}_0, s_0)$ denotes the prior $p(s_1)$, and $p(\mathbf{h}_1|\mathbf{h}_0, s_1)$ denotes $p(\mathbf{h}_1|s_1)$.

The SLDS can be thought of as a marriage between a hidden Markov model and a Linear Dynamical system. The SLDS is also called a Jump Markov model/process, switching Kalman Filter, Switching Linear Gaussian State Space models, Conditional Linear Gaussian Models.

25.2.1 Exact inference is computationally intractable

Both exact filtered and smoothed inference in the SLDS is intractable, scaling exponentially with time, see for example [172]. As an informal explanation, consider filtered posterior inference, for which, by analogy with equation (23.2.9), the forward pass is

$$p(s_{t+1}, \mathbf{h}_{t+1}|\mathbf{v}_{1:t+1}) = \sum_{s_t} \int_{\mathbf{h}_t} p(s_{t+1}, \mathbf{h}_{t+1}|s_t, \mathbf{h}_t, \mathbf{v}_{t+1}) p(s_t, \mathbf{h}_t|\mathbf{v}_{1:t}) \tag{25.2.4}$$

At timestep 1, $p(s_1, \mathbf{h}_1|\mathbf{v}_1) = p(\mathbf{h}_1|s_1, \mathbf{v}_1)p(s_1|\mathbf{v}_1)$ is an indexed set of Gaussians. At timestep 2, due to the summation over the states s_1, $p(s_2, \mathbf{h}_2|\mathbf{v}_{1:2})$ will be an indexed set of S Gaussians; similarly at timestep 3, it will be S^2 and, in general, gives rise to S^{t-1} Gaussians at time t. Even for small t, the number of components required to exactly represent the filtered distribution is computationally intractable. Analogously, smoothing is also intractable.

The origin of the intractability of the SLDS therefore differs from 'structural intractability' that we've previously encountered. In the SLDS, in terms of the cluster variables $x_{1:T}$, with $x_t \equiv (s_t, \mathbf{h}_t)$ and visible variables $\mathbf{v}_{1:T}$, the graph of the distribution is singly-connected. From a purely graph theoretic viewpoint, one would therefore envisage no difficulty in carrying out inference. Indeed, as we saw above, the derivation of the filtering algorithm is straightforward since the graph is singly-connected. However, the numerical implementation of the algorithm is intractable since the description of the messages requires an exponentially increasing number of terms.

In order to deal with this intractability, several approximation schemes have been introduced [95, 107, 171, 158, 157]. Here we focus on Gaussian inference techniques, which approximate the switch conditional posteriors using a limited mixture of Gaussians. Since the exact posterior distributions are mixtures of Gaussians, but with an exponentially large number of components, the aim is to drop low weight components such that the resulting limited number of Gaussians still accurately represents the posterior[17].

25.3 Gaussian Sum Filtering

Equation(25.2.4) describes the exact filtering recursion. Whilst the number of mixture components increases exponentially with time, intuitively one would expect that there is an effective time-scale over which the previous visible information is relevant. In general, the influence of ancient observations will be much less relevant than that of recent observations. This suggests that the 'effective time' is limited and that therefore a corresponding limited number of components in the Gaussian mixture should suffice to accurately represent the filtered posterior[6].

Our aim is to form a recursion for $p(s_t, \mathbf{h}_t|\mathbf{v}_{1:t})$, based on a Gaussian mixture approximation of $p(\mathbf{h}_t|s_t, \mathbf{v}_{1:t})$. Given an approximation of the filtered distribution $p(s_t, \mathbf{h}_t|\mathbf{v}_{1:t}) \approx q(s_t, \mathbf{h}_t|\mathbf{v}_{1:t})$, the exact recursion equation (25.2.4) is approximated by

$$q(s_{t+1}, \mathbf{h}_{t+1}|\mathbf{v}_{1:t+1}) = \sum_{s_t} \int_{\mathbf{h}_t} p(s_{t+1}, \mathbf{h}_{t+1}|s_t, \mathbf{h}_t, \mathbf{v}_{t+1}) q(s_t, \mathbf{h}_t|\mathbf{v}_{1:t}) \tag{25.3.1}$$

This approximation to the filtered posterior at the next timestep will contain S times more components than in the previous timestep and, to prevent an exponential explosion in mixture components, we will need to collapse this mixture in a suitable way. We will deal with this once the new mixture representation for the filtered posterior has been computed.

To derive the updates it is useful to break the new filtered approximation from equation (25.2.4) into continuous and discrete parts:

$$q(\mathbf{h}_t, s_t|\mathbf{v}_{1:t}) = q(\mathbf{h}_t|s_t, \mathbf{v}_{1:t}) q(s_t|\mathbf{v}_{1:t}) \tag{25.3.2}$$

and derive separate filtered update formulae, as described below.

25.3.1 Continuous filtering

The exact representation of $p(\mathbf{h}_t|s_t, \mathbf{v}_{1:t})$ is a mixture with $O\left(S^{t-1}\right)$ components. To retain computational feasibility we therefore approximate this with a limited I-component mixture

$$q(\mathbf{h}_t|s_t, \mathbf{v}_{1:t}) = \sum_{i_t=1}^{I} q(\mathbf{h}_t|i_t, s_t, \mathbf{v}_{1:t}) q(i_t|s_t, \mathbf{v}_{1:t}) \tag{25.3.3}$$

where $q(\mathbf{h}_t|i_t, s_t, \mathbf{v}_{1:t})$ is a Gaussian parameterised with mean $\mathbf{f}(i_t, s_t)$ and covariance $\mathbf{F}(i_t, s_t)$. Strictly speaking, we should use the notation $\mathbf{f}_t(i_t, s_t)$ since, for each time t, we have a set of means indexed by i_t, s_t, although we drop these dependencies in the notation used here.

An important remark is that many techniques approximate $p(\mathbf{h}_t|s_t, \mathbf{v}_{1:t})$ using a *single* Gaussian. Naturally, this gives rise to a mixture of Gaussians for $p(\mathbf{h}_t|\mathbf{v}_{1:t})$. However, in making a single Gaussian approximation to $p(\mathbf{h}_t|s_t, \mathbf{v}_{1:t})$ the representation of the posterior may be poor. Our aim here is to maintain an accurate approximation to $p(\mathbf{h}_t|s_t, \mathbf{v}_{1:t})$ by using a *mixture* of Gaussians.

To find a recursion for the approximating distribution we first assume that we know the filtered approximation $q(\mathbf{h}_t, s_t|\mathbf{v}_{1:t})$ and then propagate this forwards using the exact dynamics. To do so consider first the exact relation

$$\begin{aligned} q(\mathbf{h}_{t+1}|s_{t+1}, \mathbf{v}_{1:t+1}) &= \sum_{s_t, i_t} q(\mathbf{h}_{t+1}, s_t, i_t|s_{t+1}, \mathbf{v}_{1:t+1}) \\ &= \sum_{s_t, i_t} q(\mathbf{h}_{t+1}|s_t, i_t, s_{t+1}, \mathbf{v}_{1:t+1}) q(s_t, i_t|s_{t+1}, \mathbf{v}_{1:t+1}) \end{aligned} \tag{25.3.4}$$

Wherever possible we substitute the exact dynamics and evaluate each of the two factors above. The usefulness of decomposing the update in this way is that the new filtered approximation is of the form of a Gaussian mixture, where $q(\mathbf{h}_{t+1}|s_t, i_t, s_{t+1}, \mathbf{v}_{1:t+1})$ is Gaussian and $q(s_t, i_t|s_{t+1}, \mathbf{v}_{1:t+1})$ are the weights or mixing proportions of the components. We describe below how to compute these terms explicitly. Equation(25.3.4) produces a new Gaussian mixture with $I \times S$ components which we will collapse back to I components at the end of the computation.

Evaluating $q(\mathbf{h}_{t+1}|s_t, i_t, s_{t+1}, \mathbf{v}_{1:t+1})$

We aim to find a filtering recursion for $q(\mathbf{h}_{t+1}|s_t, i_t, s_{t+1}, \mathbf{v}_{1:t+1})$. Since this is conditional on switch states and components, this corresponds to a single LDS forward step, which can be evaluated by considering first the joint distribution

$$q(\mathbf{h}_{t+1}, \mathbf{v}_{t+1}|s_t, i_t, s_{t+1}, \mathbf{v}_{1:t}) = \int_{\mathbf{h}_t} p(\mathbf{h}_{t+1}, \mathbf{v}_{t+1}|\mathbf{h}_t, s_t, \cancel{i_t}, \cancel{s_{t+1}}, \mathbf{v}_{1:t}) q(\mathbf{h}_t|s_t, i_t, \cancel{s_{t+1}}, \mathbf{v}_{1:t}) \tag{25.3.5}$$

and subsequently conditioning on $\mathbf{v}_{t+1}$. In the above we used the exact dynamics where possible. Equation(25.3.5) states that we know the filtered information up to time t, in addition to knowing the switch states s_t, s_{t+1} and the mixture component index i_t. To ease the burden on notation we derive this for $\bar{\mathbf{h}}_t, \bar{\mathbf{v}}_t \equiv 0$ for all t. The exact forward dynamics is then given by

$$\mathbf{h}_{t+1} = \mathbf{A}(s_{t+1})\mathbf{h}_t + \boldsymbol{\eta}^h(s_{t+1}), \qquad \mathbf{v}_{t+1} = \mathbf{B}(s_{t+1})\mathbf{h}_t + \boldsymbol{\eta}^v(s_{t+1}), \tag{25.3.6}$$

Given the mixture component index i_t,

$$q(\mathbf{h}_t|\mathbf{v}_{1:t}, i_t, s_t) = \mathcal{N}\left(\mathbf{h}_t|\mathbf{f}(i_t, s_t), \mathbf{F}(i_t, s_t)\right) \tag{25.3.7}$$

we propagate this Gaussian with the exact dynamics equation (25.3.6). Then $q(\mathbf{h}_{t+1}, \mathbf{v}_{t+1}|s_t, i_t, s_{t+1}, \mathbf{v}_{1:t})$ is a Gaussian with covariance and mean elements

$$\begin{aligned}
&\boldsymbol{\Sigma}_{hh} = \mathbf{A}(s_{t+1})\mathbf{F}(i_t, s_t)\mathbf{A}^\mathsf{T}(s_{t+1}) + \boldsymbol{\Sigma}^h(s_{t+1}), \quad \boldsymbol{\Sigma}_{vv} = \mathbf{B}(s_{t+1})\boldsymbol{\Sigma}_{hh}\mathbf{B}^\mathsf{T}(s_{t+1}) + \boldsymbol{\Sigma}^v(s_{t+1}) \\
&\boldsymbol{\Sigma}_{vh} = \mathbf{B}(s_{t+1})\boldsymbol{\Sigma}_{hh} = \boldsymbol{\Sigma}_{hv}^\mathsf{T}, \quad \boldsymbol{\mu}_v = \mathbf{B}(s_{t+1})\mathbf{A}(s_{t+1})\mathbf{f}(i_t, s_t), \quad \boldsymbol{\mu}_h = \mathbf{A}(s_{t+1})\mathbf{f}(i_t, s_t)
\end{aligned} \tag{25.3.8}$$

These results are obtained from integrating the forward dynamics, Equations (25.2.1,25.2.2) over $\mathbf{h}_t$, using the results in section(8.6.3).

To find $q(\mathbf{h}_{t+1}|s_t, i_t, s_{t+1}, \mathbf{v}_{1:t+1})$ we condition $q(\mathbf{h}_{t+1}, \mathbf{v}_{t+1}|s_t, i_t, s_{t+1}, \mathbf{v}_{1:t})$ on $\mathbf{v}_{t+1}$ using the standard Gaussian Conditioning formulae definition(78) to obtain

$$q(\mathbf{h}_{t+1}|s_t, i_t, s_{t+1}, \mathbf{v}_{1:t+1}) = \mathcal{N}\left(\mathbf{h}_{t+1}|\boldsymbol{\mu}_{h|v}, \boldsymbol{\Sigma}_{h|v}\right) \tag{25.3.9}$$

with

$$\boldsymbol{\mu}_{h|v} = \boldsymbol{\mu}_h + \boldsymbol{\Sigma}_{hv}\boldsymbol{\Sigma}_{vv}^{-1}\left(\mathbf{v}_{t+1} - \boldsymbol{\mu}_v\right), \qquad \boldsymbol{\Sigma}_{h|v} = \boldsymbol{\Sigma}_{hh} - \boldsymbol{\Sigma}_{hv}\boldsymbol{\Sigma}_{vv}^{-1}\boldsymbol{\Sigma}_{vh} \tag{25.3.10}$$

where the quantities required are defined in equation (25.3.8).

Evaluating the mixture weights $q(s_t, i_t|s_{t+1}, \mathbf{v}_{1:t+1})$

Up to a normalisation constant the mixture weight in equation (25.3.4) can be found from

$$q(s_t, i_t|s_{t+1}, \mathbf{v}_{1:t+1}) \propto q(\mathbf{v}_{t+1}|i_t, s_t, s_{t+1}, \mathbf{v}_{1:t})q(s_{t+1}|i_t, s_t, \mathbf{v}_{1:t})q(i_t|s_t, \mathbf{v}_{1:t})q(s_t|\mathbf{v}_{1:t}) \tag{25.3.11}$$

The first factor in equation (25.3.11), $q(\mathbf{v}_{t+1}|i_t, s_t, s_{t+1}, \mathbf{v}_{1:t})$ is Gaussian with mean $\boldsymbol{\mu}_v$ and covariance $\boldsymbol{\Sigma}_{vv}$, as given in equation (25.3.8). The last two factors $q(i_t|s_t, \mathbf{v}_{1:t})$ and $q(s_t|\mathbf{v}_{1:t})$ are given from the previous filtered iteration. Finally, $q(s_{t+1}|i_t, s_t, \mathbf{v}_{1:t})$ is found from

$$q(s_{t+1}|i_t, s_t, \mathbf{v}_{1:t}) = \begin{cases} \langle p(s_{t+1}|\mathbf{h}_t, s_t)\rangle_{q(\mathbf{h}_t|i_t, s_t, \mathbf{v}_{1:t})} & \text{augmented SLDS} \\ p(s_{t+1}|s_t) & \text{standard SLDS} \end{cases} \tag{25.3.12}$$

where the result above for the standard SLDS follows from the independence assumptions present in the standard SLDS. In the aSLDS, the term in equation (25.3.12) will generally need to be computed numerically. A simple approximation is to evaluate equation (25.3.12) at the mean value of the distribution $q(\mathbf{h}_t|i_t, s_t, \mathbf{v}_{1:t})$. To take covariance information into account an alternative would be to draw samples from the Gaussian $q(\mathbf{h}_t|i_t, s_t, \mathbf{v}_{1:t})$ and thus approximate the average of $p(s_{t+1}|\mathbf{h}_t, s_t)$ by sampling[1].

Closing the recursion

We are now in a position to calculate equation (25.3.4). For each setting of the variable s_{t+1}, we have a mixture of $I \times S$ Gaussians. To prevent the number of components increasing exponentially with time, we numerically collapse $q(\mathbf{h}_{t+1}|s_{t+1}, \mathbf{v}_{1:t+1})$ back to I Gaussians to form

$$q(\mathbf{h}_{t+1}|s_{t+1}, \mathbf{v}_{1:t+1}) \to \sum_{i_{t+1}=1}^{I} q(\mathbf{h}_{t+1}|i_{t+1}, s_{t+1}, \mathbf{v}_{1:t+1})q(i_{t+1}|s_{t+1}, \mathbf{v}_{1:t+1}) \tag{25.3.13}$$

[1] Note that this does not equate Gaussian Sum filtering with a sequential sampling procedure, such as Particle Filtering, section(27.7). The sampling here is exact, for which no convergence issues arise.

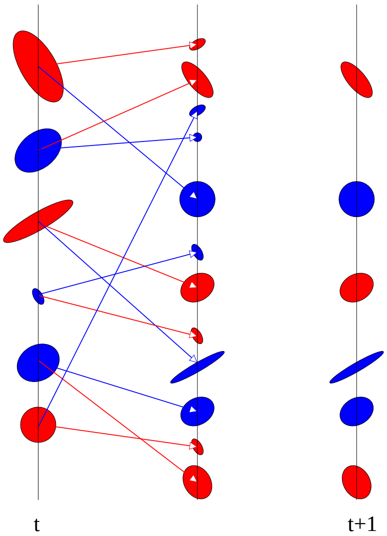


Figure 25.2: Gaussian Sum Filtering. The leftmost column depicts the previous Gaussian mixture approximation $q(\mathbf{h}_t, i_t|\mathbf{v}_{1:t})$ for two states $S = 2$ (red and blue) and three mixture components $I = 3$. There are $S = 2$ different linear systems which take each of the components of the mixture into a new filtered state, the colour of the arrow indicating which dynamic system is used. After one time-step each mixture component branches into a further S components so that the joint approximation $q(\mathbf{h}_{t+1}, s_{t+1}|\mathbf{v}_{1:t+1})$ contains $S^2 I$ components (middle column). To keep the representation computationally tractable the mixture of Gaussians for each state s_{t+1} is collapsed back to I components. This means that each coloured state needs to be approximated by a smaller I component mixture of Gaussians. There are many ways to achieve this. A naive but computationally efficient approach is to simply ignore the lowest weight components, as depicted on the right column, see `mix2mix.m`.

Any method of choice may be supplied to collapse a mixture to a smaller mixture. A straightforward approach is to repeatedly merge low-weight components, as explained in section(25.3.4). In this way the new mixture coefficients $q(i_{t+1}|s_{t+1}, \mathbf{v}_{1:t+1})$, $i_{t+1} \in 1, \ldots, I$ are defined.

This completes the description of how to form a recursion for the continuous filtered posterior approximation $q(\mathbf{h}_{t+1}|s_{t+1}, \mathbf{v}_{1:t+1})$ in equation (25.3.2).

25.3.2 Discrete filtering

A recursion for the switch variable distribution in equation (25.3.2) is

$$q(s_{t+1}|\mathbf{v}_{1:t+1}) \propto \sum_{i_t, s_t} q(s_{t+1}, i_t, s_t, \mathbf{v}_{t+1}, \mathbf{v}_{1:t}) \tag{25.3.14}$$

The r.h.s. of the above equation is proportional to

$$\sum_{s_t, i_t} q(\mathbf{v}_{t+1}|s_{t+1}, i_t, s_t, \mathbf{v}_{1:t}) q(s_{t+1}|i_t, s_t, \mathbf{v}_{1:t}) q(i_t|s_t, \mathbf{v}_{1:t}) q(s_t|\mathbf{v}_{1:t}) \tag{25.3.15}$$

for which all terms have been computed during the recursion for $q(\mathbf{h}_{t+1}|s_{t+1}, \mathbf{v}_{1:t+1})$. We now have all the quantities required to compute the Gaussian Sum approximation of the filtering forward pass. A schematic representation of Gaussian Sum Filtering is given in fig(25.2) and the pseudo code is presented in algorithm(23).

25.3.3 The likelihood $p(\mathbf{v}_{1:T})$

The likelihood $p(\mathbf{v}_{1:T})$ may be found from

$$p(\mathbf{v}_{1:T}) = \prod_{t=0}^{T-1} p(\mathbf{v}_{t+1}|\mathbf{v}_{1:t}) \tag{25.3.16}$$

where

$$p(\mathbf{v}_{t+1}|\mathbf{v}_{1:t}) \approx \sum_{i_t, s_t, s_{t+1}} q(\mathbf{v}_{t+1}|i_t, s_t, s_{t+1}, \mathbf{v}_{1:t}) q(s_{t+1}|i_t, s_t, \mathbf{v}_{1:t}) q(i_t|s_t, \mathbf{v}_{1:t}) q(s_t|\mathbf{v}_{1:t})$$

In the above expression, all terms have been computed in forming the recursion for the filtered posterior $q(\mathbf{h}_{t+1}, s_{t+1}|\mathbf{v}_{1:t+1})$.

Algorithm 23 aSLDS Forward Pass. Approximate the filtered posterior $p(s_t|\mathbf{v}_{1:t}) \equiv \alpha_t$, $p(\mathbf{h}_t|s_t, \mathbf{v}_{1:t}) \equiv \sum_{i_t} w_t(i_t, s_t)\mathcal{N}(\mathbf{h}_t|\mathbf{f}_t(i_t, s_t), \mathbf{F}_t(i_t, s_t))$. Also return the approximate log-likelihood $L \equiv \log p(\mathbf{v}_{1:T})$. I_t are the number of components in each Gaussian mixture approximation. We require $I_1 = 1, I_2 \leq S, I_t \leq S \times I_{t-1}$. $\theta(s) = \mathbf{A}(s), \mathbf{B}(s), \mathbf{\Sigma}^h(s), \mathbf{\Sigma}^v(s), \bar{\mathbf{h}}(s), \bar{\mathbf{v}}(s)$.

```
for s_1 ← 1 to S do
    {f_1(1, s_1), F_1(1, s_1), p̂} = LDSFORWARD(0, 0, v_1; θ(s_1))
    α_1 ← p(s_1) p̂
end for

for t ← 2 to T do
    for s_t ← 1 to S do
        for i ← 1 to I_{t−1}, and s ← 1 to S do
            {μ_{x|y}(i, s), Σ_{x|y}(i, s), p̂} = LDSFORWARD(f_{t−1}(i, s), F_{t−1}(i, s), v_t; θ(s_t))
            p*(s_t|i, s) ≡ ⟨p(s_t|h_{t−1}, s_{t−1} = s)⟩_{p(h_{t−1}|i_{t−1}=i, s_{t−1}=s, v_{1:t−1})}
            p′(s_t, i, s) ← w_{t−1}(i, s) p*(s_t|i, s) α_{t−1}(s) p̂
        end for
        Collapse the I_{t−1} × S mixture of Gaussians defined by μ_{x|y}, Σ_{x|y}, and weights
        p(i, s|s_t) ∝ p′(s_t, i, s) to a Gaussian with I_t components, p(h_t|s_t, v_{1:t}) ≈
        Σ_{i_t=1}^{I_t} p(i_t|s_t, v_{1:t}) p(h_t|s_t, i_t, v_{1:t}). This defines the new means f_t(i_t, s_t), covariances
        F_t(i_t, s_t) and mixture weights w_t(i_t, s_t) ≡ p(i_t|s_t, v_{1:t}).
        Compute α_t(s_t) ∝ Σ_{i,s} p′(s_t, i, s)
    end for
    normalise α_t
    L ← L + log Σ_{s_t,i,s} p′(s_t, i, s)
end for
```

25.3.4 Collapsing Gaussians

Given a mixture of N Gaussians

$$p(\mathbf{x}) = \sum_{i=1}^{N} p_i \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_i, \mathbf{\Sigma}_i) \tag{25.3.17}$$

we wish to collapse this to a smaller $K < N$ mixture of Gaussians. We describe a simple method which has the advantage of computational efficiency, but the disadvantage that no spatial information about the mixture is used[274]. First we describe how to collapse a mixture to a single Gaussian. This can be achieved by finding the mean and covariance of the mixture distribution (25.3.17). These are

$$\boldsymbol{\mu} = \sum_i p_i \boldsymbol{\mu}_i, \qquad \mathbf{\Sigma} = \sum_i p_i \left(\mathbf{\Sigma}_i + \boldsymbol{\mu}_i\boldsymbol{\mu}_i^{\mathsf{T}}\right) - \boldsymbol{\mu}\boldsymbol{\mu}^{\mathsf{T}} \tag{25.3.18}$$

To collapse a mixture then to a K-component mixture we may retain the $K-1$ Gaussians with the largest mixture weights – the remaining $N-K+1$ Gaussians are simply merged to a single Gaussian using the above method. Alternative heuristics such as recursively merging the two Gaussians with the lowest mixture weights are also reasonable.

More sophisticated methods which retain some spatial information would clearly be potentially useful. The method presented in [171] is a suitable approach which considers removing Gaussians which are spatially similar (and not just low-weight components), thereby retaining a sense of diversity over the possible solutions. In practical applications involving time series with many thousands of timepoints, speed can be a factor determining which method of collapsing Gaussians is to be preferred.

25.3.5 Relation to other methods

Gaussian Sum Filtering can be considered a form of 'analytical particle filtering', section(27.7), in which instead of point distributions (delta functions) being propagated, Gaussians are propagated. The collapse operation to a smaller number of Gaussians is analogous to resampling in Particle Filtering. Since a

Gaussian is more expressive than a delta-function, the Gaussian Sum filter is generally an improved approximation technique over using point particles. See [17] for a numerical comparison.

25.4 Gaussian Sum Smoothing

Approximating the smoothed posterior $p(\mathbf{h}_t, s_t|\mathbf{v}_{1:T})$ is more involved than filtering and requires additional approximations. For this reason smoothing is more prone to failure since there are more assumptions that need to be satisfied for the approximations to hold. The route we take here is to assume that a Gaussian Sum filtered approximation has been carried out, and then approximate the γ backward pass, analogous to that of section(23.2.4). By analogy with the RTS smoothing recursion equation (23.2.21), the exact backward pass for the SLDS reads

$$p(\mathbf{h}_t, s_t|\mathbf{v}_{1:T}) = \sum_{s_{t+1}} \int_{\mathbf{h}_{t+1}} p(\mathbf{h}_t, s_t|\mathbf{h}_{t+1}, s_{t+1}, \mathbf{v}_{1:t}) p(\mathbf{h}_{t+1}, s_{t+1}|\mathbf{v}_{1:T}) \tag{25.4.1}$$

where $p(\mathbf{h}_{t+1}, s_{t+1}|\mathbf{v}_{1:T}) = p(s_{t+1}|\mathbf{v}_{1:T})p(\mathbf{h}_{t+1}|s_{t+1}, \mathbf{v}_{1:T})$ is composed of the discrete and continuous components of the smoothed posterior at the next time step. The recursion runs backwards in time, beginning with the initialisation $p(\mathbf{h}_T, s_T|\mathbf{v}_{1:T})$ set by the filtered result (at time $t = T$, the filtered and smoothed posteriors coincide). Apart from the fact that the number of mixture components will increase at each step, computing the integral over $\mathbf{h}_{t+1}$ in equation (25.4.1) is problematic since the conditional distribution term is non-Gaussian in $\mathbf{h}_{t+1}$. For this reason it is more useful derive an approximate recursion by beginning with the exact relation

$$p(s_t, \mathbf{h}_t|\mathbf{v}_{1:T}) = \sum_{s_{t+1}} p(s_{t+1}|\mathbf{v}_{1:T}) p(\mathbf{h}_t|s_t, s_{t+1}, \mathbf{v}_{1:T}) p(s_t|s_{t+1}, \mathbf{v}_{1:T}) \tag{25.4.2}$$

which can be expressed more directly in terms of the SLDS dynamics as

$$p(s_t, \mathbf{h}_t|\mathbf{v}_{1:T}) = \sum_{s_{t+1}} p(s_{t+1}|\mathbf{v}_{1:T}) \left\langle p(\mathbf{h}_t|\mathbf{h}_{t+1}, s_t, s_{t+1}, \mathbf{v}_{1:t}, \cancel{\mathbf{v}_{t+1:T}}) \right\rangle_{p(\mathbf{h}_{t+1}|s_t, s_{t+1}, \mathbf{v}_{1:T})} \times \left\langle p(s_t|\mathbf{h}_{t+1}, s_{t+1}, \mathbf{v}_{1:T}) \right\rangle_{p(\mathbf{h}_{t+1}|s_{t+1}, \mathbf{v}_{1:T})} \tag{25.4.3}$$

In forming the recursion we assume access to the distribution $p(s_{t+1}, \mathbf{h}_{t+1}|\mathbf{v}_{1:T})$ from the future timestep. However, we also require the distribution $p(\mathbf{h}_{t+1}|s_t, s_{t+1}, \mathbf{v}_{1:T})$ which is not directly known and needs to be inferred, a computationally challenging task. In the *Expectation Correction* (EC) approach [17] one assumes the approximation (see fig(25.3))

$$p(\mathbf{h}_{t+1}|s_t, s_{t+1}, \mathbf{v}_{1:T}) \approx p(\mathbf{h}_{t+1}|s_{t+1}, \mathbf{v}_{1:T}) \tag{25.4.4}$$

resulting in an approximate recursion for the smoothed posterior,

$$p(s_t, \mathbf{h}_t|\mathbf{v}_{1:T}) \approx \sum_{s_{t+1}} p(s_{t+1}|\mathbf{v}_{1:T}) \left\langle p(\mathbf{h}_t|\mathbf{h}_{t+1}, s_t, s_{t+1}, \mathbf{v}_{1:t}) \right\rangle_{\mathbf{h}_{t+1}} \left\langle p(s_t|\mathbf{h}_{t+1}, s_{t+1}, \mathbf{v}_{1:T}) \right\rangle_{\mathbf{h}_{t+1}} \tag{25.4.5}$$

where $\langle \cdot \rangle_{\mathbf{h}_{t+1}}$ represents averaging with respect to the distribution $p(\mathbf{h}_{t+1}|s_{t+1}, \mathbf{v}_{1:T})$. In carrying out the approximate recursion, (25.4.5) we will end up with a mixture of Gaussians that grows at each timestep. To avoid the exponential explosion problem, we use a finite mixture approximation,

$$p(\mathbf{h}_{t+1}, s_{t+1}|\mathbf{v}_{1:T}) \approx q(\mathbf{h}_{t+1}, s_{t+1}|\mathbf{v}_{1:T}) = q(\mathbf{h}_{t+1}|s_{t+1}, \mathbf{v}_{1:T}) q(s_{t+1}|\mathbf{v}_{1:T}) \tag{25.4.6}$$

and plug this into the approximate recursion above. From equation (25.4.1) a recursion for the approximation is given by

$$q(\mathbf{h}_t, s_t|\mathbf{v}_{1:T}) = \sum_{s_{t+1}} q(s_{t+1}|\mathbf{v}_{1:T}) \left\langle q(\mathbf{h}_t|\mathbf{h}_{t+1}, s_t, s_{t+1}, \mathbf{v}_{1:t}) \right\rangle_{q(\mathbf{h}_{t+1}|s_{t+1}, \mathbf{v}_{1:T})} \left\langle q(s_t|\mathbf{h}_{t+1}, s_{t+1}, \mathbf{v}_{1:t}) \right\rangle_{q(\mathbf{h}_{t+1}|s_{t+1}, \mathbf{v}_{1:T})} \tag{25.4.7}$$

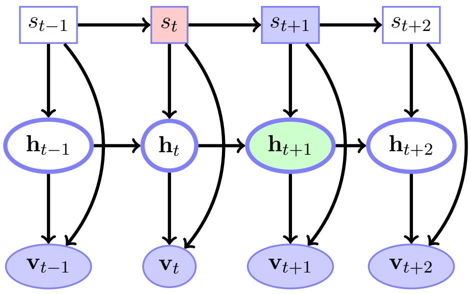


Figure 25.3: The EC backpass approximates $p(\mathbf{h}_{t+1}|s_{t+1}, s_t, \mathbf{v}_{1:T})$ by $p(\mathbf{h}_{t+1}|s_{t+1}, \mathbf{v}_{1:T})$. The motivation for this is that s_t influences $\mathbf{h}_{t+1}$ only indirectly through $\mathbf{h}_t$. However, $\mathbf{h}_t$ will most likely be heavily influenced by $\mathbf{v}_{1:t}$, so that not knowing the state of s_t is likely to be of secondary importance. The green shaded node is the variable we wish to find the posterior for. The values of the blue shaded nodes are known, and the red shaded node indicates a known variable which is assumed unknown in forming the approximation.

As for filtering, wherever possible, we replace approximate terms by their exact counterparts and parameterise the posterior using

$$q(\mathbf{h}_{t+1}, s_{t+1}|\mathbf{v}_{1:T}) = q(\mathbf{h}_{t+1}|s_{t+1}, \mathbf{v}_{1:T})q(s_{t+1}|\mathbf{v}_{1:T}) \tag{25.4.8}$$

To reduce the notational burden here we outline the method only for the case of using a single component approximation in both the Forward and Backward passes. The extension to using a mixture to approximate each $p(\mathbf{h}_{t+1}|s_{t+1}, \mathbf{v}_{1:T})$ is conceptually straightforward and deferred to section(25.4.5). In the single Gaussian case we assume we have a Gaussian approximation available for

$$q(\mathbf{h}_{t+1}|s_{t+1}, \mathbf{v}_{1:T}) = \mathcal{N}(\mathbf{h}_{t+1}|\mathbf{g}(s_{t+1}, \mathbf{G}(s_{t+1})) \tag{25.4.9}$$

25.4.1 Continuous smoothing

For given s_t, s_{t+1}, an RTS style recursion for the smoothed continuous is obtained using

$$q(\mathbf{h}_t|s_t, s_{t+1}, \mathbf{v}_{1:T}) = \int_{\mathbf{h}_{t+1}} p(\mathbf{h}_t|\mathbf{h}_{t+1}, s_t, s_{t+1}, \mathbf{v}_{1:t})q(\mathbf{h}_{t+1}|s_{t+1}, \mathbf{v}_{1:T}) \tag{25.4.10}$$

In forming the recursion, we assume that we know the distribution

$$q(\mathbf{h}_{t+1}|s_{t+1}, \mathbf{v}_{1:T}) = \mathcal{N}(\mathbf{h}_{t+1}|\mathbf{g}(s_{t+1}), \mathbf{G}(s_{t+1})) \tag{25.4.11}$$

To compute equation (25.4.10) we then perform a single update of the LDS backward recursion, section(24.4.2).

25.4.2 Discrete smoothing

The second average in (25.4.8) corresponds to a recursion for the discrete variable and is given by

$$\langle q(s_t|\mathbf{h}_{t+1}, s_{t+1}, \mathbf{v}_{1:t})\rangle_{q(\mathbf{h}_{t+1}|s_{t+1}, \mathbf{v}_{1:T})} \equiv q(s_t|s_{t+1}, \mathbf{v}_{1:T}). \tag{25.4.12}$$

The average of $q(s_t|\mathbf{h}_{t+1}, s_{t+1}, \mathbf{v}_{1:t})$ with respect to $q(\mathbf{h}_{t+1}|s_{t+1}, \mathbf{v}_{1:T})$ cannot be achieved in closed form. The simplest approach is to approximate the average by evaluation at the mean[2]

$$\langle q(s_t|\mathbf{h}_{t+1}, s_{t+1}\mathbf{v}_{1:t})\rangle_{q(\mathbf{h}_{t+1}|s_{t+1}, \mathbf{v}_{1:T})} \approx q(s_t|\mathbf{h}_{t+1}, s_{t+1}, \mathbf{v}_{1:t})\big|_{\mathbf{h}_{t+1}=\langle \mathbf{h}_{t+1}|s_{t+1}, \mathbf{v}_{1:T}\rangle} \tag{25.4.13}$$

where $\langle \mathbf{h}_{t+1}|s_{t+1}, \mathbf{v}_{1:T}\rangle$ is the mean of $\mathbf{h}_{t+1}$ with respect to $q(\mathbf{h}_{t+1}|s_{t+1}, \mathbf{v}_{1:T})$.

Replacing $\mathbf{h}_{t+1}$ by its mean gives the approximation

$$\langle q(s_t|\mathbf{h}_{t+1}, s_{t+1}, \mathbf{v}_{1:t})\rangle_{q(\mathbf{h}_{t+1}|s_{t+1}, \mathbf{v}_{1:T})} \approx \frac{1}{Z}\frac{e^{-\frac{1}{2}z_{t+1}^{\mathsf{T}}(s_t, s_{t+1})\mathbf{\Sigma}^{-1}(s_t, s_{t+1}|\mathbf{v}_{1:t})z_{t+1}(s_t, s_{t+1})}}{\sqrt{\det\left(\mathbf{\Sigma}(s_t, s_{t+1}|\mathbf{v}_{1:t})\right)}} q(s_t|s_{t+1}, \mathbf{v}_{1:t}) \tag{25.4.14}$$

[2] In general this approximation has the form $\langle f(x)\rangle_{p(x)} \approx f(\langle x\rangle_{p(x)})$.

where

$$z_{t+1}(s_t, s_{t+1}) \equiv \langle \mathbf{h}_{t+1} | s_{t+1}, \mathbf{v}_{1:T} \rangle - \langle \mathbf{h}_{t+1} | s_t, s_{t+1}, \mathbf{v}_{1:t} \rangle \tag{25.4.15}$$

and Z ensures normalisation over s_t. $\mathbf{\Sigma}(s_t, s_{t+1}|\mathbf{v}_{1:t})$ is the filtered covariance of $\mathbf{h}_{t+1}$ given s_t, s_{t+1} and the observations $\mathbf{v}_{1:t}$, which may be taken from $\mathbf{\Sigma}_{hh}$ in equation (25.3.8). Approximations which take covariance information into account can also be considered, although the above simple (and fast) method may suffice in practice [17, 190].

25.4.3 Collapsing the mixture

From section(25.4.1) and section(25.4.2) we now have all the terms in equation (25.4.8) to compute the approximation to equation (25.4.7). As for the filtering, however, the number of mixture components is multiplied by S at each iteration. To prevent an exponential explosion of components, the mixture is then collapsed to a single Gaussian

$$q(\mathbf{h}_t, s_t|\mathbf{v}_{1:T}) = q(\mathbf{h}_t|s_t, \mathbf{v}_{1:T})q(s_t|\mathbf{v}_{1:T})) \tag{25.4.16}$$

The collapse to a mixture is discussed in section(25.4.5).

25.4.4 Relation to other methods

A classical smoothing approximation for the SLDS is *generalised pseudo Bayes* (GPB) [13, 157, 156]. In GPB, the following approximation is made

$$p(s_t|s_{t+1}, \mathbf{v}_{1:T}) \approx p(s_t|s_{t+1}, \mathbf{v}_{1:t}) \tag{25.4.17}$$

which depends only on the filtered posterior for s_t and does not include any information passing through the variable $\mathbf{h}_{t+1}$. Since

$$p(s_t|s_{t+1}, \mathbf{v}_{1:t}) \propto p(s_{t+1}|s_t)p(s_t|\mathbf{v}_{1:t}) \tag{25.4.18}$$

computing the smoothed recursion for the switch states in GPB is equivalent to running the RTS backward pass on a hidden Markov model, independently of the backward recursion for the continuous variables:

$$p(s_t|\mathbf{v}_{1:T}) = \sum_{s_{t+1}} p(s_t, s_{t+1}|\mathbf{v}_{1:T}) \tag{25.4.19}$$

$$= \sum_{s_{t+1}} p(s_t|s_{t+1}, \mathbf{v}_{1:T})p(s_{t+1}|\mathbf{v}_{1:T}) \tag{25.4.20}$$

$$\approx \sum_{s_{t+1}} p(s_t|s_{t+1}, \mathbf{v}_{1:t})p(s_{t+1}|\mathbf{v}_{1:T}) \tag{25.4.21}$$

$$= \sum_{s_{t+1}} \frac{p(s_{t+1}|s_t)p(s_t|\mathbf{v}_{1:t})}{\sum_{s_t} p(s_{t+1}|s_t)p(s_t|\mathbf{v}_{1:t})} p(s_{t+1}|\mathbf{v}_{1:T}) \tag{25.4.22}$$

The only information the GPB method uses to form the smoothed distribution $p(s_t|\mathbf{v}_{1:T})$ from the filtered distribution $p(s_t|\mathbf{t}_{1:t})$ is the Markov switch transition $p(s_{t+1}|s_t)$. This approximation drops information from the future since information passed via the continuous variables is not taken into account. In contrast to GPB, the EC Gaussian smoothing technique preserves future information passing through the continuous variables. In `SLDSbackward.m` one may choose to use either EC or GBP.

25.4.5 Using mixtures in the smoothing

The extension to the mixture case is straightforward, based on the representation

$$p(\mathbf{h}_t|s_t, \mathbf{v}_{1:T}) \approx \sum_{j_t=1}^{J} q(j_t|s_t, \mathbf{v}_{1:T})q(\mathbf{h}_t|j_t, \mathbf{v}_{1:T}). \tag{25.4.23}$$

Algorithm 24 aSLDS: EC Backward Pass. Approximates $p(s_t|\mathbf{v}_{1:T})$ and $p(\mathbf{h}_t|s_t, \mathbf{v}_{1:T}) \equiv \sum_{j_t=1}^{J_t} u_t(j_t, s_t)\mathcal{N}(\mathbf{g}_t(j_t, s_t), \mathbf{G}_t(j_t, s_t))$ using a mixture of Gaussians. $J_T = I_T, J_t \leq S \times I_t \times J_{t+1}$. This routine needs the results from algorithm(23).

$\mathbf{G}_T \leftarrow \mathbf{F}_T$, $\mathbf{g}_T \leftarrow \mathbf{f}_T$, $u_T \leftarrow w_T$
for $t \leftarrow T-1$ **to** 1 **do**
  for $s \leftarrow 1$ **to** S, $s' \leftarrow 1$ **to** S, $i \leftarrow 1$ **to** I_t, $j' \leftarrow 1$ **to** J_{t+1} **do**
    $(\boldsymbol{\mu}, \boldsymbol{\Sigma})(i, s, j', s') = \text{LDSBACKWARD}(\mathbf{g}_{t+1}(j', s'), \mathbf{G}_{t+1}(j', s'), \mathbf{f}_t(i, s), \mathbf{F}_t(i, s), \theta(s'))$
    $p(i_t, s_t|j_{t+1}, s_{t+1}, v_{1:T}) = \langle p(s_t = s, i_t = i|\mathbf{h}_{t+1}, s_{t+1} = s', j_{t+1} = j', \mathbf{v}_{1:t})\rangle_{p(\mathbf{h}_{t+1}|s_{t+1}=s', j_{t+1}=j', \mathbf{v}_{1:T})}$
    $p(i, s, j', s'|\mathbf{v}_{1:T}) \leftarrow p(s_{t+1} = s'|\mathbf{v}_{1:T})u_{t+1}(j', s')p(i_t, s_t|j_{t+1}, s_{t+1}, v_{1:T})$
  end for
  for $s_t \leftarrow 1$ **to** S **do**
    Collapse the mixture defined by weights $p(i_t = i, s_{t+1} = s', j_{t+1} = j'|s_t, \mathbf{v}_{1_T}) \propto p(i, s_t, j', s'|\mathbf{v}_{1:T})$, means $\boldsymbol{\mu}(i_t, s_t, j_{t+1}, s_{t+1})$ and covariances $\boldsymbol{\Sigma}(i_t, s_t, j_{t+1}, s_{t+1})$ to a mixture with J_t components. This defines the new means $\mathbf{g}_t(j_t, s_t)$, covariances $\mathbf{G}_t(j_t, s_t)$ and mixture weights $u_t(j_t, s_t)$.
    $p(s_t|\mathbf{v}_{1:T}) \leftarrow \sum_{i_t, j', s'} p(i_t, s_t, j', s'|\mathbf{v}_{1:T})$
  end for
end for

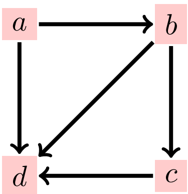


Figure 25.4: A representation of the traffic flow between junctions at a,b,c,d, with traffic lights at a and b. If $s_a = 1$ $a \to d$ and $a \to b$ carry 0.75 and 0.25 of the flow out of a respectively. If $s_a = 2$ all the flow from a goes through $a \to d$; for $s_a = 3$ all the flow goes through $a \to b$. For $s_b = 1$ the flow out of b is split equally between $b \to d$ and $b \to c$. For $s_b = 2$ all flow out of b goes along $b \to c$.

Analogously to the case with a single component,

$$q(\mathbf{h}_t, s_t|\mathbf{v}_{1:T}) = \sum_{i_t, j_{t+1}, s_{t+1}} p(s_{t+1}|\mathbf{v}_{1:T})p(j_{t+1}|s_{t+1}, \mathbf{v}_{1:T})q(\mathbf{h}_t|j_{t+1}, s_{t+1}, i_t, s_t, \mathbf{v}_{1:T}) \times \langle q(i_t, s_t|\mathbf{h}_{t+1}, j_{t+1}, s_{t+1}, \mathbf{v}_{1:t})\rangle_{q(\mathbf{h}_{t+1}|j_{t+1}, s_{t+1}, \mathbf{v}_{1:T})}$$

The average in the last line of the above equation can be tackled using the same techniques as outlined in the single Gaussian case. To approximate $q(\mathbf{h}_t|j_{t+1}, s_{t+1}, i_t, s_t, \mathbf{v}_{1:T})$ we consider this as the marginal of the joint distribution

$$q(\mathbf{h}_t, \mathbf{h}_{t+1}|i_t, s_t, j_{t+1}, s_{t+1}, \mathbf{v}_{1:T}) = q(\mathbf{h}_t|\mathbf{h}_{t+1}, i_t, s_t, j_{t+1}, s_{t+1}, \mathbf{v}_{1:t})q(\mathbf{h}_{t+1}|i_t, s_t, j_{t+1}, s_{t+1}, \mathbf{v}_{1:T})$$

As in the case of a single mixture, the problematic term is $q(\mathbf{h}_{t+1}|i_t, s_t, j_{t+1}, s_{t+1}, \mathbf{v}_{1:T})$. Analogously to equation (25.4.4), we make the assumption

$$q(\mathbf{h}_{t+1}|i_t, s_t, j_{t+1}, s_{t+1}, \mathbf{v}_{1:T}) \approx q(\mathbf{h}_{t+1}|j_{t+1}, s_{t+1}, \mathbf{v}_{1:T}) \tag{25.4.24}$$

meaning that information about the current switch state s_t, i_t is ignored. We can then form

$$p(\mathbf{h}_t|s_t, \mathbf{v}_{1:T}) = \sum_{i_t, j_{t+1}, s_{t+1}} p(i_t, j_{t+1}, s_{t+1}|s_t, \mathbf{v}_{1:T})p(\mathbf{h}_t|i_t, s_t, j_{t+1}, s_{t+1}, \mathbf{v}_{1:T})$$

This mixture can then be collapsed to a smaller mixture using any method of choice, to give

$$p(\mathbf{h}_t|s_t, \mathbf{v}_{1:T}) \approx \sum_{j_t} q(j_t|s_t, \mathbf{v}_{1:T})q(\mathbf{h}_t|j_t, \mathbf{v}_{1:T}) \tag{25.4.25}$$

The resulting procedure is sketched in algorithm(24), including using mixtures in both forward and backward passes.

Example 107 (Traffic Flow)**.** A toy example to illustrate modelling and inference with a SLDS is to consider a simple network of traffic flow, fig(25.4). Here there are 4 junctions a, b, c, d and traffic flows along the roads in the direction indicated. Traffic flows into the junction and then goes via different routes to d. Flow out of a junction must match the flow in to a junction (up to noise). There are traffic light switches at junction a and b which, depending on their state, route traffic differently along the roads.

Using ϕ to denote the clean (noise free) flow, we model the flows using the switching linear system:

$$\left.\begin{array}{r} \phi_a(t) \\ \phi_{a\to d}(t) \\ \phi_{a\to b}(t) \\ \phi_{b\to d}(t) \\ \phi_{b\to c}(t) \\ \phi_{c\to d}(t) \end{array}\right\} = \left\{\begin{array}{l} \phi_a(t-1) \\ \phi_a(t-1)\left(0.75\times\mathbb{I}\left[s_a(t)=1\right]+1\times\mathbb{I}\left[s_a(t)=2\right]+0\times\mathbb{I}\left[s_a(t)=3\right]\right) \\ \phi_a(t-1)\left(0.25\times\mathbb{I}\left[s_a(t)=1\right]+0\times\mathbb{I}\left[s_a(t)=2\right]+1\times\mathbb{I}\left[s_a(t)=3\right]\right) \\ \phi_{a\to b}(t-1)\left(0.5\times\mathbb{I}\left[s_b(t)=1\right]+0\times\mathbb{I}\left[s_b(t)=2\right]\right) \\ \phi_{a\to b}(t-1)\left(0.5\times\mathbb{I}\left[s_b(t)=1\right]+1\times\mathbb{I}\left[s_b(t)=2\right]\right) \\ \phi_{b\to c}(t-1) \end{array}\right. \tag{25.4.26}$$

By identifying the flows at time t with a 6 dimensional vector hidden variable $\mathbf{h}(t)$, we can write the above flow equations as

$$\mathbf{h}_t = \mathbf{A}(s)\mathbf{h}_{t-1} + \boldsymbol{\eta}_t^h \tag{25.4.27}$$

for a set of suitably defined matrices $\mathbf{A}(s)$ indexed by the switch variable $s = s_a \bigotimes s_b$, which takes $3 \times 2 = 6$ states. We additionally include noise terms to model cars parking or de-parking during a single time frame. The covariance $\boldsymbol{\Sigma}^h$ is diagonal with a larger variance at the inflow point a to model that the total volume of traffic entering the system can vary.

Noisy measurements of the flow into the network are taken at a

$$v_{1,t} = \phi_a(t) + \eta_1^v(t) \tag{25.4.28}$$

along with a noisy measurement of the total flow out of the system at d,

$$v_{2,t} = \phi_d(t) = \phi_{a\to d}(t) + \phi_{b\to d}(t) + \phi_{c\to d}(t) + \eta_2^v(t) \tag{25.4.29}$$

The observation model can be represented by $\mathbf{v}_t = \mathbf{B}\mathbf{h}_t + \boldsymbol{\eta}_t^v$ using a constant 2×6 projection matrix $\mathbf{B}$. This is clearly a very crude model of traffic flows since in a real system one cannot have negative flows. Nevertheless it serves to demonstrate the principles of modelling and inference using switching models.

The switch variables follow a simple Markov transition $p(s_t|s_{t-1})$ which biases the switches to remain in the same state in preference to jumping to another state. See `demoSLDStraffic.m` for details.

Given the above system and a prior which initialises all flow at a, we draw samples from the model using forward (ancestral) sampling and form the observations $\mathbf{v}_{1:100}$, fig(25.5). Using only the observations and the known model structure we then attempt to infer the latent switch variables and traffic flows using Gaussian Sum Filtering and Smoothing (EC method) with 2 mixture components per switch state, fig(25.6).

Example 108 (Following the price trend)**.** The following is a simple model of the price trend of a stock,

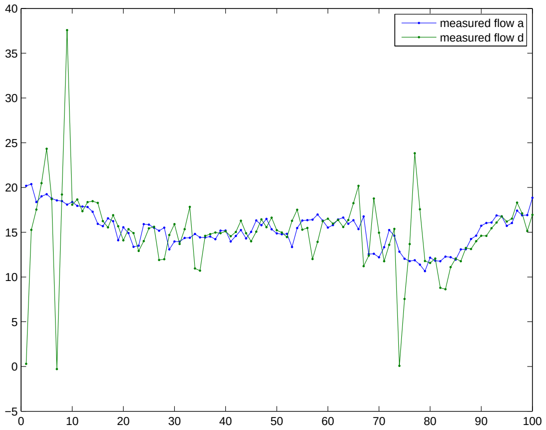


Figure 25.5: Time evolution of the traffic flow measured at two points in the network. Sensors measure the total flow into the network $\phi_a(t)$ and the total flow out of the network, $\phi_d(t) = \phi_{a\to d}(t) + \phi_{b\to d}(t) + \phi_{c\to d}(t)$. The total inflow at a undergoes a random walk. Note that the flow measured at d can momentarily drop to zero if all traffic is routed through $a \to b \to c$ two time steps.

which assumes that the price tends to continue going up (or down) for a while before it reverses direction:

$$h_1(t) = h_1(t-1) + h_2(t-1) + \eta_1^h(s_t) \tag{25.4.30}$$

$$h_2(t) = \mathbb{I}\left[s(t-1) = 1\right] h_2(t-1) + \eta_2^h(s_t) \tag{25.4.31}$$

$$v(t) = h_1(t) + \eta^v(s_t) \tag{25.4.32}$$

here h_1 represents the price and h_2 the direction. There is only a single observation variable at each time, which is the price plus a small amount of noise. There are two switch states, $\mathrm{dom}(s_t) = \{1, 2\}$. When $s_t = 1$, the model functions normally, with the direction being equal to the previous direction plus a small amount of noise. When $s_t = 2$ however, the direction is sampled from a Gaussian with a large variance. The transition $p(s_t|s_{t-1})$ is set so that normal dynamics is more likely, and when $s_t = 2$ it is likely to go back to normal dynamics the next timestep. Full details are in `SLDSpricemodel.mat`. In fig(25.7) we plot some samples from the model and also smoothed inference of the switch distribution, showing how we can a posteriori infer the likely changes in the stock price direction. See exercise(240).

25.5 Reset models

Reset models are special switching models in which the switch state isolates the present from the past, resetting the position of the latent dynamics (these are also known as changepoint models). Whilst these models are rather general, it can be helpful to consider a specific model, and here we consider the SLDS changepoint model with two states. We use the state $s_t = 0$ to denote that the LDS continues with the standard dynamics. With $s_t = 1$, however, the continuous dynamics is reset to a prior:

$$p(\mathbf{h}_t|\mathbf{h}_{t-1}, s_t) = \begin{cases} p^0(\mathbf{h}_t|\mathbf{h}_{t-1}) & s_t = 0 \\ p^1(\mathbf{h}_t) & s_t = 1 \end{cases} \tag{25.5.1}$$

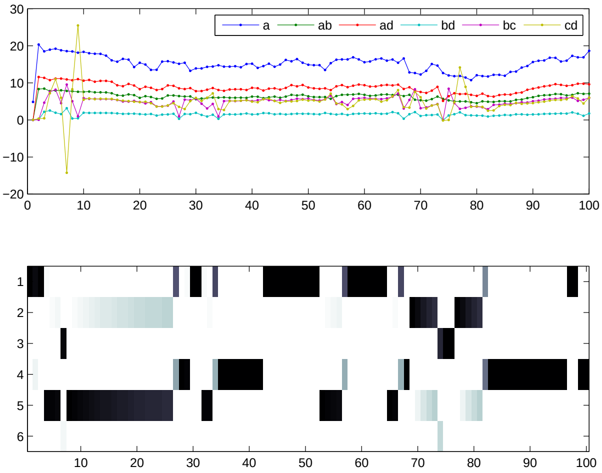
where

$$p^0(\mathbf{h}_t|\mathbf{h}_{t-1}) = \mathcal{N}\left(\mathbf{h}_t \middle| \mathbf{A}\mathbf{h}_{t-1} + \boldsymbol{\mu}^0, \boldsymbol{\Sigma}^0\right), \qquad p^1(\mathbf{h}_t) = \mathcal{N}\left(\mathbf{h}_t \middle| \boldsymbol{\mu}^1, \boldsymbol{\Sigma}^1\right) \tag{25.5.2}$$

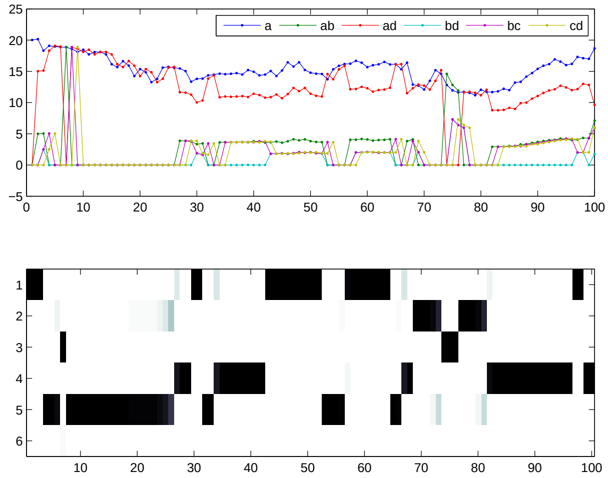
similarly we write

$$p(\mathbf{v}_t|\mathbf{h}_t, s_t) = \begin{cases} p^0(\mathbf{v}_t|\mathbf{h}_t) & s_t = 0 \\ p^1(\mathbf{v}_t|\mathbf{h}_t) & s_t = 1 \end{cases} \tag{25.5.3}$$

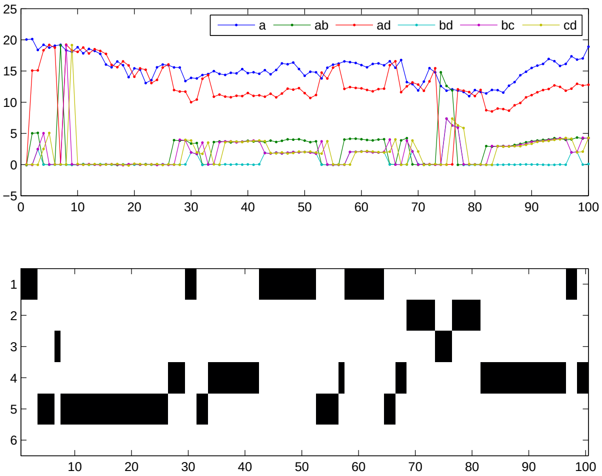
The switch dynamics are first order Markov with transition $p(s_t|s_{t-1})$. Under this model the dynamics follows a standard LDS, but when $s_t = 1$, $\mathbf{h}_t$ is reset to a value drawn from a Gaussian distribution, independent of the past. Such models might be of interest in prediction where the time-series is following a trend but suddenly changes and the past is forgotten. Whilst this may not seem like a big change to the



(a)



(b)



(c)

Figure 25.6: Given the observations from fig(25.5) we infer the flows and switch states of all the latent variables. **(a)**: Filtered flows $\langle \mathbf{h}_t|\mathbf{v}_{1:t}\rangle$ with the posterior distribution of the 6 states $p(s_t|\mathbf{v}_{1:t})$ plotted below. $I = 2$ mixtures where used in the Gaussian Sum forward pass approximation. **(b)**: Smoothed flows $\langle \mathbf{h}_t|\mathbf{v}_{1:T}\rangle$ and corresponding smoothed switch states $p(s_t|\mathbf{v}_{1:T})$. $J = 1$ mixtures where used in the Gaussian Sum backward pass approximation. **(c)**: The correct latent flows through time along with the switch variable state used to generate the data.

model, this model is computationally more tractable, scaling with $O\left(T^2\right)$, compared to $O\left(T2^T\right)$ in the general SLDS. To see this, consider the filtering recursion

$$p(\mathbf{h}_t, s_t|\mathbf{v}_{1:t}) \propto \int_{\mathbf{h}_{t-1}} \sum_{s_{t-1}} p(\mathbf{v}_t|\mathbf{h}_t, s_t)p(\mathbf{h}_t|\mathbf{h}_{t-1}, s_t)p(s_t|s_{t-1})p(\mathbf{h}_{t-1}, s_{t-1}|\mathbf{v}_{1:t-1}) \tag{25.5.4}$$

We now consider the two cases

$$p(\mathbf{h}_t, s_t = 0|\mathbf{v}_{1:t}) \propto \int_{\mathbf{h}_{t-1}} \sum_{s_{t-1}} p^0(\mathbf{v}_t|\mathbf{h}_t)p^0(\mathbf{h}_t|\mathbf{h}_{t-1})p(s_t = 0|s_{t-1})p(\mathbf{h}_{t-1}, s_{t-1}|\mathbf{v}_{1:t-1}) \tag{25.5.5}$$

$$\begin{aligned} p(\mathbf{h}_t, s_t = 1|\mathbf{v}_{1:t}) &\propto p^1(\mathbf{v}_t|\mathbf{h}_t)p^1(\mathbf{h}_t) \int_{\mathbf{h}_{t-1}} \sum_{s_{t-1}} p(s_t = 1|s_{t-1})p(\mathbf{h}_{t-1}, s_{t-1}|\mathbf{v}_{1:t-1}) \\ &\propto p^1(\mathbf{v}_t|\mathbf{h}_t)p^1(\mathbf{h}_t) \sum_{s_{t-1}} p(s_t = 1|s_{t-1})p(s_{t-1}|\mathbf{v}_{1:t-1}) \end{aligned} \tag{25.5.6}$$

Equation(25.5.6) shows that $p(\mathbf{h}_t, s_t = 1|\mathbf{v}_{1:t})$ is not a mixture model in $\mathbf{h}_t$, but contains only a single component proportional to $p^1(\mathbf{v}_t|\mathbf{h}_t)p^1(\mathbf{h}_t)$. If we use this information in equation (25.5.5) we have

$$\begin{aligned} p(\mathbf{h}_t, s_t = 0|\mathbf{v}_{1:t}) \propto &\int_{\mathbf{h}_{t-1}} p^0(\mathbf{v}_t|\mathbf{h}_t)p^0(\mathbf{h}_t|\mathbf{h}_{t-1})p(s_t = 0|s_{t-1} = 0)p(\mathbf{h}_{t-1}, s_{t-1} = 0|\mathbf{v}_{1:t-1}) \\ &+ \int_{\mathbf{h}_{t-1}} p^0(\mathbf{v}_t|\mathbf{h}_t)p^0(\mathbf{h}_t|\mathbf{h}_{t-1})p(s_t = 0|s_{t-1} = 1)p(\mathbf{h}_{t-1}, s_{t-1} = 1|\mathbf{v}_{1:t-1}) \end{aligned} \tag{25.5.7}$$

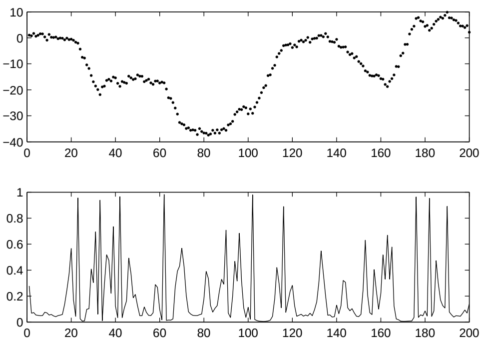


Figure 25.7: The top panel is a time series of 'prices'. The prices tend to keep going up or down with infrequent changes in the direction. Based on fitting a simple SLDS model to capture this kind of behaviour, the probability of a significant change in the price direction is given in the panel below, based on the smoothed distribution $p(s_t = 2|v_{1:T})$.

If we assume that $p(\mathbf{h}_{t-1}, s_{t-1} = 0|\mathbf{v}_{1:t-1})$ is a mixture distribution with K components, then $p(\mathbf{h}_t, s_t = 0|\mathbf{v}_{1:t})$ will contain $K+1$ components. In general, therefore, $p(\mathbf{h}_t, s_t = 0|\mathbf{v}_{1:t})$ will contain T components and $p(\mathbf{h}_t, s_t = 1|\mathbf{v}_{1:t})$ a single component. As opposed to the full SLDS case, the number of components grows only linearly with time, as opposed to exponentially. This means that the computational effort to perform exact filtering scales as $O\left(T^2\right)$.

Run-length formalism

One may also describe reset models using a 'run-length' formalism using at each time t a latent variable r_t which describes the length of the current segment[3]. If there is a change, the run-length variable is reset to zero, otherwise it is increased by 1:

$$p(r_t|r_{t-1}) = \begin{cases} P_{cp} & r_t = 0 \\ 1 - P_{cp} & r_t = r_{t-1} + 1 \end{cases} \tag{25.5.8}$$

where P_{cp} is the probability of a reset (or 'changepoint'). The joint distribution is given by

$$p(v_{1:T}, r_{1:T}) = \prod_t p(r_t|r_{t-1})p(v_t|v_{1:t-1}, r_t) \tag{25.5.9}$$

and

$$p(v_t|v_{1:t-1}, r_t) = p(v_t|v_{t-r_t:t-1}) \tag{25.5.10}$$

with the understanding that if $r_t = 0$ then $p(v_t|v_{t-r_t:t-1}) = p(v_t)$. The graphical model of this distribution is awkward to draw since the number of links depends on the run-length r_t.

Predictions can be made using

$$p(v_{t+1}|v_{1:t}) = \sum_{r_t} p(v_{t+1}|v_{t-r_t:t})p(r_t|v_{1:t}) \tag{25.5.11}$$

where the filtered 'run-length' $p(r_t|v_{1:t})$ is given by the forward recursion:

$$\begin{aligned} p(r_t, v_{1:t}) &= \sum_{r_{t-1}} p(r_t, r_{t-1}, v_{1:t-1}, v_t) = \sum_{r_{t-1}} p(r_t, v_t|r_{t-1}, v_{1:t-1})p(r_{t-1}, v_{1:t-1}) \\ &= \sum_{r_{t-1}} p(v_t|r_t, \cancel{r_{t-1}}, v_{1:t-1})p(r_t|r_{t-1}, \cancel{v_{1:t-1}})p(r_{t-1}, v_{1:t-1}) \\ &= \sum_{r_{t-1}} p(r_t|r_{t-1})p(v_t|v_{t-r_t:t-1})p(r_{t-1}, v_{1:t-1}) \end{aligned}$$

which shows that filtered inference scales with $O\left(T^2\right)$.

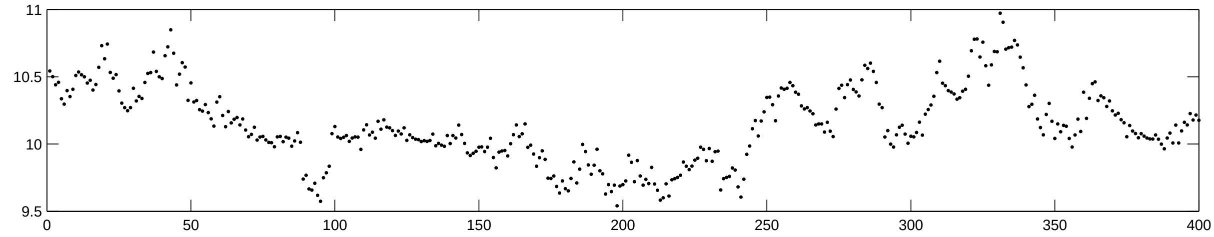


Figure 25.8: Data from an intermittent mean-reverting process. See exercise(241).

25.6 Code

`SLDSforward.m`: SLDS forward
`SLDSbackward.m`: SLDS backward (Expectation Correction)
`mix2mix.m`: Collapse a mixture of Gaussians to a smaller mixture of Gaussians
`SLDSmargGauss.m`: Marginalise an SLDS Gaussian mixture
`logeps.m`: Logarithm with offset to deal with log(0)
`demoSLDStraffic.m`: Demo of Traffic Flow using a switching Linear Dynamical System

25.7 Exercises

Exercise 240. *Consider the setup described in example(108), for which the full SLDS model is given in* `SLDSpricemodel.m`*, following the notation used in* `demoSLDStraffic.m`*. Given the data in the vector* `v` *your task is to fit a prediction model to the data. To do so, find the filtered distribution $p(h(t), s(t)|v_{1:t})$ using a mixture of $I = 2$ components. The prediction of the price at the next day is then*

$$v^{pred}(t+1) = \langle h_1(t) + h_2(t) \rangle_{p(h(t)|v_{1:t})} \tag{25.7.1}$$

where $p(h(t)|v_{1:t}) = \sum_{s_t} p(h(t), s_t|v_{1:t})$.

1. *Compute the mean prediction error*
   `mean_abs_pred_error=mean(abs(vpred(2:end)-v(2:end)))`

2. *Compute the mean naive prediction error*
   `mean_abs_pred_error_naive=mean(abs(v(1:end-1)-v(2:end)))`
   which corresponds to saying that tomorrow's price will be the same as today's.

You might find `SLDSmargGauss.m` *of interest.*

Exercise 241. *The data in fig(25.8) are observed prices from an intermittent mean-reverting process, contained in* `meanrev.mat`*. There are two states $S = 2$. There is a true (latent) price p_t and an observed price v_t (which is plotted). When $s = 1$, the true underlying price reverts back to the mean $m = 10$ with $r = 0.9$. Otherwise the true price follows a random walk*

$$p_t = \begin{cases} r(p_{t-1} - m) + m + \eta_t^p & s_t = 1 \\ p_{t-1} + \eta_t^p & s_t = 2 \end{cases} \tag{25.7.2}$$

where

$$\eta_t^p \sim \begin{cases} \mathcal{N}(\eta_t^p|0, 0.0001) & s_t = 1 \\ \mathcal{N}(\eta_t^p|0, 0.01) & s_t = 2 \end{cases} \tag{25.7.3}$$

The observed price v_t is related to the unknown price p_t by

$$v_t \sim \mathcal{N}(v_t|p_t, 0.001) \tag{25.7.4}$$

It is known that 95% of the time s_{t+1} *is in the same state as at time* t *and that at time* $t = 1$ *either state of* s *is equally likely. Also at* $t = 1$, $p_1 \sim \mathcal{N}(p_1|m, 0.1)$. *Based on this information, and using Gaussian Sum filtering with* $I = 2$ *components (use* `SLDSforward.m`*), what is the probability at time* $t = 280$ *that the dynamics is following a random walk,* $p(s_{280} = 2|v_{1:280})$*? Repeat this computation for smoothing* $p(s_{280} = 2|v_{1:400})$ *based on using Expectation Correction with* $I = J = 2$ *components.*

CHAPTER 26

Distributed Computation

26.1 Introduction

How natural organisms process information is a fascinating subject and one of the grand challenges of science. Whilst this subject is still in its early stages, loosely speaking, there are some generic properties that most such systems are believed to possess: patterns are stored in a set of neurons; recall of patterns is robust to noise; transmission between neurons is of a binary nature and is stochastic; information processing is distributed and highly modular. In this chapter we discuss some of the classical toy models that have been developed as a test bed for analysing such properties[61, 75, 64, 130]. In particular we discuss some classical models from a probabilistic viewpoint.

26.2 Stochastic Hopfield Networks

Hopfield networks are models of biological memory in which a pattern is represented by the activity of a set of V interconnected neurons. The term 'network' here refers to the set of neurons, see fig(26.1), and not the Belief Network representation of distribution of neural states unrolled through time, fig(26.2). At time t neuron i fires $v_i(t) = +1$ or is quiescent $v_i(t) = -1$ (not firing) depending on the states of the neurons at the preceding time $t-1$. Explicitly, neuron i fires depending on the potential

$$a_i(t) \equiv \theta_i + \sum_{j=1}^{V} w_{ij} v_j(t) \tag{26.2.1}$$

where w_{ij} characterizes the efficacy with which neuron j transmits a binary signal to neuron i. The threshold θ_i relates to the neuron's predisposition to firing. Writing the state of the network at time t as $\mathbf{v}(t) \equiv (v_1(t), \ldots, v_V(t))$, the probability that neuron i fires at time $t+1$ is modelled as

$$p(v_i(t+1) = 1|\mathbf{v}(t)) = \sigma_\beta(a_i(t)) \tag{26.2.2}$$

where $\sigma_\beta(x) = 1/(1+e^{-\beta x})$ and β controls the level of stochastic behaviour of the neuron. The probability of being in the quiescent state is given by normalization

$$p(v_i(t+1) = -1|\mathbf{v}(t)) = 1 - p(v_i(t+1) = 1|\mathbf{v}(t)) = 1 - \sigma_\beta(a_i(t)) \tag{26.2.3}$$

These two rules can be compactly written as

$$p(v_i(t+1)|\mathbf{v}(t)) = \sigma_\beta(v_i(t+1)a_i(t)) \tag{26.2.4}$$

which follows directly from $1 - \sigma_\beta(x) = \sigma_\beta(-x)$.

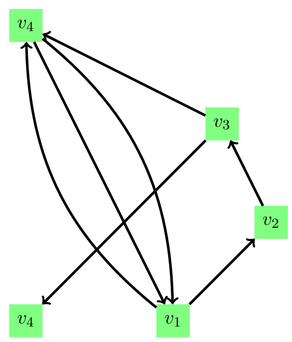


Figure 26.1: A depiction of a Hopfield network (for 5 neurons). The connectivity of the neurons is described by a weight matrix with elements w_{ij}. The graph represents a snapshot of the state of all neurons at time t which simultaneously update as function of the network at the previous time $t-1$.

In the limit $\beta \to \infty$, the neuron updates deterministically

$$v_i(t+1) = \text{sgn}\,(a_i(t)) \tag{26.2.5}$$

In a synchronous Hopfield network, all neurons update independently and simultaneously, so that we can represent the temporal evolution of the neurons as a dynamic Bayes network, fig(26.2)

$$p(\mathbf{v}(t+1)|\mathbf{v}(t)) = \prod_{i=1}^{V} p(v_i(t+1)|\mathbf{v}(t)). \tag{26.2.6}$$

Given this toy description of how neurons update, how can we use the network to do interesting things, for example to store a set of patterns and recall them under some cue. The patterns will be stored in the weights and in the following section we address how to learn suitable parameters w_{ij} and θ_i to learn temporal sequences based on a simple local learning rule.

26.3 Learning Sequences

26.3.1 A Single Sequence

Given a sequence of network states, $\mathcal{V} = \{\mathbf{v}(1), \ldots, \mathbf{v}(T)\}$, we would like the network to 'store' this sequence such that it can be recalled under some cue. That is, if the network is initialized in the correct starting state of the training sequence $\mathbf{v}(t=1)$, the remainder of the training sequence for $t>1$ should be reproduced under the deterministic dynamics equation (26.2.5), without error.

Two classical approaches to learning a temporal sequence are the *Hebb*[1] and *Pseudo Inverse* rules[130]. In both the standard Hebb and PI cases, the thresholds θ_i are usually set to zero.

Standard Hebb rule

$$w_{ij} = \frac{1}{V}\sum_{t=1}^{T-1} v_i(t+1)v_j(t) \tag{26.3.1}$$

[1]Donald Hebb, a neurobiologist actually stated[125]

> Let us assume that the persistence or repetition of a reverberatory activity (or 'trace') tends to induce lasting cellular changes that add to its stability... When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.

This statement is sometimes interpreted to mean that weights are exclusively of the correlation form equation (26.3.1)(see [267] for a discussion). This can severely limit the performance and introduce adverse storage artifacts including local minima[130].

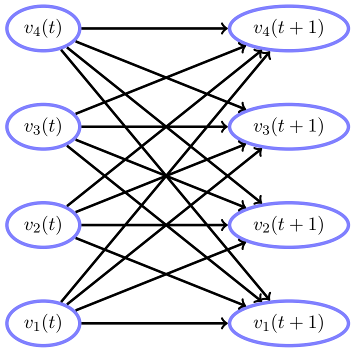


Figure 26.2: A Dynamic Bayesian Network representation of a Hopfield Network. The network operates by simultaneously generating a new set of neuron states according to equation (26.2.6). Equation(26.2.6) defines a Markov transition matrix, modelling the transition probability $\mathbf{v}(t) \to \mathbf{v}(t+1)$ and furthermore imposes the constraint that the neurons are conditionally independent given the previous state of the network.

The Hebb rule can be motivated mathematically by considering

$$\sum_j w_{ij} v_j(t) = \frac{1}{V} \sum_{\tau=1}^{T-1} v_i(\tau+1) \sum_j v_j(\tau) v_j(t) \tag{26.3.2}$$

$$= \frac{1}{V} v_i(t+1) \sum_j v_j^2(t) + \frac{1}{V} \sum_{\tau \neq t}^{T-1} v_i(\tau+1) \sum_j v_j(\tau) v_j(t) \tag{26.3.3}$$

$$= v_i(t+1) + \frac{1}{V} \sum_{\tau \neq t}^{T-1} v_i(\tau+1) \sum_j v_j(\tau) v_j(t) \tag{26.3.4}$$

If the patterns are uncorrelated then the 'interference' term

$$\Omega \equiv \sum_{\tau=1}^{T-1} v_i(\tau+1) \sum_j v_j(\tau) v_j(t)/V \tag{26.3.5}$$

will be relatively small. To see this, we first note that for randomly drawn patterns, the mean of Ω is zero, since $\tau \neq t$ and the patterns are randomly ± 1. The variance is therefore given by

$$\left\langle \Omega^2 \right\rangle = \frac{1}{V^2} \sum_{\tau,\tau' \neq t}^{T-1} \sum_{j,k} \left\langle v_i(\tau+1) v_i(\tau'+1) v_j(\tau) v_j(t) v_k(\tau') v_k(t) \right\rangle \tag{26.3.6}$$

For $j \neq k$, all the terms are independent and contribute zero on average. Therefore

$$\left\langle \Omega^2 \right\rangle = \frac{1}{V^2} \sum_{\tau,\tau'=1}^{T-1} \sum_{j} \left\langle v_i(\tau+1) v_i(\tau'+1) v_j(\tau) v_j(\tau') v_j^2(t) \right\rangle \tag{26.3.7}$$

When $\tau \neq \tau'$ all the terms are independent zero mean and contribute zero. Hence

$$\left\langle \Omega^2 \right\rangle = \frac{1}{V^2} \sum_{\tau \neq t} \sum_{j} \left\langle v_i^2(\tau+1) v_j^2(\tau) v_j^2(t) \right\rangle = \frac{T-1}{V} \tag{26.3.8}$$

Provided that the number of neurons V is significantly larger than the length of the sequence, T, then the average size of the interference will be small. In this case the term $v_i(t+1)$ in equation (26.3.4) dominates, meaning that the sign of $\sum_j w_{ij} v_j(t)$ will be that of $v_i(t+1)$, and the correct pattern sequence recalled. The Hebb rule is capable of storing a random (uncorrelated) temporal sequence of length $0.269V$ time steps[84]. However, the Hebb rule performs poorly for the case of correlated patterns since interference from the other patterns becomes significant[130, 64].

Pseudo Inverse rule

The PI rule finds a matrix $[\mathbf{W}]_{ij} = w_{ij}$ that solves the linear equations

$$\sum_j w_{ij} v_j(t) = v_i(t+1), \qquad t = 1, \ldots, T-1 \tag{26.3.9}$$

Under this condition $\text{sgn}\left(\sum_j w_{ij}v_j(t)\right) = \text{sgn}\left(v_i(t+1)\right) = v_i(t+1)$ so that patterns will be correctly recalled. In matrix notation we require

$$\mathbf{W}\mathbf{V} = \hat{\mathbf{V}} \tag{26.3.10}$$

where

$$[\mathbf{V}]_{it} = v_i(t), \quad t = 1, \ldots, T-1, \qquad \left[\hat{\mathbf{V}}\right]_{it} = v_i(t+1), \quad t = 2, \ldots, T \tag{26.3.11}$$

For $T < V$ the problem is under-determined. One solution is given by the pseudo inverse:

$$\mathbf{W} = \hat{\mathbf{V}}\left(\mathbf{V}^{\mathsf{T}}\mathbf{V}\right)^{-1}\mathbf{V}^{\mathsf{T}} \tag{26.3.12}$$

The Pseudo Inverse (PI) rule can store any sequence of V linearly independent patterns. Whilst attractive compared to the standard Hebb in terms of its ability to store longer correlated sequences, this rule suffers from very small basins of attraction for temporally correlated patterns, see fig(26.3).

The Maximum Likelihood Hebb rule

An alternative to the above classical algorithms is to view this as a problem of pattern storage in the DBN, equation (26.2.6) [19]. First, we need to clarify what we mean by 'store'. Given that we initialize the network in a state $\mathbf{v}(t = 1)$, we wish that the remaining sequence will be generated with high probability. That is, we wish to adjust the network parameters such that the probability

$$p(\mathbf{v}(T), \mathbf{v}(T-1), \ldots, \mathbf{v}(2)|\mathbf{v}(1)) \tag{26.3.13}$$

is maximal[2]. Furthermore, we might hope that the sequence will be recalled with high probability not just when initialized in the correct state but also for states close (in Hamming distance) to the correct initial state $\mathbf{v}(1)$.

Due to the Markov nature of the dynamics, the conditional likelihood is

$$p(\mathbf{v}(T), \mathbf{v}(T-1), \ldots, \mathbf{v}(2)|\mathbf{v}(1)) = \prod_{t=1}^{T-1} p(\mathbf{v}(t+1)|\mathbf{v}(t)) \tag{26.3.14}$$

This is a product of transitions from given states to given states. Since these transition probabilities are known (26.2.6,26.2.2), the conditional likelihood can be easily evaluated. The sequence log (conditional) likelihood is

$$L(\mathbf{w}, \theta) \equiv \log \prod_{t=1}^{T-1} p(\mathbf{v}(t+1)|\mathbf{v}(t)) = \sum_t \log p(\mathbf{v}(t+1)|\mathbf{v}(t)) = \sum_{t=1}^{T-1}\sum_{i=1}^{V} \log \sigma_\beta\left(v_i(t+1)a_i(t)\right) \tag{26.3.15}$$

Our task is then to find weights $\mathbf{w}$ and thresholds θ that maximisise $L(\mathbf{w}, \theta)$. There is no closed form solution and the weights therefore need to be determined numerically. This corresponds to a straightforward computational problem since the log likelihood is a convex function. To show this, we compute the Hessian (neglecting θ for expositional clarity – this does not affect the conclusions):

$$\frac{d^2L}{dw_{ij}dw_{kl}} = -\beta^2 \sum_{t=1}^{T-1} \left(v_i(t+1)v_j(t)\right)\gamma_i(t)(1-\gamma_i(t))v_k(t+1)v_l(t)\delta_{ik} \tag{26.3.16}$$

where we defined

$$\gamma_i(t) \equiv 1 - \sigma_\beta\left(v_i(t+1)a_i(t)\right). \tag{26.3.17}$$

[2] Static patterns can also be considered in this framework as a set of patterns that map to each other.

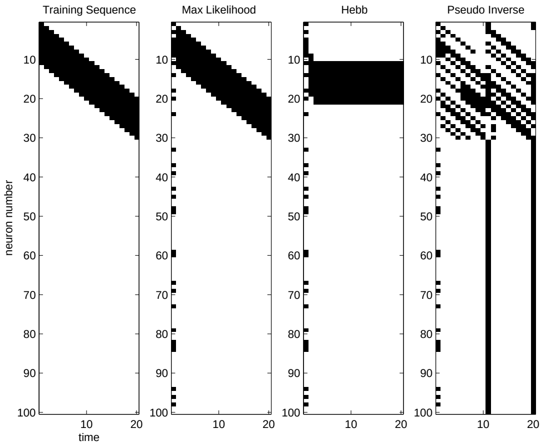


Figure 26.3: Leftmost panel: The temporally highly-correlated training sequence we desire to store. The other panels show the temporal evolution of the network after initialization in the correct starting state but corrupted with 30% noise. During recall, deterministic updates $\beta = \infty$ were used. The Maximum Likelihood rule was trained using 10 batch epochs with $\eta = 0.1$. See also `demoHopfield.m`

It is straightforward to show that the Hessian is negative definite (see exercise(245)) and hence the likelihood has a single global maximum. To increase the likelihood of the sequence, we can use a simple method such as gradient ascent[3]

$$w_{ij}^{new} = w_{ij} + \eta \frac{dL}{dw_{ij}}, \qquad \theta_i^{new} = \theta_i + \eta \frac{dL}{d\theta_i} \tag{26.3.18}$$

where

$$\frac{dL}{dw_{ij}} = \beta \sum_{t=1}^{T-1} \gamma_i(t) v_i(t+1) v_j(t), \qquad \frac{dL}{d\theta_i} = \beta \sum_{t=1}^{T-1} \gamma_i(t) v_i(t+1) \tag{26.3.19}$$

The learning rate η is chosen empirically to be sufficiently small to ensure convergence. The learning rule equation (26.3.19) can be seen as a modified Hebb learning rule, the basic Hebb rule being given when $\gamma_i(t) \equiv 1$. As learning progresses, the $\gamma_i(t)$ will typically tend to values close to either 1 or 0, and hence the learning rule can be seen as asymptotically equivalent to making an update only in the case of disagreement ($a_i(t)$ and $v_i(t+1)$ are of different signs).

This batch training procedure can be readily converted to an online in which an update occurs immediately after the presentation of two consecutive patterns.

Storage Capacity of the ML Hebb rule

The ML Hebb rule is capable of storing a sequence of V linearly independent patterns. To see this, we can form an input-output training set for each neuron i, $\{(\mathbf{v}(t), v_i(t+1)), t = 1, \ldots, T-1\}$. Each neuron has an associated weight vector $\mathbf{w}^i \equiv w_{ij}, j = 1, \ldots, V$, which forms a logistic regressor or, in the limit $\beta = \infty$, a perceptron[130]. For perfect recall of the patterns, we therefore need only that the patterns on the sequence be linearly separable. This will be the case if the patterns are linearly independent, regardless of the outputs $v_i(t+1), t = 1, \ldots, T-1$.

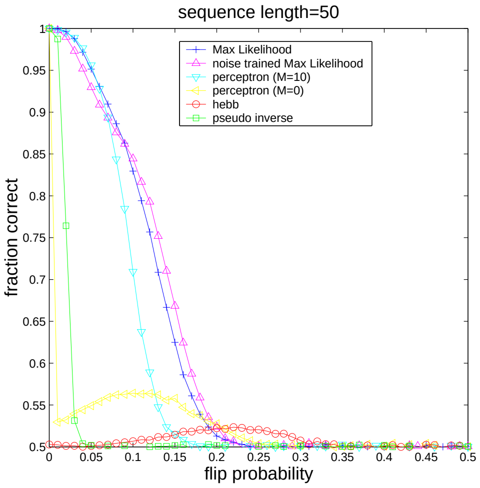
Relation to the Perceptron Rule

In the limit that the activation is large, $|a_i| \gg 1$

$$\gamma_i \approx \begin{cases} 1 & v_i(t+1)a_i < 0 \\ 0 & v_i(t+1)a_i \geq 0 \end{cases} \tag{26.3.20}$$

Provided the activation and desired next output are the same sign, no update is made for neuron i. In this limit, equation (26.3.19) is called the perceptron rule[130, 79]. For an activation a that is close to the

[3]Naturally, one can use more sophisticated methods such as the Newton method, or conjugate gradients. In theoretical neurobiology the emphasis is small gradient style updates since these are deemed to be biologically more plausible.



(a)

Figure 26.4: The fraction of neurons correct for the final state of the network $T = 50$ for a 100 neuron Hopfield network trained to store a length 50 sequence patterns. After initialization in the correct initial state at $t = 1$, the Hopfield network is updated deterministically, with a randomly chosen percentage of the neurons flipped post updating. The correlated sequence of length $T = 50$ was produced by flipping with probability 0.5, 20% of the previous state of the network. A fraction correct value of 1 indicates perfect recall of the final state, and a value of 0.5 indicates a performance no better than random guessing of the final state. For maximum likelihood 50 epochs of training were used with $\eta = 0.02$. During recall, deterministic updates $\beta = \infty$ were used. The results presented are averages over 5000 simulations, resulting in standard errors of the order of the symbol sizes.

decision boundary, a small change can lead to a different sign of the neural firing. To guard against this it is common to include a stability criterion

$$\gamma_i = \begin{cases} 1 & v_i(t+1)a_i < M \\ 0 & v_i(t+1)a_i \geq M \end{cases} \tag{26.3.21}$$

where M is an empirically chosen positive threshold.

Example 109 (Storing a correlated sequence)**.** In fig(26.3) we consider storage of a highly-correlated temporal sequence of length $T = 20$ of 100 neurons using the three learning rules: Hebb, Maximum Likelihood and Pseudo Inverse. The sequence is chosen to be highly correlated, which constitutes a difficult learning task. The thresholds θ_i are set to zero throughout to facilitate comparison. The initial state of the training sequence, corrupted by 30% noise is presented to the trained networks, and we desire that the training sequence will be generated from this initial noisy state. Whilst the Hebb rule is operating in a feasible limit for uncorrelated patterns, the strong correlations in this training sequence entails poor results. The PI rule is capable of storing a sequence of length 100 yet is not robust to perturbations from the correct initial state. The Maximum Likelihood rule performs well after a small amount of training.

Stochastic Interpretation

By straightforward manipulations, the weight update rule in equation (26.3.19) can be written as

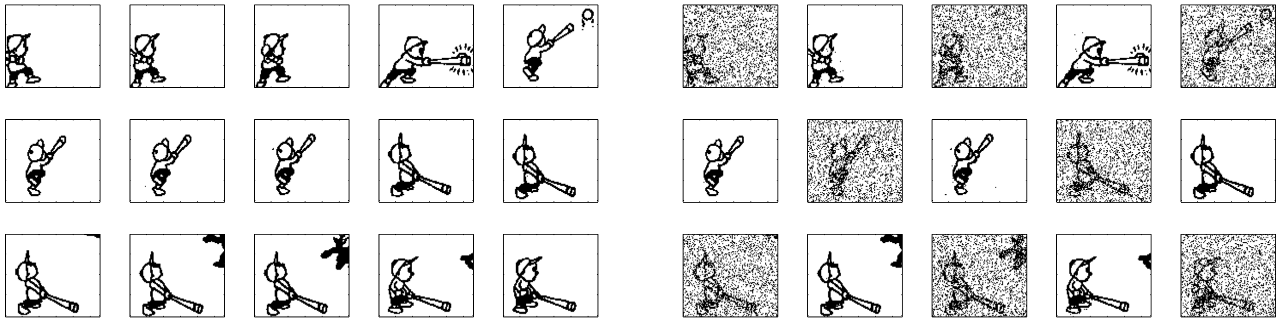
$$\frac{dL}{dw_{ij}} = \sum_{t=1}^{T-1} \frac{1}{2} \left(v_i(t+1) - \langle v_i(t+1) \rangle_{p(v_i(t+1)|a_i(t))} \right) v_j(t) \tag{26.3.22}$$

A stochastic, online learning rule is therefore

$$\Delta w_{ij}(t) = \eta \left(v_i(t+1) - \tilde{v}_i(t+1) \right) v_j(t) \tag{26.3.23}$$

where $\tilde{v}_i(t+1)$ is 1 with probability $\sigma_\beta(a_i(t))$, and -1 otherwise. Provided that the learning rate η is small, this stochastic updating will approximate the learning rule (26.3.18,26.3.19).

Example 110 (Recalling sequences under perpetual noise)**.** We compare the performance of the Maximum Likelihood learning rule (with zero thresholds θ) with the standard Hebb, Pseudo Inverse, and Perceptron rule for learning a single temporal sequence. The network is initialized to a noise corrupted



(a) (b)

Figure 26.5: **(a)**: Original $T = 25$ binary video sequence on a set of $81 \times 111 = 8991$ neurons. **(b)**: The reconstructions beginning from a 20% noise perturbed initial state. Every odd time reconstruction is also randomly perturbed. Despite the high level of noise the basis of attraction of the pattern sequence is very broad and the patterns immediately fall back close to the pattern sequence even after a single timestep.

version of the correct initial state $\mathbf{v}(t = 1)$ from the training sequence. The dynamics is then run (at $\beta = \infty$) for the same number of steps as the length of the training sequence, and the fraction of bits of the recalled final state which are the same as the training sequence final state $\mathbf{v}(T)$ is measured, fig(26.4). At each stage in the dynamics (except the last), the state of the network was corrupted with noise by flipping each neuron state with the specified flip probability. The training sequences are produced by starting from a random initial state, $\mathbf{v}(1)$, and then choosing at random 20% percent of the neurons to flip, each of the chosen neurons being flipped with probability 0.5, giving a random training sequence with a high degree of temporal correlation.

The standard Hebb rule performs relatively poorly, particularly for small flip rates, whilst the other methods perform relatively well, being robust at small flip rates. As the flip rate increases, the pseudo inverse rule becomes unstable, especially for the longer temporal sequence which places more demands on the network. The perceptron rule can perform as well as the Maximum Likelihood rule, although its performance is critically dependent on an appropriate choice of the threshold M. The results for $M = 0$ Perceptron training are poor for small flip rates. An advantage of the Maximum Likelihood rule is that it performs well without the need for fine tuning of parameters. In all cases, batch training was used.

An example for a larger network is given in fig(26.5) which consists of highly correlated sequences. For such short sequences the basin of attraction is very large and the video sequence can be stored robustly.

26.3.2 Multiple Sequences

The previous section detailed how to train a Hopfield network for a single temporal sequence. We now address the learning a set of sequences $\{\mathcal{V}^n, n = 1, \ldots, N\}$. If we assume that the sequences are independent, the log likelihood of a set of sequences is the sum of the individual sequences. The gradient is given by

$$\frac{dL}{dw_{ij}} = \beta \sum_{n=1}^{N} \sum_{t=1}^{T-1} \gamma_i^n(t) v_i^n(t+1) v_j^n(t), \qquad \frac{dL}{d\theta_i} = \beta \sum_{n=1}^{N} \sum_{t=1}^{T-1} \gamma_i^n(t) v_i^n(t+1) \tag{26.3.24}$$

where

$$\gamma_i^n(t) \equiv 1 - \sigma_\beta\left(v_i^n(t+1) a_i^n(t)\right), \qquad a_i^n(t) = \theta_i + \sum_j w_{ij} v_j^n(t) \tag{26.3.25}$$

The log likelihood remains convex since it is the sum of convex functions, so that the standard gradient based learning algorithms can be used here as well.

26.3.3 Boolean Networks

The Hopfield network is one particular parameterisation of the table $p(v_i(t+1) = 1|\mathbf{v}(t))$. However, less constrained parameters may be considered – indeed one could consider the fully unconstrained case in which each neuron i would have an associated 2^V parental states. This exponentially large number of states is impractical and an interesting restriction is to consider that each neuron has only K parents, so that each table contains 2^K entries. Learning the table parameters by Maximum Likelihood is straightforward since the log likelihood is a convex function of the table entries. Hence, for given any sequence (or set of sequences) one may readily find parameters that maximise the sequence reconstruction probability. The Maximum Likelihood method also produces large basins of attraction for the associated stochastic dynamical system. Such models are of potential interest in *Artificial Life* and *Random Boolean networks* in which emergent macroscopic behaviour appears from local update rules[155].

26.3.4 Sequence disambiguation

A limitation of first order networks defined on visible variables alone (such as the Hopfield network) is that the observation transition $p(\mathbf{v}_{t+1}|\mathbf{v}_t = \mathsf{v})$ is the same every time the joint state v is encountered. This means that if the sequence contains a subsequence such as $\mathsf{a}, \mathsf{b}, \mathsf{a}, \mathsf{c}$ this cannot be recalled with high probability since a transitions to different states, depending on time. Whilst one could attempt to resolve this sequence disambiguation problem using a higher order Markov model to account for a longer temporal context, we would lose biological plausibility. Using latent variables is an alternative way to sequence disambiguation. In the Hopfield model the recall capacity can be increased using latent variables by make a sequencing in the joint latent-visible space that is linearly independent, even if the visible variable sequence alone is not. In section(26.4) we discuss a general method that extends dynamic Bayes networks defined on visible variables alone, such as the Hopfield network, to include continuous non-linearly updating latent variables, without requiring additional approximations.

26.4 Tractable Continuous Latent Variable Models

A dynamic Bayes network with latent variables takes the form

$$p(v(1:T), h(1:T)) = p(v(1))p(h(1)|v(1)) \prod_{t=1}^{T-1} p(v(t+1)|v(t), h(t))p(h(t+1)|v(t), v(t+1), h(t)) \quad (26.4.1)$$

As we saw in chapter(23), provided all hidden variables are discrete, inference in these models is straightforward. However, in many physical systems it is more natural to assume continuous $h(t)$. In chapter(24) we saw that one such tractable continuous $h(t)$ model is given by linear Gaussian transitions and emissions - the LDS. Whilst this is useful, we cannot represent non-linear changes in the latent process using an LDS alone. The Switching LDS of chapter(25) is able to model non-linear continuous dynamics (via switching) although we saw that this leads to computational difficulties. For computational reasons we therefore seem limited to either purely discrete h (with no limitation on the discrete transitions) or purely continuous h (but be forced to use simple linear dynamics). Is there a way to have a continuous state with non-linear dynamics for which posterior inference remains tractable? The answer is yes, provided that we assume the hidden transitions are deterministic[14]. When conditioned on the visible variables, this renders the hidden unit distribution trivial. This allows the consideration of rich non-linear dynamics in the hidden space if required.

26.4.1 Deterministic latent variables

Consider a Belief Network defined on a sequence of visible variables $v_{1:T}$. To enrich the model we include additional continuous latent variables $h_{1:T}$ that will follow a non-linear Markov transition. To retain tractability of inference, we constrain the latent dynamics to be deterministic, described by

$$p(h(t+1)|v(t+1), v(t), h(t)) = \delta\left(h(t+1) - f\left(v(t+1), v(t), h(t), \theta_h\right)\right) \quad (26.4.2)$$

Here $\delta(x)$ represents the Dirac delta function for continuous hidden variables. The (possibly non-linear) function f parameterises the CPT. Whilst the restriction to deterministic CPTs appears severe, the model

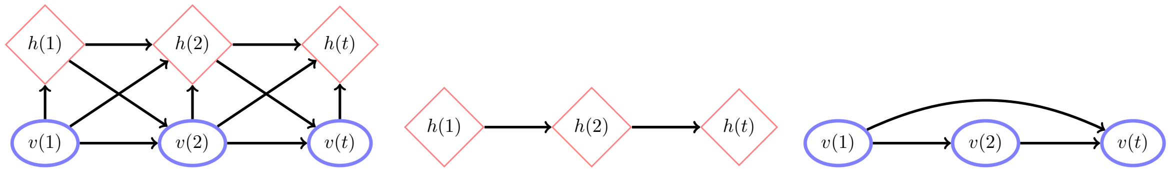


Figure 26.6: **(a)**: A first order Dynamic Bayesian Network with deterministic hidden CPTs (represented by diamonds) that is, the hidden node is certainly in a single state, determined by its parents. **(b)**: Conditioning on the visible variables forms a directed chain in the hidden space which is deterministic. Hidden unit inference can be achieved by forward propagation alone. **(c)**: Integrating out hidden variables gives a cascade style directed visible graph which so that each $v(t)$ depends on all $v(1:t-1)$.

retains some attractive features: The marginal $p(v(1:T))$ is non-Markovian, coupling all the variables in the sequence, see fig(26.6c), whilst hidden unit inference $p(h(1:T)|v(1:T))$ is deterministic, as illustrated in fig(26.6b).

The adjustable parameters of the hidden and visible CPTs are represented by θ_h and θ_v respectively. For learning, the log likelihood of a single training sequence $\mathcal{V}$ is

$$L(\theta_v, \theta_h|\mathcal{V}) = \log p(v(1)|\theta_v) + \sum_{t=1}^{T-1} \log p(v(t+1)|v(t), h(t), \theta_v) \tag{26.4.3}$$

where the hidden unit values are calculated recursively using

$$h(t+1) = f\left(v(t+1), v(t), h(t), \theta_h\right) \tag{26.4.4}$$

To maximise the log likelihood using gradient techniques we need to the derivatives with respect to the model parameters. These can be calculated as follows:

$$\frac{dL}{d\theta_v} = \frac{\partial}{\partial\theta_v}\log p(v(1)|\theta_v) + \sum_{t=1}^{T-1}\frac{\partial}{\partial\theta_v}\log p(v(t+1)|v(t), h(t), \theta_v) \tag{26.4.5}$$

$$\frac{dL}{d\theta_h} = \sum_{t=1}^{T-1}\frac{\partial}{\partial h(t)}\log p(v(t+1)|v(t), h(t), \theta_v)\frac{dh(t)}{d\theta_h} \tag{26.4.6}$$

$$\frac{dh(t)}{d\theta_h} = \frac{\partial f(t)}{\partial\theta_h} + \frac{\partial f(t)}{\partial h(t-1)}\frac{dh(t-1)}{d\theta_h} \tag{26.4.7}$$

where

$$f(t) \equiv f(v(t), v(t-1), h(t-1), \theta_h) \tag{26.4.8}$$

Hence the derivatives can be calculated by deterministic forward propagation of errors alone. The case of training multiple independently generated sequences $\mathcal{V}^n, n = 1, \ldots, N$ is a straightforward extension.

26.4.2 An augmented Hopfield network

To make the deterministic latent variable model more explicit, we consider the case of continuous hidden units and discrete, binary visible units, $v_i(t) \in \{-1, 1\}$. In particular, we restrict attention to the Hopfield model augmented with latent variables that have a simple linear dynamics (see exercise(246) for a non-linear extension):

$$\mathbf{h}(t+1) = 2\sigma(\mathbf{A}\mathbf{h}(t) + \mathbf{B}\mathbf{v}(t)) - \mathbf{1} \quad \text{deterministic latent transition} \tag{26.4.9}$$

$$p(\mathbf{v}(t+1)|\mathbf{v}(t), \mathbf{h}(t)) = \prod_{i=1}^{V}\sigma\left(v_i(t+1)\phi_i(t)\right), \qquad \boldsymbol{\phi}(t) \equiv \mathbf{C}\mathbf{h}(t) + \mathbf{D}\mathbf{v}(t) \tag{26.4.10}$$

This model generalises a recurrent stochastic heteroassociative Hopfield network[130] to include deterministic hidden units dependent on past network states. The parameters of the model are $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$. For gradient based training we require the derivatives with respect to each of these parameters. The derivative of the log likelihood for a generic parameter θ is

$$\frac{d}{d\theta}L = \sum_i \nu_i(t) \frac{d}{d\theta}\phi_i(t) \tag{26.4.11}$$

where

$$\nu_i(t) \equiv \left(1 - \sigma(v_i(t+1)) \sum_j \phi_i(t)\right) v_i(t+1) \tag{26.4.12}$$

This gives (where all indices are summed over the dimensions of the quantities they relate to):

$$\frac{d}{dA_{\alpha\beta}}\phi_i(t) = \sum_j C_{ij} \frac{d}{dA_{\alpha\beta}} h_j(t) \tag{26.4.13}$$

$$\frac{d}{dB_{\alpha\beta}}\phi_i(t) = \sum_j C_{ij} \frac{d}{dB_{\alpha\beta}} h_j(t) \tag{26.4.14}$$

$$\frac{d}{dC_{\alpha\beta}}\phi_i(t) = \delta_{i\alpha} h_\beta(t) \tag{26.4.15}$$

$$\frac{d}{dD_{\alpha\beta}}\phi_i(t) = \delta_{i\alpha} v_\beta(t) \tag{26.4.16}$$

$$\frac{d}{dA_{\alpha\beta}} h_i(t+1) = 2\sigma_i'(t+1) \sum_j A_{ij} \frac{d}{dA_{\alpha\beta}} h_j(t) + \delta_{i\alpha} h_\beta(t) \tag{26.4.17}$$

$$\frac{d}{dB_{\alpha\beta}} h_i(t+1) = 2\sigma_i'(t+1) \sum_j A_{ij} \frac{d}{dB_{\alpha\beta}} h_j(t) + \delta_{i\alpha} v_\beta(t) \tag{26.4.18}$$

$$\sigma_i'(t) \equiv \sigma(h_i(t)) \left(1 - \sigma(h_i(t))\right) \tag{26.4.19}$$

If we assume that $\mathbf{h}(1)$ is a given fixed value (say $\mathbf{0}$), we can compute the derivatives recursively by forward propagation. Gradient based training for this augmented Hopfield network is therefore straightforward to implement. This model extends the power of the original Hopfield model, being capable of resolving ambiguous transitions in sequences such as $\mathtt{a}, \mathtt{b}, \mathtt{a}, \mathtt{c}$, see example(111) and `demoHopfieldLatent.m`. In terms of a dynamic system, the learned network is an attractor with the training sequence as a stable point and demonstrates that such models are capable of learning attractor recurrent networks more powerful than those without hidden units.

Example 111 (Sequence disambiguation)**.** The sequence in fig(26.7a) contains repeated patterns and therefore cannot be reliably recalled with a first order model containing visible variables alone. To deal with this we consider a Hopfield network with 3 visible units and 7 additional hidden units with deterministic (linear) latent dynamics. The model was trained with gradient ascent to maximise the likelihood of the binary sequence in fig(26.7a). As shown in fig(26.7b), the learned network is capable of recalling the sequence correctly, even when initialised in an incorrect state, having no difficulty with the fact that the sequence transitions are ambiguous.

26.5 Neural Models

The tractable deterministic latent variable model introduced in section(26.4) presents an opportunity to extend models such as the Hopfield network to include more biologically realistic processes without losing computational tractability. First we discuss a general framework for learning in a class of neural models[15, 220], this being a special case of the deterministic latent variable models[14] and a generalisation of the spike-response model of theoretical neurobiology[105].

(a)

(b)

Figure 26.7: **(a)**: The training sequence consists of a random set of vectors ($V = 3$) over $T = 10$ time steps. **(b)**: The reconstruction using $H = 7$ hidden units. The initial state $\mathbf{v}(t = 1)$ for the recalled sequence was set to the correct initial training value albeit with one of the values flipped. Note that the method is capable of sequence disambiguation in the sense that the transitions of the form $\mathsf{a}, \mathsf{b}, \ldots, \mathsf{a}, \mathsf{c}$ can be recalled.

26.5.1 Stochastically spiking neurons

We assume that neuron i fires depending on the membrane potential $a_i(t)$ through

$$p(v_i(t+1) = 1|\mathbf{v}(t), \mathbf{h}(t)) = p(v_i(t+1) = 1|a_i(t)) \tag{26.5.1}$$

To be specific, we take throughout

$$p(v_i(t+1) = 1|a_i(t)) = \sigma(a_i(t)) \tag{26.5.2}$$

Here we to define the quiescent state as $v_i(t+1) = 0$, so that

$$p(v_i(t+1)|a_i(t)) = \sigma((2v_i(t+1) - 1)a_i(t)) \tag{26.5.3}$$

The choice of the sigmoid function $\sigma(x)$ is not fundamental and is chosen merely for analytical convenience. The log-likelihood of a sequence of visible states $\mathcal{V}$ is

$$L = \sum_{t=1}^{T-1} \sum_{i=1}^{V} \log \sigma((2v_i(t+1) - 1)a_i(t)) \tag{26.5.4}$$

and the gradient of the log-likelihood is then

$$\frac{dL}{dw_{ij}} = \sum_{t=1}^{T-1} (v_i(t+1) - \sigma(a_i(t))) \frac{da_i(t)}{dw_{ij}} \tag{26.5.5}$$

where we used the fact that $v_i \in \{0, 1\}$. Here w_{ij} are parameters of the membrane potential (see below). We take equation (26.5.5) as common in the following models in which the membrane potential $a_i(t)$ is described with increasing sophistication.

26.5.2 Hopfield membrane potential

As a first step, we show how the Hopfield network training, as described in section(26.3.1), can be recovered as a special case of the above framework. The Hopfield membrane potential is

$$a_i(t) \equiv \sum_{j=1}^{V} w_{ij} v_j(t) - b_i \tag{26.5.6}$$

where w_{ij} characterizes the efficacy of information transmission from neuron j to neuron i, and b_i is a threshold. Applying the Maximum Likelihood framework to this model to learn a temporal sequence $\mathcal{V}$ by adjustment of the parameters w_{ij} (the b_i are fixed for simplicity), we obtain the (batch) learning rule (using $da_i/dw_{ij} = v_j(t)$ in equation (26.5.4))

$$w_{ij}^{new} = w_{ij} + \eta \frac{dL}{dw_{ij}}, \qquad \frac{dL}{dw_{ij}} = \sum_{t=1}^{T-1} (v_i(t+1) - \sigma(a_i(t))) v_j(t), \tag{26.5.7}$$

where the learning rate η is chosen empirically to be sufficiently small to ensure convergence. Equation(26.5.7) matches equation (26.3.19) (which uses the ± 1 encoding).

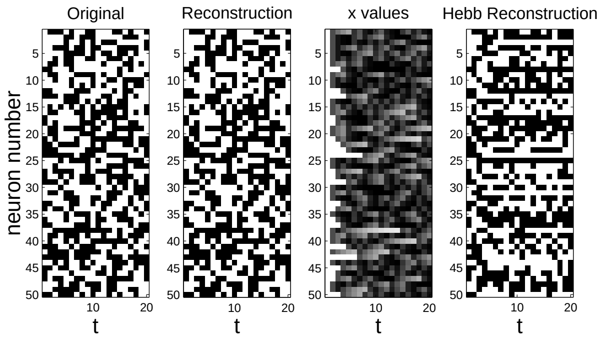


Figure 26.8: Learning with depression : $U = 0.5$, $\tau = 5$, $\delta t = 1$, $\eta = 0.25$. Despite the apparent complexity of the dynamics, learning appropriate neural connection weights is straightforward using Maximum Likelihood. The reconstruction using the standard Hebb rule by contrast is poor[15].

26.5.3 Dynamic synapses

In more realistic synaptic models, neurotransmitter generation depends on a finite rate of cell subcomponent production, and the quantity of vesicles released is affected by the history of firing[1]. Loosely speaking, when a neuron fires it releases a chemical substance from a local reservoir, this reservoir being refilled at a lower rate than the neuron can fire. If the neuron fires continually, its ability to continue firing weakens since the reservoir of release chemical is depleted. This can be accounted for by using a depression mechanism that affects the membrane potential

$$a_i(t) = w_{ij} x_j(t) v_j(t) \tag{26.5.8}$$

for depression factors $x_j(t) \in [0, 1]$. A simple dynamics for these depression factors is[277]

$$x_j(t+1) = x_j(t) + \delta t \left(\frac{1 - x_j(t)}{\tau} - U x_j(t) v_j(t) \right) \tag{26.5.9}$$

where δt, τ, and U represent time scales, recovery times and spiking effect parameters respectively. Note that these depression factor dynamics are exactly of the form of deterministic hidden variables.

It is straightforward to include these dynamic synapses in a principled way using the Maximum Likelihood learning framework. For the Hopfield potential, the learning dynamics is simply given by equations (26.5.5,26.5.9), with

$$\frac{da_i(t)}{dw_{ij}} = x_j(t) v_j(t) \tag{26.5.10}$$

Example 112 (Learning with depression)**.** In fig(26.4) we demonstrate learning a random temporal sequence of 20 time steps for an assembly of 50 neurons with dynamic depressive synapses. After learning w_{ij} the trained network is initialised in the first state of the training sequence. The remaining states of the sequence were then correctly recalled by iteration of the learned model. The corresponding generated factors $x_i(t)$ are also plotted. For comparison, we plot the results of using the dynamics having set the w_{ij} using the temporal Hebb rule, equation (26.3.1). The poor performance of the correlation based Hebb rule demonstrates the necessity, in general, to couple a dynamical system with an appropriate learning mechanism.

26.5.4 Leaky Integrate and Fire models

Leaky integrate and fire models move a step further towards biological realism in which the membrane potential increments if it receives an excitatory stimulus ($w_{ij} > 0$), and decrements if it receives an inhibitory stimulus ($w_{ij} < 0$). After firing, the membrane potential is reset to a low value below the firing threshold, and thereafter steadily increases to a resting level (see for example [61, 105]). A model that incorporates such effects is

$$a_i(t) = \left(\alpha a_i(t-1) + \sum_j w_{ij} v_j(t) + \theta^{rest} (1 - \alpha) \right) (1 - v_i(t-1)) + v_i(t-1)\theta^{fired} \tag{26.5.11}$$

Since $v_i \in \{0, 1\}$, if neuron i fires at time $t-1$ the potential is reset to θ^{fired} at time t. Similarly, with no synaptic input, the potential equilibrates to θ^{rest} with time constant $-1/\log\alpha$[15].

Despite the increase in complexity of the membrane potential over the Hopfield case, deriving appropriate learning dynamics for this new system is straightforward since, as before, the hidden variables (here the membrane potentials) update in a deterministic fashion. The potential derivatives are

$$\frac{da_i(t)}{dw_{ij}} = (1 - v_i(t-1))\left(\alpha\frac{da_i(t-1)}{dw_{ij}} + v_j(t)\right) \tag{26.5.12}$$

By initialising the derivative $\frac{da_i(t=1)}{dw_{ij}} = 0$, equations (26.5.5,26.5.11,26.5.12) define a first order recursion for the gradient which can be used to adapt w_{ij} in the usual manner $w_{ij} \leftarrow w_{ij} + \eta dL/dw_{ij}$. We could also apply synaptic dynamics to this case by replacing the term $v_j(t)$ in equation (26.5.12) by $x_j(t)v_j(t)$.

Although a detailed discussion of the properties of the neuronal responses for networks trained in this way is beyond the scope of these notes, an interesting consequence of the learning rule equation (26.5.12) is a spike time dependent learning window in qualitative agreement with experimental results[220, 183].

In summary, provided one deals with deterministic latent dynamics, essentially arbitrarily complex spatio-temporal patterns may potentially be learned, and generated under cued retrieval, for very complex neural dynamics. The spike-response model [105] can be seen as a special case of the deterministic latent variable model in which the latent variables have been explicitly integrated out.

26.6 Code

`demoHopfield.m`: Demo of Hopfield sequence learning
`HebbML.m`: Gradient ascent training of a set of sequences using Max Likelihood
`HopfieldHiddenNL.m`: Hopfield network with additional non-linear latent variables
`demoHopfieldLatent.m`: demo of Hopfield net with deterministic latent variables
`HopfieldHiddenLikNL.m`: Hopfield Net with hidden variables sequence likelihood

26.7 Exercises

Exercise 242. *Consider a very large Hopfield network $V \gg 1$ used to store a single temporal sequence of length $\mathbf{v}(1:T)$, $T \ll V$. In this case the weight matrix w may be difficult to store. Explain how to justify the assumption*

$$w_{ij} = \sum_{t=1}^{T-1} u_i(t)v_i(t+1)v_j(t) \tag{26.7.1}$$

where $u_i(t)$ are the dual parameters and derive an update rule for the dual parameters u.

Exercise 243. *A Hopfield network is used to store a raw uncompressed binary video sequence. Each image in the sequence contains 10^6 binary pixels. At a rate of 10 frames per second, how many hours of video can 10^6 neurons store?*

Exercise 244. *Derive the update equation (26.3.22).*

Exercise 245. *Show that the Hessian equation (26.3.16) is negative definite. That is*

$$\sum_{i,j,k,l} x_{ij}x_{kl}\frac{d^2L}{dw_{ij}dw_{kl}} \leq 0 \tag{26.7.2}$$

for any $x \neq 0$.

Exercise 246. *For the augmented Hopfield network of section(26.4.2),with latent dynamics*

$$h_i(t+1) = 2\sigma\left(\sum_j A_{ij}h_j(t) + B_{ij}v_j(t)\right) - 1 \tag{26.7.3}$$

derive the derivative recursions described in section(26.4.2).

Part V

Approximate Inference

CHAPTER 27

Sampling

27.1 Introduction

Sampling concerns drawing realisations $\mathsf{x}^1, \ldots, \mathsf{x}^L$ of a variable x from a distribution $p(x)$. For a discrete variable x, in the limit of a large number of samples, the fraction of samples in state x tends to $p(x = \mathsf{x})$. That is,

$$\lim_{L\to\infty} \frac{1}{L}\sum_{l=1}^{L} \mathbb{I}\left[\mathsf{x}^l = \mathsf{x}\right] = p(x = \mathsf{x}) \tag{27.1.1}$$

In the continuous case, one can consider a small region Δ such that the probability that the samples occupy Δ tends to the integral of $p(x)$ over Δ. In other words, the relative frequency $x \in \Delta$ tends to $\int_{x\in\Delta} p(x)$. Given a finite set of samples, one can then approximate expectations using

$$\langle f(x)\rangle_{p(x)} \approx \frac{1}{L}\sum_{l=1}^{L} f(\mathsf{x}^l) \tag{27.1.2}$$

This approximation holds for both discrete and continuous variables. In general sampling is used to approximate averages when direct enumeration or integration is computationally intractable.

Drawing samples from high-dimensional distributions is generally difficult and few guarantees exist to ensure that in a practical timeframe the samples produced are representative enough such that expectations can be approximated accurately. There are many different sampling algorithms, all of which 'work in principle', but each 'working in practice' only when the distribution satisfies particular properties[109]. Before we develop schemes for multi-variate distributions, we consider the univariate case.

27.1.1 Univariate sampling

In the following, we assume that a random number generator exists which is able to produce a value uniformly at random from the unit interval $[0, 1]$. We will make use of this uniform random number generator to draw samples from non-uniform distributions.

Discrete case

Consider the one dimensional discrete distribution $p(x)$ where $\text{dom}(x) = \{1, 2, 3\}$, with

$$p(x) = \begin{cases} 0.6 & x = 1 \\ 0.1 & x = 2 \\ 0.3 & x = 3 \end{cases} \tag{27.1.3}$$

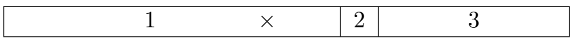


Figure 27.1: A representation of the discrete distribution equation (27.1.3). The unit interval from 0 to 1 is partitioned in parts whose lengths are equal to 0.6, 0.1 and 0.3.

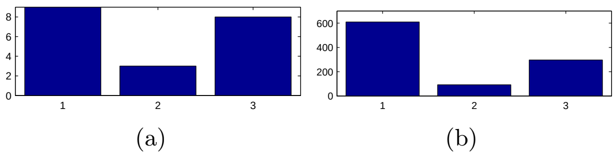


Figure 27.2: Histograms of the samples from the three state distribution $p(x) = \{0.6, 0.1, 0.3\}$. **(a)**: 20 samples. **(b)**: 1000 samples. As the number of samples increases, the relative frequency of the samples tends to the distribution $p(x)$.

This represents a partitioning of the unit interval $[0, 1]$ in which the interval $[0, 0.6]$ has been labelled as state 1, $[0.6, 0.7]$ as state 2, and $[0.7, 1.0]$ as state 3, fig(27.1). If we were to drop a point $\times$ anywhere at random, uniformly in the interval $[0, 1]$, the chance that $\times$ would land in interval 1 is 0.6, and the chance that it would be in interval 2 is 0.1 and similarly, for interval 3, 0.3. This therefore defines for us a valid sampling procedure for discrete one-dimensional distributions as described in algorithm(25).

In our example, we have $(c_0, c_1, c_2, c_3) = (0, 0.6, 0.7, 1)$. We then draw a sample uniformly from $[0, 1]$, say $u = 0.66$. Then the sampled state would be state 2, since this is in the interval $(c_1, c_2]$.

Sampling from a discrete univariate distribution is straightforward since computing the cumulant takes only $O(K)$ steps for a K state discrete variable.

Continuous case

In the following we assume that a method exists to generate samples from the uniform distribution $U(x|[0, 1])$. Intuitively, the generalisation of the discrete case to the continuous case is clear. First we calculate the cumulant density function

$$C(y) = \int_{-\infty}^{y} p(x)dx \tag{27.1.4}$$

Then we sample u uniformly from $[0, 1]$, and obtain the corresponding sample x by solving $C(x) = u \Rightarrow x = C^{-1}(u)$. Formally, therefore, sampling of a continuous univariate variable is straightforward provided we can compute the integral of the corresponding probability density function.

For special distributions, such as Gaussians, numerically efficient alternative procedures exist, usually based on co-ordinate transformations, see exercise(247).

27.1.2 Multi-variate sampling

One way to generalise the one dimensional discrete case to a higher dimensional distribution $p(x_1, \ldots, x_n)$ is to translate this into an equivalent one-dimensional distribution. This can be achieved by enumerating

Algorithm 25 Sampling from a univariate discrete distribution p with K states.

Label the K states as $i = 1, \ldots, K$, with associated probabilities p_i.
Calculate the *cumulant*

$$c_i = \sum_{j \le i} p_j$$

and set $c_0 = 0$.
Draw a value u uniformly at random from the unit interval $[0, 1]$.
Find that i for which $c_{i-1} < u \le c_i$.
Return state i as a sample from p.

all the possible joint states $(x_1, \ldots, x_n)$, giving each a unique integer i from 1 to the total number of states, and constructing a univariate distribution with probability $p(i) = p(\mathsf{x})$ for i corresponding to the multivariate state x. This then transforms the multi-dimensional distribution into an equivalent one-dimensional distribution, and sampling can be achieved as before. In general, of course, this procedure is impractical since the number of states will grow exponentially with the number of variables $x_1, \ldots, x_n$.

An alternative exact approach would be to capitalise on the relation

$$p(x_1, x_2) = p(x_2|x_1)p(x_1) \tag{27.1.5}$$

We can sample from the joint distribution $p(x_1, x_2)$ by first sampling a state for x_1 from the one-dimensional $p(x_1)$, and then, with x_1 clamped to this state, sampling a state for x_2 from the one-dimensional $p(x_2|x_1)$. It is clear how to generalise this to more variables by using a cascade decomposition:

$$p(x_1, \ldots, x_n) = p(x_n|x_{n-1}, \ldots, x_1)p(x_{n-1}|x_{n-2}, \ldots, x_1) \ldots p(x_2|x_1)p(x_1) \tag{27.1.6}$$

However, in order to apply this technique, we need to know the conditionals $p(x_i|x_{i-1}, \ldots, x_1)$. Unless these are explicitly given we need to compute these from the joint distribution $p(x_1, \ldots, x_n)$. Such conditionals will, in general, require the summation over an exponential number of states and, except for small n, generally also be impractical. For Belief Networks, however, the conditionals are specified so that this technique becomes practical, as we discuss in section(27.2).

Drawing samples from a multi-variate distribution is in general therefore a complex task and one seeks to exploit any structural properties of the distribution to make this computationally more feasible. A common approach is to seek a co-ordinate transform such that in the transformed system the distribution is factored into a product of lower dimensional distributions. A classic example of this is sampling from a multi-variate Gaussian, which can be reduced to sampling from a set of univariate Gaussians by a suitable coordinate transformation, as discussed in example(113).

Example 113 (Sampling from a multi-variate Gaussian)**.** Our interest is to draw a sample from the multi-variate Gaussian $p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\mathbf{m}, \mathbf{S})$. For a general covariance matrix $\mathbf{S}$, $p(\mathbf{x})$ does not factorise into a product of univariate distributions. However, consider the transformation

$$\mathbf{y} = \mathbf{C}^{-1}(\mathbf{x} - \mathbf{m}) \tag{27.1.7}$$

where $\mathbf{C}$ is chosen so that $\mathbf{C}\mathbf{C}^{\mathsf{T}} = \mathbf{S}$. Since this is a linear transformation, $\mathbf{y}$ is also Gaussian distributed with mean

$$\langle \mathbf{y} \rangle = \left\langle \mathbf{C}^{-1}(\mathbf{x} - \mathbf{m}) \right\rangle_{p(\mathbf{x})} = \mathbf{C}^{-1}\left(\langle \mathbf{x} \rangle_{p(\mathbf{x})} - \mathbf{m} \right) = \mathbf{C}^{-1}(\mathbf{m} - \mathbf{m}) = \mathbf{0} \tag{27.1.8}$$

Since the mean of $\mathbf{y}$ is zero, the covariance is given by

$$\left\langle \mathbf{y}\mathbf{y}^{\mathsf{T}} \right\rangle_{p(\mathbf{x})} = \mathbf{C}^{-1}\left\langle (\mathbf{x} - \mathbf{m})(\mathbf{x} - \mathbf{m})^{\mathsf{T}} \right\rangle_{p(\mathbf{x})} \mathbf{C}^{-\mathsf{T}} = \mathbf{C}^{-1}\mathbf{S}\mathbf{C}^{-\mathsf{T}} = \mathbf{C}^{-1}\mathbf{C}\mathbf{C}^{\mathsf{T}}\mathbf{C}^{-\mathsf{T}} = \mathbf{I} \tag{27.1.9}$$

Hence

$$p(\mathbf{y}) = \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{I}) = \prod_i \mathcal{N}(y_i|0, 1) = \prod_i p(y_i) \tag{27.1.10}$$

Hence a sample from $\mathbf{y}$ can be obtained by independently drawing a sample from each of the univariate 0 mean unit variance Gaussians. Given a sample for $\mathbf{y}$, a sample for $\mathbf{x}$ is obtained using

$$\mathbf{x} = \mathbf{C}\mathbf{y} + \mathbf{m} \tag{27.1.11}$$

Drawing samples from a univariate Gaussian is a well-studied topic, with a popular method being the Box-Muller technique, exercise(247).

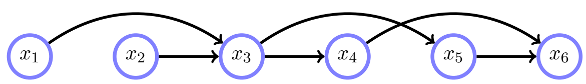


Figure 27.3: An ancestral Belief Network without any evidential variables. To sample from this distribution, we draw a sample from variable 1, and then variables, 2,...,6 in order.

27.2 Ancestral Sampling

Belief Networks take the general form:

$$p(x) = \prod_i p(x_i|\text{pa}(x_i)) \tag{27.2.1}$$

where each of the conditional distributions $p(x_i|\text{pa}(x_i))$ is known. Provided that no variables are evidential, we can sample from this distribution in a straightforward manner. For convenience, we first rename the variable indices so that parent variables always come before their children (*ancestral ordering*), for example (see fig(27.3))

$$p(x_1, \ldots, x_6) = p(x_1)p(x_2)p(x_3|x_1, x_2)p(x_4|x_3)p(x_5|x_3)p(x_6|x_4, x_5) \tag{27.2.2}$$

One can sample first from those nodes that do not have any parents (here, x_1 and x_2). Given these values, one can then sample x_3, and then x_4 and x_5 and finally x_6. Despite the presence of loops in the graph, see fig(27.3), such a *forward sampling* procedure is straightforward. This procedure holds for both discrete and continuous variables.

If one attempted to carry out an exact inference scheme using moralisation and triangulation, in more complex multiply connected graphs, cliques can become very large. However, regardless of the loop structure, ancestral sampling is straightforward.

Ancestral or Forward sampling is a case of *perfect sampling* (also termed *exact sampling*) since each sample is indeed drawn from the required distribution. This is in contrast to Markov Chain Monte Carlo methods sections(27.3,27.4) for which the samples are representative only in the limit of a large number of iterations.

27.2.1 Dealing with evidence

How can we sample from a distribution in which certain variables $x_\mathcal{E}$ are clamped to evidential states? Formally we need to sample from

$$p(x_{\backslash\mathcal{E}}|x_\mathcal{E}) = \frac{p(x_{\backslash\mathcal{E}}, x_\mathcal{E})}{p(x_\mathcal{E})} \tag{27.2.3}$$

If an evidential variable x_i has no parents, then one can simply set the variable into this state and continue forward sampling as before. For example, to compute a sample from $p(x_1, x_3, x_4, x_5, x_6|x_2)$ defined in equation (27.2.2), fig(27.3), one simply clamps the x_2 into its evidential state and continues forward sampling. The reason this is straightforward is that conditioning on x_2 merely defines a new distribution on a subset of the variables, for which the distribution is immediately known.

On the other hand, consider sampling from $p(x_1, x_2, x_3, x_4, x_6|x_5)$. Using Bayes' rule, we have

$$p(x_1, x_2, x_3, x_4, x_6|x_5) = \frac{p(x_1)p(x_2)p(x_3|x_1, x_2)p(x_4|x_3)p(x_5|x_3)p(x_6|x_4, x_5)}{\sum_{x_1,x_2,x_3,x_4,x_6} p(x_1)p(x_2)p(x_3|x_1, x_2)p(x_4|x_3)p(x_5|x_3)p(x_6|x_4, x_5)} \tag{27.2.4}$$

however, the conditioning on x_5 means that the structure of the distribution on the non-evidential variables changes – for example x_4 and x_6 become coupled. One could attempt to work out an equivalent new forward sampling structure, (see exercise(248)) although generally this will be as complex as running an exact inference approach.

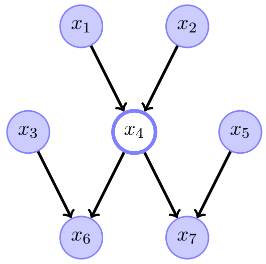


Figure 27.4: The Markov blanket of x_4. To draw a sample from $p(x_4|x_{\setminus 4})$ we clamp x_1, x_2, x_3, x_5, x_7 into their evidential states and draw a sample from $p(x_4|x_1, x_2)p(x_6|x_3, x_4)p(x_7|x_4, x_5)/Z$ where Z is a normalisation constant.

An alternative is to proceed with forward sampling from the non-evidential distribution, and then discard any samples which do not match the evidential states. This can be extremely inefficient, and is not generally recommended since the probability that a sample from $p(x)$ will be consistent with the evidence is roughly $O\left(1/\prod_i \dim x_i^e\right)$ where $\dim x_i^e$ is the number of states of evidential variable i. In principle one can ease this effect by discarding the sample as soon as any variable state is inconsistent with the evidence. Nevertheless, the number of re-starts required to obtain a valid sample would on average be very large.

27.2.2 Perfect sampling for a Markov Network

For a Markov network we can draw exact samples by forming an equivalent directed representation of the graph, see section(6.8), and subsequently using ancestral sampling on this directed graph. This is achieved by first choosing a root clique and then consistently orienting edges away from this clique. An exact sample can then be drawn from the Markov network by first sampling from the root clique and then recursively from the children of this clique. See `potsample.m`, `JTsample.m` and `demoJTreeSample.m`.

27.3 Gibbs Sampling

The inefficiency of methods such as ancestral sampling under evidence, motivates alternative techniques. A important and widespread technique is Gibbs sampling which is generally straightforward to implement.

No evidence

Assume we have a joint sample state x^1 from the multivariate distribution $p(x)$. We then consider a particular variable, x_i. Using Bayes' rule we may write

$$p(x) = p(x_i|x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)p(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) \tag{27.3.1}$$

Since we assume that someone has already provided us with a sample x^1, from which we can read off the 'parental' state $x_1^1, \ldots, x_{i-1}^1, x_{i+1}^1, \ldots, x_n^1$, we can then draw a sample x_i^2 from

$$p(x_i|\mathrm{x}_1^1, \ldots, \mathrm{x}_{i-1}^1, \mathrm{x}_{i+1}^1, \ldots, \mathrm{x}_n^1) \equiv p(x_i|\mathrm{x}_{\setminus i}) \tag{27.3.2}$$

We assume this distribution is easy to sample from since it is univariate. We call this new joint sample (in which only x_i has been updated) $\mathrm{x}^2 = \left(\mathrm{x}_1^1, \ldots, \mathrm{x}_{i-1}^1, \mathrm{x}_i^2, \mathrm{x}_{i+1}^1, \ldots, \mathrm{x}_n^1\right)$. One then selects another variable x_j to sample and, by continuing this procedure, generates a set $\mathrm{x}^1, \ldots, \mathrm{x}^L$ of samples in which each x^{l+1} differs from x^l in only a single component. The reason this is valid sampling scheme is outlined in section(27.3.1).

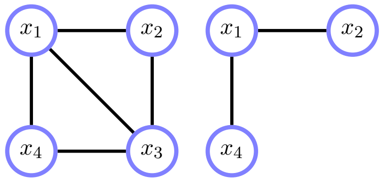
For a Belief Network, the conditional $p(x_i|x_{\setminus i})$ is defined by the *Markov blanket* of x_i:

$$p(x_i|x_{\setminus i}) = \frac{1}{Z}p(x_i|\mathrm{pa}\,(x_i)) \prod_{j \in \mathrm{ch}(i)} p(x_j|\mathrm{pa}\,(x_j)) \tag{27.3.3}$$

see for example, fig(27.4). The normalisation constant is straightforward to work out from the requirement:

$$Z = \sum_{x_i} p(x_i|\mathrm{pa}\,(x_i)) \prod_{j \in \mathrm{ch}(i)} p(x_j|\mathrm{pa}\,(x_j)) \tag{27.3.4}$$

In the case of a continuous variable x_i the summation above is replaced with integration.



(a) (b)

Figure 27.5: **(a)**: A toy 'intractable' distribution. Gibbs sampling by conditioning on all variables except one leads to a simple univariate conditional distribution. **(b)**: Conditioning on x_3 yields a new distribution that is singly-connected, for which exact sampling is straightforward.

Evidence

Evidence is easy to deal with in the Gibbs sampling procedure. One clamps for all samples the evidential variables into their evidential states. There is also no need to sample for these variables, since their states are known.

27.3.1 Gibbs sampling as a Markov chain

In Gibbs sampling we have a sample of the joint variables x^l at stage l. Based on this we produce a new joint sample x^{l+1}. This means that we can write Gibbs sampling as a procedure that draws from

$$x^{l+1} \sim q(x^{l+1}|x^l) \tag{27.3.5}$$

for some distribution $q(x^{l+1}|x^l)$. If we choose the variable to update, x_i, at random from a distribution $q(i)$, then Gibbs sampling corresponds to drawing samples using the Markov transition

$$q(x^{l+1}|x^l) = \sum_i q(x^{l+1}|x^l, i)q(i), \qquad q(x^{l+1}|x^l, i) = p(x_i^{l+1}|x_{\setminus i}^l)\prod_{j\neq i}\delta\left(x_j^{l+1} - x_j^l\right) \tag{27.3.6}$$

with $q(i) > 0$. Our interest is to show that the stationary distribution of $q(x'|x)$ is $p(x)$. We carry this out assuming x is continuous – the discrete case is analogous:

$$\int_x q(x'|x)p(x) = \sum_i q(i)\int_x q(x'|x_{\setminus i})p(x) \tag{27.3.7}$$

$$= \sum_i q(i)\int_x \prod_{j\neq i}\delta\left(x'_j - x_j\right)p(x'_i|x_{\setminus i})p(x_i, x_{\setminus i}) \tag{27.3.8}$$

$$= \sum_i q(i)\int_{x_i} p(x'_i|x'_{\setminus i})p(x_i, x'_{\setminus i}) \tag{27.3.9}$$

$$= \sum_i q(i)p(x'_i|x'_{\setminus i})p(x'_{\setminus i}) = \sum_i q(i)p(x') = p(x') \tag{27.3.10}$$

Hence, as long as we continue to draw samples according to the distribution $q(x'|x)$, in the limit of a large number of samples we will tend ultimately to draw samples from $p(x)$. Any distribution $q(i) > 0$ suffices so visiting all variables equally often is also a valid choice.

27.3.2 Structured Gibbs sampling

One can extend Gibbs sampling by using conditioning to reveal a tractable distribution on the remaining variables. For example, consider the simple distribution, fig(27.5a)

$$p(x_1, x_2, x_3, x_4) = \phi(x_1, x_2)\phi(x_2, x_3)\phi(x_3, x_4)\phi(x_4, x_1)\phi(x_1, x_3) \tag{27.3.11}$$

In single-site Gibbs sampling we would condition on three of the four variables, and sample from the remaining variable. For example

$$p(x_1|\mathsf{x}_2, \mathsf{x}_3, \mathsf{x}_4) \propto \phi(x_1, \mathsf{x}_2)\phi(\mathsf{x}_4, x_1)\phi(x_1, \mathsf{x}_3) \tag{27.3.12}$$

However, we may used more limited conditioning as long as the conditioned distribution is easy to sample from. In the case of equation (27.3.11) we can condition on x_3 alone to give

$$p(x_1, x_2, x_4|\mathsf{x}_3) \propto \phi(x_1, x_2)\phi(x_2, \mathsf{x}_3)\phi(\mathsf{x}_3, x_4)\phi(x_4, x_1)\phi(x_1, \mathsf{x}_3) \tag{27.3.13}$$

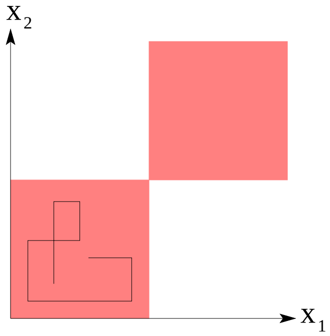


Figure 27.6: A two dimensional distribution for which Gibbs sampling fails. The distribution has mass only in the shaded quadrants. Gibbs sampling proceeds from the l^{th} sample state $(\mathrm{x}_1^l, \mathrm{x}_2^l)$ and then sampling from $p(x_2|\mathrm{x}_1^l)$, which we write $(\mathrm{x}_1^{l+1}, \mathrm{x}_2^{l+1})$ where $\mathrm{x}_1^{l+1} = \mathrm{x}_1^l$. One then continues with a sample from $p(x_1|x_2 = \mathrm{x}_2^{l+1})$, *etc.* If we start in the lower left quadrant and proceed this way, the upper right region is never explored. This is a case where the sampler is non-ergodic.

This can be written as a modified distribution, fig(27.5b)

$$p(x_1, x_2, x_4|\mathrm{x}_3) \propto \phi'(x_1, x_2)\phi'(x_4, x_1) \tag{27.3.14}$$

As a distribution on x_1, x_2, x_4 this is a singly-connected linear chain from which samples can be drawn exactly. A simple approach is compute the normalisation constant by any of the standard techniques, for example, using the Factor Graph method. One may then convert this undirected linear chain to a directed graph, and use ancestral sampling. These operations are linear in the number of variables in the conditioned distribution. Alternatively, one may form a junction tree from a set of potentials, choose a root and then form a set chain by reabsorption on the junction tree. Ancestral sampling can then be performed on the resulting oriented clique tree. This is the approach taken in `GibbsSample.m`.

In the above example one can also reveal a tractable distribution by conditioning on x_1,

$$p(x_3, x_2, x_4|\mathrm{x}_1) \propto \phi(\mathrm{x}_1, x_2)\phi(x_2, x_3)\phi(x_3, x_4)\phi(x_4, \mathrm{x}_1)\phi(\mathrm{x}_1, x_3) \tag{27.3.15}$$

and then draw a sample of x_2, x_3, x_4 from this distribution. A valid sampling procedure is then to draw a sample x_1, x_2, x_4 from equation (27.3.13) and then a sample x_3, x_2, x_4 from equation (27.3.15). These two steps are then iterated. Note that x_2 and x_4 are not constrained to be equal to their values in the previous sample. This procedure is generally to be preferred to the single-site Gibbs updating since the samples are less correlated from one sample to the next.

See `demoGibbsSample.m` and `GibbsSample.m` for a comparison of unstructured and structured sampling from a set of potentials.

27.3.3 Remarks

If the initial sample x^1 is in a part of the state space that is very unlikely then it will take some time for the samples to become representative, as only a single component of x is updated at each iteration. This motivates a so-called *burn in* stage in which the initial samples are discarded.

In single site Gibbs sampling there will be a high degree of correlation in any two successive samples, since only one variable(in the single-site updating version) is updated at each stage. An ideal 'perfect' sampling scheme would draw each x 'at random' from $p(x)$ – clearly, in general, two such perfect samples will not possess the same degree of correlation as those from Gibbs sampling. This motivates *subsampling* in which, say, every 10^{th}, sample $\mathrm{x}^K, \mathrm{x}^{K+10}, \mathrm{x}^{K+20}, \ldots$, is taken, and the rest discarded.

Due to its simplicity, Gibbs sampling is one of the most popular sampling methods and is particularly convenient when applied to Belief Networks, due to the Markov blanket property[1]. Gibbs sampling is a special case of the MCMC framework and, as with all MCMC methods, one should bear in mind that convergence can be a major issue – that is, answering questions such as 'how many samples are needed to be reasonably sure that my sample estimate $p(x_5)$ is accurate?' is, to a large extent, an unknown. Despite mathematical results for special cases, general rules of thumb and awareness on behalf of the user are required to monitor the efficiency of the sampling.

[1]The BUGS package `www.mrc-bsu.cam.ac.uk/bugs` is general purpose software for sampling from Belief Networks.

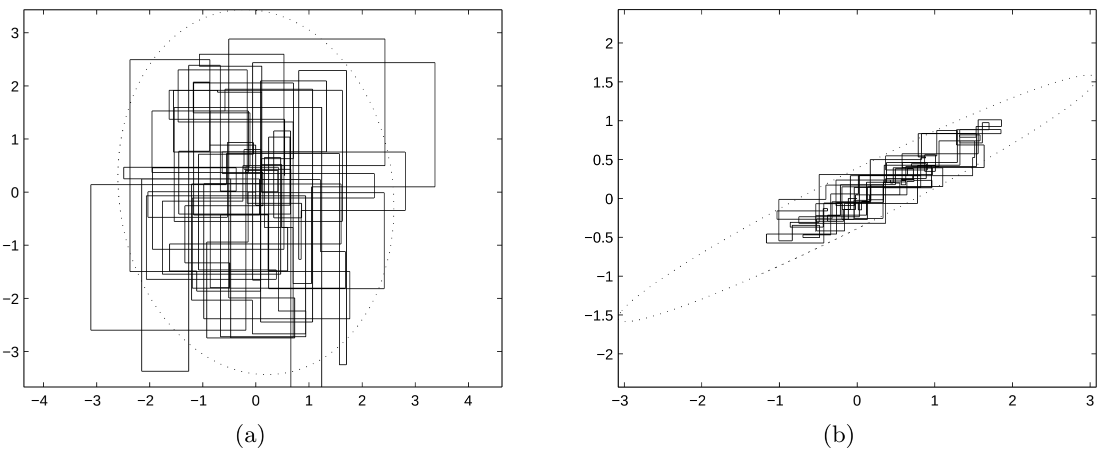


Figure 27.7: Two hundred Gibbs samples for a two dimensional Gaussian. At each stage only a single component is updated. **(a)**: For a Gaussian with low correlation, Gibbs sampling can move through the likely regions effectively. **(b)**: For a strongly correlated Gaussian, Gibbs sampling is less effective and does not rapidly explore the likely regions, see `demoGibbsGauss.m`.

Gibbs sampling assumes that we can move throughout the space effectively by only single co-ordinate updates. We also require that every state can be visited infinitely often (the sampler is *ergodic*). In fig(27.6), we show a case in which the two dimensional continuous distribution has mass only in the lower left and upper right regions. In that case, if we start in the lower left region, we will always remain there, and never explore the upper right region. This problem occurs when two regions which are not connected by a 'probable' Gibbs path.

Gibbs sampling becomes a perfect sampler when the distribution is factorised – that is the variables are independent. This suggests that in general Gibbs sampling will be less effective when variables are strongly correlated. For example, if we consider Gibbs sampling from a strongly correlated two variable Gaussian distribution, then updates will move very slowly in space, fig(27.7).

27.4 Markov Chain Monte Carlo (MCMC)

We assume we have a distribution in the form

$$p(x) = \frac{1}{Z} p^*(x) \tag{27.4.1}$$

where Z is the normalisation constant of the distribution and $p^*(x)$ is the unnormalised distribution. We assume we are able to evaluate $p^*(x = \mathsf{x})$, for any state x, but not Z, since $Z = \int_x p^*(x)$ is an intractable high dimensional summation/integration.

The idea in MCMC sampling is to sample, not directly from $p(x)$, but from a different distribution. However, in the limit of a large number of samples, effectively the samples will be from $p(x)$. To achieve this we forward sample from a Markov transition whose stationary distribution is equal to $p(x)$.

27.4.1 Markov Chains

Consider the conditional distribution $q(x^{l+1}|x^l)$. If we are given an initial sample x^1, then we can recursively generate samples $x^1, x^2, \ldots, x^L$. After a long time $L \gg 1$, the samples are from the *stationary* distribution $q_\infty(x)$ which is defined as (for a continuous variable)

$$q_\infty(x') = \int_x q(x'|x) q_\infty(x) \tag{27.4.2}$$

The condition for a discrete variable is analogous on replacing integration with summation. The idea in MCMC is, for a given distribution $p(x)$, to find a transition $q(x'|x)$ which has $p(x)$ as its stationary distribution. If we can do so, then we can draw samples from the Markov Chain by forward sampling and take these as samples from $p(x)$.

Note that for every distribution $p(x)$ there will be more than one transition $q(x'|x)$ with $p(x)$ as the equilibrium distribution. This is why there are very many different MCMC sampling methods, each with different characteristics and varying suitability for the particular distribution at hand.

Detailed balance

How do we construct a transition $q(x'|x)$ with given $p(x)$ as its stationary distribution? This problem can be simplified if we consider special transitions that satisfy the *detailed balance* condition. If we are given the marginal distribution $p(x)$, the detailed balance condition for a transition kernel q is

$$\frac{q(x'|x)}{q(x|x')} = \frac{p(x')}{p(x)}, \qquad \forall x, x' \tag{27.4.3}$$

Under this we see

$$\int_x q(x'|x)p(x) = \int_x q(x|x')p(x') = p(x') \tag{27.4.4}$$

so that $p(x)$ is the stationary distribution of $q(x'|x)$. The detailed balance requirement can make the process of constructing a suitable transition easier since only the relative value of $p(x')$ to $p(x)$ is required in equation (27.4.3), and not the absolute value of $p(x)$ or $p(x')$.

27.4.2 Metropolis-Hastings sampling

Consider the following transition

$$q(x'|x) = \tilde{q}(x'|x)f(x', x) + \delta(x' - x)\left(1 - \int_{x''} \tilde{q}(x''|x)f(x'', x)\right) \tag{27.4.5}$$

where $\tilde{q}(x'|x)$ is a so-called *proposal distribution* and $0 < f(x', x) \leq 1$ a positive function. This defines a valid distribution $q(x'|x)$ since it is non-negative and

$$\int_{x'} q(x'|x) = \int_{x'} \tilde{q}(x'|x)f(x', x) + 1 - \int_{x''} \tilde{q}(x''|x)f(x'', x) = 1 \tag{27.4.6}$$

Our interest is to set $f(x, x')$ such that the stationary distribution of $q(x'|x)$ is equal to $p(x)$ (for any proposal $\tilde{q}(x'|x)$).

In the case $x' \neq x$, then

$$q(x'|x) = \tilde{q}(x'|x)f(x', x) \tag{27.4.7}$$

For detailed balance with respect to $p(x)$, we need

$$\frac{q(x'|x)}{q(x|x')} = \frac{p(x')}{p(x)} = \frac{\tilde{q}(x'|x)f(x', x)}{\tilde{q}(x|x')f(x, x')} \tag{27.4.8}$$

Now consider the *Metropolis-Hastings acceptance function*

$$f(x', x) = \min\left(1, \frac{\tilde{q}(x|x')p(x')}{\tilde{q}(x'|x)p(x)}\right) = \min\left(1, \frac{\tilde{q}(x|x')p^*(x')}{\tilde{q}(x'|x)p^*(x)}\right) \tag{27.4.9}$$

which is defined for all x, x'. We show that this function satisfies equation (27.4.8):

$$f(x', x)\tilde{q}(x'|x)p(x) = \min\left(\tilde{q}(x'|x)p(x), \tilde{q}(x|x')p(x')\right) \tag{27.4.10}$$

$$= \min\left(\tilde{q}(x|x')p(x'), \tilde{q}(x'|x)p(x)\right) = f(x, x')\tilde{q}(x|x')p(x') \tag{27.4.11}$$

Algorithm 26 Metropolis-Hastings MCMC sampling.

Choose a starting point x^1.
for $i = 2$ to L **do**
  Draw a candidate sample x^{cand} from the proposal $\tilde{q}(x'|x^{l-1})$.
  Let $a = \frac{\tilde{q}(x^{l-1}|x^{cand})p(x^{cand})}{\tilde{q}(x^{cand}|x^{l-1})p(x^{l-1})}$
  if $a \geq 1$ **then** $x^l = x^{cand}$ ▷ Accept the candidate
  else
    draw a random value u uniformly from the unit interval $[0, 1]$.
    if $u < a$ **then** $x^l = x^{cand}$ ▷ Accept the candidate
    else
      $x^l = x^{l-1}$ ▷ Reject the candidate
    end if
  end if
end for

Hence the function $f(x', x)$ as defined above ensures that $\tilde{q}(x'|x)$ satisfies the detailed balance condition, equation (27.4.8). Using the above we have

$$\int_x q(x'|x)p(x) = \int_x \tilde{q}(x'|x)f(x', x)p(x) + p(x')\left(1 - \int_{x''} \tilde{q}(x''|x')f(x'', x')\right) \tag{27.4.12}$$

Using equation (27.4.8), this is

$$\int_x q(x'|x)p(x) = \int_x \tilde{q}(x|x')f(x, x')p(x') + p(x')\left(1 - \int_{x''} \tilde{q}(x''|x')f(x'', x')\right) = p(x') \tag{27.4.13}$$

Hence the stationary distribution of $q(x'|x)$ is $p(x)$.

How do we sample from $q(x'|x)$? Equation(27.4.5) can be interpreted as a mixture of two distributions, one proportional to $\tilde{q}(x'|x)f(x', x)$ and the other $\delta(x'|x)$ with mixture coefficient $1 - \int_{x''} \tilde{q}(x''|x)f(x'', x)$. To draw a sample from this, we draw a sample from $\tilde{q}(x'|x)$ and accept this with probability $f(x', x)$. Since drawing from $\tilde{q}(x'|x)$ and accepting are performed independently, the probability of accepting the drawn candidate is the product of these probabilities, namely $\tilde{q}(x'|x)f(x', x)$. Otherwise the candidate is rejected and we take the sample $x' = x$. Using the properties of the acceptance function, equation (27.4.9), the following is equivalent to deciding on accepting/rejecting the candidate. If

$$\tilde{q}(x|x')p^*(x') > \tilde{q}(x'|x)p^*(x) \tag{27.4.14}$$

we accept the sample from $\tilde{q}(x'|x)$. Otherwise we accept the sample x' from $q(x'|x)$ with probability $\tilde{q}(x|x')p^*(x')/\tilde{q}(x'|x)p^*(x)$. If we reject the candidate we take $x' = x$.

Note that if the candidate x' is rejected, we take the original x as the new sample. Hence at each iteration of the algorithm produces a sample – either a copy of the current sample, or the candidate sample. A rough rule of thumb is to choose a proposal distribution for which the acceptance rate is between 50% and 85%[102].

Gaussian Proposal distribution

A common proposal distribution for multivariate $\mathbf{x}$ is

$$\tilde{q}(\mathbf{x}'|\mathbf{x}) \propto e^{-\frac{1}{2\sigma^2}(\mathbf{x}'-\mathbf{x})^2} \tag{27.4.15}$$

Then $\tilde{q}(\mathbf{x}'|\mathbf{x}) = \tilde{q}(\mathbf{x}|\mathbf{x}')$, and the acceptance criterion is simply

$$f(\mathbf{x}', \mathbf{x}) = \min\left(1, \frac{p^*(\mathbf{x}')}{p^*(\mathbf{x})}\right) \tag{27.4.16}$$

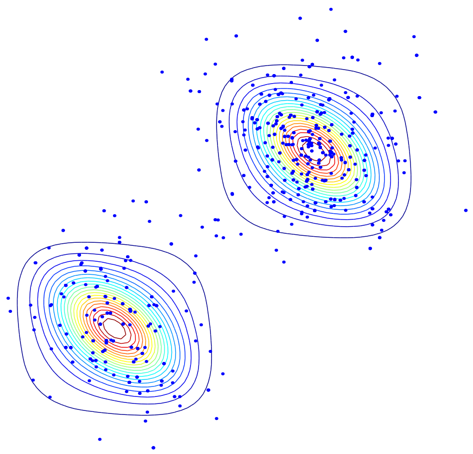


Figure 27.8: Metropolis-Hastings samples from a bi-variate distribution $p(x_1, x_2)$ using a proposal $\tilde{q}(\mathbf{x}'|\mathbf{x}) = \mathcal{N}(\mathbf{x}'|\mathbf{x}, \mathbf{I})$. We also plot the iso-probability contours of p. Although $p(x)$ is multimodal, the dimensionality is low enough and the modes sufficiently close such that a simple Gaussian proposal distribution is able to bridge the two modes. In higher dimensions, such multi-modality is more problematic. See `demoMetropolis.m`

If the unnormalised probability of the candidate state is higher than the current state, we therefore accept the candidate. Otherwise, if the unnormalised probability of the candidate state is lower than the current state, we accept the candidate only with probability $p^*(\mathbf{x}')/p^*(\mathbf{x})$. If the candidate is rejected, the new sample is taken to be a copy of the previous sample $\mathbf{x}$. See fig(27.8) for a demonstration.

In high dimensions it is unlikely that a random candidate sampled from a Gaussian will result in a candidate probability higher than the current value, exercise(250). Because of this, only very small jumps (σ^2 small) are likely to be accepted. This limits the speed at which we explore the space $\mathbf{x}$.

Note that sampling is different from finding the optimum. Provided $\mathbf{x}'$ has a higher probability than $\mathbf{x}$, we accept $\mathbf{x}'$. However, we also accept (with a specified acceptance probability) candidates that have also a *lower* probability than the current sample.

27.5 Auxiliary Variable Methods

A practical concern in MCMC methods is ensuring that one moves effectively through the significant probability regions of the distribution. For methods such as Metropolis-Hastings with local proposal distributions (local in the sense they are unlikely to propose a candidate far from the current sample), if the target distribution has isolated islands of high density, then the likelihood that we would be able to move from one island to the other is very small. If we attempt to make the proposal less local by using one with a high variance the chance then of landing at random on a high density island is remote. The aim of auxiliary variable methods is to aid the exploration of the distribution in general and in particular cases to encourage transition between isolated regions by using extra dimensions to provide a bridge over low density regions.

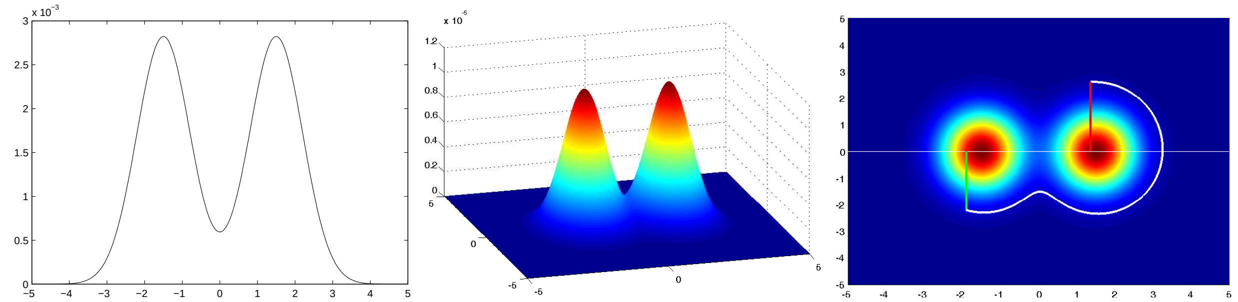
Consider drawing samples from $p(x)$ where x is a high-dimensional vector. For an auxiliary variable y we introduce a distribution $p(y|x)$, to form the joint distribution

$$p(x, y) = p(y|x)p(x) \tag{27.5.1}$$

If we draw samples (x^l, y^l) from this joint distribution then a valid set of samples from $p(x)$ is given by taking the x^l alone. If we sampled x directly from $p(x)$ and then y from $p(y|x)$, this is pointless since y would not affect the x sampling procedure. In order for this to be useful, therefore, the auxiliary variable must influence how we sample x. There are different procedures for using y to affect the x-sampling, some of the most common of which are the Hybrid MCMC and Slice-sampling methods.

27.5.1 Hybrid Monte Carlo

Hybrid MC is a method for continuous systems that aims to make non-trivial jumps in the samples and, in so doing, to jump potentially from one mode to another. Let's define the difficult distribution from



(a) (b) (c)

Figure 27.9: Hybrid Monte Carlo. **(a)**: Multi-modal distribution $p(x)$ for which we desire samples. **(b)**: HMC forms the joint distibution $p(x)p(y)$ where $p(y)$ is Gaussian. **(c)**: Starting from the point x, we first draw a y from the Gaussian $p(y)$, giving a point (x, y), green line. Then we use Hamiltonian dynamics (white line) to traverse the distribution at roughly constant energy for a fixed number of steps, giving x', y'. We accept this point if $H(x', y') > H(x, y')$ and make the new sample x' (red line). Otherwise this candidate is accepted with probability $\exp(H(x', y') - H(x, y'))$. If rejected the new sample x' is taken as a copy of x.

which we wish to sample as

$$p(\mathbf{x}) = \frac{1}{Z_{\mathbf{x}}} e^{H_{\mathbf{x}}(\mathbf{x})} \tag{27.5.2}$$

for some given 'Hamiltonian' $H_{\mathbf{x}}(\mathbf{x})$ (this is just a potential). We then define another, 'easy' distribution

$$p(\mathbf{y}) = \frac{1}{Z_{\mathbf{y}}} e^{H_{\mathbf{y}}(\mathbf{y})} \tag{27.5.3}$$

so that the joint distribution is given by

$$p(\mathbf{x}, \mathbf{y}) = p(\mathbf{x})p(\mathbf{y}) = \frac{1}{Z} e^{H_{\mathbf{x}}(\mathbf{x}) + H_{\mathbf{y}}(\mathbf{y})} = \frac{1}{Z} e^{H(\mathbf{x}, \mathbf{y})} \tag{27.5.4}$$

The HMC algorithm then alternates between drawing from $p(\mathbf{y})$ and a so-called *dynamic* step. In the dynamic step, we draw a sample from $p(\mathbf{x}, \mathbf{y})$. In the standard form of the algorithm, a multi-dimensional Gaussian is chosen for the auxiliary distribution with $\dim \mathbf{y} = \dim \mathbf{x}$, so that

$$H_{\mathbf{y}}(\mathbf{y}) = -\frac{1}{2} \mathbf{y}^{\mathsf{T}} \mathbf{y} \tag{27.5.5}$$

Sampling from $p(\mathbf{y})$ is then straightforward. In the dynamic step, the idea is to go from one point of the space $\mathbf{x}, \mathbf{y}$ to a new point $\mathbf{x}', \mathbf{y}'$ that is a non-trivial distance from $\mathbf{x}, \mathbf{y}$ and which will be accepted with a high probability. The candidate $(\mathbf{x}', \mathbf{y}')$ will have a good chance to be accepted if $H(\mathbf{x}', \mathbf{y}')$ is close to $H(\mathbf{x}, \mathbf{y})$ – this can be achieved by following a contour of equal 'energy' H.

Hamiltonian dynamics

Consider a Hamiltonian $H(\mathbf{x}, \mathbf{y})$ and that we wish to make an update $\mathbf{x}' = \mathbf{x} + \Delta\mathbf{x}$, $\mathbf{y}' = \mathbf{y} + \Delta\mathbf{y}$ for small $\Delta\mathbf{x}$ and $\Delta\mathbf{y}$ such that the energy is conserved,

$$H(\mathbf{x}', \mathbf{y}') = H(\mathbf{x}, \mathbf{y}) \tag{27.5.6}$$

We can satisfy this (up to first order) by considering the Taylor expansion

$$\begin{aligned} H(\mathbf{x}', \mathbf{y}') &= H(\mathbf{x} + \Delta\mathbf{x}, \mathbf{y} + \Delta\mathbf{y}) \\ &\approx H(\mathbf{x}) + \Delta\mathbf{x}^{\mathsf{T}} \nabla_{\mathbf{x}} H(\mathbf{x}, \mathbf{y}) + H(\mathbf{y}) + \Delta\mathbf{y}^{\mathsf{T}} \nabla_{\mathbf{y}} H(\mathbf{x}, \mathbf{y}) + O\left(|\Delta\mathbf{x}|^2\right) + O\left(|\Delta\mathbf{y}|^2\right) \end{aligned} \tag{27.5.7}$$

Algorithm 27 Hybrid Monte Carlo sampling

```
Start from x
for i = 1 to L do
    Draw a new sample y from p(y).
    Choose a random (forwards or backwards) trajectory direction.
    Starting from x, y, Hamiltonian dynamics for a fixed number of time steps giving a candidate x', y'.
    Accept x', y' if H(x', y') > H(x, y), otherwise accept it with probability exp(H(x', y') − H(x, y)).
    If rejected, we take the sample as x, y.
end for
```

Energy conservation, up to first order, therefore requires

$$\Delta\mathbf{x}^{\mathsf{T}}\nabla_{\mathbf{x}}H(\mathbf{x},\mathbf{y}) + \Delta\mathbf{y}^{\mathsf{T}}\nabla_{\mathbf{y}}H(\mathbf{x},\mathbf{y}) = 0 \tag{27.5.8}$$

This is a single scalar requirement, and there are therefore many different solutions for $\Delta\mathbf{x}$ and $\Delta\mathbf{y}$ that satisfy this single condition. It is customary to use Hamiltonian dynamics, which correspond to the setting:

$$\Delta\mathbf{x} = \epsilon\nabla_{\mathbf{y}}H(\mathbf{x},\mathbf{y}) \qquad \Delta\mathbf{y} = -\epsilon\nabla_{\mathbf{x}}H(\mathbf{x},\mathbf{y}) \tag{27.5.9}$$

where ϵ is a small value to ensure that the Taylor expansion is accurate. Hence

$$\mathbf{x}(t+1) = \mathbf{x}(t) + \epsilon\nabla_{\mathbf{y}}H_{\mathbf{y}}(\mathbf{y}) \qquad \mathbf{y}(t+1) = \mathbf{y}(t) - \epsilon\nabla_{\mathbf{x}}H_{\mathbf{x}}(\mathbf{x}) \tag{27.5.10}$$

For the HMC method, $H(\mathbf{x},\mathbf{y}) = H_{\mathbf{x}}(\mathbf{x}) + H_{\mathbf{y}}(\mathbf{y})$, so that $\nabla_{\mathbf{x}}H(\mathbf{x},\mathbf{y}) = \nabla_{\mathbf{x}}H_{\mathbf{x}}(\mathbf{x})$ and $\nabla_{\mathbf{y}}H(\mathbf{x},\mathbf{y}) = \nabla_{\mathbf{y}}H_{\mathbf{y}}(\mathbf{y})$ For the Gaussian case, $\nabla_{\mathbf{y}}H_{\mathbf{y}}(\mathbf{y}) = -\mathbf{y}$ so that

$$\mathbf{x}(t+1) = \mathbf{x}(t) - \epsilon\mathbf{y} \qquad \mathbf{y}(t+1) = \mathbf{y}(t) - \epsilon\nabla_{\mathbf{x}}H(\mathbf{x}) \tag{27.5.11}$$

There are specific ways to implement the Hamiltonian dynamics called *Leapfrog discretisation* that are more accurate than the simple time-discretisation used above, and we refer the reader to [203] for details.

In order to make a symmetric proposal distribution, at the start of the dynamic step, we choose $\epsilon = +\epsilon_0$ or $\epsilon = -\epsilon_0$ uniformly. This means that there is the same chance that we go back to the point $\mathbf{x}, \mathbf{y}$ starting from $\mathbf{x}', \mathbf{y}'$, as vice versa. We can then follow the Hamiltonian dynamics for many time steps (usually of the order of several hundred) to reach a candidate point $\mathbf{x}', \mathbf{y}'$. If the Hamiltonian dynamics was numerically well implemented, $H(\mathbf{x}', \mathbf{y}')$ will have roughly the same value as $H(\mathbf{x}, \mathbf{y})$. We then do a Metropolis step, and accept the point $\mathbf{x}', \mathbf{y}'$ if $H(\mathbf{x}', \mathbf{y}') > H(\mathbf{x}, \mathbf{y})$ and otherwise accept it with probability $\exp(H(\mathbf{x}', \mathbf{y}') - H(\mathbf{x}, \mathbf{y}))$. If rejected, we take the initial point $\mathbf{x}, \mathbf{y}$ as the sample. Combined with the $p(\mathbf{y})$ sample step, we then have the general procedure as described in algorithm(27).

In HMC we use not just the potential $H(\mathbf{x})$ to define candidate samples, but the gradient of $H(\mathbf{x})$ as well. An intuitive reason for the success of the algorithm is that it is less myopic than straightforward Metropolis, since the use of the gradient enables the algorithm to feel its way to other regions of high probability by contouring paths in the augmented space. One can also view the auxiliary variables as momentum variables – it is as if the sample has now a momentum which can carry it through the low-density $\mathbf{x}$-regions. Provided this momentum is high enough, we can escape local regions of significant probability, see fig(27.9).

27.5.2 Swendson-Wang

Originally, the SW method was introduced to alleviate the problems encountered in sampling from Ising Models close to their critical temperature[265]. At this point large islands of same-state variables form so that strong correlations appear in the distribution – the scenario under which, for example, Gibbs sampling is not well suited. The method has been generalised [87] to other models, although here we outline the procedure for the Ising model only, referring the reader to more specialised text for the extensions [37].

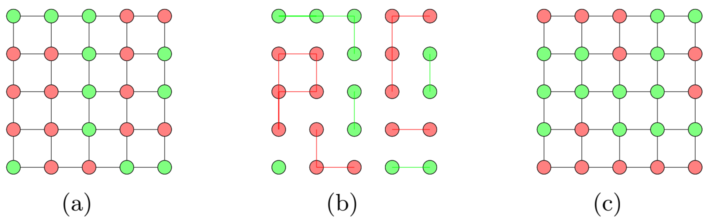


Figure 27.10: Swendson-Wang updating for $p(x) \propto \prod_{i\sim j} \exp \beta \mathbb{I}[x_i = x_j]$. **(a)**: Current sample of states (here on a nearest neighbour lattice). **(b)**: Like coloured neighbours are bonded together with probability $1-e^{-\beta}$, forming clusters of variables. **(c)**: Each cluster is given a random colour, forming the new sample.

See also [196] for the use of auxiliary variables in perfect sampling.

The Ising model with no external fields is defined on variables $x = (x_1, \ldots, x_n)$, $x_i \in \{0, 1\}$ takes the form

$$p(x) = \frac{1}{Z} \prod_{i\sim j} e^{\beta \mathbb{I}[x_i = x_j]} \tag{27.5.12}$$

which means that this is a pairwise Markov network with a potential contribution e^{β} if neighbouring nodes i and j are in the same state, and a contribution 1 otherwise. We assume that $\beta > 0$ which encourages neighbours to be in the same state. The lattice based neighbourhood structure makes this difficult to sample from, and especially when $\beta \approx 0.9$ which encourages large scale islands of same-state variables to form.

The aim is to remove the problematic terms $e^{\beta \mathbb{I}[x_i = x_j]}$ by the use of the auxiliary variables, making the conditional $p(x|y)$ easy to sample from. This is given by

$$p(x|y) \propto p(y|x)p(x) \propto p(y|x) \prod_{i\sim j} e^{\beta \mathbb{I}[x_i = x_j]} \tag{27.5.13}$$

Using $p(y|x)$ we can cancel the terms $e^{\beta \mathbb{I}[x_i = x_j]}$ by setting

$$p(y|x) = \prod_{i\sim j} p(y_{ij}|x_i, x_j) = \prod_{i\sim j} \frac{1}{z_{ij}} \mathbb{I}\left[0 < y_{ij} < e^{\beta \mathbb{I}[x_i = x_j]}\right] \tag{27.5.14}$$

where $\mathbb{I}\left[0 < y_{ij} < e^{\beta \mathbb{I}[x_i = x_j]}\right]$ denotes a uniform distribution between 0 and $e^{\beta \mathbb{I}[x_i = x_j]}$; z_{ij} is the normalisation constant $z_{ij} = e^{\beta \mathbb{I}[x_i = x_j]}$. Hence

$$p(x|y) \propto p(y|x)p(x) \tag{27.5.15}$$

$$\propto \prod_{i\sim j} \frac{1}{e^{\beta \mathbb{I}[x_i = x_j]}} \mathbb{I}\left[0 < y_{ij} < e^{\beta \mathbb{I}[x_i = x_j]}\right] e^{\beta \mathbb{I}[x_i = x_j]} \tag{27.5.16}$$

$$\propto \prod_{i\sim j} \mathbb{I}\left[0 < y_{ij} < e^{\beta \mathbb{I}[x_i = x_j]}\right] \tag{27.5.17}$$

Let's assume that we have a sample y_{ij}. If $y_{ij} > 1$, then to draw a sample from $p(x|y)$, we must have $1 < e^{\beta \mathbb{I}[x_i = x_j]}$, which means that x_i and x_j are in the same state. Otherwise, if $y_{ij} < 1$, then this places no constraint on x. Hence, wherever $y_{ij} > 1$, we bond x_i and x_j to be in the same state.

To sample from the bond variables $p(y_{ij}|x_i, x_j)$ consider first when x_i and x_j are in the same state. Then $p(y_{ij}|x_i = x_j) = U\left(y_{ij}|\left[0, e^{\beta}\right]\right)$, similarly $p(y_{ij}|x_i \neq x_j) = U\left(y_{ij}|[0, 1]\right)$. A bond will occur if $y_{ij} > 1$, which occurs with probability

$$p(y_{ij} > 1|x_i, x_j) = \int_{y_{ij}=1}^{\infty} \frac{1}{z_{ij}} \mathbb{I}\left[0 < y_{ij} < e^{\beta \mathbb{I}[x_i = x_j]}\right] = \frac{e^{\beta} - 1}{e^{\beta}} = 1 - e^{-\beta} \tag{27.5.18}$$

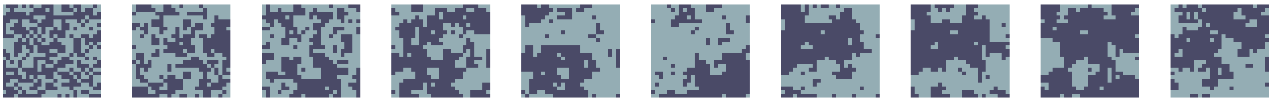


Figure 27.11: Ten successive samples from a 25 × 25 Ising model $p(x) \propto \exp\left(\sum_{i\sim j} \beta \mathbb{I}[x_i = x_j]\right)$, with $\beta = 0.88$, close to the critical temperature. The Swendson-Wang procedure is used. Starting in a random initial configuration, the samples quickly move away from this initial state, with the characteristic long-range correlations of the variables seen close to the critical temperature.

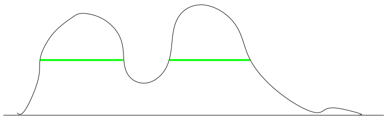


Figure 27.12: The full slice for a given y. Ideally slice sampling would draw an x sample from anywhere on the full slice (green). In general this is intractable for a complex distribution and a local approximate slice is formed instead, see fig(27.13).

Hence, if $x_i = x_j$, we bind x_i and x_j to be in the same state with probability $1 - e^{-\beta}$. On the other hand if x_i and x_j are in different states, y_{ij} is uniformly distributed between 0 and 1.

After doing this for all the x_i and x_j pairs, we will end up with a graph in which we have clusters of like-state bonded variables. The algorithm simply chooses a random state for each cluster – that is, with probability 0.5 all variables in the cluster are in state 1.

The algorithm then does the following, see fig(27.10):

1. If $x_i = x_j$, we bond variables x_i and x_j with probability $1 - e^{-\beta}$. Repeat this for all variables.
2. For each cluster formed from the above, set the state of the cluster uniformly at random.
3. Repeat (1,2) above.

This technique has found application in spatial statistics, particularly image restoration[133].

27.5.3 Slice Sampling

Slice sampling[205] is an auxiliary variable technique that aims to overcome some of the difficulties in choosing an appropriate 'length scale' in methods such as Metropolis sampling. The brief discussion here follows that presented in [180] and [42]. We want to draw samples from $p(x) = \frac{1}{Z}p^*(x)$ where the normalisation constant Z is unknown. By introducing the auxiliary variable y and defining the distribution

$$p(x, y) = \begin{cases} 1/Z & \text{for } 0 \leq y \leq p^*(x) \\ 0 & \text{otherwise} \end{cases} \tag{27.5.19}$$

we have

$$\int p(x, y)dy = \int_0^{p^*(x)} \frac{1}{Z}dy = \frac{1}{Z}p^*(x) = p(x) \tag{27.5.20}$$

which shows that the marginal of $p(x, y)$ over y is equal to the distribution we wish to draw samples from. Hence if we draw samples from $p(x, y)$, we can ignore the y samples and we will have a valid sampling scheme for $p(x)$.

To draw from $p(x, y)$ we use Gibbs sampling, first drawing from $p(y|x)$ and then from $p(x|y)$. Drawing a sample from $p(y|x)$ means that we draw a value y from the uniform distribution $U(y|\,[0, p^*(x)])$.

Given a sample y, one then draws a sample x from $p(x|y)$. Using $p(x|y) \propto p(x, y)$ we see that $p(x|y)$ is the distribution over x such that $p^*(x) > y$:

$$p(x|y) \propto \mathbb{I}[p^*(x) > y] \tag{27.5.21}$$

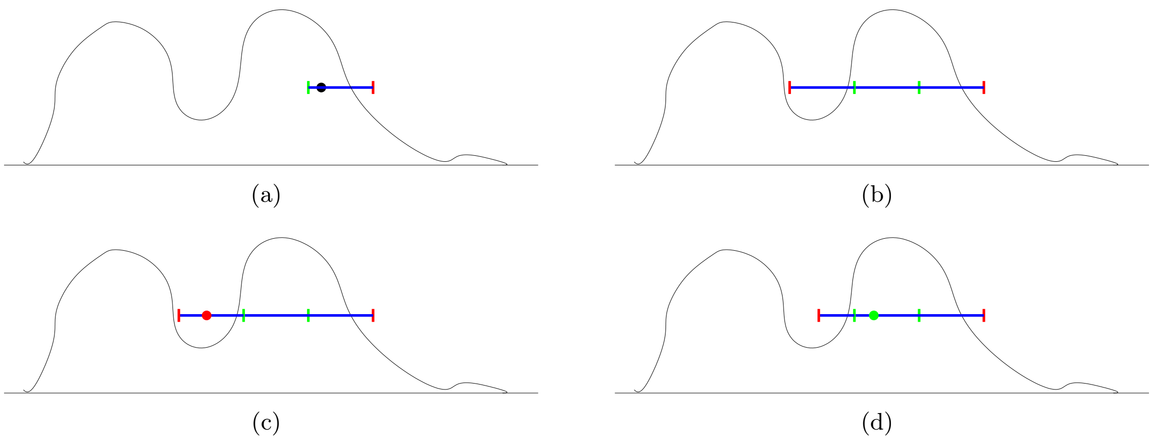


Figure 27.13: **(a)**: For the current sample x, a point y is sampled between 0 and $p^*(x)$, giving a point (x, y) (black circle). Then an interval of width w is placed around x, the blue bar. The ends of the bar denote if the point is in the slice (green) or out of the slice (red). **(b)**: The interval is increased until it hits a point out of the slice. **(c)**: Given an interval a sample x' is taken uniformly in the interval. If the candidate x' is not in the slice (red), $p(x') < y$, the candidate is rejected and the interval is shrunk. **(d)**: The sampling from the interval is repeated until a candidate is in the slice (green), and is subsequently accepted.

For a given y, we call the x that satisfy this a 'slice', fig(27.12). Computing the normalisation of this distribution is in general non-trivial since we would in principle need to search over all x to find those for which $p^*(x) > y$. Ideally we would like to get as much of the slice as feasible, since this will improve the mixing of the chain. If we concentrate on the part of the slice only very local to the current x, then the samples move through the space mix very slowly. If we attempt to guess at random a point a long way from x and check if is in the slice, this will be wasteful. The happy compromise presented in algorithm(28)[205] and described in fig(27.13) determines an appropriate local slice by adjusting the left and right regions. The technique is to start from the current x and attempt to find the largest local slice by incrementally widening the candidate slice. Once we have the largest potential slice we attempt to sample from this. If the sample point within the local slice is in fact not in the slice, this is rejected and the slice is shrunk.

This describes a valid procedure for sampling from a univariate distribution $p^*(x)$. To sample from a multivariate distribution $p(\mathbf{x})$, single variable Gibbs sampling can be used to sample from $p(x_j|x_{\setminus j})$, repeatedly choosing a new variable x_j to sample.

27.6 Importance Sampling

Importance sampling is a technique to approximate averages with respect to an intractable distribution $p(x)$. The term 'sampling' is arguably a misnomer since the method does not attempt to draw samples from $p(x)$. Rather the method draws samples from a simpler *importance distribution* $q(x)$ and then reweights them such that averages with respect to $p(x)$ can be approximated using the samples from $q(x)$. Consider $p(x) = \frac{p^*(x)}{Z}$ where $Z = \int_x p^*(x)$ is the intractable normalisation constant. The average of $f(x)$ with respect to $p(x)$ is given by

$$\int_x f(x)p(x) = \frac{\int_x f(x)p^*(x)}{\int_x p^*(x)} = \frac{\int_x f(x)\frac{p^*(x)}{q(x)}q(x)}{\int_x \frac{p^*(x)}{q(x)}q(x)} \tag{27.6.1}$$

Let $x^1, \ldots x^L$ be samples from $q(x)$, then we can approximate the average by

$$\int_x f(x)p(x) \approx \frac{\sum_{l=1}^L f(x^l)\frac{p^*(x^l)}{q(x^l)}}{\sum_{l=1}^L \frac{p^*(x^l)}{q(x^l)}} = \sum_{l=1}^L f(x^l)w^l \tag{27.6.2}$$

Algorithm 28 Slice Sampling (univariate).

Choose a starting point x^1 and step size w.
for $i = 1$ to L **do**
  Draw a vertical coordinate y uniformly from the interval $\left(0, p^*(x^i)\right)$.
  Create a horizontal interval (x_{left}, x_{right}) that contains x^i as follows:
  Draw $r \sim U\left(r|\,(0,1)\right)$
  $x_{left} = x^i - rw$, $x_{right} = x^i + (1-r)w$ ▷ Create an initial interval
  while $p^*(x_{left}) > y$ **do**
    $x_{left} = x_{left} - w$ ▷ Step out left
  end while
  while $p^*(x_{right}) > y$ **do**
    $x_{right} = x_{right} + w$ ▷ Step out right
  end while
  $accept$ = false
  while $accept$ = false **do**
    draw a random value x' uniformly from the unit interval (x_{left}, x_{right}).
    if $p^*(x') > y$ **then**
      $accept$ = true ▷ Found a valid sample
    else
      modify the interval (x_{left}, x_{right}) as follows:
      if $x' > x^i$ **then**
        $x_{right} = x'$ ▷ Shrinking
      else
        $x_{left} = x'$
      end if
    end if
  end while
  $x^{i+1} = x'$
end for

where we define the *normalised importance weights*

$$w^l = \frac{p^*(x^l)/q(x^l)}{\sum_{l=1}^{L} p^*(x^l)/q(x^l)} \tag{27.6.3}$$

In principle, this reweighting of the samples from q will give the correct result. In high dimensions there will typically only be one dominant weight with value close to 1, and the rest will be zero, particularly if the sampling distribution q is not well matched to p. In practice, therefore, when dealing with high dimensional x, numerical problems often arise with this scheme and techniques such as Sampling Importance Resampling are used to correct this[231]. See [234] for an application of importance sampling to Belief Networks.

27.6.1 Sequential importance sampling

One can apply importance sampling to temporal distributions $p(x_{1:t})$ for which the importance distribution samples $q(x_{1:t})$ are paths. In many applications such as tracking, one wishes to update ones beliefs as time increases and, as such, is required to resample and then reweight the whole path. For distributions $p(x_{1:t})$ with a Markov structure, one would expect that a local update is possible, without needing to deal with the previous path. To show this, consider the importance weights for a sample *path* $x^l_{1:t}$

$$\tilde{w}^l_t = \frac{p^*(x^l_{1:t})}{q(x^l_{1:t})} = \frac{p^*(x^l_{1:t-1})}{q(x^l_{1:t-1})} \frac{p^*(x^l_{1:t})}{p^*(x^l_{1:t-1})q(x^l_t|x^l_{1:t-1})}, \qquad \tilde{w}^l_1 = \frac{p^*(x^l_1)}{q(x^l_1)} \tag{27.6.4}$$

We can recursively define the un-normalised weights using

$$\tilde{w}^l_t = \tilde{w}^l_{t-1}\alpha^l_t, \qquad t > 1 \tag{27.6.5}$$

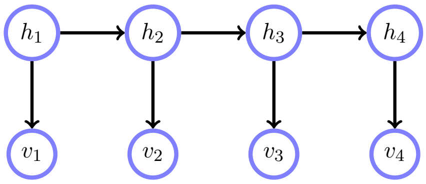


Figure 27.14: A Dynamic Bayesian Network. In many applications of interest, the emission distribution $p(v_t|h_t)$ is non-Gaussian, leading to the formal intractability of filtering/smoothing.

where

$$\alpha_t^l \equiv \frac{p^*(x_{1:t}^l)}{p^*(x_{1:t-1}^l)q(x_t^l|x_{1:t-1}^l)} \tag{27.6.6}$$

For Dynamic Bayes Networks, equation (27.6.6) will simplify considerably. For example consider distributions with a Hidden Markov independence structure,

$$p(v_{1:t}, h_{1:t}) = p(v_1|h_1)p(h_1)\prod_{t=2}^{t} p(v_t|h_t)p(h_t|h_{t-1}) \tag{27.6.7}$$

where $v_{1:t}$ are observations and $h_{1:t}$ are the random variables. Then the weight for a sample path $h_{1:t}^l$ can be defined recursively using α, for which a cancelation of terms in the numerator and denominator occurs, leaving simply

$$\alpha_t^l \equiv \frac{p(v_t|h_t^l)p(h_t^l|h_{t-1}^l)}{q(h_t^l|h_{1:t}^l)} \tag{27.6.8}$$

A difficulty is that most paths sampled from q will have small importance weight since, from the product through time, equation (27.6.5), an exponential dominance will arise as t increases. On its own, therefore, SIS is not a viable technique for sampling paths $p(h_{1:t}|v_{1:t})$. Sequential Monte Carlo methods, discussed in the next section, attempt to address some of the shortcomings of SIS to make more practical algorithms. See [81] for a more precise mathematical treatment of the shortcomings of SIS for filtering and practical improvements.

27.7 Sequential Monte Carlo

One approach to making sequential importance sampling more practical is to identify those sample paths that appear promising and make copies of them, deleting the low-weight paths. To do this more formally one uses a Monte Carlo *resampling* procedure which typically selects high weight paths by sampling from existing sample paths according to the normalised weight distribution. To make this more concrete we will address sampling in Dynamic Bayes nets of the generic form

$$p(v_{1:T}, h_{1:T}) = p(v_1|h_1)p(h_1)\prod_{t=2}^{T} p(v_t|h_t)p(h_t|h_{t-1}) \tag{27.7.1}$$

Here we focus on filtering $p(h_t|v_{1:t})$ when the emission and/or transition distributions render the application of the exact forward recursion equation (24.4.1) computationally intractable. We'll discuss the theory in relation to continuous hidden variables h, although much of what follows can also be applied to the discrete case as well as the mixed-case (for example a Switching Linear Dynamical System).

Particle Filter

Let's consider the average of some function $f(h_t)$. Based on importance samples of paths $h_{1:t}^l$ from $p(h_{1:t}^l|v_{1:t})$ along with their normalised weights w_t^l, we can approximate the average using

$$\langle f(h_t)\rangle_{p(h_{1:t}|v_{1:t})} \approx \sum_{l=1}^{L} f(h_t^l)w_t^l \tag{27.7.2}$$

The optimal importance distribution $q(h_{1:t})$ is equal to the true target distribution $p(h_{1:t})$ in which case the weights are constant, $\alpha_t = 1$. This is achieved by using the importance transition

$$q(h_t|h_{1:t-1}) \propto p(v_t|h_t)p(h_t|h_{t-1}) \tag{27.7.3}$$

We assume, however, that this is difficult to sample from if we are using non-linear transitions/emissions. An alternative is to use the importance distribution:

$$q(h_t|h_{1:t-1}) = p(h_t|h_{t-1}) \tag{27.7.4}$$

in which case, from equation (27.6.6), $\alpha_t^l = p(v_t|h_t)$ and the unnormalised weights are recursively defined by

$$\tilde{w}_t^l = \tilde{w}_{t-1}^l p(v_t|h_t^l) \tag{27.7.5}$$

A drawback of this procedure is that after a small number of iterations only very few particle weights will be significantly non-zero due to the mismatch between the importance distribution q and the target distribution p. A method that can help address this weight dominance is *resampling*. Given the weight distribution $w_t^1, \ldots, w_t^L$, one draws a new set of L particle (indices). This new set of particles will contain repeats and most likely any of the original low-weight particles will have been discarded. The weight of each of these new particles is set uniformly to $1/L$. This procedure helps select only the 'fittest' of the particles and is known as *Sampling Importance Resampling*[231].

27.7.1 Particle Filtering as an approximate Forward pass

Particle filtering can be viewed as an approximation to the exact filtering recursion. Using ρ to represent the filtered distribution,

$$\rho(h_t) \propto p(h_t|v_{1:t}) \tag{27.7.6}$$

the exact filtering recursion is

$$\rho(h_t) \propto p(v_t|h_t) \int_{h_{t-1}} p(h_t|h_{t-1})\rho(h_{t-1}) \tag{27.7.7}$$

A PF can be viewed as an approximation of equation (27.7.7) in which the message $\rho(h_{t-1})$ is approximated by a sum of δ-peaks:

$$\rho(h_{t-1}) \approx \sum_{l=1}^{L} w_{t-1}^l \delta\left(h_{t-1} - h_{t-1}^l\right) \tag{27.7.8}$$

where w_{t-1}^l are the unnormalised importance weights $\sum_{l=1}^{L} w_{t-1}^l = 1$, and h_{t-1}^l are the particles. In other words, the ρ message is represented as a weighted mixture of delta-spikes where the weight and position of the spikes are the parameters of the distribution. Using equation (27.7.8) in equation (27.7.7), we have

$$\rho(h_t) = \frac{1}{Z} p(v_t|h_t) \sum_{l=1}^{L} p(h_t|h_{t-1}^l) w_{t-1}^l \tag{27.7.9}$$

The constant Z is used to normalise the distribution $\rho(h_t)$. Although $\rho(h_{t-1})$ was a simple sum of delta peaks, in general $\rho(h_t)$ will not be – the delta-peaks get 'broadened' by the hidden-to-hidden and hidden-to-observation factors. Our task is then to approximate $\rho(h_t)$ as a new sum of delta-peaks. Below we discuss a method to achieve this for which explicit knowledge of the normalisation Z is not required. This is useful since in many tracking applications, the normalisation of the emission $p(v_t|h_t)$ is unknown.

A Monte-Carlo sampling approximation

A simple approach to forming an approximate mixture-of-delta functions representation of equation (27.7.9) is to generate a set of sample points using importance sampling. That is we generate a set of samples $h_t^1, \ldots, h_t^L$ from some importance distribution $q(h_t)$ which gives the unnormalised importance weights

$$\tilde{w}_t^l = \frac{p(v_t|h_t^l) \sum_{l'=1}^{L} p(h_t^l|h_{t-1}^{l'}) w_{t-1}^{l'}}{q(h_t^l)} \tag{27.7.10}$$

Defining the normalised weights:

$$w_t^l = \frac{\tilde{w}_t^l}{\sum_{l'} \tilde{w}_t^{l'}} \tag{27.7.11}$$

we obtain an approximation

$$\rho(h_t) \approx \sum_{l=1}^{L} w_t^l \delta\left(h_t - h_t^l\right) \tag{27.7.12}$$

Ideally one would use the importance distribution that makes the importance weights unity, namely

$$q(h_t) \propto p(v_t|h_t) \sum_{l=1}^{L} p(h_t|h_{t-1}^l) w_{t-1}^l \tag{27.7.13}$$

However, this is often difficult to sample from directly due to the unknown normalisation of the emission $p(v_t|h_t)$. A simpler alternative is to sample from the transition mixture:

$$q(h_t) = \sum_{l=1}^{L} p(h_t|h_{t-1}^l) w_{t-1}^l \tag{27.7.14}$$

To do so, one first samples a component l^* from the histogram with weights from $w_{t-1}^1, \ldots, w_{t-1}^L$. Given this sample index, say l^*, one then draws a sample from $p(h_t|h_{t-1}^{l^*})$. In this case the un-normalised weights become simply

$$\tilde{w}_t^l = p(v_t|h_t^l) \tag{27.7.15}$$

This *Forward-Sampling-Resampling* procedure is used in `demoParticleFilter.m` and in the following toy example.

Naturally, in practice, real tracking applications involve complex issues, including tracking multiple objects, transformations of the object (scaling, rotation, morphology changes). Nevertheless, the principles are largely the same and many tracking applications work by seeking simple compatibility functions, often based on the colour histogram in a template. Indeed, tracking objects in complex environments was one of the original applications of Particle Filters, [139].

Example 114 (A toy face-tracking example)**.** At time t a binary face template is in a location $\mathbf{h}_t$, which describes the upper-left corner of the template using a two-dimensional vector. At time $t = 1$ the position of the face is known, see fig(27.15a). The face template is known. In subsequent times the face moves randomly according to

$$\mathbf{h}_t = \mathbf{h}_{t-1} + \sigma \boldsymbol{\eta}_t \tag{27.7.16}$$

where $\boldsymbol{\eta}_t \sim \mathcal{N}(\boldsymbol{\eta}_t|\mathbf{0}, \mathbf{I})$, a two dimensional zero mean unit covariance noise vector. In addition, a fraction of the binary pixels in the whole image are selected at random and their states flipped. The aim is to try

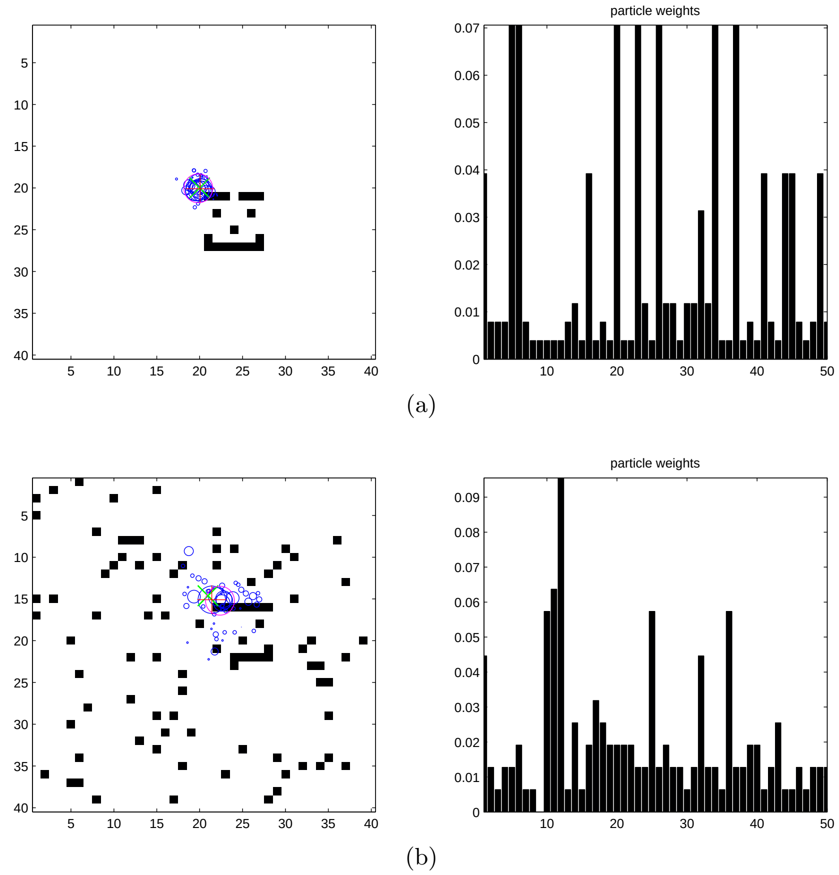


Figure 27.15: Tracking an object with a Particle Filter containing 50 particles. The small circles are the particles, scaled by their weights. The correct corner position of the face is given by the '×', the filtered average by the large circle 'o', and the most likely particle by '+'. **(a)**: Initial position of the face without noise and corresponding weights of the particles. **(b)**: Face with noisy background and the tracked corner position after 20 timesteps. The Forward-Sampling-Resampling PF method is used. Note how a healthy proportion of the weights are non-zero. See `demoParticleFilter.m`

to track the upper-left corner of the face through time.

We need to define the emission distribution $p(\mathbf{v}_t|\mathbf{h}_t)$ on the binary pixels with $v_i \in \{0, 1\}$. Consider the following *compatibility function*

$$\phi(\mathbf{v}_t, \mathbf{h}_t) = \mathbf{v}_t^\mathsf{T} \tilde{\mathbf{v}}(\mathbf{h}_t) \tag{27.7.17}$$

where $\tilde{\mathbf{v}}(\mathbf{h}_t)$ is the vector representing the image with a clean face placed at position $\mathbf{h}_t$. This measures the overlap between the face template and the noisy image restricted to the template pixels. The compatibility function is then maximal when the observed image $\mathbf{v}_t$ has the face placed at position $\mathbf{h}_t$. We can therefore tentatively define

$$p(\mathbf{v}_t|\mathbf{h}_t) \propto \phi(\mathbf{v}_t, \mathbf{h}_t) \tag{27.7.18}$$

A subtlety is that $\mathbf{h}_t$ is continuous, and in the compatibility function we first map $\mathbf{h}_t$ to the nearest integer pixel representation. We have not specified the normalisation constant of this distribution, and fortunately this is not required by the particle filtering algorithm. In fig(27.15a) 50 particles are used to track the face. The particles are plotted along with their corresponding weights. For each $t > 1$, 5% of the pixels are selected at random in the image and their states flipped. This generates background *clutter*, compounding tracking the face. We use the Forward-Sampling-Resampling method and track the face.

27.8 Code

`potsample.m`: Exact sample from a set of potentials
`ancestralsample.m`: Ancestral sample from a Belief Net
`JTsample.m`: Sampling from a consistent Junction Tree
`GibbsSample.m`: Gibbs sampling from a set of potentials

`demoMetropolis.m`: Demo of Metropolis sampling for a bimodal distribution
`metropolis.m`: Metropolis sample
`logp.m`: Log of a bimodal distribution
`demoParticleFilter.m`: Demo Particle Filtering (Forward-Sampling-Resampling method)
`placeobject.m`: Place an object in a grid
`compat.m`: Compatibility function
`demoSampleHMM.m`: Naive Gibbs sampling for a HMM

27.9 Exercises

Exercise 247 (Box-Muller method). *Let* $x_1 \sim U(x_1|[0,1])$, $x_2 \sim U(x_2|[0,1])$ *and*

$$y_1 = \sqrt{-2\log x_1}\cos 2\pi x_2, \qquad y_2 = \sqrt{-2\log x_2}\sin 2\pi x_2 \tag{27.9.1}$$

Show that

$$p(y_1, y_2) = \int p(y_1|x_1, x_2)p(y_2|x_1, x_2)p(x_1)p(x_2)dx_1dx_2 = \mathcal{N}(y_1|0,1)\,\mathcal{N}(y_2|0,1) \tag{27.9.2}$$

and suggest an algorithm to sample from a univariate normal distribution.

Exercise 248. *Consider the distribution*

$$p(x_1, \ldots, x_6) = p(x_1)p(x_2)p(x_3|x_1, x_2)p(x_4|x_3)p(x_5|x_3)p(x_6|x_4, x_5) \tag{27.9.3}$$

For x_6 *fixed in a given state* x_5*, write down a distribution on the remaining variables* $p'(x_1, x_2, x_3, x_4, x_6)$ *and explain how forward (ancestral) sampling can be carried out for this new distribution.*

Exercise 249. *Consider an Ising model on an* $M \times M$ *square lattice with nearest neighbour interactions:*

$$p(x) \propto \exp \beta \sum_{i\sim j} \mathbb{I}[x_i = x_j] \tag{27.9.4}$$

Now consider the $M \times M$ *grid as a checkerboard, and give each white square a label* w_i*, and each black square a label* b_j*, so that each square is associated with a particular variable. Show that*

$$p(b_1, b_2, \ldots, |w_1, w_2, \ldots) = p(b_1|w_1, w_2, \ldots)p(b_2|w_1, w_2, \ldots)\ldots \tag{27.9.5}$$

That is, that conditioned on the white variables, the black variables are independent. The converse is also true, that conditioned on the black variables, the white variables are independent. Explain how this can be exploited by a Gibbs sampling procedure. This procedure is known as checkerboard *or* black and white *sampling.*

Exercise 250. *Consider the symmetric Gaussian proposal distribution*

$$\tilde{q}(\mathbf{x}'|\mathbf{x}) = \mathcal{N}\left(\mathbf{x}'\middle|\mathbf{x}, \sigma_q^2\mathbf{I}\right) \tag{27.9.6}$$

and the target distribution

$$p(\mathbf{x}) = \mathcal{N}\left(\mathbf{x}\middle|\mathbf{0}, \sigma_p^2\mathbf{I}\right) \tag{27.9.7}$$

where $\dim \mathbf{x} = N$*. Show that*

$$\left\langle \log \frac{p(\mathbf{x}')}{p(\mathbf{x})} \right\rangle_{\tilde{q}(\mathbf{x}'|\mathbf{x})} = -\frac{N\sigma_q^2}{2\sigma_p^2} \tag{27.9.8}$$

Discuss how this result relates to the probability of accepting a Metropolis-Hastings update under a Gaussian proposal distribution in high-dimensions.

Exercise 251. *The file* `demoSampleHMM.m` *performs naive Gibbs sampling for a HMM in which a single variable* h_t *is updated at each timestep. A sample is taken as the state of* $h_{1:T}$ *after updating each variable, sweeping forwards through time. The parameter* λ *controls how deterministic the hidden transition matrix* $p(h_t|h_{t-1})$ *will be. Adjust* `demoSampleHMM.m` *to run 100 times, each time for the same* λ*, computing a mean absolute error over these 100 runs. Then repeat this for* $\lambda = 0.1, 1, 10, 20$*. Explain the performance of this Gibbs sampling routine as a function of* λ*?*

CHAPTER 28

Deterministic Approximate Inference

28.1 Introduction

Deterministic approximate inference methods are an alternative to the stochastic techniques discussed in chapter(27). Whilst stochastic methods are powerful and often generally applicable, they nevertheless produce sample estimates of a quantity. Even if we are able to perform perfect sampling, we would still only obtain an approximate result, due to the inherent uncertainty introduced by sampling. Also, sampling is not specifically targeted at estimating a desired quantity of interest. For example, the sampling procedure is unchanged regardless of whether we wish to estimate the mean of a quantity or say the variance. In some cases it can be that accurate results can be obtained by methods which seek to directly estimate quantities of interest, such as the marginal of a distribution, or the normalisation constant of a distribution. We'll consider here a perturbation technique (Laplace's method) and some variational techniques for approximate marginalisation.

It is important to bear in mind that no single approximation technique, deterministic or stochastic, is going to beat all others on all problems, given the same computational resources. In this sense, insight as to the properties of the approximation method used is useful in matching an approximation method to the problem at hand.

28.2 The Laplace approximation

Consider a distribution of the form

$$p(\mathbf{x}) = \frac{1}{Z} e^{-E(\mathbf{x})} \tag{28.2.1}$$

The Laplace method makes a Gaussian approximation of $p(\mathbf{x})$ based on a local perturbation expansion around a mode $\mathbf{x}^*$. First we find the mode numerically, giving

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}}\ E(\mathbf{x}) \tag{28.2.2}$$

Then a Taylor expansion up to second order around this mode gives

$$E(\mathbf{x}) \approx E(\mathbf{x}^*) + (\mathbf{x} - \mathbf{x}^*)^{\mathsf{T}} \nabla E|_{\mathbf{x}^*} + \frac{1}{2} (\mathbf{x} - \mathbf{x}^*)^{\mathsf{T}} \mathbf{H} (\mathbf{x} - \mathbf{x}^*) \tag{28.2.3}$$

where $\mathbf{H} \equiv \nabla\nabla E(\mathbf{x})|_{\mathbf{x}^*}$ is the Hessian evaluated at the mode. At the mode, $\nabla E|_{\mathbf{x}^*} = \mathbf{0}$, and an approximation of the distribution is given by the Gaussian

$$p^*(\mathbf{x}) = \frac{1}{Z^*} e^{-\frac{1}{2}(\mathbf{x}-\mathbf{x}^*)^{\mathsf{T}}\mathbf{H}(\mathbf{x}-\mathbf{x}^*)} = \mathcal{N}\left(\mathbf{x}|\mathbf{x}^*, \mathbf{H}^{-1}\right) \tag{28.2.4}$$

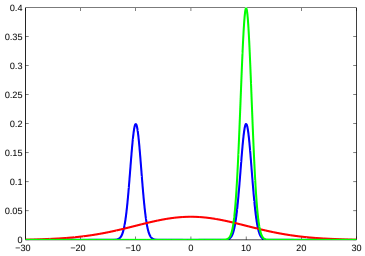


Figure 28.1: Fitting a mixture of Gaussians $p(x)$ (blue) with a single Gaussian. The green curve minimises $\mathrm{KL}(q|p)$ corresponding to fitting a local model. The red curve minimises $\mathrm{KL}(p|q)$ corresponding to moment matching.

which has mean $\mathbf{x}^*$ and covariance $\mathbf{H}^{-1}$, with $Z^* = \sqrt{\det(2\pi\mathbf{H}^{-1})}$. Similarly, we can use the above expansion to estimate the integral

$$\int_{\mathbf{x}} e^{-E(\mathbf{x})} \approx \int_{\mathbf{x}} e^{-E(\mathbf{x}^*)-\frac{1}{2}(\mathbf{x}-\mathbf{x}^*)^{\mathsf{T}}\mathbf{H}(\mathbf{x}-\mathbf{x}^*)} = e^{-E(\mathbf{x}^*)}\sqrt{\det(2\pi\mathbf{H}^{-1})} \tag{28.2.5}$$

Although the Laplace approximation fits a Gaussian to a distribution, it is not necessarily the 'best' Gaussian approximation. As we'll see below, other criteria, such as based on minimal KL divergence between $p(\mathbf{x})$ and a Gaussian approximation may be more appropriate, depending on the context. However, the major benefit of Laplace's method is its speed and simplicity.

28.3 Properties of Kullback-Leibler variational inference

Variational methods can be used to approximate a complex distribution $p(x)$ by a simpler distribution $q(x)$. Given a definition of discrepancy between an approximation $q(x)$ to $p(x)$, any free parameters of $q(x)$ are then set by minimising the discrepancy.

A particularly popular measure of the discrepancy between an approximation $q(x)$ and the intractable distribution $p(x)$ is given by the Kullback-Leibler divergence

$$\mathrm{KL}(q|p) = \langle \log q \rangle_q - \langle \log p \rangle_q \tag{28.3.1}$$

It is straightforward to show that $\mathrm{KL}(q|p) \geq 0$ and is zero if and only if the distributions p and q are identical, see section(8.8). Note that whilst the KL divergence cannot be negative, there is no upper bound on the value it can potentially take so that the discrepancy can be 'infinitely' bad.

For a distribution of the form

$$p(x) = \frac{1}{Z} e^{\phi(x)} \tag{28.3.2}$$

we have

$$\mathrm{KL}(q|p) = \langle \log q(x) \rangle_{q(x)} - \langle \log p(x) \rangle_{q(x)} = \langle \log q(x) \rangle_{q(x)} - \langle \phi(x) \rangle_{q(x)} + \log Z \tag{28.3.3}$$

Since $\mathrm{KL}(q|p) \geq 0$ this immediately gives the bound

$$\log Z \geq \underbrace{- \langle \log q(x) \rangle_{q(x)}}_{entropy} + \underbrace{\langle \phi(x) \rangle_{q(x)}}_{energy} \tag{28.3.4}$$

which is called the 'free energy' bound in the physics community[232]. Using the notation H_q for the entropy of q, we can write the bound more compactly as

$$\log Z \geq H_q + \langle \phi(x) \rangle_{q(x)} \tag{28.3.5}$$

28.3.1 Gaussian approximations using KL divergence

Minimising KL$(q|p)$

Using a simple approximation $q(x)$ of a more complex distribution $p(x)$ by minimising $\mathrm{KL}(q|p)$ tends to give a solution for $q(x)$ that focuses on a local mode of $p(x)$, thereby underestimating the variance of $p(x)$. To show this, consider approximating a mixture of two Gaussians with equal variance σ^2,

$$p(x) = \frac{1}{2}\left(\mathcal{N}\left(x|-\mu,\sigma^2\right) + \mathcal{N}\left(x|\mu,\sigma^2\right)\right) \tag{28.3.6}$$

see fig(28.1), with a single Gaussian

$$q(x) = \mathcal{N}\left(x|m,s^2\right) \tag{28.3.7}$$

We wish to find the optimal m, s^2 that minimise

$$\mathrm{KL}(q|p) = \langle \log q(x)\rangle_{q(x)} - \langle \log p(x)\rangle_{q(x)} \tag{28.3.8}$$

If we consider the case that the two Gaussian components of $p(x)$ are well separated, $\mu \gg \sigma$, then setting $q(x)$ to be centred on the left mode at $-\mu$ the Gaussian $q(x)$ only has appreciable mass close to $-\mu$, so that the second mode at μ has negligible contribution to the Kullback-Leibler divergence. In this sense one can approximate $p(x) \approx \frac{1}{2}q(x)$, so that

$$\mathrm{KL}(q|p) \approx \langle \log q(x)\rangle_{q(x)} - \langle \log p(x)\rangle_{q(x)} = \log 2 \tag{28.3.9}$$

On the other hand, setting $m = 0$, which is the correct mean of the distribution $p(x)$, very little of the mass of the mixture is captured unless s^2 is large, giving a poor fit and large KL divergence. Another way to view this is to consider $\mathrm{KL}(q|p) = \langle \log q(x)/p(x)\rangle_{q(x)}$; provided q is close to p around where q has significant mass, the ratio $q(x)/p(x)$ will be order 1 and the KL divergence small. Setting $m = 0$ means that $q(x)/p(x)$ is large where q has significant mass, and is therefore a poor fit. The optimal solution in this case is to place the Gaussian close to a single mode. Note, however, that for two modes that are less well-separated, the optimal solution will not necessarily be to place the Gaussian around a local mode.

In general, the optimal Gaussian fit needs to be determined numerically – that is, there is no closed form solution to finding the optimal mean and (co)variance parameters.

Minimising KL$(p|q)$

For fitting a Gaussian q to p based on $\mathrm{KL}(p|q)$, we have

$$\mathrm{KL}(p|q) = \langle \log p(x)\rangle_{p(x)} - \langle \log q(x)\rangle_{p(x)} \tag{28.3.10}$$

$$= -\frac{1}{2\sigma^2}\left\langle (x-m)^2\right\rangle_{p(x)} - \frac{1}{2}\log\det\left(\sigma^2\right) + \mathsf{const.} \tag{28.3.11}$$

Minimising this with respect to m and σ^2 we obtain, optimally

$$m = \langle x\rangle_{p(x)}, \qquad \sigma^2 = \left\langle (x-m)^2\right\rangle_{p(x)} \tag{28.3.12}$$

so that the optimal Gaussian fit matches the first and second moments of $p(x)$.

In the case of fig(28.1), the mean of $p(x)$ is a zero, and the variance of $p(x)$ is large. This solution is therefore dramatically different from that produced by fitting the Gaussian using $\mathrm{KL}(q|p)$. The fit found using $\mathrm{KL}(q|p)$ focusses on making q fit p locally well, whereas $\mathrm{KL}(p|q)$ focusses on making q fit p well to the global statistics of the distribution (possibly at the expense of a good local match).

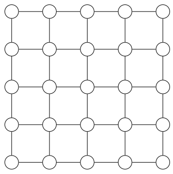


Figure 28.2: A planar pairwise Markov random field on a set of variables $x_1, \ldots, x_{25}$, representing a distribution of the form $\prod_{i\sim j} \phi(x_i, x_j)$. In statistical physics such lattice models include the Ising model on binary 'spin' variables $x_i \in \{+1, -1\}$ with $\phi(x_i, x_j) = e^{w_{ij} x_i x_j}$.

28.3.2 Moment matching properties of minimising KL($p|q$)

For simplicity, consider a factorised approximation $q(x) = \prod_i q(x_i)$. Then

$$\mathrm{KL}(p|q) = \langle \log p(x) \rangle_{p(x)} - \sum_i \langle \log q(x_i) \rangle_{p(x_i)} \tag{28.3.13}$$

The first entropic term is independent of $q(x)$ so that up to a constant independent of $q(x)$, the above is

$$\sum_i \mathrm{KL}(p(x_i)|q(x_i)) \tag{28.3.14}$$

so that optimally $q(x_i) = p(x_i)$. That is, the optimal factorised approximation is to set the factors of $q(x_i)$ to the marginals of $p(x_i)$, exercise(265). This effect is quite general – indeed for any approximating distribution in the exponential family, minimising KL($p|q$) corresponds to moment matching, see exercise(264).

In practice, one generally cannot compute the moments of $p(x)$ (since the distribution $p(x)$ is considered 'intractable'), so that fitting q to p based only on KL($p|q$) does not itself lead to a practical algorithm for approximate inference. Nevertheless, as we will see, it is a useful subroutine for local approximations, in particular Expectation Propagation.

28.4 Variational bounding using KL($q|p$)

In this section we discuss how to fit a distribution $q(x)$ from some assumed family to an 'intractable' distribution $p(x)$. As we saw above for the case of fitting Gaussians, the optimal q needs to be found numerically. This itself can be a complex task (indeed, formally this can be just as difficult as performing inference directly with the intractable p) and the reader may wonder why we trade a difficult inference task for a potentially difficult optimisation problem. The general idea is that the optimisation problem has some local smoothness properties that enable one to rapidly find a reasonable optimum based on generic optimisation methods. To make these ideas more concrete, we discuss a particular case of fitting q to a formally intractable p below.

28.4.1 Pairwise Markov random field

A canonical 'intractable' distribution is the pairwise Markov Random Field defined on binary variables $x_i \in \{+1, -1\}$, $i = 1, \ldots, D$,

$$p(x) = \frac{1}{Z(w, b)} e^{\sum_{i,j} w_{ij} x_i x_j + \sum_i b_i x_i} \tag{28.4.1}$$

Here the 'partition function' $Z(w, b)$ ensures normalisation,

$$Z(w, b) = \sum_x e^{\sum_{i,j} w_{ij} x_i x_j + \sum_i b_i x_i} \tag{28.4.2}$$

Since for $i = j$, $x_i x_j = 1$ so that without loss of generality we may set $w_{i,i}$ to zero, which we assume throughout[1]. This gives an undirected distribution with connection geometry defined by the weights $\mathbf{w}$. In practice, the weights often define local interactions on a lattice, see fig(28.2). A case for which inference in this model is required is given in example(115).

[1]Whilst inference with a general MRF is formally computationally intractable (no-known exact polynomial time methods exist), two celebrated results that we mention in passing are that for the planar MRF model (the 'Ising' model) with pure interactions ($b = 0$), the partition function is computable in polynomial time[154, 92, 174, 112, 240], as is the MAP state for attractive planar Ising models $w > 0$ [118], see section(28.8).

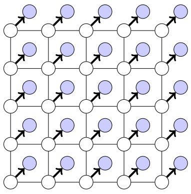


Figure 28.3: A distribution on pixels. The filled nodes indicate observed noisy pixels, the unshaded nodes a Markov Random Field on latent clean pixels. The task is to infer the clean pixels given the noisy pixels. The MRF encourages the posterior distribution on the clean pixels to contain neighbouring pixels in the same state.

Example 115 (Bayesian image denoising)**.** Consider a binary image, where x describes the state of the clean pixels. We assume a noisy pixel generating process that takes each clean pixel x_i and flips its binary state:

$$p(y|x) = \prod_i p(y_i|x_i), \qquad p(y_i|x_i) \propto e^{\gamma y_i x_i} \tag{28.4.3}$$

The probability that y_i and x_i are in the same state is $e^{\gamma}/(e^{\gamma} + e^{-\gamma})$. Our interest is to the posterior distribution on clean pixels $p(x|y)$. In order to do this we need to make an assumption as to what clean images look like. We do this using a MRF

$$p(x) \propto e^{\sum_{i\sim j} w_{ij} x_i x_j} \tag{28.4.4}$$

for some settings of $w_{ij} > 0$. This encodes the assumption that clean images tend to have neighbouring pixels in the same state. An isolated pixel in a different state to its neighbours is unlikely under this prior. We now have the joint distribution

$$p(x, y) = p(x) \prod_i p(y_i|x_i) \tag{28.4.5}$$

see fig(28.3), from which the posterior is given by

$$p(x|y) = \frac{p(y|x)p(x)}{\sum_x p(y|x)p(x)} \propto e^{\sum_{i\sim j} w_{ij} x_i x_j + \sum_i \gamma y_i x_i} \tag{28.4.6}$$

This is an Ising model with external fields (singleton terms x_i) so that quantities such as the MAP state (most a posteriori probable image), marginals and the normalisation constant (for learning) are of interest.

Kullback-Leibler based methods

For the MRF we have

$$\mathrm{KL}(q|p) = \langle \log q \rangle_q - \sum_{ij} w_{ij} \langle x_i x_j \rangle_q - \sum_i b_i \langle x_i \rangle_q + \log Z \geq 0 \tag{28.4.7}$$

Rewriting, this gives a bound on the log-partition function

$$\log Z \geq \underbrace{- \langle \log q \rangle_q}_{entropy} + \underbrace{\sum_{ij} w_{ij} \langle x_i x_j \rangle_q + \sum_i b_i \langle x_i \rangle_q}_{energy} \tag{28.4.8}$$

The bound saturates when $q = p$. However, this is clearly of little help, since we cannot compute the averages of variables $\langle x_i x_j \rangle_p, \langle x_i \rangle_p$. The idea of a variational method is to assume a simpler 'tractable' distribution q for which these averages can be computed, along with the entropy of q. Minimising the KL divergence with respect to any free parameters of $q(x)$ is then equivalent to maximising the lower bound on the log partition function.

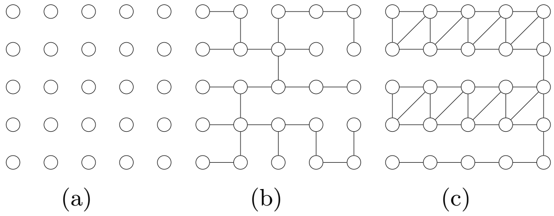


Figure 28.4: **(a)**: Naive Mean Field approximation $q(x) = \prod_i q_i(x_i)$. **(b)**: A spanning tree approximation. **(c)**: A decomposable (hypertree) approximation.

Factorised approximation

The simplest assumption is the fully factorised distribution

$$q(x) = \prod_i q_i(x_i) \tag{28.4.9}$$

The graphical model of this approximation is given in fig(28.4a). In this case

$$\log Z \geq -\sum_i \langle \log q_i \rangle_{q_i} + \sum_{ij} w_{ij} \langle x_i x_j \rangle_{q(x_i,x_j)} + \sum_i b_i \langle x_i \rangle_{q(x_i)} \tag{28.4.10}$$

For a factorised distribution and bearing in mind that $x_i \in \{+1, -1\}$,

$$\langle x_i x_j \rangle = \begin{cases} 1 & i = j \\ \langle x_i \rangle \langle x_j \rangle & i \neq j \end{cases} \tag{28.4.11}$$

For a binary variable, one may use the convenient parametrization

$$q_i(x_i = 1) = \frac{e^{\theta_i}}{e^{\theta_i} + e^{-\theta_i}} \tag{28.4.12}$$

so that

$$\langle x_i \rangle_{q_i} = +1 \times q(x_i = 1) - 1 \times q(x_i = -1) = \tanh(\theta_i) \tag{28.4.13}$$

This gives the following lower bound on the log partition function:

$$\log Z \geq \mathcal{B}(\theta) \equiv \sum_i H(\theta_i) + \sum_{i \neq j} w_{ij} \tanh(\theta_i) \tanh(\theta_j) + \sum_i b_i \tanh(\theta_i) \tag{28.4.14}$$

where $H(\theta_i)$ is the *binary entropy* of a distribution parameterised according to equation (28.4.12):

$$H(\theta_i) = \log\left(e_i^\theta + e^{-\theta_i}\right) - \theta_i \tanh(\theta_i) \tag{28.4.15}$$

Finding the best factorised approximation in the minimal Kullback-Liebler divergence sense then corresponds to maximising the bound $\mathcal{B}(\theta)$ with respect to the variational parameters θ.

The bound $\mathcal{B}$, equation (28.4.14), is non-convex in θ and typically riddled with local optima, meaning that finding the globally optimal θ is typically a computationally hard problem. It seems therefore that we have simply replaced a computationally hard problem of computing $\log Z$ by an equally hard computational problem of maximising $\mathcal{B}(\theta)$. Indeed, the 'graphical structure' of this optimisation problem matches exactly that of the original MRF. However, the hope is that by transforming a difficult discrete summation into a difficult continuous optimisation, we will be able to bring to the table techniques of continuous variable numerical optimisation to find a good approximation.

A particularly simple optimisation technique is to differentiate the bound equation (28.4.14) and equate to zero. Straightforward algebra leads to requirement that the optimal solution satisfies the equations

$$\theta_i = b_i + \sum_{j \neq i} w_{ij} \tanh(\theta_j), \forall i \tag{28.4.16}$$

One may show that updating any θ_i according to equation (28.4.16) increases $\mathcal{B}(\theta)$. This is called *asychronous updating* and is guaranteed to lead to a (local) minimum of the KL divergence, section(28.4.3).

Once a converged solution has been identified, in addition to a bound on $\log Z$, we can approximate

$$\langle x_i \rangle_p \approx \langle x_i \rangle_q = \tanh(\theta_i) \tag{28.4.17}$$

Validity of the factorised approximation

When might one expect such a naive factorised approximation to work well? Clearly, if w_{ij} is very small for $i \neq j$, the distribution p will be effectively factorised. However, a more interesting case is when each variable x_i has many neighbours. It is useful to write the MRF as

$$p(x) = \frac{1}{Z} e^{\sum_{ij} w_{ij} x_i x_j} = \frac{1}{Z} e^{D \sum_i x_i \frac{1}{D} \sum_j w_{ij} x_j} = \frac{1}{Z} e^{D \sum_i x_i z_i} \tag{28.4.18}$$

where the local 'fields' are defined as

$$z_i \equiv \frac{1}{D} \sum_j w_{ij} x_j \tag{28.4.19}$$

We now invoke a circular (but self-consistent) argument: Let's assume that $p(x)$ is factorised. Then for z_i, each of the terms x_j in the summation $\sum_j w_{ij} x_j$ is independent. Provided the w_{ij} are not strongly correlated the conditions of validity of the Central Limit theorem hold[120], and each z_i will be Gaussian distributed. Assuming that each w_{ij} is $O(1)$, the mean of z_i is

$$\langle z_i \rangle = \frac{1}{D} \sum_j w_{ij} \langle x_j \rangle = O(1) \tag{28.4.20}$$

The variance is

$$\left\langle z_i^2 \right\rangle - \langle z_i \rangle^2 = \frac{1}{D^2} \sum_{k=1}^{D} w_{ik}^2 \left(1 - \langle x_k \rangle^2\right) = O(1/D) \tag{28.4.21}$$

Hence the variance of the field z_i is much smaller than its mean value. As D increases the fluctuations around the mean therefore diminish, and we may write

$$p(x) \approx \frac{1}{Z} e^{D \sum_i x_i \langle z_i \rangle} \approx \prod_i p(x_i) \tag{28.4.22}$$

We have shown therefore that the assumption that p is approximately factorised is self-consistent in the limit of MRFs with a large number of neighbours. Hence the factorised approximation would appear to be reasonable in the extreme limits (i) a very weakly connected system $w_{ij} \approx 0$, or (ii) a large densely connected system.

The fully factorised approximation is also called the *Naive Mean Field* theory since for the MRF case it assumes that we can replace the effect of the neighbours by a mean of the field at each site.

28.4.2 General mean field equations

For a general intractable distribution $p(x)$ on discrete or continuous x, the KL divergence between a factorised approximation $q(x)$ and $p(x)$ is

$$\mathrm{KL}\left(\prod_i q(x_i) | p(x)\right) = \sum_i \langle \log q(x_i) \rangle_{q(x_i)} - \langle \log p(x) \rangle_{\prod_i q(x_i)} \tag{28.4.23}$$

Isolating the dependency of the above on a single factor $q(x_i)$ we have

$$\langle \log q(x_i) \rangle_{q(x_i)} - \left\langle \langle \log p(x) \rangle_{\prod_{j \neq i} q(x_j)} \right\rangle_{q(x_i)} \tag{28.4.24}$$

Up to a normalisation constant, this is therefore the KL divergence between $q(x_i)$ and a distribution proportional to $\exp\left(\langle \log p(x) \rangle_{\prod_{j \neq i} q(x_j)}\right)$ so that the optimal setting for $q(x_i)$ satisfies

$$q(x_i) \propto \exp\left(\langle \log p(x) \rangle_{\prod_{j \neq i} q(x_j)}\right) \tag{28.4.25}$$

This can be used as an update equation to define a new approximation in terms of the previous approximation factors. These are the mean-field equations. Note that if the normalisation constant of $p(x)$ is unknown, this presents no problem since this constant is simply absorbed into the normalisation of the factors $q(x_i)$ – in other words, one may replace $p(x)$ with the unnormalised $p^*(x)$ in equation (28.4.25). Beginning with an initial randomly chosen set of distributions $q(x_i)$, the mean-field equations are iterated until convergence. Asynchronous updating is guaranteed to decrease the KL divergence at each stage, section(28.4.3).

Whilst the fully factorised approximation is rather severe, even this may not be enough to render the mean-field equations tractably implementable. To do so we need to be able to compute $\langle \log p^*(x) \rangle_{\prod_{j \neq i} q(x_j)}$. For some models of interest this is still not possible, and additional approximations are required to compute the mean-field equations and the corresponding bound on the log partition function[235].

Example 116 ('Intractable' mean-field approximation)**.** Consider the posterior distribution from a Relevance Vector Machine classification problem, section(18.2.3):

$$p(\mathbf{w}|\mathcal{D}) \propto \mathcal{N}\left(\mathbf{w}\middle|\mathbf{0}, s^2\mathbf{I}, \prod\right)_n \sigma\left(c_n \mathbf{w}^\mathsf{T}\mathbf{x}^n\right) \tag{28.4.26}$$

The terms $\sigma\left(c_n \mathbf{w}^\mathsf{T}\mathbf{x}^n\right)$ render $p(\mathbf{w}|\mathcal{D})$ non-Gaussian. We can find a Gaussian approximation $q(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ by minimising the Kullback-Leibler divergence

$$\mathrm{KL}(q(\mathbf{w})|p(\mathbf{w}|\mathcal{D})) = \langle \log q(\mathbf{w}) \rangle_{q(\mathbf{w})} - \langle \log p(\mathbf{w}|\mathcal{D}) \rangle_{q(\mathbf{w})} \tag{28.4.27}$$

The entropic term is straightforward since this is the negative entropy of a Gaussian. However, we also require the 'energy' which includes a contribution

$$\left\langle \log \sigma\left(c_n \mathbf{w}^\mathsf{T}\mathbf{x}^n\right)\right\rangle_{\mathcal{N}(\mathbf{w}|\mu,\Sigma)} \tag{28.4.28}$$

There is no closed form expression for this. One approach is to use additional variational approximations, [140]. Another approach is to recognise that the multi-variate average can be reduced to a uni-variate Gaussian average:

$$\left\langle \log \sigma\left(c_n \mathbf{w}^\mathsf{T}\mathbf{x}^n\right)\right\rangle_{\mathcal{N}(\mathbf{w}|\mu,\Sigma)} = \langle \log \sigma(c_n a) \rangle_{\mathcal{N}(a|m,\tau^2)}, \qquad m = \boldsymbol{\mu}^\mathsf{T}\mathbf{x}^n, \qquad \tau^2 = (\mathbf{x}^n)^\mathsf{T}\boldsymbol{\Sigma}\mathbf{x}^n \tag{28.4.29}$$

and the uni-variate Gaussian average can be carried out using quadrature. This approach was used in [22] to approximate the posterior distribution of Bayesian Neural Networks.

28.4.3 Asynchronous updating guarantees approximation improvement

For a factorised variational approximation equation (28.4.23), we claim that each update equation (28.4.25) reduces the Kullback-Leibler approximation error. To show this we write a single updated distribution as

$$q_i^{new} = \frac{1}{Z_i} \exp \langle \log p(x) \rangle_{\prod_{j \neq i} q_j^{old}} \tag{28.4.30}$$

The joint distribution under this single update is

$$q^{new} = q_i^{new} \prod_{j \neq i} q_j^{old} \tag{28.4.31}$$

Our interest is the change in the approximation error under this single mean-field update. This is measured by

$$\Delta \equiv \mathrm{KL}(q^{new}|p) - \mathrm{KL}\left(q^{old}|p\right) \tag{28.4.32}$$

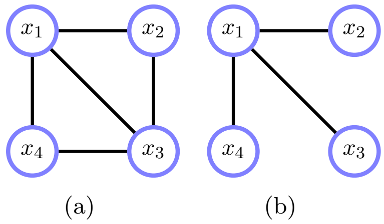


Figure 28.5: **(a)**: A toy 'intractable' distribution. **(b)**: A structured singly-connected approximation.

Using

$$\mathrm{KL}(q^{new}|p) = \langle \log q_i^{new} \rangle_{q_i^{new}} + \sum_{j \neq i} \left\langle \log q_j^{old} \right\rangle_{q_j^{old}} - \left\langle \langle \log p(x) \rangle_{\prod_{j \neq i} q_j^{old}} \right\rangle_{q_i^{new}} \tag{28.4.33}$$

and defining the un-normalised distribution

$$q_i^*(x_i) = \exp \langle \log p(x) \rangle_{\prod_{j \neq i} q_j^{old}} = Z_i q_i^{new} \tag{28.4.34}$$

then

$$\Delta = \langle \log q_i^{new} \rangle_{q_i^{new}} - \left\langle \log q_i^{old} \right\rangle_{q_i^{old}} - \left\langle \langle \log p \rangle_{\prod_{j \neq i} q_j^{old}} \right\rangle_{q_i^{new}} + \left\langle \langle \log p \rangle_{\prod_{j \neq i} q_j^{old}} \right\rangle_{q_i^{old}} \tag{28.4.35}$$

$$= \langle \log q_i^* \rangle_{q_i^{new}} - \log Z_i - \left\langle \log q_i^{old} \right\rangle_{q_i^{old}} - \langle \log q_i^* \rangle_{q_i^{new}} + \langle \log q_i^* \rangle_{q_i^{old}} \tag{28.4.36}$$

$$= -\log Z_i - \left\langle \log q_i^{old} \right\rangle_{q_i^{old}} + \langle \log q_i^* \rangle_{q_i^{old}} \tag{28.4.37}$$

$$= -\mathrm{KL}\left(q_i^{old} | q_i^{new}\right) \leq 0 \tag{28.4.38}$$

Hence

$$\mathrm{KL}\left(q^{new}|p\right) \leq \mathrm{KL}\left(q^{old}|p\right) \tag{28.4.39}$$

so that updating a single component of q at a time is guaranteed to improve the approximation. Note that this result is quite general, holding for any distribution $p(x)$. In the case of a Markov network the guaranteed approximation improvement is equivalent to a guaranteed increase (strictly speaking a non-decrease) in the lower bound on the partition function.

28.4.4 Structured variational approximation

One can extend the power of the KL variational approximation by using non-factorised $q(x)$[236, 24]. Those for which averages of the variables can be computed in linear time include spanning trees, fig(28.4b) and decomposable graphs fig(28.4c). For example, for the distribution

$$p(x_1, x_2, x_3, x_4) = \frac{1}{Z} \phi(x_1, x_2)\phi(x_2, x_3)\phi(x_3, x_4)\phi(x_4, x_1)\phi(x_1, x_3) \tag{28.4.40}$$

a tractable q distribution would be, fig(28.5)

$$q(x_1, x_2, x_3, x_4) = \frac{1}{\tilde{Z}} \tilde{\phi}(x_1, x_2)\tilde{\phi}(x_1, x_3)\tilde{\phi}(x_1, x_4) \tag{28.4.41}$$

In this case we have

$$\mathrm{KL}(q|p) = H_q(x_1, x_2) + H_q(x_1, x_3) + H_q(x_1, x_4) - 3H_q(x_1) + \sum_{i \sim j} \langle \log \phi(x_i, x_j) \rangle_{q(x_i, x_j)} \tag{28.4.42}$$

Since q is singly-connected, computing the marginals and entropy is straightforward (since the entropy requires only pairwise marginals on graph neighbours).

More generally one can exploit any structural approximation with an arbitrary hypertree width w by use of the Junction Tree algorithm in combination with the KL divergence. However, the computational expense increases exponentially with the hypertree width[291].

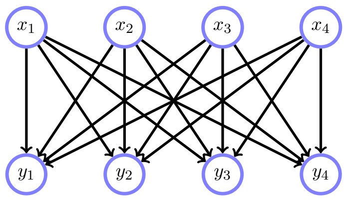


Figure 28.6: An information transfer problem. For a fixed distribution $p(x) = \prod_i p(x_i)$ and parameterised distributions $p(y_j|\mathbf{x}) = \sigma\left(\mathbf{w}_i^\mathsf{T}\mathbf{x}\right)$, find the optimal parameters $\mathbf{w}_i$ that maximise the mutual information between the variables x and y. Such considerations are popular in theoretical neuroscience and aim to understand how the receptive fields $\mathbf{w}_i$ of a neuron relate to the statistics of the environment $p(x)$.

28.5 Mutual information maximisation : A KL variational approach

The Kullback-Leibler variational approach has use also in information theory, and we take a short interlude here to demonstrate one application. In information theory a fundamental goal is to maximise information transfer, measured by the (see also definition(87))

$$I(X, Y) \equiv H(X) - H(X|Y) \tag{28.5.1}$$

where the and *conditional entropy*re defined

$$H(X) \equiv -\langle \log p(x) \rangle_{p(x)}, \qquad H(X|Y) \equiv -\langle \log p(x|y) \rangle_{p(x,y)} \tag{28.5.2}$$

Here we are interested in the situation in which $p(x)$ is fixed, but $p(y|x, \theta)$ has adjustable parameters θ that we wish to set to maximise $I(X, Y)$. In this case $H(X)$ is constant and the optimisation problem is equivalent to minimising the conditional entropy $H(X|Y)$. Unfortunately, in many cases of practical interest $H(X|Y)$ is computationally intractable. See example(117) below for a motivating example. We discuss in section(28.5.1) a general procedure based on the Kullback-Leibler divergence to approximately maximise the mutual information.

Example 117. Consider a neural transmission system in which $x_i \in \{0, 1\}$ denotes an emitting neuron in a non-firing state (0) or firing state (1), and $y_j \in \{0, 1\}$ a receiving neuron. If each receiving neuron fires independently, depending only on the emitting neurons, we have

$$p(\mathbf{y}|\mathbf{x}) = \prod_i p(y_i|\mathbf{x}) \tag{28.5.3}$$

where for example we could use

$$p(y_i = 1|\mathbf{x}) = \sigma\left(\mathbf{w}_i^\mathsf{T}\mathbf{x}\right) \tag{28.5.4}$$

If we make the simple assumption that emitting neurons fire independently,

$$p(\mathbf{x}) = \prod_i p(x_i) \tag{28.5.5}$$

then for $p(\mathbf{x}|\mathbf{y})$ all components of the $\mathbf{x}$ variable are dependent, see fig(28.6). This defines a complex high-dimensional $p(\mathbf{x}, \mathbf{y})$ for which the conditional entropy is typically intractable.

28.5.1 The information maximisation algorithm

Consider

$$\mathrm{KL}(p(x|y)|q(x|y)) \geq 0 \tag{28.5.6}$$

This immediately gives a bound

$$\sum_x p(x|y) \log p(x|y) - \sum_x p(x|y) \log q(x|y) \geq 0 \tag{28.5.7}$$

Algorithm 29 IM algorithm

1: Choose a class of approximating distributions Q (for example factorised)
2: Initialise the parameters θ
3: **repeat**
4: For fixed $q(x|y)$, find

$$\theta^{new} = \underset{\theta}{\text{argmax}}\ \tilde{I}(X,Y) \tag{28.5.10}$$

5: For fixed θ,

$$q^{new}(x|y) = \underset{q(x|y)\in Q}{\text{argmax}}\ \tilde{I}(X,Y) \tag{28.5.11}$$

where Q is a chosen class of distributions.
6: **until** converged

$$\Rightarrow \sum_{x,y} p(y)p(x|y)\log p(x|y) \geq \sum_{x,y} p(x,y)\log q(x|y) \tag{28.5.8}$$

From the definition, the left of the above is $-H(X|Y)$. Hence

$$I(X,Y) \geq H(X) + \langle \log q(x|y)\rangle_{p(x,y)} \equiv \tilde{I}(X,Y) \tag{28.5.9}$$

From this lower bound on the mutual information we arrive at the *information maximisation* (IM) algorithm[20]. Given a distribution $p(x)$ and a parameterised distribution $p(y|x,\theta)$, we seek to maximise $\tilde{I}(X,Y)$ with respect to θ. A co-ordinate wise optimisation procedure is presented in algorithm(29).

The *Blahut-Arimoto algorithm* in information theory (see for example [181]) is a special case in which the optimal decoder

$$q(x|y) \propto p(y|x,\theta)p(x) \tag{28.5.12}$$

is used. In applications where the Blahut-Arimoto algorithm is intractable to implement, the IM algorithm can provide an alternative by restricting q to a tractable family of distributions (tractable in the sense that the lower bound can be computed).

The Blahnut-Arimoto algorithm is then analogous to the EM algorithm for Maximum Likelihood and guarantees a non-decrease of the mutual information at each stage of the update. Similarly, the IM procedure is analogous to a Generalised EM procedure and each step of the procedure cannot decrease the lower bound on the mutual information.

28.5.2 Linear Gaussian decoder

A special case of the IM framework is to use a linear Gaussian decoder

$$q(\mathbf{x}|\mathbf{y}) = \mathcal{N}(x|\mathbf{U}\mathbf{y},\boldsymbol{\Sigma}) \Rightarrow \log q(\mathbf{x}|\mathbf{y}) = (\mathbf{x}-\mathbf{U}\mathbf{y})^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\mathbf{U}\mathbf{y}) \tag{28.5.13}$$

Plugging this into the MI bound, equation (28.5.9), and optimising with respect to $\boldsymbol{\Sigma}$, and $\mathbf{U}$, we obtain

$$\boldsymbol{\Sigma} = \left\langle (\mathbf{x}-\mathbf{U}\mathbf{y})(\mathbf{x}-\mathbf{U}\mathbf{y})^{\mathsf{T}}\right\rangle, \qquad \mathbf{U} = \left\langle \mathbf{x}\mathbf{y}^{\mathsf{T}}\right\rangle\left\langle \mathbf{y}\mathbf{y}^{\mathsf{T}}\right\rangle^{-1} \tag{28.5.14}$$

where $\langle\cdot\rangle \equiv \langle\cdot\rangle_{p(\mathbf{x},\mathbf{y})}$. Plugging this into the MI bound we obtain

$$I(X,Y) \geq H(X) - \frac{1}{2}\log\det\left(\left\langle \mathbf{x}\mathbf{x}^{\mathsf{T}}\right\rangle - \left\langle \mathbf{x}\mathbf{y}^{\mathsf{T}}\right\rangle\left\langle \mathbf{y}\mathbf{y}^{\mathsf{T}}\right\rangle^{-1}\left\langle \mathbf{y}\mathbf{x}^{\mathsf{T}}\right\rangle\right) + \text{const.} \tag{28.5.15}$$

Up to irrelevant constants, this is equivalent to *Linsker's as-if-Gaussian approximation* to the Mutual Information, [173]. One can therefore view Linsker's approach as a special case of the IM algorithm restricted to linear-Gaussian decoders. In principle, one can therefore improve on Linsker's method by considering more powerful non-linear-Gaussian decoders. Applications of this technique to Neural systems are discussed in [20].

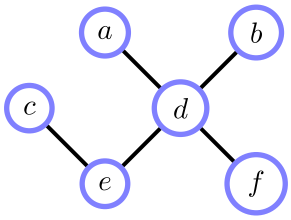


Figure 28.7: Classical Belief Propagation can be derived by considering how to compute the marginal of a variable on a MRF. In this case the marginal $p(d)$ depends on messages transmitted via the neighbours of d. By defining local messages on the links of the graph, a recursive algorithm for computing all marginals can be derived, see text.

28.6 Loopy Belief Propagation

Belief Propagation is a technique for marginal inference $p(x_i)$ for singly-connected distributions $p(x)$. There are different formulations of BP, the most modern treatment being the sum-product algorithm on the corresponding factor graph, as described in section(5.1.2). Note that the algorithm is purely local – the updates are unaware of the global structure of the graph, depending only on the local neighbourhood structure. This means that even if the graph is multiply-connected (it is loopy) one can still apply the algorithm and 'see what happens'. Provided the loops in the graph are relatively long, one may hope that running 'loopy' BP will converge to a good approximation of the true marginals. In general, this cannot be guaranteed, but when the method converges the results can be surprisingly accurate.

In the following we will show how loopy BP can also be motivated by a variational objective. To do so, the most natural connection is with the classical BP algorithm (rather than the factor graph sum-product) algorithm. For this reason we briefly describe below the classical BP approach.

28.6.1 Classical BP on an undirected graph

A more classical treatment can be derived by considering how to calculate a marginal in terms of messages on an undirected graph. Consider calculating the marginal $p(d) = \sum_{a,b,c,e,f} p(a, b, c, d, e, f)$ for the pairwise Markov network in fig(28.7). We denote both a node and its state by the same symbol, so that $\sum_b \phi(d, b)$ denotes summation over the states of the variable b. This results in a message $\lambda_{b\to d}(d)$ which contains information passing from node b to node d and is a function of the state of node d. This works as follows:

$$p(d) = \frac{1}{Z} \underbrace{\sum_b \phi(b, d)}_{\lambda_{b\to d}(d)} \underbrace{\sum_a \phi(a, d)}_{\lambda_{a\to d}(d)} \underbrace{\sum_f \phi(d, f)}_{\lambda_{f\to d}(d)} \underbrace{\sum_e \phi(d, e) \underbrace{\sum_c \phi(c, e)}_{\lambda_{c\to e}(e)}}_{\lambda_{e\to d}(d)} \tag{28.6.1}$$

where we define messages $\lambda_{n_1\to n_2}(n_2)$ sending information from node n_1 to node n_2 as a function of the state of node n_2. In general, a node x_i passes a message to node x_j via

$$\lambda_{x_i\to x_j}(x_j) = \sum_{x_i} \phi(x_i, x_j) \prod_{k\in \mathrm{ne}(i), k\neq j} \lambda_{x_k\to x_i}(x_i) \tag{28.6.2}$$

This algorithm is equivalent to the sum-product algorithm provided the graph is singly-connected.

28.6.2 Loopy BP as a variational procedure

A variational procedure that corresponds to loopy BP can be derived by considering the terms of a standard variational approximation based on the Kullback-Leibler divergence $\mathrm{KL}(q|p)$. For a pairwise MRF defined on potentials $\phi(x_i, x_j)$,

$$p(x) = \frac{1}{Z} \prod_{i\sim j} \phi(x_i, x_j) \tag{28.6.3}$$

and approximating distribution $q(x)$, the Kullback-Leibler bound is

$$\log Z \geq -\langle \log q(x)\rangle_{q(x)} + \underbrace{\sum_{i\sim j} \langle \log \phi(x_i, x_j)\rangle_{q(x)}}_{energy} \tag{28.6.4}$$

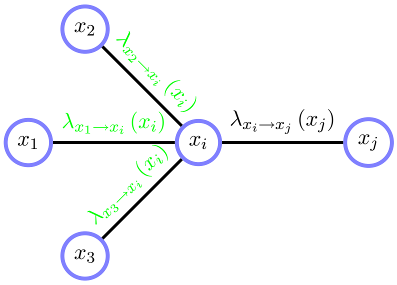


Figure 28.8: Loopy Belief Propagation. Once a node has received incoming messages from all neighbours (excluding the one it wants to send a message to), it may send an outgoing message to a neighbour:

$$\lambda_{x_i \to x_j}(x_j) = \sum_{x_i} \phi(x_i, x_j) \lambda_{x_1 \to x_i}(x_i) \lambda_{x_2 \to x_i}(x_i) \lambda_{x_3 \to x_i}(x_i)$$

Since

$$\langle \log \phi(x_i, x_j) \rangle_{q(x)} = \langle \log \phi(x_i, x_j) \rangle_{q(x_i, x_j)} \tag{28.6.5}$$

it's clear that each contribution to the energy depends on $q(x)$ only via the pairwise marginals $q(x_i, x_j)$. This suggests that these marginals should form the natural parameters of any approximation. Can we then find an expression for the entropy $-\langle \log q(x) \rangle_{q(x)}$ in terms of these pairwise marginals? Consider a case in which the required marginals are

$$q(x_1, x_2), q(x_2, x_3), q(x_3, x_4) \tag{28.6.6}$$

Given these marginals, the energy term is straightforward to compute, and we are left with requiring only the entropy of q. Either by appealing to the junction tree representation, or by straightforward algebra, one can show that we can uniquely express q in terms of the marginals as

$$q(x) = \frac{q(x_1, x_2) q(x_2, x_3) q(x_3, x_4)}{q(x_2) q(x_3)} \tag{28.6.7}$$

An intuitive way to arrive at this result is by examining the numerator of equation (28.6.7). The variable x_2 appears twice, as does the variable x_3 and, since any joint distribution cannot have replicated variables on the left of any conditioning, we must compensate for the additional x_2 and x_3 variables by dividing by these marginals. In this case, the entropy of $q(x)$ can be written as

$$H_q(x) = -\langle \log q(x) \rangle_{q(x)} = H_q(x_1, x_2) + H_q(x_2, x_3) + H_q(x_3, x_4) - H_q(x_2) - H_q(x_3) \tag{28.6.8}$$

More generally, chapter(6), any decomposable graph can be represented as

$$q(x) = \frac{\prod_c q(\mathcal{X}_c)}{\prod_s q(\mathcal{X}_s)} \tag{28.6.9}$$

where the $q(\mathcal{X}_c)$ are the marginals defined on cliques of the graph, with $\mathcal{X}_c$ being the variables of the clique, and the $q(\mathcal{X}_s)$ are defined on the separators (intersections of neighbouring cliques). The expression for the entropy of the distribution is then given by a sum of marginal entropies, minus the entropy of the marginals defined on the separators.

Bethe Free energy

Consider now a MRF corresponding to a non-decomposable graph, for example the 4-cycle

$$p(x) = \frac{1}{Z} \phi(x_1, x_2) \phi(x_2, x_3) \phi(x_3, x_4) \phi(x_4, x_1) \tag{28.6.10}$$

The energy requires therefore that we know

$$q(x_1, x_2), q(x_2, x_3), q(x_3, x_4), q(x_4, x_1) \tag{28.6.11}$$

Assuming that these marginals are given, can we find an expression for the entropy of the joint distribution $q(x_1, x_2, x_3, x_4)$ in terms of its pairwise marginals $q(x_i, x_j)$? In general this is not possible since the graph corresponding to the marginals contains loops (so that the junction tree representation would result in cliques greater than size 2). However, a simple 'no overcounting' approximation is to write

$$q(x) \approx \frac{q(x_1, x_2) q(x_2, x_3) q(x_3, x_4) q(x_4, x_1)}{q(x_1) q(x_2) q(x_3) q(x_4)} \tag{28.6.12}$$

subject to the constraints

$$\sum_{x_i} q(x_i, x_j) = q(x_j) \tag{28.6.13}$$

and similarly for the other marginals. An entropy approximation using this representation is therefore

$$H_q(x) \approx H_q(x_1, x_2) + H_q(x_2, x_3) + H_q(x_3, x_4) + H_q(x_1, x_4) - \sum_{i=1}^{4} H_q(x_i) \tag{28.6.14}$$

With this approximation the log partition function is known in statistical physics as the (negative) *Bethe free energy*. Our interest is then to maximise this expression with respect to the parameters $q(x_i, x_j)$ subject to marginal consistency constraints, $\sum_{x_i} q(x_i, x_j) = q(x_j)$. These may be enforced using Lagrange multipliers. One can write the Bethe free energy as

$$\mathcal{F}(q, \lambda) \equiv -\sum_{i \sim j} H_q(x_i, x_j) + \sum_i H_q(x_i) + \sum_{i \sim j} \langle \log \phi(x_i, x_j) \rangle_{q(x_i, x_j)} + \sum_{i \sim j} \lambda_{i,j} \left(q(x_i) - \sum_{x_i} q(x_i, x_j) \right) \tag{28.6.15}$$

where $i \sim j$ denotes the unique neighbouring edges on the graph (each edge is counted only once). This is no longer a bound on the log partition function since the entropy approximation is not a lower bound on the true entropy. The task is now to maximise this 'approximate bound' with respect to the parameters, namely all the pairwise marginals $q(x_i, x_j)$ and the Lagrange multipliers λ.

Belief Propagation

A simple scheme to maximise equation (28.6.15) is to use a fixed point iteration by equating the derivatives of the Bethe free energy with respect to the parameters $q(x_i, x_j)$ to zero, and likewise for the Lagrange multipliers. One may show that the resulting set of fixed point equations, on eliminating q, is equivalent to (undirected) Belief Propagation for which, in general, a node x_i passes a message to node x_j using

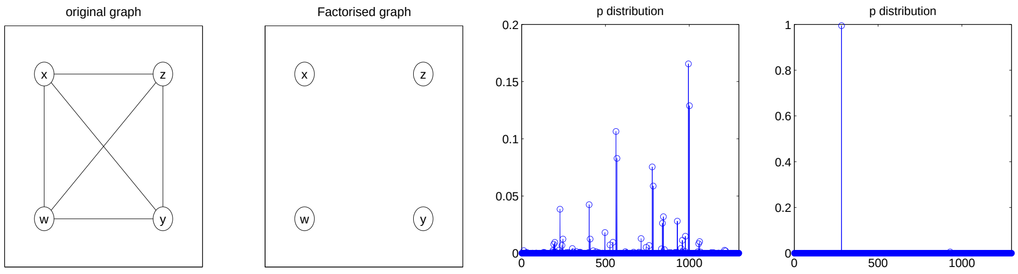
$$\lambda_{x_i \to x_j}(x_j) = \sum_{x_i} \phi(x_i, x_j) \prod_{k \in \text{ne}(i), k \neq j} \lambda_{x_k \to x_i}(x_i) \tag{28.6.16}$$

At convergence the marginal $p(x_i)$ is then approximated by

$$q(x_i) \propto \prod_{i \in \text{ne}(j)} \lambda_{x_j \to x_i}(x_i) \tag{28.6.17}$$

the prefactor being determined by normalisation. For a singly-connected distribution p, this message passing scheme converges and the marginal corresponds to the exact result. For non-singly connected structures (loopy), running this loopy Belief Propagation will generally result in an approximation. One can therefore view Loopy BP as a form of approximate variational inference. Naturally, we can dispense with the Bethe free energy if desired and run the associated Loopy Belief Propagation algorithm directly on the undirected graph.

The convergence of Loopy Belief Propagation which can be heavily dependent on the topology of the graph and also the message updating schedule[288, 199]. The potential benefit of the Bethe free energy viewpoint is that it gives an objective that is required to be optimised, opening up the possibility of more general optimisation techniques than BP. The so-called double-loop techniques iteratively isolate convex contributions to the Bethe Free energy, interleaved with concave contributions. At each stage, the resulting optimisiations can be carried out efficiently[297, 131].



(a) (b) (c)

Figure 28.9: **(a)**: The Markov network (left) that we wish to approximate the marginals $p(w), p(x), p(y), p(z)$ for. All tables are drawn from a uniform distribution raised to a power α. On the right is shown the naive mean field approximation factorised structure. **(b)**: There are $6^4 = 1295$ states of the distribution. Shown is a randomly sampled distribution for $\alpha = 5$ which has many isolated peaks, suggesting the distribution is far from factorised. In this case the MF and Gibbs sampling approximations may perform poorly. **(c)**: As α is increased to 25, typically only one state of the distribution dominates. See `demoMFBPGibbs.m`.

Validity of loopy belief propagation

For a MRF which has a loop, computationally this means that a perturbation in a variable on the loop eventually reverberates to the same variable. However, if there are a large number of variables in the loop, and the individual neighbouring links are not all extremely strong, the numerical effect of the loop is small in the sense that influence of the variable on itself is negligible. In such cases one would expect the loopy BP approximation to be accurate. An area of particular success for Loopy Belief Propagation inference is in error correction based on low density parity check codes, which are designed to have this long-loop property[180]. In many examples of practical interest (for example an MRF with nearest neighbour interactions on a lattice), however, loops can be very short. In such cases a naive implementation of Loopy BP will fail. A natural extension is to cluster variables to alleviate some of the issues arising from strong local dependencies; this technique is called the *Kikuchi* or *Cluster Variation method*[151]. More elaborate ways of clustering variables can be considered using *region graphs*[289].

Example 118. The file `demoMFBPGibbs.m` compares the performance of naive Mean Field theory, Belief Propagation and unstructured Gibbs sampling on marginal inference in a pairwise Markov network

$$p(w, x, y, z) = \phi_{wx}(w, x)\phi_{wy}(w, y)\phi_{wz}(w, z)\phi_{xy}(x, y)\phi_{xz}(x, z)\phi_{yz}(y, z) \tag{28.6.18}$$

in which all variables take 6 states. In the experiment the tables are selected from a uniform distribution raised to a power α. For α small, all the tables are essentially flat and therefore the variables become independent, a situation for which MF, BP and Gibbs sampling are ideally suited. As α is increased, say to 5, the dependencies amongst the variables increase and the methods perform worse, especially MF and Gibbs. As α is increased to say 25, the distribution becomes sharply peaked around a single state, such that the posterior is effectively factorised, see fig(28.9). This suggests that a MF approximation (and also Gibbs sampling) should work well. However, finding this state is computationally difficult and both methods often get stuck in local minima. Belief propagation seems less susceptible to being trapped in local minima in this regime and tends to outperform both MF and Gibbs sampling.

28.7 Expectation Propagation

As we've seen for Switching Kalman Filters, message passing routines fail when the messages are not representable in a compact form. This limits BP to cases such as discrete networks, or more generally exponential family messages. Expectation Propagation extends the applicability of BP to cases in which

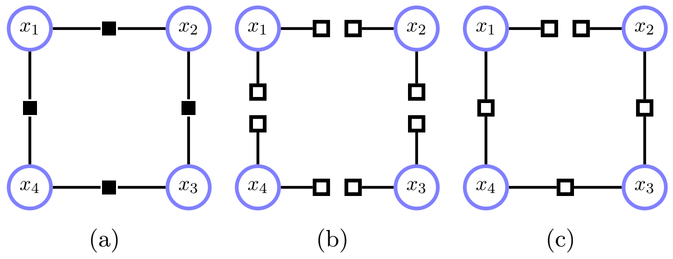


Figure 28.10: **(a)**: Multiply-connected factor graph representing $p(x)$. **(b)**: Expectation propagation approximates (a) in terms of a tractable factor graph. The open factors indicate the factors are parameters of the approximation. The basic EP approximation is to replace all factors in $p(x)$ by product factors. **(c)**: Tree structured EP.

the messages are not in the exponential family by projecting the messages back to the exponential family at each stage. This projection is obtained by using a Kullback-Leibler measure[193, 243, 195].

In order to explain how EP works it is useful to look at a specific example. Consider a distribution of the form

$$p(x) = \frac{1}{Z}\prod_i \phi_i(\mathcal{X}) \tag{28.7.1}$$

In EP one identifies those factors $\phi_i(\mathcal{X})$ which, if replaced by simpler factors $\tilde{\phi}_i(\mathcal{X})$, would render the distribution $\tilde{p}(x)$ tractable. One then sets any free parameters of $\tilde{\phi}_i(\mathcal{X})$ by minimising the Kullback-Leibler divergence $\text{KL}(p|\tilde{p})$. For example, consider a pairwise MRF

$$p(x) = \frac{1}{Z}\phi_{1,2}(x_1,x_2)\phi_{2,3}(x_2,x_3)\phi_{3,4}(x_3,x_4)\phi_{4,1}(x_4,x_1) \tag{28.7.2}$$

with Factor Graph as depicted in fig(28.10a). If we replaced all terms $\phi_{ij}(x_i,x_j)$ by approximate factors $\tilde{\phi}_{ij}(x_i)\tilde{\phi}_{ij}(x_j)$ then the resulting joint distribution $\tilde{p}$ would be factorised and hence tractable. Since the variable x_i appears in more than one term from $p(x)$, we need to index the approximation factors. A convenient way to do this is

$$\tilde{p} = \frac{1}{\tilde{Z}}\tilde{\phi}_{2\to1}(x_1)\,\tilde{\phi}_{1\to2}(x_2)\,\tilde{\phi}_{3\to2}(x_2)\,\tilde{\phi}_{2\to3}(x_3)\,\tilde{\phi}_{4\to3}(x_3)\,\tilde{\phi}_{3\to4}(x_4)\,\tilde{\phi}_{1\to4}(x_4)\,\tilde{\phi}_{4\to1}(x_1) \tag{28.7.3}$$

which is represented in fig(28.10b).

The idea in EP is now to determine the optimal approximation term by the self-consistent requirement that, on replacing it with its exact form, there is no difference to the marginal of $\tilde{p}$. For example, let's try to set the approximation parameters $\tilde{\phi}_{3\to2}(x_2)$ and $\tilde{\phi}_{2\to3}(x_3)$. To do so we first replace the contribution $\tilde{\phi}_{3\to2}(x_2)\,\tilde{\phi}_{2\to3}(x_3)$ by $\phi(x_2,x_3)$. This gives

$$\tilde{p}_* = \frac{1}{\tilde{Z}_*}\tilde{\phi}_{2\to1}(x_1)\,\tilde{\phi}_{1\to2}(x_2)\,\phi(x_2,x_3)\tilde{\phi}_{4\to3}(x_3)\,\tilde{\phi}_{3\to4}(x_4)\,\tilde{\phi}_{1\to4}(x_4)\,\tilde{\phi}_{4\to1}(x_1) \tag{28.7.4}$$

Now consider the Kullback-Leibler divergence between this distribution and our approximation,

$$\text{KL}(\tilde{p}_*|\tilde{p}) = \langle \log \tilde{p}_*\rangle_{\tilde{p}_*} - \langle \log \tilde{p}\rangle_{\tilde{p}_*} \tag{28.7.5}$$

Since our interest is in updating $\tilde{\phi}_{3\to2}(x_2)$ and $\tilde{\phi}_{2\to3}(x_3)$, we isolate the contribution from these parameters in the Kullback-Leibler divergence which gives

$$\text{KL}(\tilde{p}_*|\tilde{p}) = \log \tilde{Z} - \left\langle \log \tilde{\phi}_{3\to2}(x_2)\,\tilde{\phi}_{2\to3}(x_3)\right\rangle_{\tilde{p}_*(x_2,x_3)} + \text{const.} \tag{28.7.6}$$

Also, since $\tilde{p}$ is factorised, up to a constant proportionality factor, the dependence of $\tilde{Z}$ on $\tilde{\phi}_{3\to2}(x_2)$ and $\tilde{\phi}_{2\to3}(x_3)$ is

$$\tilde{Z} \propto \sum_{x_2}\tilde{\phi}_{1\to2}(x_2)\,\tilde{\phi}_{3\to2}(x_2)\sum_{x_3}\tilde{\phi}_{2\to3}(x_3)\,\tilde{\phi}_{4\to3}(x_3) \tag{28.7.7}$$

Differentiating the Kullback-Leibler divergence equation (28.7.6) with respect to $\tilde{\phi}_{3\to 2}(x_2)$ and equating to zero, we obtain

$$\frac{\tilde{\phi}_{1\to 2}(x_2)\,\tilde{\phi}_{3\to 2}(x_2)}{\sum_{x_2}\tilde{\phi}_{1\to 2}(x_2)\,\tilde{\phi}_{3\to 2}(x_2)} = \tilde{p}_*(x_2) \tag{28.7.8}$$

Similarly, optimising *w.r.t.* $\tilde{\phi}_{2\to 3}(x_3)$ gives

$$\frac{\tilde{\phi}_{2\to 3}(x_3)\,\tilde{\phi}_{4\to 3}(x_3)}{\sum_{x_3}\tilde{\phi}_{2\to 3}(x_3)\,\tilde{\phi}_{4\to 3}(x_3)} = \tilde{p}_*(x_3) \tag{28.7.9}$$

These equations only determine the approximation factors up to a proportionality constant. We can therefore write the optimal updates as

$$\tilde{\phi}_{3\to 2}(x_2) = z_{3\to 2}\frac{\tilde{p}_*(x_2)}{\tilde{\phi}_{1\to 2}(x_2)} \tag{28.7.10}$$

and

$$\tilde{\phi}_{2\to 3}(x_3) = z_{2\to 3}\frac{\tilde{p}_*(x_3)}{\tilde{\phi}_{4\to 3}(x_3)} \tag{28.7.11}$$

where $z_{3\to 2}$ and $z_{2\to 3}$ are proportionality terms. We can determine the proportionalities by the requirement that the term approximation $\tilde{\phi}_{3\to 2}(x_2)\,\tilde{\phi}_{2\to 3}(x_3)$ has the same effect on the normalisation of $\tilde{p}$ as it has on $\tilde{p}_*$. That is

$$\begin{aligned}\sum_{x_1,x_2,x_3,x_4} &\tilde{\phi}_{2\to 1}(x_1)\,\tilde{\phi}_{1\to 2}(x_2)\,\tilde{\phi}_{3\to 2}(x_2)\,\tilde{\phi}_{2\to 3}(x_3)\,\tilde{\phi}_{4\to 3}(x_3)\,\tilde{\phi}_{3\to 4}(x_4)\,\tilde{\phi}_{1\to 4}(x_4)\,\tilde{\phi}_{4\to 1}(x_1)\\ &= \sum_{x_1,x_2,x_3,x_4}\tilde{\phi}_{2\to 1}(x_1)\,\tilde{\phi}_{1\to 2}(x_2)\,\phi(x_2,x_3)\tilde{\phi}_{4\to 3}(x_3)\,\tilde{\phi}_{3\to 4}(x_4)\,\tilde{\phi}_{1\to 4}(x_4)\,\tilde{\phi}_{4\to 1}(x_1)\end{aligned} \tag{28.7.12}$$

which, on substituting in the updates equation (28.7.10) and equation (28.7.11) reduces to

$$z_{2\to 3}z_{3\to 2} = \frac{z^*_{2,3}}{\tilde{z}_{2,3}} \tag{28.7.13}$$

where

$$\tilde{z}_{2,3} = \sum_{x_2,x_3}\tilde{\phi}_{1\to 2}(x_2)\,\frac{\tilde{p}_*(x_2)}{\tilde{\phi}_{1\to 2}(x_2)}\,\frac{\tilde{p}_*(x_3)}{\tilde{\phi}_{4\to 3}(x_3)}\tilde{\phi}_{4\to 3}(x_3) \tag{28.7.14}$$

and

$$z^*_{2,3} = \sum_{x_2,x_3}\tilde{\phi}_{1\to 2}(x_2)\,\phi(x_2,x_3)\tilde{\phi}_{4\to 3}(x_3) \tag{28.7.15}$$

Any choice of local normalisations $z_{2\to 3}$, $z_{3\to 2}$ that satisfies equation (28.7.13) suffices to ensure that the scale of the term approximation matches. For example, one may set

$$z_{2\to 3} = z_{3\to 2} = \sqrt{\frac{z^*_{2,3}}{\tilde{z}_{2,3}}} \tag{28.7.16}$$

Once set, an approximation for the global normalisation constant of p is

$$Z \approx \tilde{Z} \tag{28.7.17}$$

The above gives a procedure for updating the terms $\tilde{\phi}_{3\to 2}(x_2)$ and $\tilde{\phi}_{2\to 3}(x_3)$. One then chooses another term and replaces it with its approximation, until the parameters of the approximation converge. The generic procedure is outlined in algorithm(30).

Algorithm 30 Expectation Propagation: approximation of $p(x) = \frac{1}{Z}\prod_i \phi_i(\mathcal{X}_i)$.

1: Decide on a set of terms $\phi_i(\mathcal{X}_i)$ to replace with $\tilde{\phi}_i(\mathcal{X}_i)$ in order to reveal a tractable distribution

$$\tilde{p}(x) = \frac{1}{\tilde{Z}}\prod_i \tilde{\phi}_i(\mathcal{X}_i) \tag{28.7.18}$$

2: Initialise the all parameters $\tilde{\phi}_i(\mathcal{X}_i)$.
3: **repeat**
4: Select a term $\phi_i(\mathcal{X}_i)$ from p to update.
5: Replace the term $\phi_i(\mathcal{X}_i)$ by the tractable term $\tilde{\phi}_i(\mathcal{X}_i)$ to form

$$\tilde{\phi}_* \equiv \frac{\prod_j \tilde{\phi}_j(\mathcal{X}_j)}{\tilde{\phi}_i(\mathcal{X}_i)}\phi_i(\mathcal{X}_i) = \phi_i(\mathcal{X}_i)\prod_{j\neq i}\tilde{\phi}_j(\mathcal{X}_j) \tag{28.7.19}$$

6: Find the parameters of $\tilde{\phi}_i(\mathcal{X}_i)$ by

$$\tilde{\phi}_i(\mathcal{X}_i) \propto \underset{\tilde{\phi}_i(\mathcal{X}_i)}{\text{argmin}}\ \text{KL}(\tilde{p}_*|\tilde{p}) \tag{28.7.20}$$

where

$$\tilde{p}_* \propto \tilde{\phi}_*, \qquad \tilde{p}(x) \propto \prod_i \tilde{\phi}_i(\mathcal{X}_i) \tag{28.7.21}$$

7: Set any proportionality terms of $\tilde{\phi}_i(\mathcal{X}_i)$ by requiring

$$\sum_x \phi_i(\mathcal{X}_i)\prod_{j\neq i}\tilde{\phi}_j(\mathcal{X}_j) = \sum_x\prod_j \tilde{\phi}_j(\mathcal{X}_j) \tag{28.7.22}$$

8: **until** converged
9: return

$$\tilde{p}(x) = \frac{1}{\tilde{Z}}\prod_i \tilde{\phi}_i(\mathcal{X}_i), \qquad \tilde{Z} = \sum_x\prod_i \tilde{\phi}_i(\mathcal{X}_i) \tag{28.7.23}$$

as an approximation to $p(x)$, where $\tilde{Z}$ approximates the normalisation constant Z.

Comments on EP

- For the MRF example above, EP corresponds to Belief Propagation (the sum-product form on the factor graph). This is intuitively clear since in both EP and BP the product of messages incoming to a variable is proportional to the approximation of the marginal of that variable. A slight difference is the schedule. In EP all messages corresponding to a term approximation are updated simultaneously (in the above $\tilde{\phi}_{3\to 2}(x_2)$ and $\tilde{\phi}_{2\to 3}(x_3)$), whereas in BP they are updated in arbitrary order.

- EP is a useful extension of BP to cases in which the BP messages cannot be easily represented. In the case that the approximating distribution $\tilde{p}$ is in the exponential family, the minimal Kullback-Leibler criterion equates to matching moments of the approximating distribution to p^*. See [243] for a more detailed discussion.

- In general there is no need to replace all terms in the joint distribution with factorised approximations. One only needs that the resulting approximating distribution is tractable; this results in a *structured Expectation Propagation* algorithm, see fig(28.10c).

- EP and its extensions are closely related to other variational procedures such as Tree-Reweighting[284] and fractional EP[292] designed to compensate for message overcounting effects.

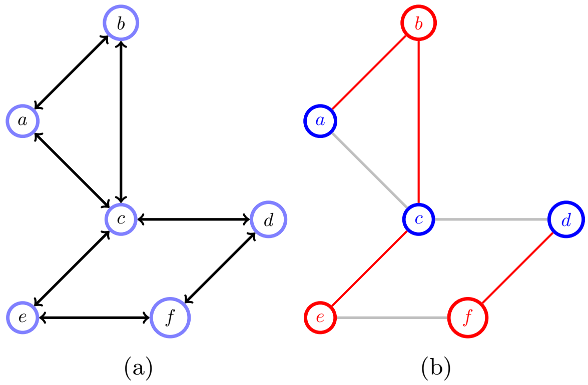


Figure 28.11: **(a)**: A graph with bidirectional weights $w_{ij} = w_{ji}$. **(b)**: A graph cut partitions the nodes into two groups $\mathcal{S}$ (blue) and $\mathcal{T}$ (red). The weight of the cut is the sum of the edge weights from $\mathcal{S}$ (blue) to $\mathcal{T}$ (red). Intuitively, it is clear that after assigning nodes to state 1 (for blue) and 0 (red) that the weight of the cut corresponds to the summed weights of neighbours in different states. Here we highlight those weight contributions. The non-highlighted edges do not contribute to the cut weight. Note that only one of the edge directions contributes to the cut.

28.8 MAP for MRFs

Consider a pairwise MRF $p(x) \propto e^{E(x)}$ with

$$E(x) \equiv \sum_{i \sim j} f(x_i, x_j) + \sum_i g(x_i, x_i^0) \tag{28.8.1}$$

where $i \sim j$ denotes neighbouring variables. Here the terms $f(x_i, x_j)$ represent pairwise interactions. The terms $g(x_i, x_i^0)$ represent unary interactions, but for convenience can be written as pairwise interactions for fixed (non-variable) x^0. Typically the term $f(x_i, x_j)$ is used to ensure that neighbouring variables x_i and x_j are in similar states; the term $g(x_i, x_i^0)$ is used to prefer x_i to be close to a desired state x_i^0. Such models have application in areas such as Computer Vision and image restoration in which an observed noisy image x^0 is to be cleaned, fig(28.3). To do so we seek a clean image x for which each clean pixel value x_i is close to the observed noisy pixel value x_i^0, whilst being in a similar state to its clean neighbours.

28.8.1 MAP assignment

The MAP assignment of a set of variables $x_1, \ldots, x_D$ corresponds to that joint x that maximises $E(x)$. For a general graph connectivity we cannot naively exploit dynamic programming intuitions to find an exact solution since the graph is loopy. A simple algorithm is to first initialise all x at random. Then select a variable x_i and find the state of x_i that maximally improves $E(x)$, keeping all other variables fixed. One then repeats this selection and local maximal state computation until convergence. This is called *Iterated Conditional Modes*[36]. Due to the Markov properties its clear that we can improve on this ICM method by simultaneously optimising all variables conditioned on their respective Markov blankets (similar to the approach used in black-white sampling). Another improvement is to update only a subset of the variables, where the subset has the form of singly-connected structure. By recursively clamping variables to reveal a singly-connected structure on un-clamped variables, one may find an approximate solution by solving a sequence of tractable problems.

Remarkably, in the special case of binary variables and positive w discussed below, an efficient exact algorithm exists for finding the MAP state, regardless of the topology.

28.8.2 Attractive binary MRFs

Consider finding the MAP of a MRF with binary variables $\text{dom}(x_i) = \{0, 1\}$ and positive connections $w_{ij} \geq 0$. In this case our task is to find the assignment x that maximises

$$E(x) \equiv \sum_{i \sim j} w_{ij} \mathbb{I}[x_i = x_j] + \sum_i c_i x_i \tag{28.8.2}$$

where $i \sim j$ denotes neighbouring variables and $c_i \in \Re$. Note that for binary variables $x_i \in \{0, 1\}$,

$$\mathbb{I}[x_i = x_j] = x_i x_j + (1 - x_i)(1 - x_j) \tag{28.8.3}$$

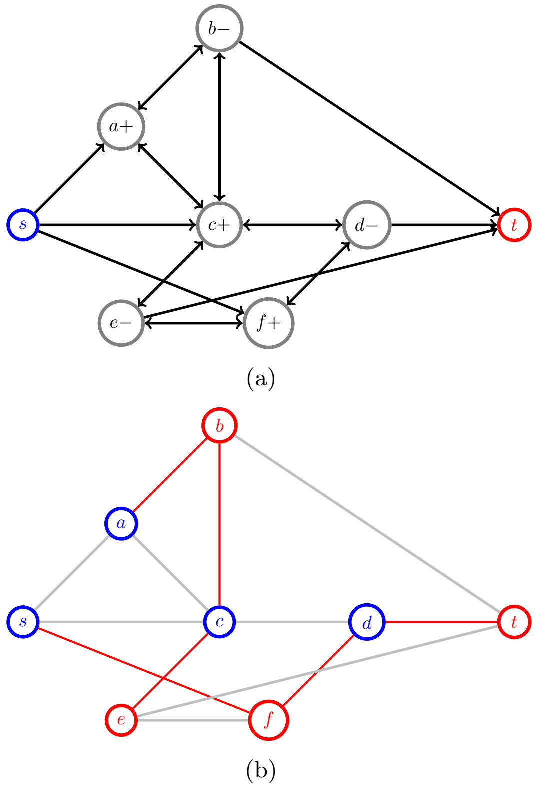


Figure 28.12: **(a)**: A Graph with bidirectional weights $w_{ij} = w_{ji}$ augmented with a source node s and sink node t. Each node has a corresponding bias whose sign is indicated. The source node is linked to the nodes corresponding to positive bias, and the nodes with negative bias to the sink. **(b)**: A graph cut partitions the nodes into two groups $\mathcal{S}$ (blue) and $\mathcal{T}$ (red), where $\mathcal{S}$ is the union of the source node and nodes in state 1, $\mathcal{T}$ is the union of the sink node and nodes in state 0. The weight of the cut is the sum of the edge weights from $\mathcal{S}$ (blue) to $\mathcal{T}$ (red). The red lines indicate contributions to the cut, and can be considered penalties since we wish to find the minimal cut. For example a being in state 1 (blue) does not incur a penalty since $c_a > 0$; on the other hand, variable f being in state 0 (red) incurs a penalty since $c_f > 0$.

For this particular case an efficient MAP algorithm exists for arbitrary topology of w[118]. The algorithm first translates the MAP assignment problem into an equivalent min s-t-cut problem[39], for which efficient algorithms exist. In min s-t-cut, we need a graph with positive weights on the edges. This is clearly satisfied if $w_{ij} > 0$, although the bias terms $\sum_i c_i x_i$ need to be addressed.

Dealing with the bias terms

To translate the MAP assignment problem to a min-cut problem we need to deal with the additional linear terms $\sum_i c_i x_i$. First consider the effect of including a new node x_* and connecting this to each existing node i with weight c_i. This adds then a term

$$\sum_i c_i \mathbb{I}\left[x_i = x_*\right] = \sum_i c_i \left(x_i x_* + (1 - x_i)(1 - x_*)\right) \tag{28.8.4}$$

If we set x_* in state 1, this will then add terms

$$\sum_i c_i x_i \tag{28.8.5}$$

Otherwise, if we set x_* in state 0 we obtain

$$\sum_i c_i (1 - x_i) = -\sum_i c_i x_i + \mathsf{const.} \tag{28.8.6}$$

Since our requirement is that we need the weights to be positive we see that we can achieve this by defining two additional nodes. We define a source node x_s, set to state 1 and connect it to those x_i which have positive c_i, defining $w_{si} = c_i$. In addition we define a sink node $x_t = 0$ and connect all nodes with negative c_i, to x_t, using weight $w_{it} = -c_i$, (which is therefore positive).

For the source node clamped to $x_s = 1$ and the sink node to $x_t = 0$, then including the source and sink, we have

$$E(x) = \sum_{i \sim j} w_{ij} \mathbb{I}\left[x_i = x_j\right] + \mathsf{const.} \tag{28.8.7}$$

is equal to the energy function, equation (28.8.2).

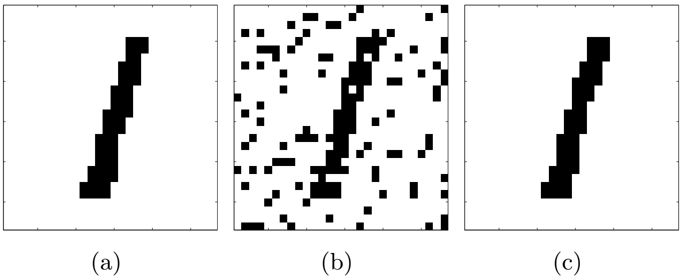


Figure 28.13: **(a)**: Clean image. **(b)**: Noisy image. **(c)**: Restored image using ICM. See `demoMRFclean.m`.

Definition 115 (Graph Cut)**.** For a graph G with vertices $v_1, \ldots, v_D$, and weights $w_{ij} > 0$ a cut is a partition of the vertices into two disjoint groups, called $\mathcal{S}$ and $\mathcal{T}$. The weight of a cut is then defined as the sum of the weights that leave $\mathcal{S}$ and land in $\mathcal{T}$, see fig(28.11).

The weight of a cut corresponds to the sum of weights between mismatched neighbours, see fig(28.11b). That is,

$$cut(x) = \sum_{i\sim j} w_{ij}\mathbb{I}\left[x_i \neq x_j\right] \tag{28.8.8}$$

Since $\mathbb{I}\left[x_i \neq x_j\right] = 1 - \mathbb{I}\left[x_i = x_j\right]$, we can define the weight of the cut equivalently as

$$cut(x) = \sum_{i\sim j} w_{ij}\left(1 - \mathbb{I}\left[x_i = x_j\right]\right) = -\sum_{i\sim j} w_{ij}\mathbb{I}\left[x_i = x_j\right] + \mathsf{const.} \tag{28.8.9}$$

so that the minimal cut assignment will correspond to maximising $\sum_{i\sim j} w_{ij}\mathbb{I}\left[x_i = x_j\right]$. In the MRF case, our translation into a weighted graph with positive interactions then requires that we identify the source and all other variables assigned to state 1 with $\mathcal{S}$, and the sink and all variables in state 0 with $\mathcal{T}$, see fig(28.12). A fundamental result is that the min s-t-cut solution corresponds to the max-flow solution from the source s to the sink t, see for example [39]. There are efficient algorithms for max-flow, see for example [47], which take $O\left(D^3\right)$ operations or less. This means that one can find the exact MAP assignment of an attractive binary MRF efficiently in $O\left(D^3\right)$ operations. In `MaxFlow.m` we implement the Ford-Fulkerson (Edmonds-Karp-Dinic breadth first search variant)[86], see also exercise(252).

Example 119 (Analysing dirty pictures)**.** In fig(28.13) we present a noisy binary y image that we wish to clean. To do so we use an objective

$$E(x) = \sum_{ij} w_{ij}\mathbb{I}\left[x_i = x_j\right] + \sum_{i}\mathbb{I}\left[x_i = y_i\right] \tag{28.8.10}$$

The variables x_i, $i = 1, \ldots, 784$ are defined on a 28×28 grid and where $w_{ij} = 10$ if x_i and x_j are neighbours on the grid. Using

$$\mathbb{I}\left[x_i = y_i\right] = x_i\left(2y_i - 1\right) + \mathsf{const.} \tag{28.8.11}$$

we have a standard binary MRF MAP problem with 'bias' $b = 2y - \mathbf{1}$. Once can then find the exact optimal x by the min-cut procedure. However, our implementation of this is slow and instead we use the simpler ICM algorithm, with results as shown in fig(28.13).

(a) (b) (c)

Figure 28.14: **(a)**: Noisy image (left half) with restored image (right half). **(b)**: Noisy image. **(c)**: Restored image. In both cases the α-expansion method was used, with suitable interactions w and bias c to ensure reasonable results. From [47].

28.8.3 Potts model

An extension of the previous model is to the case when the variables are non-binary, which is termed the *Potts model*:

$$E(x) = \sum_{i \sim j} w_{ij} \mathbb{I}\left[x_i = x_j\right] + \sum_i c_i \mathbb{I}\left[x_i = x_i^0\right] \tag{28.8.12}$$

where $w_{ij} > 0$ and the x_i^0 are known. This model has immediate application in non-binary image restoration, and also in clustering based on a similarity score. Whilst no efficient exact algorithm is known, a useful approach is to approximate the problem as a sequence of binary problems.

Potts to binary MRF translation

Consider the *α-expansion* representation

$$x_i = s_i \alpha + (1 - s_i) x_i^{old} \tag{28.8.13}$$

where $s_i \in \{0, 1\}$ and for a given $\alpha \in \{0, 1, 2, \ldots, N\}$. This restricts x_i to be either the state x_i^{old} or α, depending on the binary variable s_i. Using a new binary variable s we can therefore restrict x to a subpart of the full space and write a new objective function in terms of s alone:

$$E(s) = \sum_{i \sim j} w'_{ij} \mathbb{I}\left[s_i = s_j\right] + \sum_i c'_i s_i + \mathsf{const.} \tag{28.8.14}$$

for $w'_{ij} > 0$. This new problem is of the form of an attractive binary MRF which can be solved exactly using the graph cuts procedure. The idea is then to choose another α value (at random) and then find the optimal s for the new α. In this way we are guaranteed to iteratively increase E.

For a given α and x^{old}, the transformation of the Potts model objective is given by using $s_i \in \{0, 1\}$ and considering

$$\begin{aligned}
\mathbb{I}\left[x_i = x_j\right] &= \mathbb{I}\left[s_i \alpha + (1 - s_i) x_i^{old} = s_j \alpha + (1 - s_j) x_j^{old}\right] \\
&= (1 - s_i)(1 - s_j) \mathbb{I}\left[x_i^{old} = x_j^{old}\right] + (1 - s_i) s_j \mathbb{I}\left[x_i^{old} = \alpha\right] + s_i (1 - s_j) \mathbb{I}\left[x_j^{old} = \alpha\right] + s_i s_j \\
&= s_i s_j u_{ij} + a_i s_i + b_j s_j + \mathsf{const.}
\end{aligned} \tag{28.8.15}$$

with

$$u_{ij} \equiv 1 - \mathbb{I}\left[x_i^{old} = \alpha\right] - \mathbb{I}\left[x_j^{old} = \alpha\right] + \mathbb{I}\left[x_i^{old} = x_j^{old}\right] \tag{28.8.16}$$

and similarly defined a_i, b_i. By enumeration it is straightforward to show that u_{ij} is either 0, 1 or 2. Using the mathematical identity

$$s_i s_j = \frac{1}{2}\left(\mathbb{I}\left[s_i = s_j\right] + s_i + s_j - 1\right) \tag{28.8.17}$$

we can write,

$$\mathbb{I}[x_i = x_j] = \frac{u_{ij}}{2}\left(\mathbb{I}[s_i = s_j] + s_i + s_j\right) + a_i s_i + b_j s_j + \text{const.} \tag{28.8.18}$$

Hence terms $w_{ij}\mathbb{I}[x_i = x_j]$ translate to positive interaction terms $\mathbb{I}[s_i = s_j]\, w_{ij}u_{ij}/2$. All the unary terms are easily exactly mapped into corresponding unary terms $c'_i s_i$ for c'_i defined as the sum of all unary terms in s_i. This shows that the positive interaction w_{ij} in terms of the original variables x maps to a positive interaction in the new variables s. Hence we can find the maximal state of s using a graph cut algorithm. A related (though different) procedure is outlined in [48].

Example 120 (Potts model for image reconstruction)**.** An example image restoration problem for nearest neighbour interactions on a pixel lattice and suitably chosen w, c is given in fig(28.14). The images are non-binary and therefore the optimal MAP assignment cannot be computed exactly in an efficient way. The alpha-expansion technique was used here combined with an efficient min-cut approach, see [47] for details.

28.9 Further Reading

Approximate inference is a highly active research area and increasingly links to convex optimisation[46] are being developed. See [284] for a general overview and [244] for recent application of convex optimisation to approximate inference in a practical machine learning application.

28.10 Code

`LoopyBP.m`: Loopy Belief Propagation (Factor Graph formalism)
`demoLoopyBP.m`: Demo of loopy Belief Propagation
`demoMFBPGibbs.m`: Comparison of Mean Field, Belief Propagation and Gibbs sampling

`demoMRFclean.m`: Demo of analysing a dirty picture
`MaxFlow.m`: Max-Flow Min-Cut algorithm (Ford-Fulkerson)
`binaryMRFmap.m`: Optimising a binary MRF

28.11 Exercises

Exercise 252. *For the max-flow-min-cut problem, under the convention that the source node x_s is clamped to state 1, and the sink node x_t to state 0, a standard min-cut algorithm returns that joint x which minimises*

$$\sum_{ij} w_{ij}\mathbb{I}[x_i = 1]\,\mathbb{I}[x_j = 0] \tag{28.11.1}$$

Explain how this can be written in the form

$$\sum_{ij} \tilde{w}_{ij}\mathbb{I}[x_i \neq x_j] \tag{28.11.2}$$

Exercise 253. *Using* BRMLtoolbox*, write a routine* `KLdiv(q,p)` *that returns the Kullback-Leibler divergence between two discrete distributions q and p defined as potentials* `q` *and* `p`*.*

Exercise 254. *The file* `p.mat` *contains a distribution $p(x, y, z)$ on ternary state variables. Using* BRMLtoolbox*, find the best approximation $q(x, y)q(z)$ that minimises the Kullback-Leibler divergence $KL(q|p)$ and state the value of the minimal Kullback-Leibler divergence for the optimal q.*

Exercise 255. *Consider the pairwise MRF defined on a 2×2 lattice, as given in* `pMRF.mat`*. Using* BRMLtoolbox*,*

1. *Find the optimal fully factorised approximation $\prod_{i=1}^4 q_i^{BP}$ by Loopy Belief Propagation, based on the factor graph formalism.*

2. *Find the optimal fully factorised approximation $\prod_{i=1}^4 q_i^{MF}$ by solving the variational Mean Field equations.*

3. *By pure enumeration, compute the exact marginals p_i.*

4. *Averaged over all 4 variables, compute the mean expected deviation in the marginals*

$$\frac{1}{4}\sum_{i=1}^{4}\frac{1}{2}\sum_{j=1}^{2}|q_i(x=j)-p_i(x=j)|$$

for both the BP and MF approximations, and comment on your results.

Exercise 256. *In* `LoopyBP.m` *the message schedule is chosen at random. Modify the routine to choose a schedule using a forward-reverse elimination sequence on a random spanning tree.*

Exercise 257 (Double Integration Bounds)**.** *Consider a bound*

$$f(x) \geq g(x) \tag{28.11.3}$$

Then for

$$\tilde{f}(x) \equiv \int_a^x f(x)dx, \qquad \tilde{g}(x) \equiv \int_a^x g(x)dx \tag{28.11.4}$$

Show that:

1. $$\tilde{f}(x) \geq \tilde{g}(x), \qquad \text{for } x \geq a \tag{28.11.5}$$

2. $$\hat{f}(x) \geq \hat{g}(x) \qquad \text{for all } x \tag{28.11.6}$$

   where

   $$\hat{f}(x) \equiv \int_a^x \tilde{f}(x)dx, \qquad \hat{g}(x) \equiv \int_a^x \tilde{g}(x)dx \tag{28.11.7}$$

The significance is that this double integration (or summation in the case of discrete variables) is a general procedure for generating a new bound from an existing bound [169].

Exercise 258. *Starting from*

$$e^x \geq 0 \tag{28.11.8}$$

and using the double integration procedure, show that

$$e^x \geq e^a(1 + x - a)$$

1. *By replacing $x \to \mathbf{s}^\mathsf{T}\mathbf{W}\mathbf{s}$ for $\mathbf{s} \in \{0,1\}^D$, and $a \to \mathbf{h}^\mathsf{T}\mathbf{s}$ derive a bound on the partition function of a Boltzmann distribution*

   $$Z = \sum_{\mathbf{s}} e^{\mathbf{s}^\mathsf{T}\mathbf{W}\mathbf{s}} \tag{28.11.9}$$

2. *Show that this bound is equivalent to the Mean Field bound on the partition function.*

3. *Discuss how one can generate tighter bounds on the partition function of a Boltzmann distribution by further application of the double integration procedure.*

Exercise 259. *Derive Linkser's bound on the Mutual Information, equation (28.5.15).*

Exercise 260. *Consider the average of a positive function $f(x)$ with respect to a distribution $p(x)$*

$$J = \log \int_x p(x) f(x) \tag{28.11.10}$$

where $f(x) \geq 0$. The simplest version of Jensen's inequality states that

$$J \geq \int_x p(x) \log f(x) \tag{28.11.11}$$

1. *By considering a distribution $r(x) \propto p(x)f(x)$, and $KL(q|r)$, for some variational distribution $q(x)$, show that*

$$J \geq -KL(q(x)|p(x)) + \langle \log f(x) \rangle_{q(x)} \tag{28.11.12}$$

The bound saturates when $q(x) \propto p(x)f(x)$. This shows that if we wish to approximate the average J, the optimal choice for the approximating distribution depends on both the distribution $p(x)$ and integrand $f(x)$.

2. *Furthermore, show that*

$$J \geq -KL(q(x)|p(x)) - KL(q(x)|f(x)) - H(q(x)) \tag{28.11.13}$$

where $H(q(x))$ is the entropy of $q(x)$. The first term encourages q to be close to p. The second encourages q to be close to f, and the third encourages q to be sharply peaked.

Exercise 261. *For a Markov Random field over D binary variables $x_i \in \{0, 1\}$, $i = 1, \ldots, D$, we define*

$$p(x) = \frac{1}{Z} e^{\mathbf{x}^T \mathbf{W} \mathbf{x}} \tag{28.11.14}$$

show that

$$p(x_i) = \frac{Z_{\backslash i}}{Z} \tag{28.11.15}$$

where

$$Z_{\backslash i} \equiv \sum_{x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_D} e^{\mathbf{x}^T \mathbf{W} \mathbf{x}} \tag{28.11.16}$$

and explain why a bound on the marginal $p(x_i)$ requires both upper and lower bounds on partition functions.

Exercise 262. *Consider a directed graph such that the capacity of an edge $x \to y$ is $c(x, y) \geq 0$. The flow on an edge $f(x, y) \geq 0$ must not exceed the capacity of the edge. The aim is to maximise the flow from a defined source node s to a defined sink node t. In addition flow must be conserved such that for any node other than the source or sink ($y \neq s, t$),*

$$\sum_x f(x, y) = \sum_x f(y, x) \tag{28.11.17}$$

A cut is defined as a partition of the nodes into two non-overlapping sets $\mathcal{S}$ and $\mathcal{T}$ such that s is in $\mathcal{S}$ and t in $\mathcal{T}$. Show that:

1. *The net flow from s to t, $val(f)$ is the same as the net flow from $\mathcal{S}$ to $\mathcal{T}$:*

$$val(f) = \sum_{x \in \mathcal{S}, y \in \mathcal{T}} f(x, y) - \sum_{y \in \mathcal{T}, x \in \mathcal{S}} f(y, x) \tag{28.11.18}$$

2. $val(f) \leq \sum_{x \in \mathcal{S}, y \in \mathcal{T}} f(x,y)$ *namely that the flow is upper bounded by the capacity of the cut.*

The max-flow-min-cut theorem further states that the maximal flow is actually equal to the capacity of the cut.

Exercise 263 (Potts to Ising translation)**.** *Consider the function $E(x)$ defined on a set of multistate variables $dom(x_i) = \{0,1,2,\ldots,N\}$,*

$$E(x) = \sum_{i \sim j} w_{ij} \mathbb{I}\left[x_i = x_j\right] + \sum_i c_i \mathbb{I}\left[x_i = x_i^0\right] \tag{28.11.19}$$

where $w_{ij} > 0$ and observed pixel states x_i^0 are known, as are c_i. Our interest is to find an approximate maximisation of $E(x)$. Using the restricted parameteristion

$$x_i = s_i \alpha + (1 - s_i) x_i^{old} \tag{28.11.20}$$

where $s_i \in \{0,1\}$ and for a given $\alpha \in \{0,1,2,\ldots,N\}$, show how to write $E(x)$ as a function of the binary variables

$$E(s) = \sum_{i \sim j} w'_{ij} \mathbb{I}\left[s_i = s_j\right] + \sum_i c'_i s_i + \text{const.} \tag{28.11.21}$$

for $w'_{ij} > 0$. This new problem is of the form of an attractive binary MRF which can be solved exactly using the graph cuts procedure.

Exercise 264. *Consider an approximating distribution in the exponential family,*

$$q(x) = \frac{1}{Z(\boldsymbol{\phi})} e^{\boldsymbol{\phi}^{\mathsf{T}} \mathbf{g}(x)} \tag{28.11.22}$$

We wish to use $q(x)$ to approximate a distribution $p(x)$ using the KL divergence

$$KL(p|q) \tag{28.11.23}$$

1. *Show that optimally*

$$\langle \mathbf{g}(x) \rangle_{p(x)} = \langle \mathbf{g}(x) \rangle_{q(x)} \tag{28.11.24}$$

2. *Show that a Gaussian can be written in the exponential form*

$$\mathcal{N}\left(x|\mu, \sigma^2\right) = \frac{1}{Z(\boldsymbol{\phi})} e^{\boldsymbol{\phi}^{\mathsf{T}} \mathbf{g}(x)} \tag{28.11.25}$$

   where $g_1(x) = x$, $g_2(x) = x^2$ and suitably chosen $\boldsymbol{\phi}$.

3. *Hence show that the optimal Gaussian fit $\mathcal{N}\left(x|\mu, \sigma^2\right)$ to any distribution, in the minimal $KL(p|q)$ sense matches the moments:*

$$\mu = \langle x \rangle_{p(x)}, \qquad \sigma^2 = \left\langle x^2 \right\rangle_{p(x)} - \langle x \rangle_{p(x)}^2 \tag{28.11.26}$$

Exercise 265. *We wish to find a Gaussian approximation $q(x) = \mathcal{N}\left(x|m, s^2\right)$ to a distribution $p(x)$. Show that*

$$KL(p|q) = -\langle \log q(x) \rangle_{p(x)} + \text{const.} \tag{28.11.27}$$

Write the KL divergence explicitly as a function of m and s^2 and confirm the general result that the optimal m and s^2 that minimise $KL(p|q)$ are given by setting the mean and variance of q to those of p.

Exercise 266. *For a pairwise binary Markov Random Field, p with partition function*

$$Z(w, b) = \sum_x e^{\sum_{i,j} w_{ij} x_i x_j + \sum_i b_i x_i} \tag{28.11.28}$$

show that the means can be computed using

$$\frac{\partial}{\partial b_i} \log Z(w, b) = \frac{1}{Z(w, b)} \sum_x x_i e^{\sum_{i \sim j} w_{ij} x_i x_j + \sum_i b_i x_i} = \langle x_i \rangle_p \tag{28.11.29}$$

and that similarly the covariance is given by

$$\langle x_i x_j \rangle_p - \langle x_i \rangle_p \langle x_j \rangle_p = \frac{\partial^2}{\partial b_i \partial b_j} \log Z(w, b) \tag{28.11.30}$$

Exercise 267. *The naive mean field theory applied to a pairwise MRF*

$$p(x) \propto e^{\sum_{i,j} w_{ij} x_i x_j + \sum_i b_i x_i} \tag{28.11.31}$$

$dom(x_i) = \{0, 1\}$, gives a factorised approximation $q(x) = \prod_i q(x_i)$, based on minimising $KL(q|p)$. Using this we can approximate

$$\langle x_i x_j \rangle_p \approx \langle x_i \rangle_q \langle x_j \rangle_q , \qquad i \neq j \tag{28.11.32}$$

Consider the relation

$$p(x_i, x_j) = p(x_i|x_j)p(x_j) \tag{28.11.33}$$

Explain how to use a modified naive mean field method to find an approximation

$$q(x_i|x_j) \tag{28.11.34}$$

Using this 'algebraic perturbation', explain how to make an improved, non-factorised approximation of $\langle x_i x_j \rangle$.

Exercise 268. *Derive the EP updates equation (28.7.8) and equation (28.7.9).*

Exercise 269. *You are given a set of datapoints labelled 1 to N and a similarity 'metric' $w_{ij} \geq 0, i, j = 1, \ldots, N$ which denotes the similarity of the points i and j. You want to assign each datapoint to a cluster index $c^n \in \{1, \ldots, K\}$. For a subset of the datapoints you have a preference for the cluster index. Explain how to use a Potts model to formulate an objective function for this 'semi-supervised' clustering problem.*

APPENDIX A

Background Mathematics

A.1 Linear Algebra

A.1.1 Vector Algebra

Let $\mathbf{x}$ denote the n-dimensional column vector with components

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

Definition 116 (scalar product)**.** The *scalar product* $\mathbf{w} \cdot \mathbf{x}$ is defined as:

$$\mathbf{w} \cdot \mathbf{x} = \sum_{i=1}^{n} w_i x_i = \mathbf{w}^\mathsf{T}\mathbf{x} \tag{A.1.1}$$

and has a natural geometric interpretation as:

$$\mathbf{w} \cdot \mathbf{x} = |\mathbf{w}|\,|\mathbf{x}|\cos(\theta) \tag{A.1.2}$$

where θ is the angle between the two vectors. Thus if the lengths of two vectors are fixed their inner product is largest when $\theta = 0$, whereupon one vector is a constant multiple of the other. If the scalar product $\mathbf{x}^\mathsf{T}\mathbf{y} = 0$, then $\mathbf{x}$ and $\mathbf{y}$ are *orthogonal* (they are a right angles to each other).

The length of a vector is denoted $|\mathbf{x}|$, the squared length is given by

$$|\mathbf{x}|^2 = \mathbf{x}^\mathsf{T}\mathbf{x} = \mathbf{x}^2 = x_1^2 + x_2^2 + \cdots + x_n^2 \tag{A.1.3}$$

A *unit vector* $\mathbf{x}$ has $\mathbf{x}^\mathsf{T}\mathbf{x} = 1$.

Definition 117 (Linear dependence)**.** A set of vectors $\mathbf{x}^1, \ldots, \mathbf{x}^n$ is linearly dependent if there exists a vector $\mathbf{x}^j$ that can be expressed as a linear combination of the other vectors. Vice-versa, if the only solution to

$$\sum_{i=1}^{n} \alpha_i \mathbf{x}^i = \mathbf{0} \tag{A.1.4}$$

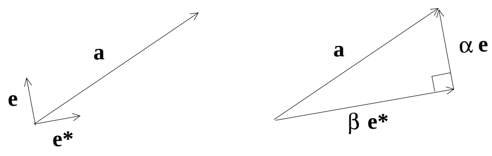


Figure A.1: Resolving a vector **a** into components along the orthogonal directions **e** and $\mathbf{e}^*$. The projection of **a** onto these two directions are lengths α and β along the directions **e** and $\mathbf{e}^*$.

is for all $\alpha_i = 0, i = 1, \ldots, n$, the vectors $\mathbf{x}^1, \ldots, \mathbf{x}^n$ are *linearly independent*.

A.1.2 The scalar product as a projection

Suppose that we wish to resolve the vector **a** into its components along the orthogonal directions specified by the unit vectors **e** and $\mathbf{e}^*$. That is $|\boldsymbol{e}| = |\boldsymbol{e}|^* = 1$ and $\mathbf{e} \cdot \mathbf{e}^* = 0$. This is depicted in fig(A.1). We are required to find the scalar values α and β such that

$$\mathbf{a} = \alpha\mathbf{e} + \beta\mathbf{e}^* \tag{A.1.5}$$

From this we obtain

$$\mathbf{a} \cdot \mathbf{e} = \alpha\mathbf{e} \cdot \mathbf{e} + \beta\mathbf{e}^* \cdot \mathbf{e}, \qquad \mathbf{a} \cdot \mathbf{e}^* = \alpha\mathbf{e} \cdot \mathbf{e}^* + \beta\mathbf{e}^* \cdot \mathbf{e}^* \tag{A.1.6}$$

From the orthogonality and unit lengths of the vectors **e** and $\mathbf{e}^*$, this becomes simply

$$\mathbf{a} \cdot \mathbf{e} = \alpha, \qquad \mathbf{a} \cdot \mathbf{e}^* = \beta \tag{A.1.7}$$

A set of vectors is orthonormal if they are mutually orthogonal and have unit length. This means that we can write the vector **a** in terms of the orthonormal components **e** and $\mathbf{e}^*$ as

$$\mathbf{a} = (\mathbf{a} \cdot \mathbf{e})\,\mathbf{e} + (\mathbf{a} \cdot \mathbf{e}^*)\,\mathbf{e}^* \tag{A.1.8}$$

One can see therefore that the scalar product between **a** and **e** projects the vector **a** onto the (unit) direction **e**. The projection of a vector **a** onto a direction specified by **f** is therefore

$$\frac{\mathbf{a} \cdot \mathbf{f}}{|\boldsymbol{f}|^2}\mathbf{f} \tag{A.1.9}$$

A.1.3 Lines in space

A line in 2 (or more) dimensions can be specified as follows. The vector of any point along the line is given, for some s, by the equation

$$\mathbf{p} = \mathbf{a} + s\mathbf{u}, \qquad s \in \mathcal{R}. \tag{A.1.10}$$

where **u** is parallel to the line, and the line passes through the point **a**, see fig(A.2). This is called the parametric representation of the line. An alternative specification can be given by realising that all vectors along the line are orthogonal to the normal of the line, **n** (**u** and **n** are orthonormal). That is

$$(\mathbf{p} - \mathbf{a}) \cdot \mathbf{n} = 0 \Leftrightarrow \mathbf{p} \cdot \mathbf{n} = \mathbf{a} \cdot \mathbf{n} \tag{A.1.11}$$

If the vector **n** is of unit length, the right hand side of the above represents the shortest distance from the origin to the line, drawn by the dashed line in fig(A.2) (since this is the projection of **a** onto the normal direction).

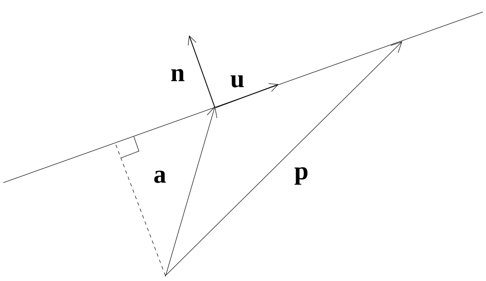


Figure A.2: A line can be specified by some position vector on the line, **a**, and a unit vector along the direction of the line, **u**. In 2 dimensions, there is a unique direction, **n**, perpendicular to the line. In three dimensions, the vectors perpendicular to the line lie in a plane, whose normal vector is in the direction of the line, **u**.

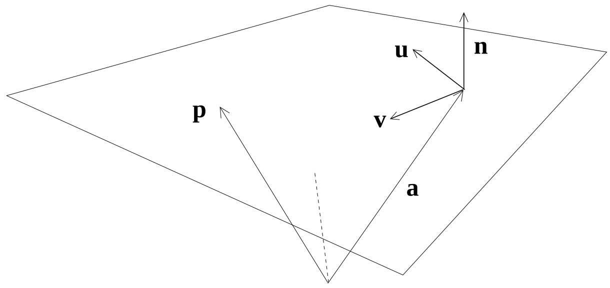


Figure A.3: A plane can be specified by a point in the plane, **a** and two, non-parallel directions in the plane, **u** and **v**. The normal to the plane is unique, and in the same direction as the directed line from the origin to the nearest point on the plane.

A.1.4 Planes and hyperplanes

A line is a one dimensional hyperplane. To define a two-dimensional plane (in arbitrary dimensional space) one may specify two vectors **u** and **v** that lie in the plane (they need not be mutually orthogonal), and a position vector **a** in the plane, see fig(A.3). Any vector **p** in the plane can then be written as

$$\mathbf{p} = \mathbf{a} + s\mathbf{u} + t\mathbf{v}, \qquad (s,t) \in \mathcal{R}. \tag{A.1.12}$$

An alternative definition is given by considering that any vector within the plane must be orthogonal to the normal of the plane **n**.

$$(\mathbf{p} - \mathbf{a}) \cdot \mathbf{n} = 0 \Leftrightarrow \mathbf{p} \cdot \mathbf{n} = \mathbf{a} \cdot \mathbf{n} \tag{A.1.13}$$

The right hand side of the above represents the shortest distance from the origin to the plane, drawn by the dashed line in fig(A.3). The advantage of this representation is that it has the same form as a line. Indeed, this representation of (hyper)planes is independent of the dimension of the space. In addition, only two vectors need to be defined – a point in the plane, **a**, and the normal to the plane **n**.

A.1.5 Matrices

An $m \times n$ matrix **A** is a collection of scalar $m \times n$ values arranged in a rectangle of m rows and n columns. A vector can be considered a $n \times 1$ matrix. If the element of the i-th row and j-th column is A_{ij}, then $\mathbf{A}^{\mathsf{T}}$ denotes the matrix that has A_{ji} there instead - the *transpose* of **A**. For example **A** and its transpose are :

$$\mathbf{A} = \begin{pmatrix} 2 & 3 & 4 \\ 4 & 5 & 9 \\ 6 & 7 & 1 \end{pmatrix} \qquad \mathbf{A}^{\mathsf{T}} = \begin{pmatrix} 2 & 4 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 1 \end{pmatrix} \tag{A.1.14}$$

The i, j element of matrix **A** can be written A_{ij} or in cases where more clarity is required, $[\mathbf{A}]_{ij}$ (for example $\left[\mathbf{A}^{-1}\right]_{ij}$).

Definition 118 (transpose)**.** The transpose $\mathbf{B}^{\mathsf{T}}$ of the n by m matrix **B** is the m by n matrix D with components

$$\left[\mathbf{B}^{\mathsf{T}}\right]_{kj} = \mathbf{B}_{jk}\,; \qquad k = 1, \ldots, m \quad j = 1, \ldots, n\,. \tag{A.1.15}$$

$\mathbf{B}$, $\left(\mathbf{B}^{\mathsf{T}}\right)^{\mathsf{T}} = \mathbf{B}$ and $(\mathbf{AB})^{\mathsf{T}} = \mathbf{B}^{\mathsf{T}}\mathbf{A}^{\mathsf{T}}$. If the shapes of the matrices **A**,**B** and **C** are such that it makes sense to calculate the product **ABC**, then

$$(\mathbf{ABC})^{\mathsf{T}} = \mathbf{C}^{\mathsf{T}}\mathbf{B}^{\mathsf{T}}\mathbf{A}^{\mathsf{T}} \tag{A.1.16}$$

A square matrix **A** is symmetric if $\mathbf{A}^{\mathsf{T}} = \mathbf{A}$. A square matrix is called *Hermitian* if

$$\mathbf{A} = \mathbf{A}^{\mathsf{T}*} \tag{A.1.17}$$

where * denotes the complex conjugate operator. For Hermitian matrices, the eigenvectors form an orthogonal set, with real eigenvalues.

Definition 119 (Matrix addition)**.** For two matrix $\mathbf{A}$ and $\mathbf{B}$ of the same size,

$$[\mathbf{A}+\mathbf{B}]_{ij} = [\mathbf{A}]_{ij} + [\mathbf{B}]_{ij} \tag{A.1.18}$$

Definition 120 (Matrix multiplication)**.** For an l by n matrix $\mathbf{A}$ and an n by m matrix B, the product $\mathbf{AB}$ is the l by m matrix with elements

$$[\mathbf{AB}]_{ij} = \sum_{j=1}^{n} [\mathbf{A}]_{ij} [\mathbf{B}]_{jk} \; ; \qquad i = 1, \ldots, l \quad k = 1, \ldots, m \, . \tag{A.1.19}$$

For example

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} a_{11}x_1 + a_{12}x_2 \\ a_{21}x_2 + a_{22}x_2 \end{pmatrix} \tag{A.1.20}$$

Note that even if $\mathbf{BA}$ is defined as well, that is if $l = n$, generally $\mathbf{BA}$ is not equal to $\mathbf{AB}$ (when they do we say they *commute*). The matrix $\mathbf{I}$ is the *identity matrix*, necessarily square, with 1's on the diagonal and 0's everywhere else. For clarity we may also write $\mathbf{I}_m$ for an square $m \times m$ identity matrix. Then for an $m \times n$ matrix $\mathbf{A}$

$$\mathbf{I}_m\mathbf{A} = \mathbf{A}\mathbf{I}_n = \mathbf{A} \tag{A.1.21}$$

The identity matrix has elements $[\mathbf{I}]_{ij} = \delta_{ij}$ given by the *Kronecker delta*:

$$\delta_{ij} \equiv \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \tag{A.1.22}$$

Definition 121 (Trace)**.**

$$\text{trace}\,(\mathbf{A}) = \sum_i A_{ii} = \sum_i \lambda_i \tag{A.1.23}$$

where λ_i are the eigenvalues of $\mathbf{A}$.

A.1.6 Linear transformations

Rotations

If we assume that rotation of a two-dimensional vector $\mathbf{x} = (x, y)^\mathsf{T}$ can be accomplished by matrix multiplication $\mathbf{Rx}$ then, since matrix multiplication is distributive, we only need to work out how the axes unit vectors $\boldsymbol{i} = (1, 0)^\mathsf{T}$ and $\boldsymbol{j} = (0, 1)^\mathsf{T}$ transform since

$$\mathbf{Rx} = x\mathbf{R}\boldsymbol{i} + y\mathbf{R}\boldsymbol{j} \tag{A.1.24}$$

The unit vectors $\boldsymbol{i}$ and $\boldsymbol{j}$ under rotation by θ degrees transform to vectors

$$\mathbf{R}\boldsymbol{i} = \begin{pmatrix} r_{11} \\ r_{21} \end{pmatrix} = \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix} \qquad \mathbf{R}\boldsymbol{j} = \begin{pmatrix} r_{12} \\ r_{22} \end{pmatrix} = \begin{pmatrix} -\sin\theta \\ \cos\theta \end{pmatrix} \tag{A.1.25}$$

From this, one can simply read off the values for the elements

$$\mathbf{R} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \tag{A.1.26}$$

A.1.7 Determinants

Definition 122 (Determinant)**.** For a square matrix $\mathbf{A}$, the determinant is the volume of the transformation of the matrix $\mathbf{A}$ (up to a sign change). That is, we take a hypercube of unit volume and map each vertex under the transformation, and the volume of the resulting object is defined as the determinant. Writing $[\mathbf{A}]_{ij} = a_{ij}$,

$$\det\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} = a_{11}a_{22} - a_{21}a_{12} \tag{A.1.27}$$

$$\det\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = a_{11}\left(a_{22}a_{33} - a_{23}a_{32}\right) - a_{12}\left(a_{21}a_{33} - a_{31}a_{23}\right) + a_{13}\left(a_{21}a_{32} - a_{31}a_{22}\right) \tag{A.1.28}$$

The determinant in the (3×3) case has the form

$$a_{11}\det\begin{pmatrix} a_{12} & a_{23} \\ a_{32} & a_{33} \end{pmatrix} - a_{12}\det\begin{pmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{pmatrix} + a_{13}\det\begin{pmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{pmatrix} \tag{A.1.29}$$

The determinant of the (3×3) matrix $\mathbf{A}$ is given by the sum of terms $(-1)^{i+1}a_{1i}\det(\mathbf{A}_i)$ where $\mathbf{A}_i$ is the (2×2) matrix formed from $\mathbf{A}$ by removing the i^{th} row and column. This form of the determinant generalises to any dimension. That is, we can define the determinant recursively as an expansion along the top row of determinants of reduced matrices. The absolute value of the determinant is the volume of the transformation.

$$\det\left(\mathbf{A}^{\mathsf{T}}\right) = \det(\mathbf{A}) \tag{A.1.30}$$

For square matrices $\mathbf{A}$ and $\mathbf{B}$ of equal dimensions,

$$\det(\mathbf{AB}) = \det(\mathbf{A})\det(\mathbf{B}), \qquad \det(\mathbf{I}) = 1 \Rightarrow \det\left(\mathbf{A}^{-1}\right) = 1/\det(\mathbf{A}) \tag{A.1.31}$$

For any matrix $\mathbf{A}$ which collapses dimensions, then the volume of the transformation is zero, and so is the determinant. If the determinant is zero, the matrix cannot be invertible since given any vector $\mathbf{x}$, given a 'projection' $\mathbf{y} = \mathbf{Ax}$, we cannot uniquely compute which vector $\mathbf{x}$ was projected to $\mathbf{y}$– there will in general be an infinite number of solutions.

Definition 123 (Orthogonal matrix)**.** A square matrix $\mathbf{A}$ is orthogonal if $\mathbf{AA}^{\mathsf{T}} = \mathbf{I} = \mathbf{A}^{\mathsf{T}}\mathbf{A}$. From the properties of the determinant, we see therefore that an orthogonal matrix has determinant ± 1 and hence corresponds to a volume preserving transformation – *i.e.* a rotation.

Definition 124 (Matrix rank)**.** For an $m \times n$ matrix $\mathbf{X}$ with n columns, each written as an m-vector:

$$\mathbf{X} = \left[\mathbf{x}^1, \ldots, \mathbf{x}^n\right] \tag{A.1.32}$$

the rank of $\mathbf{X}$ is the maximum number of linearly independent columns (or equivalently rows). A $n \times n$ square matrix is full rank if the rank is n and the matrix is non-singular. Otherwise the matrix is reduced rank and is singular.

A.1.8 Matrix inversion

Definition 125 (Matrix inversion)**.** For a square matrix $\mathbf{A}$, its inverse satisfies

$$\mathbf{A}^{-1}\mathbf{A} = \mathbf{I} = \mathbf{A}\mathbf{A}^{-1} \tag{A.1.33}$$

It is not always possible to find a matrix $\mathbf{A}^{-1}$ such that $\mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$. In that case, we call the matrix $\mathbf{A}$ *singular*. Geometrically, singular matrices correspond to 'projections': if we were to take the transform of each of the vertices $\mathbf{v}$ of a binary hypercube $\mathbf{A}\mathbf{v}$, the volume of the transformed hypercube would be zero. If you are given a vector $\mathbf{y}$ and a singular transformation, $\mathbf{A}$, one cannot uniquely identify a vector $\mathbf{x}$ for which $\mathbf{y} = \mathbf{A}\mathbf{x}$ - typically there will be a whole space of possibilities.

Provided the inverses matrices exist

$$(\mathbf{A}\mathbf{B})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1} \tag{A.1.34}$$

For a non-square matrix $\mathbf{A}$ such that $\mathbf{A}\mathbf{A}^{\mathsf{T}}$ is invertible, then the pseudo inverse, defined as

$$\mathbf{A}^{\dagger} = \mathbf{A}^{\mathsf{T}}\left(\mathbf{A}\mathbf{A}^{\mathsf{T}}\right)^{-1} \tag{A.1.35}$$

satisfies $\mathbf{A}\mathbf{A}^{\dagger} = \mathbf{I}$.

A.1.9 Computing the matrix inverse

For a 2×2 matrix, it is straightforward to work out for a general matrix, the explicit form of the inverse. If the matrix whose inverse we wish to find is $\mathbf{A} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$, then the condition for the inverse is

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{A.1.36}$$

Multiplying out the left hand side, we obtain the four conditions

$$\begin{pmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{A.1.37}$$

It is readily verified that the solution to this set of four linear equations is given by

$$\begin{pmatrix} e & f \\ g & h \end{pmatrix} = \frac{1}{ad - bc}\begin{pmatrix} d & -c \\ -b & a \end{pmatrix} = \mathbf{A}^{-1} \tag{A.1.38}$$

The quantity $ad - bc$ is the determinant of $\mathbf{A}$. There are many ways to compute the inverse of a general matrix, and we refer the reader to more specialised texts.

Note that, if one wants to solve only a linear system, although the solution can be obtained through matrix inversion, this should not be use. Often, one needs to solve huge dimensional linear systems of equations, and speed becomes an issue. These equations can be solved much more accurately and quickly using elimination techniques such as Gaussian Elimination.

A.1.10 Eigenvalues and eigenvectors

The eigenvectors of a matrix correspond to the natural coordinate system, in which the geometric transformation represented by $\mathbf{A}$ can be most easily understood.

Definition 126 (Eigenvalues and Eigenvectors)**.** For a square matrix $\mathbf{A}$, $\mathbf{e}$ is an eigenvector of $\mathbf{A}$ with eigenvalue λ if

$$\mathbf{A}\mathbf{e} = \lambda\mathbf{e} \tag{A.1.39}$$

$$\det(\mathbf{A}) = \prod_{i=1}^{n} \lambda_i \tag{A.1.40}$$

Hence a matrix is singular if it has a zero eigenvalue. The trace of a matrix can be expressed as

$$\text{trace}(\mathbf{A}) = \sum_i \lambda_i \tag{A.1.41}$$

For an $(n \times n)$ dimensional matrix, there are (including repetitions) n eigenvalues, each with a corresponding eigenvector. We can reform equation (A.1.39) as

$$(\mathbf{A} - \lambda\mathbf{I})\,\mathbf{e} = \mathbf{0} \tag{A.1.42}$$

This is a linear equation, for which the eigenvector $\mathbf{e}$ and eigenvalue λ is a solution. We can write equation (A.1.42) as $\mathbf{B}\mathbf{e} = \mathbf{0}$, where $\mathbf{B} \equiv \mathbf{A} - \lambda\mathbf{I}$. If $\mathbf{B}$ has an inverse, then a solution is $\mathbf{e} = \mathbf{B}^{-1}\mathbf{0} = \mathbf{0}$, which trivially satisfies the eigen-equation. For any non-trivial solution to the problem $\mathbf{B}\mathbf{e} = \mathbf{0}$, we therefore need $\mathbf{B}$ to be non-invertible. This is equivalent to the condition that $\mathbf{B}$ has zero determinant. Hence λ is an eigenvalue of A if

$$\det(\mathbf{A} - \lambda\mathbf{I}) = 0 \tag{A.1.43}$$

This is known as the characteristic equation. This determinant equation will be a polynomial of degree n and the resulting equation is known as the characteristic polynomial. Once we have found an eigenvalue, the corresponding eigenvector can be found by substituting this value for λ in equation (A.1.39) and solving the linear equations for $\mathbf{e}$. It may be that the for an eigenvalue λ the eigenvector is not unique and there is a space of corresponding vectors. Geometrically, the eigenvectors are special directions such that the effect of the transformation $\mathbf{A}$ along a direction $\mathbf{e}$ is simply to scale the vector $\mathbf{e}$. For a rotation matric $\mathbf{R}$ in general there will be no direction preserved under the rotation so that the eigenvalues and eigenvectors are complex valued (which is why the Fourier representation, which corresponds to representation in a rotated basis, is necessarily complex).

Remark 10 (Orthogonality of eigenvectors of symmetric matrices)**.** For a real symmetric matric $\mathbf{A} = \mathbf{A}^{\mathsf{T}}$, and two of its eigenvectors $\mathbf{e}^i$ and $\mathbf{e}^j$ of $\mathbf{A}$ are orthogonal $(\mathbf{e}^i)^{\mathsf{T}}\mathbf{e}^j = 0$ if the eigenvalues λ_i and λ_j are different.

The above can be shown by considering:

$$\mathbf{A}\mathbf{e}^i = \lambda_i\mathbf{e}^i \Rightarrow (\mathbf{e}^j)^{\mathsf{T}}\mathbf{A}\mathbf{e}^i = \lambda_i(\mathbf{e}^j)^{\mathsf{T}}\mathbf{e}^i \tag{A.1.44}$$

Since $\mathbf{A}$ is symmetric, the left hand side is equivalent to

$$((\mathbf{e}^j)^{\mathsf{T}}\mathbf{A})\mathbf{e}^i = (\mathbf{A}\mathbf{e}^j)^{\mathsf{T}}\mathbf{e}^i = \lambda_j(\mathbf{e}^j)^{\mathsf{T}}\mathbf{e}^i \Rightarrow \lambda_i(\mathbf{e}^j)^{\mathsf{T}}\mathbf{e}^i = \lambda_j(\mathbf{e}^j)^{\mathsf{T}}\mathbf{e}^i \tag{A.1.45}$$

If $\lambda_i \neq \lambda_j$, this condition can be satisfied only if $(\mathbf{e}^j)^{\mathsf{T}}\mathbf{e}^i = 0$, namely that the eigenvectors are orthogonal.

A.1.11 Matrix decompositions

The observation that the eigenvectors of a symmetric matrix are orthogonal leads directly to the spectral decomposition formula below.

Definition 127 (Spectral decomposition)**.** A symmetric matrix $\mathbf{A}$ has an eigen-decomposition

$$\mathbf{A} = \sum_{i=1}^{n} \lambda_i \mathbf{e}_i \mathbf{e}_i^{\mathsf{T}} \tag{A.1.46}$$

where λ_i is the eigenvalue of eigenvector $\mathbf{e}_i$ and the eigenvectors form an orthogonal set,

$$\left(\mathbf{e}^i\right)^{\mathsf{T}} \mathbf{e}^j = \delta_{ij} \left(\mathbf{e}^i\right)^{\mathsf{T}} \mathbf{e}^i \tag{A.1.47}$$

In matrix notation

$$\mathbf{A} = \mathbf{E}\boldsymbol{\Lambda}\mathbf{E}^{\mathsf{T}} \tag{A.1.48}$$

where $\mathbf{E}$ is the matrix of eigenvectors and $\boldsymbol{\Lambda}$ the corresponding diagonal eigenvalue matrix. More generally, for a square non-symmetric non-singular $\mathbf{A}$ we can write

$$\mathbf{A} = \mathbf{E}\boldsymbol{\Lambda}\mathbf{E}^{-1} \tag{A.1.49}$$

Definition 128 (Singular Value Decomposition)**.** The SVD decomposition of a $n \times p$ matrix $\mathbf{X}$ is

$$\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^{\mathsf{T}} \tag{A.1.50}$$

where $\dim \mathbf{U} = n \times n$ with $\mathbf{U}^{\mathsf{T}}\mathbf{U} = \mathbf{I}_n$. Also $\dim \mathbf{V} = p \times p$ with $\mathbf{V}^{\mathsf{T}}\mathbf{V} = \mathbf{I}_p$. The matrix $\mathbf{S}$ has $\dim S = n \times p$ with zeros everywhere except on the diagonal entries. The 'singular values' are the diagonal entries $[\mathbf{S}]_{ii}$ and are positive. The singular values are ordered so that the upper left diagonal element of $\mathbf{S}$ contains the largest singular value.

Quadratic forms

Definition 129 (Quadratic form)**.**

$$\mathbf{x}^{\mathsf{T}}\mathbf{A}\mathbf{x} + \mathbf{x}^{\mathsf{T}}\mathbf{b} \tag{A.1.51}$$

Definition 130 (Positive definite matrix)**.** A symmetric matrix $\mathbf{A}$, with the property that $\mathbf{x}^{\mathsf{T}}\mathbf{A}\mathbf{x} \geq 0$ for any vector $\mathbf{x}$ is called *nonnegative definite*. A symmetric matrix $\mathbf{A}$, with the property that $\mathbf{x}^{\mathsf{T}}\mathbf{A}\mathbf{x} > 0$ for any vector $\mathbf{x} \neq 0$ is called *positive definite*. A positive definite matrix has full rank and is thus invertible. Using the eigen-decomposition of $\mathbf{A}$,

$$\mathbf{x}^{\mathsf{T}}\mathbf{A}\mathbf{x} = \sum_i \lambda_i \mathbf{y}^{\mathsf{T}}\mathbf{e}^i(\mathbf{e}^i)^{\mathsf{T}}\mathbf{x} = \sum_i \lambda_i \left(\mathbf{x}^{\mathsf{T}}\mathbf{e}^i\right)^2 \tag{A.1.52}$$

which is greater than zero if and only if all the eigenvalues are positive. Hence $\mathbf{A}$ is positive definite if and only if all its eigenvalues are positive.

A.2 Matrix Identities

Definition 131 (Trace-Log formula)**.** For a positive definite matrix $\mathbf{A}$,

$$\text{trace}\left(\log \mathbf{A}\right) \equiv \log \det\left(\mathbf{A}\right) \tag{A.2.1}$$

Note that the above logarithm of a matrix is not the element-wise logarithm. In MATLAB the required function is `logm`. In general for an analytic function $f(x)$, $f(\mathbf{M})$ is defined via the power-series expansion of the function. On the right, since $\det(\mathbf{A})$ is a scalar, the logarithm is the standard logarithm of a scalar.

Definition 132 (Matrix Inversion Lemma (Woodbury formula))**.** Provided the appropriate inverses exist:

$$\left(\mathbf{A} + \mathbf{U}\mathbf{V}^{\mathsf{T}}\right)^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}\left(\mathbf{I} + \mathbf{V}^{\mathsf{T}}\mathbf{A}^{-1}\mathbf{U}\right)^{-1}\mathbf{V}^{\mathsf{T}}\mathbf{A}^{-1} \tag{A.2.2}$$

Eigenfunctions

$$\int_x K(x', x)\phi_a(x) = \lambda_a \phi_a(x') \tag{A.2.3}$$

By an analogous argument that proves the theorem of linear algebra above, the eigenfunctions are orthogonal of a real symmetric kernel, $K(x', x) = K(x, x')$ are orthogonal:

$$\int_x \phi_a(x)\phi_b^*(x) = \delta_{ab} \tag{A.2.4}$$

where $\phi^*(x)$ is the complex conjugate of $\phi(x)$[1]. From the previous results, we know that a symmetric real matrix $\mathbf{K}$ must have a decomposition in terms eigenvectors with positive, real eigenvalues. Since this is to be true for any dimension of matrix, it suggests that we need the (real symmetric) kernel function itself to have a decomposition (provided the eigenvalues are countable)

$$K(x^i, x^j) = \sum_\mu \lambda_\mu \phi_\mu(x^i)\phi_\mu^*(x^j) \tag{A.2.5}$$

since then

$$\sum_{i,j} y_i K(x^i, x^j) y_j = \sum_{i,j,\mu} \lambda_\mu y_i \phi_\mu(x^i)\phi_\mu^*(x^j) y_j = \sum_\mu \lambda_\mu \underbrace{\left(\sum_i y_i \phi_\mu(x^i)\right)}_{z_i} \underbrace{\left(\sum_i y_i \phi_\mu^*(x^i)\right)}_{z_i^*} \tag{A.2.6}$$

which is greater than zero if the eigenvalues are all positive (since for complex z, $zz^* \geq 0$). If the eigenvalues are uncountable (which happens when the domain of the kernel is unbounded), the appropriate decomposition is

$$K(x^i, x^j) = \int \lambda(s)\phi(x^i, s)\phi^*(x^j, s) ds \tag{A.2.7}$$

A.3 Multivariate Calculus

[1]This definition of the inner product is useful, and particularly natural in the context of translation invariant kernels. We are free to define the inner product, but this conjugate form is often the most useful.

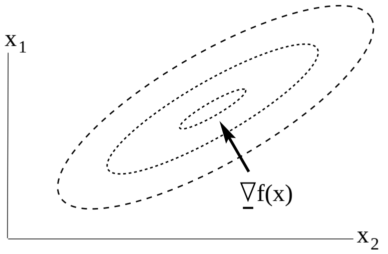


Figure A.4: Interpreting the gradient. The ellipses are contours of constant function value, $f = \mathsf{const}$. At any point $\mathbf{x}$, the gradient vector $\nabla f(\mathbf{x})$ points along the direction of maximal increase of the function.

Definition 133 (Partial derivative)**.** Consider a function of n variables, $f(x_1, x_2, \ldots, x_n)$ or $f(\mathbf{x})$. The partial derivative of f wrt x_1 at $\mathbf{x}^*$ is defined as the following limit (when it exists)

$$\left.\frac{\partial f}{\partial x_1}\right|_{\mathbf{x}=\mathbf{x}^*} = \lim_{h \to 0} \frac{f(x_1^* + h, x_2^*, \ldots, x_n^*) - f(\mathbf{x}^*)}{h} \tag{A.3.1}$$

The *gradient vector* of f will be denoted by ∇f or $\mathbf{g}$

$$\nabla f(\mathbf{x}) \equiv \mathbf{g}(\mathbf{x}) \equiv \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix} \tag{A.3.2}$$

A.3.1 Interpreting the gradient vector

Consider a function $f(\mathbf{x})$ that depends on a vector $\mathbf{x}$. We are interested in how the function changes when the vector $\mathbf{x}$ changes by a small amount : $\mathbf{x} \to \mathbf{x} + \boldsymbol{\delta}$, where $\boldsymbol{\delta}$ is a vector whose length is very small. According to a Taylor expansion, the function ϕ will change to

$$f(\mathbf{x} + \boldsymbol{\delta}) = f(\mathbf{x}) + \sum_i \delta_i \frac{\partial f}{\partial x_i} + O\left(\boldsymbol{\delta}^2\right) \tag{A.3.3}$$

We can interpret the summation above as the scalar product between the vector ∇f with components $[\nabla f]_i = \frac{\partial f}{\partial x_i}$ and $\boldsymbol{\delta}$.

$$f(\mathbf{x} + \boldsymbol{\delta}) = f(\mathbf{x}) + (\nabla f)^{\mathsf{T}} \boldsymbol{\delta} + O\left(\boldsymbol{\delta}^2\right) \tag{A.3.4}$$

The gradient points along the direction in which the function increases most rapidly. Why? Consider a direction $\hat{\mathbf{p}}$ (a unit length vector). Then a displacement, δ units along this direction changes the function value to

$$f(\mathbf{x} + \delta\hat{\mathbf{p}}) \approx f(\mathbf{x}) + \delta \nabla f(\mathbf{x}) \cdot \hat{\mathbf{p}} \tag{A.3.5}$$

The direction $\hat{\mathbf{p}}$ for which the function has the largest change is that which maximises the overlap

$$\nabla f(\mathbf{x}) \cdot \hat{\mathbf{p}} = |\nabla f(\mathbf{x})||\hat{\mathbf{p}}| \cos\theta = |\nabla f(\mathbf{x})| \cos\theta \tag{A.3.6}$$

where θ is the angle between $\hat{\mathbf{p}}$ and $\nabla f(\mathbf{x})$. The overlap is maximised when $\theta = 0$, giving $\hat{\mathbf{p}} = \nabla f(\mathbf{x})/|\nabla f(\mathbf{x})|$. Hence, the direction along which the function changes the most rapidly is along $\nabla f(\mathbf{x})$.

A.3.2 Higher derivatives

The 'first derivative' of a function of n variables is an n-vector; the 'second-derivative' of an n-variable function is defined by the n^2 partial derivatives of the n first partial derivatives *w.r.t.* the n variables

$$\frac{\partial}{\partial x_i}\left(\frac{\partial f}{\partial x_j}\right) \qquad i = 1, \ldots, n; \;\; j = 1, \ldots, n \tag{A.3.7}$$

which is usually written

$$\frac{\partial^2 f}{\partial x_i \partial x_j}, \quad i \neq j \qquad \frac{\partial^2 f}{\partial {x_i}^2}, \quad i = j \tag{A.3.8}$$

If the partial derivatives $\partial f/\partial x_i$, $\partial f/\partial x_j$ and $\partial^2 f/\partial x_i \partial x_j$ are continuous, then $\partial^2 f/\partial x_i \partial x_j$ exists and

$$\partial^2 f/\partial x_i \partial x_j = \partial^2 f/\partial x_j \partial x_i \,. \tag{A.3.9}$$

These n^2 second partial derivatives are represented by a square, symmetric matrix called the *Hessian* matrix of $f(\mathbf{x})$.

$$H_f(\mathbf{x}) = \begin{pmatrix} \frac{\partial^2 f}{\partial {x_1}^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & & \vdots \\ \frac{\partial^2 f}{\partial x_1 \partial x_n} & \cdots & \frac{\partial^2 f}{\partial {x_n}^2} \end{pmatrix} \tag{A.3.10}$$

A.3.3 Chain rule

Consider $f(x_1, \ldots, x_n)$. Now let each x_j be parameterized by $u_1, \ldots, u_m$, *i.e.* $x_j = x_j(u_1, \ldots, u_m)$. What is $\partial f/\partial u_\alpha$?

$$\delta f = f(x_1 + \delta x_1, \ldots, x_n + \delta x_n) - f(x_1, \ldots, x_n) = \sum_{j=1}^{n} \frac{\partial f}{\partial x_j} \delta x_j + \text{higher order terms}$$

$$\delta x_j = \sum_{\alpha=1}^{m} \frac{\partial x_j}{\partial u_\alpha} \delta u_\alpha + \text{higher order terms}$$

So

$$\delta f = \sum_{j=1}^{n} \sum_{\alpha=1}^{m} \frac{\partial f}{\partial x_j} \frac{\partial x_j}{\partial u_\alpha} \delta u_\alpha + \text{higher order terms} \tag{A.3.11}$$

Therefore

Definition 134 (Chain rule)**.**

$$\frac{\partial f}{\partial u_\alpha} = \sum_{j=1}^{n} \frac{\partial f}{\partial x_j} \frac{\partial x_j}{\partial u_\alpha} \tag{A.3.12}$$

or in vector notation

$$\frac{\partial}{\partial u_\alpha} f(\mathbf{x}(\mathbf{u})) = \nabla f^{\mathsf{T}}(\mathbf{x}(\mathbf{u})) \frac{\partial \mathbf{x}(\mathbf{u})}{\partial u_\alpha} \tag{A.3.13}$$

Definition 135 (Directional derivative)**.** Assume f is differentiable. We define the scalar directional derivative $(D_{\mathbf{v}} f)(\mathbf{x}^*)$ of f in a direction $\mathbf{v}$ at a point $\mathbf{x}^*$. Let $\mathbf{x} = \mathbf{x}^* + h\mathbf{v}$, Then

$$(D_{\mathbf{v}} f)(\mathbf{x}^*) = \left. \frac{d}{dh} f(\mathbf{x}^* + h\mathbf{v}) \right|_{h=0} = \sum_j v_j \left. \frac{\partial f}{\partial x_j} \right|_{\mathbf{x}=\mathbf{x}^*} = \nabla f^{\mathsf{T}} \mathbf{v} \tag{A.3.14}$$

A.3.4 Matrix calculus

Definition 136 (Derivative of a matrix trace)**.** For matrices $\mathbf{A}$, and $\mathbf{B}$

$$\frac{\partial}{\partial \mathbf{A}} \text{trace}\left(\mathbf{A}\mathbf{B}\right) \equiv \mathbf{B}^{\mathsf{T}} \tag{A.3.15}$$

Definition 137 (Derivative of $\log \det (\mathbf{A})$)**.**

$$\partial \log \det (\mathbf{A}) = \partial \text{trace} (\log \mathbf{A}) = \text{trace}\left(\mathbf{A}^{-1} \partial \mathbf{A}\right) \tag{A.3.16}$$

So that

$$\frac{\partial}{\partial \mathbf{A}} \log \det (\mathbf{A}) = \mathbf{A}^{-\mathsf{T}} \tag{A.3.17}$$

Definition 138 (Derivative of a matrix inverse)**.** For an invertible matrix $\mathbf{A}$,

$$\partial \mathbf{A}^{-1} \equiv -\mathbf{A}^{-\mathsf{T}} \partial \mathbf{A} \mathbf{A}^{-1} \tag{A.3.18}$$

A.4 Inequalities

A.4.1 Convexity

Definition 139 (Convex function)**.** A function $f(x)$ is defined as convex if for any x, y and $0 \leq \lambda \leq 1$

$$f(\lambda x + (1 - \lambda) y) \leq \lambda f(x) + (1 - \lambda) f(y) \tag{A.4.1}$$

If $-f(x)$ is convex, $f(x)$ is called concave.

An intuitive picture of a convex function is to consider first the quantity $\lambda x + (1 - \lambda)y$. As we vary λ from 0 to 1, this traces points between x ($\lambda = 0$) and y ($\lambda = 1$). Hence for $\lambda = 0$ we start at the point $x, f(x)$ and as λ increase trace a straight line towards the point $y, f(y)$ at $\lambda = 1$. Convexity states that the function f always lies below this straight line. Geometrically this means that the function $f(x)$ is always always increasing (never non-decreasing). Hence if $d^2 f(x)/dx^2 > 0$ the function is convex.

As an example, the function $\log x$ is concave since its second derivative is negative:

$$\frac{d}{dx} \log x = \frac{1}{x}, \qquad \frac{d^2}{dx^2} \log x = -\frac{1}{x^2} \tag{A.4.2}$$

A.4.2 Jensen's inequality

For a convex function $f(x)$, it follows directly from the definition of convexity that

$$f(\langle x \rangle_{p(x)}) \leq \langle f(x) \rangle_{p(x)} \tag{A.4.3}$$

for any distribution $p(x)$.

A.5 Optimisation

A.5.1 Critical points

When all first-order partial derivatives at a point are zero (*i.e.* $\nabla f = \mathbf{0}$) then the point is said to be a stationary or critical point. Can be a minimum, maximum or saddle point.

Necessary first-order condition for a minimum

There is a minimum of f at $\mathbf{x}^*$ if $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for all $\mathbf{x}$ sufficiently close to $\mathbf{x}^*$. Let $\mathbf{x} = \mathbf{x}^* + h\mathbf{v}$ for small h and some direction $\mathbf{v}$. Then by a Taylor expansion, for small h,

$$f(\mathbf{x}^* + h\mathbf{v}) = f(\mathbf{x}^*) + h\nabla f^{\mathsf{T}}\mathbf{v} + O(h^2) \tag{A.5.1}$$

and thus for a minimum

$$h\nabla f^{\mathsf{T}}\mathbf{v} + O(h^2) \geq 0 \tag{A.5.2}$$

Choosing $\mathbf{v}$ to be $-\nabla f$ the condition becomes

$$-h\nabla f^{\mathsf{T}}\nabla f + O(h^2) \geq 0 \tag{A.5.3}$$

and is violated for small positive h unless $|\nabla f|^2 = \nabla f^{\mathsf{T}}\nabla f = 0$. So $\mathbf{x}^*$ can only be a local minimum if $|\nabla f(\mathbf{x}^*)| = 0$, *i.e.* if $\nabla f(\mathbf{x}^*) = \mathbf{0}$.

Necessary second-order condition for a minimum

At a stationary point $\nabla f = \mathbf{0}$. Hence the Taylor expansion is given by

$$f(\mathbf{x}^* + h\mathbf{v}) = f(\mathbf{x}^*) + h^2\mathbf{v}^{\mathsf{T}}H_f\mathbf{v} + O(h^3) \tag{A.5.4}$$

Thus the minimum condition requires that $\mathbf{v}^{\mathsf{T}}H_f\mathbf{v} \geq 0$, *i.e.* the Hessian is non-negative definite.

Definition 140 (Conditions for a minimum)**.** Sufficient conditions for a minimum at $\mathbf{x}^*$ are (i) $\nabla f(\mathbf{x}^*) = 0$ and (ii) $H_f(\mathbf{x}^*)$ is positive definite.

For a quadratic function $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^{\mathsf{T}}\mathbf{A}\mathbf{x} - \mathbf{b}^{\mathsf{T}}\mathbf{x} + c$, with symmetric $\mathbf{A}$ the necessary condition $\nabla f(\mathbf{x}^*) = \mathbf{0}$ reads:

$$\mathbf{A}\mathbf{x}^* - \mathbf{b} = 0 \tag{A.5.5}$$

If $\mathbf{A}$ is invertible this equation has the unique solution $\mathbf{x}^* = \mathbf{A}^{-1}\mathbf{b}$. If $\mathbf{A}$ is positive definite, $\mathbf{x}^*$ is a minimum.

A.6 Gradient Descent

Almost all of the search techniques that we consider are iterative, *i.e.* we proceed towards the minimum $\mathbf{x}^*$ by a sequence of steps. On the kth step we take a step of length α_k in the direction $\mathbf{p}_k$,

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{p}_k \tag{A.6.1}$$

The length of the step can either be chosen using prior knowledge, or by carrying out a line search in the direction $\mathbf{p}_k$. It is the way that $\mathbf{p}_k$ is chosen that tends to distinguish the different methods of multivariate optimization that we will discuss.

We shall assume that we can analytically evaluate the gradient of f and will often use the shorthand notation

$$\mathbf{g}_k = \nabla f(\mathbf{x}_k) \; . \tag{A.6.2}$$

Typically we will want to choose $\mathbf{p}_k$ using only gradient information; for large problems it can be very expensive to compute the Hessian, and this can also require a large amount of storage.
Consider the change of variables $\mathbf{x} = \mathbf{M}\mathbf{y}$. Then

$$g(\mathbf{y}) = f(\mathbf{x}) = f(\mathbf{M}\mathbf{y}) \tag{A.6.3}$$

and

$$\frac{\partial g}{\partial y_i} = \sum_j \frac{\partial f}{\partial x_i}\frac{\partial x_i}{\partial y_j} \tag{A.6.4}$$

so that

$$\nabla_{\mathbf{y}} g = \mathbf{M}\nabla_{\mathbf{x}} f \tag{A.6.5}$$

Then the change in g is different from the change in f, even though the only difference between the two functions is the coordinate system. This unfortunate sensitivity to the parameterisation of the function is partially addressed in first order methods such as gradient descent by the natural gradient which uses a prefactor designed to compensate for some of the lost invariance. We refer the reader to [7] for a description of this method.

A.6.1 Gradient descent with fixed stepsize

Locally, if we are at point $\mathbf{x}_k$, we can decrease $f(\mathbf{x})$ by taking a step in the direction $-\mathbf{g}(\mathbf{x})$. If we make the update equation

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta \mathbf{g}_k \tag{A.6.6}$$

then we are doing gradient descent with fixed stepsize η. If η is non-infinitesimal, it is always possible that we will step over the true minimum. Making η very small guards against this, but means that the optimization process will take a very long time to reach a minimum.

To see why gradient descent works, consider the general update

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha \mathbf{p}_k \tag{A.6.7}$$

For small α we can expand f around $\mathbf{x}_k$ using Taylor's theorem:

$$f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) \approx f(\mathbf{x}_k) + \alpha_k \mathbf{g}_k^{\mathsf{T}} \mathbf{p}_k \; . \tag{A.6.8}$$

With $\mathbf{p}_k = -\mathbf{g}_k$ and for small positive α_k, we see a guaranteed reduction:

$$f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) \approx f(\mathbf{x}_k) - \alpha_k ||\mathbf{g}_k||^2 \, . \tag{A.6.9}$$

A.6.2 Gradient descent with momentum

A simple idea that can improve convergence of gradient descent is to include at each iteration a proportion of the change from the previous iteration. one uses

$$\Delta \mathbf{x}_{k+1} = -\eta \frac{\partial E}{\partial \mathbf{x}} + \alpha \Delta \mathbf{x}_k \tag{A.6.10}$$

where α is the *momentum coefficient*.

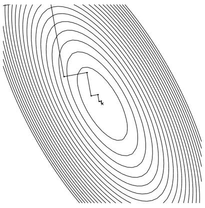


Figure A.5: Optimisation using line search along steepest descent directions. Rushing off following the steepest way downhill from a point (and continuing for a finite time in that direction) doesn't always result in the fastest way to get to the bottom!

A.6.3 Gradient descent with line searches

An extension to the idea of gradient descent is to choose the direction of steepest descent, as indicated by the gradient $\mathbf{g}$, but to calculate the value of the step to take which most reduces the value of E when moving in that direction. This involves solving the one-dimensional problem of minimizing $E(\mathbf{x}_k - \lambda \mathbf{g}_k)$ with respect to λ, and is known as a *line search*. That step is then taken and the process repeated again.

Finding the size of the step takes a little work; for example, you might find three points along the line such that the error at the intermediate point is less than at the other two, so that there is some minimum along the line lies between the first and second or between the second and third, and some kind of interval-halving approach can then be used to find it. (The minimum found in this way, just as with any sort of gradient-descent algorithm, may not be a global minimum of course.) There are several variants of this theme. Notice that if the step size is chosen to reduce E as much as it can in that direction, then no further improvement in E can be made by moving in that direction for the moment. Thus the next step will have no component in that direction; that is, the next step will be at right angles to the one just taken. This can lead to zig-zag type behaviour in the optimisation, see fig(A.5).

A.6.4 Exact Line Search Condition

At the k-th step, we chose α_k to minimize $f(\mathbf{x}_k + \alpha_k \mathbf{p}_k)$. So setting $F(\lambda) = f(\mathbf{x}_k + \lambda \mathbf{p}_k)$, at this step we solve the one-dimensional minimization problem for $F(\lambda)$. Thus our choice of $\alpha_k = \lambda^*$ will satisfy $F'(\alpha_k) = 0$. Now

$$
\begin{aligned}
F'(\alpha_k) &= \frac{d}{dh} F(\alpha_k + h)_{|h=0} = \frac{d}{dh} f(\mathbf{x}_k + \alpha_k \mathbf{p}_k + h\mathbf{p}_k)_{|h=0} \\
&= \frac{d}{dh} f(\mathbf{x}_{k+1} + h\mathbf{p}_k)_{|h=0} = (D_{\mathbf{p}_k} f)(\mathbf{x}_{k+1}) = \nabla f^{\mathsf{T}}(\mathbf{x}_{k+1})\mathbf{p}_k
\end{aligned}
\tag{A.6.11}
$$

So $F'(\alpha_k) = 0$ means the directional derivative in the search direction must vanish at the new point and this gives the Exact Line Search Condition:

$$
0 = \mathbf{g}_{k+1}^{\mathsf{T}} \mathbf{p}_k \ . \tag{A.6.12}
$$

For a quadratic function $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^{\mathsf{T}}\mathbf{A}\mathbf{x} - \mathbf{b}^{\mathsf{T}}\mathbf{x} + c$, with symmetric A, we can use the condition to analytically calculate α_k. Since $\nabla f(\mathbf{x}_{k+1}) = \mathbf{A}\mathbf{x}_k + \alpha_k \mathbf{A}\mathbf{p}_k - \mathbf{b} = \nabla f(\mathbf{x}_k) + \alpha_k \mathbf{A}\mathbf{p}_k$ we find

$$
\alpha_k = -\frac{\mathbf{p}_k^{\mathsf{T}} \mathbf{g}_k}{\mathbf{p}_k^{\mathsf{T}} \mathbf{A} \mathbf{p}_k} \ . \tag{A.6.13}
$$

A.7 Multivariate Minimization: Quadratic functions

The goal of this section is to derive efficient algorithms for minimizing multivariate quadratic functions. We shall begin by summarizing some properties of quadratic functions, and as byproduct obtain an efficient method for checking whether a symmetric matrix is positive definite.

A.7.1 Minimising quadratic functions using line search

Consider minimising the quadratic function

$$
f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^{\mathsf{T}}\mathbf{A}\mathbf{x} - \mathbf{b}^{\mathsf{T}}\mathbf{x} + c \tag{A.7.1}
$$

where $\mathbf{A}$ is positive definite and symmetric. (If $\mathbf{A}$ is not symmetric, consider instead the symmetrised matrix $(\mathbf{A} + \mathbf{A}^\mathsf{T})/2$, which gives the same function f). Although we know where the minimum of this function is, just using linear algebra, we wish to use this function as a toy model for more complex functions which however locally look approximately quadratic. One approach is to search along a particular direction $\mathbf{p}$, and find a minimum along this direction. We can then search for a deeper minima by looking in different directions. That is, we can search along a line $\mathbf{x}^0 + \lambda\mathbf{p}$ such that the function attains a minimum. That is, the directional derivative is zero along this line, This has solution,

$$\lambda = \frac{\left(\mathbf{b} - \mathbf{A}\mathbf{x}^0\right)\cdot\mathbf{p}}{\mathbf{p}^\mathsf{T}\mathbf{A}\mathbf{p}} \equiv \frac{-\nabla f(\mathbf{x}^0)\cdot\mathbf{p}}{\mathbf{p}^\mathsf{T}\mathbf{A}\mathbf{p}} \tag{A.7.2}$$

Now we've found the minimum along the line through $\mathbf{x}^0$ with direction $\mathbf{p}$. But how should we choose the line search direction $\mathbf{p}$? It would seem sensible to choose successive line search directions $\mathbf{p}$ according to $\mathbf{p}^{new} = -\nabla f(\mathbf{x}^*)$, so that each time we minimise the function along the line of steepest descent. However, this is far from the optimal choice in the case of minimising quadratic functions. A much better set of search directions are those defined by the vectors conjugate to $\mathbf{A}$.

If the matrix $\mathbf{A}$ were diagonal, then the minimisation is straightforward and can be carried out independently for each dimension. If we could find an invertible matrix $\mathbf{P}$ with the property that $\mathbf{P}^\mathsf{T}\mathbf{A}\mathbf{P}$ is diagonal then the solution is easy since for

$$f(\hat{\mathbf{x}}) = \frac{1}{2}\hat{\mathbf{x}}^\mathsf{T}\mathbf{P}^\mathsf{T}\mathbf{A}\mathbf{P}\hat{\mathbf{x}} - \mathbf{b}^\mathsf{T}\mathbf{P}\hat{\mathbf{x}} + c \tag{A.7.3}$$

with $\mathbf{x} = \mathbf{P}\hat{\mathbf{x}}$, we can compute the minimum for each dimension of $\hat{\mathbf{x}}$ separately and then retransform to find $\mathbf{x}^* = \mathbf{P}\hat{\mathbf{x}}^*$.

Definition 141 (Conjugate vectors)**.** The vectors $\mathbf{p}_i$, $i = 1, \ldots, k$ are called conjugate to the matrix A, if and only if for $i, j = 1, \ldots, k$ and $i \neq j$:

$$\mathbf{p}_i^\mathsf{T}\mathbf{A}\mathbf{p}_j = 0 \quad \text{and} \quad \mathbf{p}_i^\mathsf{T}\mathbf{A}\mathbf{p}_i > 0\,. \tag{A.7.4}$$

The two conditions guarantee that conjugate vectors are linearly independent: Assume that

$$\mathbf{0} = \sum_{j=1}^{k}\alpha_j\mathbf{p}_j = \sum_{j=1}^{i-1}\alpha_j\mathbf{p}_j + \alpha_i\mathbf{p}_i + \sum_{j=i+1}^{k}\alpha_j\mathbf{p}_j \tag{A.7.5}$$

Now multiplying from the left with $\mathbf{p}_i^\mathsf{T}\mathbf{A}$ yields $0 = \alpha_i\mathbf{p}_i^\mathsf{T}\mathbf{A}\mathbf{p}_i$. So α_i is zero since we know that $\mathbf{p}_i^\mathsf{T}\mathbf{A}\mathbf{p}_i > 0$. As we can make this argument for any $i = 1, \ldots, k$, all of the α_i must be zero.

A.7.2 Gram-Schmidt construction of conjugate vectors

Let $\mathbf{P} = (\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_k)$, where the columns are formed from $\mathbf{A}$-conjugate vectors and note that we start with an n by k matrix, $k \leq n$. The reason for this is that we are aiming at an incremental procedure, where columns are successively added to $\mathbf{P}$. Since $(\mathbf{P}^\mathsf{T}\mathbf{A}\mathbf{P})_{ij} = \mathbf{p}_i^\mathsf{T}\mathbf{A}\mathbf{p}_j$ the matrix $\mathbf{P}^\mathsf{T}\mathbf{A}\mathbf{P}$ will be diagonal if $\mathbf{p}_i^\mathsf{T}\mathbf{A}\mathbf{p}_j = 0$ for $i \neq j$. Assume we already have k conjugate vectors $\mathbf{p}_1, \ldots, \mathbf{p}_k$ and let $\mathbf{v}$ be a vector which is linearly independent of $\mathbf{p}_1, \ldots, \mathbf{p}_k$. We then set

$$\mathbf{p}_{k+1} = \mathbf{v} - \sum_{j=1}^{k}\frac{\mathbf{p}_j^\mathsf{T}\mathbf{A}\mathbf{v}}{\mathbf{p}_j^\mathsf{T}\mathbf{A}\mathbf{p}_j}\mathbf{p}_j \tag{A.7.6}$$

for which it is clear that the vectors $\mathbf{p}_1, \ldots, \mathbf{p}_{k+1}$ are conjugate if $\mathbf{A}$ is positive definite. Using the Gram-Schmidt procedure we can construct n conjugate vectors for a positive definite matrix in the following way. We start with n linearly independent vectors $\mathbf{u}_1, \ldots, \mathbf{u}_n$, we might chose $\mathbf{u}_i = \mathbf{e}_i$, the unit vector in the i^{th} direction. We then set $\mathbf{p}_1 = \mathbf{u}_1$ and use (A.7.6) to compute $\mathbf{p}_2$ from $\mathbf{p}_1$ and $\mathbf{v} = \mathbf{u}_2$. Next we set $\mathbf{v} = \mathbf{u}_3$ and compute $\mathbf{p}_3$ from $\mathbf{p}_1, \mathbf{p}_2$ and $\mathbf{v}$. Continuing in this manner we obtain n conjugate vectors. Note that at each stage of the procedure the vectors $\mathbf{u}_1, \ldots, \mathbf{u}_k$ span the same subspace as the vectors $\mathbf{p}_1, \ldots, \mathbf{p}_k$. What is going to happen if $\mathbf{A}$ is not positive definite? If we could find n conjugate vectors, $\mathbf{A}$ would be positive definite, and so at some point k the Gram-Schmidt procedure must break down. This will happen if $\mathbf{p}_k^\mathsf{T}\mathbf{A}\mathbf{p}_k \leq 0$. So by trying out the Gram-Schmidt procedure, we can in fact find out whether a matrix is positive definite.

A.7.3 The conjugate vectors algorithm

Let us assume that when minimising $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\mathsf{T}\mathbf{A}\mathbf{x} - \mathbf{b}^\mathsf{T}\mathbf{x} + c$ we first construct n vectors $\mathbf{p}_1, \ldots, \mathbf{p}_n$ conjugate to $\mathbf{A}$ which we use as our search directions. So

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k \; . \tag{A.7.7}$$

At each step we chose α_k by an exact line search, thus

$$\alpha_k = -\frac{\mathbf{p}_k^\mathsf{T}\mathbf{g}_k}{\mathbf{p}_k^\mathsf{T}\mathbf{A}\mathbf{p}_k} \; . \tag{A.7.8}$$

This conjugate vectors algorithm, has the geometrical interpretation that not only is the directional derivative zero at the new point along the direction $\mathbf{p}_k$, it is zero along all the previous search directions $\mathbf{p}_1, \ldots, \mathbf{p}_k$).

Theorem 1 (Luenberger expanding subspace theorem)**.**

Let $\{\mathbf{p}_i\}_{i=1}^n$ be a sequence of vectors in $\mathbb{R}^n$ conjugate to the (positive definite) matrix $\mathbf{A}$ and $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\mathsf{T}\mathbf{A}\mathbf{x} - \mathbf{b}^\mathsf{T}\mathbf{x} + c$. Then for any $\mathbf{x}_1$ the sequence $\{\mathbf{x}_k\}$ generated according to (A.7.7) and (A.7.8) has the property that the directional derivative of f in the direction $\mathbf{p}_i$ vanishes at the point $\mathbf{x}_{k+1}$ if $i \leq k$; *i.e.* $D_{\mathbf{p}_i} f(\mathbf{x}_{k+1}) = 0$.

Proof: For $i \leq k$, we can write $\mathbf{x}_{k+1}$ as:

$$\mathbf{x}_{k+1} = \mathbf{x}_{i+1} + \sum_{j=i+1}^{k} \alpha_j \mathbf{p}_j \; . \tag{A.7.9}$$

Since $\nabla f(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b}$ we have

$$\nabla f(\mathbf{x}_{k+1}) = \mathbf{A}\mathbf{x}_{k+1} - \mathbf{b} = \mathbf{A}\mathbf{x}_{i+1} - \mathbf{b} + \mathbf{A}\sum_{j=i+1}^{k} \alpha_j \mathbf{p}_j = \nabla f(\mathbf{x}_{i+1}) + \sum_{j=i+1}^{k} \alpha_j \mathbf{A}\mathbf{p}_j \tag{A.7.10}$$

So

$$D_{\mathbf{p}_i} f(\mathbf{x}_{k+1}) = \mathbf{p}_i^\mathsf{T}\nabla f(\mathbf{x}_{k+1}) = \mathbf{p}_i^\mathsf{T}\nabla f(\mathbf{x}_{i+1}) + \sum_{j=i+1}^{k} \alpha_j \mathbf{p}_i^\mathsf{T}\mathbf{A}\mathbf{p}_j = (D_{\mathbf{p}_i} f)(\mathbf{x}_{i+1}) + \sum_{j=i+1}^{k} \alpha_j \mathbf{p}_i^\mathsf{T}\mathbf{A}\mathbf{p}_j \tag{A.7.11}$$

Now $(D_{\mathbf{x}_{i+1}} f)(\mathbf{p}_i) = 0$ since the point $\mathbf{x}_{i+1}$ was obtained by an exact line search in the direction $\mathbf{p}_i$. But all of the terms in the sum over j also vanish since $j > i$ and $\mathbf{p}_i^\mathsf{T}\mathbf{A}\mathbf{p}_j = 0$ by conjugacy. So $(D_{\mathbf{p}_i f})(\mathbf{x}_{k+1}) = 0$.

The subspace theorem shows, that because we use conjugate vectors, optimizing in the direction $\mathbf{p}_k$, does not spoil the optimality *w.r.t.* to the previous search directions. In particular after having carried out n steps of the algorithm we have $(D_{\mathbf{x}_{n+1}} f)(\mathbf{p}_i) = \nabla f^\mathsf{T}(\mathbf{x}_{n+1})\mathbf{p}_i = 0$, for $i = 1, \ldots, n$. The n equations can be written in a more compact form as:

$$\nabla f^\mathsf{T}(\mathbf{x}_{n+1})(\mathbf{p}_1, \mathbf{p}_2, \ldots \mathbf{p}_n) = \mathbf{0} \; . \tag{A.7.12}$$

The square matrix $\mathbf{P} = (\mathbf{p}_1, \mathbf{p}_2, \dots \mathbf{p}_n)$ is invertible since the $\mathbf{p}_i$ are conjugate, so $\nabla f(\mathbf{x}_{n+1}) = \mathbf{0}$: The point $\mathbf{x}_{n+1}$ is the minimum $\mathbf{x}^*$ of the quadratic function f. So in contrast to gradient descent, for a quadratic function the conjugate vectors algorithm converges in a finite number of steps.

A.7.4 The conjugate gradients algorithm

The conjugate gradients algorithm is a special case of the conjugate vectors algorithm, in which the Gram-Schmidt procedure becomes very simple. We do not use a predetermined set of conjugate vectors but construct these 'on-the-fly'. After k-steps of the conjugate vectors algorithm we need to construct a vector $\mathbf{p}_{k+1}$ which is conjugate to $\mathbf{p}_1, \dots, \mathbf{p}_k$. This could be done by applying the Gram-Schmidt procedure to any vector $\mathbf{v}$ which is linearly independent of the vectors $\mathbf{p}_1, \dots, \mathbf{p}_k$. In the conjugate gradients algorithm one makes the special choice $\mathbf{v} = -\nabla f(\mathbf{x}_{k+1})$. By the subspace theorem the gradient at the new point $\mathbf{x}_{k+1}$ is orthogonal to $\mathbf{p}_i$, $i = 1, \dots, k$. So $\nabla f(\mathbf{x}_{k+1})$ is linearly independent of $\mathbf{p}_1, \dots, \mathbf{p}_k$ and a valid choice for $\mathbf{v}$, unless $\nabla f(\mathbf{x}_{k+1}) = \mathbf{0}$. In the latter case $\mathbf{x}_{k+1}$ is our minimum and we are done, and from now on we assume that $\nabla f(\mathbf{x}_{k+1}) \neq \mathbf{0}$. Using the notation $\mathbf{g}_k = \nabla f(\mathbf{x}_k)$, the equation for the new search direction given by the Gram-Schmidt procedure is:

$$\mathbf{p}_{k+1} = -\mathbf{g}_{k+1} + \sum_{i=1}^{k} \frac{\mathbf{p}_i^{\mathsf{T}} \mathbf{A} \mathbf{g}_{k+1}}{\mathbf{p}_i^{\mathsf{T}} \mathbf{A} \mathbf{p}_i} \mathbf{p}_i \,. \tag{A.7.13}$$

Since $\mathbf{g}_{k+1}$ is orthogonal to $\mathbf{p}_i$, $i = 1, .., k$, by the subspace theorem we have $\mathbf{p}_{k+1}^{\mathsf{T}} \mathbf{g}_{k+1} = -\mathbf{g}_{k+1}^{\mathsf{T}} \mathbf{g}_{k+1}$. So α_{k+1} can be written as

$$\alpha_{k+1} = \frac{\mathbf{g}_{k+1}^{\mathsf{T}} \mathbf{g}_{k+1}}{\mathbf{p}_{k+1}^{\mathsf{T}} \mathbf{A} \mathbf{p}_{k+1}} \,, \tag{A.7.14}$$

and in particular $\alpha_{k+1} \neq 0$. We now want to show that because we have been using the conjugate gradients algorithm at the previous steps as well, in equation (A.7.13) all terms but the last in the sum over i vanish. We shall assume that $k > 0$ since in the first step ($k = 0$) we just set $\mathbf{p}_1 = -\mathbf{g}_1$. First note that

$$\mathbf{g}_{i+1} - \mathbf{g}_i = \mathbf{A}\mathbf{x}_{i+1} - \mathbf{b} - (\mathbf{A}\mathbf{x}_i - \mathbf{b}) = \mathbf{A}(\mathbf{x}_{i+1} - \mathbf{x}_i) = \alpha_i \mathbf{A} \mathbf{p}_i \tag{A.7.15}$$

and since $\alpha_i \neq 0$:

$$\mathbf{A}\mathbf{p}_i = (\mathbf{g}_{i+1} - \mathbf{g}_i)/\alpha_i. \tag{A.7.16}$$

So in equation (A.7.13):

$$\mathbf{p}_i^{\mathsf{T}} \mathbf{A} \mathbf{g}_{k+1} = \mathbf{g}_{k+1}^{\mathsf{T}} \mathbf{A} \mathbf{p}_i = \mathbf{g}_{k+1}^{\mathsf{T}} (\mathbf{g}_{i+1} - \mathbf{g}_i)/\alpha_i = (\mathbf{g}_{k+1}^{\mathsf{T}} \mathbf{g}_{i+1} - \mathbf{g}_{k+1}^{\mathsf{T}} \mathbf{g}_i)/\alpha_i \tag{A.7.17}$$

Since the $\mathbf{p}_i$ where obtained by applying the Gram-Schmidt procedure to the gradients $\mathbf{g}_i$, the subspace theorem $\mathbf{g}_{k+1}^{\mathsf{T}} \mathbf{p}_i = 0$, implies, also $\mathbf{g}_{k+1}^{\mathsf{T}} \mathbf{g}_i = 0$ for $i = 1, .., k$. This shows that

$$\mathbf{p}_i^{\mathsf{T}} \mathbf{A} \mathbf{g}_{k+1} = (\mathbf{g}_{k+1}^{\mathsf{T}} \mathbf{g}_{i+1} - \mathbf{g}_{k+1}^{\mathsf{T}} \mathbf{g}_i)/\alpha_i = \begin{cases} 0 & \text{if } 1 \leq i < k \\ \mathbf{g}_{k+1}^{\mathsf{T}} \mathbf{g}_{k+1}/\alpha_k & \text{if } i = k \end{cases} \tag{A.7.18}$$

Hence equation (A.7.13) simplifies to

$$\mathbf{p}_{k+1} = -\mathbf{g}_{k+1} + \frac{\mathbf{g}_{k+1}^{\mathsf{T}} \mathbf{g}_{k+1}/\alpha_k}{\mathbf{p}_k^{\mathsf{T}} \mathbf{A} \mathbf{p}_k} \mathbf{p}_k \,. \tag{A.7.19}$$

This can be brought into an even simpler form by applying equation (A.7.14) to α_k:

$$\mathbf{p}_{k+1} = -\mathbf{g}_{k+1} + \frac{\mathbf{g}_{k+1}^{\mathsf{T}} \mathbf{g}_{k+1}}{\mathbf{p}_k^{\mathsf{T}} \mathbf{A} \mathbf{p}_k} \frac{\mathbf{p}_k^{\mathsf{T}} \mathbf{A} \mathbf{p}_k}{\mathbf{g}_k^{\mathsf{T}} \mathbf{g}_k} \mathbf{p}_k = -\mathbf{g}_{k+1} + \frac{\mathbf{g}_{k+1}^{\mathsf{T}} \mathbf{g}_{k+1}}{\mathbf{g}_k^{\mathsf{T}} \mathbf{g}_k} \mathbf{p}_k \tag{A.7.20}$$

We shall write this in the form

$$\mathbf{p}_{k+1} = -\mathbf{g}_{k+1} + \beta_k \mathbf{p}_k \quad \text{where } \beta_k = \frac{\mathbf{g}_{k+1}^{\mathsf{T}} \mathbf{g}_{k+1}}{\mathbf{g}_k^{\mathsf{T}} \mathbf{g}_k} \,. \tag{A.7.21}$$

Algorithm 31 Conjugate Gradients for minimising a function $f(\mathbf{x})$

```
k = 1
Choose x_1.
p_1 = -g_1
while g_k ≠ 0 do
    α_k = argmin_{α_k} f(x_k + α_k p_k)          ▷ Line Search
    x_{k+1} := x_k + α_k p_k
    β_k := g_{k+1}^T g_{k+1} / (g_k^T g_k)
    p_{k+1} := -g_{k+1} + β_k p_k
    k = k + 1
end while
```

The formula (A.7.21) for β_k is due to Fletcher and Reeves. Since the gradients are orthogonal, β_k can also be written as

$$\beta_k = \frac{\mathbf{g}_{k+1}^\mathsf{T}(\mathbf{g}_{k+1} - \mathbf{g}_k)}{\mathbf{g}_k^\mathsf{T}\mathbf{g}_k}, \tag{A.7.22}$$

this is the Polak-Ribiere formula. The choice between the two expression for β_k can be of some importance if f is not quadratic.

A.7.5 Newton's method

Consider a function $f(\mathbf{x})$ that we wish to find the minimum of. A Taylor expansion up to second order gives

$$f(\mathbf{x} + \boldsymbol{\Delta}) = f(\mathbf{x}) + \boldsymbol{\Delta}^\mathsf{T}\nabla f + \frac{1}{2}\boldsymbol{\Delta}^\mathsf{T}\mathbf{H}\boldsymbol{\Delta} + O(|\boldsymbol{\Delta}|^3) \tag{A.7.23}$$

The matrix $\mathbf{H}$ is the Hessian. Differentiating the right hand side with respect to $\boldsymbol{\Delta}$ (or, equivalently, completing the square), we find that the right hand side has its lowest value when

$$\nabla f = \mathbf{H}\boldsymbol{\Delta} \Rightarrow \boldsymbol{\Delta} = \mathbf{H}^{-1}\nabla f \tag{A.7.24}$$

Hence, an optimisation routine to minimise E is given by the Newton update

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{H}^{-1}\nabla f \tag{A.7.25}$$

A benefit of Newton method over gradient descent is that the decrease in the objective function is invariant under a linear change of co-ordinates, $\mathbf{y} = \mathbf{M}\mathbf{x}$.

A.7.6 Quasi-Newton methods

For large-scale problems the inversion of the Hessian is computationally demanding, especially if the matrix is close to singular. An alternative is to set up the iteration

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k\mathbf{S}_k\mathbf{g}_k. \tag{A.7.26}$$

This is a very general form; if $\mathbf{S}_k = \mathbf{A}^{-1}$ then we have Newton's method, while if $\mathbf{S}_k = \mathbf{I}$ we have steepest descent. In general it would seem to be a good idea to choose $\mathbf{S}_k$ to be an approximation to the inverse Hessian. Also note that it is important that $\mathbf{S}_k$ be positive definite so that for small α_k we obtain a descent method. The idea behind most quasi-Newton methods is to try to construct an approximate inverse Hessian $\tilde{\mathbf{H}}_k$ using information gathered as the descent progresses, and to set $\mathbf{S}_k = \tilde{\mathbf{H}}_k$. As we have seen, for a quadratic optimization problem we have the relationship

$$\mathbf{g}_{k+1} - \mathbf{g}_k = \mathbf{A}(\mathbf{x}_{k+1} - \mathbf{x}_k) \tag{A.7.27}$$

Algorithm 32 Quasi-Newton for minimising a function $f(\mathbf{x})$

$k = 1$
Choose $\mathbf{x}_1$
$\tilde{\mathbf{H}}_1 = \mathbf{I}$
while $\mathbf{g}_k \neq \mathbf{0}$ **do**
  $\mathbf{p}_k = -\tilde{\mathbf{H}}_k \mathbf{g}_k$
  $\alpha_k = \underset{\alpha_k}{\text{argmin}}\ f(\mathbf{x}_k + \alpha_k \mathbf{p}_k)$ ▷ Line Search
  $\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k \mathbf{p}_k$
  $\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$, $\mathbf{y}_k = \mathbf{g}_{k+1} - \mathbf{g}_k$, and update $\tilde{\mathbf{H}}_{k+1}$
  $k = k + 1$
end while

Defining

$$\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k \qquad \text{and} \qquad \mathbf{y}_k = \mathbf{g}_{k+1} - \mathbf{g}_k \tag{A.7.28}$$

we see that equation A.7.27 becomes

$$\mathbf{y}_k = \mathbf{A}\mathbf{s}_k \tag{A.7.29}$$

It is reasonable to demand that

$$\tilde{\mathbf{H}}_{k+1}\mathbf{y}_i = \mathbf{s}_i \qquad 1 \leq i \leq k \tag{A.7.30}$$

After n linearly independent steps we would then have $\tilde{\mathbf{H}}_{n+1} = \mathbf{A}^{-1}$. For $k < n$ there are an infinity of solutions for $\tilde{\mathbf{H}}_{k+1}$ satisfying equation A.7.30. A popular choice is the Broyden-Fletcher-Goldfarb-Shanno (or BFGS) update, given by

$$\tilde{\mathbf{H}}_{k+1} = \tilde{\mathbf{H}}_k + \left(1 + \frac{\mathbf{y}_k^\mathsf{T}\tilde{\mathbf{H}}_k\mathbf{y}_k}{\mathbf{y}_k^\mathsf{T}\mathbf{s}_k}\right)\frac{\mathbf{s}_k\mathbf{s}_k^\mathsf{T}}{\mathbf{s}_k^\mathsf{T}\mathbf{y}_k} - \frac{\mathbf{s}_k\mathbf{y}_k^\mathsf{T}\tilde{\mathbf{H}}_k + \tilde{\mathbf{H}}_k\mathbf{y}_k\mathbf{s}_k^\mathsf{T}}{\mathbf{s}_k^\mathsf{T}\mathbf{y}_k} \tag{A.7.31}$$

This is a rank-2 correction to $\tilde{\mathbf{H}}_k$ constructed from the vectors $\mathbf{s}_k$ and $\tilde{\mathbf{H}}_k\mathbf{y}_k$.
The direction vectors $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_k$, $\mathbf{p}_k = -\tilde{\mathbf{H}}_k\mathbf{g}_k$, produced by the algorithm obey

$$\mathbf{p}_i^\mathsf{T}\mathbf{A}\mathbf{p}_j = 0 \qquad 1 \leq i < j \leq k \tag{A.7.32}$$

$$\tilde{\mathbf{H}}_{k+1}\mathbf{A}\mathbf{p}_i = \mathbf{p}_i \qquad 1 \leq i \leq k \tag{A.7.33}$$

Equation A.7.33 is called the hereditary property. In our notation $\mathbf{s}_k = \alpha_k\mathbf{p}_k$, and as the α's are non-zero, equation A.7.32 can also be written as

$$\mathbf{s}_i^\mathsf{T}\mathbf{A}\mathbf{s}_j = 0 \qquad 1 \leq i < j \leq k \tag{A.7.34}$$

Since the $\mathbf{p}_k$'s are $\mathbf{A}$-conjugate and since we successively minimize f in these directions, we see that the BFGS algorithm is a conjugate direction method; with the choice of $\mathbf{H}_1 = I$ it is in fact the conjugate gradient method. Note that the storage requirements for Quasi Newton methods scale quadratically with the number of variables, and hence tends to be used for smaller problems. Limited memory BFGS reduces the storage by only using the l latest updates in computing the approximate Hessian inverse, equation (A.7.31). In contrast, the memory requirements for pure Conjugate Gradient methods scale only linearly with the dimension of $\mathbf{x}$.

A.7 Constrained Optimisation using Lagrange multipliers

Single Constraint

Consider first the problem of minimising $f(\mathbf{x})$ subject to a single constraint $c(\mathbf{x}) = 0$. Imagine that we have already identified an $\mathbf{x}$ that satisfies the constraint, that is $c(\mathbf{x}) = 0$. How can we tell if this $\mathbf{x}$ minimises the function f? We are only allowed to search for lower function values around this $\mathbf{x}$ in directions which are consistent with the constraint. That is, $c(\mathbf{x}+\boldsymbol{\delta}) = 0$.

$$c(\mathbf{x}+\boldsymbol{\delta}) \approx c(\mathbf{x}) + \boldsymbol{\delta} \cdot \nabla c(\mathbf{x}) \tag{A.7.1}$$

Hence, in order that the constraint remains satisfied, we can only search in a direction such that $\boldsymbol{\delta} \cdot \nabla c(\mathbf{x}) = \mathbf{0}$, that is in directions $\boldsymbol{\delta}$ that are orthogonal to $\nabla c(\mathbf{x})$. So, let us explore the change in f along a direction $\mathbf{n}$ where $\mathbf{n} \cdot \nabla c(\mathbf{x}) = 0$,

$$f(\mathbf{x}+\epsilon\mathbf{n}) \approx f(\mathbf{x}) + \epsilon \nabla f(\mathbf{x}) \cdot \mathbf{n}. \tag{A.7.2}$$

Since we are looking for a point $\mathbf{x}$ that minimises the function f, we require $\mathbf{x}$ to be a stationary point, $\nabla f(\mathbf{x}) \cdot \mathbf{n} = 0$. That is, $\nabla f(\mathbf{x})$ must lie parallel to $\nabla c(\mathbf{x})$, so that

$$\nabla f(\mathbf{x}) = \lambda \nabla c(\mathbf{x}) \tag{A.7.3}$$

for some $\lambda \in \mathbb{R}$. To solve the optimisation problem therefore, we look for a point $\mathbf{x}$ such that $\nabla f(\mathbf{x}) = \lambda \nabla c(\mathbf{x})$, for some λ, and for which $c(\mathbf{x}) = 0$. An alternative formulation of this dual requirement is to look for $\mathbf{x}$ and λ that jointly minimise the *Lagrangian*

$$\mathcal{L}(\mathbf{x}, \lambda) = f(\mathbf{x}) - \lambda c(\mathbf{x}) \tag{A.7.4}$$

Differentiating with respect to $\mathbf{x}$, we get the requirement $\nabla f(\mathbf{x}) = \lambda \nabla c(\mathbf{x})$, and differentiaing with respect to λ, we get that $c(\mathbf{x}) = 0$.

Multiple constraints

Consider the problem of optimising $f(\mathbf{x})$ subject to the constraints $c_i(\mathbf{x}) = 0$, $i = 1, \ldots, r < n$, where n is the dimensionality of the space. Denote by S the $n - r$ dimensional subspace of $\mathbf{x}$ which obeys the constraints. Assume that $\mathbf{x}^*$ is such an optimum. As in the unconstrained case, we consider perturbations $\mathbf{v}$ to $\mathbf{x}^*$, but now such that $\mathbf{v}$ lies in S

$$c_i(\mathbf{x}^* + h\mathbf{v}) = c_i(\mathbf{x}^*) + \mathbf{v}^{\mathsf{T}} \nabla c_i(\mathbf{x}^*) + O(h^2) \tag{A.7.5}$$

Let $\mathbf{a}_i^* = \nabla c_i(\mathbf{x}^*)$. Thus for the perturbation to stay within S, we require that $\mathbf{v}^{\mathsf{T}}\mathbf{a}_i^* = 0$ for all $i = 1, \ldots r$. Let $\mathbf{A}^*$ be the matrix whose columns are $\mathbf{a}_1^*, \mathbf{a}_2^*, \ldots \mathbf{a}_r^*$. Then this condition can be rewritten as $\mathbf{A}^*\mathbf{v} = \mathbf{0}$. We also require for a local optimum that $\mathbf{v}^{\mathsf{T}} \nabla f = 0$ for all $\mathbf{v}$ in S. We see that ∇f must be orthogonal to $\mathbf{v}$, and that $\mathbf{v}$ must be orthogonal to the $\mathbf{a}_i^*$'s. Thus ∇f must be a linear combination of the $\mathbf{a}_i^*$'s, *i.e.*

$$\nabla f = \sum_{i=1}^{r} \lambda_i^* \mathbf{a}_i^* = {\mathbf{A}^*}^{\mathsf{T}} \boldsymbol{\lambda}^* \tag{A.7.6}$$

Geometrically this says that the gradient vector is normal to the tangent plane to S at $\mathbf{x}^*$. These conditions give rise to the method of Lagrange multipliers for optimisation problems with equality constraints. The method requires finding $\mathbf{x}^*$ and $\boldsymbol{\lambda}^*$ which solve the equations

$$\nabla f = \sum_i \mathbf{a}_i(\mathbf{x}) \lambda_i \tag{A.7.7}$$

$$c_i(\mathbf{x}) = 0 \quad \text{for } i = 1, \ldots r \tag{A.7.8}$$

There are $n + r$ equations and $n + r$ unknowns, so the system is well-determined. However, the system is nonlinear (in $\mathbf{x}$) in general, and so may not be easy to solve. We can restate these conditions by introducing the Lagrangian function

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) - \sum_i \lambda_i c_i(\mathbf{x}). \tag{A.7.9}$$

The partial derivatives of $\mathcal{L}$ with respect to $\mathbf{x}$ and $\boldsymbol{\lambda}$ reproduce equations A.7.7 and A.7.8. Hence a necessary condition for a local minimizer is that $\mathbf{x}^*$, $\boldsymbol{\lambda}^*$ is a stationary point of the Lagrangian function. Note that this stationary point is not a minimum but a saddle point, as $\mathcal{L}$ depends linearly on $\boldsymbol{\lambda}$. We have given first-order necessary and sufficient conditions for a local optimum. To show that this optimum is a local minimum, we would need to consider second-order conditions, analogous to the positive definiteness of the Hessian in the unconstrained case; this can be done, but will not be considered here.

Bibliography

[1] L. F. Abbott, J. A. Varela, K. Sen, and S. B. Nelson. Synaptic depression and cortical gain control. *Science*, 275:220–223, 1997.

[2] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. A Learning Algorithm for Boltzmann Machines. *Cognitive Science*, 9:147–169, 1985.

[3] R. P. Adams and D. J. C. MacKay. Bayesian Online Changepoint Detection. Cavendish laboratory, department of physics, University of Cambridge, Cambridge, UK, 2006. arXiv:0710.3742v1 [stat.ML].

[4] E. Airoldi, D. Blei, E. Xing, and S. Fienberg. A latent mixed membership model for relational data. In *LinkKDD '05: Proceedings of the 3rd international workshop on Link discovery*, pages 82–89, New York, NY, USA, 2005. ACM.

[5] E. M. Airoldi, D. M. Blei, S. E. Fienberg, and E. P. Xing. Mixed membership stochastic blockmodels. *Journal of Machine Learning Research*, 9:1981–2014, 2008.

[6] D. L. Alspach and H. W. Sorenson. Nonlinear Bayesian Estimation Using Gaussian Sum Approximations. *IEEE Transactions on Automatic Control*, 17(4):439–448, 1972.

[7] S-i. Amari. Natural Gradient Works Efficiently in Learning. *Neural Computation*, 10(2):251–276, 1998.

[8] S-i. Amari. Natural Gradient Learning for Over and Under-Complete Bases in ICA. *Neural Computation*, 11:1875–1883, 1999.

[9] I. Androutsopoulos, J. Koutsias, K. V. Chandrinos, and C. D. Spyropoulos. An experimental comparison of naive bayesian and keyword-based anti-spam filtering with personal e-mail messages. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 160–167, New York, NY, USA, 2000. ACM.

[10] S. Arora and C. Lund. Hardness of approximations. In *Approximation algorithms for NP-hard problems*, pages 399–446. PWS Publishing Co., Boston, MA, USA, 1997.

[11] F. R. Bach and M. I. Jordan. Thin junction trees. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems (NIPS)*, number 14, pages 569–576, Cambridge, MA, 2001. MIT Press.

[12] F. R. Bach and M. I. Jordan. A probabilistic interpretation of canonical correlation analysis. Computer Science Division and Department of Statistics 688, University of California Berkeley, Berkeley, USA, 2005.

[13] Y. Bar-Shalom and Xiao-Rong Li. *Estimation and Tracking : Principles, Techniques and Software.* Artech House, Norwood, MA, 1998.

[14] D. Barber. Dynamic Bayesian Networks with Deterministic Tables. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems (NIPS)*, number 15, pages 713–720, Cambridge, MA, 2003. MIT Press.

[15] D. Barber. Learning in Spiking Neural Assemblies. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems (NIPS)*, number 15, pages 149–156, Cambridge, MA, 2003. MIT Press.

[16] D. Barber. Are two Classifiers performing equally? A treatment using Bayesian Hypothesis Testing. IDIAP-RR 57, IDIAP, Rue de Simplon 4, Martigny, CH-1920, Switerland, May 2004. IDIAP-RR 04-57.

[17] D. Barber. Expectation Correction for smoothing in Switching Linear Gaussian State Space models. *Journal of Machine Learning Research*, 7:2515–2540, 2006.

[18] D. Barber. Clique Matrices for Statistical Graph Decomposition and Parameterising Restricted Positive Definite Matrices. In D. A. McAllester and Myllym editors, *Uncertainty in Artificial Intelligence*, number 24, pages 26–33, Corvallis, Oregon, USA, 2008. AUAI press.

[19] D. Barber and F. V. Agakov. Correlated sequence learning in a network of spiking neurons using maximum likelihood. Informatics Research Reports EDI-INF-RR-0149, Edinburgh University, 2002.

[20] D. Barber and F.V. Agakov. The IM Algorithm: A variational approach to Information Maximization. In *Advances in Neural Information Processing Systems (NIPS)*, number 16, 2004.

[21] D. Barber and C. M. Bishop. Bayesian Model Comparison by Monte Carlo Chaining. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems NIPS 9*. MIT Press, 1997. ISSN 1049 5258 ISBN 0 262 20107 0.

[22] D. Barber and C. M. Bishop. Ensemble Learning in Bayesian Neural Networks. In *Neural Networks and Machine Learning*, pages 215–237. Springer, 1998.

[23] D. Barber and S. Chiappa. Unified Inference for Variational Bayesian Linear Gaussian State-Space Models. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems (NIPS)*, number 19, pages 81–88, Cambridge, MA, 2007. MIT Press.

[24] D. Barber and W. Wiegerinck. Tractable Variational Structures for Approximating Graphical Models. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems (NIPS)*, number 11, pages 183–189, Cambridge, MA, 1999. MIT Press.

[25] D. Barber and C. K. I. Williams. Gaussian processes for Bayesian classification via hybrid Monte Carlo. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems NIPS 9*, pages 340–346, Cambridge, MA, 1997. MIT Press.

[26] R. J. Baxter. *Exactly solved models in statistical mechanics*. Academic Press, 1982.

[27] M. J. Beal, F. Falciani, Z. Ghahramani, C. Rangel, and D. L. Wild. A Bayesian approach to reconstructing genetic regulatory networks with hidden factors. *Bioinformatics*, (21):349–356, 2005.

[28] A. Becker and D. Geiger. A sufficiently fast algorithm for finding close to optimal clique trees. *Artificial Intelligence*, 125(1-2):3–17, 2001.

[29] A. J. Bell and T. J. Sejnowski. An Information-Maximization Approach to Blind Separation and Blind Deconvolution. *Neural Computation*, 7(6):1129–1159, 1995.

[30] R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957. Paperback edition by Dover Publications (2003).

[31] Y. Bengio and P. Frasconi. Input-Output HMMs for sequence processing. *IEEE Trans. Neural Networks*, (7):1231–1249, 1996.

[32] A. L. Berger, S. D. Della Pietra, and V. J. D. Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71, 1996.

[33] J. O. Berger. *Statistical Decision Theory and Bayesian Analysis*. Springer, second edition, 1985.

[34] D. P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, second edition, 2000.

[35] J. Besag. Spatial Interactions and the Statistical Analysis of Lattice Systems. *Journal of the Royal Statistical Society, Series B*, 36(2):192–236, 1974.

[36] J. Besag. On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society, Series B*, 48:259–302, 1986.

[37] J. Besag and P. Green. Spatial statistics and Bayesian computation. *Journal of the Royal Statistical Society, Series B*, 55:25–37, 1993.

[38] G. J. Bierman. Measurement updating using the U-D factorization. *Automatica*, 12:375–382, 1976.

[39] N. L. Biggs. *Discrete Mathematics.* Oxford University Press, 1990.

[40] K. Binder and A. P. Young. Spin glasses: Experimental facts, theoretical concepts, and open questions. *Rev. Mod. Phys.*, 58(4):801–976, Oct 1986.

[41] C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

[42] C. M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.

[43] C. M. Bishop and M. Svensén. Bayesian hierarchical mixtures of experts. In U. Kjaerulff and C. Meek, editors, *Proceedings Nineteenth Conference on Uncertainty in Artificial Intelligence*, pages 57–64. Morgan Kaufmann, 2003.

[44] D. Blei, A. Ng, and M. Jordan. Latent Dirichlet allocation. *Journal of machine Learning Research*, (3):993–1022, 2003.

[45] R. R. Bouckaert. *Bayesian belief networks: from construction to inference.* PhD thesis, University of Utrecht, 1995.

[46] S. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004.

[47] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(9):1124–1137, 2004.

[48] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23:1222–1239, 2001.

[49] M. Brand. Incremental singular value decomposition of uncertain data with missing values. In *European Conference on Computer Vision (ECCV)*, pages 707–720, 2002.

[50] J. Breese and D. Heckerman. Decision-theoretic troubleshooting: A framework for repair and experiment. In E. Horvitz and F. Jensen, editors, *Uncertainty in Artificial Intelligence*, number 12, pages 124–132, San Francisco, CA, 1996. Morgan Kaufmann.

[51] W. Buntine. Theory refinement on Bayesian networks. In *Uncertainty in Artificial Intelligence*, number 7, pages 52–60, San Francisco, CA, 1991. Morgan Kaufmann.

[52] A. Cano and S. Moral. *Advances in Intelligent Computing – IPMU 1994*, chapter Heuristic Algorithms for the Triangulation of Graphs, pages 98–107. Number 945 in Lectures Notes in Computer Sciences. Springer-Verlag, 1995.

[53] O. Cappé, E. Moulines, and T. Ryden. *Inference in Hidden Markov Models.* Springer, New York, 2005.

[54] E. Castillo, J. M. Gutierrez, and A. S. Hadi. *Expert Systems and Probabilistic Network Models.* Springer, 1997.

[55] A. T. Cemgil. Bayesian Inference in Non-negative Matrix Factorisation Models. Technical Report CUED/F-INFENG/TR.609, University of Cambridge, July 2008.

[56] A. T. Cemgil, B. Kappen, and D. Barber. A Generative Model for Music Transcription. *IEEE Transactions on Audio, Speech and Language Processing*, 14(2):679–694, 2006.

[57] H. S. Chang, M. C. Fu, J. Hu, and S. I. Marcus. *Simulation-based Algorithms for Markov Decision Processes.* Springer, 2007.

[58] S. Chiappa and D. Barber. Bayesian Linear Gaussian State Space Models for Biosignal Decomposition. *Signal Processing Letters*, 14(4):267–270, 2007.

[59] S. Chib and M. Dueker. Non-Markovian Regime Switching with Endogenous States and Time-Varying State Strengths. Econometric Society 2004 North American Summer Meetings 600, Econometric Society, August 2004.

[60] C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968.

[61] P. S. Churchland and T. J. Sejnowski. *The Computational Brain.* MIT Press, Cambridge, MA, USA, 1994.

[62] D. Cohn and H. Chang. Learning to probabilistically identify authoritative documents. In P. Langley, editor, *International Conference on Machine Learning*, number 17, pages 167–174. Morgan Kaufmann, 2000.

[63] D. Cohn and T. Hofmann. The Missing Link - A Probabilistic Model of Document Content and Hypertext Connectivity. Number 13, pages 430–436, Cambridge, MA, 2001. MIT Press.

[64] A. C. C. Coolen, R. Kühn, and P. Sollich. *Theory of Neural Information Processing Systems*. Oxford University Press, 2005.

[65] G. F. Cooper and E. Herskovits. A Bayesian Method for the Induction of Probabilistic Networks from Data. *Machine Learning*, 9(4):309–347, 1992.

[66] A. Corduneanu and C. M. Bishop. Variational Bayesian Model Selection for Mixture Distributions. In T. Jaakkola and T. Richardson, editors, *Artifcial Intelligence and Statistics*, pages 27–34. Morgan Kaufmann, 2001.

[67] M. T. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley, 1991.

[68] R. G. Cowell, A. P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer, 1999.

[69] D. R. Cox and N. Wermuth. *Multivariate Dependencies*. Chapman and Hall, 1996.

[70] N. Cristianini and J. Shawe-Taylor. *An Introduction To Support Vector Machines*. Cambridge University Press, 2000.

[71] P. Dangauthier, R. Herbrich, T. Minka, and T. Graepel. Trueskill through time: Revisiting the history of chess. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems (NIPS)*, number 19, pages 569–576, Cambridge, MA, 2007. MIT Press.

[72] H. A. David. *The method of paired comparisons*. Oxford University Press, New York, 1988.

[73] A. P. Dawid. Influence diagrams for causal modelling and inference. *International Statistical Review*, 70:161–189, 2002.

[74] A. P. Dawid and S. L. Lauritzen. Hyper Markov Laws in the Statistical Analysis of Decomposable Graphical Models. *Annals of Statistics*, 21(3):1272–1317, 1993.

[75] P. Dayan and L.F. Abbott. *Theoretical Neuroscience*. MIT Press, 2001.

[76] P. Dayan and G. E. Hinton. Using Expectation-Maximization for Reinforcement Learning. *Neural Computation*, 9:271–278, 1997.

[77] T. De Bie, N. Cristianini, and R. Rosipal. *Handbook of Geometric Computing : Applications in Pattern Recognition, Computer Vision, Neuralcomputing, and Robotics*, chapter Eigenproblems in Pattern Recognition. Springer-Verlag, 2005.

[78] R. Dechter. Bucket Elimination: A unifying framework for probabilistic inference algorithms. In E. Horvitz and F. Jensen, editors, *Uncertainty in Artificial Intelligence*, pages 211–219, San Francisco, CA, 1996. Morgan Kaufmann.

[79] S. Diederich and M. Opper. Learning of Correlated Patterns in Spin-Glass Networks by Local Learning Rules. *Physical Review Letters*, 58(9):949–952, 1986.

[80] R. Diestel. *Graph Theory*. Springer, 2005.

[81] A. Doucet and A. M. Johansen. A Tutorial on Particle Filtering and Smoothing: Fifteen years later. In D. Crisan and B. Rozovsky, editors, *Oxford Handbook of Nonlinear Filtering*. Oxford University Press, 2009.

[82] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley-Interscience Publication, 2000.

[83] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis : Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1999.

[84] A. Düring, A. C. C. Coolen, and D. Sherrington. Phase diagram and storage capacity of sequence processing neural networks. *Journal of Physics A*, 31:8607–8621, 1998.

[85] J. M. Gutierrez E. Castillo and A. S. Hadi. *Expert Systems and Probabilistic Network Models*. Springer Verlag, 1997.

[86] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, 1972.

[87] R. Edwards and A. Sokal. Generalization of the fortium-kasteleyn-swendson-wang representation and monte carlo algorithm. *Physical Review D*, 38:2009–2012, 1988.

[88] A. E. Elo. *The rating of chess players, past and present.* Arco, New York, second edition, 1986.

[89] E. Erosheva, S. Fienberg, and J. Lafferty. Mixed membership models of scientific publications. In *Proceedings of the National Academy of Sciences*, volume 101, pages 5220–5227, 2004.

[90] R-E. Fan, P-H. Chen, and C-J. Lin. Working Set Selection Using Second Order Information for Training Support Vector Machines. *Journal of Machine Learning Research*, 6:1889–1918, 2005.

[91] G. H. Fischer and I. W. Molenaar. *Rasch Models: Foundations, Recent Developments, and Applications.* Springer, New York, 1995.

[92] M. E. Fisher. Statistical Mechanics of Dimers on a Plane Lattice. *Physical Review*, 124:1664 – 1672, 1961.

[93] B. Frey. Extending Factor Graphs as to Unify Directed and Undirected Graphical Models. In C. Meek and U. Kjærulff, editors, *Uncertainty in Artificial Intelligence*, number 19, pages 257–264. Morgan Kaufmann, 2003.

[94] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian Network Classifiers. *Machine Learning*, 29:131–163, 1997.

[95] S. Frühwirth-Schnatter. *Finite Mixture and Markov Switching Models.* Springer, 2006.

[96] M. Frydenberg. The chain graph Markov property. *Scandanavian Journal of Statistics*, 17:333–353, 1990.

[97] T. Furmston and D. Barber. Solving deterministic policy (PO)MDPs using Expectation-Maximisation and Antifreeze. In E. Suzuki and M. Sebag, editors, *European Conference on Machine Learning and Principles and Practice of Knowledge discovery in Databases*, September 2009. Workshop on Learning and Data Mining for Robotics.

[98] A. Galka, O. Yamashita, T. Ozaki, R. Biscay, and P. Valdes-Sosa. A solution to the dynamical inverse problem of EEG generation using spatiotemporal Kalman filtering. *NeuroImage*, (23):435–453, 2004.

[99] P. Gandhi, F. Bromberg, and D. Margaritis. Learning markov network structure using few independence tests. In *Proceedings of the SIAM International Conference on Data Mining*, pages 680–691, 2008.

[100] M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979.

[101] A. Gelb. *Applied optimal estimation.* MIT press, 1974.

[102] A. Gelman, G. O. Roberts, and W. R. Gilks. Efficient Metropolis jumping rules. In J. O. Bernardo, J. M. Berger, A. P. Dawid, and A. F. M. Smith, editors, *Bayesian Statistics*, volume 5, pages 599–607. Oxford University Press, 1996.

[103] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. In *Readings in uncertain reasoning*, pages 452–472, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc.

[104] M. G. Genton. Classes of kernels for machine learning: A statistics perspective. *Journal of Machine Learning Research*, 2:299–312, 2001.

[105] W. Gerstner and W. M. Kistler. *Spiking Neuron Models.* Cambridge University Press, 2002.

[106] Z. Ghahramani and M. J. Beal. Variational Inference for Bayesian Mixtures of Factor Analysers. In S. A. Solla, T. K. Leen, and K-R. Müller, editors, *Advances in Neural Information Processing Systems (NIPS)*, number 12, pages 449–455, Cambridge, MA, 2000. MIT Press.

[107] Z. Ghahramani and G. E. Hinton. Variational learning for switching state-space models. *Neural Computation*, 12(4):963–996, 1998.

[108] A. Gibbons. *Algorithmic Graph Theory.* Cambridge University Press, 1991.

[109] W. R. Gilks, S. Richardson, and D. J. Spiegelhalter. *Markov chain Monte Carlo in practice*. Chapman & Hall, 1996.

[110] M. Girolami and A. Kaban. On an equivalence between PLSI and LDA. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 433–434, New York, NY, USA, 2003. ACM Press.

[111] M. E. Glickman. Parameter estimation in large dynamic paired comparison experiments. *Applied Statistics*, 48:377–394, 1999.

[112] A. Globerson and T. Jaakkola. Approximate inference using planar graph decomposition. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems (NIPS)*, number 19, pages 473–480, Cambridge, MA, 2007. MIT Press.

[113] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *Communications ACM*, 35:61–70, 1992.

[114] G. H. Golub and C. F. van Loan. *Matrix Computations*. Johns Hopkins University Press, 3rd edition, 1996.

[115] M. C. Golumbic and I. Ben-Arroyo Hartman. *Graph Theory, Combinatorics, and Algorithms*. Springer-Verlag, 2005.

[116] C. Goutis. A graphical method for solving a decision analysis problem. *IEEE Transactions on Systems, Man and Cybernetics*, 25:1181–1193, 1995.

[117] P. J. Green and B. W. Silverman. *Nonparametric Regression and Generalized Linear Models*, volume 58 of *Monographs on Statistics and Applied Probability*. Chapman and Hall, 1994.

[118] D. M. Greig, B. T. Porteous, and A. H. Seheult. Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society, Series B*, 2:271–279, 1989.

[119] M. S. Grewal and A. P. Andrews. *Kalman filtering: theory and practice*. Prentice-Hall, 1993.

[120] G. Grimmett and D. Stirzaker. *Probability and Random Processes*. Oxford University Press, second edition, 1992.

[121] S. F. Gull. Bayesian data analysis: straight-line fitting. In J. Skilling, editor, *Maximum entropy and Bayesian methods (Cambridge 1988)*, pages 511–518. Kluwer, 1989.

[122] A. K. Gupta and D. K. Nagar. *Matrix Variate Distributions*. Chapman and Hall/CRC, Boca Raton, Florida USA, 1999.

[123] D. J. Hand and K. Yu. Idiot's Bayes—Not So Stupid After All? *International Statistical Review*, 69(3):385–398, 2001.

[124] D. R. Hardoon, S. Szedmak, and J. Shawe-Taylor. Canonical Correlation Analysis: An Overview with Application to Learning Methods. *Neural Computation*, 16(12):2639–2664, 2004.

[125] D. O. Hebb. *The organization of behavior*. Wiley, New York, 1949.

[126] D. Heckerman. A Tutorial on Learning With Bayesian Networks. Technical Report MSR-TR-95-06, Microsoft Research, Redmond, WA, March 1996. Revised November 1996.

[127] D. Heckerman, D. Geiger, and D. Chickering. Learning Bayesian Networks: The Combination of Knowledge and Statistical Data. *Machine Learning*, 20(3):197–243, 1995.

[128] R. Herbrich, T. Minka, and T. Graepel. TrueSkill$^{\mathbf{TM}}$: A Bayesian Skill Rating System. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems (NIPS)*, number 19, pages 569–576, Cambridge, MA, 2007. MIT Press.

[129] H. Hermansky. Should recognizers have ears? *Speech Communication*, 25:3–27, 1998.

[130] J. Hertz, A. Krogh, and R. Palmer. *Introduction to the theory of Neural Computation*. Addison-Wesley, 1991.

[131] T. Heskes. Convexity arguments for efficient minimization of the Bethe and Kikuchi free energies. *Journal of Artificial Intelligence Research*, 26:153–190, 2006.

[132] L. J. Heyer, S. Kruglyak, and S. Yooseph. Exploring Expression Data: Identification and Analysis of Coexpressed Genes. *Genome Research*, 9:1106–1115, 1999.

[133] D. M. Higdon. Auxiliary variable methods for Markov chain Monte Carlo with applications. *Journal of the American Statistical Association*, 93(442):585–595, 1998.

[134] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, (313):504–507, 2006.

[135] T. Hofmann, J. Puzicha, and M. I. Jordan. Learning from dyadic data. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems (NIPS)*, pages 466–472, Cambridge, MA, 1999. MIT Press.

[136] R. A. Howard and J. E. Matheson. Influence diagrams. *Decision Analysis*, 2(3), 2005. Republished version of the original 1981 report.

[137] A. Hyvärinen, J. Karhunen, and E. Oja. *Independent Component Analysis*. Wiley, 2001.

[138] Aapo Hyvärinen. Consistency of Pseudolikelihood Estimation of Fully Visible Boltzmann Machines. *Neural Computation*, 18(10):2283–2292, 2006.

[139] M. Isard and A. Blake. CONDENSATION Conditional Density Propagation for Visual Tracking. *International Journal of Computer Vision*, 29:5–28, 1998.

[140] T. S. Jaakkola and M. I. Jordan. Variational probabilistic inference and the qmr-dt network. *Journal of Artificial Intelligence Research*, 10:291–322, 1999.

[141] T. S. Jaakkola and M. I. Jordan. Bayesian parameter estimation via variational methods. *Statistics and Computing*, 10(1):25–37, 2000.

[142] R. A. Jacobs, F. Peng, and M. A. Tanner. A Bayesian approach to model selection in hierarchical mixtures-of-experts architectures. *Neural Networks*, 10(2):231–241, 1997.

[143] E. T. Jaynes. *Probability Theory : The Logic of Science*. Cambridge University Press, 2003.

[144] F. Jensen, F. V. Jensen, and D. Dittmer. From influence diagrams to junction trees. In *Proceedings of the 10th Annual Conference on Uncertainty in Artificial Intelligence (UAI-94)*, pages 367–373, San Francisco, CA, 1994. Morgan Kaufmann.

[145] F. V. Jensen and F. Jensen. Optimal Junction Trees. In R. Lopez de Mantaras and D. Poole, editors, *Uncertainty in Artificial Intelligence*, number 10, pages 360–366, San Francisco, CA, 1994. Morgan Kaufmann.

[146] F. V. Jensen and T. D. Nielson. *Bayesian Networks and Decision Graphs*. Springer Verlag, second edition, 2007.

[147] M. I. Jordan and R. A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6:181–214, 1994.

[148] B. H. Juang, W. Chou, and C. H. Lee. Minimum classification error rate methods for speech recognition. *IEEE Transactions on Speech and Audio Processing*, 5:257–265, 1997.

[149] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

[150] H. J. Kappen. An introduction to stochastic control theory, path integrals and reinforcement learning. In *Proceedings 9th Granada seminar on Computational Physics: Computational and Mathematical Modeling of Cooperative Behavior in Neural Systems*, volume 887, pages 149–181. American Institute of Physics, 2007.

[151] H. J. Kappen and W. Wiegerinck. Novel iteration schemes for the Cluster Variation Method. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems (NIPS)*, number 14, pages 415–422, Cambridge, MA, 2002. MIT Press.

[152] Y. Karklin and M. S. Lewicki. Emergence of complex cell properties by learning to generalize in natural scenes. *Nature*, (457):83–86, November 2008.

[153] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *Siam Journal on Scientific Computing*, 20(1):359–392, 1998.

[154] P. W. Kasteleyn. Dimer Statistics and Phase Transitions. *Journal of Mathematical Physics*, 4(2):287–293, 1963.

[155] S. A. Kauffman. *At Home in the Universe: The Search for Laws of Self-Organization and Complexity.* Oxford University Press, Oxford, UK, 1995.

[156] C-J. Kim. Dynamic linear models with Markov-switching. *Journal of Econometrics*, 60:1–22, 1994.

[157] C-J. Kim and C. R. Nelson. *State-Space models with regime switching.* MIT Press, 1999.

[158] G. Kitagawa. The Two-Filter Formula for Smoothing and an implementation of the Gaussian-sum smoother. *Annals of the Institute of Statistical Mathematics*, 46(4):605–623, 1994.

[159] U. B. Kjaerulff and A. L. Madsen. *Bayesian Networks and Influence Diagrams : A Guide to Construction and Analysis.* Springer, 2008.

[160] A. Krogh, M. Brown, I. Mian, K. Sjolander, and D. Haussler. Hidden Markov models in computational biology: Applications to protein modeling. *Journal of Molecular Biology*, 235:1501–1531, 1994.

[161] S. Kullback. *Information Theory and Statistics.* Dover, 1968.

[162] K. Kurihara, M. Welling, and Y. W. Teh. Collapsed Variational Dirichlet Process Mixture Models. In *Proceedings of the International Joint Conference on Artificial Intelligence*, volume 20, pages 2796–2801, 2007.

[163] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In C. E. Brodley and A. P. Danyluk, editors, *International Conference on Machine Learning*, number 18, pages 282–289, San Francisco, CA, 2001. Morgan Kaufmann.

[164] H. Lass. *Elements of Pure and Applied Mathematics.* McGraw-Hill (reprinted by Dover), 1957.

[165] S. L. Lauritzen. *Graphical Models.* Oxford University Press, 1996.

[166] S. L. Lauritzen, A. P. Dawid, B. N. Larsen, and H-G. Leimer. Independence properties of directed Markov fields. *Networks*, 20:491–505, 1990.

[167] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of Royal Statistical Society B*, 50(2):157 – 224, 1988.

[168] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems (NIPS)*, number 13, pages 556–562, Cambridge, MA, 2001. MIT Press.

[169] M. A. R. Leisink and H. J. Kappen. A Tighter Bound for Graphical Models. In *Neural Computation*, volume 13, pages 2149–2171. MIT Press, 2001.

[170] V. Lepar and P. P. Shenoy. A Comparison of Lauritzen-Spiegelhalter, Hugin, and Shenoy-Shafer Architectures for Computing Marginals of Probability Distributions. In G. Cooper and S. Moral, editors, *Uncertainty in Artificial Intelligence*, number 14, pages 328–333, San Francisco, CA, 1998. Morgan Kaufmann.

[171] U. Lerner, R. Parr, D. Koller, and G. Biswas. Bayesian Fault Detection and Diagnosis in Dynamic Systems. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AIII-00)*, pages 531–537, 2000.

[172] U. N. Lerner. *Hybrid Bayesian Networks for Reasoning about Complex Systems.* Computer science department, Stanford University, 2002.

[173] R. Linsker. Improved local learning rule for information maximization and related applications. *Neural Networks*, 18(3):261–265, 2005.

[174] Y. L. Loh, E. W. Carlson, and M. Y. J. Tan. Bond-propagation algorithm for thermodynamic functions in general two-dimensional Ising models. *Physical Review B*, 76(1):014404, 2007.

[175] H. Lopes and M. West. Bayesian model assessment in factor analysis. *Statistica Sinica*, (14):41–67, 2003.

[176] T. J. Loredo. From Laplace To Supernova Sn 1987A: Bayesian Inference In Astrophysics. In P.F. Fougere, editor, *Maximum Entropy and Bayesian Methods*, pages 81–142. Kluwer, 1990.

[177] D. J. C. MacKay. Bayesian interpolation. *Neural Computation*, 4(3):415–447, 1992.

[178] D. J. C. MacKay. Probable Networks and plausisble predictions – a review of practical Bayesian methods for supervised neural networks. *Network: Computation in Neural Systems*, 6(3):469–505, 1995.

[179] D. J. C. MacKay. Introduction to Gaussian Processes. In *Neural Networks and Machine Learning*, volume 168 of *NATO advanced study institute on generalization in neural networks and machine learning*, pages 133–165. Springer, August 1998.

[180] D. J. C. MacKay. *Information Theory, Inference and Learning Algorithms.* Cambridge University Press, 2003.

[181] U. Madhow. *Fundamentals of Digital Communication.* Cambridge University Press, 2008.

[182] K. V. Mardia, J. T. Kent, and J. M. Bibby. *Multivariate Analysis.* Academic Press, 1997.

[183] H. Markram, J. Lubke, M. Frotscher, and B. Sakmann. Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. *Science*, 275:213–215, 1997.

[184] G. McLachlan and T. Krishnan. *The EM Algorithm and Extensions.* John Wiley and Sons, 1997.

[185] G. McLachlan and D. Peel. *Finite Mixture Models.* Wiley Series in Probability and Statistics. Wiley-Interscience, 2000.

[186] E. Meeds, Z. Ghahramani, R. M. Neal, and S. T. Roweis. Modeling Dyadic Data with Binary Latent Factors. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 19, pages 977–984, Cambridge, MA, 2007. MIT Press.

[187] M. Meila. An Accelerated Chow and Liu Algorithm: Fitting Tree Distributions to High-Dimensional Sparse Data. In I. Bratko, editor, *International Conference on Machine Learning*, pages 249–257, San Francisco, CA, 1999. Morgan Kaufmann.

[188] M. Meila and M. I. Jordan. Triangulation by continuous embedding. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems (NIPS)*, number 9, pages 557–563, Cambridge, MA, 1997. MIT Press.

[189] B. Mesot and D. Barber. A Bayesian Alternative to Gain Adaptation in Autoregressive Hidden Markov Models. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages II:437–440, 2007.

[190] B. Mesot and D. Barber. Switching Linear Dynamical Systems for Noise Robust Speech Recognition. *IEEE Transactions of Audio, Speech and Language Processing*, 15(6):1850–1858, 2007.

[191] N. Meuleau, M. Hauskrecht, K-E. Kim, L. Peshkin, Kaelbling. L. P., T. Dean, and C. Boutilier. Solving Very Large Weakly Coupled Markov Decision Processes. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 165–172, 1998.

[192] T. Mills. *The Econometric Modelling of Financial Time Series.* Cambridge University Press, 2000.

[193] T. Minka. Expectation Propagation for approximate Bayesian inference. In J. Breese and D. Koller, editors, *Uncertainty in Artificial Intelligence*, number 17, pages 362–369, San Francisco, CA, 2001. Morgan Kaufmann.

[194] T. Minka. A comparison of numerical optimizers for logistic regression. Technical report, Microsoft Research, 2003. research.microsoft.com/∼minka/papers/logreg.

[195] T. Minka. Divergence measures and message passing. Technical Report MSR-TR-2005-173, Microsoft Research Ltd., Cambridge, UK, December 2005.

[196] A. Mira, J. Møller, and G. O. Roberts. Perfect slice samplers. *Journal of the Royal Statistical Society*, 63(3):593–606, 2001. Series B (Statistical Methodology).

[197] C. Mitchell, M. Harper, and L. Jamieson. On the complexity of explicit duration HMM's. *Speech and Audio Processing, IEEE Transactions on*, 3(3):213–217, May 1995.

[198] T. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

[199] J. Mooij and H. J. Kappen. Sufficient conditions for convergence of Loopy Belief Propagation. *IEEE Information Theory*, 53:4422–4437, 2007.

[200] A. Moore. A tutorial on kd-trees. Technical report, 1991. Available from http://www.cs.cmu.edu/∼awm/papers.html.

[201] J. Moussouris. Gibbs and Markov Random Systems with Constraints. *Journal of Statistical Physics*, 10:11–33, 1974.

[202] R. M. Neal. Connectionist Learning of Belief Networks. *Artificial Intelligence*, 56:71–113, 1992.

[203] R. M. Neal. Probabilistic inference using Markov Chain Monte Carlo methods. CRG-TR-93-1, Dept. of Computer Science, University of Toronto, 1993.

[204] R. M. Neal. Markov Chain Sampling Methods for Dirichlet Process Mixture Models. *Journal of Computational and Graphical Statistics*, 9(2):249–265, 2000.

[205] R. M. Neal. Slice sampling. *Annals of Statistics*, 31:705–767, 2003.

[206] R. E. Neapolitan. *Learning Bayesian Networks*. Prentice Hall, 2003.

[207] A. V. Nefian, Luhong L., Xiaobo P., Liu X., C. Mao, and K. Murphy. A coupled HMM for audio-visual speech recognition. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 2, pages 2013–2016, 2002.

[208] D. Nilsson. An efficient algorithm for finding the m most probable configurations in a probabilistic expert system. *Statistics and Computing*, 8:159–173, 1998.

[209] D. Nilsson and J. Goldberger. Sequentially finnding the N-best list in Hidden Markov Models. *Internation Joint Conference on Artificial Intelligence (IJCAI)*, 17, 2001.

[210] A. B. Novikoff. On convergence proofs on perceptrons. In *Symposium on the Mathematical Theory of Automata (New York, 1962)*, volume 12, pages 615–622, Brooklyn, N.Y., 1963. Polytechnic Press of Polytechnic Institute of Brooklyn.

[211] F. J. Och and H. Ney. Discriminative training and maximum entropy models for statistical machine translation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 295–302, Philadelphia, July 2002.

[212] B. A. Olshausen and D. J. Field. Sparse coding with an overcomplete basis set: A strategy employed by V1? *Vision Research*, 37:3311–3325, 1998.

[213] A. V. Oppenheim, R. W. Shafer, M. T. Yoder, and W. T. Padgett. *Discrete-Time Signal Processing*. Signal Processing. Prentice Hall, third edition, 2009.

[214] P. Paatero and U. Tapper. Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. *Environmetrics*, 5:111–126, 1994.

[215] V. Pavlovic, J. M. Rehg, and J. MacCormick. Learning switching linear models of human motion. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems (NIPS)*, number 13, pages 981–987, Cambridge, MA, 2001. MIT Press.

[216] J. Pearl. *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[217] J. Pearl. *Causality: Models, Reasoning and Inference*. Cambridge University Press, 2000.

[218] B. A. Pearlmutter and L. C. Parra. Maximum Likelihood Blind Source Separation: A Context-Sensitive Generalization of ICA. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems (NIPS)*, number 9, pages 613–619, Cambridge, MA, 1997. MIT Press.

[219] K. B. Petersen and O. Winther. The EM algorithm in independent component analysis. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 5, pages 169–172, 2005.

[220] J-P. Pfister, T. Toyiozumi, D. Barber, and W. Gerstner. Optimal Spike-Timing Dependent Plasticity for Precise Action Potential Firing in Supervised learning. *Neural Computation*, 18:1309–1339, 2006.

[221] J. Platt. Fast Training of Support Vector Machines Using Sequential Minimal Optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, pages 185–208. MIT Press, 1999.

[222] I. Porteous, D. Newman, A. Ihler, A. Asuncion, P. Smyth, and M. Welling. Fast collapsed Gibbs sampling for Latent Dirichlet Allocation. In *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 569–577, New York, NY, USA, 2008. ACM.

[223] J. E. Potter and R. G. Stern. Statistical filtering of space navigation measurements. In *American Institute of Aeronautics and Astronautics Guidance and Control Conference*, volume 13, pages 775–801, Cambridge, Mass., August 1963.

[224] S. J. D. Prince, Y. Fu, U. Mohammed, and J. Elder. Probabilistic models for inference about identity. *IEEE transactions in Pattern Recognition and Machine Intelligence*, 2009. In press.

[225] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. of the IEEE*, 77(2):257–286, 1989.

[226] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning.* MIT Press, 2006.

[227] H. E. Rauch, G. Tung, and C. T. Striebel. Maximum Likelihood estimates of linear dynamic systems. *American Institute of Aeronautics and Astronautics Journal (AIAAJ)*, 3(8):1445–1450, 1965.

[228] T. Richardson and P. Spirtes. Ancestral Graph Markov Models. *Annals of Statistics*, 30(4):962–1030, 2002.

[229] D. Rose, R. E. Tarjan, and E. S. Lueker. Algorithmic aspects of vertex elimination of graphs. *SIAM Journal on Computing*, (5):266–283, 1976.

[230] F. Rosenblatt. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, 65(6):386–408, 1958.

[231] D. B. Rubin. Using the SIR algorithm to simulate posterior distributions. In M. H. Bernardo, K. M. Degroot, D. V. Lindley, and A. F. M. Smith, editors, *Bayesian Statistics 3.* Oxford University Press, 1988.

[232] D. Saad and M. Opper. *Advanced Mean Field Methods Theory and Practice.* MIT Press, 2001.

[233] R. Salakhutdinov, S. Roweis, and Z. Ghahramani. Optimization with EM and Expectation-Conjugate-Gradient. In T. Fawcett and N. Mishra, editors, *International Conference on Machine Learning*, number 20, pages 672–679, Menlo Park, CA, 2003. AAAI Press.

[234] A. Salmerón, A. Cano, and S. Moral. Importance Sampling in Bayesian Networks using Probability Trees. *Computational Statistics and Data Analysis*, (34):387–413, 2000.

[235] L. K. Saul, T. S. Jaakkola, and M. J. Jordan. Mean field theory for sigmoid belief networks. *Journal of Artificial Intelligence Research*, 4:61–76, 1996.

[236] L. K. Saul and M. I. Jordan. Exploiting tractable substructures in intractable networks. In D. S. Touretzky, M. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems (NIPS)*, number 8, pages 486–492, Cambridge, MA, 1996. MIT Press.

[237] L. Savage. *The Foundations of Statistics.* Wiley, 1954.

[238] R. D. Schachter. Bayes-ball: The rational pastime (for determining irrelevance and requisite information in belief networks and influence diagrams). In G. Cooper and S. Moral, editors, *Uncertainty in Artificial Intelligence*, number 14, pages 480–487, San Francisco, CA, 1998. Morgan Kaufmann.

[239] B. Schölkopf, A. Smola, and K. R. Müller. Nonlinear Component Analysis as a Kernel Eigenvalue Problem. *Neural Computation*, 10:1299–1319, 1998.

[240] N. N. Schraudolph and D. Kamenetsky. Efficient Exact Inference in Planar Ising Models. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems (NIPS)*, number 21, pages 1417–1424, Cambridge, MA, 2009. MIT Press.

[241] E. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 6(2):461–464, 1978.

[242] M. Seeger. Gaussian Processes for Machine Learning. *International Journal of Neural Systems*, 14(2):69–106, 2004.

[243] M. Seeger. Expectation propagation for exponential families. Technical report, Department of EECS, Berkeley, 2005. www.kyb.tuebingen.mpg.de/bs/people/seeger.

[244] M. Seeger and H. Nickisch. Large scale variational inference and experimental design for sparse generalized linear models. Technical Report 175, Max Planck Institute for Biological Cybernetics, September 2008.

[245] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis.* Cambridge University Press, 2004.

[246] S. Siddiqi, B. Boots, and G. Gordon. A Constraint Generation Approach to Learning Stable Linear Dynamical Systems. In J. C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems (NIPS)*, number 20, pages 1329–1336, Cambridge, MA, 2008. MIT Press.

[247] T. Silander, P. Kontkanen, and P. Myllymäki. On Sensitivity of the MAP Bayesian Network Structure to the Equivalent Sample Size Parameter. In R. Parr and L. van der Gaag, editors, *Uncertainty in Artificial Intelligence*, number 23, pages 360–367, Corvallis, Oregon, USA, 2007. AUAI press.

[248] S. S. Skiena. *The algorithm design manual.* Springer-Verlag, New York, USA, 1998.

[249] E. Smith and M. S. Lewicki. Efficient auditory coding. *Nature*, 439(7079):978–982, 2006.

[250] P. Smolensky. *Parallel Distributed Processing: Volume 1: Foundations*, chapter Information processing in dynamical systems: Foundations of harmony theory, pages 194–281. MIT Press, Cambridge, MA, 1986.

[251] G. Sneddon. *Studies in the atmospheric sciences*, chapter A Statistical Perspective on Data Assimilation in Numerical Models. Number 144 in Lecture Notes in Statistics. Springer-Verlag, 2000.

[252] P. Sollich. Bayesian Methods for Support Vector Machines: Evidence and Predictive Class Probabilities. *Machine Learning*, 46(1-3):21–52, 2002.

[253] D. X. Song, D. Wagner, and X. Tian. Timing Analysis of Keystrokes and Timing Attacks on SSH. In *Proceedings of the 10th conference on USENIX Security Symposium.* USENIX Association, 2001.

[254] A. S. Spanias. Speech coding: a tutorial review. *Proceedings of the IEEE*, 82(10):1541–1582, Oct 1994.

[255] D. J. Spiegelhalter, A. P. Dawid, S. L. Lauritzen, and R. G. Cowell. Bayesian analysis in expert systems. *Statistical Science*, 8(3):219–247, 1993.

[256] P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction, and Search.* MIT press, 2 edition, 2000.

[257] N. Srebro. Maximum Likelihood Bounded Tree-Width Markov Networks. In J. Breese and D. Koller, editors, *Uncertainty in Artificial Intelligence*, number 17, pages 504–511, San Francisco, CA, 2001. Morgan Kaufmann.

[258] H. Steck. *Constraint-Based Structural Learning in Bayesian Networks using Finite Data Sets.* PhD thesis, Technical University Munich, 2001.

[259] H. Steck. Learning the Bayesian Network Structure: Dirichlet Prior vs Data. In D. A. McAllester and Myllym editors, *Uncertainty in Artificial Intelligence*, number 24, pages 511–518, Corvallis, Oregon, USA, 2008. AUAI press.

[260] H. Steck and T. Jaakkola. On the Dirichlet prior and Bayesian regularization. In S. Becker, S. Thrun, and K. Obermayer, editors, *NIPS*, pages 697–704. MIT Press, 2002.

[261] M. Studený. *On mathematical description of probabilistic conditional independence structures.* PhD thesis, Academy of Sciences of the Czech Republic, 2001.

[262] M. Studený. On non-graphical description of models of conditional independence structure. In *HSSS Workshop on Stochastic Systems for Individual Behaviours*. Louvain la Neueve, Belgium, 22-23 January 2001.

[263] C. Sutton and A. McCallum. An introduction to conditional random fields for relational learning. In L. Getoor and B. Taskar, editors, *Introduction to Statistical Relational Learning.* MIT press, 2006.

[264] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, 1998.

[265] R. J. Swendsen and J-S. Wang. Nonuniversal critical dynamics in Monte Carlo simulations. *Physical Review Letters*, 58:86–88, 1987.

[266] B. K. Sy. A Recurrence Local Computation Approach Towards Ordering Composite Beliefs in Bayesian Belief Networks. *International Journal of Approximate Reasoning*, 8:17–50, 1993.

[267] T. Sejnowski. The Book of Hebb. *Neuron*, 24:773–776, 1999.

[268] R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13(3):566–579, 1984.

[269] S. J. Taylor. *Modelling Financial Time Series.* World Scientific, second edition, 2008.

[270] Y. W. Teh, D. Newman, and M. Welling. A Collapsed Variational Bayesian Inference Algorithm for Latent Dirichlet Allocation. In J. C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems (NIPS)*, number 20, pages 1481–1488, Cambridge, MA, 2008. MIT Press.

[271] Y. W. Teh and M. Welling. The unified propagation and scaling algorithm. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems (NIPS)*, number 14, pages 953–960, Cambridge, MA, 2002. MIT Press.

[272] M. Tipping and C. M. Bishop. Mixtures of probabilistic principal component analysers. *Neural Computation*, 11(2):443–482, 1999.

[273] M. E. Tipping. Sparse Bayesian learning and the relevance vector machine. *Journal of Machine Learning Research*, 1:211–244, 2001.

[274] D. M. Titterington, A. F. M. Smith, and U. E. Makov. *Statistical analysis of finite mixture distributions.* Wiley, 1985.

[275] E. Todorov. Efficient computation of optimal actions. *Proceedings of the National Academy of Sciences of the United States of America (PNAS)*, 106(28):11478–11483, 2009.

[276] M. Toussaint, S. Harmeling, and A. Storkey. Probabilistic inference for solving (PO)MDPs. Research Report EDI-INF-RR-0934, University of Edinburgh, School of Informatics, 2006.

[277] M. Tsodyks, K. Pawelzik, and H. Markram. Neural Networks with Dynamic Synapses. *Neural Computation*, 10:821–835, 1998.

[278] P. Van Overschee and B. De Moor. *Subspace Identification for Linear Systems; Theory, Implementations, Applications.* Kluwer, 1996.

[279] V. Vapnik. *The Nature of Statistical Learning Theory.* Springer, New York, 1995.

[280] M. Verhaegen and P. Van Dooren. Numerical Aspects of Different Kalman Filter Implementations. *IEEE Transactions of Automatic Control*, 31(10):907–917, 1986.

[281] T. Verma and J. Pearl. Causal networks : Semantics and expressiveness. In R. D. Schacter, T. S. Levitt, L. N. Kanal, and J.F. Lemmer, editors, *Uncertainty in Artificial Intelligence*, volume 4, pages 69–76, Amsterdam, 1990. North-Holland.

[282] T. O. Virtanen, A. T. Cemgil, and S. J. Godsill. Bayesian extensions to nonnegative matrix factorisation for audio signal modelling. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 1825–1828, 2008.

[283] G. Wahba. *Support Vector Machines, Repreducing Kernel Hilbert Spaces, and Randomized GACV*, pages 69–88. MIT Press, 1999.

[284] M. J. Wainwright and M. I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1-2):1–305, 2008.

[285] H. Wallach. Efficient training of conditional random fields. Master's thesis, Division of Informatics, University of Edinburgh, 2002.

[286] Y. Wang, J. Hodges, and B. Tang. Classification of Web Documents Using a Naive Bayes Method. *15th IEEE International Conference on Tools with Artificial Intelligence*, pages 560–564, 2003.

[287] S. Waterhouse, D. Mackay, and T. Robinson. Bayesian methods for mixtures of experts. In D. S. Touretzky, M. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems (NIPS)*, number 8, pages 351–357, Cambridge, MA, 1996. MIT Press.

[288] Y. Weiss and W. T. Freeman. Correctness of Belief Propagation in Gaussian Graphical Models of Arbitrary Topology. *Neural Computation*, 13(10):2173–2200, 2001.

[289] M. Welling, T. P. Minka, and Y. W. Teh. Structured Region Graphs: Morphing EP into GBP. In F. Bacchus and T. Jaakkola, editors, *Uncertainty in Artificial Intelligence*, number 21, pages 609–614, Corvallis, Oregon, USA, 2005. AUAI press.

[290] J. Whittaker. *Graphical Models in Applied Multivariate Statistics.* John Wiley & Sons, 1990.

[291] W. Wiegerinck. Variational approximations between mean field theory and the Junction Tree algorithm. In C. Boutilier and M. Goldszmidt, editors, *Uncertainty in Artificial Intelligence*, number 16, pages 626–633, San Francisco, CA, 2000. Morgan Kaufmann.

[292] W. Wiegerinck and T. Heskes. Fractional Belief Propagation. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems (NIPS)*, number 15, pages 438–445, Cambridge, MA, 2003. MIT Press.

[293] C. K. I. Williams. Computing with infinite networks. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems NIPS 9*, pages 295–301, Cambridge, MA, 1997. MIT Press.

[294] C. K. I. Williams and D. Barber. Bayesian classification with Gaussian processes. *IEEE Trans Pattern Analysis and Machine Intelligence*, 20:1342–1351, 1998.

[295] C. Yanover and Y. Weiss. Finding the M Most Probable Configurations Using Loopy Belief Propagation. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems (NIPS)*, number 16, pages 1457–1464, Cambridge, MA, 2004. MIT Press.

[296] S. Young, D. Kershaw, J. Odell, D. Ollason, V. Valtchev, and P. Woodland. *The HTK Book Version 3.0*. Cambridge University Press, 2000.

[297] A. L. Yuille and A. Rangarajan. The concave-convex procedure. *Neural Computation*, 15(4):915–936, 2003.

[298] J.-H. Zhao, P. L. H. Yu, and Q. Jiang. ML estimation for factor analysis: EM or non-EM? *Statistics and Computing*, 18(2):109–123, 2008.

[299] O. Zoeter. *Monitoring non-linear and switching dynamical systems*. PhD thesis, Radboud University Nijmegen, 2005.

Index