Implicit Representation Networks*

David Barber
Department of Computer Science
University College London

June 25, 2014

Abstract

We discuss a form of Neural Network in which the inputs to the network are also learned as part of the parameter optimisation process. We focus here on the case of unsupervised learning, showing that the reconstruction error for such models is low. The results suggest that the mirrored structure used in standard Neural Network autoencoders is not ideal and that autoencoders should rather focus complexity on the encoding and not the decoding part of the network. We also introduce a novel training algorithm that can be used to train both standard Neural Networks and Implicit Representation networks based on layer insertion.

1 Introduction

Neural Networks and their 'deep' versions have recently re-emerged as popular representations of functions, [3]. Part of this popularity is due to improvement in the state-of-the-art in several areas, driven partly by novel training algorithms and increased computational resources. We discuss an extension of a form of non-linear Principal Components Analysis (PCA) which is interesting to contrast to standard neural networks. This method, which we call Implicit Representation Networks (IRN) has some interesting properties and sheds some light also on the structure of more classical autoencoders. Such networks can also be used for classification and we prefer therefore to view them as more than generalisations of non-linear PCA.

1.1 Neural Networks

It is useful to first consider the standard Neural Network (NN) framework for function approximation. In a feedforward NN a vector input $\mathbf{x}$ is mapped to a vector output through intermediate layers. For a network with L layers, we write the vector function that the network computes as

$$f(\mathbf{x}|\mathcal{W}) \equiv \sigma_L\left(\mathbf{W}_L\mathbf{h}_{L-1} + \mathbf{b}_L\right) \tag{1}$$

where the 'hidden layer' values are given by

$$\mathbf{h}_l = \sigma_l\left(\mathbf{W}_l\mathbf{h}_{l-1} + \mathbf{b}_l\right), \quad l = 2, \ldots, L-1, \quad \mathbf{h}_1 = \sigma_1\left(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1\right) \tag{2}$$

The weight matrices $\mathbf{W}_1, \ldots, \mathbf{W}_L$ and biases $\mathbf{b}_1, \ldots, \mathbf{b}_L$ form the collection of adjustable parameters $\mathcal{W}$. The dimension of each hidden layer is given by $H_l \equiv \dim(\mathbf{h}_l)$. This can be depicted using a directed acyclic graph (DAG) as in fig(1a) which shows a NN with a single hidden layer. More compactly, we can write $\mathbf{x} \to H_1, H_2, \ldots, H_{L-1} \to \mathbf{y}$. The transfer functions $\sigma_l(x)$ can be chosen to be different for each layer (or even differently for units within the same layer). For notational convenience, we will drop the dependence on the biases in the following.

There is a variety of popular non-linear transfer functions used in the literature. For the most part, unless stated otherwise, we will make use of the anti-symmetric shifted sigmoid transfer function[1].

$$\sigma(x) \equiv \frac{e^x}{1+e^x} - \frac{1}{2} \tag{3}$$

*UCL Department of Computer Science Technical Report. Draft version 0.1.

[1]This function has the useful property that, for small x, $\sigma(x) \approx x/4$, meaning that identifying when the transfer function behaves roughly as a linear function is straightforward. Additionally it has the property that $\sigma(0) = 0$, with $\sigma(\infty) = \frac{1}{2}$.

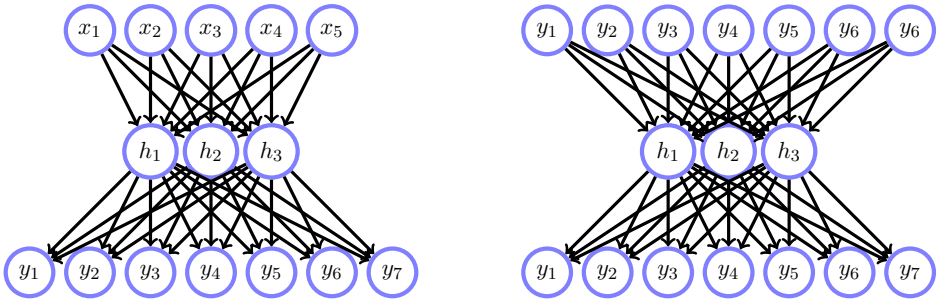


Figure 1: **(a)**: Generic input-output neural network with a single hidden layer. **(b)**: Autoencoder with a single hidden layer

NNs can be use to model mappings from a vector input to (in general) a vector output. The loss function can be suitably chosen (for example squared loss in the case of regression and negative log-likelihood in the case of classification). For example, NNs can be used to solve a regression problem by minimising the squared loss on a set of input-output example pairs, $(\mathbf{x}^n, \mathbf{y}^n)$, $n = 1, \ldots, N$:

$$E(\mathcal{W}) = \sum_{n=1}^{N} [\mathbf{y}^n - \mathbf{f}(\mathbf{x}^\mathbf{n}|\mathcal{W})]^2 \tag{4}$$

However, a well document difficulty with 'deep' networks (networks with L larger than say 10) is to find a good initialisation of the weights $\mathcal{W}$. For this reason, 'layerwise' training algorithms and unsupervised-initialisation approaches have been popular [3].

1.2 Autoencoder

A particularly interesting aspect of NNs is their use to discover low-dimensional representations in high-dimensional data. One way to achieve this is to use an autoencoder. Given a dataset of D-dimensional vectors, $\mathbf{y}^1, \ldots, \mathbf{y}^N$ for a squared loss autoencoder the objective is to mimimise

$$E(\mathcal{W}) = \sum_{n=1}^{N} [\mathbf{y}^n - \mathbf{f}(\mathbf{y}^\mathbf{n}|\mathcal{W})]^2 \tag{5}$$

wrt $\mathcal{W}$. The graph for a single hidden layer is shown in fig(1b). The 'bottleneck' vector $\mathbf{h}_B^n$ (where B is the layer with the smallest dimension) for datavector $\mathbf{y}^n$ corresponds to the low-dimensional representation of $\mathbf{y}^n$. These representations can be used for compression and visualisation or subsequent processing such as classification.

1.2.1 PCA and autoencoders

It is useful to recall the relation between autoencoders and Principal Components Analysis (PCA). PCA can be viewed within the autoencoder framework by using a single hidden layer with linear output function $\sigma_2(x) = x$. In this case it is well known that the optimal squared loss reconstruction error is given by the PCA solution [1]. To show this we first stack all the training data into a single $D \times N$ matrix $\mathbf{Y} = \left[\mathbf{y}^1, \ldots, \mathbf{y}^N\right]$ and consider a NN approximation with squared loss

$$E = ||\mathbf{Y} - \tilde{\mathbf{Y}}||^2$$

where for our single layer network with linear output our NN function is given by $\tilde{\mathbf{Y}} = \mathbf{W}_2\sigma_1(\mathbf{W}_1\mathbf{Y})$. The rank of this NN approximation is at most H since this is the rank of $\mathbf{W}_2$. In the completely unconstrained case the optimal rank H choice for $\tilde{\mathbf{Y}}$ is given by the singular value decomposition of $\mathbf{Y} = \mathbf{USV}^\mathsf{T}$, and taking the largest H singular values:

$$\tilde{\mathbf{Y}} = \mathbf{U}_H\mathbf{S}_H\mathbf{V}_H^\mathsf{T}$$

If we can therefore find a NN such that $\mathbf{U}_H\mathbf{S}_H\mathbf{V}_H^\mathsf{T} = \mathbf{W}_2\sigma_1(\mathbf{W}_1\mathbf{Y})$, we also have the fully optimal NN solution. This can be achieved by setting

$$\mathbf{W}_2 = \mathbf{U}_H, \qquad \sigma_1(\mathbf{W}_1\mathbf{Y}) = \mathbf{S}_H\mathbf{V}_H^\mathsf{T}$$

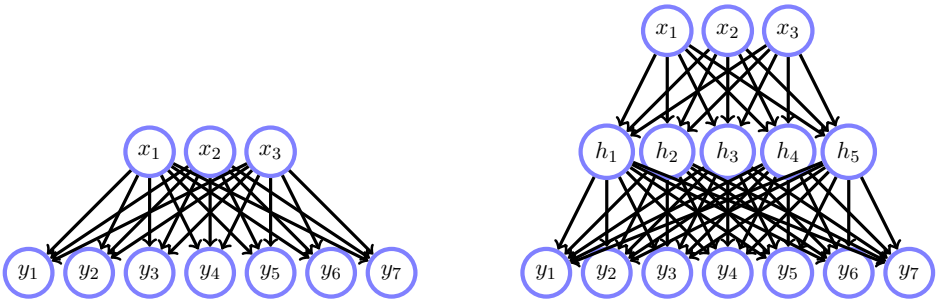


Figure 2: **(a)**: Generic input-output neural network with a single hidden layer. **(b)**: NN with a single hidden layer.

For this to happen we need then

$$\mathbf{S}_H \mathbf{V}_H^\mathsf{T} = \sigma_1(\mathbf{W}_1 \mathbf{U} \mathbf{S} \mathbf{V}^\mathsf{T})$$

If we set $\sigma_1(x) = x$, $\mathbf{W}_1 = \mathbf{U}_H^\mathsf{T}$, then $\mathbf{W}_1\mathbf{U} = (\mathbf{I}_H|\mathbf{0})$ and the result follows.

The conclusion is that, for a linear output $\sigma_2(x) = x$, the optimal transfer function on the single-hidden layer autoencoder for minimal squared reconstruction loss is given by $\sigma_1(x) = x$, with the weights set by the SVD (PCA) of $\mathbf{Y}$. For a non-linear transfer function $\sigma_1(x)$, provided that it has a region of input x for which the output is approximately linear $\sigma_1(x) \approx \lambda x$ (for some fixed λ) then we can also accurately approximate (but not improve upon) the PCA reconstruction error. Thus if we are to improve on the PCA low-dimensional representation, the conclusion is that we must use a non-linear output function σ_L. See [5] for a recent review on PCA and related NN extensions.

An equivalent view of PCA is that it minimises

$$E(\mathbf{W}, \mathbf{W}) = ||\mathbf{Y} - \mathbf{W}\mathbf{X}||^2$$

where the $|Y| \times H$ dimensional matrix $\mathbf{W}$ spans the subspace of the approximation and the columns of the $H \times N$ dimensional matrix $\mathbf{x}$ are the components (low dimensional representations) of the data. PCA can therefore equivalently be viewed either as a linear autoassociater $\mathbf{y} \to H \to \mathbf{y}$ (in which case the weights $\mathbf{W}_1$ and $\mathbf{W}_2$ are learned from the data) or as a linear heteroassociater $\mathbf{x} \to \mathbf{y}$ (in which both the weights $\mathbf{W}$ and representations $\mathbf{X}$ are learned). Note, however, that no such equivalence between the auto $\mathbf{y} \to H_1, \ldots, H_{L-1} \to \mathbf{y}$ and hetero $\mathbf{x} \to H_1, \ldots, H_{L-1} \to \mathbf{y}$ networks exists in general.

2 Implicit Representation Networks

Motivated by the equivalence of PCA and autoencoders, an alternative to an autoencoder is to use a network in which the $\mathbf{x}^n$ are also parameters to be learned, see for example [7, 6]. In the case of desiring a low-dimensional representation $\mathbf{x}$ of an high-dimensional vector $\mathbf{y}$, for squared loss, we may minimize

$$E(\mathcal{W}, \mathcal{X}) = \sum_{n=1}^{N} \left[\mathbf{y}^n - \mathbf{f}(\mathbf{x}^n|\mathcal{W})\right]^2 \tag{6}$$

with respect to both $\mathcal{W}$ and $\mathcal{X} = \left\{\mathbf{x}^1, \ldots, \mathbf{x}^N\right\}$. The parameters $\mathbf{x}^n$ then form the set of low dimensional representations of the high dimensional datapoint $\mathbf{y}^n$. For a novel datapoint $\mathbf{y}^*$, we can find the best representation $\mathbf{x}^*$ through:

$$\mathbf{x}^* = \underset{\mathbf{x}}{\text{argmin}} \ \left[\mathbf{y}^* - \mathbf{f}(\mathbf{x}|\mathcal{W})\right]^2 \tag{7}$$

where $\mathcal{W}$ is that which gives rise to the minimal squared loss on the training data.

As we discussed in section(1.2.1), for the case of linear transfer functions, this is equivalent to training an autoencoder, though in the more general case of non-linear transfer functions, the methods will in general be different. Whilst similar to the 'reconstruction' part of an autoencoder, the IRN model is strictly more powerful since the representation $\mathbf{x}$ is freeform and not constrained to be some parametric function

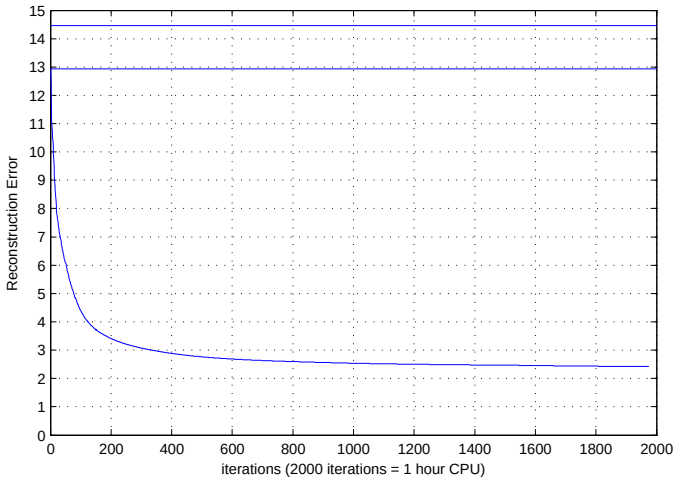


Figure 3: Training error evolution with each iteration of the LBFGS procedure. It takes around 10 minutes to achieve a reconstruction error of 3, equivalent to the deep autoencoder results in [3] (which takes around 10 hours to train). For comparison, PCA has an error of 14.46. 'Inverse PCA' has an error of 12.93 and was used to initialise the 1 layer IRN. Inserting an additional layer with 100 hidden units decreases the reconstruction error to 2.38 after another 1000 iterations (not shown).

of the datapoint $\mathbf{y}$. One might view the IRN as similar to an autoencoder $\mathbf{y} \to \mathbf{x} \to H_1, \ldots, H_L \to \mathbf{y}$ in which the mapping from $\mathbf{y}$ to the latent representation $\mathbf{x}$ is given by the optimisation

$$\mathbf{x} = \underset{\mathbf{x}}{\text{argmin}} \ [\mathbf{y} - \mathbf{f}(\mathbf{x}|\mathcal{W})]^2 \tag{8}$$

This is a generalisation of the non-linear PCA model (fig(2a)) see for example [7, 6] since here we allow $f(\mathbf{x}|\mathcal{W})$ to have hidden layers (fig(2b)). More generally, we define a IRN model through an error

$$E(\mathcal{W}, \mathcal{X}) = \sum_{n=1}^{N} L\left(\mathbf{y}^n, \mathbf{f}(\mathbf{x}^n|\mathcal{W})\right) \tag{9}$$

for some loss function $L(x, y)$. We then find

$$(\mathcal{W}^*, \mathcal{X}^*) = \underset{\mathcal{W},\mathcal{X}}{\text{argmin}} \ E(\mathcal{W}, \mathcal{X}) \tag{10}$$

The representation for a novel data vector $\mathbf{y}^*$ is then given by

$$\mathbf{x}^* = \underset{\mathbf{x}}{\text{argmin}} \ L\left(\mathbf{y}^*, \mathbf{f}(\mathbf{x}|\mathcal{W}^*)\right) \tag{11}$$

2.1 MNIST demonstration

It is interesting to see how the IRN model performs in finding low-dimensional representations for the classical MNIST data problem[2]. The MNIST dataset consists of handwritten digits ($28 \times 28 = 784$ pixel images) with 60,000 training images and 10,000 test images. Each image also has an associated class label (0 to 9) that can be used for testing classification performance as well. Each component of each training image is originally a greyscale value between 0 and 255 which we linearly scaled to a value between -0.5 and 0.5. We define the reconstruction error [3] by

$$R \equiv \frac{1}{N} \sum_{n,i} (y_i^n - \tilde{y}_i^n)^2$$

Using an autoencoder with layer structure[3] $\mathbf{y}(784) \to 1000, 500, 250, 30, 250, 500, 1000 \to \mathbf{y}$, [3] obtained a reconstruction error of approximately 3, which is obtained after around 10 hours on a core i7 processor. The reconstruction error obtained by this deep autoencoder, with 30 dimensional bottleneck representations, is significantly lower than that from PCA (error of 14.46) and alternative methods [3]. The

[2] `http://yann.lecun.com/exdb/mnist`

[3] The network had a linear transfer function in the bottleneck layer.

(a) (b)

Figure 4: **(a)**: Reconstructions using $H = 30$ components. From the Top: Original image, IRN $\mathbf{x}(30) \to \mathbf{y}$ using no hidden layer (reconstruction error=2.42); IRN using $\mathbf{x}(3) \to 100 \to \mathbf{y}$ (reconstruction error=2.38); PCA using 30 components (reconstruction error=14.46) **(b)**: Top 10 worst IRN (1 layer) reconstructions (sorted by reconstruction error) using 30 components. From the Top: Original image, IRN using 1 layer; IRN using the $30 \to 100 \to 784$ network; PCA.

impressive reconstruction performance of this deep autoencoder partially provided the recent incentive to revisit NNs as low dimensional unsupervised learning models.

We first applied the IRN approach with no-hidden layer $\mathbf{x} \to \mathbf{y}$ (using a 30 dimensional $\mathbf{x}$ and no biases). In fig(3) we show the decrease in the reconstruction error with each update of the optimiser[4]. The method finds 30 dimensional representations that have lower reconstruction error than the autoencoder [3] in only a few minutes of computation. The reconstructions for 10 randomly selected representatives of each image class are shown in fig(4a). The reconstructions are so close to the original data that it is difficult to see the difference between the IRN reconstructions and the original data. In fig(4b) we show the 10 worst reconstructions according to the IRN model with no hidden layer. We also trained a $\mathbf{x}(30) \to 100 \to \mathbf{y}$ network on the 60,000 training examples[5], which has a training set reconstruction error of 2.38. A $\mathbf{x}(30) \to 50, 100 \to \mathbf{y}$ network has error around 2.29.

The test set consists of an additional 10,000 images. Fixing the weights $\mathcal{W}$ based on the training data, and then computing the optimal representations $\mathcal{X}^*$ by minimising the reconstruction error on the test images, the no-hidden layer $\mathbf{x}(30) \to \mathbf{y}$ IRN model has reconstruction error 2.51; the single hidden layer $\mathbf{x}(30) \to 100 \to \mathbf{y}$ IRN model has reconstruction error 2.78. In both cases, therefore, the accuracy of the reconstruction for novel test data is not appreciably worse than for the training data.

Without additional constraints such as sparsity, there is no explicit meaning to the values of the components of the vector $\mathbf{x}$ since the only contribution of $\mathbf{x}$ to the network is via $\mathbf{W}_1\mathbf{x}$; we can equivalently write this as $\mathbf{W}_1\mathbf{R}\mathbf{R}^{-1}\mathbf{x}$ for any invertible matrix $\mathbf{R}$ and thus may arbitrarily redefine the weights and the representation $\mathbf{x}$ accordingly. In this sense, there is little meaning to examining the effect of a single unit in the $\mathbf{x}$-layer and we therefore perform ICA [4] to determine the independent directions in this layer. We perform a similar analysis for the other layers in the network, see fig(5). Loosely speaking, the network has learned latent independent directions in the $\mathbf{x}$-space that correspond by and large to digit directions. The difficulty in training this network is shown in fig(6) which demonstrates that despite the increased complexity, the model struggles to find more than a 30 dimensional description of the data in lower layers. It is interesting to note how computationally straightforward it is to find representations with low reconstruction error based on essentially standard non-linear PCA; however, improving on these results by including hidden layers in the network is computationally more demanding. For the multi-layer case we examined a variety of techniques, including greedy layerwise methods and also layer insertion techniques. All these techniques are derived from training a single no-hidden-layer network, as described below.

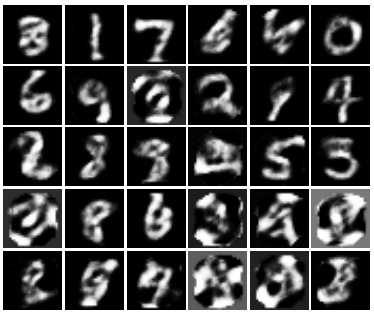
3 Training a no hidden layer IRN model

Here we focus on training a IRN model with no-hidden layer based on squared loss. That is, we wish to find $\mathbf{W}$ and $\mathbf{X}$ that minimises the reconstruction error

$$\sum_n \left(\mathbf{y}^n - \sigma(\mathbf{W}\mathbf{x}^n)\right)^2 = ||\mathbf{Y} - \sigma(\mathbf{W}\mathbf{X})||^2 \tag{12}$$

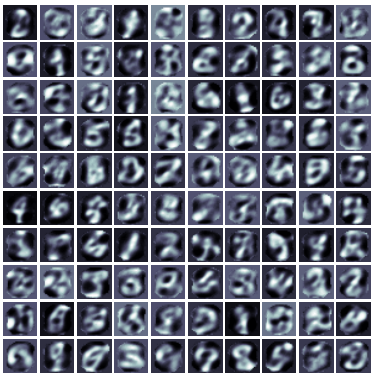
[4]LBFGS in Mark Schmidt's `minfunc.m`. `http://www.di.ens.fr/~mschmidt/Software/minFunc.html`

[5]The training time for this network however is considerably longer, taking around 2 hours.



(a)

(b)



(c)

Figure 5: Representations of the functions of each hidden layer for a $\mathbf{x}(30) \rightarrow 50, 100 \rightarrow \mathbf{y}$ network. ICA is performed on the set of hidden unit vectors $\mathbf{h}^n$ to find new 'independent' hidden unit directions with ICA mixing matrix $\mathbf{A}$. Setting $\mathbf{z}$ to the zero vector, except for $z(i) = 1$, we then form the reconstruction based on activating the hidden layer to value $\mathbf{Az}$ – this forms the representation of each independent direction in the hidden layer. **(a)**: Representation for the 30 dimensional $\mathbf{x}$. **(b)**: The representation of the 50 dimensional hidden layer $\mathbf{h}_1$. **(c)**: The representation of the 50 dimensional hidden layer $\mathbf{h}_2$. It is interesting that at the top layer (a) we see that the network has learned a low dimensional representation that has clear 'digit' directions. Some digits, such as '0' are easier to reconstruct than for example an '8' and this may well reflect that there appears to be only a single '0' direction in $\mathbf{x}$ space, whereas there are perhaps 4 directions for representing the '8' digits. To a partial extent, the layers closer to $\mathbf{x}$ represent 'higher' level features; however this aspect is not particularly distinct and arguably represents a difficulty in the training of this network. .

This is a non-trivial task since the objective is not jointly convex in $\mathbf{W}$ and $\mathbf{X}$.

3.1 Inverse PCA

A useful initialisation can be obtained by considering the bound given in lemma(3.1) below (see section(7.1) for a proof).

Lemma 3.1. *For all* $\mathbf{x}$ *and* $\mathbf{y}$ *with* $-\frac{1}{2} \leq y_i \leq \frac{1}{2}$

$$||\mathbf{y} - \sigma(\mathbf{x})||^2 \leq \frac{1}{16}||\sigma^{-1}(\mathbf{y}) - \mathbf{x}||^2$$

Used in the IRN model, lemma(3.1) gives the bound

$$||\mathbf{Y} - \sigma(\mathbf{WX})||^2 \leq \frac{1}{16}||\sigma^{-1}(\mathbf{Y}) - \mathbf{WX}||^2 \tag{13}$$

Lemma(3.1) suggests a simple way to initialise a single layer IRN. Note first that we can optimise the MVT bound using PCA: we first form $\sigma^{-1}(\mathbf{Y})$ and find the $X = \dim(\mathbf{x})$ dimensional eigen-decomposition (PCA) of this quantity (equivalently, we compute the SVD of $\sigma^{-1}(\mathbf{Y})$); then $\mathbf{W}$ is set to the first X columns of the matrix $\mathbf{U}$ and $\mathbf{X}$ is set to the first X rows of $\mathbf{SV}^\mathsf{T}$. These 'inverse PCA' parameters can then be used to initialise the training of the IRN network using a gradient based method (LBFGS for example). For the shifted logistic sigmoid transfer function, this initialisation approach typically works well. The method holds more generally for any monotonic transfer function in that

$$||\mathbf{Y} - \sigma(\mathbf{WX})||^2 \leq \frac{1}{\left(d\sigma/dx|_0\right)^2}||\sigma^{-1}(\mathbf{Y}) - \mathbf{WX}||^2 \tag{14}$$

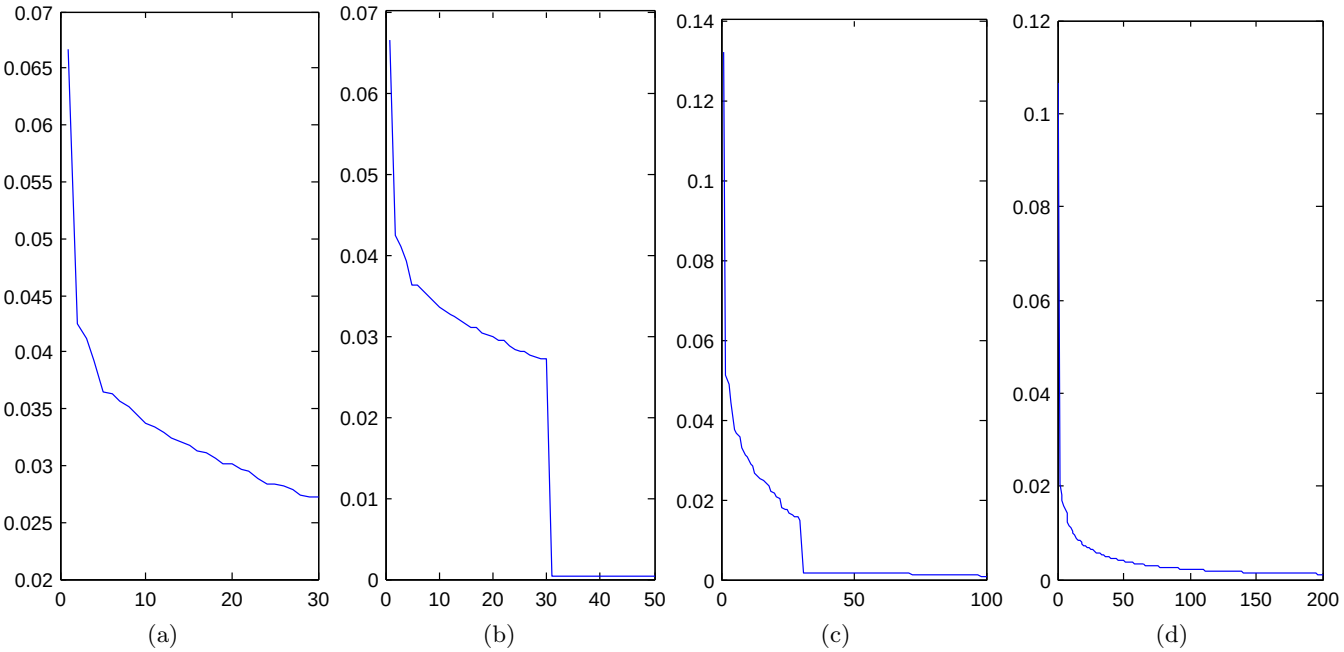


Figure 6: The normalised singular values for the different layers of the $\mathbf{x}(30) \to 50, 100 \to \mathbf{y}$ network. The dominance of the first 30 components in both hidden layers demonstrates a difficulty in training the network. **(a)**: 30 dimensional $\mathbf{x}$ layer. **(b)**: 50 dimensional $\mathbf{h}_1$ layer. **(c)**: 100 dimensional $\mathbf{h}_2$ layer. **(d)**: 784 training data $\mathbf{y}$ layer, truncated at 200 singular values.

3.2 Kullback-Leibler bound

For the case of a transfer function bounded between 0 and 1, another useful bound can be obtained from considering the KL divergence between two Bernoulli distributions on the random variable $z \in \{0, 1\}$.

$$p(z = 1) = \theta, \qquad p(z = 0) = 1 - \theta; \qquad q(z = 1) = \phi, \qquad q(z = 0) = 1 - \phi;$$

where $\theta \in [0, 1]$ and $\phi \in [0, 1]$. Then (see for example Lemma 12.6.1 in [2])

$$\mathrm{KL}(p|q) \geq 2(\theta - \phi)^2 \tag{15}$$

where

$$\mathrm{KL}(p|q) \equiv \theta \ln \theta + (1 - \theta) \ln(1 - \theta) - \theta \ln(\phi) - (1 - \theta) \ln(1 - \phi) \tag{16}$$

Setting $\theta = y_i^n$ and $\phi = \sigma\left(\sum_j W_{ij} x_j^n\right)$, and summing over the data index n, we immediately obtain an upper bound on the squared loss[6]. This bound is convex in $\mathbf{W}$ for fixed $\mathbf{x}^n$ and vice versa. One particularly useful aspect of this bound is in finding the representations for a novel $\mathbf{x}^*$, with $\mathbf{W}$ fixed to its setting from the training data – this bound gives a convex objective to enable one to rapidly find the representation $\mathbf{x}^*$.

4 Multi-Layer training

Given the computational simplicity (and good results) from the simple $\mathbf{x} \to \mathbf{y}$ network, we chose to initialise the multilayer network based on the training from the no-hidden-layer case. In this way we will always be able to find a multilayer network with loss no worse than for the no-hidden-layer case.

There are a variety of closely related approaches, and we describe here the main approach that we used in the experiments. We first define a new transfer function

$$\sigma_\beta(x) = \beta\sigma(x) + (1 - \beta)\frac{x}{4} \tag{17}$$

where $\sigma(x)$ is as defined in equation (3). The motivation is that this provides a smooth interpolation between a linear transfer function and our odd sigmoid function, with the gradient of both functions matching at the origin.

[6]Indeed, in [3] the authors train their model based on entropic loss, rather than squared error loss. However, since the entropic loss upper bounds the squared loss (up to additive constants and a constant scale factor, minimising the entropic loss will typically drive down the squared loss as well.

4.1 Layer Insertion

Given the network

$$f(\mathbf{x}|\mathbf{W}) = \sigma(\mathbf{W}\mathbf{x})$$

we can insert a linear layer such that the new network has the same mapping:

$$f(\mathbf{x}|\mathcal{W}) = \sigma(\mathbf{W}_2\mathbf{W}_1\mathbf{x})$$

provided we can find matrices $\mathbf{W}_1$ and $\mathbf{W}_2$ such that $\mathbf{W}_2\mathbf{W}_1 = \mathbf{W}$. For example, imagine that we have trained a $\mathbf{x}(30) \to 784$ network, with a 784×30 dimensional weight matrix $\mathbf{W}$. We now wish to insert a 50 dimensional layer to form a $\mathbf{x}(30) \to 50 \to 784$ network that produces the same input-output mapping.

That is, we require

$$\mathbf{W} = \mathbf{W_2}\mathbf{W_1} \tag{18}$$

To achieve this we can define the partitioned matrices

$$\mathbf{W}_2 = [\mathbf{W} \;\; \mathbf{0}_{784\times 20}], \qquad \mathbf{W}_1 = \begin{pmatrix} \mathbf{I}_{30\times 30} \\ \mathbf{0}_{20\times 30} \end{pmatrix}$$

The matrix $\mathbf{W}_2$ has dimension 784×50 and $\mathbf{W}_1$ dimension 50×30, as required. However, the zeros in this construction can make it difficult for an optimiser to move away from this parameter setting. For this reason we post multiply $\mathbf{W}_2$ by a scaled random orthogonal matrix $\mathbf{R}$ and pre-multiply $\mathbf{W}_1$ by the transpose of $\mathbf{R}$. With this construction, the new network

$$\sigma\left(\mathbf{W}_2\sigma_\beta\left(\mathbf{W}_1\frac{\mathbf{x}}{4}\right)\right)$$

matches the original no-hidden-layer network when we set $\beta = 0$. To improve on this representation we gradually increase β from its starting value of zero. The non-linearity is then gradually increased during training of the multilayer network[7]. For the case of additional hidden layers, we repeat this process, defining linear transfer functions for the additional layers. It is interesting to note that this linear to non-linear training approach would also be a potentially useful way to train a standard neural network; this requires only a small modification to the scheme.

Whilst this does work, unfortunately, progress away from the initial no-hidden-layer solution can be slow and alternative methods would be beneficial. We have experimented with various greedy layerwise methods, but none has performed as well as robustly as the above approach.

5 Discussion

The simplicity of the non-linear PCA model and its effectiveness at finding good low dimensional representations suggests that the multilayer extension is a potentially useful research direction.

The results also shed some light on the structure of autoencoders. As we have seen, a $\mathbf{x}(30) \to \mathbf{y}$ mapping has excellent reconstruction performance. This can be viewed as the reconstruction part of an autoencoder network; provided therefore that we have obtained a good low-dimensional representation, at least for the MNIST problem, a relatively trivial decoder is all that is required to produce good performance. This suggests that the standard autoencoder model in which the input to bottleneck structure is mirrored from the bottleneck to the output, is not ideal. Rather, there should be a much more complex mapping from the input to the bottleneck and a relatively simple mapping from the bottleneck to the output, resulting in a highly non-symmetric style autoencoder.

We have also experimented with the IRN model for classification. By training a separate IRN model for each digit class, and assigning a novel test digit to that model which best reconstructs it, we obtain a classification accuracy of around 98%. Whilst this is reasonable, there is still some gap to the best available NN methods[8], and more research is required to understand how to bridge this performance gap. We have also considered discriminative classification models of the softmax form

$$p(c|\mathbf{y},\mathcal{W}) = \int_{\mathbf{x}} p(c|\mathbf{x},\mathbf{y},\mathcal{W})p(\mathbf{x}|\mathbf{y}) = \int_{\mathbf{x}} \frac{\exp -L(\mathbf{y}, f(\mathbf{x},\mathcal{W}_c))}{\sum_{c'}\exp -L(\mathbf{y}, f(\mathbf{x},\mathcal{W}_{c'}))} p(\mathbf{x}|\mathbf{y},\mathcal{W})$$

where $\mathcal{W}_c$ are parameters for class c. Similar to the method of fitting a separate unsupervised model to each digit class, these have accuracy around the 98% mark; more research is required to understand why these techniques do not produce more accurate classifiers.

[7] Another approach would be to include β as part of the parameter set to be optimised.

[8] `http://yann.lecun.com/exdb/mnist`

6 Summary

The simplicity and reconstruction accuracy of Implicit Representation Networks is interesting and sheds some light on the structure of autoencoders for unsupervised learning, suggesting that the standard picture of structures that are mirrored around the bottleneck layer is not optimal. Further insights on training deeper structures would be useful, as would potential additional constraints to make classification more accurate and representations more interpretable.

Acknowledgements

I would like to thank Tristan Deleu for deriving the MVT bound and Simona Ullo for useful discussions.

7 Appendix

7.1 Proof of Lemma 3.1

Proof For vector valued function $f(\mathbf{x})$ the Mean Value Theorem states

$$f(\mathbf{y}) - f(\mathbf{x}) = \nabla f^{\mathsf{T}}((1-c)\mathbf{x} + c\mathbf{y})(\mathbf{y} - \mathbf{x})$$

The Schwarz inquality then gives

$$||f(\mathbf{y}) - f(\mathbf{x})|| \leq ||\nabla f^{\mathsf{T}}((1-c)\mathbf{x} + c\mathbf{y})||||\mathbf{y} - \mathbf{x}||$$

For $f(x) \equiv \sigma(x)$, the gradient is bounded, that is $|\sigma'(x)| \leq 1/4$, hence

$$||\sigma(\mathbf{y}) - \sigma(\mathbf{x})|| \leq \frac{1}{4}||\mathbf{y} - \mathbf{x}||$$

The result follows using the substitution $\mathbf{y} \to \sigma^{-1}(\mathbf{y})$. □

References

[1] H. Bourlard and Y. Kamp. Auto-Association by Multilayer Perceptrons and Singular Value Decomposition. *Biological Cybernetics*, 59:291–294, 1988.

[2] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley-Interscience, 2006.

[3] G. E. Hinton and R. R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(28):504–507, July 2006.

[4] A. Hyvärinen, J. Karhunen, and E. Oja. *Independent Component Analysis*. Wiley, 2001.

[5] J. Qiu, H. Wang, J. Lu, B. Zhang, and K. L. Du. Neural Network Implementations for PCA and Its Extensions. *ISRN Artificial Intelligence*, 2012. Article ID 847305.

[6] M. Scholz, M. Fraunholz, and J. Selbig. *Principal Manifolds for Data Visualization and Dimension Reduction*, volume 58 of *LNCSE*, chapter Nonlinear principal component analysis: neural network models and applications, pages 44–67. Springer, 2007.

[7] M. Scholz, F. Kaplan, C. L. Guy, J. Kopka, and J. Selbig. Non-linear PCA: a missing data approach. *Bioinformatics*, 21(20):3887–3895, 2005.