Advancement in Machine Learning: from Graphs, Graphical Models, and Neural Networks to Human Brains

Renqiang (Martin) Min

Program in Computational Biology and Bioinformatics

Molecular Biophysics & Biochemistry

Yale University

Dec 1, 2011

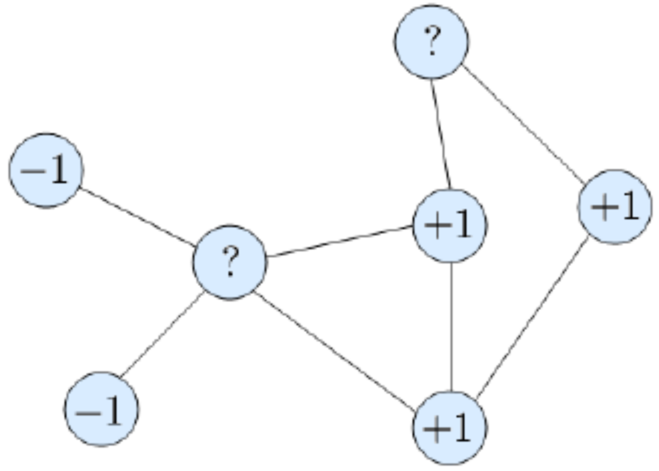
Outline

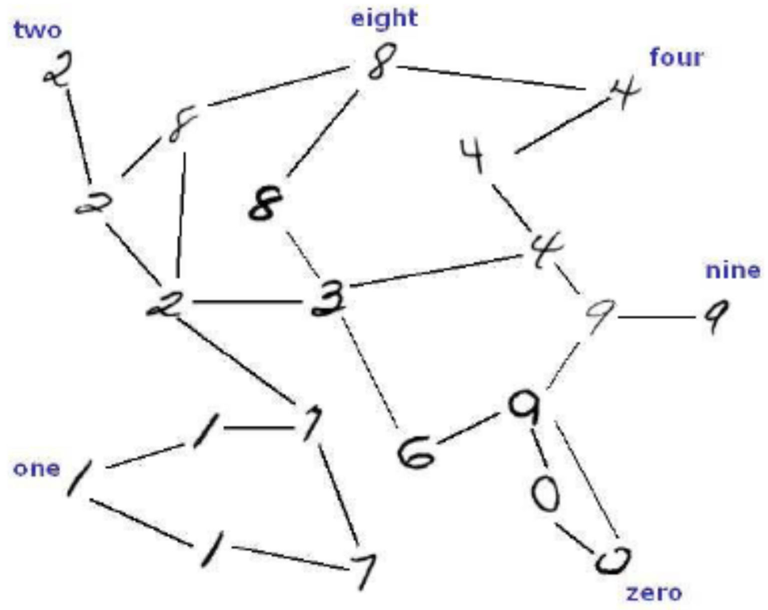
- Graph-based learning: label propagation
- Graphical models: Probabilistic PCA, Mixture of Gaussians, Hidden Markov Models
- Neural Networks: traditional neural nets, Boltzmann machines, deep belief nets, deep Boltzmann machines
- Machine Learning vs. Human Learning: Challenges and limitations of existing modeling frameworks
- Future research

Machine Learning is different from data mining or statistics

- **High-dimensional data (often 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.**

Graph-based learning (Xiaojin Zhu, Dengyong Zhou)



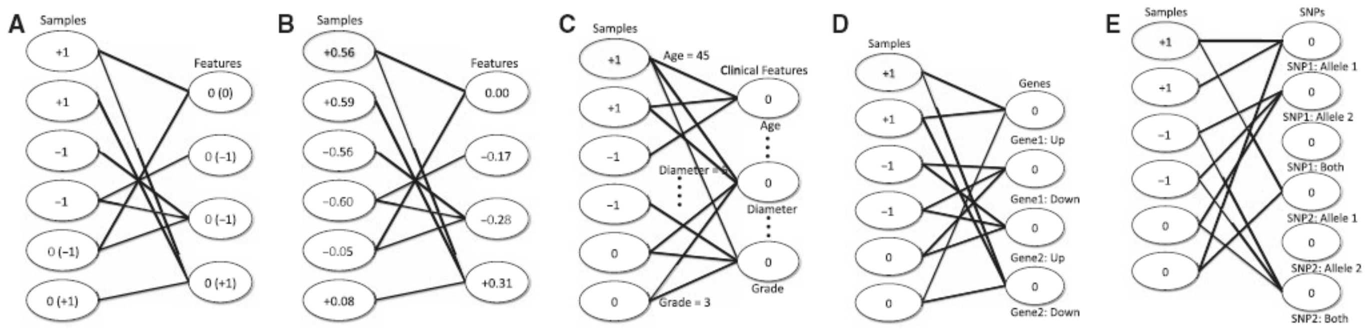


$$\min_{\boldsymbol{f}} \; (\boldsymbol{f}-\boldsymbol{y})^\top(\boldsymbol{f}-\boldsymbol{y}) + c\boldsymbol{f}^T L \boldsymbol{f},$$

$$\boldsymbol{f} = (I + cL)^{-1}\boldsymbol{y}$$

where $\boldsymbol{y} = (y_1, \ldots, y_p, 0, \ldots, 0)^\top$, and the matrix L is called the *graph Laplacian matrix*

Applications (Rui Kuang)



Graph labeling | Predicted labels | Clinical variables | Gene expressions | SNPs

$$\Omega(f) = \sum_{(v,u)\in E} w(v,u)\left(\frac{f(v)}{\sqrt{d(v)}} - \frac{f(u)}{\sqrt{d(u)}}\right)^2 + \varrho\sum_{v\in V}(f(v)-y(v))^2 + \varrho\sum_{u\in U}(f(u)-y(u))^2, \qquad (1)$$

$$\frac{\partial\Omega}{\partial f} = 2(I-S)*f^* + 2\varrho(f^*-y) = 0.$$

Let $\alpha = 1/(1+\varrho)$ and after rearrangement, the closed-form solution f^* can be computed as follows,

$$f^* = \frac{\varrho}{1+\varrho}\left(I - \frac{1}{1+\varrho}S\right) * y = (1-\alpha)(I-\alpha S)^{-1} * y. \qquad (2)$$

(1) Normalize the bipartite graph by computing $B = D_V^{-\frac{1}{2}} * W * D_U^{-\frac{1}{2}}$.

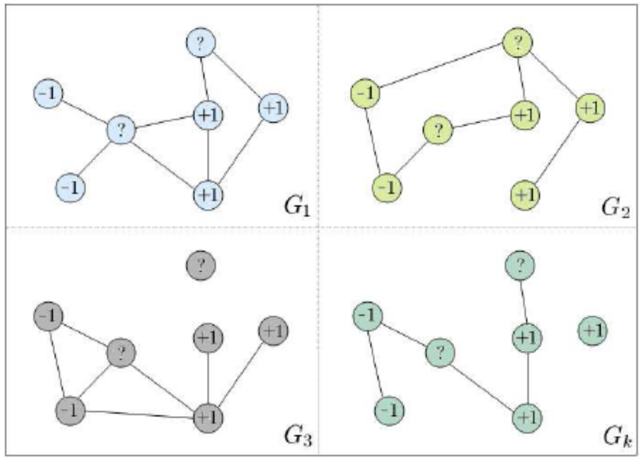
(2) Choose parameter α and perform a two direction propagation, until convergency (t denotes the time step):

- For each $v \in V$,
  $f(v)^t = (1-\alpha)y(v) + \alpha\sum_{u\in U} B_{i_v i_u} f(u)^{t-1}$

- For each $u \in U$,
  $f(u)^t = (1-\alpha)y(u) + \alpha\sum_{v\in V} B_{i_v i_u} f(v)^{t-1}$

(3) The sequence f^t converges to its limit f^* and f^* gives the class labels on the unlabeled vertices in both V and U.

Graph-based learning with mutiple networks (Tsuda, Bioinformatics, 2005)



$$\min_{\boldsymbol{f},\xi,\gamma} \quad (\boldsymbol{f}-\boldsymbol{y})^\top(\boldsymbol{f}-\boldsymbol{y}) + c\gamma + c_0 \sum_{k=1}^m \xi_k$$
$$\boldsymbol{f}^T L_k \boldsymbol{f} \le \gamma + \xi_k, \quad \xi_k \ge 0, \gamma \ge 0.$$

The dual problem then reads

$$\min_\alpha \quad \boldsymbol{y}^\top (I + \textstyle\sum_{k=1}^m \alpha_k L_k)^{-1} \boldsymbol{y} \equiv d(\alpha)$$
$$0 \le \alpha_k \le c_0, \ \textstyle\sum_k \alpha_k \le c.$$

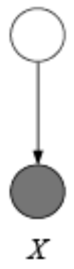
$$\frac{\partial d}{\partial \alpha_j} = -\boldsymbol{y}^\top (I + \sum_{k=1}^m \alpha_k L_k)^{-1} L_j (I + \sum_{k=1}^m \alpha_k L_k)^{-1} \boldsymbol{y}.$$

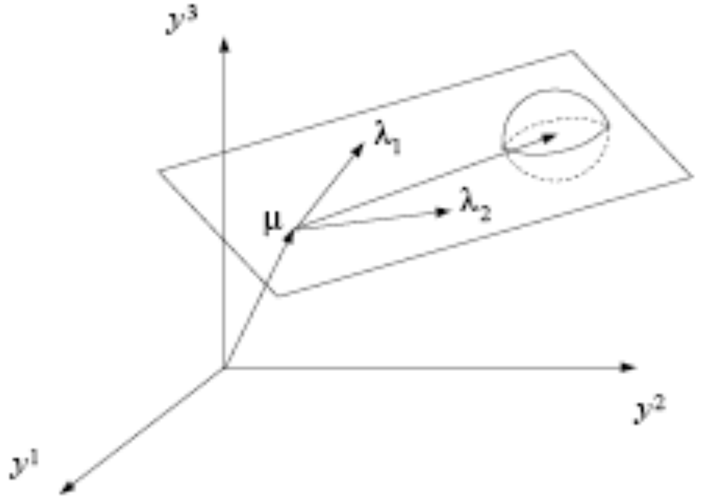
Graphical Models: nodes as variables instead of data points

- Define joint probability distributions intuitively and conveniently using graphs.
- Nodes represent variables and edges represent statistical dependencies.
- Missing edge patterns correspond to conditional independencies.

Continuous Latent Variables

- In many models there are some *underlying causes* of the data.
- Mixture models use a discrete class variable: clustering.
- Sometimes, it is more appropriate to think in terms of continuous *factors* which control the data we observe. Geometrically, this is equivalent to thinking of a data *manifold* or subspace.

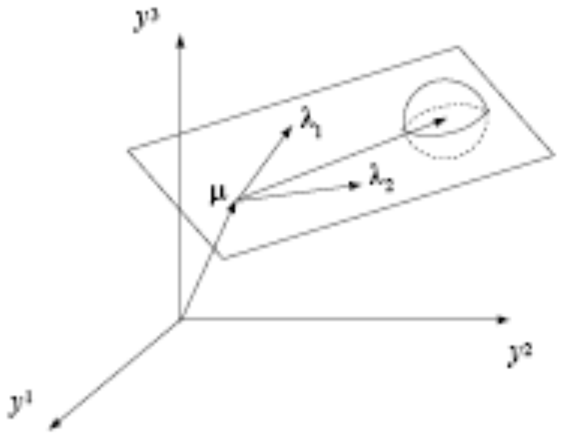




- To generate data, first generate a point within the manifold then add noise. Coordinates of point are components of latent variable.

Factor Analysis

- When we assume that the subspace is *linear* and that the underlying latent variable has a Gaussian distribution we get a model known as *factor analysis*:
  — data $\mathbf{y}$ (p-dim);
  — latent variable $\mathbf{x}$ (k-dim)



$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|0, I)$$
$$p(\mathbf{y}|\mathbf{x}, \theta) = \mathcal{N}(\mathbf{y}|\mu + \Lambda\mathbf{x}, \Psi)$$

where μ is the mean vector, Λ is the p by k *factor loading matrix*, and Ψ is the *sensor noise covariance* (ususally diagonal).

- Important: since the product of Gaussians is still Gaussian, the joint distribution $p(\mathbf{x}, \mathbf{y})$, the other marginal $p(\mathbf{y})$ and the conditional $p(\mathbf{x}|\mathbf{y})$ are also Gaussian.

Marginal Data Distribution

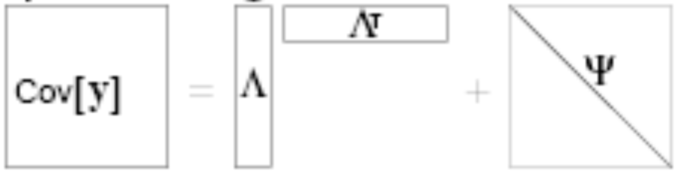
- Just as with discrete latent variables, we can compute the marginal density $p(\mathbf{y}|\theta)$ by summing out $\mathbf{x}$. But now the sum is an integral:
$$p(\mathbf{y}|\theta) = \int_{\mathbf{x}} p(\mathbf{x})p(\mathbf{y}|\mathbf{x},\theta)d\mathbf{x} = \mathcal{N}(\mathbf{y}|\mu\ ,\ \Lambda\Lambda^\top + \Psi)$$
which can be done by completing the square in the exponent.
- However, since the marginal is Gaussian, we can also just compute its mean and covariance. (Assume noise uncorrelated with data.)
$$\begin{aligned}
E[\mathbf{y}] &= E[\mu + \Lambda\mathbf{x} + \text{noise}] = \mu + \Lambda E[\mathbf{x}] + E[\text{noise}] \\
&= \mu + \Lambda \cdot 0 + 0 = \mu \\
\text{Cov}[\mathbf{y}] &= E[(\mathbf{y} - \mu)(\mathbf{y} - \mu)^\top] \\
&= E[(\mu + \Lambda\mathbf{x} + \text{noise} - \mu)(\mu + \Lambda\mathbf{x} + \text{noise} - \mu)^\top] \\
&= E[(\Lambda\mathbf{x} + n)(\Lambda\mathbf{x} + n)^\top] = \Lambda E(\mathbf{x}\mathbf{x}^\top)\Lambda^\top + E(nn^\top) \\
&= \Lambda\Lambda^\top + \Psi
\end{aligned}$$

FA = Constrained Covariance Gaussian

- Marginal density for factor analysis ($\mathbf{y}$ is p-dim, $\mathbf{x}$ is k-dim):

$$p(\mathbf{y}|\theta) = \mathcal{N}(\mathbf{y}|\mu\,,\ \Lambda\Lambda^\top + \Psi)$$

- So the effective covariance is the low-rank outer product of two long skinny matrices plus a diagonal matrix:



- In other words, factor analysis is just a constrained Gaussian model. (If Ψ were not diagonal then we could model any Gaussian and it would be pointless.)
- Learning: how should we fit the ML parameters?
- It is easy to find μ: just take the mean of the data. From now on assume we have done this and re-centred $\mathbf{y}$.
- What about the other parameters?

EM for Factor Analysis

- We will do maximum likelihood learning using (surprise, surprise) the EM algorithm.
  E-step: $q_n^{t+1} = p(\mathbf{x}^n|\mathbf{y}^n, \theta^t)$
  M-step: $\theta^{t+1} = \mathrm{argmax}_\theta \sum_n \int_{\mathbf{X}} q^{t+1}(\mathbf{x}^n|\mathbf{y}^n) \log p(\mathbf{y}^n, \mathbf{x}^n|\theta) d\mathbf{x}^n$
- For E-step we need the conditional distribution (inference)
  For M-step we need the expected log of the complete data.

$$\mathbf{E}-\mathbf{step} : q_n^{t+1} = p(\mathbf{x}^n|\mathbf{y}^n, \theta^t) = \mathcal{N}(\mathbf{x}^n|\mathbf{m}^n, \mathbf{V}^n)$$

$$\mathbf{M}-\mathbf{step} : \Lambda^{t+1} = \mathrm{argmax}_\Lambda \sum_n \langle \ell_c(\mathbf{x}^n, \mathbf{y}^n) \rangle_{q_n^{t+1}}$$

$$\Psi^{t+1} = \mathrm{argmax}_\Psi \sum_n \langle \ell_c(\mathbf{x}^n, \mathbf{y}^n) \rangle_{q_n^{t+1}}$$

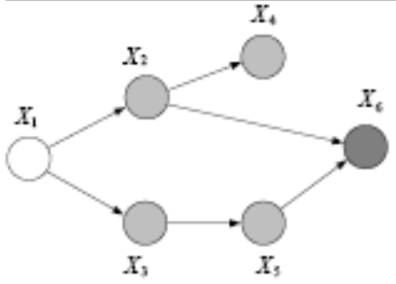
Final Algorithm: EM for Factor Analysis

- First, set μ equal to the sample mean $(1/N)\sum_n \mathbf{y}_n$, and subtract this mean from all the data.
- Now run the following iterations:

$$\begin{aligned}
\mathbf{E-step}: q^{t+1} &= p(\mathbf{x}|\mathbf{y},\theta^t) = \mathcal{N}(\mathbf{x}^n|\mathbf{m}^n,\mathbf{V}^n) \\
\mathbf{V}^n &= (I + \Lambda^\top \Psi^{-1}\Lambda)^{-1} \\
\mathbf{m}^n &= \mathbf{V}^n \Lambda^\top \Psi^{-1}(\mathbf{y}-\mu) \\
\mathbf{M-step}: \Lambda^{t+1} &= \left(\sum_n \mathbf{y}^n \mathbf{m}^{n\top}\right)\left(\sum_n \mathbf{V}^n\right)^{-1} \\
\Psi^{t+1} &= \frac{1}{N}\mathrm{diag}\left[\sum_n \mathbf{y}^n\mathbf{y}^{n\top} + \Lambda^{t+1}\sum_n \mathbf{m}^n\mathbf{y}^{n\top}\right]
\end{aligned}$$

Bayesian Networks

EXAMPLE



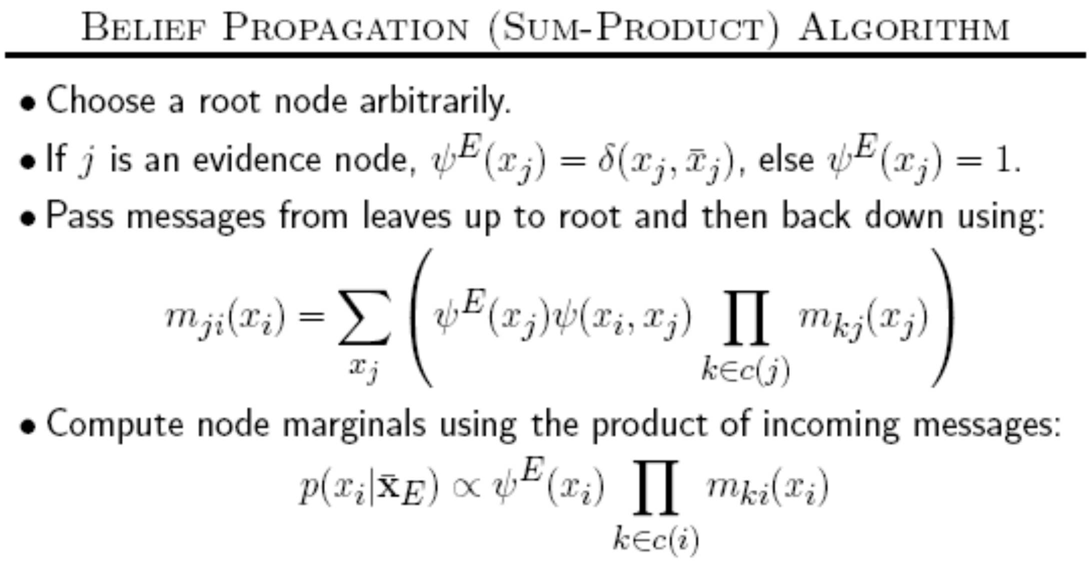
The key is to factor and then apply the distributive law.

$$p(x_1|\bar{x}_6) = p(x_1, \bar{x}_6)/p(\bar{x}_6) = p(x_1, \bar{x}_6)/\sum_{x_1'} p(x_1', \bar{x}_6)$$

$$\begin{aligned}
p(x_1, \bar{x}_6) &= \sum_{x_2}\sum_{x_3}\sum_{x_4}\sum_{x_5} p(x_1)p(x_2|x_1)p(x_3|x_1)p(x_4|x_2)p(x_5|x_3)p(\bar{x}_6|x_2, x_5) \\
&= p(x_1)\sum_{x_2} p(x_2|x_1)\sum_{x_3} p(x_3|x_1)\sum_{x_4} p(x_4|x_2)\sum_{x_5} p(x_5|x_3)p(\bar{x}_6|x_2, x_5) \\
&= p(x_1)\sum_{x_2} p(x_2|x_1)\sum_{x_3} p(x_3|x_1)\Phi_5(x_2, x_3)\sum_{x_4} p(x_4|x_2) \\
&= p(x_1)\sum_{x_2} p(x_2|x_1)\Phi_4(x_2)\sum_{x_3} p(x_3|x_1)\Phi_5(x_2, x_3) \\
&= p(x_1)\sum_{x_2} p(x_2|x_1)\Phi_4(x_2)\Phi_3(x_1, x_2) \\
&= p(x_1)\Phi_2(x_1)
\end{aligned}$$

Trees

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



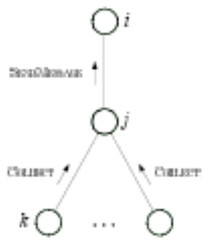
BELIEF PROPAGATION (SUM-PRODUCT) ALGORITHM

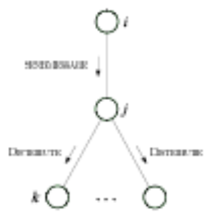
- Choose a root node arbitrarily.
- If j is an evidence node, $\psi^E(x_j) = \delta(x_j, \bar{x}_j)$, else $\psi^E(x_j) = 1$.
- Pass messages from leaves up to root and then back down using:

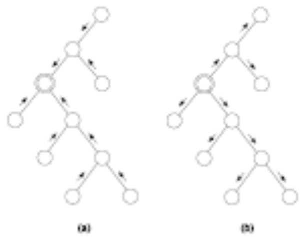
$$m_{ji}(x_i) = \sum_{x_j} \left(\psi^E(x_j)\psi(x_i, x_j) \prod_{k \in c(j)} m_{kj}(x_j) \right)$$

- Compute node marginals using the product of incoming messages:

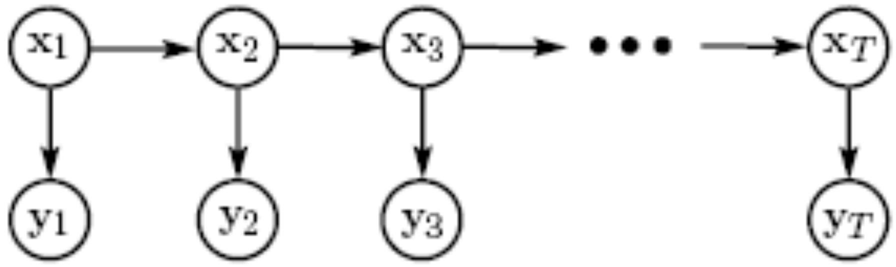
$$p(x_i|\bar{\mathbf{x}}_E) \propto \psi^E(x_i) \prod_{k \in c(i)} m_{ki}(x_i)$$







HMM Graphical Model



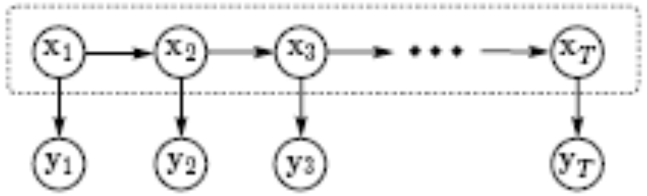
- Hidden states $\{x_t\}$, outputs $\{\mathbf{y}_t\}$
  Joint probability factorizes:

$$\mathsf{P}(\{\mathbf{x}\},\{\mathbf{y}\}) = \prod_{t=1}^{T} \mathsf{P}(x_t|\mathbf{x}_{t-1})\mathsf{P}(\mathbf{y}_t|x_t)$$
$$= \pi_{\mathbf{x}_1} \prod_{t=1}^{T-1} S_{x_t,x_{t+1}} \prod_{t=1}^{T} A_{x_t}(\mathbf{y}_t)$$

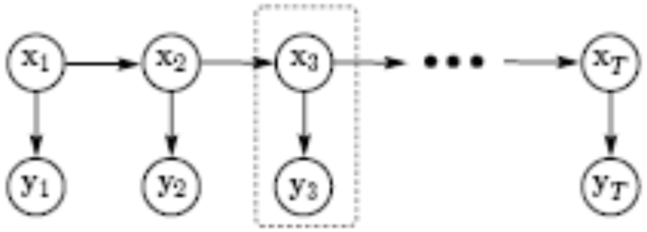
- NB: Data are *not* i.i.d.
  There is no easy way to use plates to show this model. (Why?)

Links to Other 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.

HMMs: Inference of Hidden States

- What if we we want to estimate the hidden states given observations? To start with, let us estimate a single hidden state:

$$p(x_t|\{\mathbf{y}\}) = \gamma(x_t) = \frac{p(\{\mathbf{y}\}|x_t)p(x_t)}{p(\{\mathbf{y}\})}$$

$$= \frac{p(\mathbf{y}_1^t|x_t)p(\mathbf{y}_{t+1}^T|x_t)p(x_t)}{p(\mathbf{y}_1^T)}$$

$$= \frac{p(\mathbf{y}_1^t, x_t)p(\mathbf{y}_{t+1}^T|x_t)}{p(\mathbf{y}_1^T)}$$

$$p(x_t|\{\mathbf{y}\}) = \gamma(x_t) = \frac{\alpha(x_t)\beta(x_t)}{p(\mathbf{y}_1^T)}$$

where
$$\alpha_j(t) = p(\ \mathbf{y}_1^t\ ,\ x_t = j\ )$$
$$\beta_j(t) = p(\mathbf{y}_{t+1}^T\ |\ x_t = j\ )$$
$$\gamma_i(t) = p(x_t = i\ |\ \mathbf{y}_1^T)$$

Forward-Backward Algorithm

- We compute these quantites efficiently using another recursion. Use total prob. of all paths going through state i at time t to compute the *conditional* prob. of being in state i at time t:

$$\begin{aligned}\gamma_i(t) &= p(x_t = i \mid \mathbf{y}_1^T) \\ &= \alpha_i(t)\beta_i(t)/L\end{aligned}$$

  where we defined:

$$\beta_j(t) = p(\mathbf{y}_{t+1}^T \mid x_t = j\ )$$

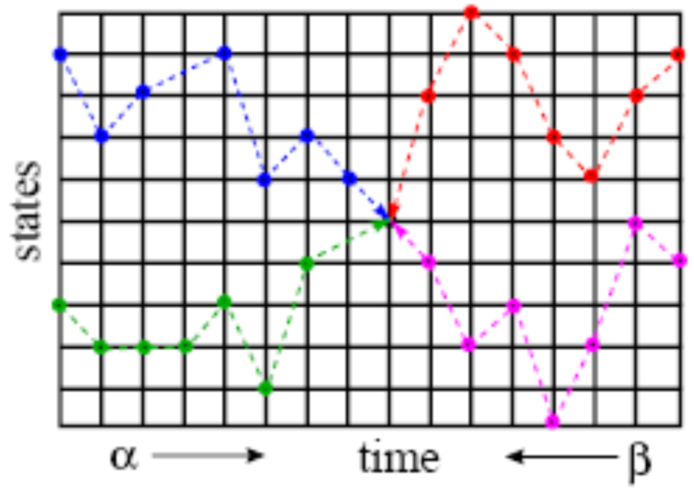
- There is also a simple recursion for $\beta_j(t)$:

$$\begin{aligned}\beta_j(t) &= \sum_i S_{ji}\beta_i(t+1)A_i(\mathbf{y}_{t+1}) \\ \beta_j(T) &= 1\end{aligned}$$

- $\alpha_i(t)$ gives total *inflow* of prob. to node (t, i)
  $\beta_i(t)$ gives total *outflow* of prob.

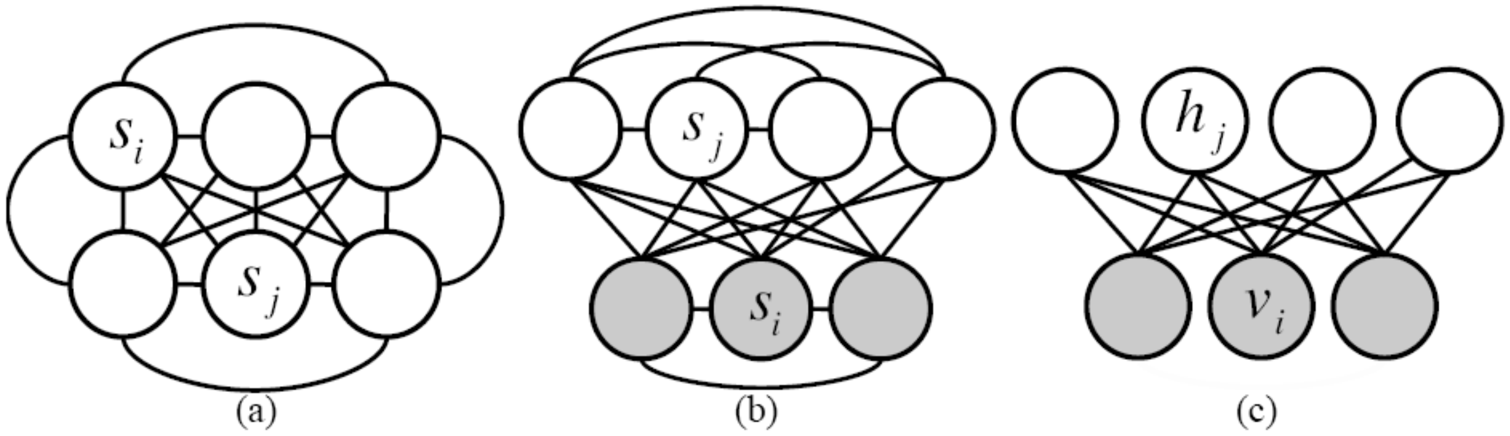
Forward-Backward Algorithm

- $\alpha_i(t)$ gives total *inflow* of prob. to node (t, i)
  $\beta_i(t)$ gives total *outflow* of prob.



- Bugs again: we just let the bugs run forward from time 0 to t and backward from time T to t.
- In fact, we can just do one forward pass to compute all the $\alpha_i(t)$ and one backward pass to compute all the $\beta_i(t)$ and then compute any $\gamma_i(t)$ we want. Total cost is $O(K^2T)$.

Undirected Graphical Models: Boltzmann Machines (Hinton)



a) A Boltzmann machine. b) A Boltzmann machine partitioned into visible (shaded) and hidden units. c) A restricted Boltzmann machine.

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

$$p(\mathbf{v},\mathbf{h}) = \frac{\exp\left(-E(\mathbf{v},\mathbf{h})\right)}{Z} \qquad Z = \sum_{\mathbf{v}',\mathbf{h}'} E(\mathbf{v}',\mathbf{h}').$$

$$\Delta W_{ij} \propto \langle v_i h_j \rangle_{\text{data}} - \langle v_i h_j \rangle_{\text{model}}.$$

$$p(h_j = 1|\mathbf{v}) = \frac{1}{1+\exp(-b_j - \sum_i W_{ij} v_i)}.$$

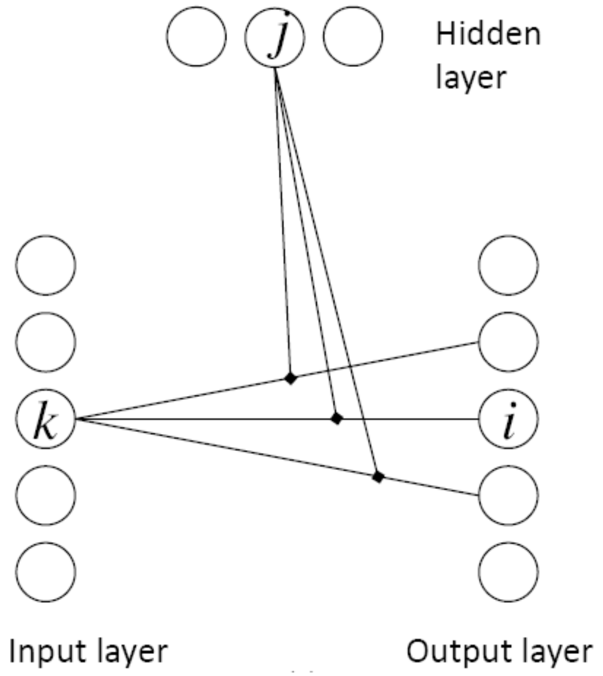
$$p(v_i = 1|\mathbf{h}) = \frac{1}{1+\exp(-a_i - \sum_j W_{ij} h_j)}.$$

$$\Delta W_{ij} \propto \langle v_i h_j \rangle_{\text{data}} - \langle v_i h_j \rangle_{\text{recon}},$$

$$\Delta a_i \propto \langle v_i \rangle_{\text{data}} - \langle v_i \rangle_{\text{recon}},$$

$$\Delta b_j \propto \langle h_j \rangle_{\text{data}} - \langle h_j \rangle_{\text{recon}}.$$

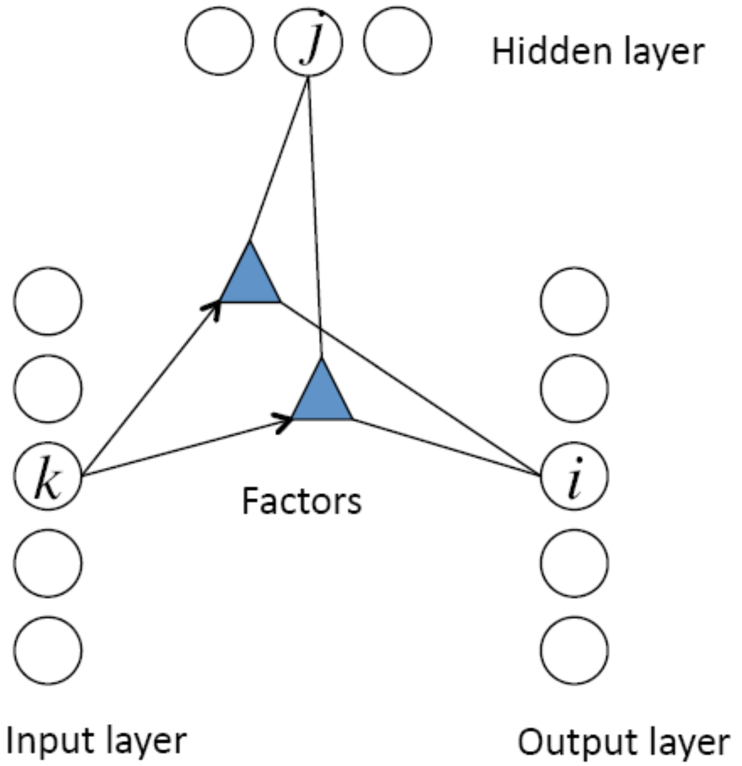
Gated RBM (Hinton)



$$E(\mathbf{v},\mathbf{h}|\mathbf{x}) = -\sum_{ijk} W_{ijk} v_i h_j x_k - \sum_{ij} c_{ij} v_i h_j - \sum_i a_i v_i - \sum_j b_j h_j$$

$$\Delta W_{ijk} \propto \langle v_i h_j x_k \rangle_{\text{data}} - \langle v_i h_j x_k \rangle_{\text{recon}}.$$

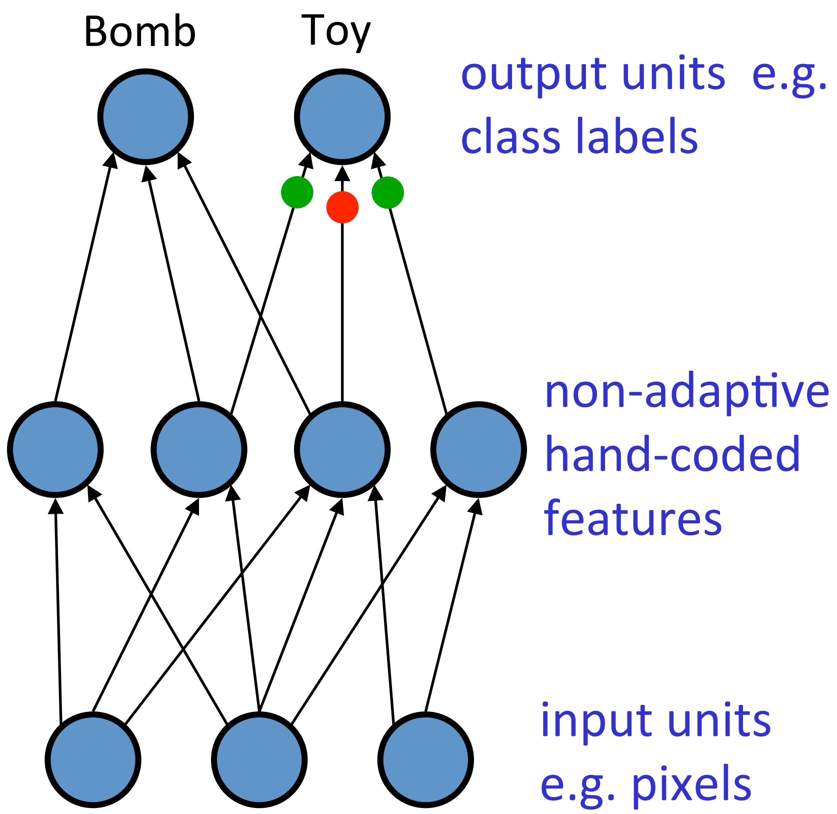
Factorization to reduce parameters (Hinton)



$$E\left(\mathbf{v},\mathbf{h}|\mathbf{x}\right) = -\sum_{f}\sum_{ijk} W^{\mathbf{v}}_{if} W^{\mathbf{h}}_{jf} W^{\mathbf{x}}_{kf} v_i h_j x_k - \sum_{ij} c_{ij} v_i h_j - \sum_{i} a_i v_i - \sum_{j} b_j h_j$$

Supervised Learning: First-Generation Neural Networks

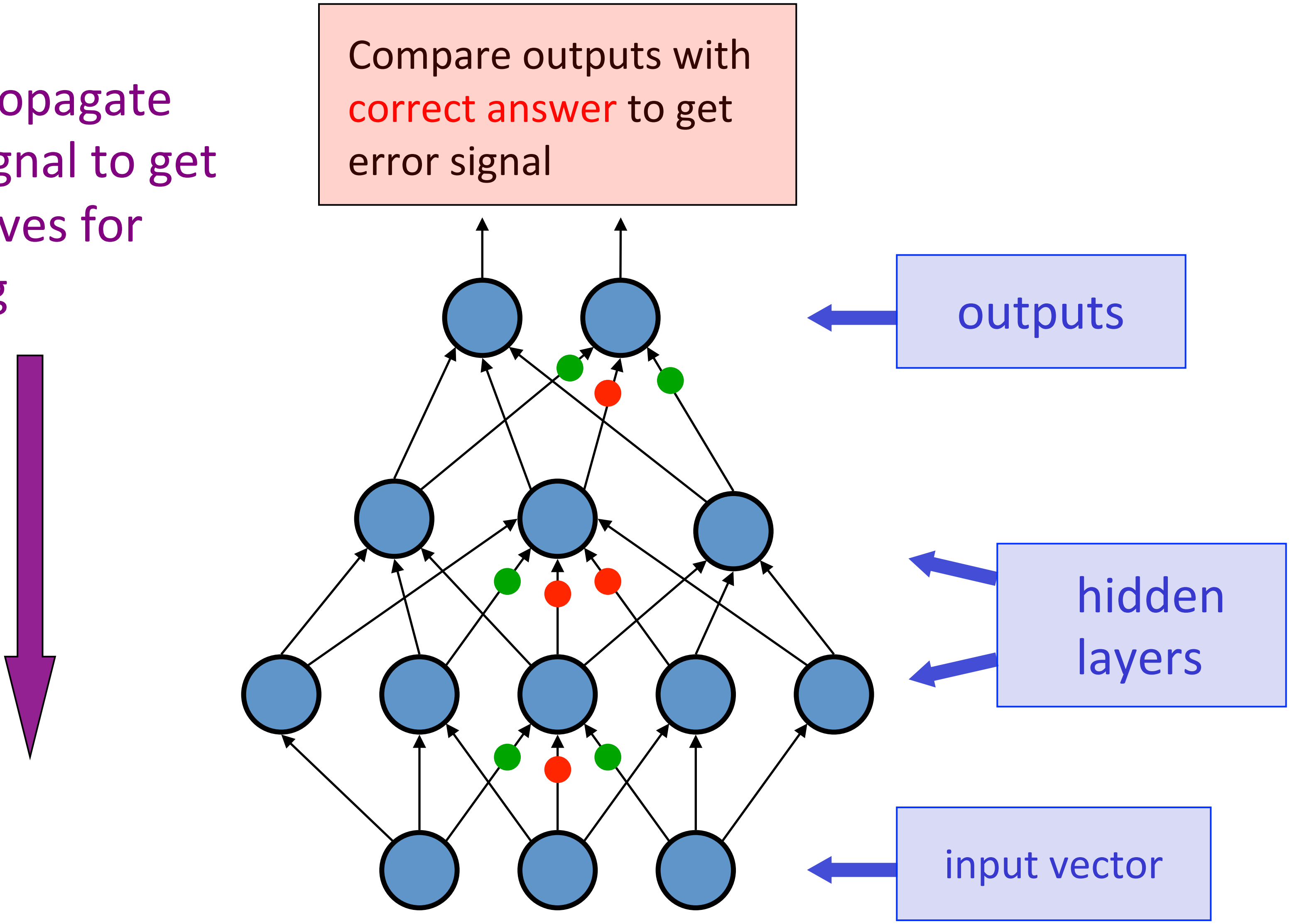
- Perceptrons (~1960) used a layer of hand-coded features and tried to recognize objects by learning how to weight these features.
  - There was a neat learning algorithm for adjusting the weights.
  - But perceptrons are fundamentally limited in what they can learn to do.


Bomb
Toy
output units e.g. class labels
non-adaptive hand-coded features
input units e.g. pixels

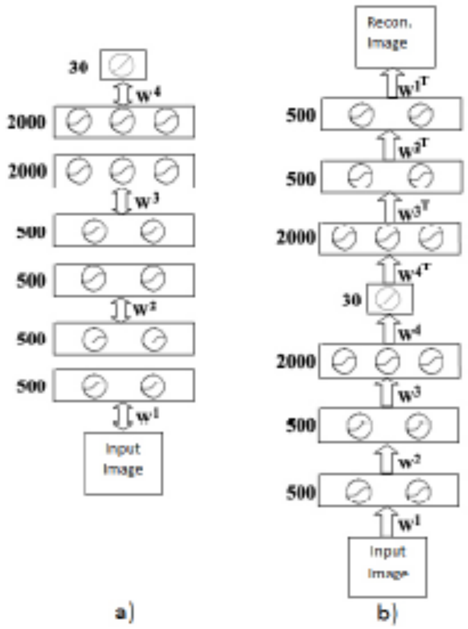
Sketch of a typical perceptron from the 1960's

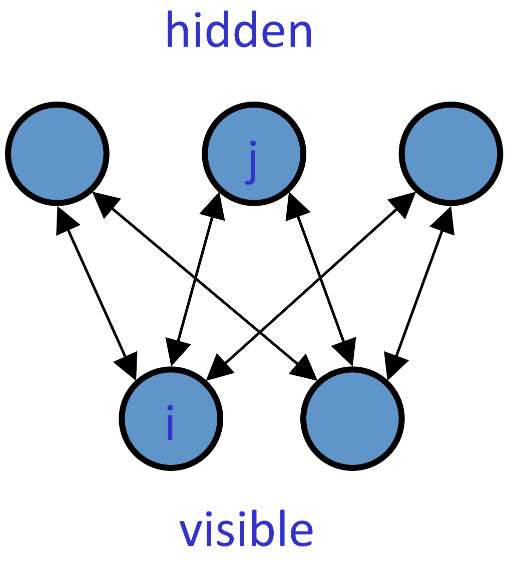
Second-generation neural networks (~1985)

Back-propagate error signal to get derivatives for learning

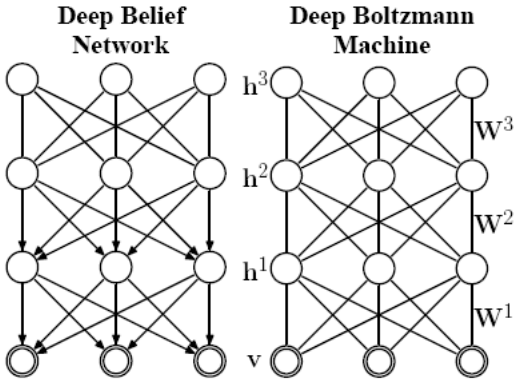


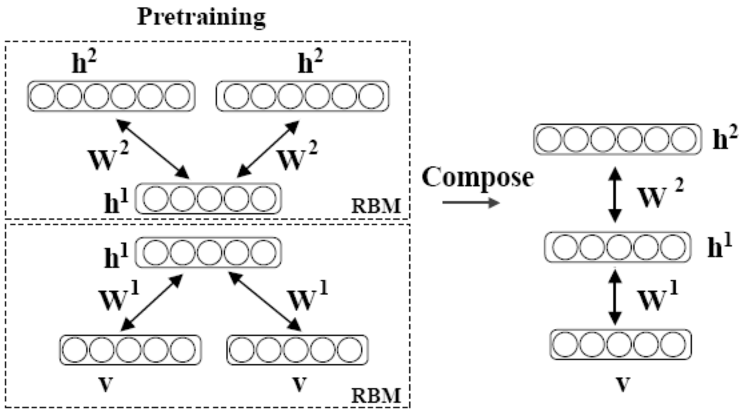
Third-generation neural networks





Deep Models (Rus, Tijimen, Hinton)





$$p(h_j^1 = 1|\mathbf{v}, \mathbf{h}^2) = \sigma\Big(\sum_i W_{ij}^1 v_i + \sum_m W_{jm}^2 h_j^2\Big),$$

$$p(h_m^2 = 1|\mathbf{h}^1) = \sigma\Big(\sum_j W_{im}^2 h_i^1\Big),$$

$$p(v_i = 1|\mathbf{h}^1) = \sigma\Big(\sum_j W_{ij}^1 h_j\Big).$$

$$\begin{aligned}\ln p(\mathbf{v};\theta) \;\geq\;& \frac{1}{2}\sum_{i,k} L_{ik} v_i v_k + \frac{1}{2}\sum_{j,m} J_{jm}\mu_j\mu_m \\ &+ \sum_{i,j} W_{ij} v_i \mu_j - \ln Z(\theta) \\ &+ \sum_j \left[\mu_j \ln \mu_j + (1-\mu_j)\ln(1-\mu_j)\right].\end{aligned}$$

$$\mu_j \leftarrow \sigma\Big(\sum_i W_{ij} v_i + \sum_{m\backslash j} J_{mj}\mu_m\Big).$$

Boltzmann Machine Learning Procedure:

Given: a training set of N data vectors $\{\mathbf{v}\}_{n=1}^N$.

1. Randomly initialize parameters θ^0 and M fantasy particles. $\{\tilde{\mathbf{v}}^{0,1}, \tilde{\mathbf{h}}^{0,1}\}, ..., \{\tilde{\mathbf{v}}^{0,M}, \tilde{\mathbf{h}}^{0,M}\}$
2. For t=0 to T (# of iterations)
   (a) For each training example $\mathbf{v}^n$, n=1 to N
      - Randomly initialize μ and run mean-field updates Eq. 8 until convergence.
      - Set $\mu^n = \mu$.

   (b) For each fantasy particle m=1 to M
      - Obtain a new state $(\tilde{\mathbf{v}}^{t+1,m}, \tilde{\mathbf{h}}^{t+1,m})$ by running a k-step Gibbs sampler using Eqs. 4, 5, initialized at the previous sample $(\tilde{\mathbf{v}}^{t,m}, \tilde{\mathbf{h}}^{t,m})$.

   (c) Update

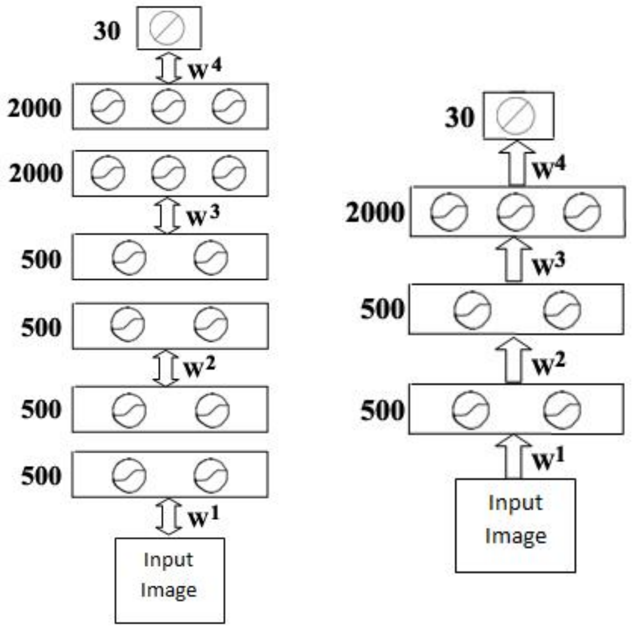
   $$W^{t+1} = W^t + \alpha_t\Big(\frac{1}{N}\sum_{n=1}^N \mathbf{v}^n(\mu^n)^\top - \frac{1}{M}\sum_{m=1}^M \tilde{\mathbf{v}}^{t+1,m}(\tilde{\mathbf{h}}^{t+1,m})^\top\Big).$$

   Similarly update parameters L and J.

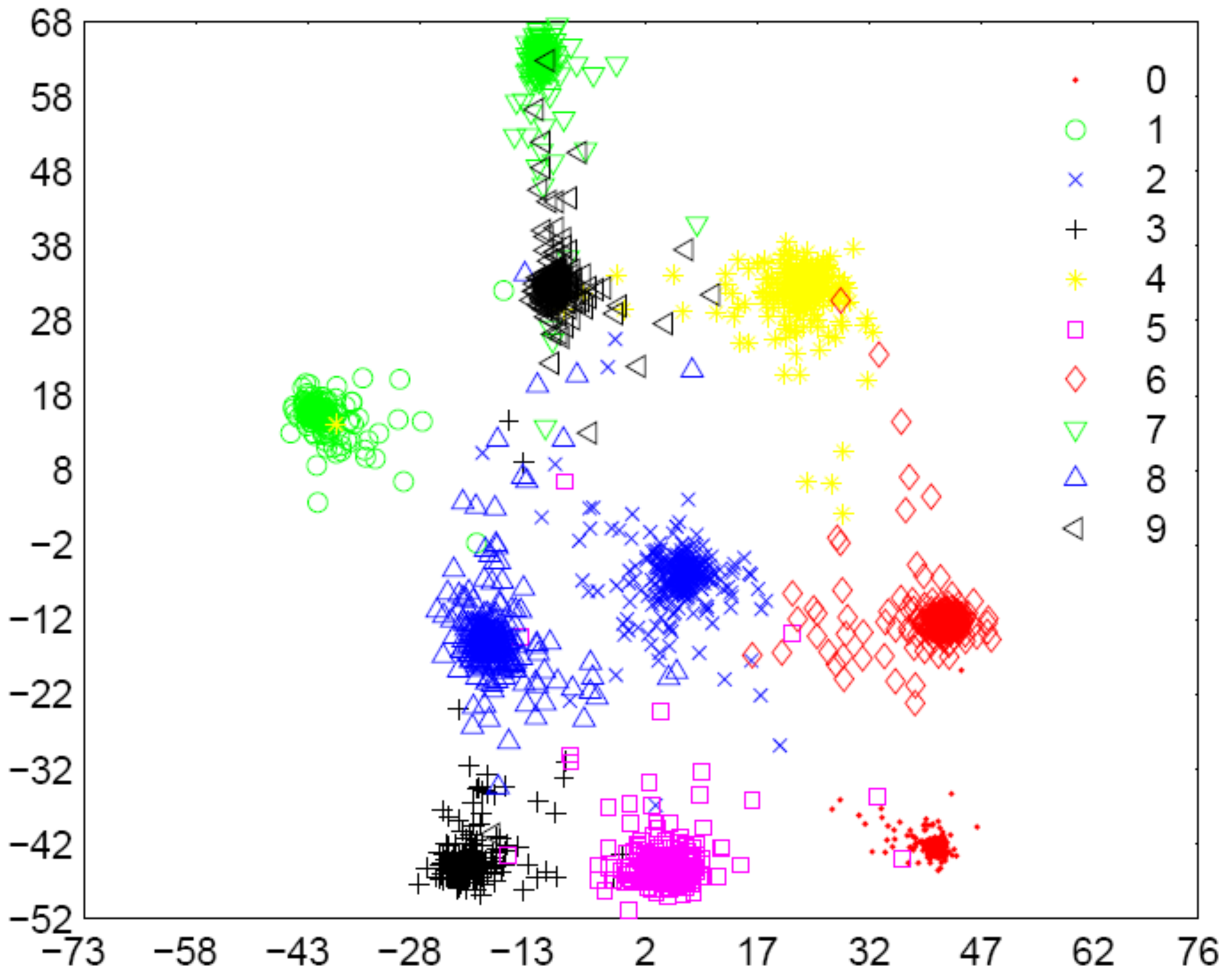
   (d) Decrease α_t.

Extend Supervised Linear Embedding Methods with Deep Neural Networks (Min et al., 2010)

- 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



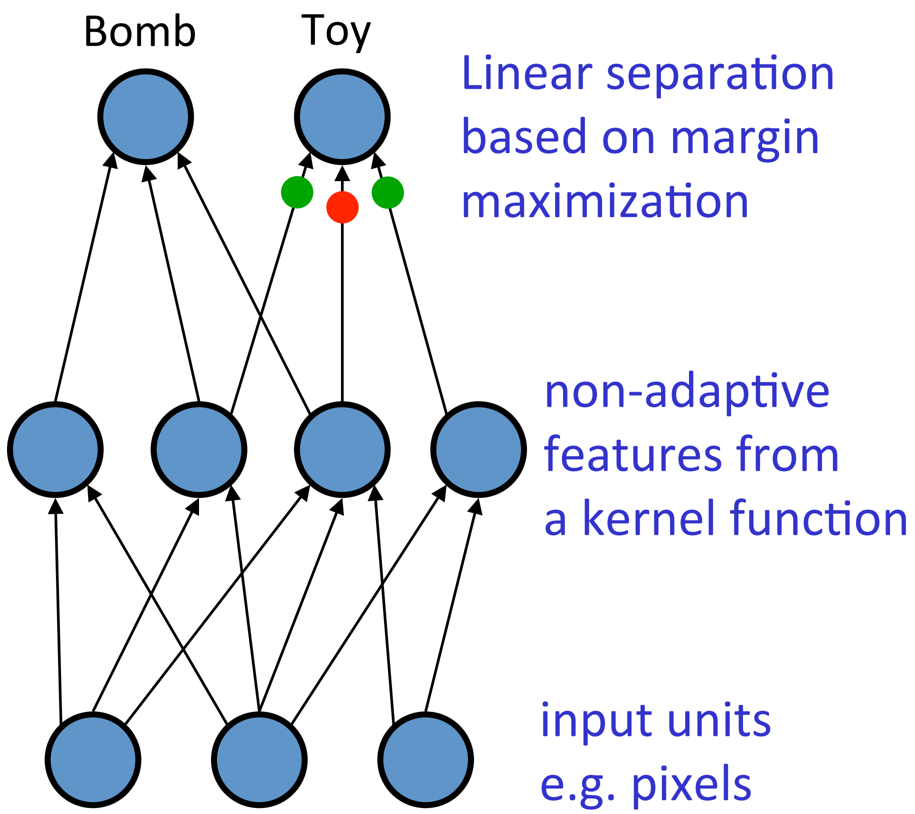
Embedding Results on USPS Digits (dt-MCML)



Supervised Learning: SVM as First-Generation Neural Networks

- Linear SVM: a margin maximization linear separating hyperplane

- Non-linear SVM: adaptive feature mapping using kernel functions, and a linear classifier in the feature space

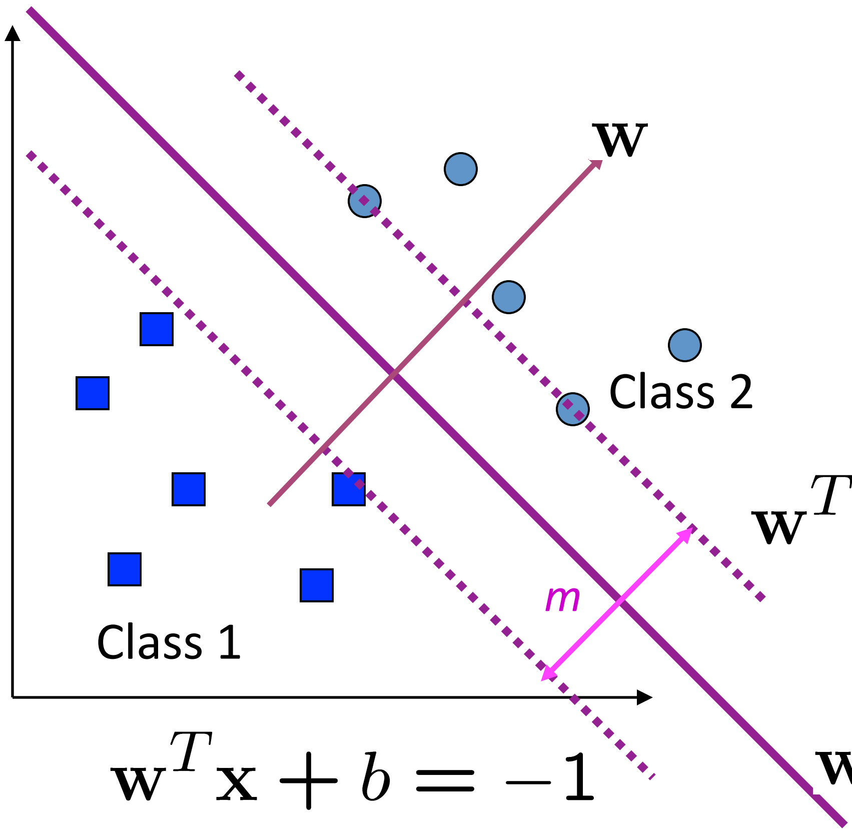


Bomb
Toy
Linear separation based on margin maximization
non-adaptive features from a kernel function
input units e.g. pixels

Supervised Learning: SVM

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

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



$$\mathbf{w}^T\mathbf{x} + b = 1$$

$$\mathbf{w}^T\mathbf{x} + b = -1$$

$$\mathbf{w}^T\mathbf{x} + b = 0$$

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

Supervised Learning: SVM

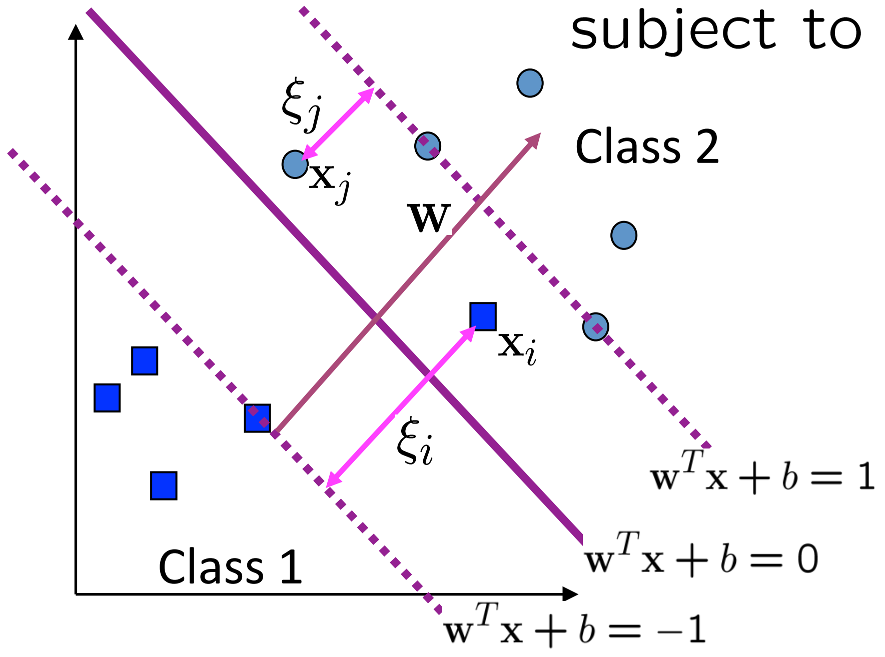
Minimize $\frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\xi_i$

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

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

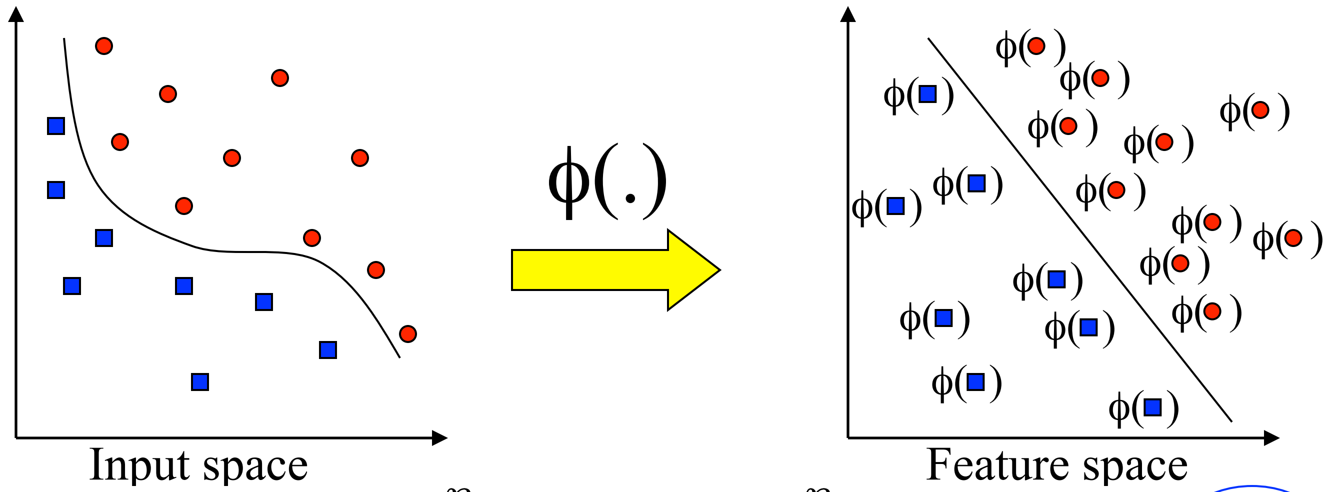
$$\text{subject to } C \geq \alpha_i \geq 0, \sum_{i=1}^{n}\alpha_i y_i = 0$$

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



ξ_j
$\mathbf{x}_j$
$\mathbf{w}$
Class 2
$\mathbf{x}_i$
ξ_i
$\mathbf{w}^T\mathbf{x} + b = 1$
$\mathbf{w}^T\mathbf{x} + b = 0$
Class 1
$\mathbf{w}^T\mathbf{x} + b = -1$

Non-linear Mapping

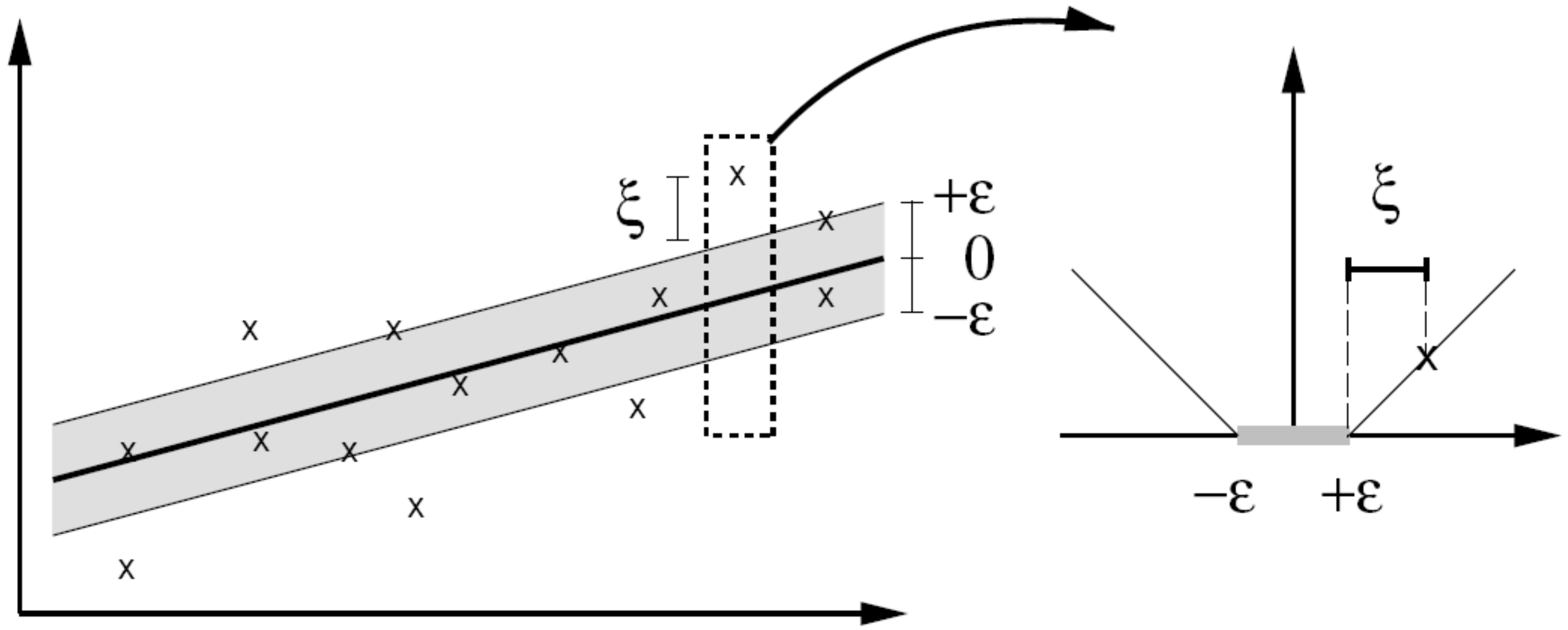


$$\text{max. } W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

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

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

ε-SV Regression Estimation



Formulation as an Optimization Problem

Estimate a linear regression

$$f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b$$

with precision ε by minimizing

$$\begin{array}{ll} \text{minimize} & \tau(\mathbf{w}, \boldsymbol{\xi}, \boldsymbol{\xi}^*) = \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{m} (\xi_i + \xi_i^*) \\ \text{subject to} & (\langle \mathbf{w}, \mathbf{x}_i \rangle + b) - y_i \leq \varepsilon + \xi_i \\ & y_i - (\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \leq \varepsilon + \xi_i^* \\ & \xi_i, \xi_i^* \geq 0 \end{array}$$

for all $i = 1, \ldots, m$.

Dual Problem, In Terms of Kernels

For $C > 0, \varepsilon \geq 0$ chosen a priori,

$$\text{maximize} \quad W(\boldsymbol{\alpha}, \boldsymbol{\alpha}^*) = -\varepsilon \sum_{i=1}^{m} (\alpha_i^* + \alpha_i) + \sum_{i=1}^{m} (\alpha_i^* - \alpha_i) y_i - \frac{1}{2} \sum_{i,j=1}^{m} (\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j) k(\mathbf{x}_i, \mathbf{x}_j)$$

$$\text{subject to} \quad 0 \leq \alpha_i, \alpha_i^* \leq C, \; i = 1, \ldots, m, \; \text{and} \; \sum_{i=1}^{m} (\alpha_i - \alpha_i^*) = 0.$$

The regression estimate takes the form

$$f(\mathbf{x}) = \sum\nolimits_{i=1}^{m} (\alpha_i^* - \alpha_i) k(\mathbf{x}_i, \mathbf{x}) + b,$$

ν-SV Regression

Again, use ν to eliminate another parameter:
Estimate ε from the data s.t. the ν-property holds.

Primal problem: for $0 \leq \nu \leq 1$, minimize

$$\tau(\mathbf{w}, \varepsilon) = \frac{1}{2}\|\mathbf{w}\|^2 + C\left(\nu\varepsilon + 1/m\sum_{i=1}^{m}|y_i - f(\mathbf{x}_i)|_\varepsilon\right)$$

Dual Problem

for $\nu \geq 0, C > 0,$

$$\underset{\boldsymbol{\alpha}^{(*)} \in \mathbb{R}^m}{\text{maximize}}\ W(\boldsymbol{\alpha}^{(*)}) = \sum_{i=1}^{m} (\alpha_i^* - \alpha_i) y_i - \frac{1}{2} \sum_{i,j=1}^{m} (\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j) k(x_i, x_j),$$

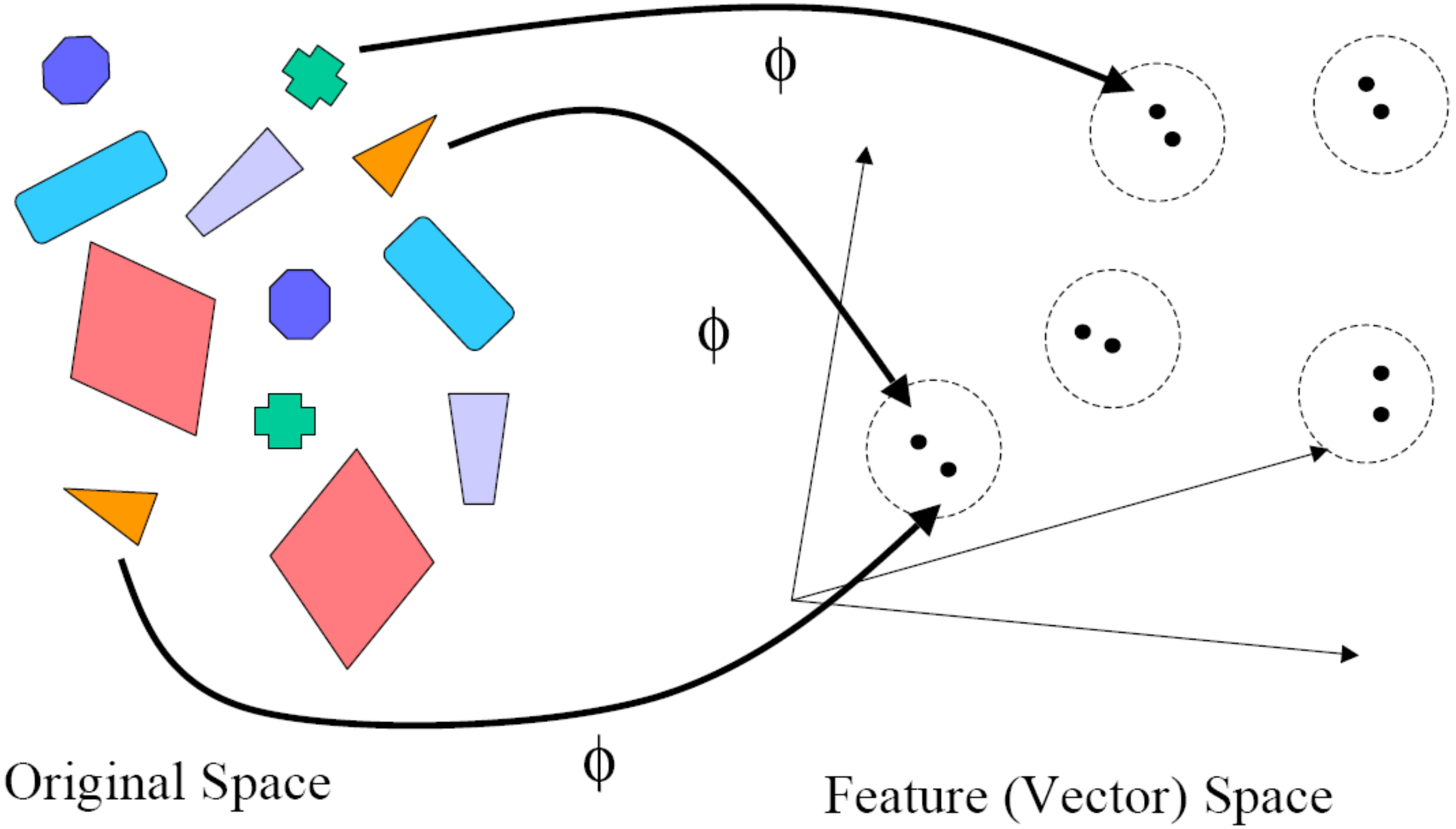
$$\text{subject to } \sum_{i=1}^{m} (\alpha_i - \alpha_i^*) = 0,$$

$$\alpha_i^{(*)} \in \left[0, \tfrac{C}{m}\right],$$

$$\sum_{i=1}^{m} (\alpha_i + \alpha_i^*) \leq C \cdot \nu.$$

$$f(x) = \sum_{i=1}^{m} (\alpha_i^* - \alpha_i) k(x_i, x) + b$$

Kernel Methods : the mapping



Kernel : more formal definition

- A kernel k(x,y)
  - is a similarity measure
  - defined by an implicit mapping ϕ,
  - from the original space to a vector space (feature space)
  - such that: $k(x,y)=\phi(x)\bullet\phi(y)$
- This similarity measure and the mapping include:
  - Invariance or other a priori knowledge
  - Simpler structure (linear representation of the data)
  - The class of functions the solution is taken from
  - Possibly infinite dimension (hypothesis space for learning)
  - … but still computational efficiency when computing $k(x,y)$

Valid Kernels

- The function $k(x,y)$ is a valid kernel, if there exists a mapping ϕ into a vector space (with a dot-product) such that k can be expressed as $k(x,y)=\phi(x)\bullet\phi(y)$
- Theorem: $k(x,y)$ is a valid kernel if k is positive definite and symmetric (Mercer Kernel)
  - A function is P.D. if $\int K(\mathbf{x},\mathbf{y}) f(\mathbf{x}) f(\mathbf{y}) d\mathbf{x} d\mathbf{y} \geq 0 \quad \forall f \in L_2$
  - In other words, the Gram matrix **K** (whose elements are $k(x_i,x_j)$) must be positive definite for all x_i, x_j of the input space
  - One possible choice of $\phi(x)$: $k(\bullet,x)$ (maps a point x to a function $k(\bullet,x)$ $\rightarrow$ feature space with infinite dimension!)

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

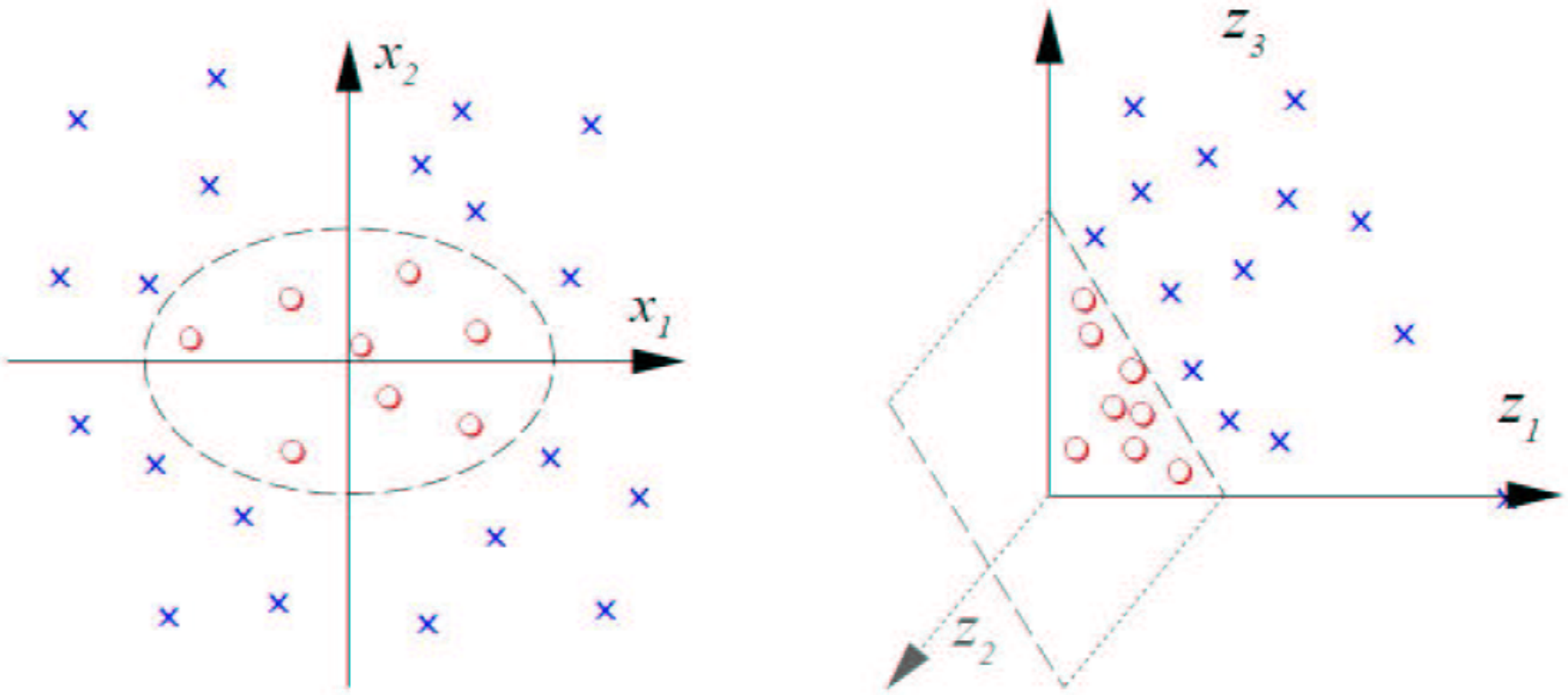
$$K(\mathbf{x},\mathbf{y}) = \tanh(\kappa\mathbf{x}^T\mathbf{y} + \theta)$$

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

Example of Kernels (I)

- Polynomial Kernels: $k(x,y)=(x \cdot y)^d$
  - Assume we know most information is contained in monomials (e.g. multiword terms) of degree *d* (e.g. d=2: x_1^2, x_2^2, $x_1 x_2$)
  - Theorem: the (implicit) feature space contains all possible monomials of degree d (ex: *n*=250; d=5; dim F=10^{10})
  - But kernel computation is only marginally more complex than standard dot product!
  - For $k(x,y)=(x \cdot y+1)^d$, the (implicit) feature space contains all possible monomials up to degree d !

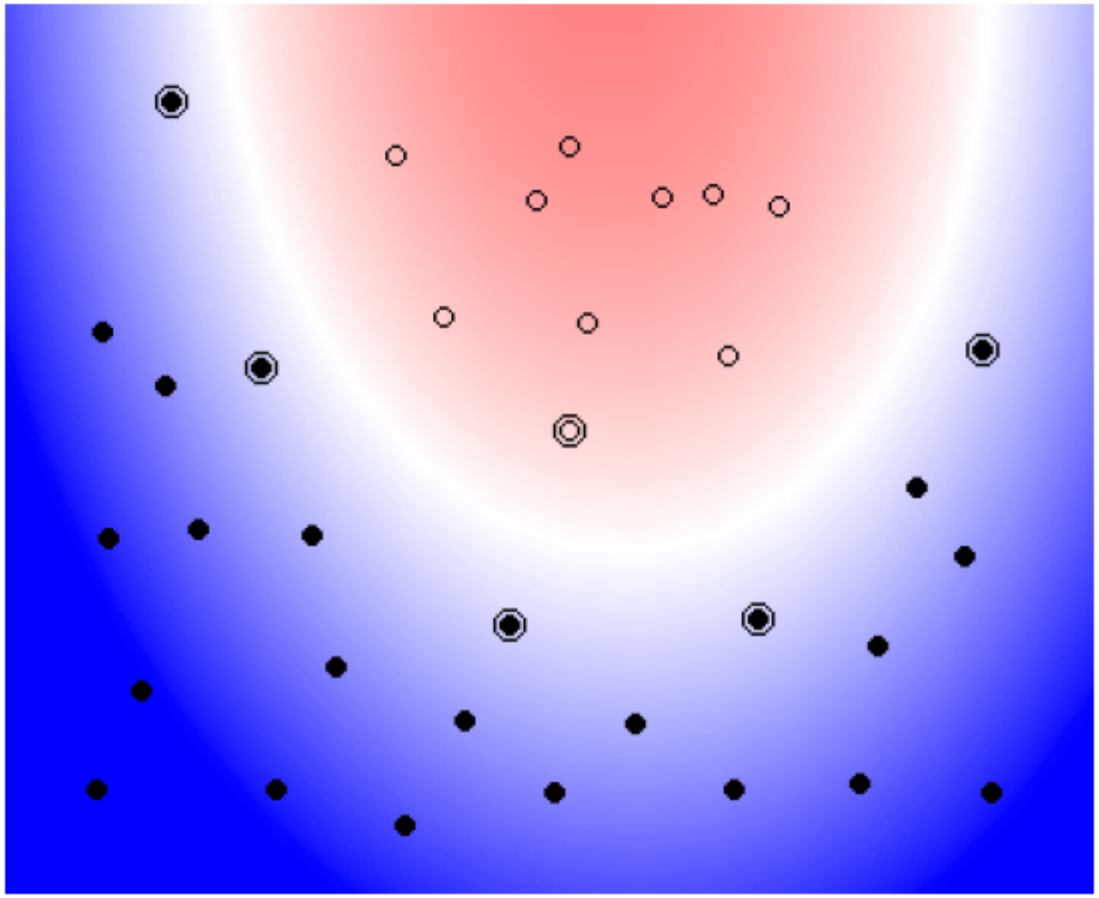
$$\Phi : \mathbb{R}^2 \to \mathbb{R}^3$$
$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2}\, x_1 x_2, x_2^2)$$



Example: SVM with Polynomial of Degree 2

Kernel: $K(\vec{x}_i, \vec{x}_j) = [\vec{x}_i \cdot \vec{x}_j + 1]^2$

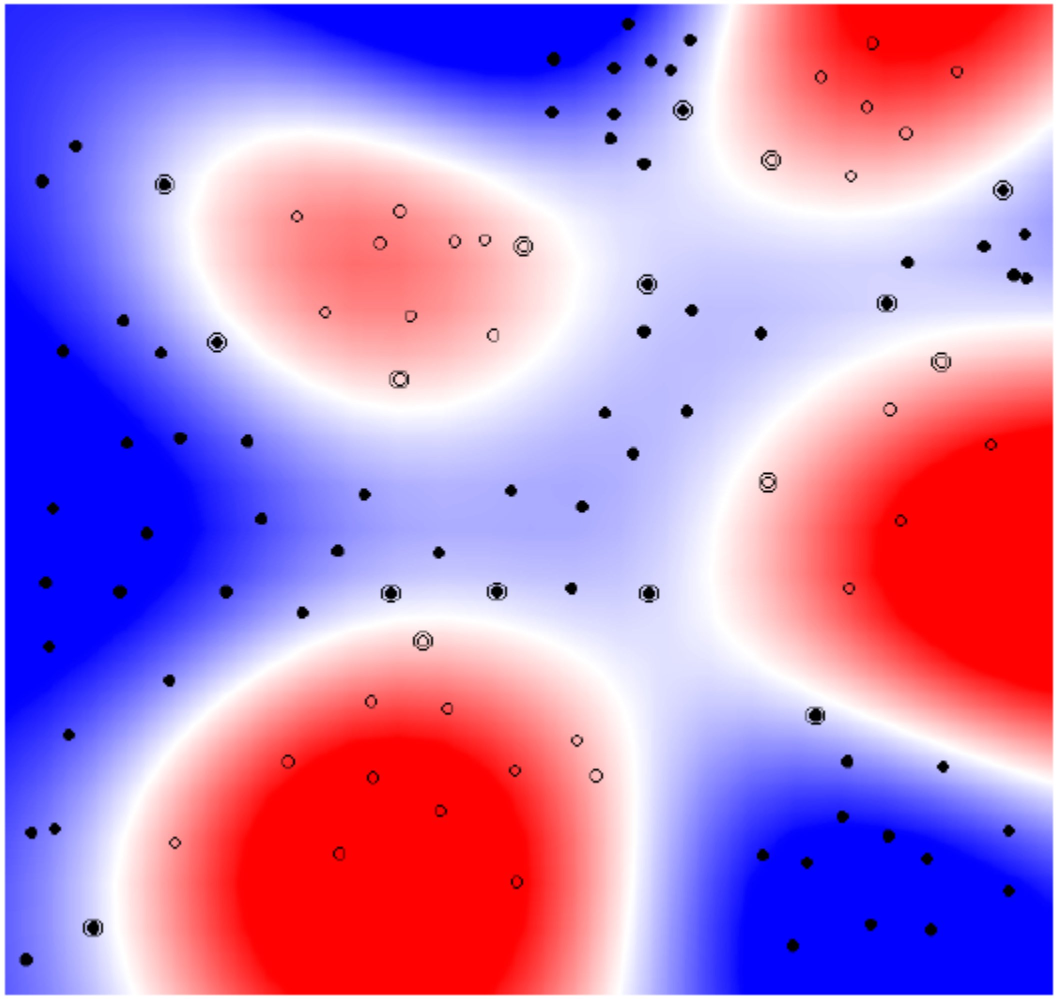
plot by Bell SVM applet



Example: SVM with RBF-Kernel

Kernel: $K(\vec{x}_i, \vec{x}_j) = \exp(-|\vec{x}_i - \vec{x}_j|^2 / \sigma^2)$

plot by Bell SVM applet



Selecting a Kernel

Things to take into consideration:

- kernel can be thought of as a similarity measure
    - examples in the same class should have high kernel value
    - examples in different classes should have low kernel value
    - ideal kernel: equivalence relation $K(\vec{x}_i, \vec{x}_j) = sign(y_i y_j)$
- normalization also applies to kernel
    - relative weight for implicit features
    - normalize per example for directional data

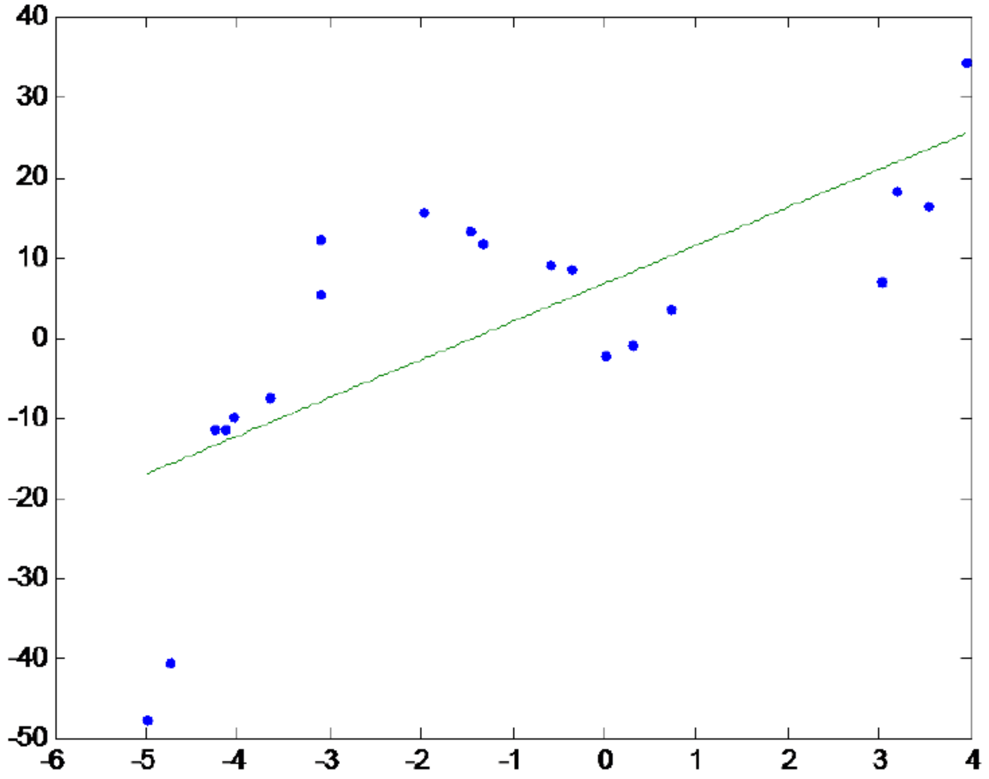
$$K(\vec{x}_i, \vec{x}_j) = \frac{K(\vec{x}_i, \vec{x}_j)}{\sqrt{K(\vec{x}_i, \vec{x}_i)}\sqrt{K(\vec{x}_j, \vec{x}_j)}}$$

    - potential problems with large numbers, for example polynomial kernel $K(\vec{x}_i, \vec{x}_j) = [\vec{x}_i \cdot \vec{x}_j + 1]^d$ for large d

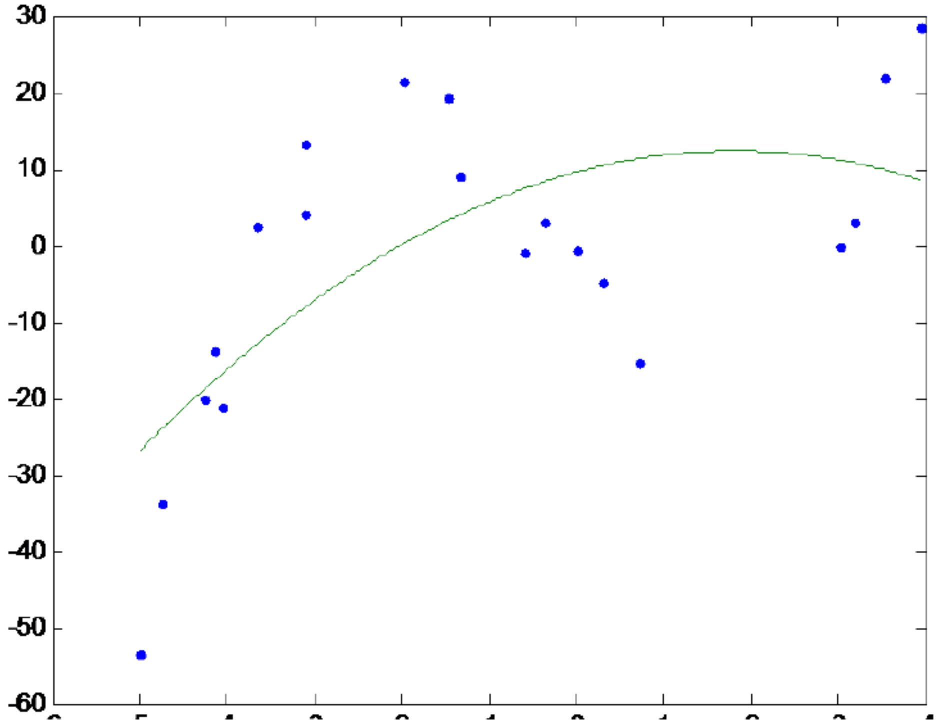
Selecting a Kernel

- There is no absolute rules for choosing the right kernel, adapted to a particular problem
- Kernel design can start from the desired feature space, from combination or from data
- Some considerations are important:
  - Use kernel to introduce a priori (domain) knowledge
  - Be sure to keep some information structure in the feature space
  - Experimentally, there is some “robustness” in the choice, if the chosen kernels provide an acceptable trade-off between
    - simpler and more efficient structure (e.g. linear separability), which requires some “explosion”
    - Information structure preserving, which requires that the “explosion” is not too strong.

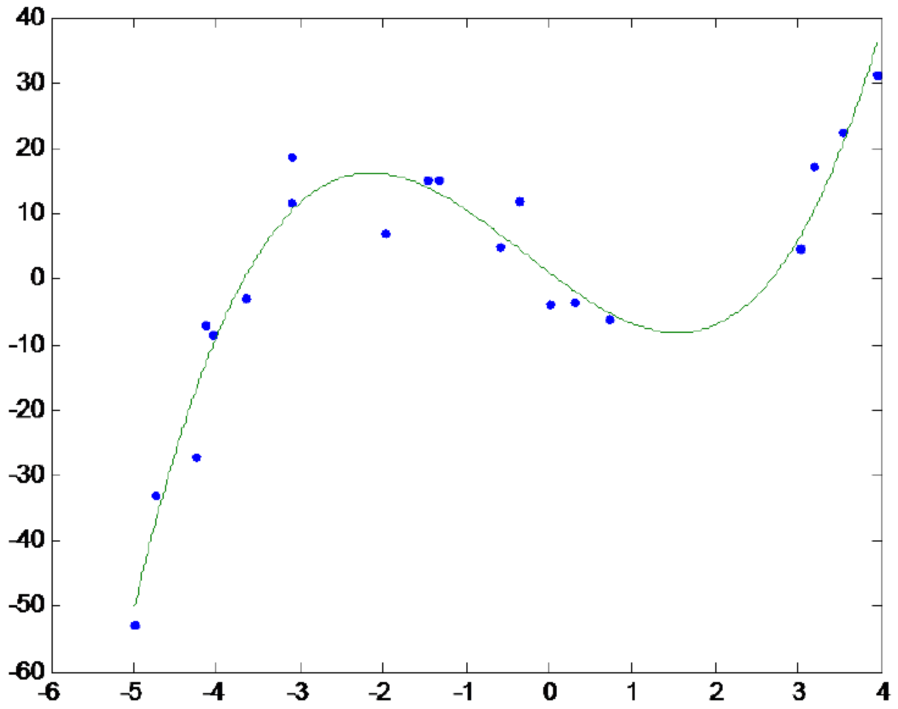
Fitting a degree-1 polynomial



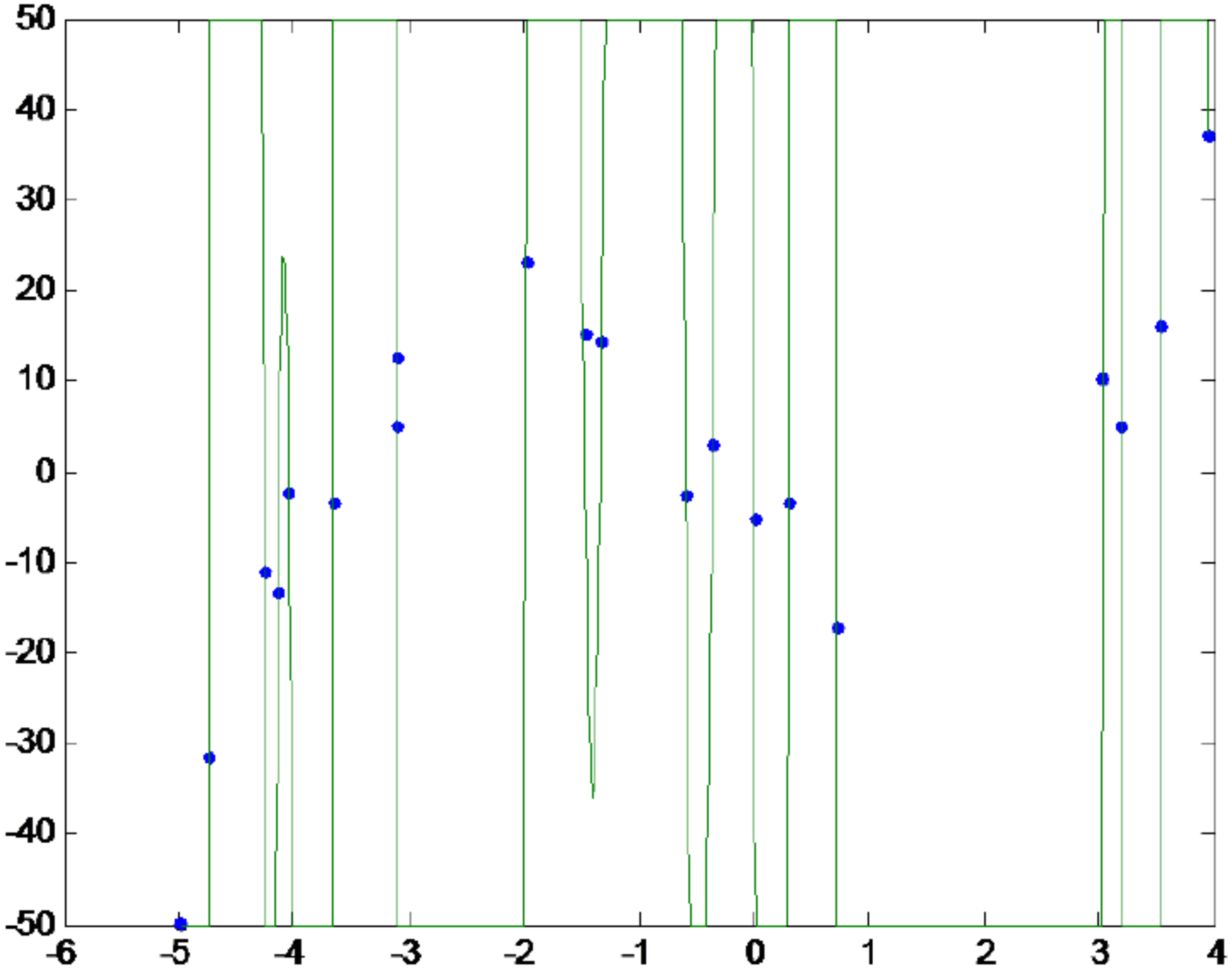
Fitting a degree-2 polynomial



Fitting a degree-3 polynomial



Fitting a degree-19 polynomial



Random-Walk Kernel (Positive K)

Given a base kernel K let P_{ij} be the probability $P(x_i \rightarrow x_j)$

$$P^t = (D^{-1}K)^t \qquad D_{ii} = \sum_k K_{ik}$$

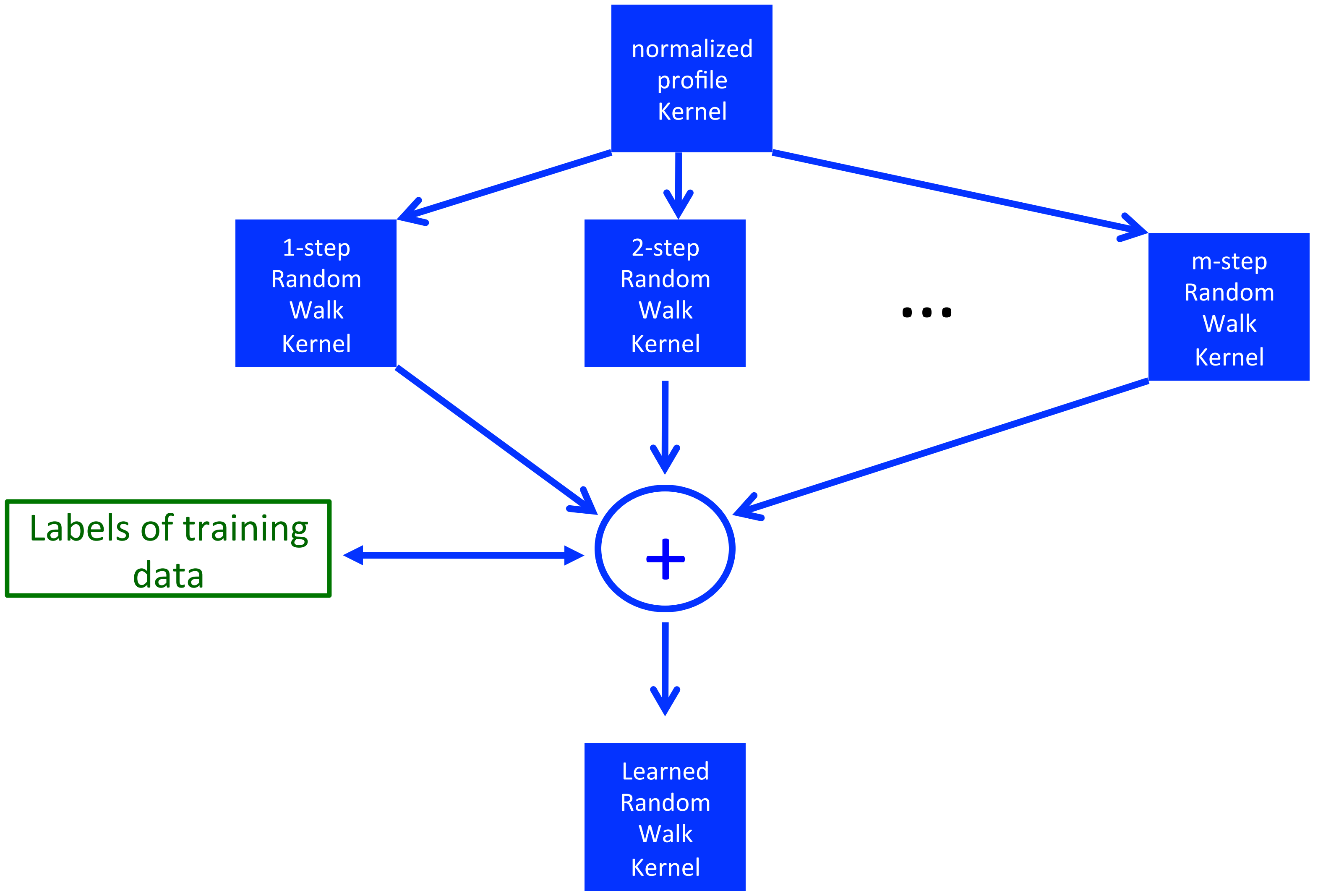
$$L = D^{-\frac{1}{2}} K D^{-\frac{1}{2}} \qquad L = U \Lambda U^T$$

$$\tilde{L} = U \Lambda^t U^T \qquad \tilde{K} = \tilde{D}^{-\frac{1}{2}} \tilde{L} \tilde{D}^{-\frac{1}{2}}$$

$\tilde{D}$ is a diagonal matrix with $\tilde{D}_{ii} = \tilde{L}_{ii}$

$$\tilde{K} = \tilde{D}^{-\frac{1}{2}} D^{\frac{1}{2}} P^t D^{-\frac{1}{2}} \tilde{D}^{-\frac{1}{2}}$$

Learned Random-Walk Kernel (Min et. al,. 2009)


normalized profile Kernel
1-step Random Walk Kernel
2-step Random Walk Kernel
...
m-step Random Walk Kernel
Labels of training data
+
Learned Random Walk Kernel

Learned Random-Walk Kernel

$$
\begin{aligned}
min_{\mu}\, max_{\alpha} \quad & 2\alpha^T\mathbf{1} - \alpha^T(K_{tr} \otimes yy^T)\alpha, \\
s.t. \quad & \alpha^T y = 0 \\
& \mathbf{0} \leq \alpha \leq C\mathbf{1}, \\
& K = \mu_0 \tilde{K}^0 + \textstyle\sum_{k=1}^{m} \mu_k \tilde{K}^k, \\
& \textstyle\sum_{k=0}^{m} \mu_k = 1, \\
& \mu_k \geq 0, \qquad k = 0, \ldots, m,
\end{aligned}
$$

SVM learning

Kernel learning

$\tilde{K}^0$ is the base kernel for deriving the improved random-walk kernel, $\tilde{K}^k$ is the random-walk kernel with a k-step random walk, and, m, is the maximal number of random steps performed.

Proofs (Vapnik thought I was wrong)

$$\begin{aligned} min_{\alpha,t} \quad & t, \\ s.t. \quad t \geq \alpha^T(\tilde{K}_{tr}^k \otimes yy^T)\alpha - 2\alpha^T\mathbf{1}, & \quad k = 0, \ldots, m, \\ \alpha^T y = 0 & \\ \mathbf{0} \leq \alpha \leq C\mathbf{1}, & \end{aligned}$$

$$min_{\mu:\ \ \mu \geq \mathbf{0},\ \ \sum_{k=0}^m \mu_k = 1} \ max_{\alpha:\ \ \alpha^T y = 0,\ \ \mathbf{0} \leq \alpha \leq C\mathbf{1}} \ 2\alpha^T\mathbf{1} - \alpha^T[(\sum_{k=0}^m \mu_k \tilde{K}_{tr}^k) \otimes yy^T]\alpha$$

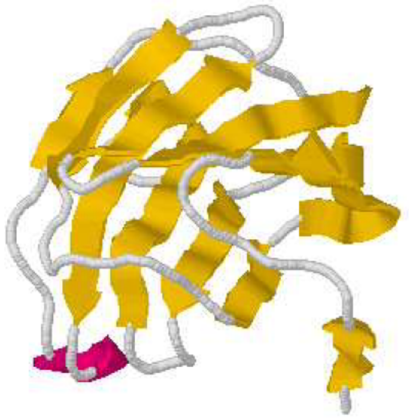
$$= max_{\alpha:\ \ \alpha^T y = 0,\ \ \mathbf{0} \leq \alpha \leq C\mathbf{1}} \ min_{\mu:\ \ \mu \geq \mathbf{0},\ \ \sum_{k=0}^m \mu_k = 1} \ 2\alpha^T\mathbf{1} - \alpha^T[(\sum_{k=0}^m \mu_k \tilde{K}_{tr}^k) \otimes yy^T]\alpha$$

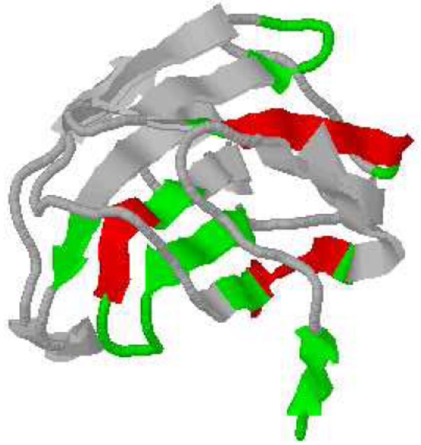
$$\boxed{\begin{aligned} &= max_{\alpha:\ \ \alpha^T y = 0,\ \ \mathbf{0} \leq \alpha \leq C\mathbf{1}} \ min_{\mu:\ \ \mu \geq \mathbf{0},\ \ \sum_{k=0}^m \mu_k = 1} \ \sum_{k=0}^m \mu_k [2\alpha^T\mathbf{1} - \alpha^T(\tilde{K}_{tr}^k \otimes yy^T)\alpha] \\ &= max_{\alpha:\alpha^T y = 0, \mathbf{0} \leq \alpha \leq C\mathbf{1}} min_k [2\alpha^T\mathbf{1} - \alpha^T(\tilde{K}_{tr}^k \otimes yy^T)\alpha] \end{aligned}}$$

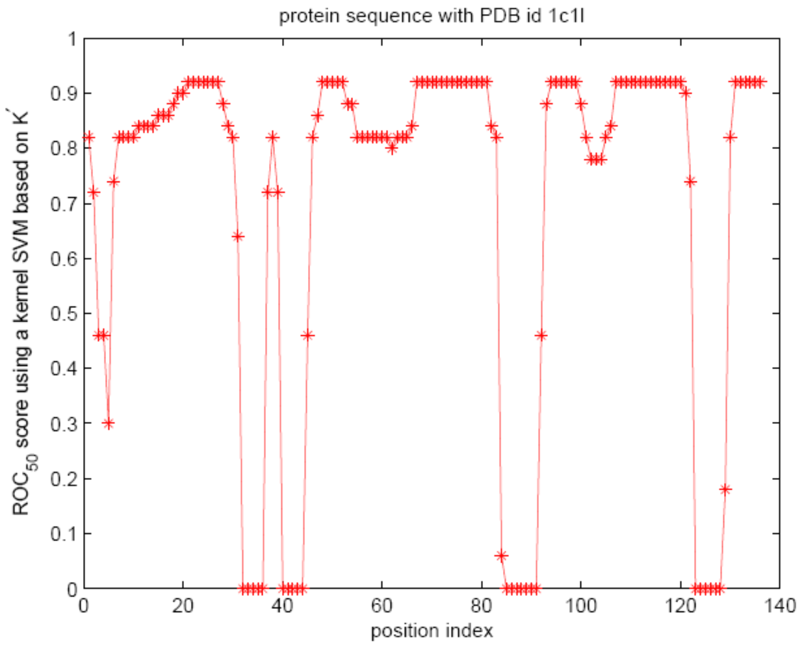
$$= max_{\alpha,t:\ \ \alpha^T y = 0,\ \ \mathbf{0} \leq \alpha \leq C\mathbf{1}, t \leq 2\alpha^T\mathbf{1} - \alpha^T(\tilde{K}_{tr}^k \otimes yy^T)\alpha} \quad t$$

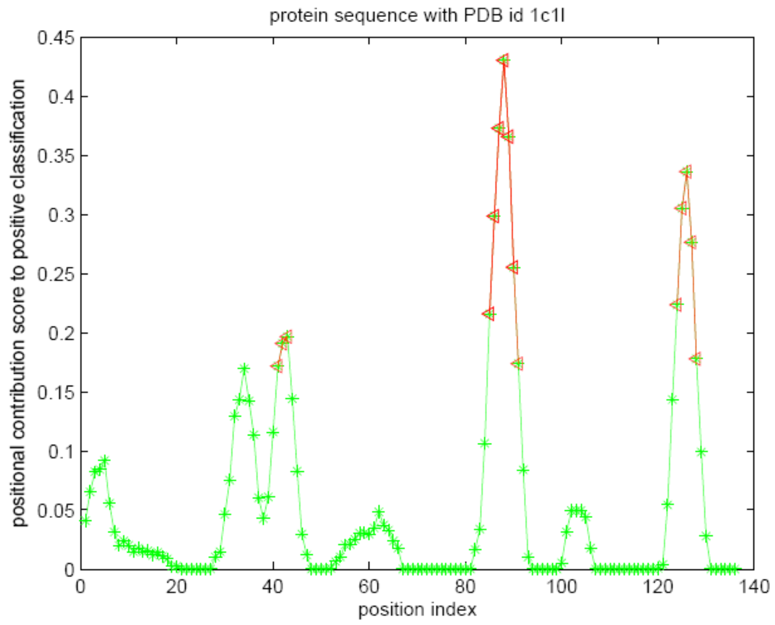
$$= min_{\alpha,t:\alpha^T y = 0, \mathbf{0} \leq \alpha \leq C\mathbf{1},\ \ t \geq \alpha^T(\tilde{K}_{tr}^k \otimes yy^T)\alpha - 2\alpha^T\mathbf{1}} \quad t$$

Discovered Protein Sequence Motifs (PDB id 1c1l)

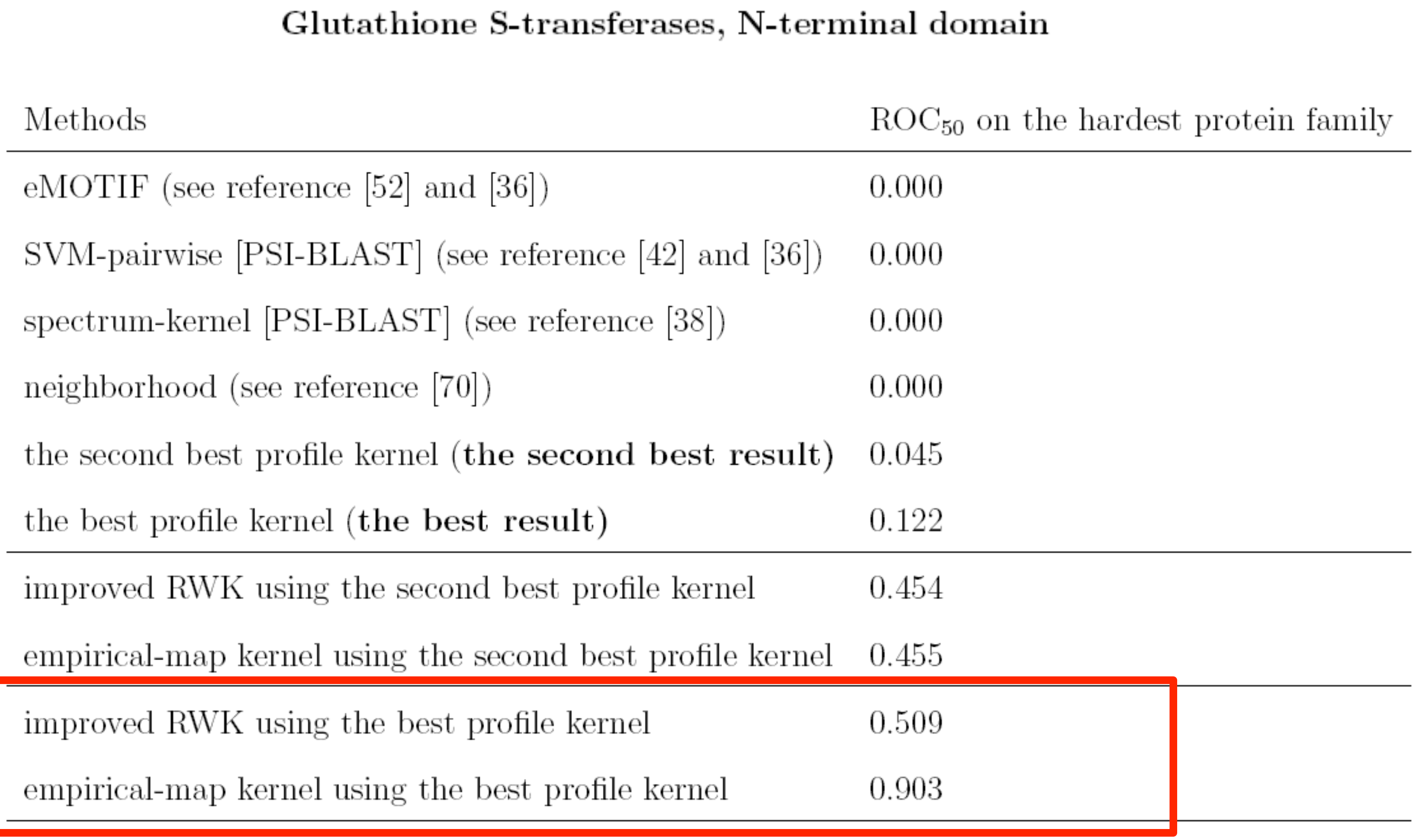





protein sequence with PDB id 1c1l
ROC$_{50}$ score using a kernel SVM based on K´
position index



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

Debate on (Artificial) Human Learning: Connectionism or Symbolism?

- Multi-stage information processing
- Reasoning and perception
- Mathematics
- Theorem proving
- Logic
- Creative thinking

Conclusions

- Machine learning is not data mining or statistics, and it has some many hard-core computer science problems remaining to solve
- Machine learning has already come into our everyday life
- Machine learning still has a long way to go
- Human intelligence is nothing but more “complex” systems
- We need biologically plausible models to build machine brains

The Ultimate Future of Machine Learning

We Are "Equal"

Mike Robort

Williams Gates Professor of Biomedical Informatics, Molecular Biophysics & Biochemistry and Computer Science
A.B. 2039, Harvard University
Ph.D. 2043, Cambridge University
Joined Yale Faculty 2047

Professor Robort does research in the new field of Disease-informatics. He won the Nobel Prize in Physiology for discovering ABCBBCNBC007 for the no-pain one-smell treat of cancer in 2067.

Acknowledgement

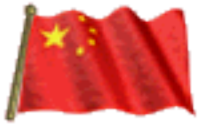
University of Toronto

- Geoffrey Hinton
- Anthony Bonner
- Sam Roweis
- Radford Neal
- David Stanley

Yale University

- Mark Gerstein
- Chao Cheng
- Koon-Kiu Yan
- Pedro Alves
- Gang Fang

The End



Thank You