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

Renqiang (Martin) Min

Joint work with

David Stanley, Zineng Yuan, Anthony Bonner, and Zhaolei Zhang

Department of Computer Science

University of Toronto

Dec 7th, 2009

kNN Classification with Metric Learning

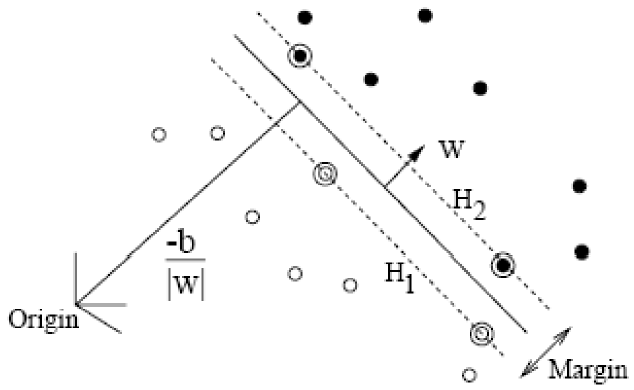
- KNN is popular in almost all fields of data analysis: simple and effective
- When kNN fails:
  - A lot of class-irrelevant features present
  - Bad distance metric adopted
- Distance Metric Learning required for good performance
  - Linear feature transformation is widely used, but it is incapable of capturing higher-order statistics hidden in input feature vector components
  - Non-Linear feature transformation using a kernel trick has also been tried, but it is not scalable to large datasets
  - Neural Networks has also been used to learn non-linear mappings to improve kNN classification, but the neural networks used are often shallow.

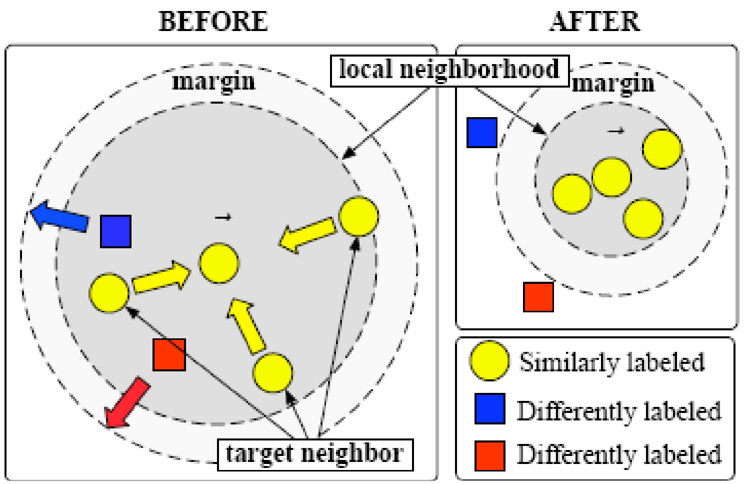
Why deep and non-linear feature mapping

- The difference between statistics, machine learning, and data mining:
  - Statistics prefer simple models to distinguish data with simple structure from noise
  - The task of machine learning and data mining, especially machine learning, is to extract a huge amount of meaningful structure from data, which can often only be represented by complicated model
- Models with shallow architectures fail to represent complex structure hidden in input data:
  - For e.g., perceptron, kernel SVM, neural network with one hidden layer
- Models with deep architectures mimic human brains to perform multi-stage information processing to extract meaningful structure from high-dimensional sensory input
  - Humans can easily recognize shapes and objects and easily extract gist information from complex scenes because human brains has a deep architecture
  - Deep non-linear mapping has many layers, each layer models the combination of patterns in the layer below
  - Researchers often use Neural Networks to construct deep non-linear mapping

Large Margin Learning

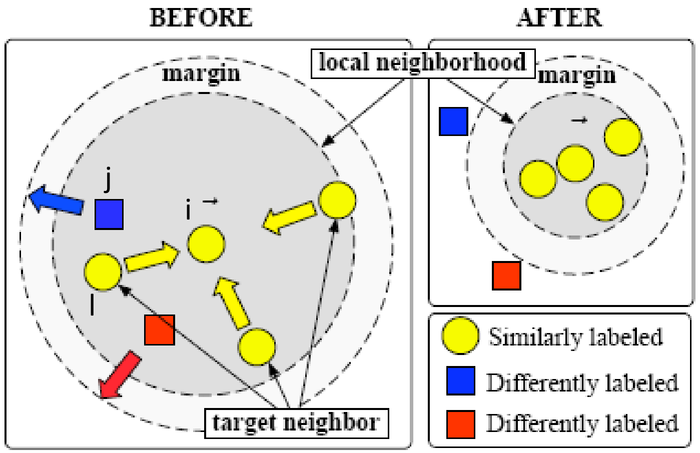
- In 1990s, many researchers abandoned neural networks and turned to use SVMs
- Maximizing margin enables robust classifiers to be learned
  - Linear SVM and Kernel SVM
  - Linear metric learning toward the goal of large-margin separation in the kNN classification framework (Weinberger, NIPS 2005)
  - Limited due to shallow architecture and linear mapping used





The Next-Generation Machine Learning Models: Large Margin Learning with Deep Architectures

- We want to learn a powerful model with deep architecture and at the same time we want it to be robust in the sense of large margin classification
- Our approach:
  - Learn a deep neural network (a deep encoder or auto-encoder)
  - Maintain large-margin classification boundaries in the learned feature (code) space
  - We chose kNN as the classification method to be used in the code space
- Objective function:



$y_{ij} = 1$ to represent that i and j are in the same class

$\eta_{ij} \in \{0,1\}$ to

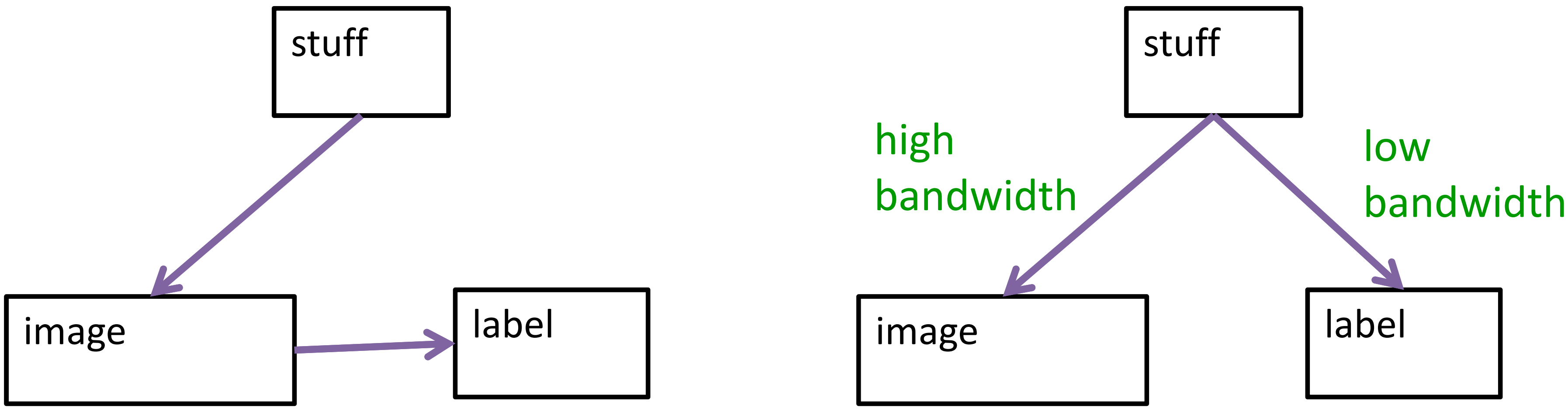
indicate whether input $\vec{x}_j$ is a target neighbor of input $\vec{x}_i$.

$\gamma_{ij} = 1$ if and only if i is an impostor neighbor of j

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

$$min_f \;\; \ell_f = \sum_{ilj} \eta_{il}\gamma_{ij}\ell_{ilj},$$

How to Learn A Deep Supervised Model by Hinton



- Hypothesis by Geoff Hinton: Recognizing Objects by Generating Objects First (if you want to do computer vision, do computer graphics first)
  - Perform unsupervised learning to learn a good generative model first
  - Then fine-tune the model parameters by minimizing the loss function of the supervised learning model

Learn a Deep Generative Model Using Restricted Boltzmann Machines by Hinton (1)

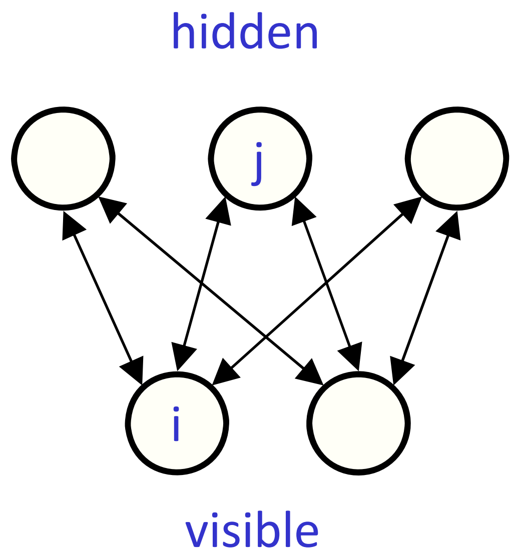
binary state of visible unit i

binary state of hidden unit j

$$E(v,h) = -\sum_{i,j} v_i h_j w_{ij}$$

Energy with configuration v on the visible units and h on the hidden units

weight between units i and j

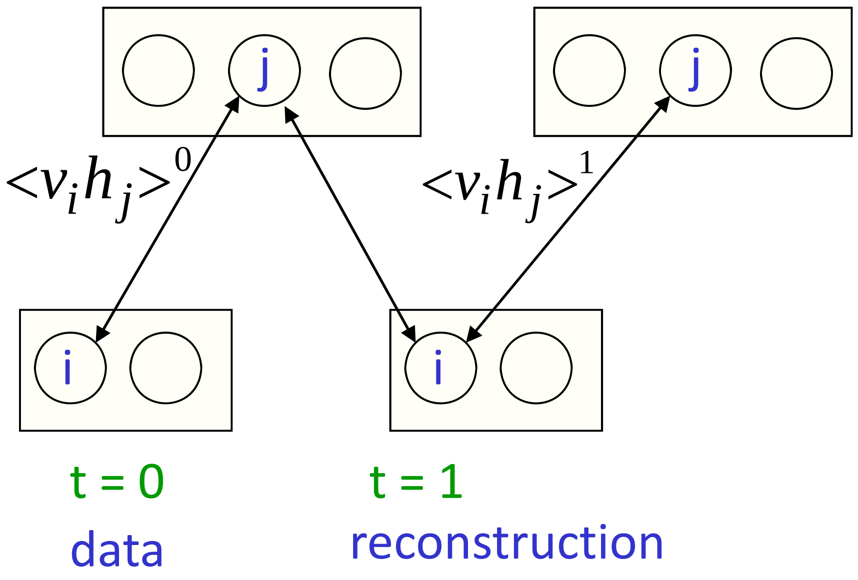


$$p(v,h) \propto e^{-E(v,h)}$$

$$-\frac{\partial E(v,h)}{\partial w_{ij}} = v_i h_j$$

$$\frac{\partial \log p(v)}{\partial w_{ij}} = \langle v_i h_j \rangle^0 - \langle v_i h_j \rangle^\infty$$

Learn a Deep Generative Model Using Restricted Boltzmann Machines by Hinton (2)



Start with a training vector on the visible units.

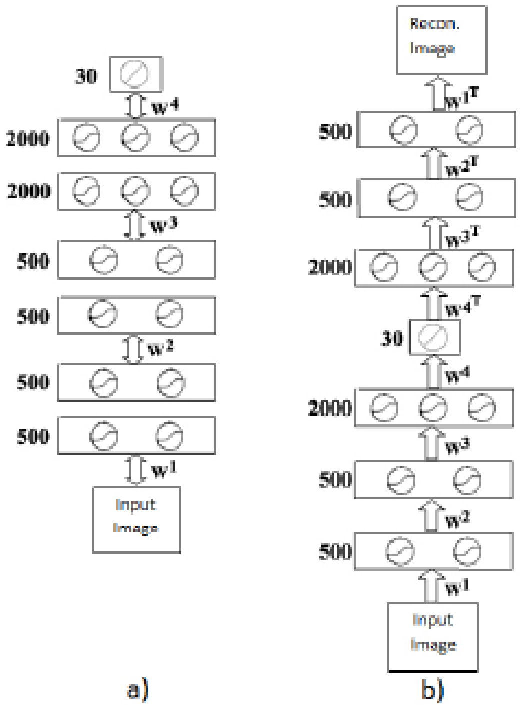
Update all the hidden units in parallel

Update the all the visible units in parallel to get a “reconstruction”.

Update the hidden units again.

$$\Delta w_{ij} = \varepsilon \, (<v_i h_j>^0 - <v_i h_j>^1)$$

Learn a Deep Generative Model Using Restricted Boltzmann Machines (3)



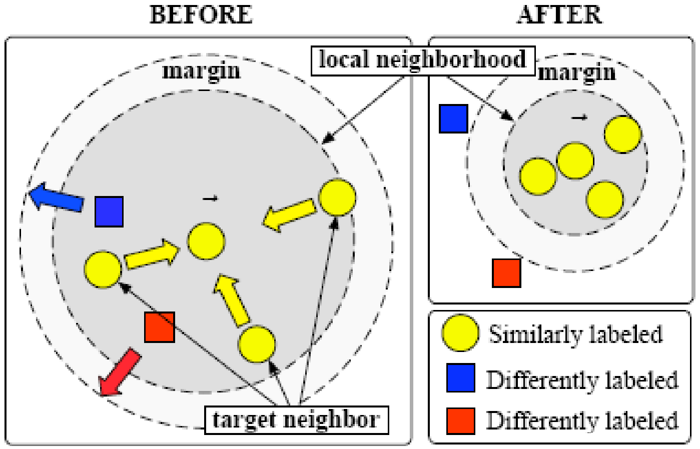
$$E(\mathbf{v},\mathbf{h}) = -\sum_{ij} W_{ij} v_i h_j - \sum_i v_i b_i - \sum_j h_j c_j$$

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

$$\Delta W_{ij} = \epsilon(< v_i h_j >_{data} - < v_i h_j >_1), \;\; binary \;\; \mathbf{h},$$

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

Gradient Calculations of the loss function of Dnet-kNN



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

- For each data point i, create triples (i, l, j), where l is one of i's top k true nearest neighbors, and j is one of i's top m imposter nearest neighbors from every other class than the class of i, m>>k
- Searching on these triples to look for active violated margin constraints

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

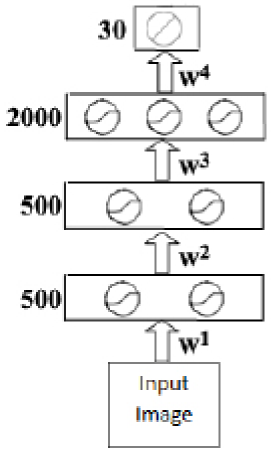
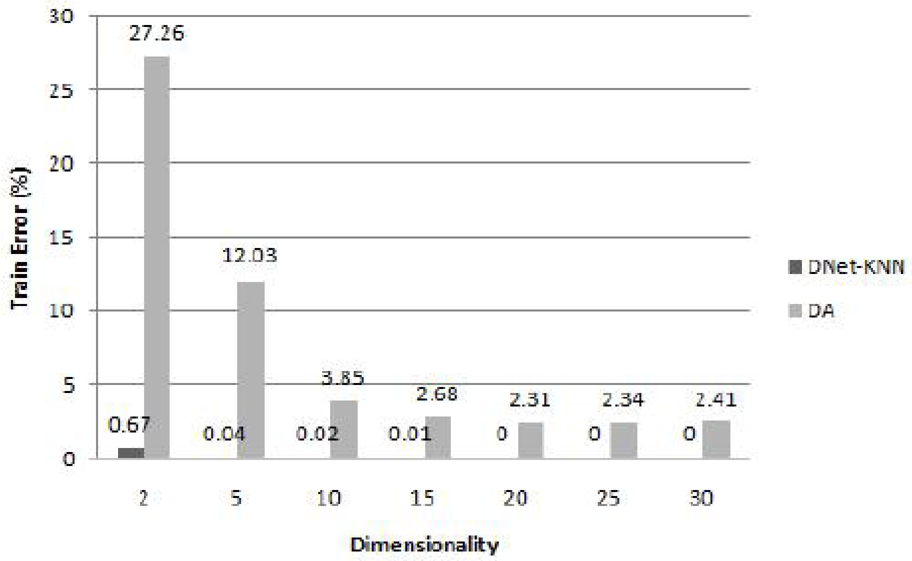


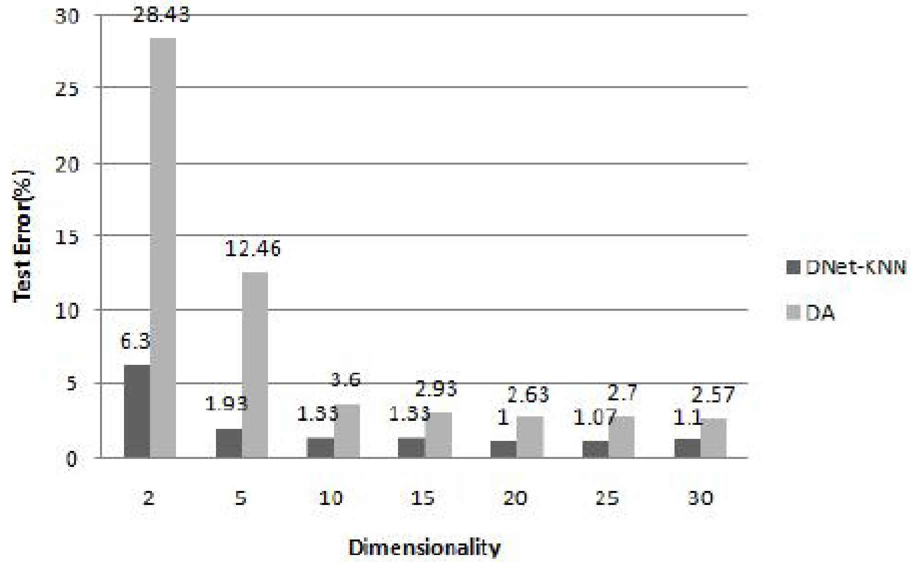
Figure 2. Deep Encoder

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**: $\mathbf{W}$.

Classification Results of Dnet-kNN on USPS Handwritten Digits





	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

	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

Embedding Results of Dnet-kNN on USPS Handwritten Digits

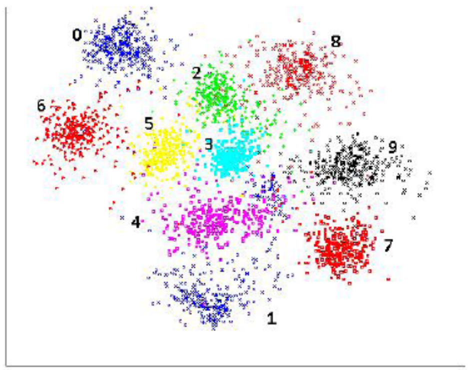


Figure 5. Two-dimensional embedding of 3000 USPS-fixed test data using the Deep Neural Network kNN classifier (DNet-kNN).

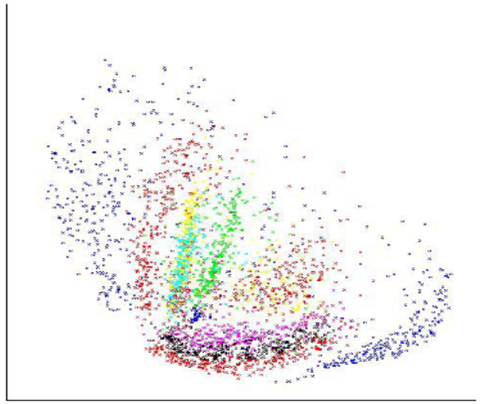


Figure 6. Two-dimensional embedding of 3000 USPS-fixed test data using the Deep Autoencoder (DA).

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

Classification Results of Dnet-kNN on MNIST Handwritten Digits

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

Embedding Results of Dnet-kNN on MNIST Handwritten Digits

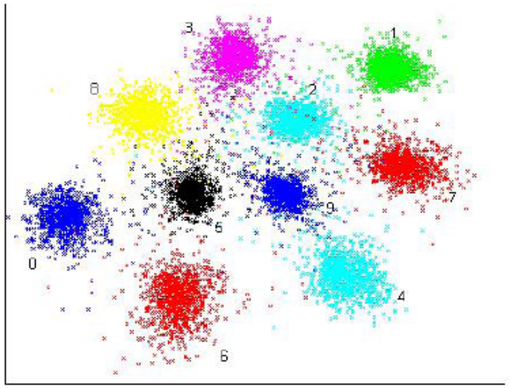


Figure 8. Two-dimensional embedding of 10,000 MNIST test data using the Deep Neural Network kNN classifier (DNet-kNN).

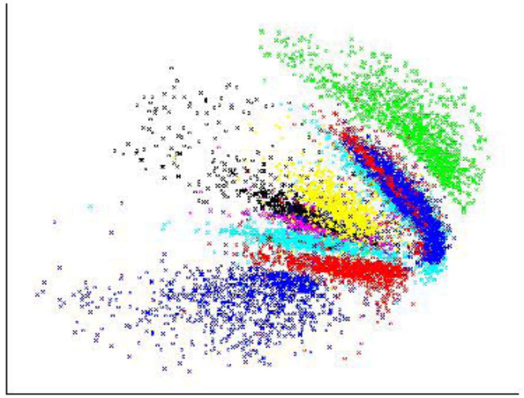


Figure 9. Two-dimensional embedding of 10,000 MNIST test data using the Deep Autoencoder (DA).

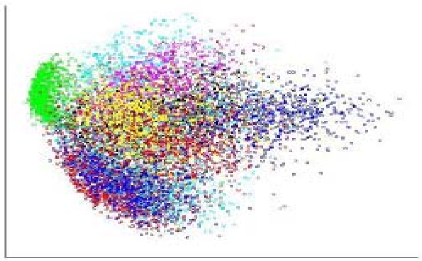


Figure 10. Two-dimensional embedding of 10,000 MNIST test data using PCA.

Classification Results of Dnet-kNN on 20newsgroup Text Data

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

The released implementation of LMNN failed to work on this binary dataset.

Future Work

- We used mini-batch training for Dnet-kNN on large datasets. Instead of fixing imposter nearest neighbors, we can dynamically update imposter nearest neighbors in each dynamically changing mini-batch.
- Instead of training a general deep neural network, pre-training Dnet-kNN using a deep hand-coded convolutional neural network will possibly greatly improve the classification performance (see Lu Cun's research)
- Learn large-margin linear classifiers in the feature space produced by deep (convolutional) neural networks
- Learn a deep network from a Bayesian perspective to constrain the weights to be learned
- Learn the weight matrices in a regularization framework for classification and discuss possible generalization bound for deep learners

Acknowledgement

University of Toronto

Geoff Hinton

Lee Zamparo

Hebrew University

Amir Globerson

Fudan University

Ke Jin