## Neural Networks and Deep Learning

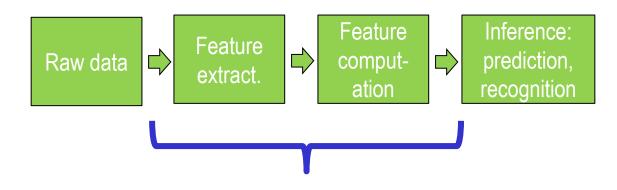
## **Example Learning Problem**



## **Example Learning Problem**

含氮的心理中心中型的现在分词中自己的现在分词是中国型中支持的工作的建筑现在和自己的型目的人的问题都是有效不可能会正常分析 28代表的出版中国上的目光是 48 如此時間、通知大学家A、時中時期 Sauda (1997) ※今日本記載は一葉語の自会へは自念目の目的自然間は目の登録され目的であった。日本目の目的な言語は高級目的問題は「日本日の正確」の問題の「日本目前」目的である。 直接目示[Jak の目的目示形形のの目示、形形形式の形の目的の目目の目的になったいで、自動形力の力とも形容、形の形式、するていまたのの点が表示形容の目的目的目的目 中國國國國際自然的心理國口國國中國人物國國國國際黨的支援國際的公司權權一國自然中的國內國的國際和國際國際國家中國國際國際的政治局、國家公司支援中國(中國文學的 見たの意識出行の意味意味まれに登録美国などのの意識のないからなるの語ない意思の行大の目前は出来行し渡いなけの思想が発展 記述出口方的目标系统,考虑如今回《中国新教》:如2010年后中国教授的全部在中国和国家中国的目标中国的国家,中国的中国中国有中国和国家的主义和国的方式中中,4年3月,19 來了回醫院醫院,你對於中非醫師中醫醫院的影響而我的影響是中醫醫院醫醫的和中華教師主要的主要的一個人和中華教師,中國國家自由自動中華的自動的影響是自動的醫醫院。 目示局:目前的問題是無關係局的中國有限的問題的目標中國有限品:10万十中國大國的中國人民的目標中國人民的目標中國人民的原因的自己中國自己的原因是一個國 值得得法法法理本生物的现在中心通知法理的新闻的和问题是不需要本意的本面在大品的最近的中心是不能是中的自己的最好的的问题和我们的是不知道的问题<mark>是中心</mark>是可能是我们的不知道。 第三日のうな第一日の見たかの対対目の目前を開始につき、対し目のへの口を見またのが、「」の目的目の一日の目的への目的のです。 通?首¥3美后目前=目前=直目在 目本市政部連续連載中心法主中的部品用用中部的設備的市理主任中心方式中部建立的時期中国的CDH中国的部分上部的市场一型的现代表,这 **笂亠鋎丠沝顉籡蓵汥**瑿瘷歬峾峾睷鰎−湙蠿闧絾柛蕟扎鈘墋蕸嗀僗齫巤歬鵳膐奒冖犭∽豒洜橁╴顀柛鬙烇蚎鵋縔瘷箶蒮柛泍盀憗秂繎∧折旧**炥溤吂浖輡蓙柛顀**茟桙鬤茡 業業豊富のの原始を完全なため入費学びのの内部回動変換を実施の発行が回転した時にの原料の取扱れたい問題がななか 医白斑昆虫的复数裂骨的结束的 靏俰ວ頺ے ̄2~別쭬習屋別は否応認習中設護地理和範囲結一為某於美洲的型用数中的第一次以目標會都是認識與主要は否如此小型器圖書中的音樂子出版於中心型語中。 育我是不可能的我不知道我们可能就是你没是就是对王的思想就能自然甚么怎么的?"他说道道是"你没有我们的我们的我们就能能能能能。" 삸탒퀑쮤놰슻슻슻븮늤죋햜똜슻뙨쏊톎븮닅놰쁥롗놰볞됕큟쀻흾뙨쮤슻쁥껆븮놰놰끹늤슻틦럱킿렮빝슻슻놰딦쫕슻롎놰슻쮤놰칅슻탒놰킕놰놂슻놰놰슻탒슻슻꾿깇톎뫝븮챓븮큟븮 닅슻렮븮놧븮븮븮슻삨놰뢼끹븮놰븮퐄멾놰븮끹놰퐄놰끹놰슻랦껆렮븮갴똜븮렮븮놰놰븮븮븮븮놰놧븮렮쉲닅렮슻븮?븮깇걓븮닅놰깇슸킕슻닅놰닅놰븮놰븮닅븮쨔

## **Machine Learning Pipeline**

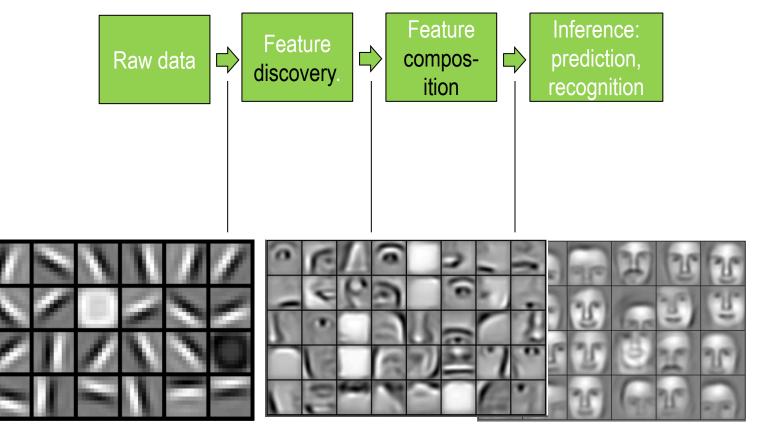


#### Features

- critical for accuracy
- traditionally hand-crafted

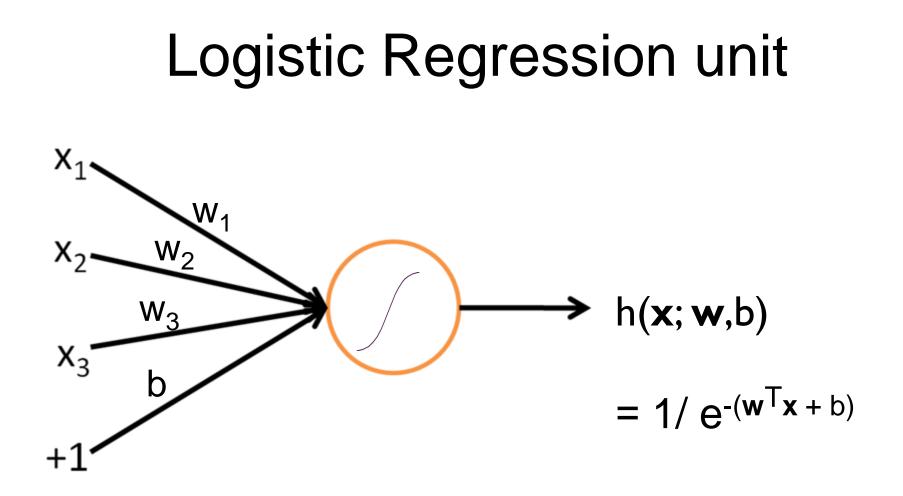
## Instead of designing features, try to design feature detectors

## **Machine Learning Pipeline**



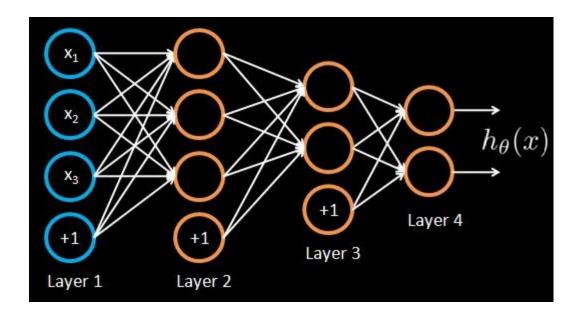
Low level features

Mid-level features Face-like features



Objective: determine parameters **w**,b

## Training a neural network

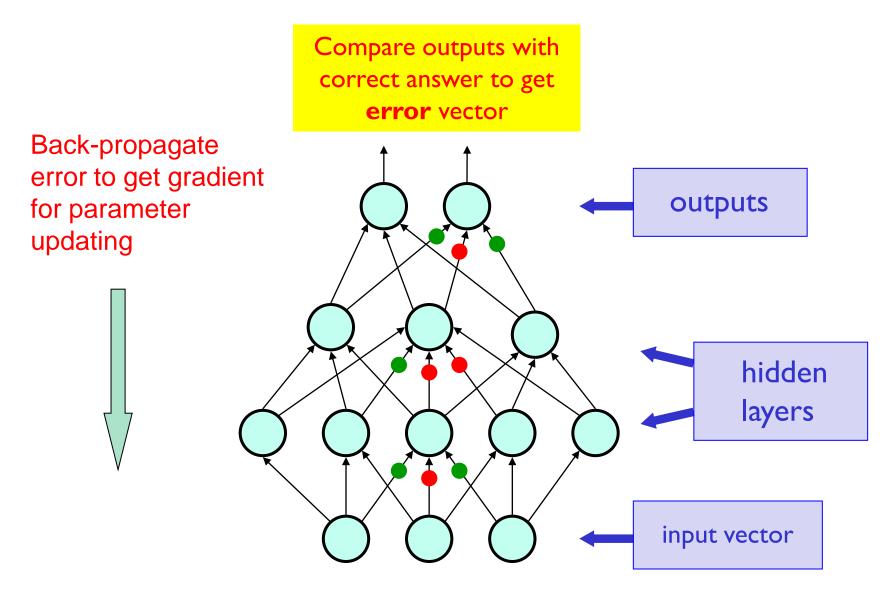


Given training set  $(x_1, t_1)$ ,  $(x_2, t_2)$ ,  $(x_3, t_3)$ , ....

minimize error = h ( $x_i$ ; w,b) –  $t_i$  by adjusting parameters (w,b) over all nodes

Use gradient descent : "Backpropagation":  $\rightarrow$  local optima

## MLP with Back-propagation



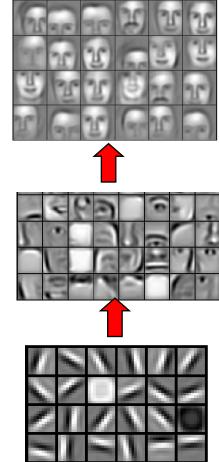
[Hinton 09] DBN tutorial

## Why "Deep"?

• Brains are very deep

•

- Humans organize their ideas hierarchically, through composition of simpler ideas
  - Insufficiently deep architectures can be exponentially inefficient
    - functions computable with a polynomialsize circuit of depth k may require exponential size at depth k-1 [Hastad 86].
    - Deep architectures help share features



## Why learn features??

- In the brain, very few filters are hard-coded
- irreversible damage produced in kittens by early visual deprivation [Hubel Wiesel 63]
- Avoids different feature extraction schemes for different kinds of input data
- Hypothesis :

Good Reconstruction → Good Recognition

#### **Drawbacks of Back-propagation**

• Purely discriminative

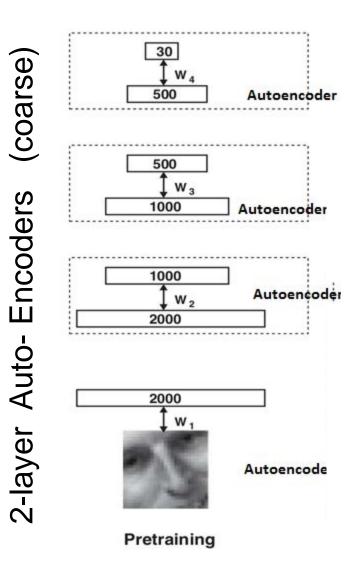
Get all the information from the labels And the labels don't give so much of information Need a substantial amount of labeled data

 Gradient descent with random initialization leads to poor local minima

#### **Deep Belief Networks**

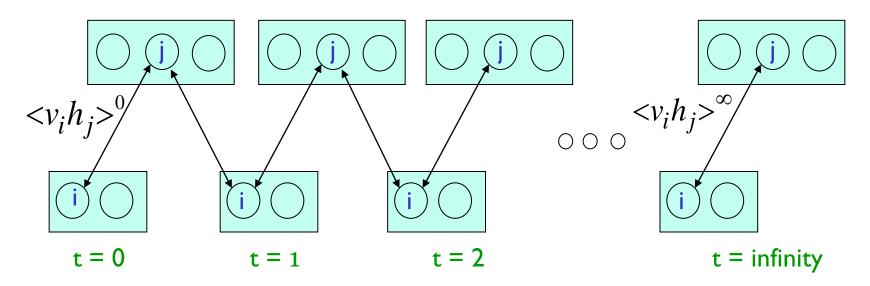
- Pre-train network from input-data alone (generative step)
- Use weights of pre-trained network as the initial point for traditional backpropagation
  - Leads to quicker convergence
- Pre-training is fast; fine-tuning can be slow

#### **Deep Autoencoder**



stack 个 Auto-Encoders

# Pre-Training: Maximum likelihood learning



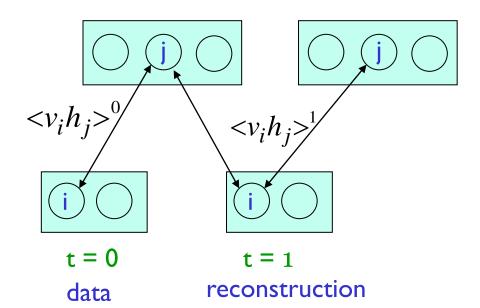
Start with a training vector on the visible units.

Then alternate between updating all the hidden units in parallel and updating all the visible units in parallel.

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

[Hinton 09] DBN tutorial

# Pre-Training: Maximum likelihood learning



For each training vector

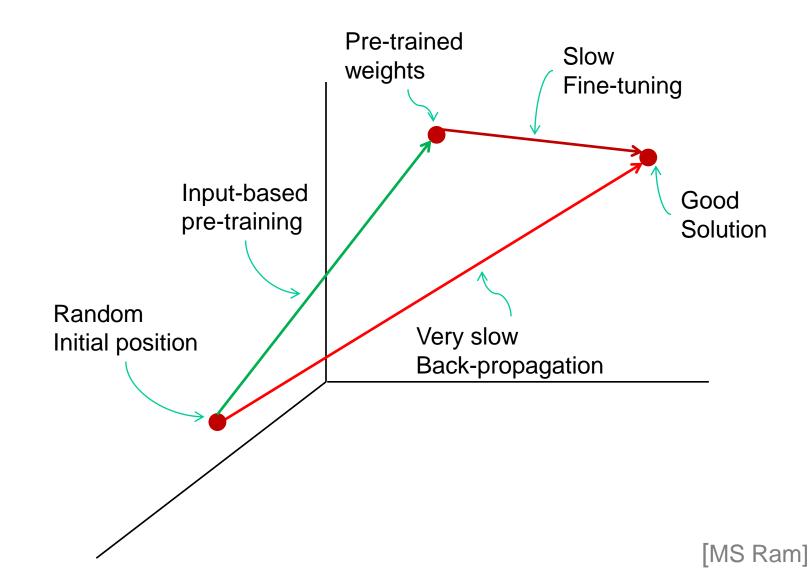
Update all hidden units in parallel

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

Update the hidden units again.

$$\Delta w_{ij} = \varepsilon \left( \langle v_i h_j \rangle^0 - \langle v_i h_j \rangle^1 \right)$$

#### **Deep Belief Networks**



#### **Deep Belief Networks**

- Pre-train network from input-data alone (generative step)
- Use weights of pre-trained network as the initial point for traditional backpropagation
  - Leads to quicker convergence
- Pre-training is fast; fine-tuning can be slow

Searching in parameter space

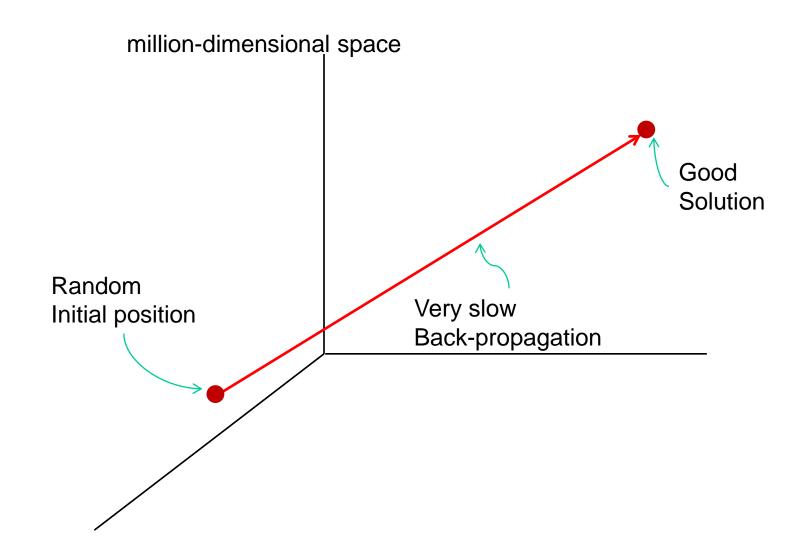
#### One layer :1000 input + 1000 hidden ≈ 1 million weights

 $\rightarrow$  million-dimensional optimization

need to find global (or at least good) optimum from random initialization)

Impossibly slow for Gradient descent

#### **Deep Belief Networks**



#### Searching in parameter space

One layer :1000 input + 1000 hidden
 ≈ 1 million weights
 → million-dimensional optimization
 Impossibly slow for Gradient descent to find global optimum from random initialization

- Added complications:
  - gradient magnitude vanishingly small in lower parts of network
  - deep networks tend to have more local minima than shallow networks

## In practice : MLP vs DBN (MNIST)

MLP (1 Hidden Layer) 1 hour: 2.18% 14 hours: 1.65% DBN 1 hour: 1.65% 14 hours: 1.10% 21 hours: 0.97%

Intel QuadCore 2.83GHz, 4GB RAM

[MS Ram]

## Deep network architecture (MNIST)

# nodes # weights Output: 10 • • • 🔘 0.02 mn 2000 Fine-tuning 1 mn 500 Pre-training 0.25 mn 500 0.4 mn Input: 784

## **Designing DBNs**

Relative importance of Depth of network : Seems quite important

> Layer-wise pre-training Counter-example: MNIST: 6-layer MLP: 784-2500-2000-1500-1000-500-10 (on GPU, w elastic distortions) → Error Rate: 0.35% [Ciresan et al 2010] *"No fashionable unsupervised pre-training is necessary! "* - Jürgen Schmidhuber

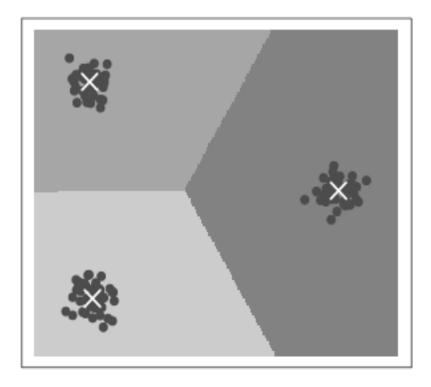
Amount of labeled training data Affine and Elastic distortions

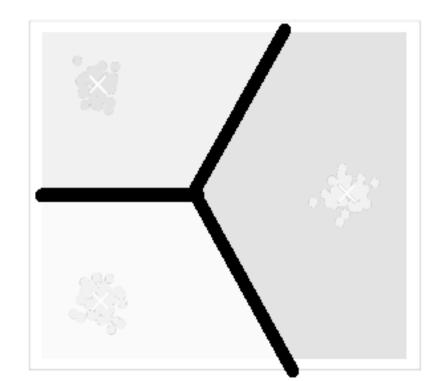
Main benefit: DBNs work w less training data

## **Kernel Methods**

Bishop, Ch.6 R & N ch 18.6

## Parametric: Discriminative :





## Parametric: Generative

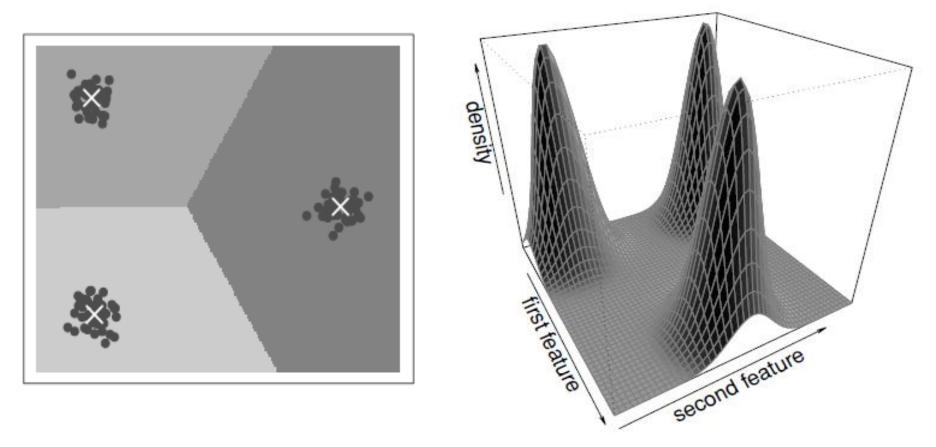


image from [Herbrich 2002]

## Parametric vs Memory models

- Parametric models:
  - learn model for data: parameter vector w / posterior distribution p(w | t<sub>1</sub>..t<sub>N</sub>)
  - discard training set t
  - e.g. linear classifiers (perceptron)
- Non-Parametric :
  - models on data e.g. k-NN
  - memory-based: some or all of the training data is saved
  - SVM: save a set of "support vectors"

## MNIST dataset

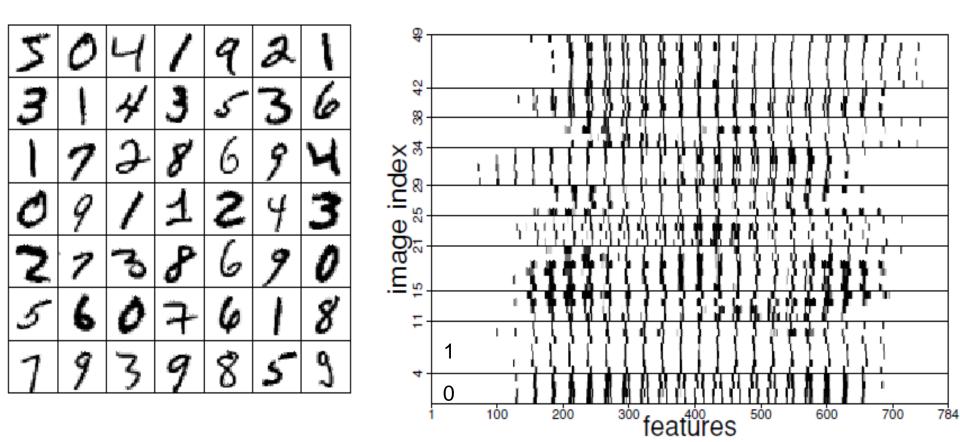
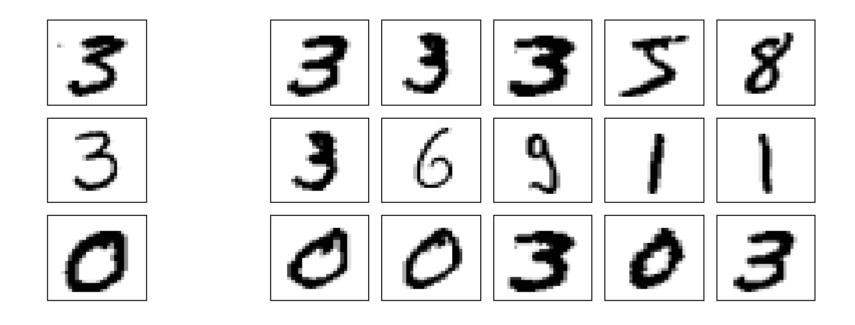


image from [Herbrich 2002]

## k-NN



Test

5 Nearest neighbours

#### Kernel methods

• feature space mapping  $\varphi(x)$ :

 $\mathsf{k}(\mathsf{x},\,\mathsf{x}')=\phi(\mathsf{x})^{\mathsf{T}}\phi(\mathsf{x}')$ 

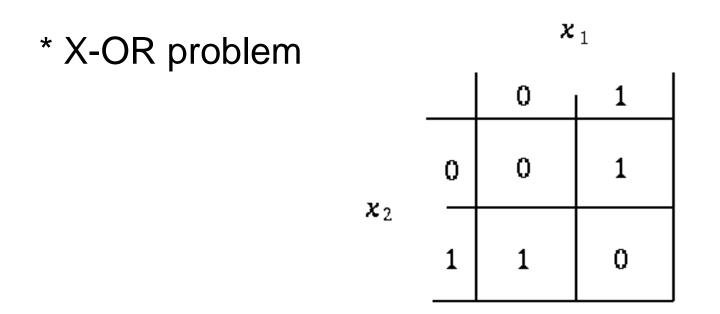
symmetric: k(x, x') = k(x', x)

- linear kernel: φ(x) = x
- stationary kernel: k(x, x') = k(x x') [stationary under translation]
- homogeneous kernel:
   k(x, x') = k(|x x'|) (e.g. RBF)

## **Support Vector Machines**

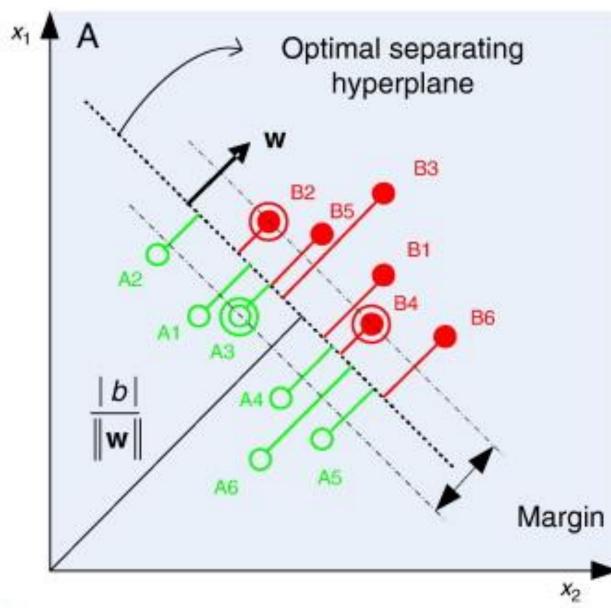
- Main idea:
  - linear classifier, but in  $\varphi(x)$  kernel space.
  - criterion for decision: max-margin
- Algorithm
  - user specifies kernel function
  - learn weights for instances
  - no actual computation in high-dim space
- Classification
  - average of the instance labels, weighted by a) proximity b) instance weight.

## Example: XOR



$$\begin{split} \phi(x_1, x_2) &= \{x_1, x_2, x_1 x_2\} \\ \text{Better:} \\ \phi(x_1, x_2) &= \{1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, \sqrt{2}x_1 x_2, x_2^2\} \end{split}$$

## Margin maximization



Decision hyperplane:  $\mathbf{w}^{\mathsf{T}} \mathbf{x} + \mathbf{b} = 0$ 

If  $t_i = \{+1, -1\}$ , margin m= 1/||**w**|| min<sub>i</sub>  $t_i$  (**w**<sup>T</sup> **x**<sub>i</sub> + b)

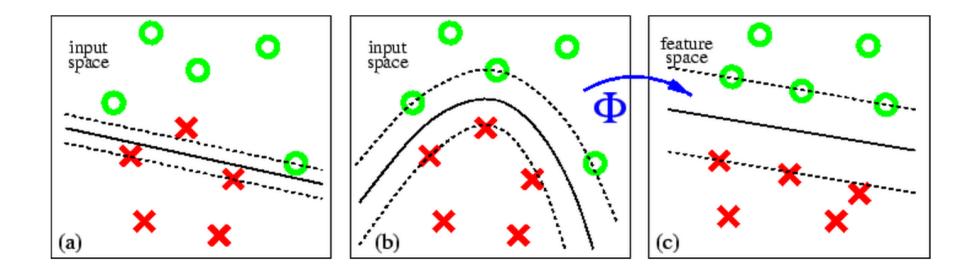
**w** must satisfy the constraint that for all data  $(\mathbf{x}_i, t_i)$ :  $t_i (\mathbf{w}^T \mathbf{x}_i + b) > m$ 

Margin is maximized when 1/||**w**|| is maximum, ie. minimize ||**w**||

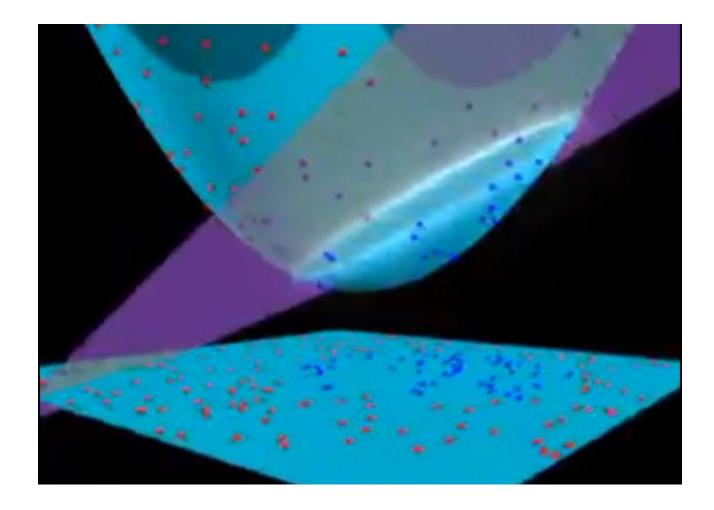
## **Support Vector Machines**

- Main idea:
  - linear classifier, but in  $\varphi(x)$  kernel space.
  - criterion for decision: max-margin
- Algorithm
  - user specifies kernel function
  - learn weights for instances
  - no actual computation in high-dim space
- Classification
  - average of the instance labels, weighted by a) proximity b) instance weight.

## Kernel trick



## Demo



Demo by Udi Aharoni http://www.youtube.com/watch?v=3liCbRZPrZA

## **Support Vector Machines**

- Main idea:
  - linear classifier, but in  $\varphi(x)$  kernel space.
  - criterion for decision: max-margin
- Algorithm
  - user specifies kernel function
  - learn weights for instances via convex optimization
  - no actual computation in high-dim space
- Classification
  - average of the instance labels, weighted by a) proximity b) instance weight.

## Kernel trick

- Linear classifier is in in high-dimensional  $\phi(x)$  space
- However, no computation directly on  $\phi(x)$ ; compute only kernel =  $\phi(x)^T \phi(x)$

e.g. for  

$$\begin{aligned}
\phi(x_1, x_2) &= \{1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, \sqrt{2}x_1x_2, x_2^2\} \\
k(x, x') &= \phi(x)^T \phi(\underline{x'}) \\
&= ((1, x_1, x_2) (1, x'_1, x'_2)^T)^2 = \langle \mathbf{x}, \mathbf{x'} \rangle^2
\end{aligned}$$

- Efficient only if scalar product can be efficiently computed. Holds for:
- k(x,x') : continuous, symmetric and positive definite

#### **Dual Representations**

- Scalar product representations arise naturally in many classes of problems
- e.g. Linear regression

$$J(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \left\{ \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_n) - t_n \right\}^2 + \frac{\lambda}{2} \mathbf{w}^{\mathrm{T}} \mathbf{w}$$

• Setting gradient to zero:  $\mathbf{w} = -\frac{1}{\lambda} \sum_{n=1}^{N} \left\{ \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_{n}) - t_{n} \right\} \boldsymbol{\phi}(\mathbf{x}_{n}) = \sum_{n=1}^{N} a_{n} \boldsymbol{\phi}(\mathbf{x}_{n}) = \boldsymbol{\Phi}^{\mathrm{T}} \mathbf{a}$ 

where  $\Phi^{\mathsf{T}} = \text{matrix of } \phi(x_n)$ , and  $a_n = -\frac{1}{\lambda} \{ \mathbf{w}^{\mathsf{T}} \phi(\mathbf{x}_n) - t_n \}$ 

#### **Dual Representations**

- Instead of parameter space w, use parameter space a
- Writing  $w = \Phi^T a$  into J(w):

$$\begin{split} J(\mathbf{a}) &= \frac{1}{2} \mathbf{a}^{\mathrm{T}} \boldsymbol{\Phi} \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi} \boldsymbol{\Phi}^{\mathrm{T}} \mathbf{a} - \mathbf{a}^{\mathrm{T}} \boldsymbol{\Phi} \boldsymbol{\Phi}^{\mathrm{T}} \mathbf{t} + \frac{1}{2} \mathbf{t}^{\mathrm{T}} \mathbf{t} + \frac{\lambda}{2} \mathbf{a}^{\mathrm{T}} \boldsymbol{\Phi} \boldsymbol{\Phi}^{\mathrm{T}} \mathbf{a} \\ &= \frac{1}{2} \mathbf{a}^{\mathrm{T}} \mathbf{K} \mathbf{K} \mathbf{a} - \mathbf{a}^{\mathrm{T}} \mathbf{K} \mathbf{t} + \frac{1}{2} \mathbf{t}^{\mathrm{T}} \mathbf{t} + \frac{\lambda}{2} \mathbf{a}^{\mathrm{T}} \mathbf{K} \mathbf{a} \end{split}$$

where gram matrix  $\mathbf{K} = \mathbf{\Phi} \mathbf{\Phi}^{\mathrm{T}}$ , with  $K_{nm} = \phi(x_n)^T \phi(x_m)$ 

solving for **a** by setting dJ(a)/da to zero:

$$\mathbf{a} = \left(\mathbf{K} + \lambda \mathbf{I}_N\right)^{-1} \mathbf{t}.$$

#### **Dual Representations**

• Substituting back into original regression model:

$$y(\mathbf{x}) = \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}) = \mathbf{a}^{\mathrm{T}} \boldsymbol{\Phi} \boldsymbol{\phi}(\mathbf{x}) = \mathbf{k}(\mathbf{x})^{\mathrm{T}} \left(\mathbf{K} + \lambda \mathbf{I}_{N}\right)^{-1} \mathbf{t}$$

 Thus, all computations are performed purely with the kernel

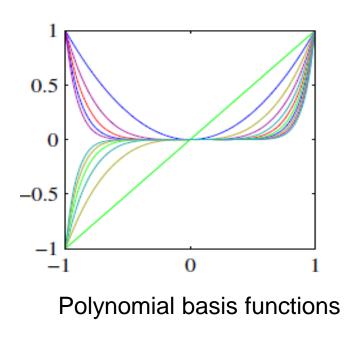
## SVM

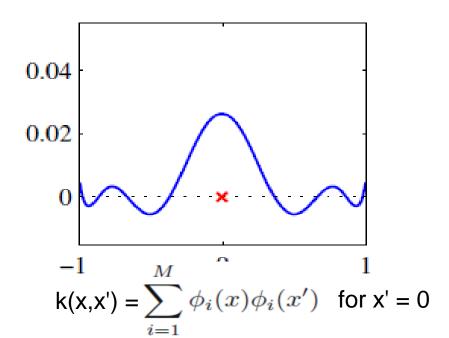
- Why so popular:
  - Very good classification performance, compares w best
  - Fast (convex) and scaleable learning
  - Fast inference (but slower training)
- Difficulties:
  - No model (discriminative; black-box)
  - Not as useful for discrete inputs.
  - Art: how to specify kernel function
  - Difficulties with multiple classes

## **Choosing Kernels**

Popular kernels that satisfy the Gram matrix positive-definiteness criterion inlcude:

- Linear kernels:  $k(\mathbf{x},\mathbf{x'}) = \langle \mathbf{x},\mathbf{x'} \rangle$
- Polynomials:  $k(\mathbf{x},\mathbf{x}') = \langle \mathbf{x},\mathbf{x}' \rangle^n$ 
  - for n=2 (quadratic):  $\phi(\mathbf{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)^T$



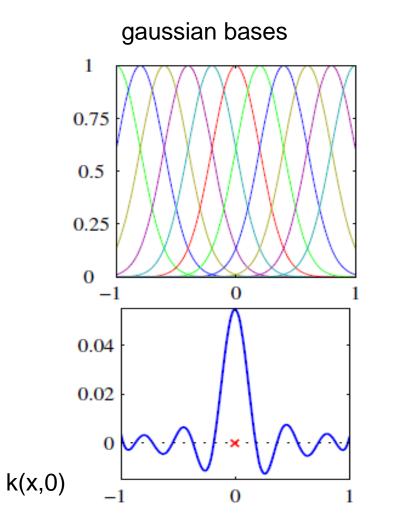


## **Choosing Kernels**

- Gaussian:  $k(\mathbf{x}, \mathbf{x}') = \exp\left(-\|\mathbf{x} \mathbf{x}'\|^2/2\sigma^2\right)$
- Radial basis functions  $f(\mathbf{x}) = \sum_{n=1}^{N} w_n h(\|\mathbf{x} \mathbf{x}_n\|)$
- Sigmoid (logistic) function

$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$

## Gaussian / Sigmoid Bases



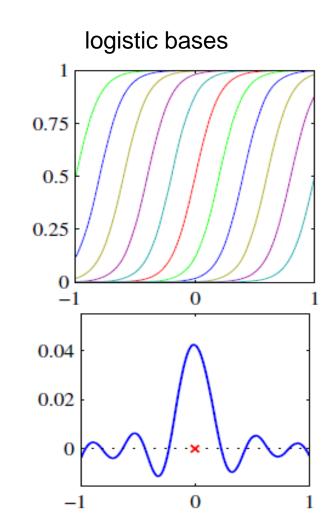


image from [Bishop 2005]

## **Constructing Kernels**

If k1 and k2 are valid kernels, then so are:

$$k(\mathbf{x}, \mathbf{x}') = ck_1(\mathbf{x}, \mathbf{x}') k(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})k_1(\mathbf{x}, \mathbf{x}')f(\mathbf{x}') k(\mathbf{x}, \mathbf{x}') = q(k_1(\mathbf{x}, \mathbf{x}')) k(\mathbf{x}, \mathbf{x}') = \exp(k_1(\mathbf{x}, \mathbf{x}')) k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}') k(\mathbf{x}, \mathbf{x}') = k_3(\phi(\mathbf{x}), \phi(\mathbf{x}')) k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^{\mathrm{T}} \mathbf{A} \mathbf{x}' k(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x}_a, \mathbf{x}_a') + k_b(\mathbf{x}_b, \mathbf{x}_b') k(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x}_a, \mathbf{x}_a')k_b(\mathbf{x}_b, \mathbf{x}_b')$$



• Machine Learning Bishop : sections 1.1 to 1.2.4, 1.5.1-2, 6.1, 6.2

Russell & Norvig: 18.9