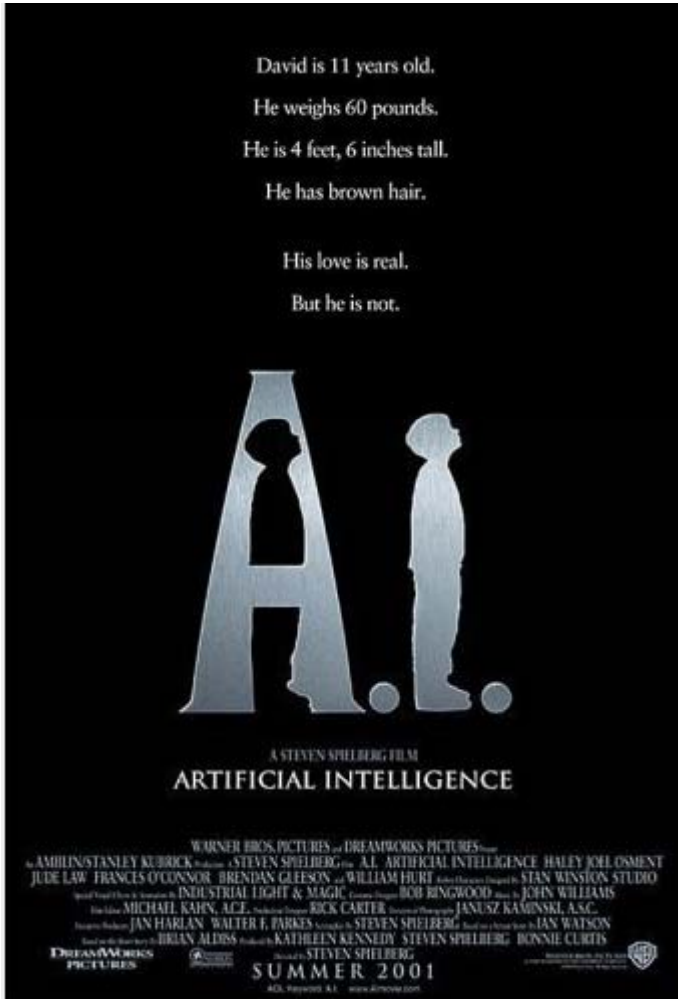
Introduction to Machine Learning

Renqiang (Martin) Min
Molecular Biophysics & Biochemistry
Yale University
And
Machine Learning Group
Department of Computer Science
University of Toronto
Feb 9, 2011

What's Machine Learning

- Make machine learn like a human, and build a machine brain that can adjust itself



A spectrum of machine learning tasks

Typical Statistics----------Artificial Intelligence

- Low-dimensional data (e.g. less than 100 dimensions)
- Lots of noise in the data
- There is not much structure in the data, and what structure there is, can be represented by a fairly simple model.
- The main problem is distinguishing true structure from noise.

- High-dimensional data (e.g. more than 100 dimensions)
- The noise is not sufficient to obscure the structure in the data if we process it right.
- There is a huge amount of structure in the data, but the structure is too complicated to be represented by a simple model.
- The main problem is figuring out a way to represent the complicated structure so that it can be learned.

Machine Learning Topics

- Supervised Learning:

  - Classification: Multinomial Logistic Regression, (Kernel) SVM, Neural Networks, Decision Trees, Random Forest, kNN, Naïve Bayes Classifier

  - Regression: (Non-)Linear Regression, (kernel) SVR, Neural Networks

  - Prediction: Given time series of stock information, predict the price of Google stock in

  - packages in Weka or Matlab or R

- Unsupervised Learning:

  - Generative model: Markov Random Field (Restricted) Boltzmann Machines, Deep Belief Networks, HMMs, Bayesian Networks

  - Density Estimation: Mixture of Gaussians, KDE, Graphical Models

  - Clustering: MofG, k-means, (Bi-)Spectral Clustering

  - Dimensionality Reduction: PCA, ICA, MDS, ISOMAP, Neural Networks

  - Rule Learning

  - Outlier Detection: one-class SVM

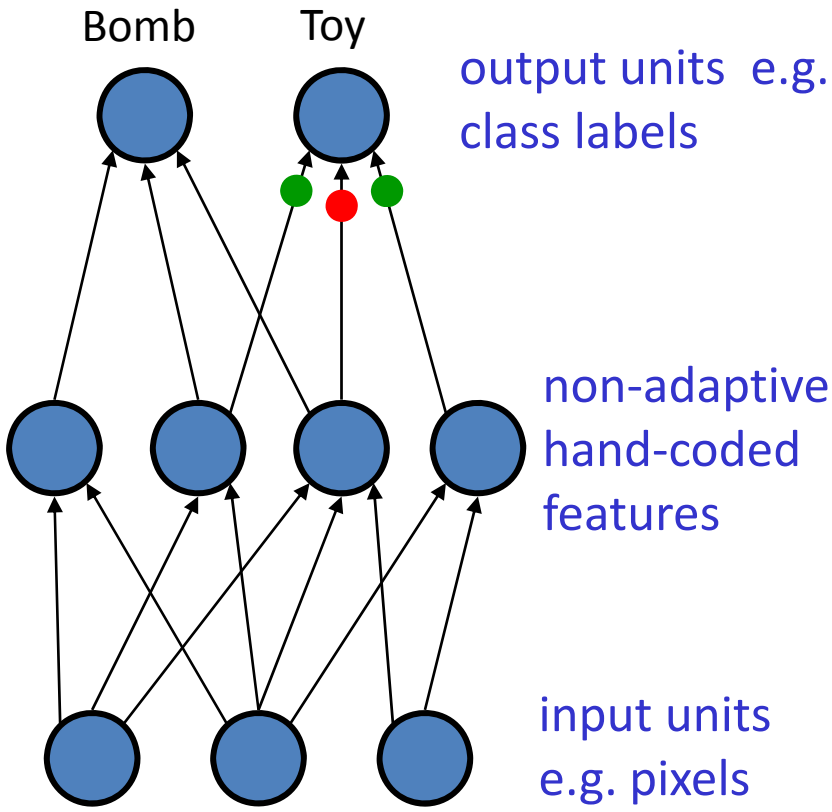
- Reinforcement Learning:
  - Robot planning, Robot control
  - Game playing

Reinforcement Learning

- Given sequences of inputs, a action set, and a reward/punishment scheme, learn a sequence of actions that maximize expected reward
- Demo
- Not many applications in Bioinformatics

Supervised Learning: First-Generation Neural Networks

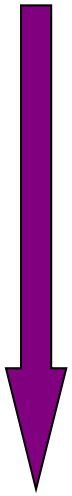
- Perceptrons (~1960) used a layer of hand-coded features and tried to recognize objects by learning how to weight these features.
  - There was a neat learning algorithm for adjusting the weights.
  - But perceptrons are fundamentally limited in what they can learn to do.

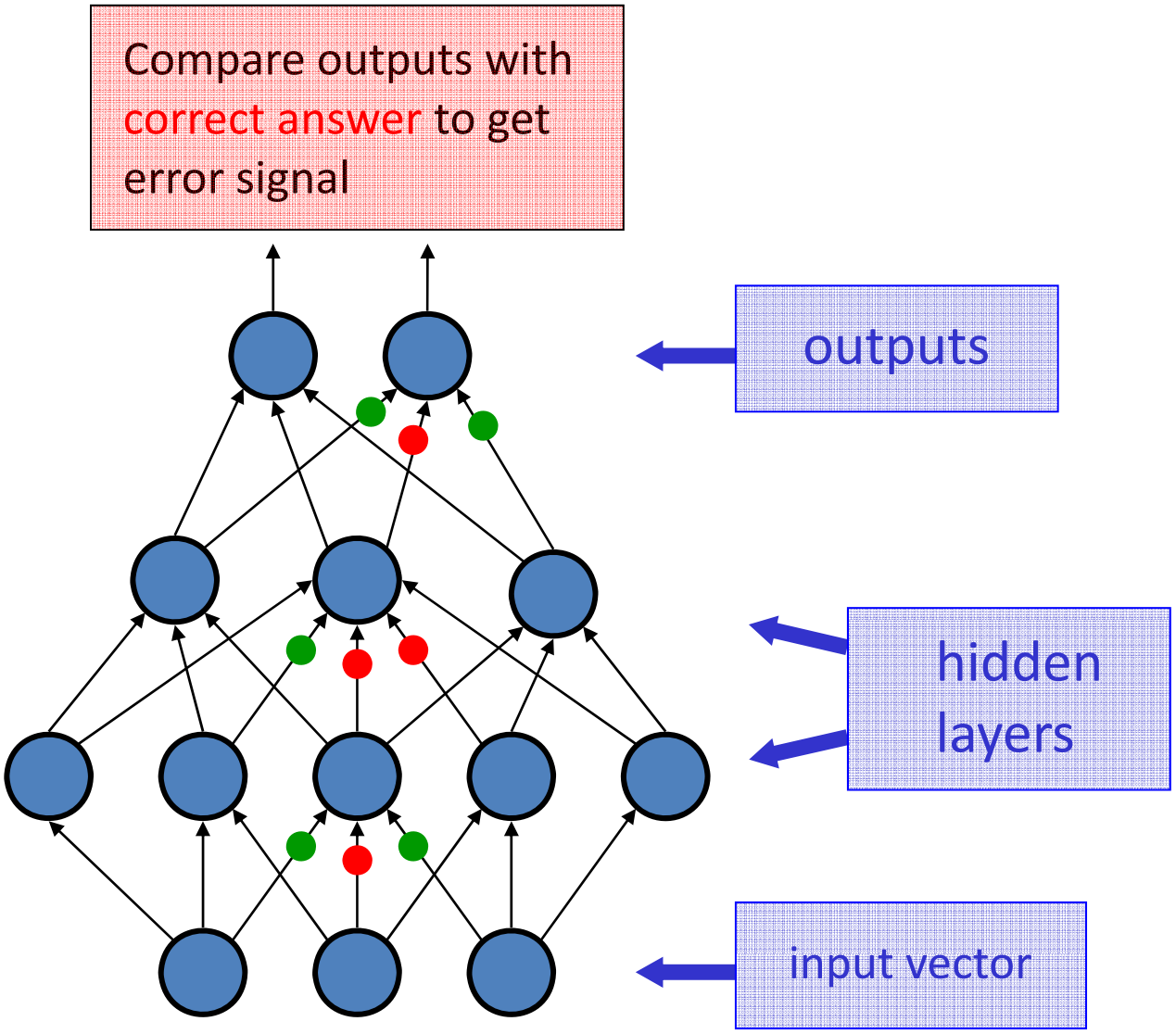


Sketch of a typical perceptron from the 1960's

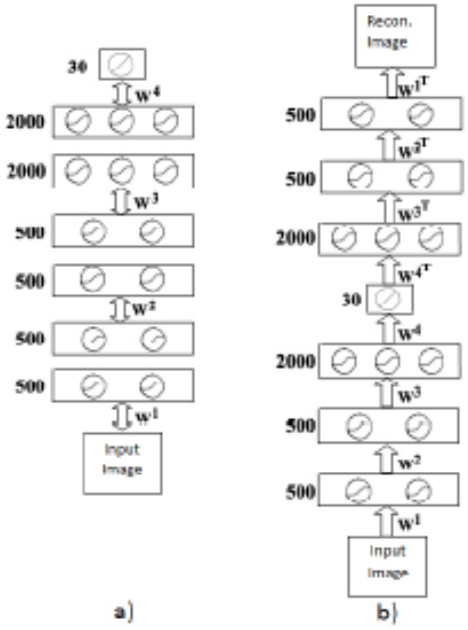
Second-generation neural networks (~1985)

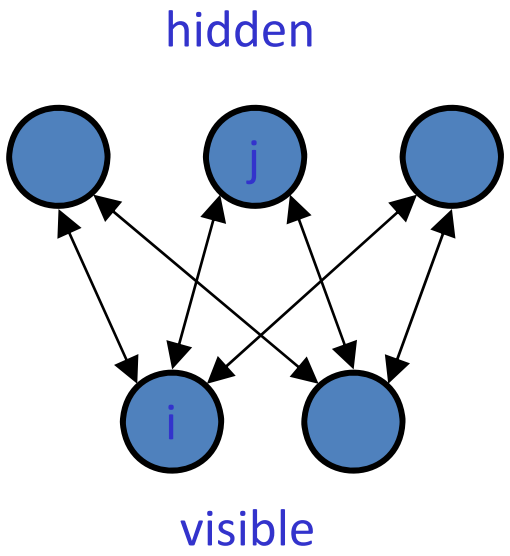
Back-propagate error signal to get derivatives for learning





Third-generation neural networks

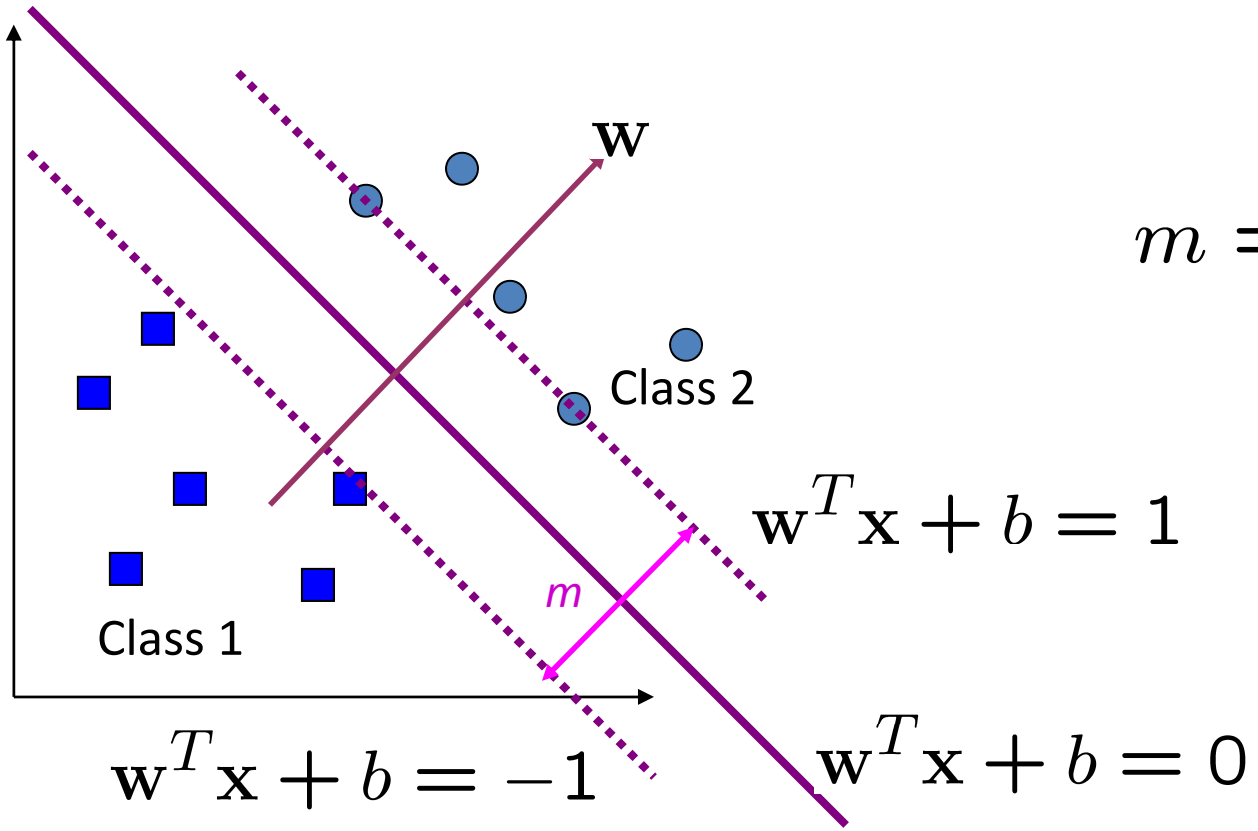




Supervised Learning: SVM

Minimize $\frac{1}{2}||\mathbf{w}||^2$

subject to $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \qquad \forall i$



$$m = \frac{2}{||\mathbf{w}||}$$

Supervised Learning: SVM

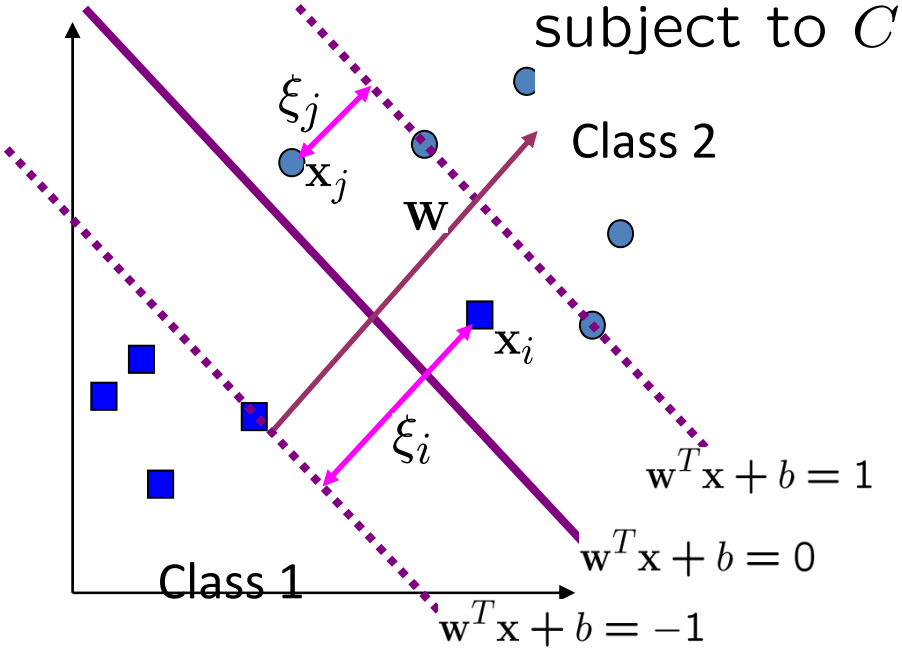
Minimize $\frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\xi_i$

subject to $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$

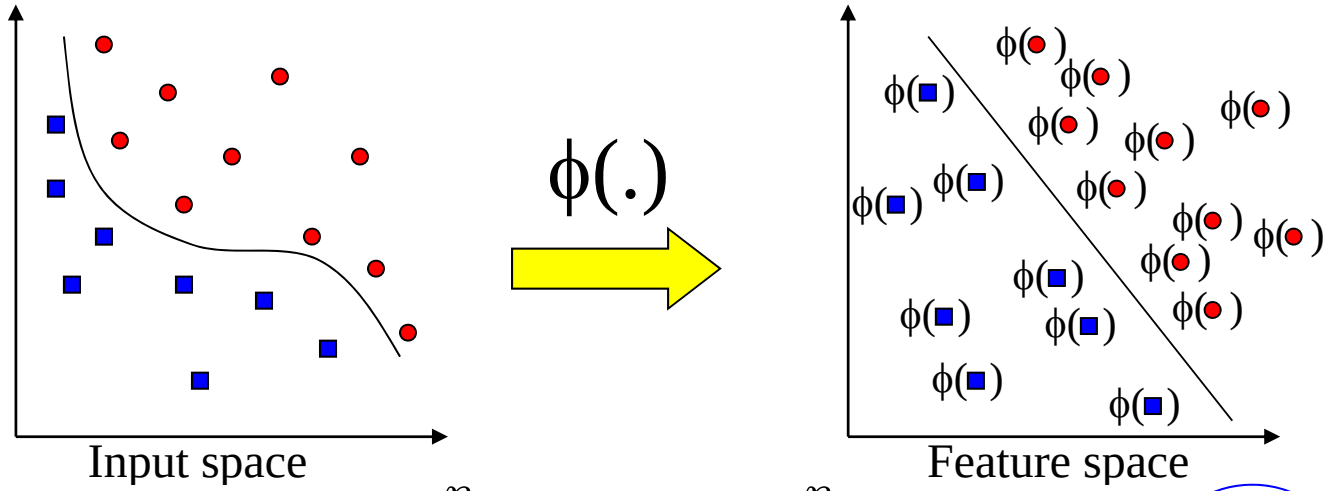
$$\text{max. } W(\boldsymbol{\alpha}) = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1,j=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j$$

$$\text{subject to } C \geq \alpha_i \geq 0, \sum_{i=1}^{n}\alpha_i y_i = 0$$

$$\mathbf{w} = \sum_{j=1}^{s}\alpha_{t_j} y_{t_j}\mathbf{x}_{t_j}$$



Non-linear Mapping



$$\text{max. } W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\text{subject to } C \geq \alpha_i \geq 0, \sum_{i=1}^{n} \alpha_i y_i = 0$$

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

Kernels

- Polynomial kernel with degree *d*

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T\mathbf{y} + 1)^d$$

- Radial basis function kernel with width σ

$$K(\mathbf{x}, \mathbf{y}) = \exp(-||\mathbf{x} - \mathbf{y}||^2/(2\sigma^2))$$

  - Closely related to radial basis function neural networks
  - The feature space is infinite-dimensional

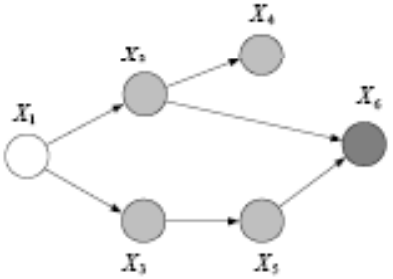
- Sigmoid with parameter κ and θ

$$K(\mathbf{x}, \mathbf{y}) = \tanh(\kappa\mathbf{x}^T\mathbf{y} + \theta)$$

  - It does not satisfy the Mercer condition on all κ and θ

Unsupervised Learning: Graphical Models (can represent any Probabilistic Model) Learning & Inference

EXAMPLE



The key is to factor and then apply the distributive law.

$$p(\mathrm{x}_1|\bar{\mathrm{x}}_6) = p(\mathrm{x}_1, \bar{\mathrm{x}}_6)/p(\bar{\mathrm{x}}_6)$$
$$= p(\mathrm{x}_1, \bar{\mathrm{x}}_6)/\sum_{\mathrm{x}_1'} p(\mathrm{x}_1', \bar{\mathrm{x}}_6)$$

$$p(\mathrm{x}_1, \bar{\mathrm{x}}_6) = \sum_{\mathrm{x}_2}\sum_{\mathrm{x}_3}\sum_{\mathrm{x}_4}\sum_{\mathrm{x}_5} p(\mathrm{x}_1)p(\mathrm{x}_2|\mathrm{x}_1)p(\mathrm{x}_3|\mathrm{x}_1)p(\mathrm{x}_4|\mathrm{x}_2)p(\mathrm{x}_5|\mathrm{x}_3)p(\bar{\mathrm{x}}_6|\mathrm{x}_2, \mathrm{x}_5)$$
$$= p(\mathrm{x}_1)\sum_{\mathrm{x}_2} p(\mathrm{x}_2|\mathrm{x}_1)\sum_{\mathrm{x}_3} p(\mathrm{x}_3|\mathrm{x}_1)\sum_{\mathrm{x}_4} p(\mathrm{x}_4|\mathrm{x}_2)\sum_{\mathrm{x}_5} p(\mathrm{x}_5|\mathrm{x}_3)p(\bar{\mathrm{x}}_6|\mathrm{x}_2, \mathrm{x}_5)$$
$$= p(\mathrm{x}_1)\sum_{\mathrm{x}_2} p(\mathrm{x}_2|\mathrm{x}_1)\sum_{\mathrm{x}_3} p(\mathrm{x}_3|\mathrm{x}_1)\Phi_5(\mathrm{x}_2, \mathrm{x}_3)\sum_{\mathrm{x}_4} p(\mathrm{x}_4|\mathrm{x}_2)$$
$$= p(\mathrm{x}_1)\sum_{\mathrm{x}_2} p(\mathrm{x}_2|\mathrm{x}_1)\Phi_4(\mathrm{x}_2)\sum_{\mathrm{x}_3} p(\mathrm{x}_3|\mathrm{x}_1)\Phi_5(\mathrm{x}_2, \mathrm{x}_3)$$
$$= p(\mathrm{x}_1)\sum_{\mathrm{x}_2} p(\mathrm{x}_2|\mathrm{x}_1)\Phi_4(\mathrm{x}_2)\Phi_3(\mathrm{x}_1, \mathrm{x}_2)$$
$$= p(\mathrm{x}_1)\Phi_2(\mathrm{x}_1)$$

Trees

- Examples: HMMs, Chow-Liu Trees
- Efficient Belief Propagation algorithms

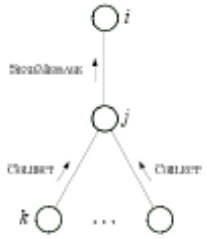
Belief Propagation (Sum-Product) Algorithm

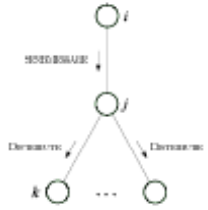
- Choose a root node arbitrarily.
- If j is an evidence node, $\psi^E(x_j) = \delta(x_j, \bar{x}_j)$, else $\psi^E(x_j) = 1$.
- Pass messages from leaves up to root and then back down using:

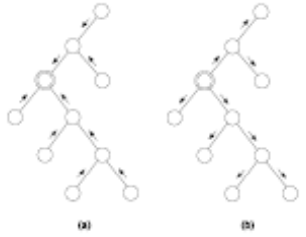
$$m_{ji}(x_i) = \sum_{x_j} \left(\psi^E(x_j)\psi(x_i, x_j) \prod_{k \in c(j)} m_{kj}(x_j) \right)$$

- Compute node marginals using the product of incoming messages:

$$p(x_i|\bar{\mathbf{x}}_E) \propto \psi^E(x_i) \prod_{k \in c(i)} m_{ki}(x_i)$$

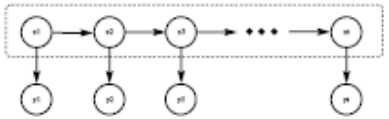




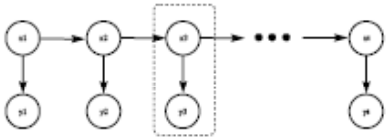


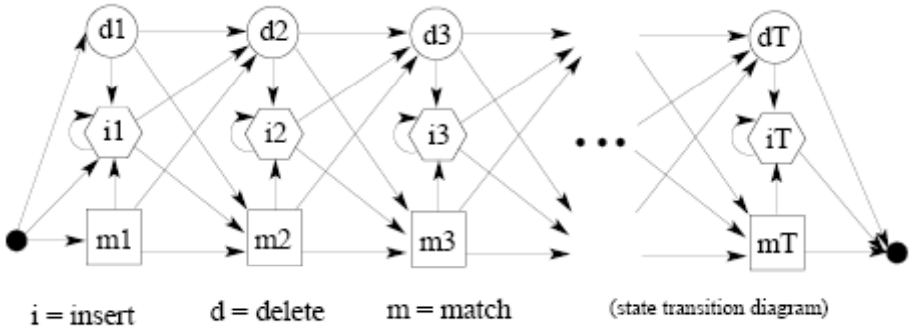
Hidden Markov Models

- You can think of an HMM as:
  A Markov chain with stochastic measurements.



or
A mixture model with states coupled across time.





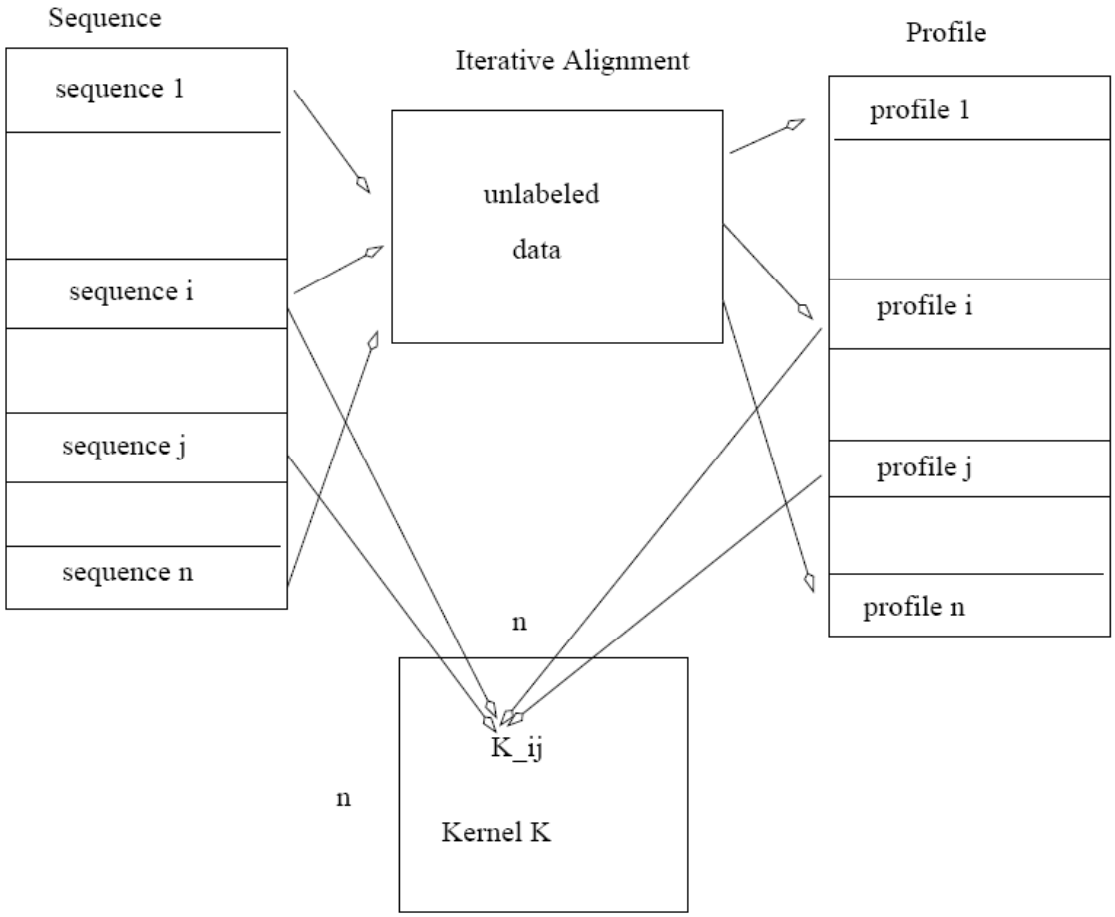
- The future is independent of the past given the present.
  However, conditioning on all the observations couples hidden states.

- Speech recognition.
- Language modeling.
- Information retrieval.
- Motion video analysis/tracking.
- Protein sequence and genetic sequence alignment and analysis.
- Financial time series prediction.

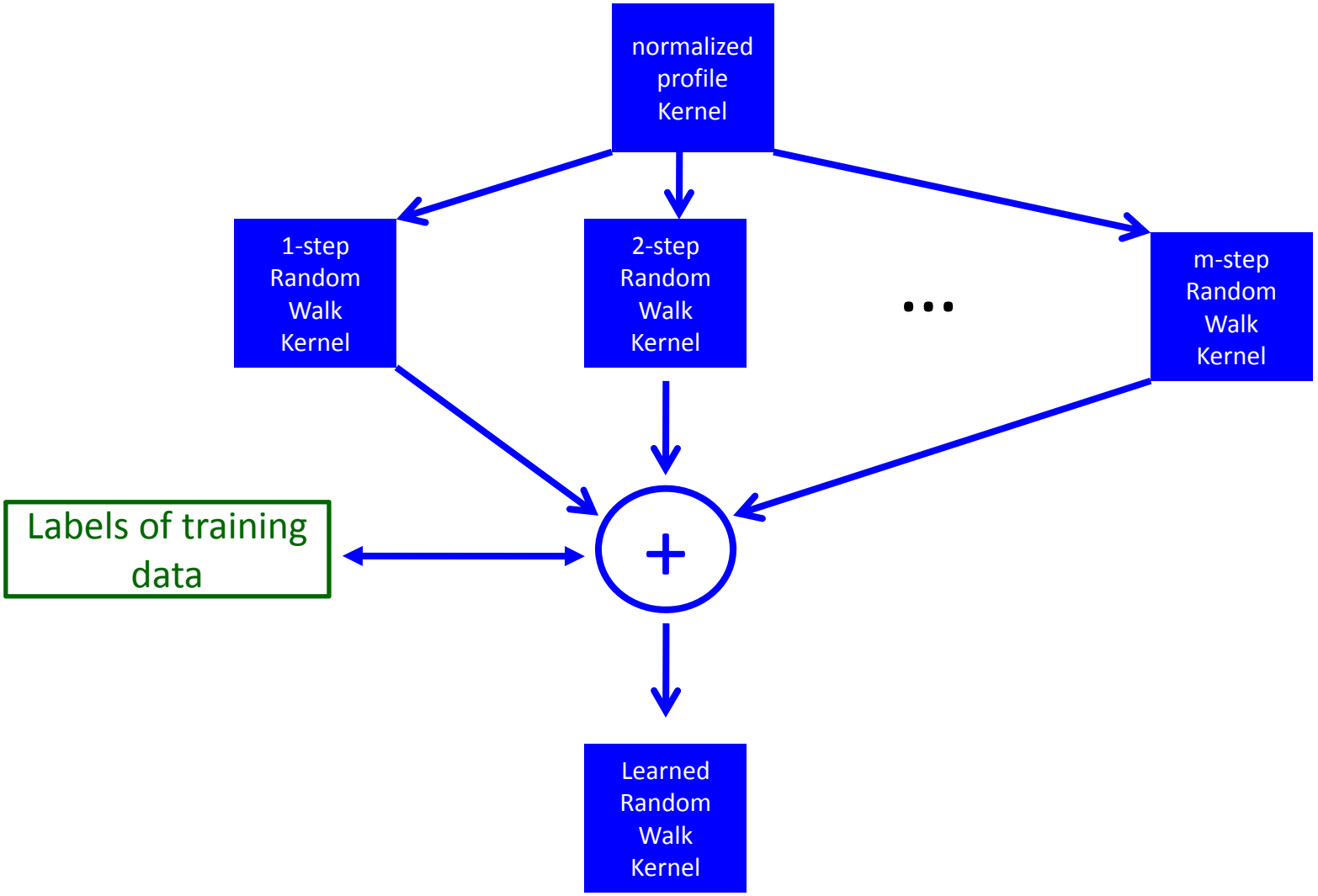
Kevin Murphy's BayesNet Package and Matlab Package or code it yourself (easy to implement, lots of open source code online)

Kernel Methods for Classifying Proteins to SCOP Super-families

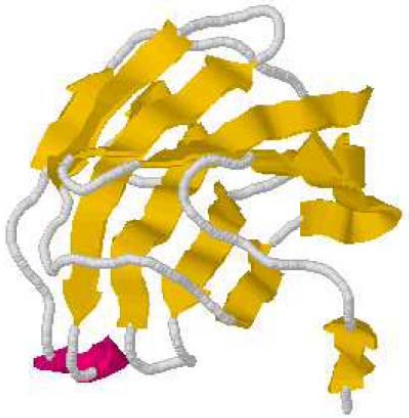
- Represent each protein sequence by a PSWM

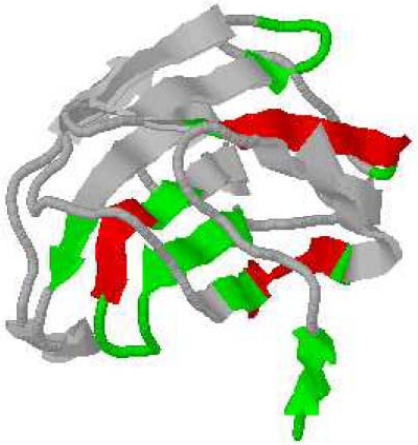


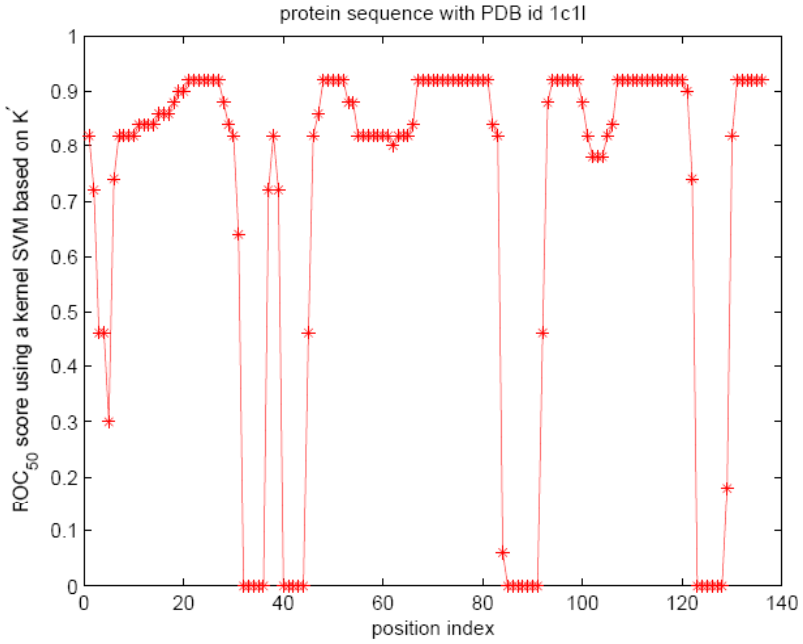
Learned Random-Walk Kernel

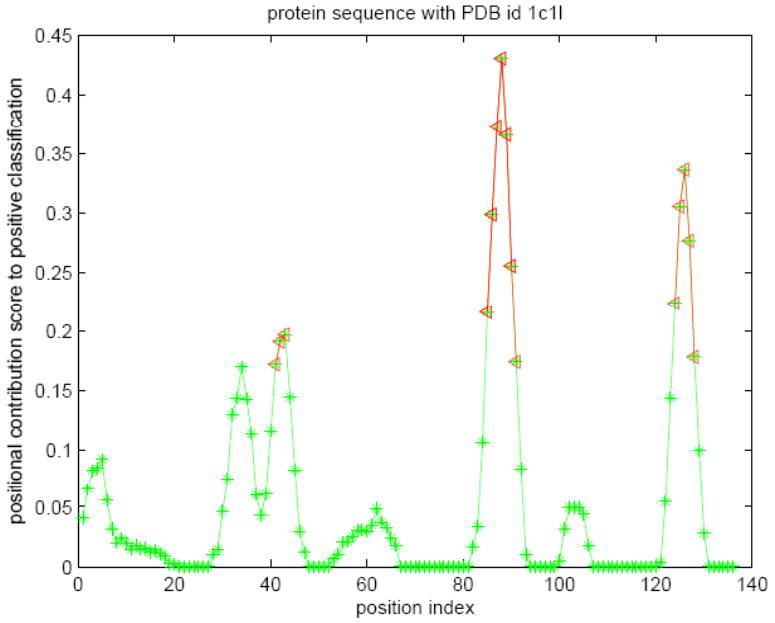


Discovered Protein Sequence Motifs (PDB id 1c1l)







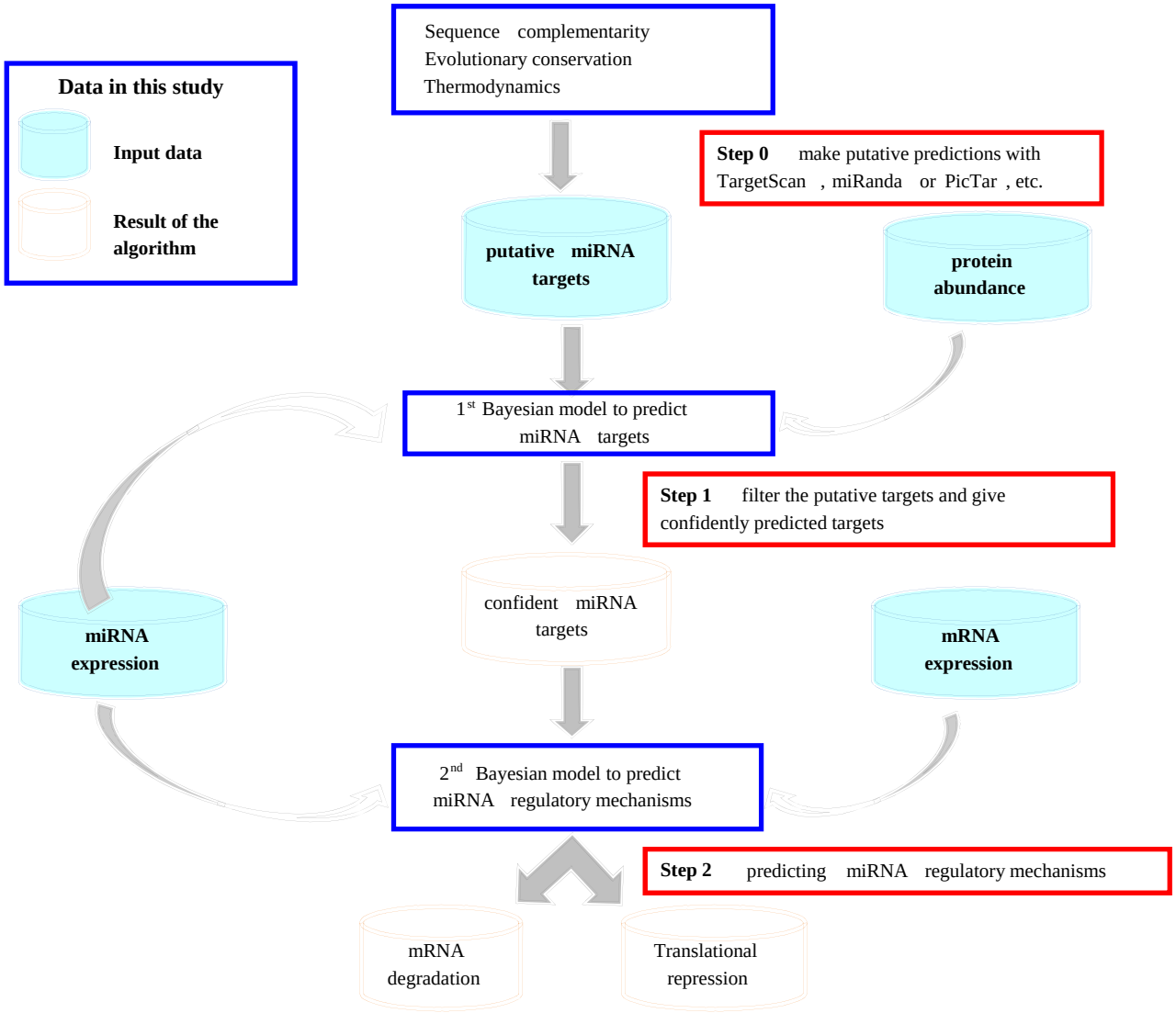


Experimental Results on Protein Remote Homology Identification

Glutathione S-transferases, N-terminal domain

Methods	ROC_{50} on the hardest protein family
eMOTIF (see reference [52] and [36])	0.000
SVM-pairwise [PSI-BLAST] (see reference [42] and [36])	0.000
spectrum-kernel [PSI-BLAST] (see reference [38])	0.000
neighborhood (see reference [70])	0.000
the second best profile kernel (**the second best result**)	0.045
the best profile kernel (**the best result**)	0.122
improved RWK using the second best profile kernel	0.454
empirical-map kernel using the second best profile kernel	0.455
improved RWK using the best profile kernel	0.509
empirical-map kernel using the best profile kernel	0.903
HMMER	0.000
SAM	0.000

Graphical Models for MiRNA Regulation Inference



Four types of experimental data are taken as input: (1) a set of putative miRNA targets, (2) protein abundance, (3) miRNA expression profiles, and (4) mRNA expression profiles.

20

Models – WinBUGS or Matlab or C++

$$p(k|r,\gamma) = \binom{k+r-1}{r-1}\gamma^r(1-\gamma)^k$$
$$= \frac{\Gamma(k+r)}{k!\Gamma(r)}\gamma^r(1-\gamma)^k.$$

$$\lambda = r\frac{1-\gamma}{\gamma}$$

$$p(k|r,\gamma) = NB(k|\lambda,r)$$
$$= \frac{\lambda^k}{k!}\frac{\Gamma(r+k)}{\Gamma(r)(r+\lambda)^k}\frac{1}{1+\frac{\lambda}{r}}.$$

r approaches infinity,

$$p \sim beta(1,1),$$
$$\tau_t \sim uniform(0,50),$$
$$\rho_t \sim gamma(\alpha,\alpha),$$
$$\alpha \sim uniform(0,+\infty),$$
$$\omega_j \sim exponential(\beta),$$
$$\beta \sim uniform(0,1000),$$
$$r_t \sim exponential(a),$$
$$a \sim uniform(0,1000),$$

$$p(b_{ij}=1|\delta_{ij}=1) = h,$$
$$p(b_{ij}=1|\delta_{ij}=0) = 0,$$

$$\Phi_j \sim exponential(\Psi),$$
$$\Psi \sim uniform(0,+\infty),$$
$$c_t \sim uniform(-50,+50),$$
$$h \sim beta(1,1).$$

$NB(k|\lambda,r)$ approaches a Possion distribution with mean parameter λ.

$$p(W_{it}=k|\theta_{it},r_t) = NB(k|\theta_{it},r_t).$$

$$ln(\theta_{it}) = ln(\tau_t) - \rho_t\sum_{j=1}^{J}\omega_j\delta_{ij}M_{jt}$$

$$P(R_{it}=k) = q_{it}^{(1-k)}(1-q_{it})^k, k=0 \ \ or \ \ 1.$$

$$logit(q_{it}) = log(\frac{q_{it}}{1-q_{it}}) = \sum_{j=1}^{H}\Phi_j b_{ij}M_{jt} + c_t$$

b_{ij} is a binary latent variable

Classification and Dimensionality Reduction by Deep Neural Networks

- Two-D embedding for data visualization
- 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
  - 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

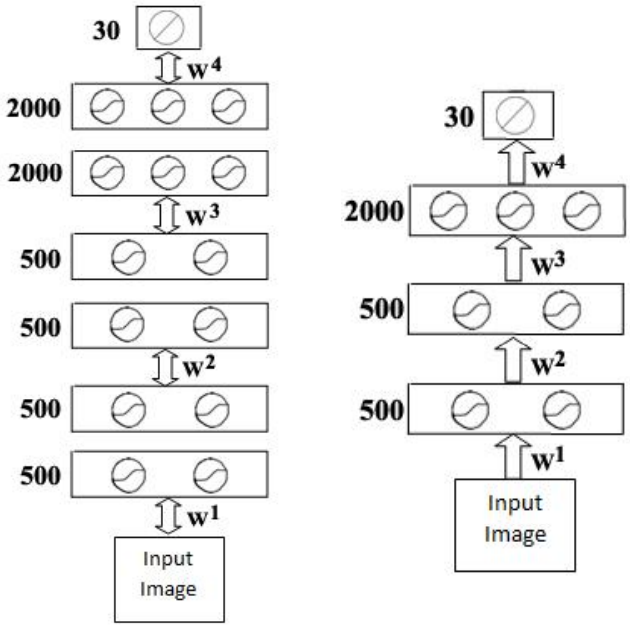
- Models with shallow architectures fail to represent complex structure hidden in input data:
  - For e.g., perceptron, kernel SVM, neural network with one hidden layer
- Models with deep architectures mimic human brains to perform multi-stage information processing to extract meaningful structure from 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

Deep Neural Networks

- Deep neural networks pre-trained with RBMs are capable of generating powerful non-linear embeddings
  - Deep neural networks are good at extracting meaningful structure from high-dimensional input features

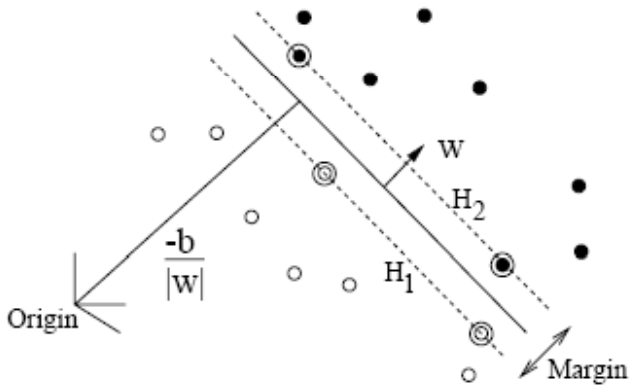
Extend Supervised Linear Embedding Methods with Deep Neural Networks

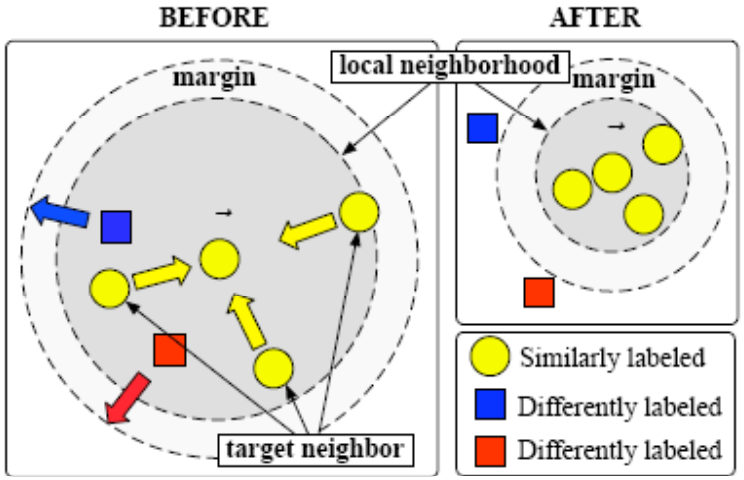
- Maximize Margin for kNN classification (LMNN)
- Maximally Collapsing Metric Learning (MCML) learns a linear mapping to collapse all the points in the same class to one point
- Neighborhood Component Analysis (NCA) learns a linear mapping by maximizing the expected number of points correctly classified
- We can use a deep neural network pre-trained with RBMs to learn a deep supervised non-linear embedding by optimizing the cost of LMNN, MCML, and NCA for both high-dimensional data visualization and classification



Large Margin Learning

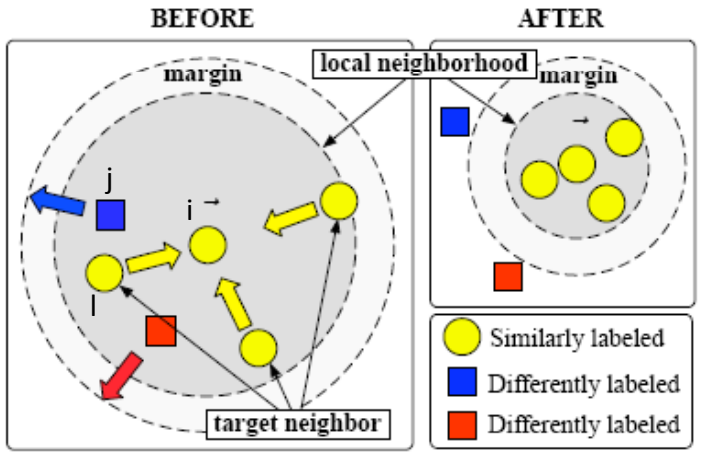
- In 1990s, many researchers abandoned neural networks and turned to use SVMs
- Maximizing margin enables robust classifiers to be learned
  - Linear SVM and Kernel SVM
  - Linear metric learning toward the goal of large-margin separation in the kNN classification framework (Weinberger, NIPS 2005)
 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





DNet-kNN: 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

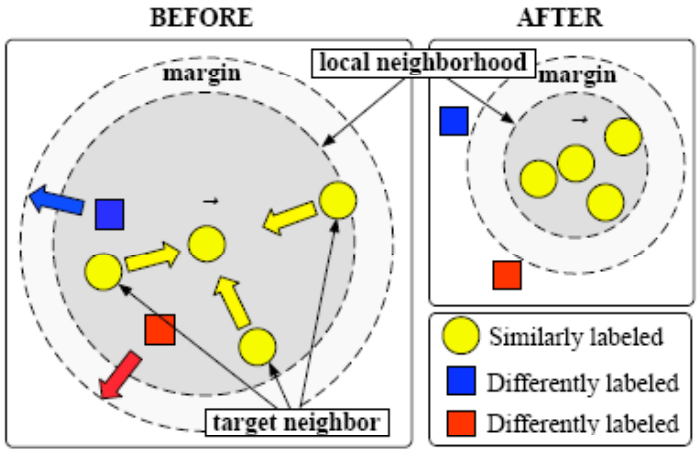
$\eta_{ij} \in \{0, 1\}$ to indicate whether input $\vec{x}_j$ is a target neighbor of input $\vec{x}_i$.

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

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

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

Gradient Calculations of the loss function of Dnet-kNN



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

- 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

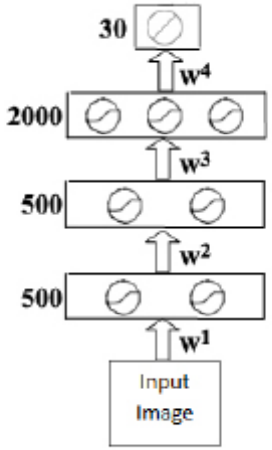


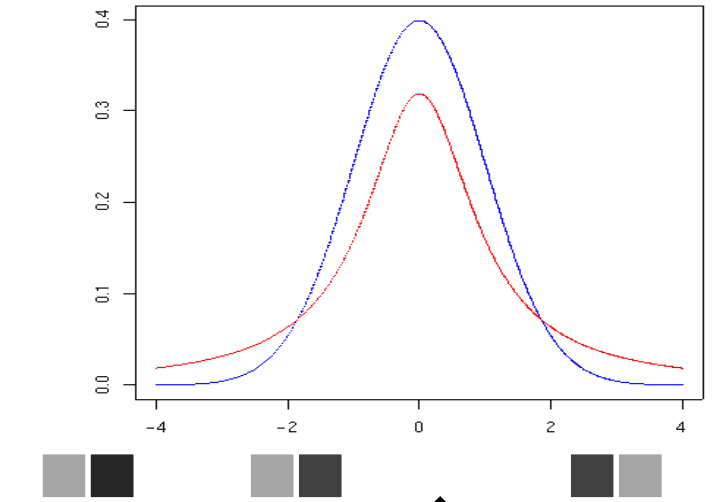
Figure 2. Deep Encoder

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**: $\mathbf{W}$.

Supervised Peaky and Multimodal Class Collapsing

- Make similar data points in the same class stay close together
- Allow dissimilar data points in the same class to be put far apart in the embedding space
- Different classes of data should be put even further apart



2D embedding

dt-MCML and dt-NCA

- Using a t-distribution for modeling conditional probabilities in the embedded space, dt-MCML collapses classes while dt-NCA maximizes the expected number of points correctly classified

- Collapsing classes works well for very low-dimensional embedding such as two-d embedding, but is unnecessary and might cause overfitting when the dimensionality of the embedded space is large

- dt-NCA is more suitable for higher-dimensional embedding than dt-MCML

dt-MCML

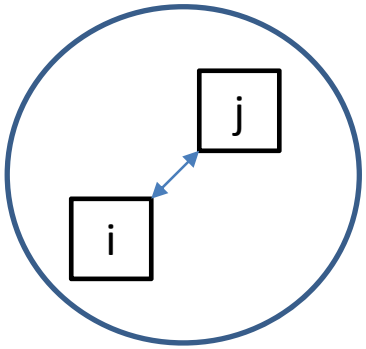
- Unlike in MCML, we use symmetric q distribution to simplify gradient computation:

$$\ell_{dt-MCML} = KL(P||Q) = \sum_i \sum_{j:j\neq i} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

$$p_{ij} \propto 1 \text{ iff } y^{(i)} = y^{(j)},\ p_{ij} = 0 \text{ iff } y^{(i)} \neq y^{(j)} \qquad \sum_{ij} p_{ij} = 1$$

$$q_{ij} = \frac{(1 + d_{ij}^2/\alpha)^{-\frac{1+\alpha}{2}}}{\sum_{kl:k\neq l}(1 + d_{kl}^2/\alpha)^{-\frac{1+\alpha}{2}}}, \quad q_{ii} = 0 \qquad d_{ij}^2 = ||f(\mathbf{x}^{(i)}) - f(\mathbf{x}^{(j)})||^2$$

- This objective function is equivalent to the negative log product of q_{ij}s



- Prevent data points in the same class spreadout

dt-NCA

$$\ell_{dt-NCA} = -\sum_{ij:i\neq j} \delta_{ij} q_{j|i},$$

δ_{ij} equals 1 if $y^{(i)} = y^{(j)}$ and 0 otherwise

$$q_{j|i} = \frac{(1 + d_{ij}^2/\alpha)^{-\frac{1+\alpha}{2}}}{\sum_{k:k\neq i}(1 + d_{ik}^2/\alpha)^{-\frac{1+\alpha}{2}}}, \quad q_{i|i} = 0.$$

- dt-NCA uses asymmetric q distribution while dt-MCML uses symmetric q distribution

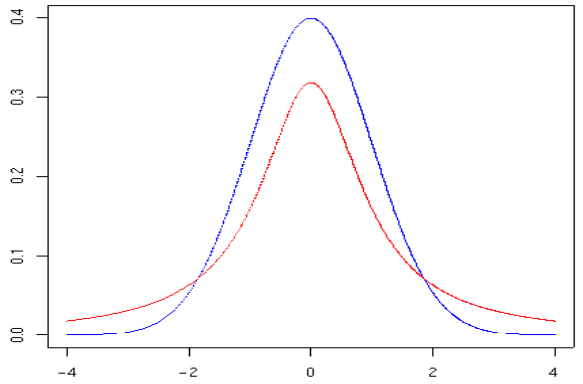
- dt-NCA maximizes the sum of the probabilities q_ij while dt-MCML maximizes the product of the probabilities q_ij

the advantages of using a t-distribution

- In t-SNE, there are no supervision signals, and t-distribution helps to avoid “crowding problem”

- In dt-MCML and dt-NCA:

- allow one class of data to be embedded to different modes



- result in tighter clusters in the embedding

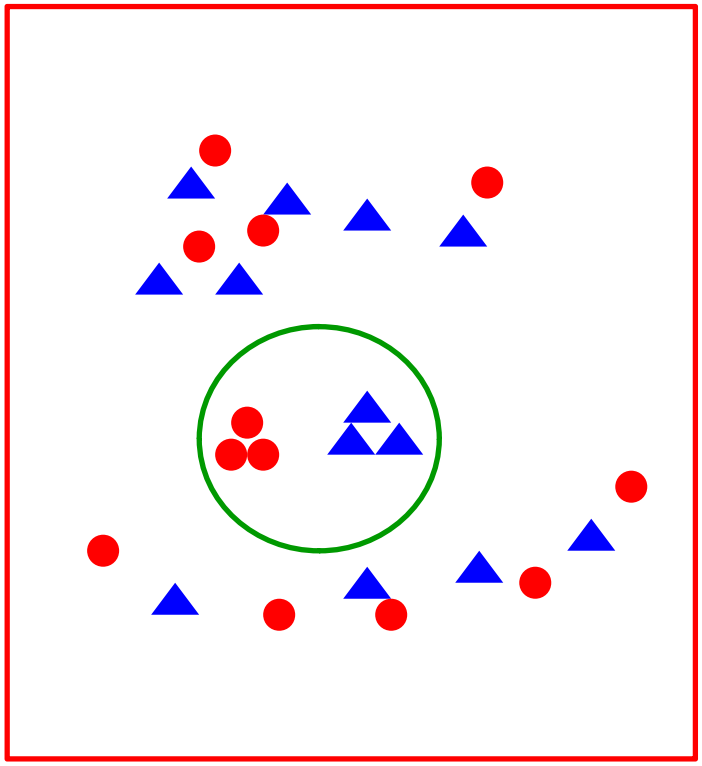
- allow larger separations between classes

- make gradient-based optimization easier: the gradient of the tail of a t-distribution is much deeper than that of a Gaussian

Symmetric / Asymmetric dt-MCML and dt-NCA

dt-MCML
- Symmetric
- Asymmetric

dt-NCA
- Asymmetric
- Symmetric



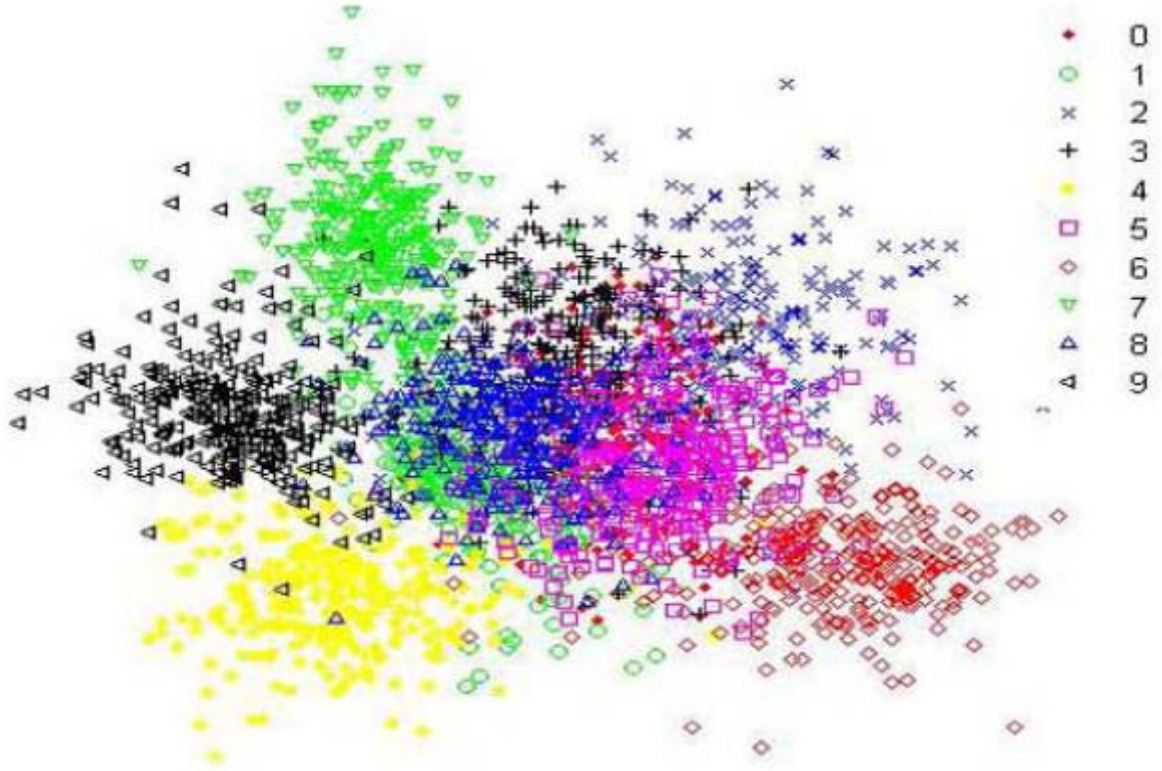
Embedding Results on USPS Digits

Table 1. Mean and standard deviation of test error (in %) on 2-dimensional and 30-dimensional embedding for various techniques on the 6 splits of USPS data set.

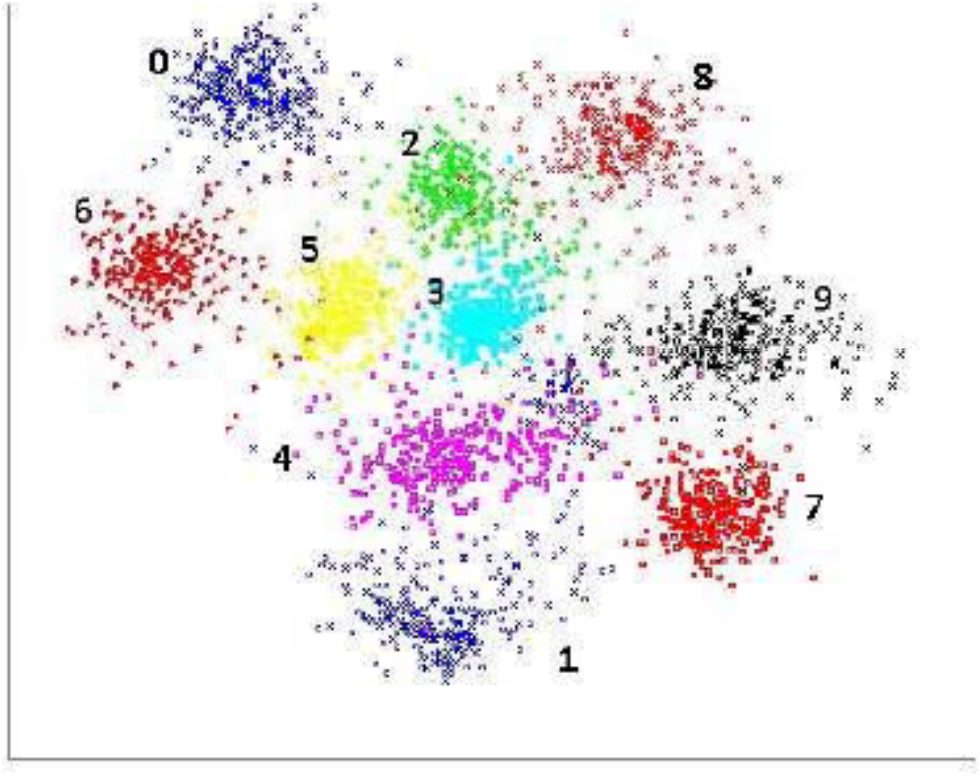
Dimensionality d	**2D**	**30D**
MCML	35.63 ± 0.44	5.53 ± 0.39
dG-MCML	3.37 ± 0.18	1.67 ± 0.21
dt-MCML ($\alpha = d - 1$)	$\mathbf{2.46 \pm 0.35}$	1.73 ± 0.47
dt-MCML (learned α)	2.80 ± 0.36	1.61 ± 0.36
dG-NCA	10.22 ± 0.76	1.91 ± 0.22
dt-NCA ($\alpha = d - 1$)	5.11 ± 0.28	$\mathbf{1.15 \pm 0.21}$
dt-NCA (learned α)	6.69 ± 0.92	1.17 ± 0.07

DNet-kNN	5.40 (0.90)	**1.14 (0.20)**

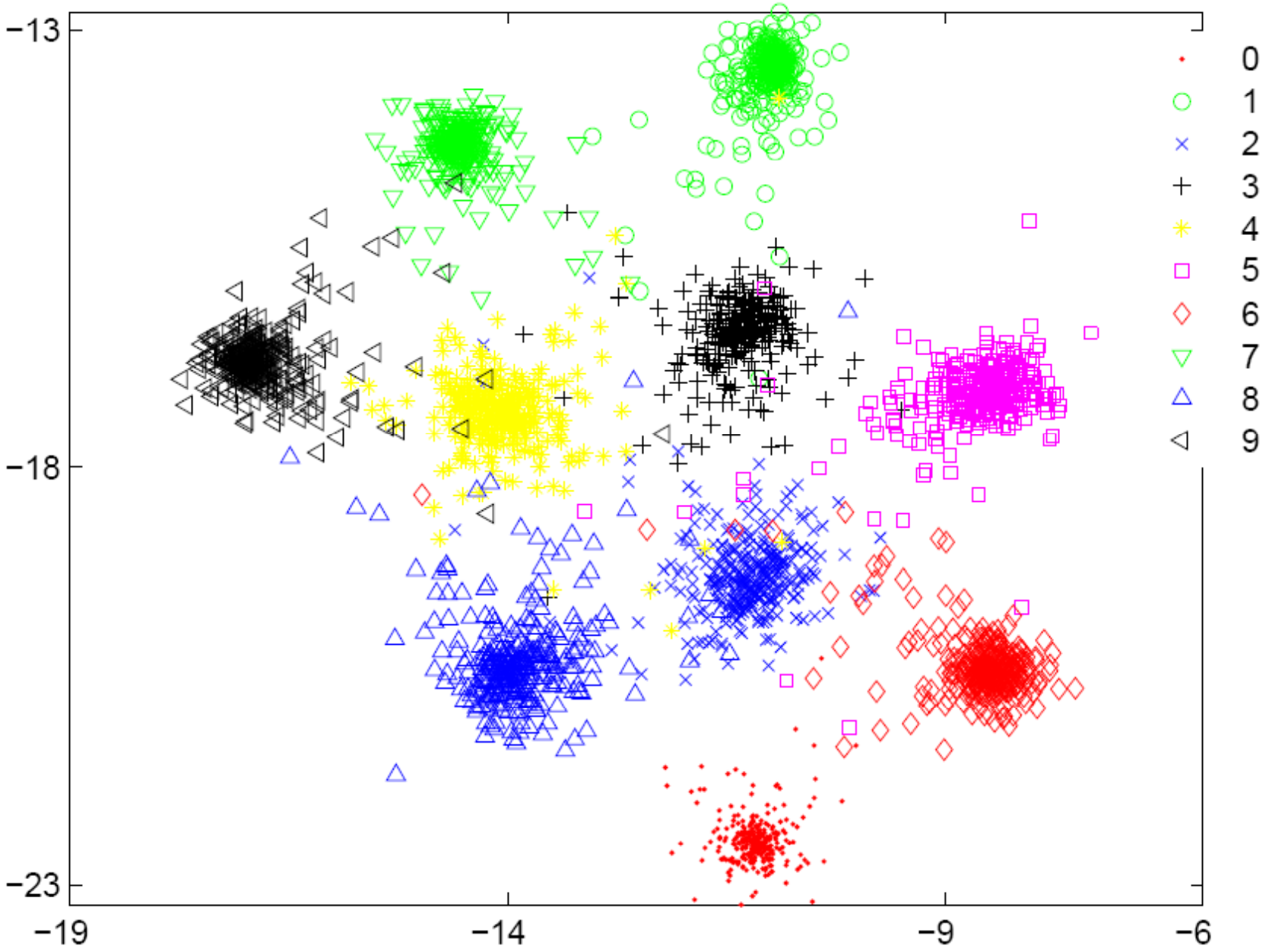
Embedding Results on USPS Digits (MCML)



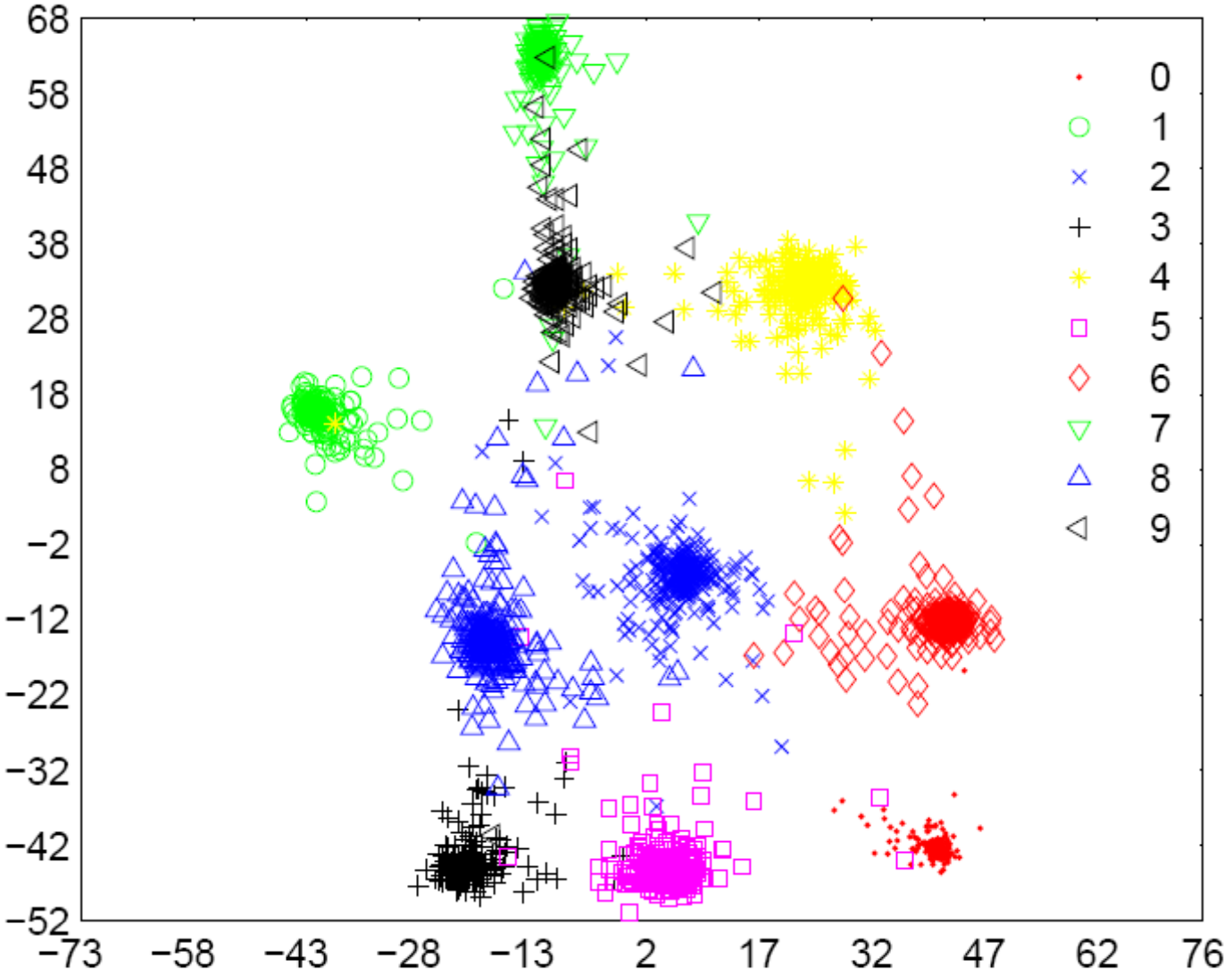
Embedding Results on USPS Digits (Dnet-kNN)



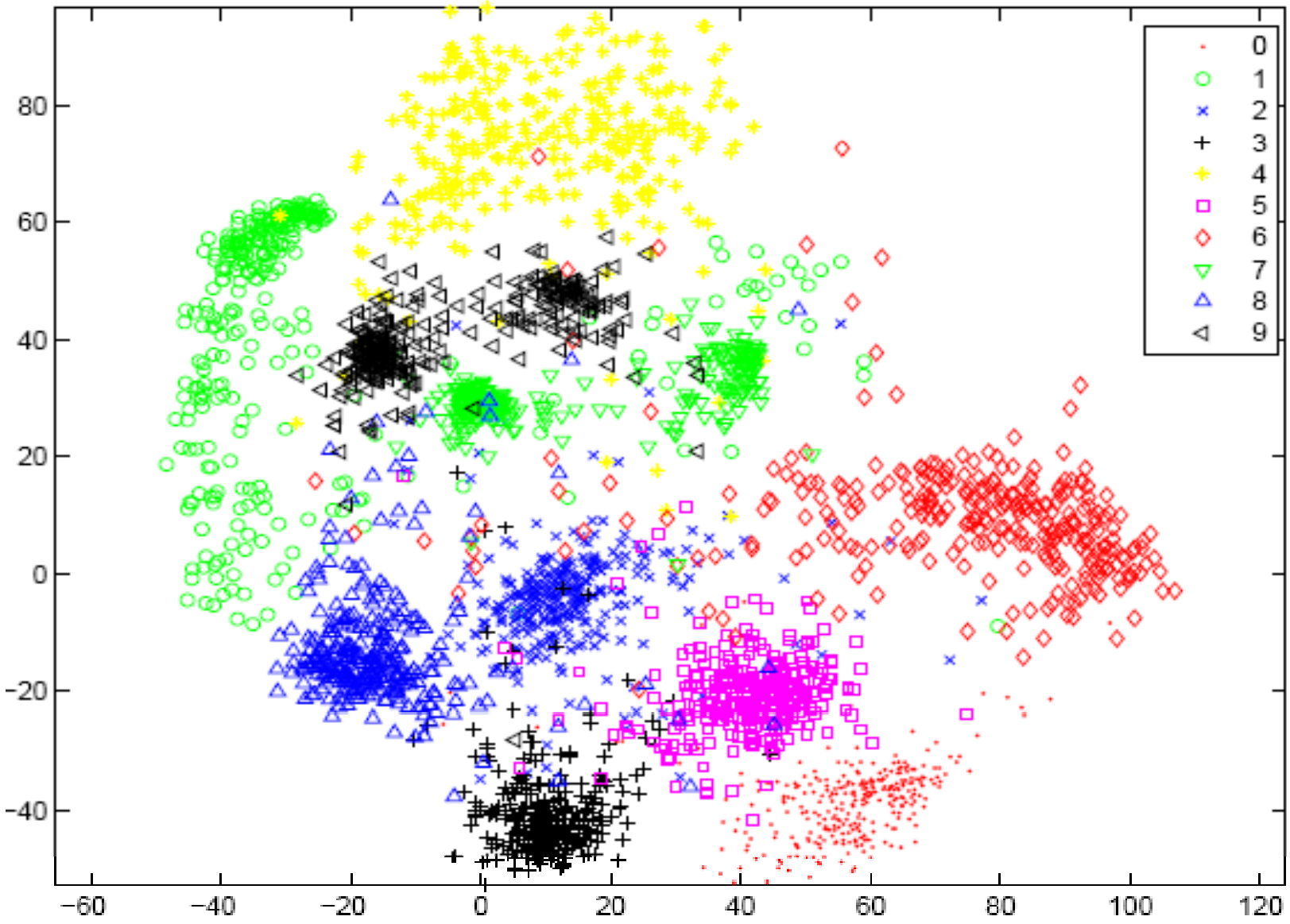
Embedding Results on USPS Digits (dG-MCML)



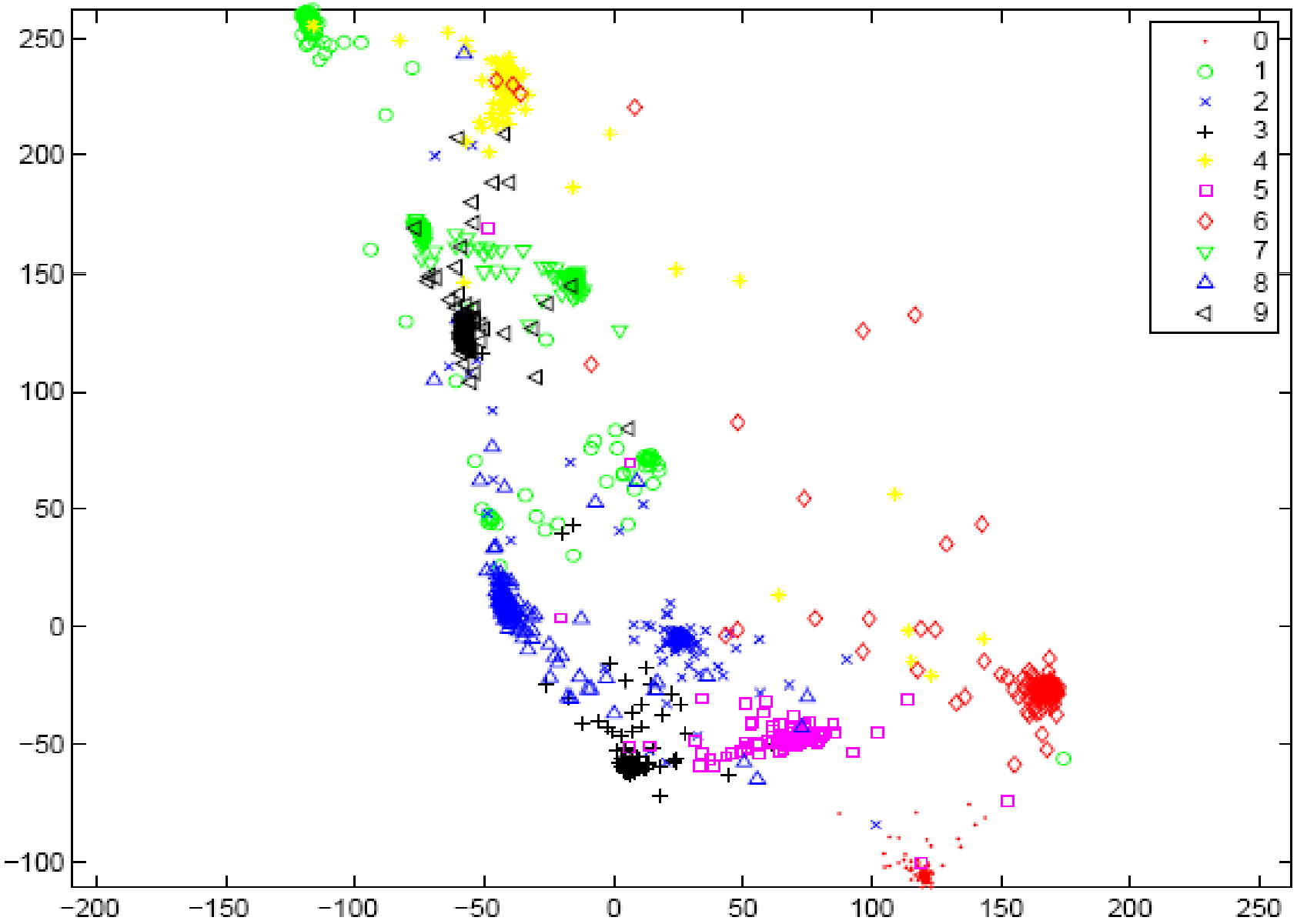
Embedding Results on USPS Digits (dt-MCML)



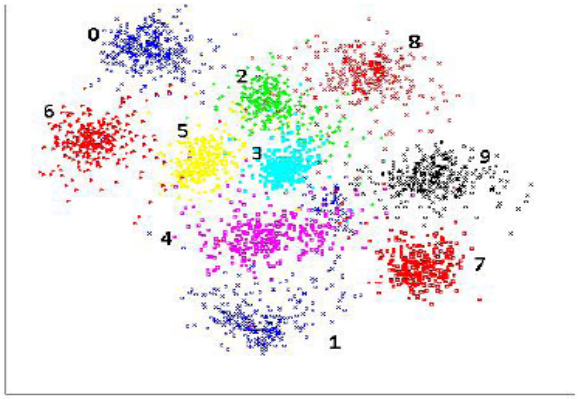
Embedding Results on USPS Digits (dG-NCA)



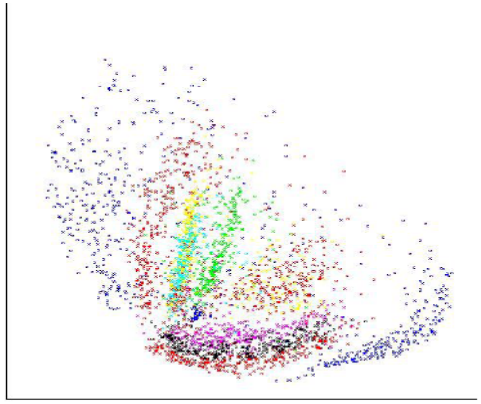
Embedding Results on USPS Digits (dt-NCA)



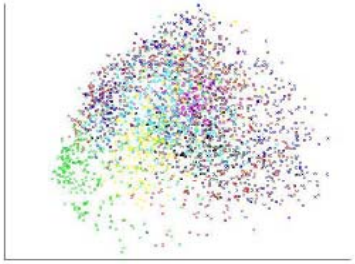
Embedding Results on USPS Handwritten Digits



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



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



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

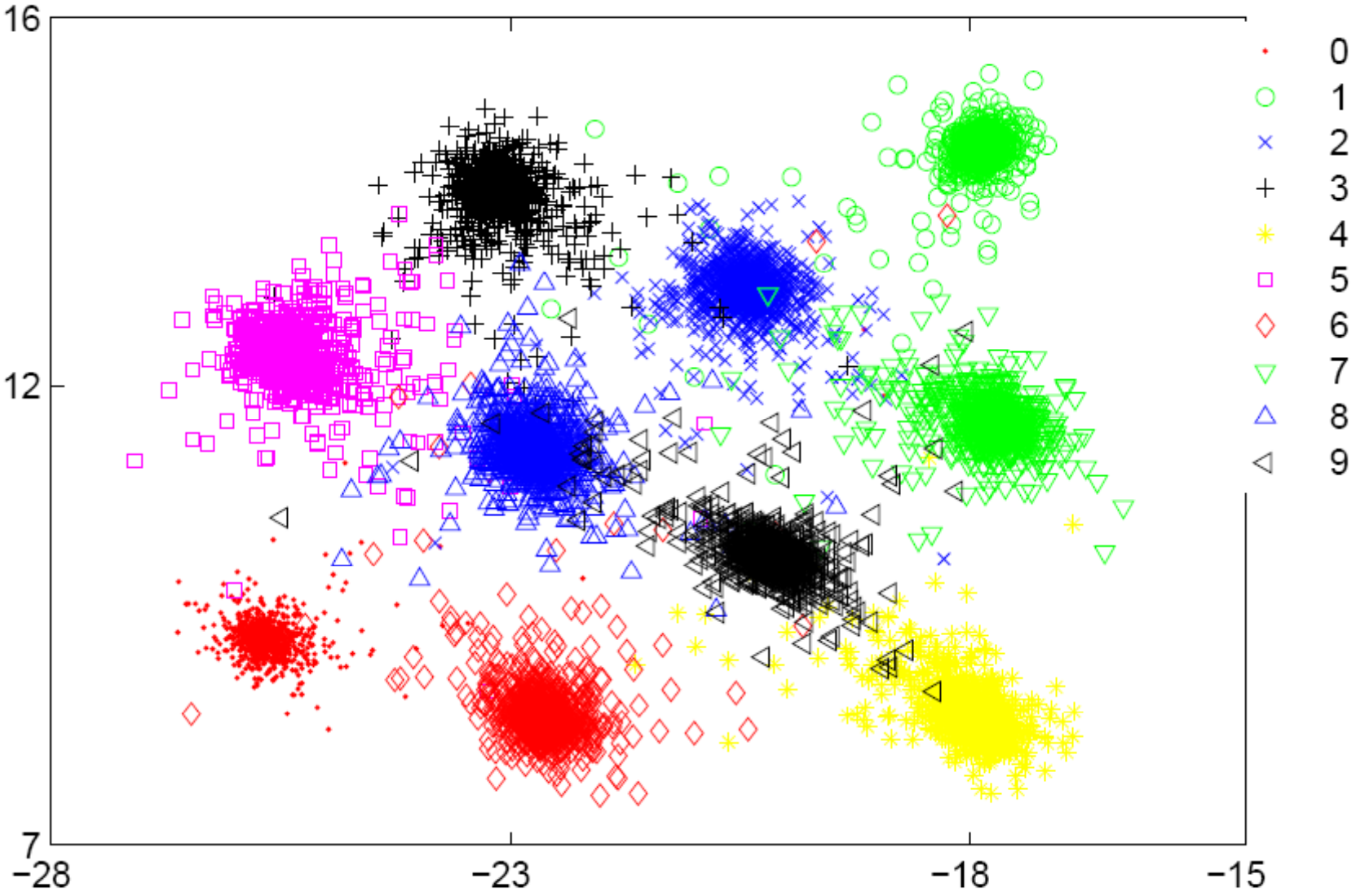
2D and 30D Embedding Results on MNIST Handwritten Digits

Table 2. Test error (in %) on 2-dimensional and 30-dimensional embedding for various techniques on the MNIST data set.

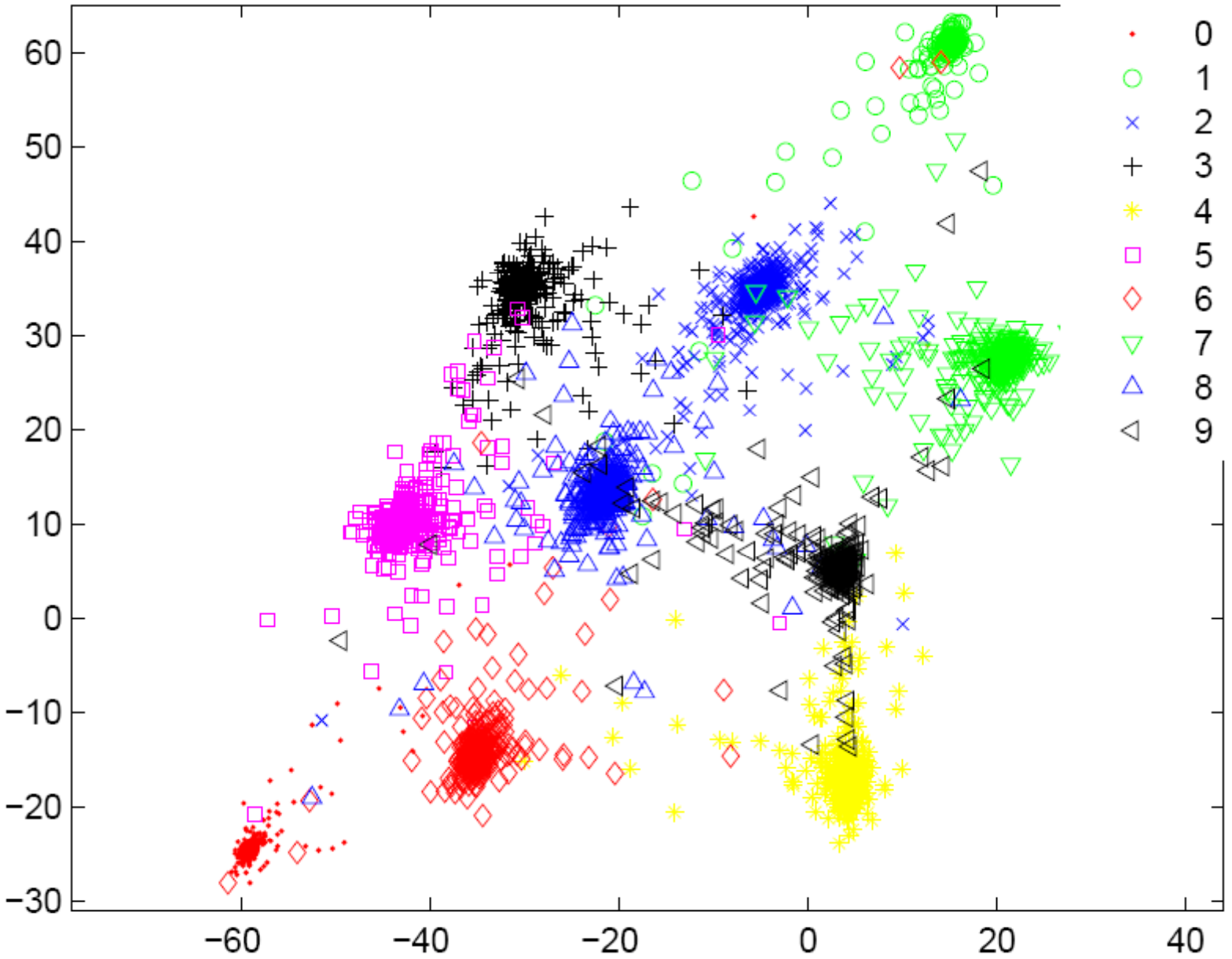
Dimensionality d	**2D**	**30D**
dG-MCML	2.13	1.49
dt-MCML ($\alpha = d - 1$)	**2.03**	1.63
dt-MCML (learned α)	2.14	1.49
dG-NCA	7.95	1.11
dt-NCA ($\alpha = d - 1$)	3.48	**0.92**
dt-NCA (learned α)	3.79	0.93

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

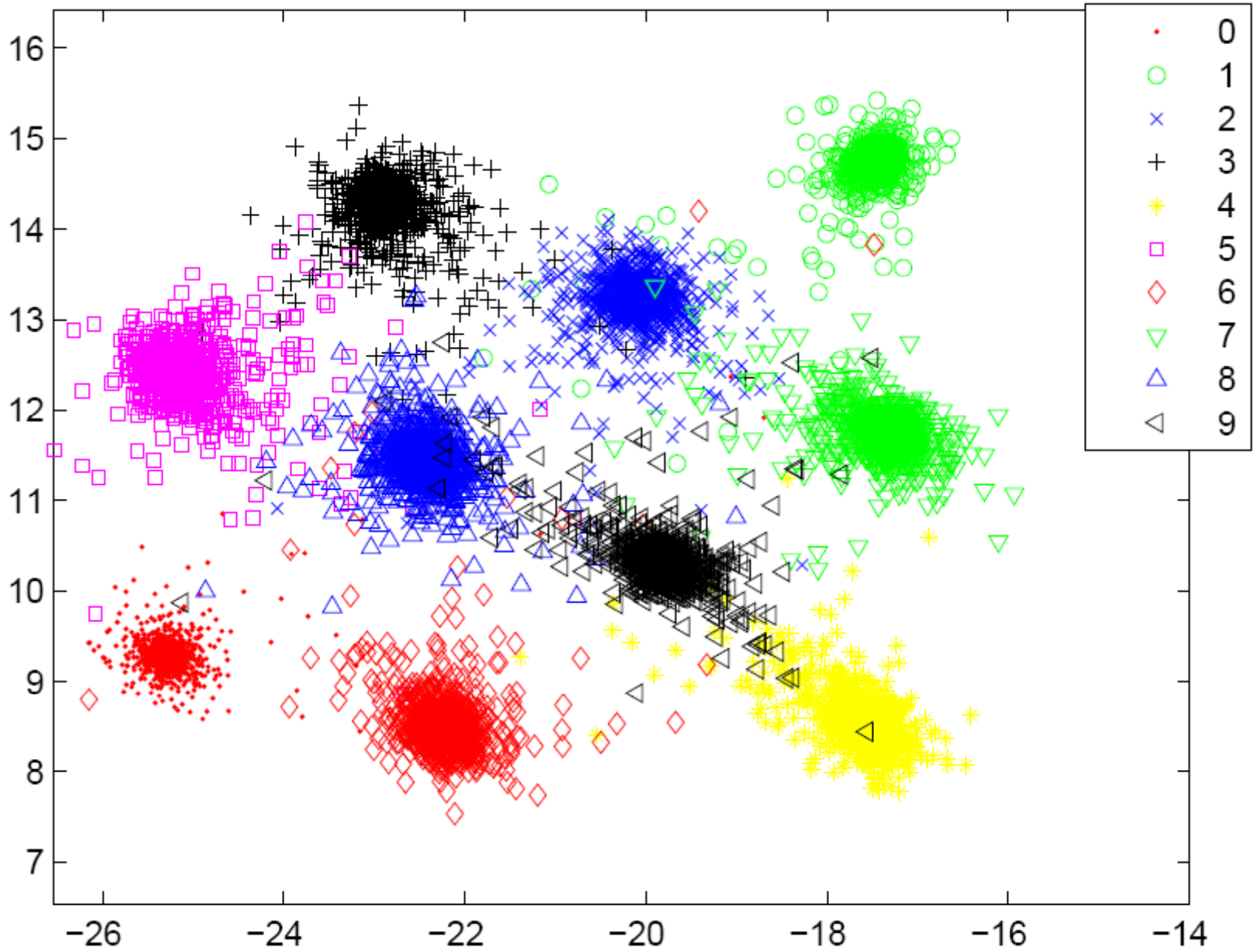
Embedding Results on MNIST Digits (dG-MCML)



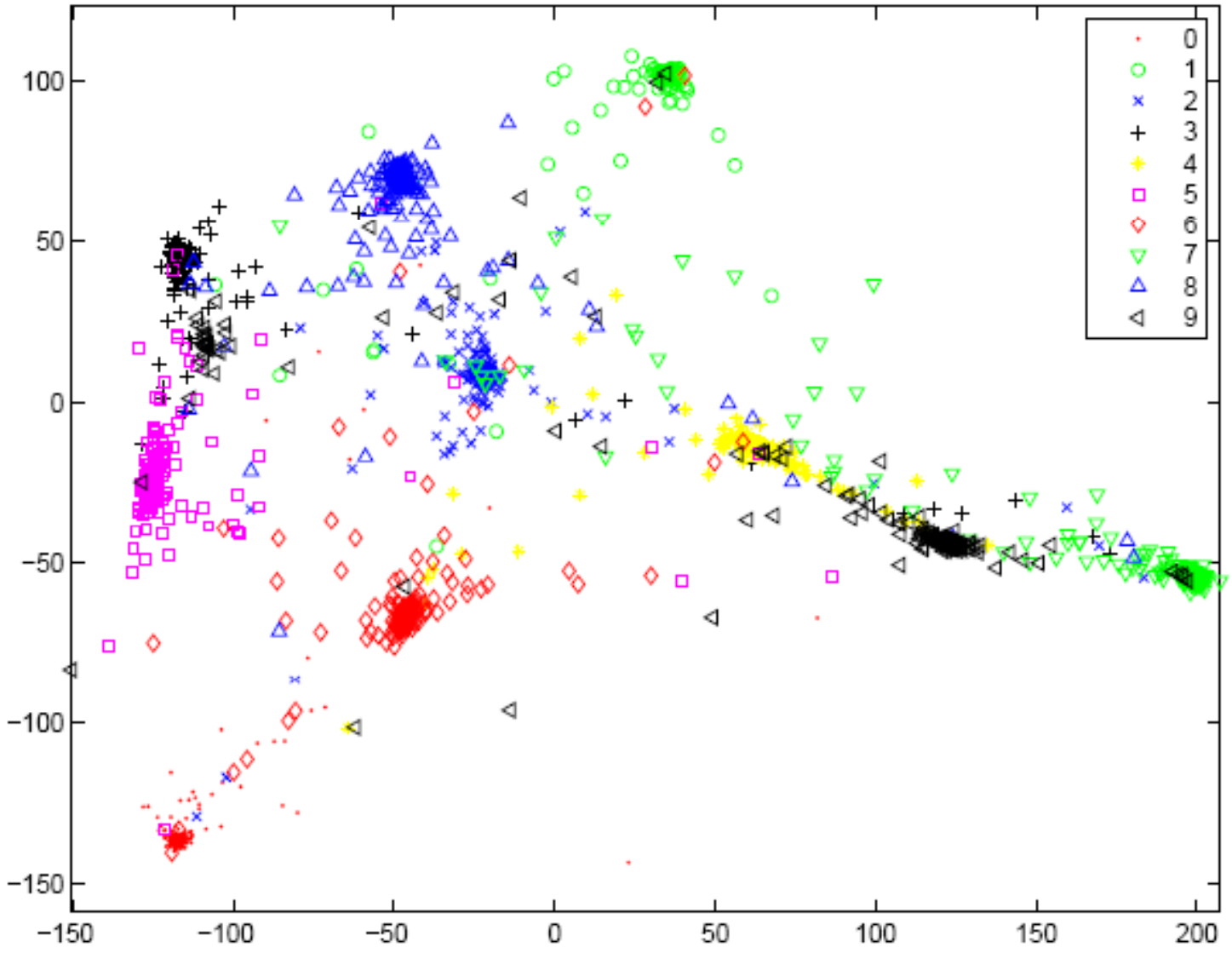
Embedding Results on MNIST Digits (dt-MCML)



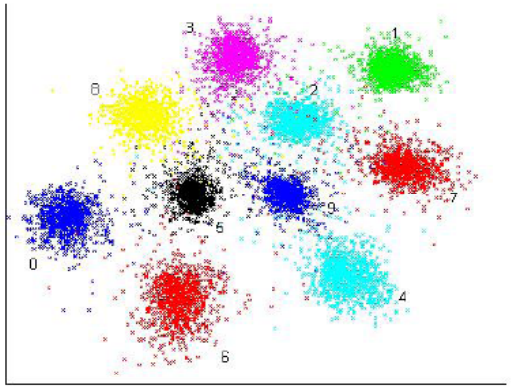
Embedding Results on MNIST Digits (dG-NCA)



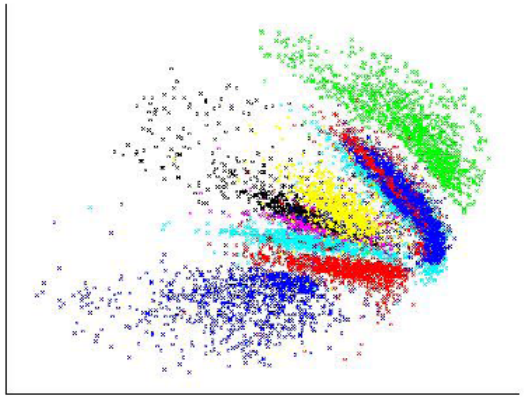
Embedding Results on MNIST Digits (dt-NCA)



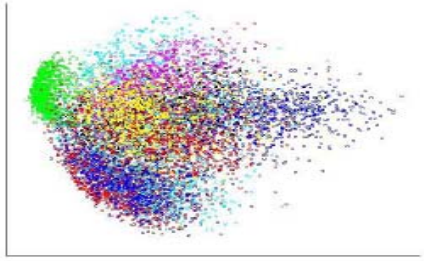
Embedding Results on MNIST Digits (Other Methods)



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



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



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

Conclusions

- Machine Learning is not just a black box
- Designing Graphical Models is art
- Non-linear dimensionality and kernel methods are cool and useful
- Most methods are available in packages

Acknowledgement

University of Toronto

Geoff Hinton

Hebrew University

Amir Globerson

Questions

Thank You