Neural Computing

- Backprop variations:
  - Stopping criteria vs. early stopping
  - Regularization methods: weight decay
  - Second order methods
- Hebbian Learning
- Self-Organizing Maps
- Boltzmann machines
- Support Vector Machines

Stopping criteria

300 training epochs have elapsed, or
95% of the training data has error < 0.1, or
90% of training data is correctly classified and no improvement has been seen for 5 epochs

Compare with early stopping (training stops when/if validation set error starts to increase or soon afterwards)

Weight decay

A very popular regularization method

Smaller weights = smoother hypotheses

Using batch learning:

$$W_{t+1} = W_t - \eta \nabla E_{\mathbf{w}} - 2\eta\,(\lambda/N)\,W_t$$

Large λ: $W \rightarrow 0$

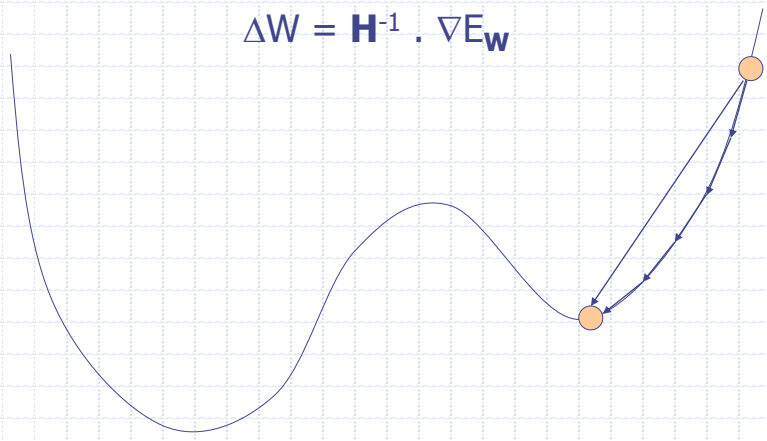
Small λ: shrink weights before moving in the direction of the gradient

Extension: different decay rates at each layer

Second order methods

Use Hessian matrix approximation to take into account curvature

$$\Delta W = \mathbf{H}^{-1} \,.\, \nabla E_{\mathbf{w}}$$



Hessian Matrix

- Square matrix of second-order partial derivatives of a scalar-valued function
- Describes the local curvature of a function of many variables
- Used in Taylor Series for decomposing any function f(x) at a given point a:

Hessian

$$f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \cdots.$$

Example

$f(x)=x^2$ Let $x_0=3$

$f'(x_0)=6$ $f''(x_0)=2$

$x_0 - \mathbf{H}^{-1} . \nabla E_{\mathbf{w}} = 3 - 6/2 = 0$

Converges in 1 iteration!

But… computing $\mathbf{H}^{-1}$ can be very expensive

Levenberg–Marquardt

- Second order method
- Uses an approximation of **H** by averaging the products of the gradients (**H** becomes symmetric and positive)
- It is therefore a quasi-Newton method

Hebbian Learning

Hebb's postulate: When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.

Hebb's Rule

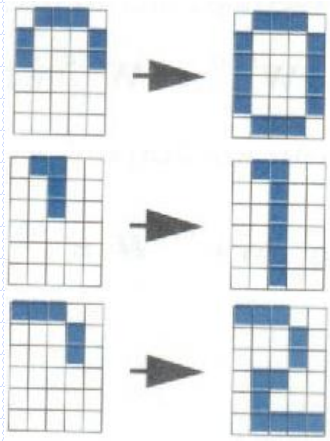
- If two neurons on either side of a synapse are activated simultaneously, increase the weight of the synapse. Otherwise, decrease the weight.

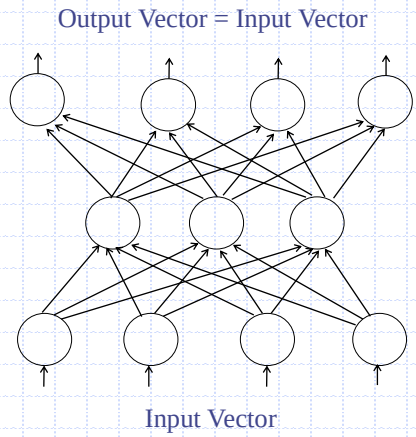
$$\Delta W_{ij} = \eta \cdot f(A_i, A_j)$$

- Thus, a good candidate learning rule for neurons with bipolar activation is:

$$\Delta W_{ij} = \eta \cdot A_i \cdot A_j$$

Auto-associative Memory: Data Compression

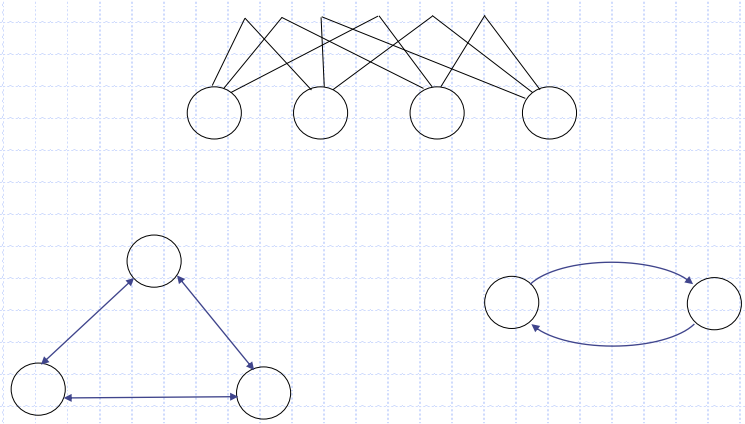




Hopfield Networks

- Single layer recurrent networks (i.e. with feedback)
- Each neuron is a perceptron with sign(x) as activation function
- $W_{ij} = W_{ji}$
- There're no self-feedbacks (bias = 0)
- Use Hebb's rule for learning

Hopfield Networks Architecture



Activation Function Sign

At each time *t*, each neuron *i* has activation $A_i(t) = 1$ or -1
Selecting neuron *i* to calculate its new activation value at t+1:

$$A_i(t+1) = sign\left(\sum_j W_{ij} A_j(t)\right)$$

where :

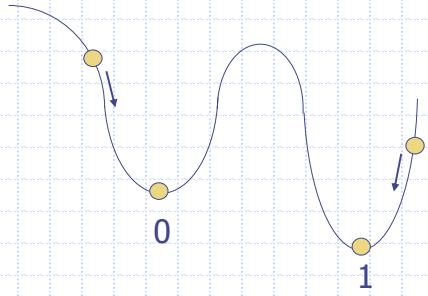
$$sign(x) = 1 \text{ if } x > 0$$

$$sign(x) = -1 \text{ if } x < 0$$

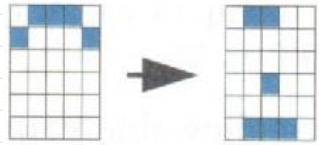
$$sign(x) = A_i(t) \text{ if } x = 0$$

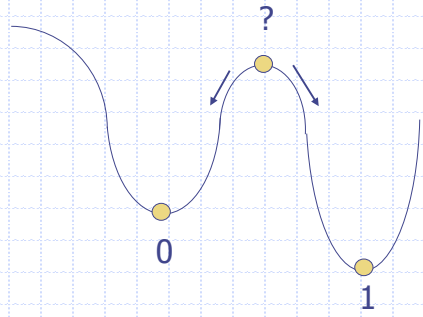
Associative Memory: Content-addressable directory

- Retrieves data by producing whichever one of the stored patterns most closely resembles the input pattern.



Spurious Patterns / Noise





Hopfield Networks as Associative Memory

- Storage Phase:
  - To store M N-dimensional vectors ξ_μ
  - Let $\xi_{\mu i}$ denote the i-th element of ξ_μ

$$W_{ji} = \frac{1}{N}\sum_{\mu=1}^{M}(\xi_{\mu j} \cdot \xi_{\mu i})$$

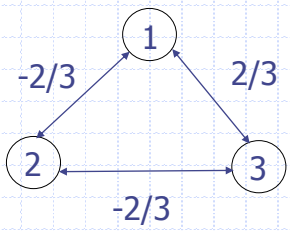
  - This is also called a one-shot learning!

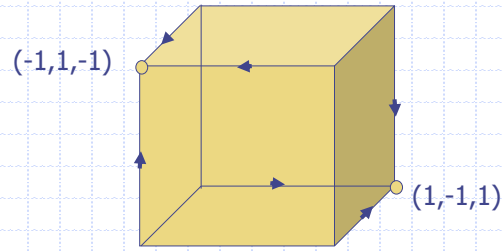
Hopfield Networks as Associative Memory

- Retrieval Phase:
  - An N-dimensional vector of activations ξ_p is imposed on the network;
  - At each time *t*, a neuron *i* is selected and its activation state is updated to $A_i(t+1)$ immediately before another neuron is selected;
  - Neurons are selected in order until a time invariant vector of activations (stable state) ξ_s is found, i.e. until $A_i(t+1) = A_i(t)$ for all neurons in the network.

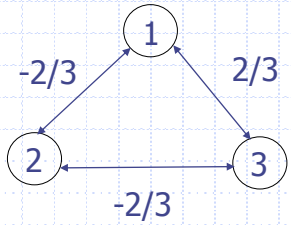
Hopfield Energy Function

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}(W_{ji} \cdot A_i \cdot A_j), \quad i \neq j$$





Stable State = Lowest Energy

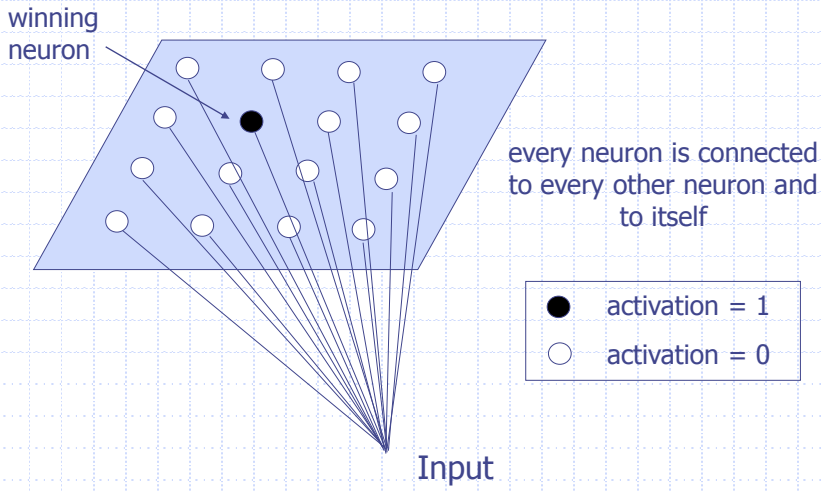


	Energy
-1 -1 -1	?
-1 -1 1	2/3
-1 1 -1	-2
-1 1 1	2/3
1 -1 -1	?
1 -1 1	-2
1 1 -1	?
1 1 1	2/3

Competitive Learning

- The neurons compete with each other to determine a winner
  - After competition, only one neuron will have a positive activation
- Mathematically:
  - Let m denote the dimension of the input space.
  - Let $\mathbf{x} = [x_1, x_2, \ldots, x_m]$ be an input vector.
  - Let $\mathbf{W}_j = [W_{j1}, W_{j2}, \ldots, W_{jm}]$ be the weight vector of neuron j (j = 1,2,…,n); where n is the total number of neurons in the network).

Two Dimensional Feature Map



Winning Criteria

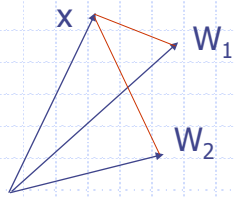
- The neuron whose weight vector best matches the current input vector wins
  - Compare the inner product $\mathbf{W}_j \mathbf{x}$ for j = 1,2,…,n and select the largest.
  - Alternatively, minimize the Euclidean distance between $\mathbf{W}_j$ and $\mathbf{x}$

$$i(\mathbf{x}) = \min_j \left\| \mathbf{x} - \mathbf{W}_j \right\|$$

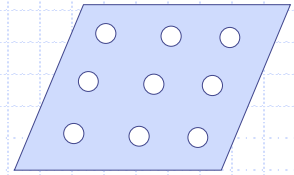
  - i(x) identifies the neuron i that best matches input x

Result of Competition

- A continuous input space is mapped into a discrete output space (the feature map).



winning neuron is the one whose weight vector is closest to input vector in the Euclidean space



Topological Space

Self Organising Maps

1. Competition
2. Co-operation
3. Synaptic Adaptation

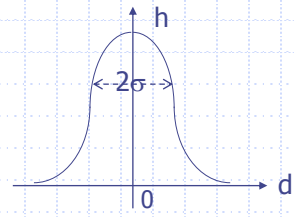
Co-operation: the winning neuron locates the centre of the topological neighbourhood of co-operating neurons.

A firing neuron tends to excite the neurons in its immediate neighbourhood more than those farther away.

Co-operation

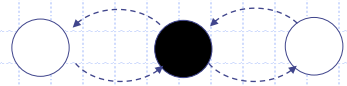
- Let h_{ji} denote a neighbourhood centred on winning neuron i and containing a set of co-operating neurons, one of which is j.
- Let d_{ji} denote the distance between winning neuron i and its neighbour j.
  - We want h_{ji} to be symmetric w.r.t d_{ji}
  - We want h_{ji} to decrease monotonically with increasing d_{ji}
  - A typical choice is the Gaussian function:

$$h_{ji} = \exp\left(-\frac{d_{ji}^2}{2\sigma^2}\right)$$

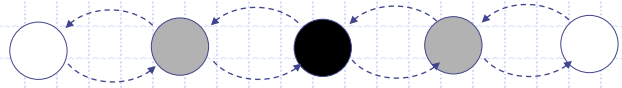


Biological Motivation

- Competition implements *lateral inhibition*



- Co-operation implements *lateral interaction*



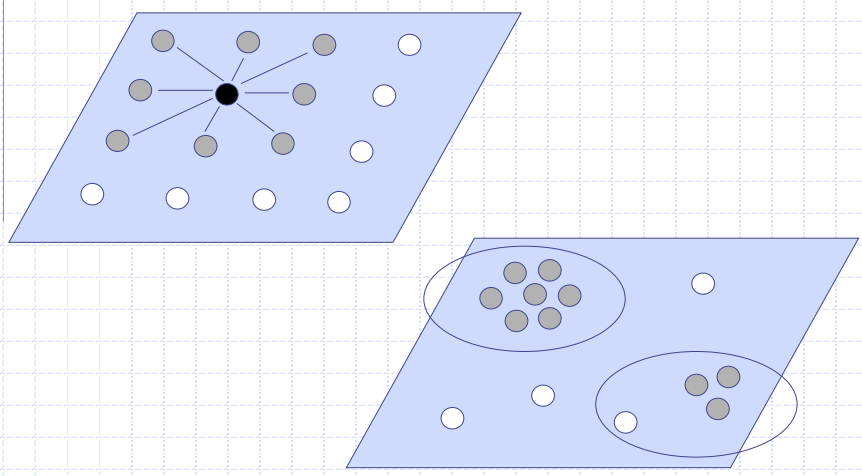
Synaptic Adaptation

- For the network to be self-organising, the weight vectors are required to change according to the input vector.
- A variation of Hebb's rule seems appropriate:

$$\Delta \mathbf{W}_j = \eta \cdot h_{ji} \cdot (\mathbf{x} - \mathbf{W}_j)$$

- It has the effect of *moving* the weight vector W_i of winning neuron i towards the input vector x, and the weight vector of neighbouring neurons W_j also towards x but less than W_i (with discount h_{ji})
- Upon repeated presentation of training data, the weight vectors follow the distribution of the input vectors.

Clustering



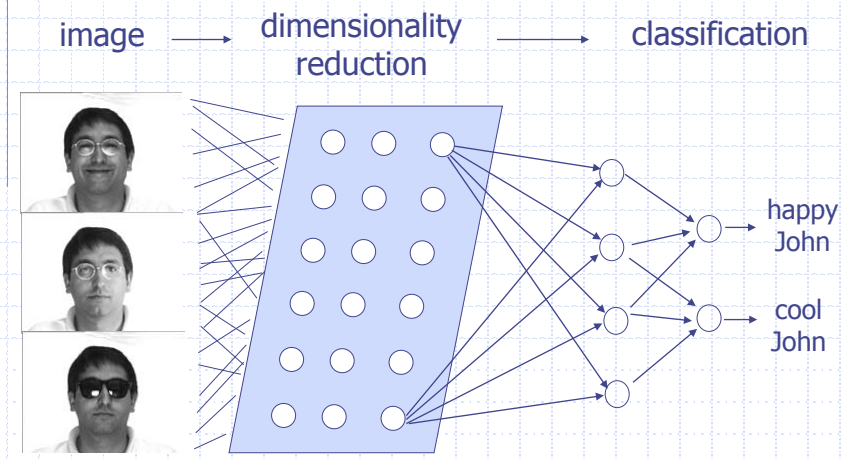
Properties of Self Organising Maps

- **Dimensionality Reduction**: self organising maps transform input pattern of arbitrary dimension into one or two dimension discrete map.
- **Feature Extraction**: given data from an input space with a nonlinear distribution, the self organising map is able to select a set of best features for approximating the distribution.
- **Topological Ordering**: The feature map computed by the self organising network is topologically ordered in the sense that the spatial location of a neuron in the lattice corresponds to a particular domain or feature of input patterns.

Some Applications of SOMs

- To locate similar patterns in maps and satellite images (e.g.: meteorology).
- To predict potential areas for good oil prospecting based on geological data.
- Organisation and retrieval from large document collections (WEBSOM).

Hybrid SOM for Face Recognition



Boltzmann Machines

- Stochastic variant of the Hopfield network
- Use the Boltzmann distribution as sampling function
- Good for learning joint data distributions
- Slow in practice but efficient with restricted connectivity

Boltzmann Machines (2)

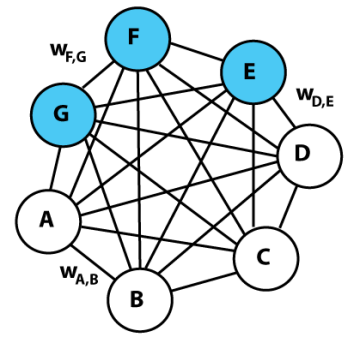
- Now neurons are *on* (resp. *off*) with probability p_i (resp. 1-p_i)
- The energy E_i of a Boltzmann machine is similar to that of a Hopfield net:

$$E = -\left(\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} (W_{ji} \cdot A_i \cdot A_j) + \sum_{i=1}^{N} (\theta_i \cdot A_i) \right), \quad i \neq j$$

- The use of a bias (denoted W_{0i} or θ_i) is permitted

Boltzmann machines (3)

The network learns a joint distribution on the visible units



The difference in energy when a single unit *i* is flipped from *off* (zero) to *on* (one) is given by:

$$\Delta E_i = E_{i=on} - E_{i=off} = -\sum_j W_{ij} \cdot A_j + \theta_i, \quad i \neq j, \; A_j \in \{0,1\}$$

Gibbs sampling (cont.)

- Start with an arbitrary state
- Neurons are repeatedly visited in order
- Calculate new value for neuron i in {0,1} according to:

$$p_i = 1/(1+\exp(-\Delta E_i/T))$$

until the network reaches thermal equilibrium…

Simulated annealing

- This is stochastic hill-climbing with a twist…
- If a move is as good as or better than current state, accept it.
- If not, it may be accepted anyway according to a Boltzmann distribution
- The "temperature" adjusts the probability of acceptance, lower being more like hill-climbing!

Simulated annealing (cont.)

- Temperature is adjusted during the run according to a "cooling schedule", i.e. from high to low (why?)
- Provably able to find optimum in infinite time
  - Annealing in metallurgy involves heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. The heat causes the atoms to become unstuck from their initial positions (a local minimum of the internal energy) and wander randomly through states of higher energy; the slow cooling gives them more chances of finding configurations with lower internal energy than the initial one.

Boltzmann learning rule

Positive phase: visible units' states are clamped to a particular binary state vector sampled from the training set.

Negative phase: the network is allowed to run freely, i.e. no units have their state determined by external data.

$$\Delta \mathbf{W}_{ji} = \frac{1}{T}(p^{+}_{ji} - p^{-}_{ji})$$

Boltzmann learning rule (cont.)

- $p+$ is probability of units i and j both being on when the machine is at equilibrium on the positive phase.
- $p-$ is probability of units i and j both being on when the machine is at equilibrium on the negative phase.

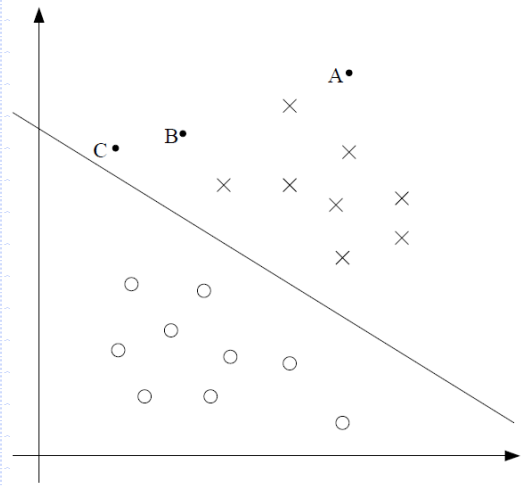
Further reading:

Deep Boltzmann Machines:

www.utstat.toronto.edu/~rsalakhu/papers/dbm.pdf

Support Vector Machines

A good separation: hyperplane with the largest distance to the nearest training data point of any class (functional margin)



Functional Margins

Linear classifier *h* for a binary classification problem with labels (classes) y in {-1,1} and input vectors (features) x:

$$h_{w,b}(x) = g(w^T x + b)$$

where g(z)=1 if z ≥ 0; g(z)=-1 otherwise.

Given training example $(x^{(i)},y^{(i)})$, the functional margin of (w,b) w.r.t. $(x^{(i)},y^{(i)})$ is defined as:

$$\hat{\gamma}^{(i)} = y^{(i)}(w^T x + b)$$

If $y^{(i)}=1$, for the margin to be large then $w^T x+b$ needs to be a large positive number

Normalization

$$g(w^T x+b) = g(2w^T x+2b)$$

Scaling (w,b) can make the margin arbitrarily large without really changing anything.

Solution: replace (w,b) with $(w/\|w\|, b/\|w\|)$

Given a set of examples S, we're interested in maximizing the distance from (w,b) to the smallest of the functional margins in S.

Convex Optimization problem

Constraint: make functional margin w.r.t. S equal to 1

Note that maximizing $1/\|w\|$ is the same as minimizing $\|w\|^2$

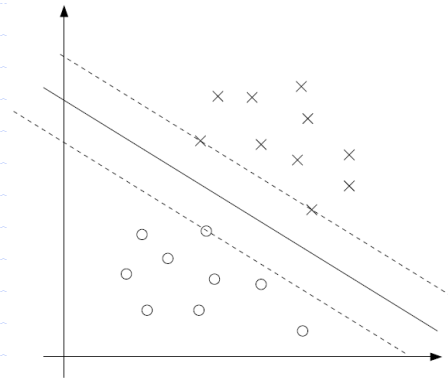
$$\min_{\gamma,w,b} \quad \frac{1}{2}||w||^2$$
$$\text{s.t.} \quad y^{(i)}(w^T x^{(i)} + b) \geq 1, \quad i = 1, \ldots, m$$

i.e. quadratic objective (a.k.a. cost) function with only linear constraints = computationally efficient for large m (but not very large m)

Support Vectors

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2}||w||^2 - \sum_{i=1}^{m} \alpha_i \left[y^{(i)}(w^T x^{(i)} + b) - 1\right]$$

α_i is non-zero only for the three points in the parallel lines



For details c.f. Lagrange duality and Karush Kuhn Tucker KKT conditions!

Using Lagrange duality

$$\max_\alpha \quad W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle$$

$$\text{s.t.} \quad \alpha_i \geq 0, \quad i = 1, \ldots, m$$

$$\sum_{i=1}^{m} \alpha_i y^{(i)} = 0,$$

If we manage to find the above αi's, testing for new point x only needs to calculate inner products between x and the support vectors

$$w^T x + b = \left(\sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} \right)^T x + b$$

$$= \sum_{i=1}^{m} \alpha_i y^{(i)} \langle x^{(i)}, x \rangle + b.$$

Kernels – feature mapping

Solving XOR with a single perceptron requires an increase in dimensionality

Solving XOR with an SVM requires feature mapping, e.g. if x_i in {-1,1}, let $\phi(x)=-x_1x_2$

x_1	x_2	class	$-x_1x_2$
-1	-1	-1	-1
-1	1	1	1
1	-1	1	1
1	1	-1	-1

SVM Kernels

Given a feature mapping Φ we define a Kernel between two data points x and z as $K(x,z)= \Phi(x)^T\Phi(z)$

e.g. new point x and support vector z

For high-dimensional vectors, calculating $\Phi(x)$ may be very expensive, but not calculating, say, $K(x,z)=(x^Tz)^2$

Mercer theorem

Given a function K how can we tell that it is a valid Kernel, i.e. that there is a feature mapping Φ such that $K(x,z)= \Phi(x)^T\Phi(z)$ for all x, z?

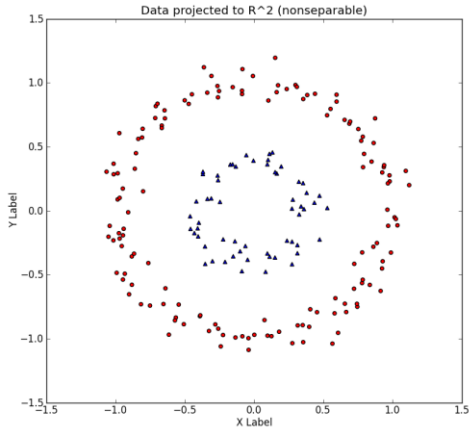
Given m data points, let Kernel matrix **K** be m x m matrix with i,j entry equal to $K(x^i,x^j)$. Then K is valid iff **K** is symmetric, positive, semi-definite

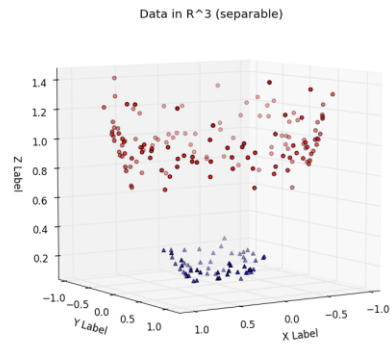
Kernel trick

If a learning algorithm can be written in terms of inner products <x,z> between input vectors x and z then the use of polynomial kernel $K(x,z)=(x^Tz)^d$ or Gaussian kernel $K(x,z)=\exp(|x-z|^2/2\sigma^2)$ offers an improvement in computational efficiency by avoiding the computation of the feature mapping $\phi(x)$

We simply **replace all inner products** <x,z> **in an SVM computation by** K(x,z)

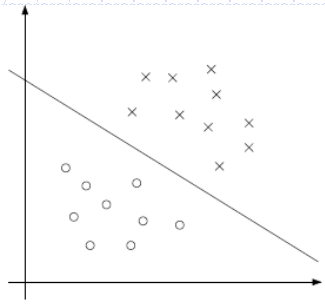
Example: $(x_1,x_2) \rightarrow (x_1,x_2,x_1^2+x_2^2)$

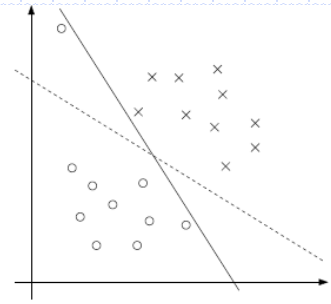




http://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html

Outliers and Regularization





$$\min_{\gamma,w,b} \quad \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\xi_i$$

Penalty for examples with functional margin less than 1

$$\text{s.t.} \quad y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i, \quad i = 1, \ldots, m$$
$$\xi_i \geq 0, \quad i = 1, \ldots, m.$$