A Deep Non-Linear Feature Mapping for Large-Margin kNN Classification

Renqiang Min
Department of Computer Science
University of Toronto
minrq@cs.toronto.edu

David A. Stanley
Electrical and Computer Engineering
University of Toronto
davearthur.stanley@gmail.com

Zineng Yuan
Department of Molecular Genetics
University of Toronto
zineng.yuan@utoronto.ca

Anthony Bonner
Department of Computer Science
University of Toronto
bonner@cs.toronto.edu

Zhaolei Zhang
Banting and Best Deptartment of Medical Research
University of Toronto
zhaolei.zhang@utoronto.ca

Abstract

KNN is one of the most popular data mining methods for classification, but it often fails to work well with inappropriate choice of distance metric or due to the presence of numerous class-irrelevant features. Linear feature transformation methods have been widely applied to extract class-relevant information to improve kNN classification, which is very limited in many applications. Kernels have also been used to learn powerful non-linear feature transformations, but these methods fail to scale to large datasets. In this paper, we present a scalable non-linear feature mapping method based on a deep neural network pretrained with Restricted Boltzmann Machines for improving kNN classification in a large-margin framework, which we call DNet-kNN. DNet-kNN can be used for both classification and for supervised dimensionality reduction. The experimental results on two benchmark handwritten digit datasets and one newsgroup text dataset show that DNet-kNN has much better performance than large-margin kNN using a linear mapping and kNN based on a deep autoencoder pretrained with Restricted Boltzmann Machines.

1 Introduction

kNN is one of the most popular classification methods due to its simplicity and reasonable effectiveness: it doesn't require fitting a model and it has been shown to have good performance for classifying many types of data. However, the good classification performance of kNN is highly dependent on the metric used for computing pairwise distances between data points. In practice, we often use Euclidean distances as similarity metric to calculate k nearest neighbors of data points of interest. To classify high-dimensional data in real applications, we often need to learn or choose a good distance metric.

Previous work on metric learning in [22] and [6] learns a global linear transformation matrix in the original feature space of data points to make similar data points stay closer while making dissimilar data points move farther apart using additional similarity or label information. And in [5], a global linear transformation is applied to the original feature space of data points to learn Mahalanobis metrics, which requires all data points in the same class collapse to one point. Making data points in the same class collapse is unnecessary for kNN classification. It may produce poor performance when data points cannot be essentially squeezed to points, which is often true for some class containing multiple patterns. In [13], a global linear transformation is learned to directly improve kNN classification to achieve the goal of a large margin. This method has been shown to yield significant improvement over kNN classification, but the linear transformation often fails to give good performance in high-dimensional space and a pre-processing step of dimensionality reduction by PCA is often required for success.

In many situations, a linear transformation is not powerful enough to capture the underlying class-specific data manifold. A locally adaptive distance metric learning is

used in [7], but locality is hard to be specified in advance. Therefore, we need to resort to more powerful non-linear transformations, so that each data point will stay closer to its nearest neighbors having the same class as itself than to any other data in the non-linearly transformed feature space. Kernel tricks can achieve this goal sometimes, and they have been used to kernelize some of the above methods in order to improve kNN classification [5]. The method in [18] extends the work in [13] to perform linear dimensionality reduction to improve large-margin kNN classification and kernelized the method in [13]. However, the kernel-based approaches behave almost like template-based approaches. If the chosen kernel cannot well reflect the true class-related structure of the data, the resulting performance will be bad. Besides, kernel-based approaches often have difficulty in handling large datasets.

We might want to achieve non-linear mappings by learning a directed multi-layer belief net or a deep autoencoder, and then perform kNN classification using the hidden distributed representations of the original input data. However, a multi-layer belief net often suffers from the "explaining away" effect, that is, the top hidden units become dependent conditional on the bottom visible units, which makes inference intractable; and learning a deep autoencoder with back-propagation is almost impossible because the gradient back-propagated to the lower layers from the output often becomes very noisy and meaningless. Fortunately, recent research has shown that training a deep generative model called Deep Belief Net is feasible by pretraining the deep net using a type of undirected graphical model called Restricted Boltzmann Machine (RBM) [11]. RBMs produce "complementary priors" to make the inference process in a deep belief net much easier, and the deep net can be trained greedily layer by layer using the simple and efficient learning rule of RBM. The greedy layer-wise pretraining strategy has made learning models with deep architectures possible [14, 1]. Moreover, the greedy pretraining idea has also been successfully applied to initialize the weights of a Deep Autoencoder (DA) to learn a very powerful non-linear mapping for dimensionality reduction, which is illustrated in Fig. 1a) and 1b). Besides, the idea of deep learning has motivated researchers to use powerful generative models with deep architectures to learn better discriminative models [21].

In this paper, by combining the idea of deep learning and large-margin discriminative learning, we propose a new kNN classification and supervised dimensionality reduction method called DNet-kNN. It learns a non-linear feature transformation to directly achieve the goal of large-margin kNN classification, which is based on a Deep Encoder Network pretrained with RBMs as shown in Fig 2. Our approach is mainly inspired by the work in [13], [18] and [12]. Given the labels of some or all training data, it allows us to learn a non-linear feature mapping to minimize the invasions to each data point's genuine neighborhood by other impostor nearest neighbors, which favors kNN classification directly. Previous researchers once used an autoencoder or a deep autoencoder for non-linear dimensionality reduction to improve kNN [12, 16, 15]. None of these approaches used an objective function as directly as what we use here for improving kNN classification. The approach discussed in [3] uses a convolution net to learn a similarity metric discriminatively, but it was handcrafted. Our approach based on general deep neural networks is more flexible and the connection weight matrices between layers are automatically learned from data.

We applied DNet-kNN on the USPS and MNIST handwritten digit datasets for classification. The test error we obtained on the MNIST benchmark dataset is 0.94%, which is better than that obtained by deep belief net, deep autoencoder and SVM [4, 12, 11]. In addition, our fine-tuning process is very fast and converges to a good local minimum within several iterations of conjugate-gradient update. Our experimental results show that: (1) a good generative model can be used as a pretraining stage to improve discriminative learning; (2) pretraining with generative models in a layer-wise greedy way makes it possible to learn a good discriminative model with deep architecture; (3) pretraining with RBMs makes discriminative learning process much faster than that without pretraining; (4) pretraining helps to find a much better local minimum than without pretraining. These conclusions are consistent with the results of previous research trials on deep networks [14, 1, 21, 11, 12].

We organize this paper as follows: in section 2, we introduce kNN classification using linear transformations in a large-margin framework. In section 3, we describe previous work on RBM and training models with deep architectures. In section 4, we present DNet-kNN, which trains a Deep Encoder Network for improving large-margin kNN classification. In section 5, we present our experimental results on USPS handwritten digit data, MNIST [20] handwritten digit data, and 20 newsgroup data with binary features. In section 6, we conclude the paper with some discussions and propose possible extensions of our current method.

2 Large-margin kNN classification using linear transformation

In this section, we review the large-margin framework of kNN classification using a linear transformation described in [13], which is called LMNN. Given a set of data points $\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)} : i = 1, \ldots, n\}$ and additional neighborhood information η, where $\mathbf{x}^i \in R^D$, $y^{(i)} \in \{1, \ldots, c\}$ for labeled data points, c is the total number of classes, and $\eta_{il} = 1$ if l is one of i's k target neighbors, we seek a distance function $d(i, l)$ for pairwise data points i and l such that the given neighborhood information will be preserved

in the transformed feature space corresponding to the distance function. If $d(i,l)$ is based on Mahanalobis distances, then it admits the following form:

$$d_{\mathbf{A}}(i,l) = (\mathbf{x}^{(i)} - \mathbf{x}^{(l)})^T \mathbf{A}^T \mathbf{A} (\mathbf{x}^{(i)} - \mathbf{x}^{(l)}), \quad (1)$$

where $\mathbf{A}$ is a linear transformation matrix. Based on the goal of margin maximization, we learn the parameters of the distance function, $\mathbf{A}$, such that, for each data point i, the distance between i and each data point j from another class will be at least 1 plus the largest distance between i and its k target neighbors. Using a binary matrix $y_{ij} = 1$ to represent that i and j are in the same class and 0 otherwise for the labeled data points, we can formulate the above problem as an optimization problem:

$$min_{\mathbf{A}} \quad \sum_{il} \eta_{il} d_{\mathbf{A}}(i,l) + C \sum_{ilj} \eta_{il}(1 - y_{ij}) h(1 + d_{\mathbf{A}}(i,l) - d_{\mathbf{A}}(i,j)), \quad (2)$$

where C is a penalty coefficient penalizing constraint violations, and $h(\cdot)$ is a hinge loss function with $h(z) = max(z, 0)$. If $\mathbf{A}$ is a $D \times D$ matrix, this problem corresponds to the work in [13]; if $\mathbf{A}$ is a $d \times D$ matrix where $d < D$, this problem corresponds to the work in [18]. When a non-square matrix $\mathbf{A}$ is learned for dimensionality reduction, the resulting problem is non-convex, stochastic gradient descent and conjugate gradient descent are often used to solve the problem. When $\mathbf{A}$ is constrained to be a full-rank square matrix, we can solve $\mathbf{A}^T\mathbf{A}$ directly and the resulting problem is convex. Alternating projection or simple gradient-based methods can be applied here [13].

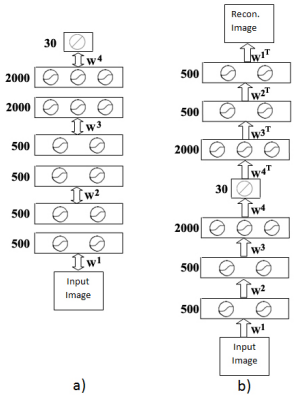

Figure 1. RBM pretraining and deep autoencoder

3 RBM and Deep Neural Network

On large datasets, rich information existing in data features often enables us to build powerful generative models to learn the constraints and the structures underlying the given data. The learned information often reveals the characteristics of data points belonging to different classes. In [12], it is shown that a deep belief net composed of stacked Restricted Boltzmann Machines (RBM) can perform handwritten digit classification remarkably well [17]. RBM is an undirected graphical model with one visible layer $\mathbf{v}$ and one hidden layer $\mathbf{h}$. There are symmetric connections $\mathbf{W}$ between the hidden layer and the visible layer, but there are no within-layer connections. For a RBM with stochastic binary visible units $\mathbf{v}$ and stochastic binary hidden units $\mathbf{h}$, the joint probability distribution of a configuration $(\mathbf{v}, \mathbf{h})$ of RBM is defined based on its energy as follows:

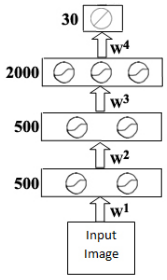

Figure 2. Deep Encoder

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{ij} W_{ij} v_i h_j - \sum_i v_i b_i - \sum_j h_j c_j \quad (3)$$

$$p(v,h) = \frac{1}{Z} exp(-E(\mathbf{v}, \mathbf{h})), \quad (4)$$

where $\mathbf{b}$ and $\mathbf{c}$ are biases, and Z is the partition function with $Z = \sum_{\mathbf{u},\mathbf{g}} exp(-E(\mathbf{u}, \mathbf{g}))$. The good property due to the structure of RBM is that, given the visible states, each hidden unit is conditionally independent, and given the hidden states, the visible units are conditionally independent.

$$p(v_i = 1|\mathbf{h}) = sigmoid(\sum_j W_{ij} h_j + b_i), \quad (5)$$

$$p(h_j = 1|\mathbf{v}) = sigmoid(\sum_i W_{ij} v_i + c_j), \quad (6)$$

where $sigmoid(z) = \frac{1}{1+exp(-z)}$. This beneficial property allows us to get unbiased samples from the posterior distribution of the hidden units given an input data vector. For a RBM with binary stochastic visible units and Gaussian stochastic hidden units js with variance σ_js, the energy function becomes:

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{ij} W_{ij} v_i \frac{h_j}{\sigma_j} - \sum_i v_i b_i + \sum_j \frac{(h_j - c_j)^2}{2\sigma_j^2}.$$

The conditional probabilities $p(v_i = 1|\mathbf{h})$ and $p(h_j|\mathbf{v})$ become:

$$p(v_i = 1|\mathbf{h}) = sigmoid(\sum_j W_{ij}\frac{h_j}{\sigma_j} + b_i),$$
$$p(h_j|\mathbf{v}) = \mathcal{N}(\sigma_j \sum_i W_{ij} v_i + c_j, \sigma_j),$$

where $\mathcal{N}(\mu, \sigma)$ is a Gaussian distribution with mean μ and variance σ. In practice, we often set all σ_js to 1. By minimizing the negative log-likelihood of the observed input data vectors using gradient descent, the update rule for the weight $\mathbf{W}$ turns out to be,

$$\begin{aligned}\Delta W_{ij} &= \epsilon(< v_i h_j >_{data} - < v_i h_j >_\infty), \ \ binary \ \ \mathbf{h}, \\ \Delta W_{ij} &= \epsilon(< v_i \frac{h_j}{\sigma_j} >_{data} - < v_i \frac{h_j}{\sigma_j} >_\infty), \ \ Gaussian \ \ \mathbf{h}, \end{aligned} \quad (7)$$

where ϵ is learning rate, $< \cdot >_{data}$ denotes the expectation with respect to the data distribution and $< \cdot >_\infty$ denotes the expectation with respect to the model distribution. In practice, we do not have to sample from the equilibrium distribution of the model, and even one-step reconstruction samples work very well [9].

$$\begin{aligned}\Delta W_{ij} &= \epsilon(< v_i h_j >_{data} - < v_i h_j >_1), \ \ binary \ \ \mathbf{h}, \\ \Delta W_{ij} &= \epsilon(< v_i \frac{h_j}{\sigma_j} >_{data} - < v_i \frac{h_j}{\sigma_j} >_1), \ \ Gaussian \ \ \mathbf{h}, \end{aligned} \quad (8)$$

Although the above update rule does not follow the gradient of the log-likelihood of data exactly, it approximately follows the gradient of another objective function [2]. In [11], it is shown that a deep belief net based on stacked RBMs can be trained greedily layer by layer. Given some observed input data, we train a RBM to get the hidden representations of the data. We can view the learned hidden representations as new data and train another RBM. We can repeat this procedure many times. It is shown that in [11], under this greedy training strategy, we always get a better model $p(\mathbf{h})$ for hidden representations of the original input data if the number of features in the added layer does not decrease, and the following varational lower bound of the log-likelihood of the observed input data never decreases.

$$\begin{aligned} logp(\mathbf{v}) &= log(\sum_h p(\mathbf{v}|\mathbf{h})p(\mathbf{h})) \\ &= log(\sum_{\mathbf{h}} q(\mathbf{h}|\mathbf{v})\frac{p(\mathbf{v}|\mathbf{h})p(\mathbf{h})}{q(\mathbf{h}|\mathbf{v})}) \\ &\geq \sum_{\mathbf{h}} q(\mathbf{h}|\mathbf{v}) log((p(\mathbf{v}|\mathbf{h})p(\mathbf{h})) - \\ &\quad \sum_{\mathbf{h}} q(\mathbf{h}|\mathbf{v}) log q(\mathbf{h}|\mathbf{v}). \end{aligned}$$

In [12], the greedy training strategy is used to initialize the weights of a deep autoencoder as shown in Fig. 1a) and then back-propagation is used for tuning the weights of the network as shown in Fig 1b). This time the lower-bound guarantee no longer holds, but the greedy pre-training still works very well in practice [12].

4 Large-margin kNN classification using deep neural networks

The work in [12] and [11] made full use of the capabilities of generative models, but label information is only weakly used. In the following, we describe DNet-kNN. DNet-kNN is based on a deep encoder network with 4 hidden layers as shown in Fig. 2, in which the input layer and the first three hidden layers all have binary stochastic units, and the last hidden layer (feature output layer) has Gaussian stochastic units with variance 1. The number of hidden units in each layer is listed on the left in both Fig. 1 and Fig. 2. We use stacked RBMs to initialize the weights of the encoder (we can also optionally further use a deep autoencoder to find a better initialization). Then we fine-tune the weights of the encoder by minimizing the following objective function:

$$\ell_{ilj} = h(1 + d_f(i, l) - d_f(i, j)), \quad (9)$$

$$min_f \ \ell_f = \sum_{ilj} \eta_{il}\gamma_{ij}\ell_{ilj}, \quad (10)$$

where $\gamma_{ij} = 1$ if and only if i is an impostor neighbor of j, which will be discussed in details later. The definition of η_{il} is the same as discussed before in section 2, and $d_f(i, l) = ||f(\mathbf{x}^{(i)}) - f(\mathbf{x}^{(l)})||^2$. The function $f(\cdot) : R^D \rightarrow R^d$ is a continuous non-linear mapping, and each component of $f(\mathbf{x})$ is a continuous function of the input vector $\mathbf{x}$, and the parameters $\mathbf{W}$ of f are connection weight matrices in a deep neural network. For example, in Fig. 2, $\mathbf{W} = \{\mathbf{W}^i, i = 1, \ldots, 4\}$. This equation differs from Eqn 2 in two ways. First, the distance between data points i and j is computed by the Euclidean distance using the feature vectors output by the top layer of the encoder f in Fig. 2. Secondly, the objective function ℓ_f focuses on maximizing the margin and neglects the term reducing the distance between nearest neighbors. Additionally, unlike Hinton's deep autoencoder, we no longer minimize the reconstruction error, since it was found that this criterion reduced the ability of the code vectors to accurately describe subtle differences between classes in practice.

To reduce the complexity of the back-propagation training, we use simplified versions of η_{il} and γ_{ij} in the objective function 10, as compared to those described in Section 2. For each index i, $\eta_{il} = 1$ only if l is one of i's top k nearest neighbors among the data points having the same class label as i (in-class nearest neighbors). In contrast, for

each i, the js for which $\gamma_{ij} = 1$ are selected from the set of impostor nearest neighbors of i, which is the union of the m nearest neighbors from each and every class other than the class of i. For example, in the case of digit recognition with ten classes, there are a total of $m \times 9$ impostor js for each data point i. This method of choosing impostor nearest neighbors is optimal for kNN classification because, by selecting m impostor neighbors from every other class, we help ensure that all potential competitors are removed.

Let $\mathbf{y}^{(i)} = f(\mathbf{x}^{(i)})$ be the low dimensional code vector of $\mathbf{x}^{(i)}$ generated by the Deep Encoder Network. Then the time complexity of computing Eq. 10 is $O((c-1)kmn)$, which is a significant improvement over the time complexity of Eq. 2, which is $O(kn^2)$. For the purposes of calculating nearest neighbors and impostor nearest neighbors, we use Euclidean distances in the pixel space. This means that the η_{il} and y_{ij} do not need to be recalculated each time the code vectors are updated. Unfortunately, due to the nonlinear mapping, this may mean that ordinary data points in the pixel space may become impostors in the code space and will not be taken into account in the objective function. However, it is likely that the mapping is quasi-linear. Therefore, by taking a large value for m, we find that this captures most of the impostors in the code space, as evidenced by our low kNN classification errors. In our experiments, we use k=5 and $m = 30$.

To improve the computation time of calculating the objective function gradient, a $((c-1)kmn)$ x 3 matrix of triples was generated. These triples represent the sets of all allowed indices i, j, and l in Eq 10 for which η_{il} and γ_{ij} are non-zero. Therefore, in the triples matrix, the entries in the 2nd column represent the in-class nearest neighbors relative to the first column, and the entries in the 3rd column represent the impostor nearest neighbors relative to the first column. The triples matrix is used in calculating both the gradient of the objective function, and the value of the objective function itself.

The gradient of the objective function, relative to the code vector $\mathbf{y}^{(i)} = f(\mathbf{x}^{(i)})$ is given by:

$$\begin{aligned}\frac{\partial \ell_f}{\partial \mathbf{y}^{(i)}} = \quad & -2\textstyle\sum_{jl} \eta_{il}\gamma_{ij}\theta_{ilj}(\mathbf{y}^{(l)} - \mathbf{y}^{(j)}) \\ & -2\textstyle\sum_{jk} \eta_{ki}\gamma_{kj}\theta_{kij}(\mathbf{y}^{(k)} - \mathbf{y}^{(i)}) \\ & +2\textstyle\sum_{kl} \eta_{kl}\gamma_{ki}\theta_{kli}(\mathbf{y}^{(k)} - \mathbf{y}^{(i)}) \end{aligned} \tag{11}$$

where θ is the flag for margin violations: $\theta_{klj} = 1$ if $(1 + d_f(k,l) - d_f(k,j)) > 0$.

While this equation seems to be unwieldy for implementation, the use of the triples matrix makes the computation much easier. In Eq 11, we calculated each sum individually, using the triples matrix to determine the appropriate indices, and then combined them later. For example, to determine the value of the first summation term,

$$\sum_{jl} \eta_{il}\gamma_{ij}\theta_{ilj}(\mathbf{y}^{(l)} - \mathbf{y}^{(j)}),$$

we simply search through the triples matrix to identify all the triples that yield a margin violation ($\theta = 1$). Then, we choose those that have index i in their first column. Thus, this specific set of triples tells us the appropriate indices to use in the first sum. Specifically, the second column of the triples matrix becomes the l index values, and the third column becomes the j index values. Likewise, the same strategy is repeated for the second and third summations. After computing $\frac{\partial \ell_f}{\partial \mathbf{y}_i}$, we compute the derivative of the objective function ℓ_f with respect to the network weight matrix $\mathbf{W}^4$ using the following chain rule:

$$\frac{\partial \ell_f}{\partial \mathbf{W}^4} = \sum_{i=1}^{n}\sum_{s=1}^{d} \frac{\partial \ell_f}{\partial y_s^{(i)}} \frac{\partial y_s^{(i)}}{\partial \mathbf{W}^4}, \tag{12}$$

where d is the dimensionality of feature vector $\mathbf{y}^{(i)}$ and $y_s^{(i)}$ is the s-th component of $\mathbf{y}^{(i)}$. And the derivatives $\frac{\partial \ell_f}{\partial \mathbf{W}^i}, i = 1, 2, 3$, are computed using traditional back-propagation algorithm referring to the network structure in Fig. 2. The training procedure of DNet-kNN is shown in

Algorithm 1 The training procedure of DNet-kNN (the description in [] is optional).

1: **Input**: training data $\{\mathbf{x}^{(i)}, y^{(i)} : i = 1, \ldots, n\}$, k, m, $[T]$.
2: pretrain the network in Fig. 2 with RBMs using Eq. 8 to get initial network weights $\mathbf{W}^{init}$.
3: [Further train a deep autoencoder for T iterations to get $\mathbf{W}^{init\_new}$, and set $\mathbf{W}^{init} = \mathbf{W}^{init\_new}$.]
4: calculate each data point i's k true nearest neighbors in its class, $i = 1, \ldots, n$.
5: calculate each i's $m \times (c-1)$ imposter nearest neighbors, $i = 1, \ldots, n$.
6: create triples (i, l, j).
7: set $\mathbf{W} = \mathbf{W}^{init}$.
8: **while** ($< not\ \ convergence >$)
9: update $\mathbf{W}$ using conjugate gradient based on Eq. 11-12
10: **Output**: $\mathbf{W}$.

Algorithm 1. The input parameter T and the operations in the square bracket on line 3 are optional. The convergence criteria is achieved when either the training error no longer decreases or the objective function ℓ_f no longer decreases. After we get the learned network weights $\mathbf{W}$ in Fig. 2, we compute test data points' feature vectors $\mathbf{y}$s using $\mathbf{W}$ in a bottom-up pass and predict their labels by performing kNN classification in the $\mathbf{y}$ space.

5 Experimental Results

We will test our model DNet-kNN for both classification and dimensionality reduction on two handwritten digit datasets, USPS and MNIST, and one binary newsgroup text dataset. We demonstrate two different types of classification: standard kNN and minimum energy classification. For standard kNN, after we finish learning the non-linear mapping by discriminative fine-tuning, we can directly compute pairwise Euclidean distances for kNN classification, which is used in DNet-kNN. Alternatively, for minimum energy classification, after we calculate the feature vectors of training data and test data, we can also predict the class label of a test data point by the class to which the test data point is assigned to have the lowest energy defined by Eq. 10. This minimum energy classification is denoted by "-E" in the experimental results. In both USPS and MINIST experiments, we set $k = 5$ and $m = 30$. On the two benchmark datasets, we compare DNet-kNN to some other well-known methods, in which LMNN requires a pre-processing by PCA for its success. As suggested by the authors of LMNN, we reduced the digit data to 100 dimensions first using PCA and then used LMNN to learn a linear transformation.

The network structure in Fig. 1 for Deep Autoencoder (DA) and the one in Fig. 2 for DNet-kNN (the number of hidden units in each layer is listed on the left) were used because they have good cross-validation classification performance for handwritten digit classification as discussed in [16]. Another reason for using this architecture is to facilitate method comparisons. It is possible to explore other good network architechures for DNet-kNN. As in [12], in the greedy layer-wise pretraining stage of both DNet-kNN and deep autoencoder based on RBMs, we used mini-batch training with batch size 100, we set learning rate to 0.1 and weight decay coefficient to 2e-4. We set initial momentum to 0.5 in the first 5 iterations and final momentum to 0.9 in later iterations. We performed 50 iterations of pretraining using stacked RBMs on all the three datasets, and we used conjugate gradient method to minimize the margin violation cost in Eq. 10.

5.1 Experimental Results on USPS Dataset

We downloaded the USPS digit dataset from a public link[1]. In this dataset, each digit is represented by a 16 by 16 gray-level image. From this dataset, several different preparations are used. The first preparation is USPS-fixed, which takes the first 800 data points from each of the ten digit classes to create an 8000 point training set. The test set for USPS-fixed then consists of a further 3000 data points, with 300 from each data class.

[1] http://www.cs.toronto.edu/ roweis/data/usps_all.mat

Second to sixth preparations, called US1 to US5 are then obtained from USPS-fixed by randomly shuffling the data points for each class between training and testing datasets.

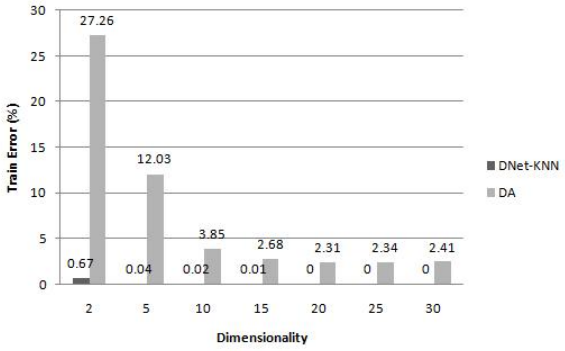

Figure 3. Training error on USPS-fixed using two different classification methods: the Deep Neural Network kNN classifier (DNet-kNN) and the deep autoencoder (DA).

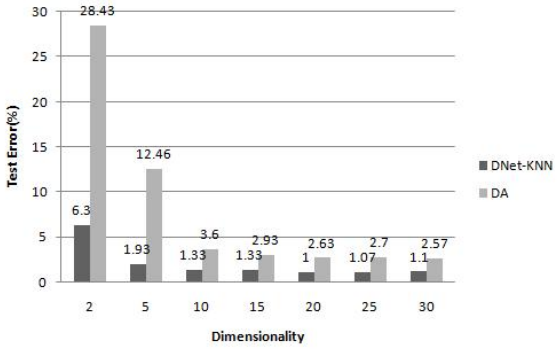

Figure 4. Test error on USPS-fixed using two different classification methods: the Deep Neural Network kNN classifier (DNet-kNN) and the deep autoencoder (DA).

In Figures 3 and 4, we observe the training errors and test errors for different dimensionality codes. In all cases, the DNet-kNN classification outperforms the deep autoencoder (DA). Furthermore, as the dimensionality of the codes increases, the classification error decreases. This trend coninues from d=2 up till d=20, and then levels off. As is discussed in section 4, DNet-kNN is pretrained with RBMs. If there is no pretraining, the error rate on USPS-fixed is

18.27%.

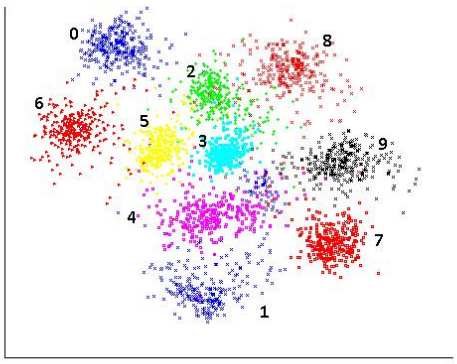

Figure 5. Two-dimensional embedding of 3000 USPS-fixed test data using the Deep Neural Network kNN classifier (DNet-kNN).

Fig. 5-7 compares the dimensionality reduction to two dimensions by DNet-kNN to the the ones by the deep autoencoder and PCA. The DNet-kNN clearly produces superior clustering of data point classes in two-dimensional space. There are still some class overlaps, however, because the back-propagation algorithm we use to optimize for kNN classification is not the best choice to improve visualization. This is because the objective function chooses which data points it considers to be in the set of impostor nearest neighbors (allowed j's) using the pixel space rather than the code space (see Section 4). However, visualization requires reduction to very low-dimensional spaces, and the mapping from pixel space to code space must become highly non-linear as dimensionality is reduced. Therefore, the pixel space becomes a poorer representation of spatial relationships in the code space and the correct choice of impostor nearest neighbors becomes less reliable during visualization.

Table 1. Training error of different methods on 5 random splits of USPS dataset (%). The lowest errors are shown in bold. DNet code dim=30

	US1	US2	US3	US4	US5
DNet-kNN	**0.00**	**0.00**	**0.00**	**0.00**	**0.01**
LMNN	0.76	1.10	0.71	0.85	0.95
DA	2.66	2.35	2.28	2.24	2.48
kNN	5.12	5.09	4.95	5.08	4.93

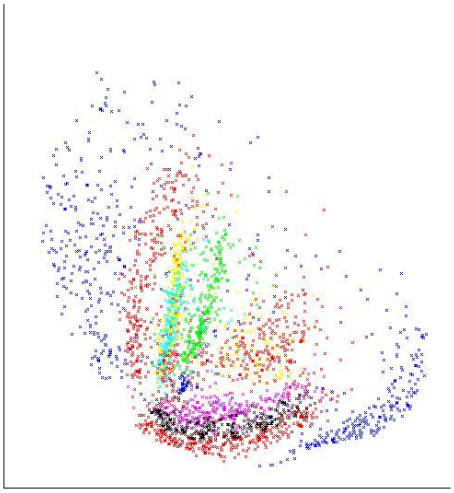

Figure 6. Two-dimensional embedding of 3000 USPS-fixed test data using the Deep Autoencoder (DA).

Table 2. Test error of different methods on the same 5 random splits of USPS dataset. "-E" denotes the energy classification method (%). The lowest errors are shown in bold. DNet code dim=30

	US1	US2	US3	US4	US5
DNet-kNN	**1.20**	**0.87**	**1.43**	**1.06**	1.13
DNet-kNN-E	1.43	0.97	1.50	1.20	**1.00**
LMNN	2.20	2.13	2.36	2.33	1.93
LMNN-E	1.77	1.53	1.80	1.80	1.63
DA	2.80	2.36	2.33	1.93	2.23
kNN	4.47	4.93	5.23	4.17	5.37

Tables 1 and 2 show respectively the training and test error on multiple USPS-random data sets. DNet-kNN almost consistently outperforms the other methods.

5.2 Experimental Results on MNIST Dataset

This section deals with the MNIST dataset, which is another digit set available online[2]. In MNIST, each digit is

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

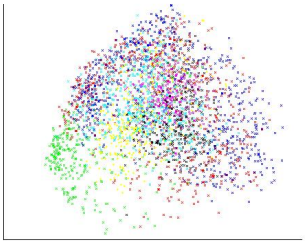

Figure 7. Two-dimensional embedding of 3000 USPS-fixed test data using PCA.

represented by a 28 by 28 gray-level image. This dataset contains 60,000 training samples and 10,000 test samples. For the USPS dataset, it was possible to do both the pre-training and the back-propagation on a single batch of data. However, given the size of the MNIST dataset, the training data had to be broken into smaller batches of 10,000 randomly selected data points. Then, RBM training and back-propagation could be applied iteratively to each batch. In all our experiments, batch size was set to 10,000.

Fig. 8-10 shows the mapping of the MNIST test data onto a reduced space using the DNet-kNN, the deep autoencoder and PCA. As with the USPS dataset, DNet-kNN shows a significant improvement over the deep autoencoder and PCA.

Table 3. Test error of different methods on the benchmark MNIST dataset. "-E" denotes the energy classification method (%). For different kNN-based methods, k = 5.

Methods	results
DNet-kNN (dim = 30, batch size=1.0e4)	**0.94**
DNet-kNN-E (dim = 30, batch size=1.0e4)	0.95
Deep Autoencoder (dim = 30, batch size=1.0e4)	2.13
Non-linear NCA based on a Deep Autoencoder ([16]	1.03
Deep Belief Net [11]	1.25
SVM: degree 9 [4]	1.4
kNN (pixel space)	3.05
LMNN	2.62
LMNN-E	1.58
DNet-kNN (dim = 2, batch size=1.0e4)	2.65
DNet-kNN-E (dim = 2, batch size=1.0e4)	2.65
Deep Autoencoder (dim = 2, batch size=1.0e4)	24.7

Table 3 shows the classification error of the DNet-kNN as compared to other common classification techniques on the MNIST dataset. Despite the fact that we must use batches, the DNet-kNN still produces the best classifications. This indicates that the DNet-kNN classifier is highly robust, since it can perform well when limited to seeing only part of the dataset at any one time.

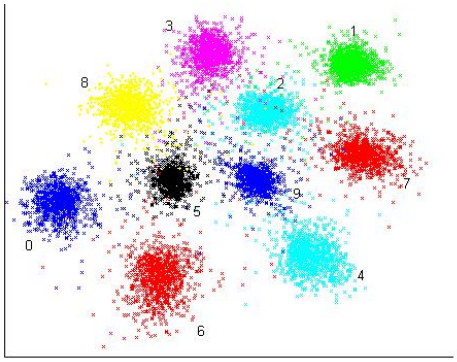

Figure 8. Two-dimensional embedding of 10,000 MNIST test data using the Deep Neural Network kNN classifier (DNet-kNN).

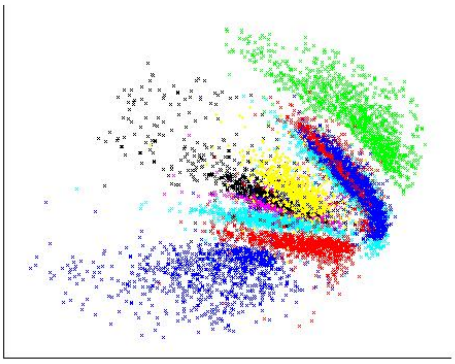

Figure 9. Two-dimensional embedding of 10,000 MNIST test data using the Deep Autoencoder (DA).

Finally, it is worth noting that, unlike the deep autoencoder, the fine tuning of the DNet-kNN classifier during back-propagation displays extremely fast convergence. Often, the error reaches a minimum after three to five iterations. This is due to the fact that the RBM pretraining

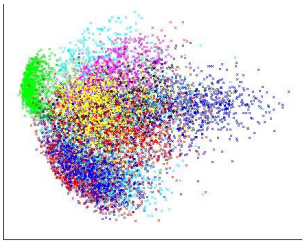

Figure 10. Two-dimensional embedding of 10,000 MNIST test data using PCA.

has provided an ideal starting point and also that we are using a supervised learning algorithm, as opposed to an unsupervised algorithm as in the deep autoencoder. For DNet-kNN without pretraining with RBMs, after minimizing the margin violations using conjugate gradient method for one week on a machine with 2.6GHz CPU and 64GB memory, the test error rate is still above 3.0%.

5.3 Experimental Results on Binary 20 Newsgroup Dataset

Besides handwritten digit recognition, we also applied our method to classify 20 newsgroup text data with binary features, which is publicly available [3]. This dataset contains binary occurrences for 100 words across 16242 postings. Our task is to use the binary feature vectors to classify these posts into four categories, which are computer, recreation, science, and talks. We performed 5-fold cross validation: divide these posts into 5 folds, train on four folds and test on the remaining fold, and cycle this procedure 5 times. The test errors on 5 different test folds for different methods are shown in Table 4, in which each column corresponds to one test fold. LMNN failed to work on this binary dataset. Table 4 clearly shows that DNet-kNN and DNet-kNN-E performed much better than other methods. On fold 5, DNet-kNN and DNet-kNN -E produced the lowest error rates among all the 5 test folds. Therefore, we used fold 5 as test data and made them embedded into a two dimensional space. In this case, the test error rate produced by DA is 28.4%. Based on the intialization of only stacked RBMs, the error rate produced by DNet-kNN is 24.0%, and the error rate produced by DNet-kNN-E is 21.5%, but based on the initialization of both stacked RBMs and DA, the error rate produced by DNet-kNN is 22.6%, and the error rate produced by DNet-kNN-E is 20.6%. These results suggested that the combination of stacked RBMs and DA helped to find better initialization weight matrices for DNet-kNN.

[3] http://www.cs.toronto.edu/ roweis/data/20news_w100.mat

Table 4. Test error of different methods for 5-fold cross validation on binary 20 newsgroup text data. "-E" denotes the energy classification method (%). The lowest errors are shown in bold. DNet code dim=30

	1	2	3	4	5
DNet-kNN	23.8	22.9	23.1	24.0	22.1
DNet-kNN-E	**19.1**	**18.8**	**18.9**	**19.6**	**17.9**
DA	27.0	25.0	27.0	28.4	27.0
kNN	32.6	33.1	32.8	34.3	30.9

6 Discussions and future research

In this paper, we have presented a new non-linear feature mapping method called DNet-kNN that uses a deep encoder network pretrained with RBMs to achieve the goal of large-margin kNN classification. Our experimental results on USPS and MNIST handwritten digits and newsgroup text data show that, DNet-kNN is powerful in both classification and non-linear embedding. Our results suggest that, pretraining with a good generative model is very helpful for learning a good discriminative model, and the pretraining makes discriminative learning much faster, and it often helps to find a much better local minimum especially in a deep architecture than without pretraining. Our findings verified the hypothesis discussed in [10].

On huge datasets, the current implementation of our method only works by using mini-batches. We essentially compute the genuine nearest neighbors and impostor nearest neighbors in each mini-batch, which might be not optimal over the whole dataset. We will develop a dynamic version of DNet-kNN, in which the mini-batches will change dynamically during training and we dynamically update the true nearest neighbors and impostor nearest neighbors of each data point. And we will explore more efficient algorithms for computing the derivative in Eq. 11.

The classification performance of DNet-kNN can be possibly further improved by training deep autoencoder first, and then fine-tune network weights by minimizing Eq. 10. Additionally, we plan to use the label information of training data to constrain the distances between pairwise data points in the same class. In specific, we will add a penalty term using supervised stochastic neighbor embedding (SNE) [8] or t-SNE [19] to constrain the within-class distances for further improving low dimensional embedding.

The method developed in this paper is general and can be readily applied to text data with bag-of-words or TF-IDF representation for document classification and clustering, and it can also be applied to gene expression micro-array data for gene function prediction and gene module identification.

Acknowledgement

We thank Geoff Hinton for his guidance and inspiration. We thank Lee Zamparo for proofreading the manuscript and Ke Jin for drawing Figure 1 and Figure 2. We thank Amir Globerson for reading and discussing the first version of this paper.

References

[1] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 153–160. MIT Press, 2007.

[2] M. A. Carreira-Perpignan and H. G. E. On contrastive divergence learning. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 10, 2005.

[3] S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 1:539–546, 2005.

[4] D. DeCoste and B. Schölkopf. Training invariant support vector machines. *Machine Learning*, 46:161–190, 2002.

[5] A. Globerson and S. Roweis. Metric learning by collapsing classes. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *NIPS 18*, pages 451–458. MIT Press, Cambridge, MA, 2006.

[6] J. Goldberger, S. Roweis, G. Hinton, and R. Salakhutdinov. Neighbourhood components analysis. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *NIPS 17*, pages 513–520. MIT Press, Cambridge, MA, 2005.

[7] T. Hastie and R. Tibshirani. Discriminant adaptive nearest neighbor classification. *IEEE Trans. Pattern Anal. Mach. Intell.*, 18(6):607–616, 1996.

[8] G. Hinton and S. Roweis. Stochastic neighbor embedding. In *Advances in Neural Information Processing Systems 15*, pages 833–840. MIT Press.

[9] G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Comp.*, 14(8):1771–1800, August 2002.

[10] G. E. Hinton. To recognize shapes, first learn to generate images. *Progress in brain research*, 165:535–547, 2007.

[11] G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, 2006.

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

[13] J. B. K. Weinberger and L. Saul. Distance metric learning for large margin nearest neighbor classification. In Y. Weiss and B. Sch editors, *NIPS 18*.

[14] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. *ICML2007*, pages 473–480, 2007.

[15] R. Min. A non-linear dimensionality reduction method for improving nearest neighbour classification. *Thesis, DCS, University of Toronto*, 2005.

[16] R. Salakhutdinov and G. Hinton. Learning a nonlinear embedding by preserving class neighbourhood structure. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 11, 2007.

[17] Y. W. Teh and G. E. Hinton. Rate-coded restricted boltzmann machines for face recognition. In *NIPS*, pages 908–914, 2000.

[18] L. Torresani and K. chih Lee. Large margin component analysis. In B. Sch editor, *NIPS 19*. MIT Press.

[19] L. van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, November 2008.

[20] H. Wang and S. Bengio. The mnist database of handwritten upper-case letters. IDIAP-Com 04, IDIAP, 2002.

[21] J. Weston, F. Ratle, and R. Collobert. Deep learning via semi-supervised embedding, 2008.

[22] E. P. Xing, A. Y. Ng, M. I. Jordan, and S. Russell. Distance metric learning with application to clustering with side-information. In S. T. S. Becker and K. Obermayer, editors, *NIPS 15*. MIT Press.