Learning a Nonlinear Embedding by Preserving Class Neighbourhood Structure

Ruslan Salakhutdinov and Geoffrey Hinton
Department of Computer Science
University of Toronto
Toronto, Ontario M5S 3G4

Abstract

We show how to pretrain and fine-tune a multilayer neural network to learn a nonlinear transformation from the input space to a low-dimensional feature space in which K-nearest neighbour classification performs well. We also show how the non-linear transformation can be improved using unlabeled data. Our method achieves a much lower error rate than Support Vector Machines or standard backpropagation on a widely used version of the MNIST handwritten digit recognition task. If some of the dimensions of the low-dimensional feature space are not used for nearest neighbor classification, our method uses these dimensions to explicitly represent transformations of the digits that do not affect their identity.

1 Introduction

Learning a similarity measure or distance metric over the input space X is an important task in machine learning. A good similarity measure can provide insight into how high-dimensional data is organized and it can significantly improve the performance of algorithms like K-nearest neighbours (KNN) that are based on computing distances [4].

For any given distance metric D (*e. g.* Euclidean) we can measure similarity between two input vectors $\mathbf{x}^a, \mathbf{x}^b \in X$ by computing $D[f(\mathbf{x}^a|W), f(\mathbf{x}^b|W)]$, where $f(\mathbf{x}|W)$ is a function $f : X \to Y$ mapping the input vectors in X into a feature space Y and is parameterized by W (see fig. 1). As noted by [8] learning a similarity measure is closely related to the problem of feature extraction, since for any fixed D, any feature extraction algorithm can be thought of as learning a similarity metric. Previous work studied the case when D is Euclidean distance and $f(\mathbf{x}|W)$ is a simple linear projection $f(\mathbf{x}|W) = W\mathbf{x}$. The Euclidean distance in the feature space is then the Mahalanobis distance in the input space:

$$D[f(\mathbf{x}^a), f(\mathbf{x}^b)] = (\mathbf{x}^a - \mathbf{x}^b)^T W^T W (\mathbf{x}^a - \mathbf{x}^b) \qquad (1)$$

Linear discriminant analysis (LDA) learns the matrix W which minimizes the ratio of within-class distances to between-class distances. Goldberger et.al.[9] learned the linear transformation that optimized the performance of KNN in the resulting feature space. This differs from LDA because it allows two members of the same class to be far apart in the feature space so long as each member of the class is close to K other class members. Globerson and Roweis [8] learned the matrix W such that the input vectors from the same class mapped to a tight cluster. They showed that their method approximates the local covariance structure of the data and is therefore not based on Gaussian assumption as opposed to LDA which uses global covariance structure. Weinberger et.al.[18] also learned W with the twin goals of making the K-nearest neighbours belong to the same class and making examples from different classes be separated by a large margin. They succeeded in achieving a test error rate of 1.3% on the MNIST dataset[15].

A linear transformation has a limited number of parameters and it cannot model higher-order correlations between the original data dimensions. In this paper, we show that a non-linear transformation function $f(\mathbf{x}|W)$ with many more parameters can discover low-dimensional representations that work much better than existing linear methods provided the dataset is large enough to allow the parameters to be estimated.

The idea of using a multilayer neural network to learn a nonlinear function $f(\mathbf{x}|W)$ that maximizes agreement between output signals goes back to [2]. They showed that it is possible to learn to extract depth from stereo images of smooth, randomly textured surfaces by maximizing the mutual information between the one-dimensional outputs of two or more neural networks. Each network looks at a local patch of both images and tries to extract a number that has high mutual information with the number extracted by networks looking at nearby stereo patches. The only property that is coherent across space is the depth of the surface so that is what the networks learn to extract. A similar approach has been used to extract structure that is coherent across time [17].

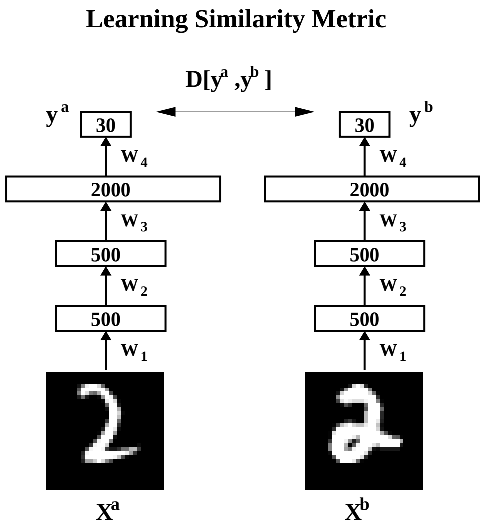


Figure 1: After learning a non-linear transformation from images to 30-dimensional code vectors, the Euclidean distance between code vectors can be used to measure the similarity between images.

Generalizing this idea to networks with multi-dimensional, real-valued outputs is difficult because the true mutual information depends on the entropy of the output vectors and this is hard to estimate efficiently for multi-dimensional outputs. Approximating the entropy by the log determinant of a multidimensional Gaussian works well for learning linear transformations [7], because a linear transformation cannot alter how Gaussian a distribution is. But it does not work well for learning non-linear transformations [21] because the optimization cheats by making the Gaussian approximation to the entropy as bad as possible. The mutual information is the *difference* between the individual entropies and the joint entropy, so it can be made to appear very large by learning individual output distributions that resemble a hairball. When approximated by a Gaussian, a large hairball has a large determinant but its true entropy is very low because the density is concentrated into the hairs rather than filling the space.

The structure in an *iid* set of image pairs can be decomposed into the structure in the whole *iid* set of individual images, ignoring the pairings, plus the additional structure in the way they are paired. If we focus on modeling only the additional structure in the pairings, we can finesse the problem of estimating the entropy of a multi-dimensional distribution. The additional structure can be modeled by finding a non-linear transformation of each image into a low-dimensional code such that paired images have codes that are much more similar than images that are not paired. Adopting a probabilistic approach, we can define a probability distribution over all possible pairs of images, $\mathbf{x}^a, \mathbf{x}^b$ by using the squared distances between their codes, $f(\mathbf{x}^a), f(\mathbf{x}^b)$:

$$p(\mathbf{x}^a, \mathbf{x}^b) = \frac{||f(\mathbf{x}^a) - f(\mathbf{x}^b)||^2}{\sum_{k<l} ||f(\mathbf{x}^k) - f(\mathbf{x}^l)||^2} \qquad (2)$$

We can then learn the non-linear transformation by maximizing the log probability of the pairs that actually occur in the training set. The normalizing term in Eq. 2 is quadratic in the number of training cases rather than exponential in the number of pixels or the number of code dimensions because we are only attempting to model the structure in the pairings, not the structure in the individual images or the mutual information between the code vectors.

The idea of using Eq. 2 to train a multilayer neural network was originally described in [9]. They showed that a network would extract a two-dimensional code that explicitly represented the size and orientation of a face if it was trained on pairs of face images that had the same size and orientation but were otherwise very different. Attempts to extract more elaborate properties were less successful partly because of the difficulty of training multilayer neural networks with many hidden layers, and partly because the amount of information in the pairings of N images is less than $\log N$ bits per pair. This means that a very large number of pairs is required to train a large number of parameters.

Chopra et.al. [3] have recently used a non-probabilistic version of the same approach to learn a similarity metric for faces that assigns high similarity to very different images of the same person and low similarity to quite similar images of different people. They achieve the same effect as Eq. 2 by using a carefully hand-crafted penalty function that uses both positive (similar) and negative (dissimilar) examples. They greatly reduce the number of parameters to be learned by using a convolutional multilayer neural network and achieve impressive results on a face verification task.

We have recently discovered a very effective and entirely unsupervised way of training a multi-layer, non-linear ”encoder” network that transforms the input data vector $\mathbf{x}$ into a low-dimensional feature representation $f(\mathbf{x}|W)$ that captures a lot of the structure in the input data [14]. This unsupervised algorithm can be used as a *pretraining* stage to initialize the parameter vector W that defines the mapping from input vectors to their low-dimensional representation. After the initial pretraining, the parameters can be fine-tuned by performing gradient descent in the Neighbourhood Component Analysis (NCA) objective function introduced by [9]. The learning results in a non-linear transformation of the input space which has been optimized to make KNN perform well in the low-dimensional feature space. Using this nonlinear NCA algorithm to map MNIST digits into the 30-dimensional feature space, we achieve an error rate of 1.08%. Support Vector Machines have a significantly higher error rate of 1.4% on the same version of the MNIST task [5].

In the next section we briefly review Neighborhood Components Analysis and generalize it to its nonlinear counterpart. In section 3, we show how one can efficiently perform pretraining to extract useful features from binary or real-valued data. In section 4 we show that nonlinear NCA significantly outperforms linear methods on the MNIST dataset of handwritten digits. In section 5 we show how nonlinear NCA can be regularized by adding an extra term to the objective function. The extra term is the error in reconstructing the data from the code. Using this regularizing term, we show how nonlinear NCA can benefit from additional, unlabeled data. We further demonstrate the superiority of regularized nonlinear NCA when only small fraction of the images are labeled.

2 Learning Nonlinear NCA

We are given a set of N labeled training cases $(\mathbf{x}^a, c^a)$, $a = 1, 2, ..., N$, where $\mathbf{x}^a \in R^d$, and $c^a \in \{1, 2, ..., C\}$. For each training vector $\mathbf{x}^a$, define the probability that point a selects one of its neighbours b (as in [9, 13]) in the transformed feature space as:

$$p_{ab} = \frac{\exp(-d_{ab})}{\sum_{z \neq a} \exp(-d_{az})}, \qquad p_{aa} = 0 \tag{3}$$

We focus on the Euclidean distance metric:

$$d_{ab} = \| f(\mathbf{x}^a|W) - f(\mathbf{x}^b|W) \|^2$$

and $f(\cdot|W)$ is a multi-layer neural network parametrized by the weight vector W (see fig 1). The probability that point a belongs to class k depends on the relative proximity of all other data points that belong to class k:

$$p(c^a = k) = \sum_{b:c^b=k} p_{ab} \tag{4}$$

The NCA objective (as in [9]) is to maximize the expected number of correctly classified points on the training data:

$$O_{NCA} = \sum_{a=1}^{N} \sum_{b:c^a=c^b} p_{ab} \tag{5}$$

One could alternatively maximize the sum of the log probabilities of correct classification:

$$O_{ML} = \sum_{a=1}^{N} \log \big(\sum_{b:c^a=c^b} p_{ab} \big) \tag{6}$$

When $f(\mathbf{x}|W) = W\mathbf{x}$ is constrained to be a linear transformation, we get linear NCA. When $f(\mathbf{x}|W)$ is defined by a multilayer, non-linear neural network, we can explore a much richer class of transformations by backpropagating the derivatives of the objective functions in Eq. 5 or 6 with respect to parameter vector W through the layers of the encoder network. In our experiments, the NCA objective O_{NCA} of Eq. 5 worked slightly better than O_{ML}. We suspect that this is because O_{NCA} is more robust to handling outliers. O_{ML}, on the other hand, would strongly penalize configurations where a point in the feature space does not lie close to any other member of its class. The derivatives of Eq. 5 are given in the appendix.

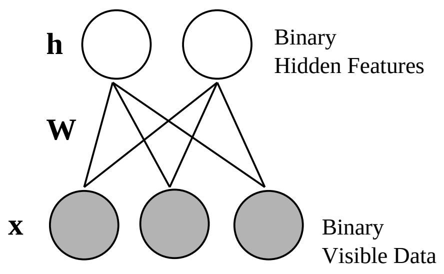


Figure 2: A Restricted Boltzmann Machine. The top layer represents a vector of stochastic binary features $\mathbf{h}$ and and the bottom layer represents a vector of stochastic binary "visible" variables $\mathbf{x}$. When modeling real-valued visible variables, the bottom layer is composed of linear units with Gaussian noise.

3 Pretraining

In this section we describe an unsupervised way to learn an adaptive, multi-layer, non-linear "encoder" network that transforms the input data vector $\mathbf{x}$ into its low-dimensional feature representation $f(\mathbf{x}|W)$. This learning is treated as a *pretraining* stage that discovers good low-dimensional representations. Subsequent fine-tuning of the weight vector W is carried out using the objective function in Eq. 5

3.1 Modeling Binary Data

We model binary data (e.g. the MNIST digits) using a Restricted Boltzmann Machine [6, 16, 11] (see fig. 2). The "visible" stochastic binary input vector $\mathbf{x}$ and "hidden" stochastic binary feature vector $\mathbf{h}$ are modeled by products of conditional Bernoulli distributions:

$$p(h_j = 1|\mathbf{x}) = \sigma(b_j + \sum_i W_{ij} x_i) \tag{7}$$

$$p(x_i = 1|\mathbf{h}) = \sigma(b_i + \sum_j W_{ij} h_j) \tag{8}$$

where $\sigma(z) = 1/(1 + e^{-z})$ is the logistic function, W_{ij} is a symmetric interaction term between input i and feature j, and b_i, b_j are biases. The biases are part of the overall parameter vector, W. The marginal distribution over visible vector $\mathbf{x}$ is:

$$p(\mathbf{x}) = \sum_{\mathbf{h}} \frac{\exp(-E(\mathbf{x}, \mathbf{h}))}{\sum_{\mathbf{u},\mathbf{g}} \exp(-E(\mathbf{u}, \mathbf{g}))} \tag{9}$$

where $E(\mathbf{x}, \mathbf{h})$ is an energy term (i.e. a negative log probability + an unknown constant offset) given by:

$$E(\mathbf{x}, \mathbf{h}) = -\sum_i b_i x_i - \sum_j b_j h_j - \sum_{i,j} x_i h_j W_{ij} \tag{10}$$

The parameter updates required to perform gradient ascent in the log-likelihood can be obtained from Eq. 9:

$$\Delta W_{ij} = \epsilon \frac{\partial \log p(\mathbf{x})}{\partial W_{ij}} = \epsilon(<x_i h_j>_{data} - <x_i h_j>_{model})$$

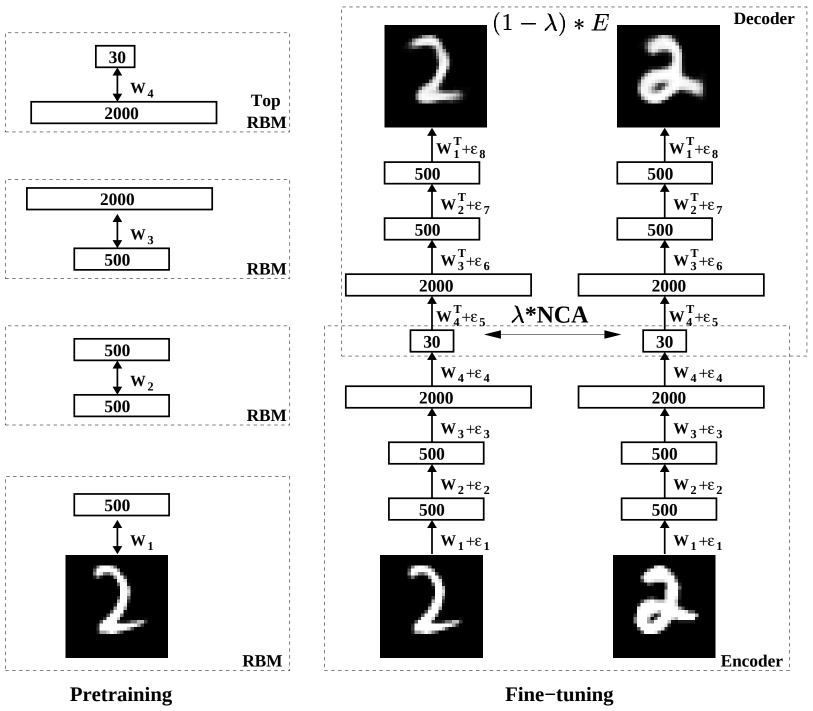


Figure 3: Left panel: Pretraining consists of learning a stack of RBM's in which the feature activations of one RBM are treated as data by the next RBM. Right panel: After pretraining, the RBM's are "unrolled". Setting $\lambda = 1$ results in nonlinear NCA, setting $\lambda = 0$ results in a deep multi-layer autoencoder. For $\lambda \in (0, 1)$, the NCA objective is combined with autoencoder reconstruction error E to create regularized nonlinear NCA. The network is fine-tuned by backpropagation.

where ϵ is the learning rate, $<\cdot>_{data}$ denotes an expectation with respect to the data distribution and $<\cdot>_{model}$ is an expectation with respect to the distribution defined by the model. To circumvent the difficulty of computing $<\cdot>_{model}$, we use 1-step Contrastive Divergence [11]:

$$\Delta W_{ij} = \epsilon(<x_i h_j>_{data} - <x_i h_j>_{recon}) \quad (11)$$

The expectation $< x_i h_j >_{data}$ defines the frequency with which input i and feature j are on together when the features are being driven by the observed data from the training set using Eq. 7. After stochastically activating the features, Eq. 8 is used to "reconstruct" binary data. Then Eq. 7 is used again to activate the features and $< x_i h_j >_{recon}$ is the corresponding frequency when the features are being driven by the reconstructed data. The learning rule for the biases is just a simplified version of Eq. 11.

3.2 Modeling Real-valued Data

Welling *et. al.* [19] introduced a class of two-layer undirected graphical models that generalize Restricted Boltzmann Machines (RBM's) to exponential family distributions. This allows them to model images with real-valued pixels by using visible units that have a Gaussian distribution whose mean is determined by the hidden units:

$$p(x_i = x|\mathbf{h}) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\Big(-\frac{(x - b_i - \sigma_i \sum_j h_j w_{ij})^2}{2\sigma_i^2}\Big) \quad (12)$$

$$p(h_j = 1|\mathbf{x}) = \sigma\big(b_j + \textstyle\sum_i W_{ij} \frac{x_i}{\sigma_i}\big) \quad (13)$$

The marginal distribution over visible units $\mathbf{x}$ is given by Eq. 9. with an energy term:

$$E(\mathbf{x}, \mathbf{h}) = \sum_i \frac{(x_i - b_i)^2}{2\sigma_i^2} - \sum_j b_j h_j - \sum_{i,j} h_j w_{ij} \frac{x_i}{\sigma_i} \quad (14)$$

The gradient of the log-likelihood function is:

$$\frac{\partial \log p(\mathbf{x})}{\partial W_{ij}} = <\frac{x_i}{\sigma_i} h_j>_{data} - <\frac{x_i}{\sigma_i} h_j>_{model}$$

If we set variances $\sigma_i^2 = 1$ for all visible units i, the parameter updates are the same as defined in Eq. 11.

3.3 Greedy Recursive Pretraining

After learning the first layer of hidden features we have an undirected model that defines $p(\mathbf{x}, \mathbf{h})$ via a consistent pair of conditional probabilities, $p(\mathbf{h}|\mathbf{x})$ and $p(\mathbf{x}|\mathbf{h})$. A different way to express what has been learned is $p(\mathbf{x}|\mathbf{h})$ and $p(\mathbf{h})$. Unlike a standard directed model, this $p(\mathbf{h})$ does not have its own separate parameters. It is a complicated, non-factorial prior on $\mathbf{h}$ that is defined implicitly by the weights. This peculiar decomposition into $p(\mathbf{h})$ and $p(\mathbf{x}|\mathbf{h})$ suggests a recursive algorithm: keep the learned $p(\mathbf{x}|\mathbf{h})$ but replace $p(\mathbf{h})$ by a better prior over $\mathbf{h}$.

For any approximating distribution $Q(\mathbf{h}|\mathbf{x})$ we can write:

$$\log p(\mathbf{x}) \geq \sum_{\mathbf{h}} Q(\mathbf{h}|\mathbf{x})[\log p(\mathbf{h}) + \log p(\mathbf{x}|\mathbf{h})] - \sum_{\mathbf{h}} Q(\mathbf{h}|\mathbf{x}) \log Q(\mathbf{h}|\mathbf{x}) \quad (15)$$

If we set $Q(\mathbf{h}|\mathbf{x})$ to be the true posterior distribution, $p(\mathbf{h}|\mathbf{x})$ (Eq. 7), the bound becomes tight. By freezing the parameter vector W at the value W_{frozen} (Eq. 7,8) we freeze $p(\mathbf{x}|\mathbf{h})$, and if we continue to use W^T_{frozen} to compute the distribution over $\mathbf{h}$ given $\mathbf{x}$ we also freeze $Q(\mathbf{h}|\mathbf{x}, W^T_{\text{frozen}})$. When $p(\mathbf{h})$ is implicitly defined by W_{frozen}, $Q(\mathbf{h}|\mathbf{x}, W^T_{\text{frozen}})$ is the true posterior, but when a better distribution is learned for $p(\mathbf{h})$, $Q(\mathbf{h}|\mathbf{x}, W^T_{\text{frozen}})$ is only an approximation to the true posterior. Nevertheless, the loss caused by using an approximate posterior is less than the gain caused by using a better model for $p(\mathbf{h})$, provided this better model is learned by optimizing the variational bound in Eq. 15. Maximizing this bound with W frozen at W_{frozen} is equivalent to maximizing:

$$\sum_{\mathbf{h}} Q(\mathbf{h}|\mathbf{x}, W^T_{\text{frozen}}) \log p(\mathbf{h})$$

This amounts to maximizing the probability of a set of $\mathbf{h}$ vectors that are sampled with probability $Q(\mathbf{h}|\mathbf{x}, W^T_{\text{frozen}})$, *i.e.* it amounts to treating the hidden activity vectors produced by applying W^T_{frozen} to the real data as if they were data for the next stage of learning.[1] Provided the number of features per layer does not decrease, [12] showed that each extra layer increases the variational lower bound in Eq. 15 on the log probability of data. This bound does not apply if the layers get smaller, as they do in an encoder, but, as shown in [14], the pretraining algorithm still works very well as a way to initialize a subsequent stage of fine-tuning. The pretraining finds a point that lies in a good region of parameter space and the myopic fine-tuning then performs a local gradient search that finds a nearby point that is considerably better.

After learning the first layer of features, a second layer is learned by treating the activation probabilities of the existing features, when they are being driven by real data, as the data for the second-level binary RBM (see fig. 3). To suppress noise in the learning signal, we use the real-valued activation *probabilities* for the visible units of every RBM, but to prevent each hidden unit from transmitting more than one bit of information from the data to its reconstruction, the pretraining always uses stochastic binary values for the hidden units.

The hidden units of the top RBM are modeled with stochastic real-valued states sampled from a Gaussian whose mean is determined by the input from that RBM's logistic visible units. This allows the low-dimensional codes to make good use of continuous variables and also facilitates comparisons with linear NCA. The conditional distributions are given in Eq. 12,13, with roles of $\mathbf{h}$ and $\mathbf{x}$ reversed. Throughout all of our experiments we set variances $\sigma_j^2 = 1$ for all hidden units j, which simplifies learning. The parameter updates in this case are the same as defined in Eq. 11.

This greedy, layer-by-layer training can be repeated several times to learn a deep, hierarchical model in which each layer of features captures strong high-order correlations between the activities of features in the layer below.

Recursive Learning of Deep Generative Model:

1. Learn the parameters W^1 of a Bernoulli or Gaussian model.
2. Freeze the parameters of the lower-level model and use the activation probabilities of the binary features, when they are being driven by training data, as the data for training the next layer of binary features.
3. Freeze the parameters W^2 that define the 2^{nd} layer of features and use the activation probabilities of those features as data for training the 3^{rd} layer of features.
4. Proceed recursively for as many layers as desired.

3.4 Details of the training

To speed-up the pretraining, we subdivided the MNIST dataset into small mini-batches, each containing 100 cases, and updated the weights after each mini-batch. Each layer was greedily pretrained for 50 passes (epochs) through the entire training dataset.[2] For fine-tuning model parameters using the NCA objective function we used the method of conjugate gradients[3] on larger mini-batches of 5000 with three line searches performed for each mini-batch in each epoch. To determine an adequate number of epochs and avoid overfitting, we fine-tuned on a fraction of the training data and tested performance on the remaining validation data. We then repeated the fine-tuning on the entire training dataset for 50 epochs.

We also experimented with various values for the learning rate, momentum, and weight-decay parameters used in the pretraining. Our results are fairly robust to variations in these parameters and also to variations in the number of layers and the number of units in each layer. The precise weights found by the pretraining do not matter as long as it finds a good region from which to start the fine-tuning.

4 Experimental Results

In this section we present experimental results for the MNIST handwritten digit dataset. The MNIST dataset [15]

[1]We can initialize the new model of the average conditional posterior over $\mathbf{h}$ by simply using the existing learned model but with the roles of the hidden and visible units reversed. This ensures that our new model starts with exactly the same $p(\mathbf{h})$ as our old one.

[2]The weights were updated using a learning rate of 0.1, momentum of 0.9, and a weight decay of $0.0002 \times \text{weight} \times \text{learning rate}$. The weights were initialized with small random values sampled from a zero-mean normal distribution with variance 0.01.

[3]Code is available at
http://www.kyb.tuebingen.mpg.de/bs/people/carl/code/minimize/

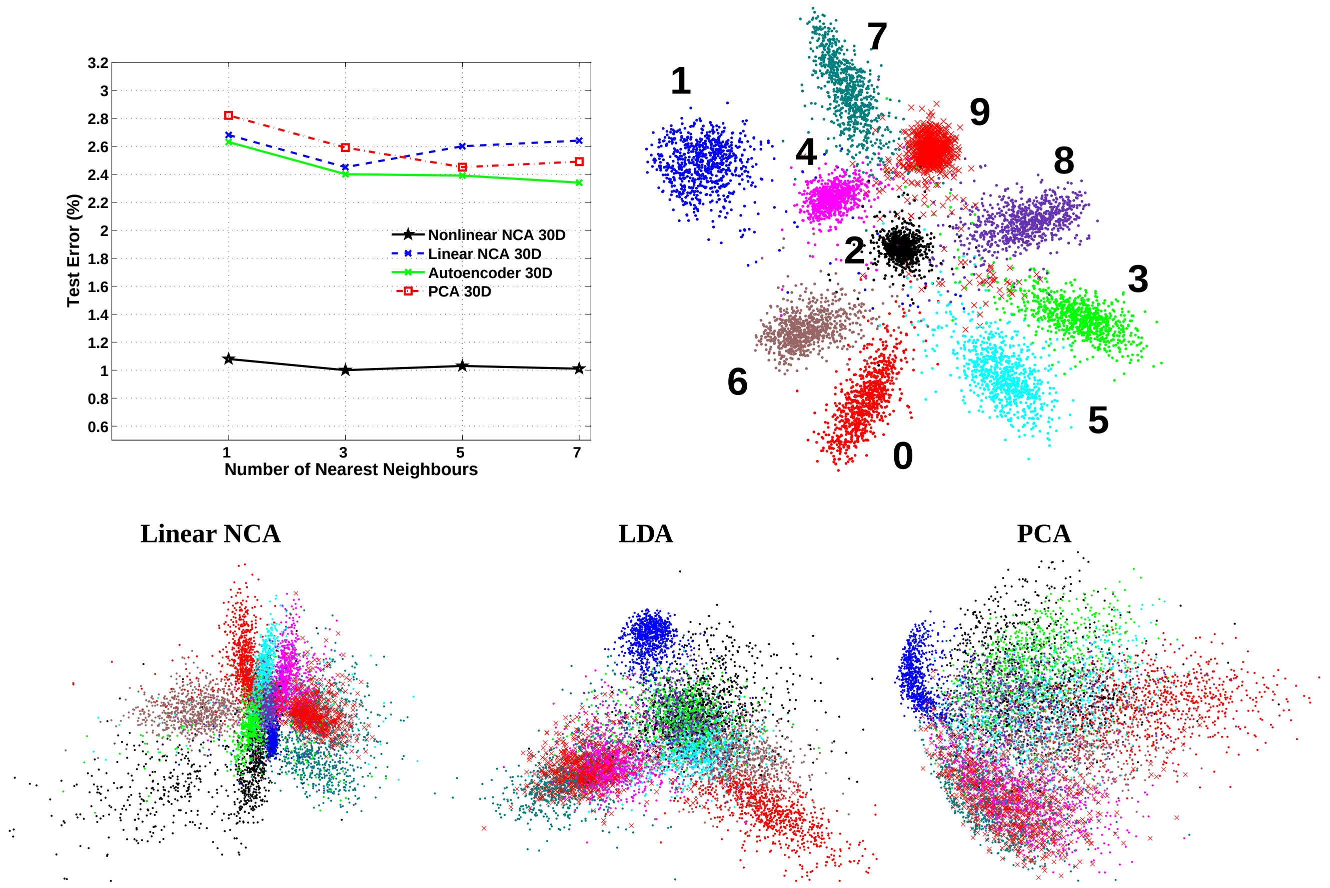


Figure 4: The top left panel shows KNN results on the MNIST test set. The top right panel shows the 2-dimensional codes produced by nonlinear NCA on the test data using a 784-500-500-2000-2 encoder. The bottom panels show the 2-dimensional codes produced by linear NCA, Linear Discriminant Analysis, and PCA.

contains 60,000 training and 10,000 test images of 28 $\times$ 28 handwritten digits. Out of 60,000 training images, 10,000 were used for validation. The original pixel intensities were normalized to lie in the interval $[0, 1]$ and had a preponderance of extreme values.

We used a $28\times28-500-500-2000-30$ architecture as shown as fig. 3, similar to one used in [12]. The 30 code units were linear and the remaining hidden units were logistic. Figure 4 shows that Nonlinear NCA, after 50 epochs of training, achieves an error rate of 1.08%, 1.00%, 1.03%, and 1.01% using 1,3,5, and 7 nearest neighbours. This is compared to the best reported error rates (without using any domain-specific knowledge) of 1.6% for randomly initialized backpropagation and 1.4% for Support Vector Machines [5]. Linear methods such as linear NCA or PCA are much worse than nonlinear NCA. Figure 4 (right panel) shows the 2-dimensional codes produced by nonlinear NCA compared to linear NCA, Linear Discriminant Analysis, and PCA.

5 Regularized Nonlinear NCA

In many application domains, a large supply of unlabeled data is readily available but the amount of labeled data, which can be expensive to obtain, is very limited so nonlinear NCA may suffer from overfitting.

After the pretraining stage, the individual RBM's at each level can be "unrolled" as shown in figure 3 to create a deep autoencoder. If the stochastic activities of the binary features are replaced by deterministic, real-valued probabilities, we can then backpropagate through the entire network to fine-tune the weights for optimal reconstruction of the data. Training such deep autoencoders, which does not require any labeled data, produces low-dimensional codes that are good at reconstructing the input data vectors, and tend to preserve class neighbourhood structure [14].

The NCA objective, that encourages codes to lie close to other codes belonging to the same class, can be combined with the autoencoder objective function (see fig. 3) to maximize:

$$C = \lambda O_{NCA} + (1 - \lambda)(-\mathrm{E}) \tag{16}$$

where O_{NCA} is defined in Eq. 5, E is the reconstruction error, and λ is a trade-off parameter. When the derivative of the reconstruction error E is backpropagated through the autoencoder, it is combined, at the code level, with the derivatives of O_{NCA}.

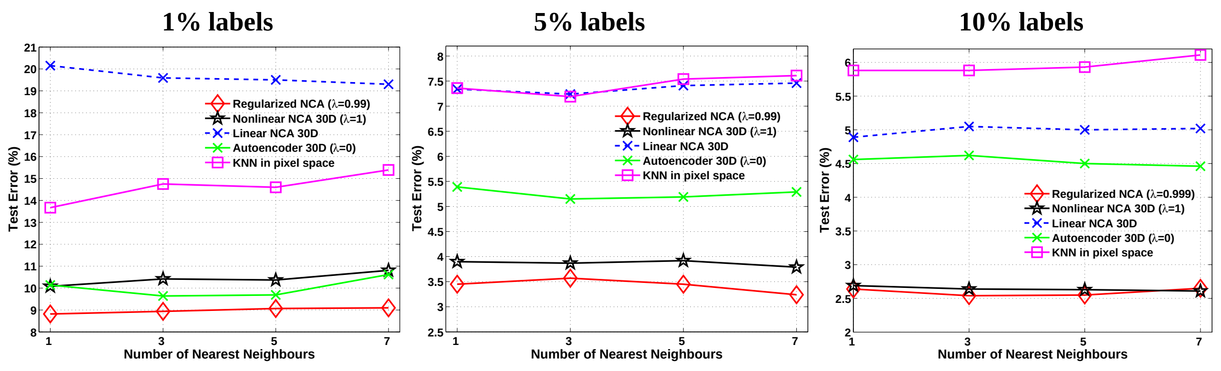
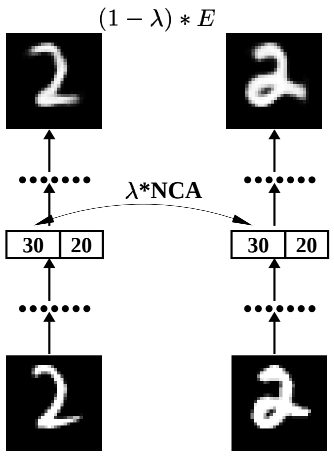


Figure 5: KNN on the MNIST test set when only a small fraction of class labels is available. Linear NCA and KNN in pixel space do not take advantage of the unlabeled data.



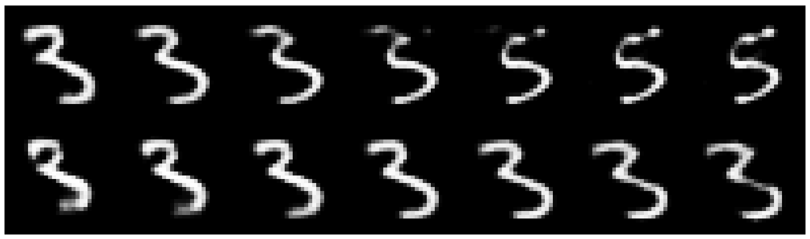


Figure 6: Left panel: The NCA objective function is only applied to the first 30 code units, but all 50 units are used for image reconstruction. Right panel: The top row shows the reconstructed images as we vary the activation of code unit 25 from 1 to -23 with a stepsize of 4. The bottom row shows the reconstructed images as we vary code unit 42 from 1 to -23.

This setting is particularly useful for semi-supervised learning tasks. Consider having a set of N_l labeled training data $(\mathbf{x}^l, c^l)$, where as before $\mathbf{x}^l \in R^d$, and $c^l \in \{1, 2, ..., C\}$, and a set of N_u unlabeled training data $\mathbf{x}^u$. Let $N = N_l + N_u$. The overall objective to maximize can be written as:

$$O = \lambda \frac{1}{N_l} \sum_{l=1}^{N_l} \sum_{b|c^l=c^b} p_{lb} + (1-\lambda)\frac{1}{N}\sum_{n=1}^{N} -E^n \qquad (17)$$

where E^n is the reconstruction error for the input data vector x^n. For the MNIST dataset we use the cross-entropy error:

$$E^p = -\sum_i x_i^n \log \hat{x}_i^n - \sum_i (1 - x_i^n) \log(1 - \hat{x}_i^n) \qquad (18)$$

where $x_i^n \in [0, 1]$ is the intensity of pixel i for the training example n, and $\hat{x}_i^n$ is the intensity of its reconstruction.

When the number of labeled example is small, regularized nonlinear NCA performs better than nonlinear NCA ($\lambda = 1$), which uses the unlabeled data for pretraining but ignores it during the fine-tuning. It also performs better than an autoencoder ($\lambda = 0$), which ignores the labeled set. To test the effect of the regularization when most of the data is unlabeled, we randomly sampled 1%, 5% and 10% of the handwritten digits in each class and treated them as labeled data. The remaining digits were treated as unlabeled data. Figure 5 reveals that regularized nonlinear NCA($\lambda = 0.99$)[4] outperforms both nonlinear NCA ($\lambda = 1$) and an autoencoder ($\lambda = 0$). Even when the entire training set is labeled, regularized NCA still performs slightly better.

5.1 Splitting codes into class-relevant and class-irrelevant parts

To allow accurate reconstruction of a digit image, the code must contain information about aspects of the image such as its orientation, slant, size and stroke thickness that are not relevant to its classification. These irrelevant aspects inevitably contribute to the Euclidean distance between codes and harm classification. To diminish this unwanted effect, we used 50-dimensional codes but only used the first 30 dimensions in the NCA objective function. The remaining 20 dimensions were free to code all those aspects of an image that do not affect its class label but are important for reconstruction.

Figure 6 shows how the reconstruction is affected by changing the activity level of a single code unit. Changing a unit among the first 30 changes the class; changing a unit among the last 20 does not. With $\lambda = 0.99$ the split codes achieve an error rate of 1.00% 0.97% 0.98% 0.97%

[4]The parameter λ was selected, using cross-validation, from among the values $\{0.5, 0.9, 0.99, 0.999\}$.

using 1,3,5, and 7 nearest neighbours. We also computed the 3NN error rate on the test set using only the last 20 code units. It was 4.3%, clearly indicating that the class-relevant information is concentrated in the first 30 units.

6 Conclusions

We have shown how to pretrain and fine-tune a deep nonlinear encoder network to learn a similarity metric over the input space that facilitates nearest-neighbor classification. Using the reconstruction error as a regularizer and split codes to suppress the influence of class-irrelevant aspect of the image, our method achieved the best reported error rate of 1.00% on a widely used version of the MNIST handwritten digit recognition task that does not use any domain-specific knowledge. The regularized version of our method can make good use of large amounts of unlabeled data, so the classification accuracy is high even when the amount of labeled training data is very limited. Comparison to other recent methods for learning similarity measures [1, 10, 20] remains to be done.

Acknowledgments
We thank Sam Roweis for many helpful discussions. This research was supported by NSERC, CFI and OTI. GEH is a fellow of CIAR and holds a CRC chair.

References

[1] Aharon Bar-Hillel, Tomer Hertz, Noam Shental, and Daphna Weinshall. Learning distance functions using equivalence relations. In *ICML*, pages 11–18. AAAI Press, 2003.

[2] S. Becker and G. E. Hinton. A self-organizing neural network that discovers surfaces in random-dot stereograms. *Nature*, 355(6356):161–163, 1992.

[3] S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *IEEE Computer Vision and Pattern Recognition or CVPR*, pages I: 539–546, 2005.

[4] T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, IT-13(1):21–7, January 1967.

[5] D. Decoste and B. Schölkopf. Training invariant support vector machines. *Machine Learning*, 46(1/3):161, 2002.

[6] Y. Freund and D. Haussler. Unsupervised learning of distributions on binary vectors using two layer networks. In *Advances in Neural Information Processing Systems 4*, pages 912–919, San Mateo, CA., 1992. Morgan Kaufmann.

[7] K. Fukumizu, F. R. Bach, and M. I. Jordan. Dimensionality reduction for supervised learning with reproducing kernel hilbert spaces. *Journal of Machine Learning Research*, 5:73–99, 2004.

[8] A. Globerson and S. T. Roweis. Metric learning by collapsing classes. In *NIPS*, 2005.

[9] J. Goldberger, S. T. Roweis, G. E. Hinton, and Ruslan Salakhutdinov. Neighbourhood components analysis. In *NIPS*, 2004.

[10] Tomer Hertz, Aharon Bar-Hillel, and Daphna Weinshall. Boosting margin based distance functions for clustering. In *ICML*, 2004.

[11] G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1711–1800, 2002.

[12] G. E. Hinton, S. Osindero, and Y. W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18, 2006.

[13] G. E. Hinton and S. T. Roweis. Stochastic neighbor embedding. In *NIPS*, pages 833–840. MIT Press, 2002.

[14] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, July 2006.

[15] The MNIST dataset is available at http://yann.lecun.com/exdb/mnist/index.html.

[16] P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. In *Parallel Distributed Processing: Volume 1: Foundations*, pages 194–281. MIT Press, Cambridge, 1986.

[17] J. V. Stone and N. Harper. Temporal constraints on visual learing: a computational model. *Perception*, 28:1089–1104, 2002.

[18] K. Q. Weinberger, J. Blitzer, and L. K. Saul. Distance metric learning for large margin nearest neighbor classification. In *NIPS*, 2005.

[19] M. Welling, M. Rosen-Zvi, and G. E. Hinton. Exponential family harmoniums with an application to information retrieval. In *NIPS 17*, pages 1481–1488, Cambridge, MA, 2005. MIT Press.

[20] Eric P. Xing, Andrew Y. Ng, Michael I. Jordan, and Stuart J. Russell. Distance metric learning with application to clustering with side-information. In *NIPS*, pages 505–512. MIT Press, 2002.

[21] R. S. Zemel and G. E. Hinton. Discovering and using the single viewpoint constraint. In *NIPS 3*, San Mateo, CA, 1991.

Appendix

We have O_{NCA} objective function of Eq. 5:

$$O_{NCA} = \sum_{a=1}^{N} \sum_{b:c^a=c^b} \frac{\exp\left(- \parallel f(\mathbf{x}^a|W) - f(\mathbf{x}^b|W) \parallel^2\right)}{\sum_{z\neq a} \exp\left(- \parallel f(\mathbf{x}^a|W) - f(\mathbf{x}^z|W) \parallel^2\right)}$$

Denote $d_{ab} = f(\mathbf{x}^a|W) - f(\mathbf{x}^b|W)$, then the derivatives of O_{NCA} with respect to parameter vector W for the a^{th} training case are

$$\frac{\partial O_{NCA}}{\partial W} = \frac{\partial O_{NCA}}{\partial f(\mathbf{x}^a|W)} \frac{\partial f(\mathbf{x}^a|W)}{\partial W}$$

where

$$\begin{aligned} \frac{\partial O_{NCA}}{\partial f(\mathbf{x}^a|W)} \quad = \quad & -2\Big[\sum_{b:c^a=c^b} p_{ab} d_{ab} - \sum_{b:c^a=c^b} p_{ab} \big[\sum_{z\neq a} p_{az} d_{az} \big] \Big] + \\ & 2\Big[\sum_{l:c^l=c^a} p_{la} d_{la} - \sum_{l\neq a} \big[\sum_{q:c^l=c^q} p_{lq} \big] p_{la} d_{la} \Big] \end{aligned}$$

and $\frac{\partial f(\mathbf{x}^a|W)}{\partial W}$ is computed using standard backpropagation.