

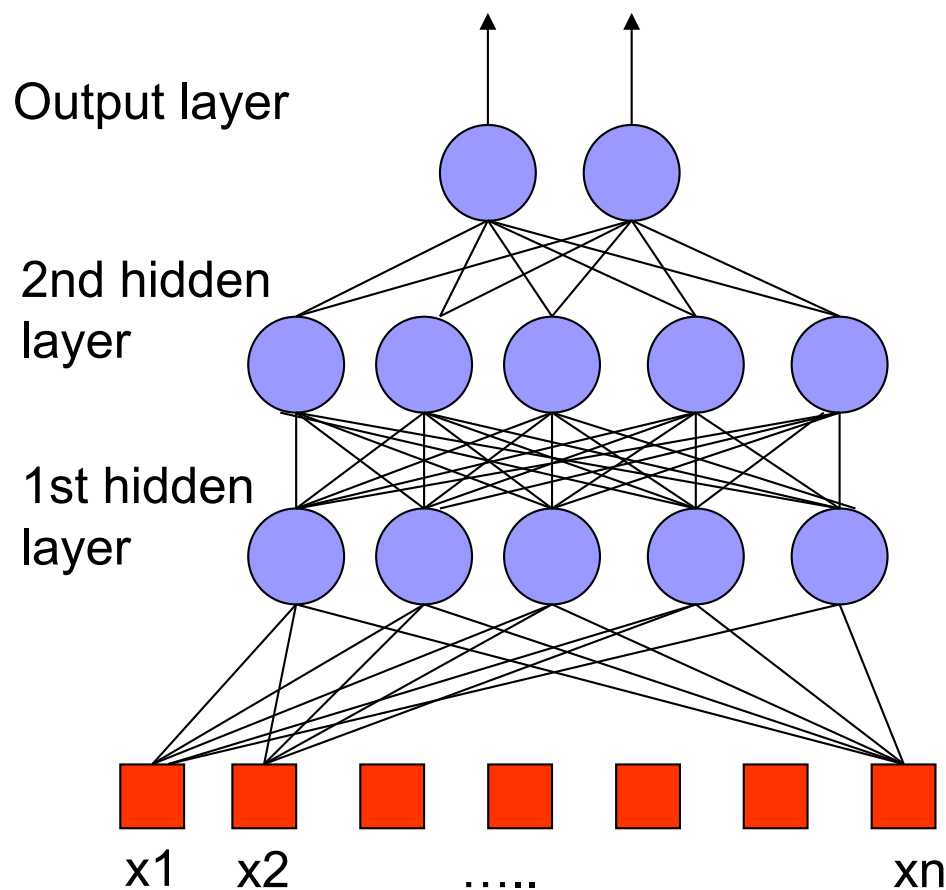
Various Neural Networks



Neural Networks

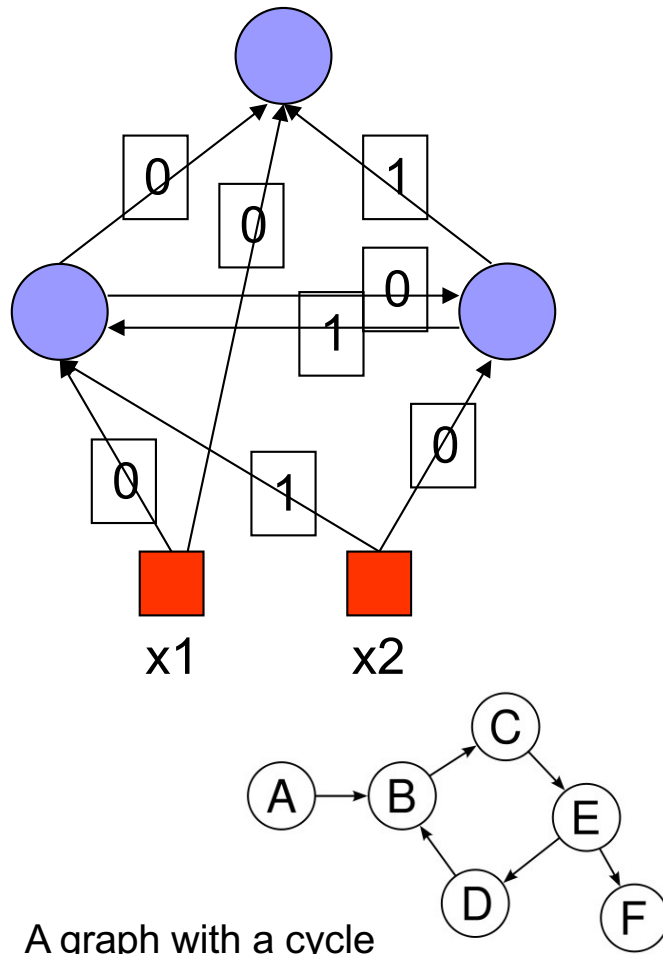
- A mathematical model to solve engineering problems
 - Group of connected neurons to realize compositions of non linear functions
- Tasks
 - Classification
 - Discrimination
 - Estimation
- 2 types of networks
 - Feed forward Neural Networks
 - Recurrent Neural Networks

Feed Forward Neural Networks



- The information is propagated from the inputs to the outputs
- Computations of functions from n input variables by compositions of functions
- Time has no role (NO cycle between outputs and inputs)

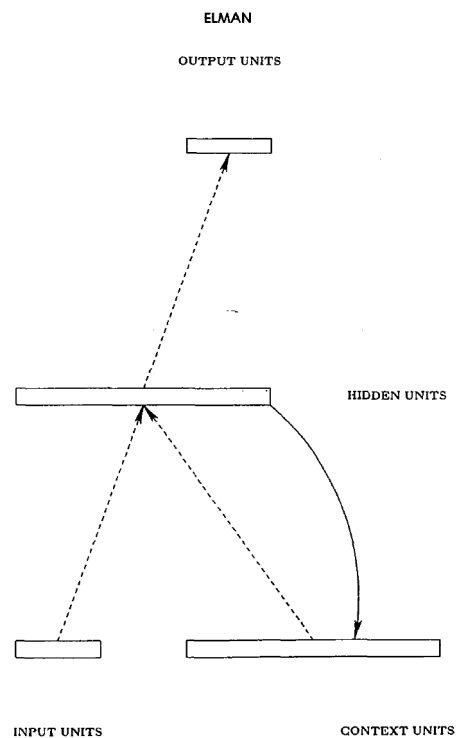
Recurrent Neural Networks



A graph with a cycle

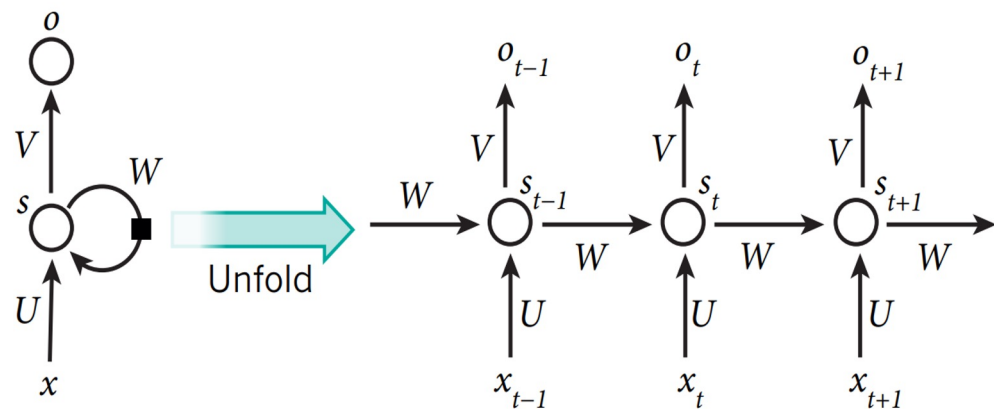
- Can have arbitrary topologies. i.e. connections between units form a cycle.
- Can model systems with internal states (dynamic ones)
- Delays are associated to a specific weight
- Training is more difficult
- Performance may be problematic
 - Stable Outputs may be more difficult to evaluate
 - Unexpected behavior (oscillation, chaos, ...)

Recurrent Neural Networks

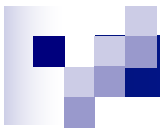


Elman's original proposal

"Finding structure in time"



Unfolding a RNN in time



Properties of Neural Networks

- Supervised networks are universal approximators
- Theorem : Any limited function can be approximated by a neural network with a finite number of hidden neurons to an arbitrary precision



Supervised Learning

- The desired response of the neural network in function of particular inputs is well known.
- A “teacher” may provide examples and teach the neural network how to fulfill a certain task



Unsupervised learning

- Idea : group typical input data in function of resemblance criteria un-known a priori
- Data clustering
- No need of a “teacher”
 - The network finds itself the correlations between the data
 - Examples of such networks :
 - Self-Organizing Feature Maps (SOM)



Semi-Supervised Learning

- Halfway between supervised and unsupervised
- The desired response of the neural network is known only for a subset of the inputs, together with unlabeled data.



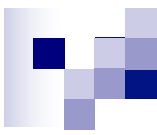
Reinforcement Learning

- The desired response of the neural network in function of particular inputs is not known.
- Only a *reward* or *penalty* value is provided with the input examples
- *Q-learning*: delayed rewards

Example

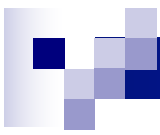
65473 60198 68544
70065 70117 19032 96720
27260 61820 19559
74136 ~~19137~~ 43101
20878 60521 38002
48640-2398 20907 14868

Examples of handwritten postal codes
drawn from a database available from the US Postal service



What is needed to create a NN ?

- Determination of relevant inputs
- Collection of data for the learning and testing phases of the neural network
- Finding the optimum number of hidden nodes
- Learning the parameters
- Evaluate the performances of the network
- If performances are not satisfactory then review all the precedent points

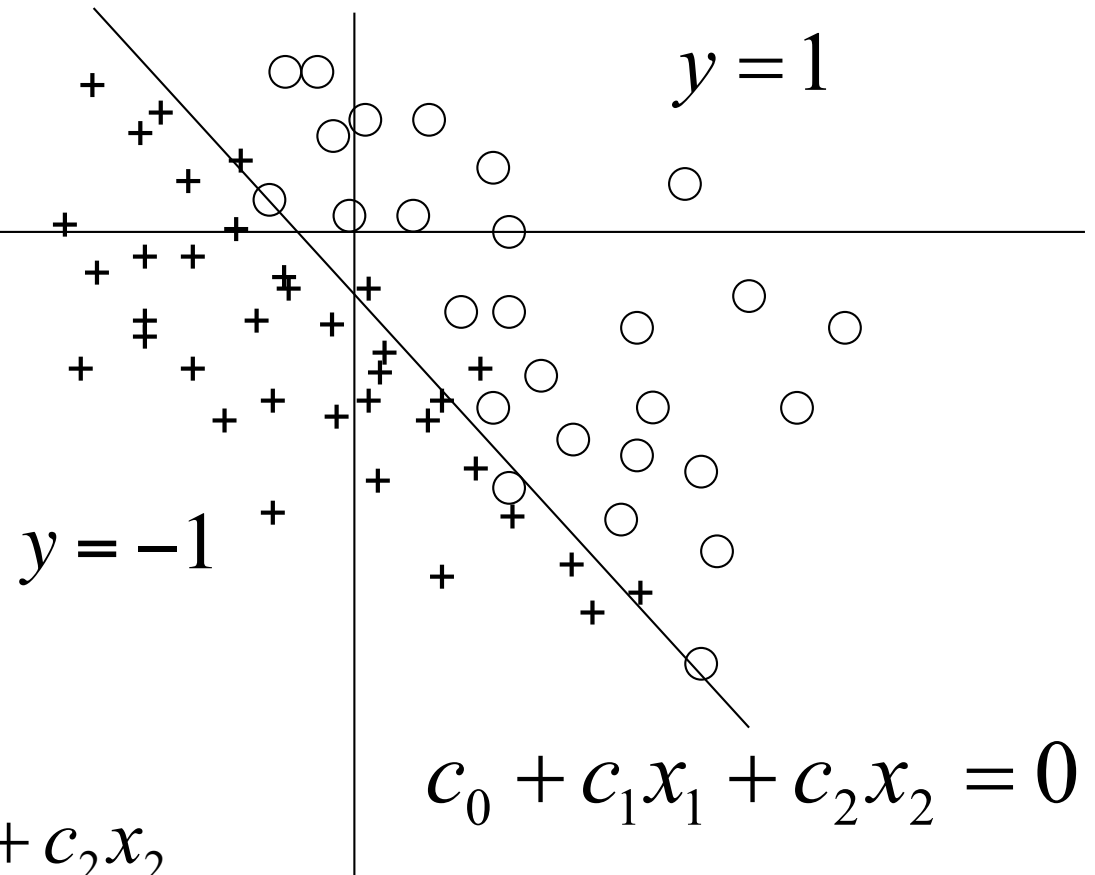
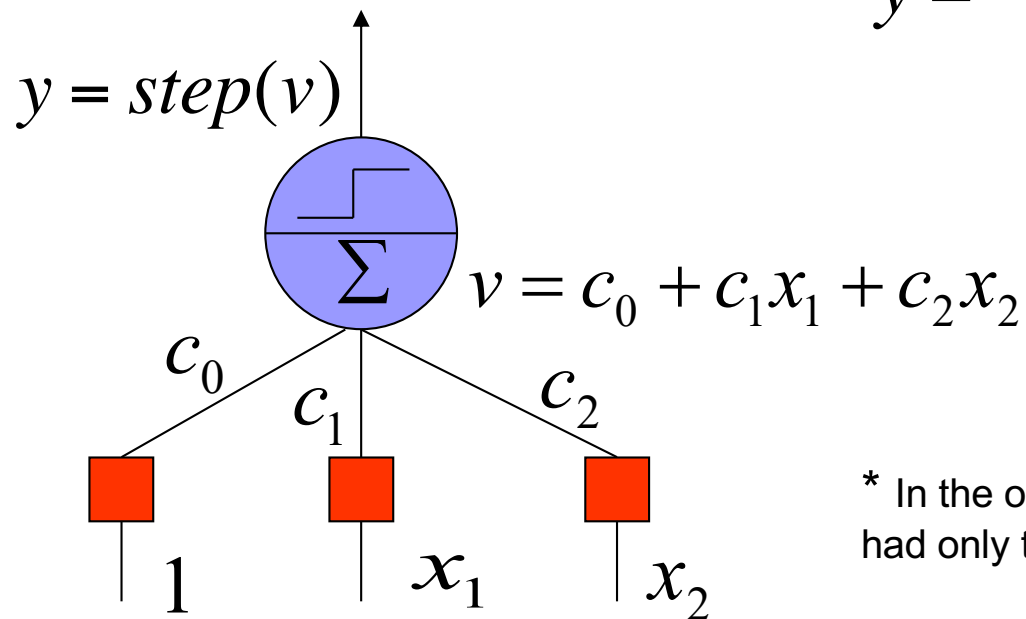


Popular neural architectures

- Perceptron
- Multi-Layer Perceptron (MLP)
- Radial Basis Function Network (RBFN)
- Self-Organizing Feature Maps (SOM)
- Other architectures

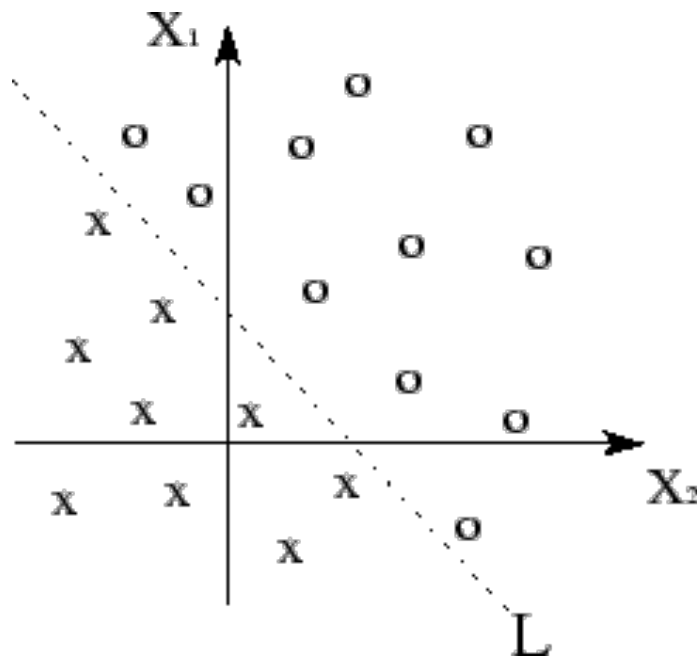
Perceptron

- Rosenblatt (1962)
- Linear separation
- Inputs : Vector of real values*
- Outputs : 1 or -1



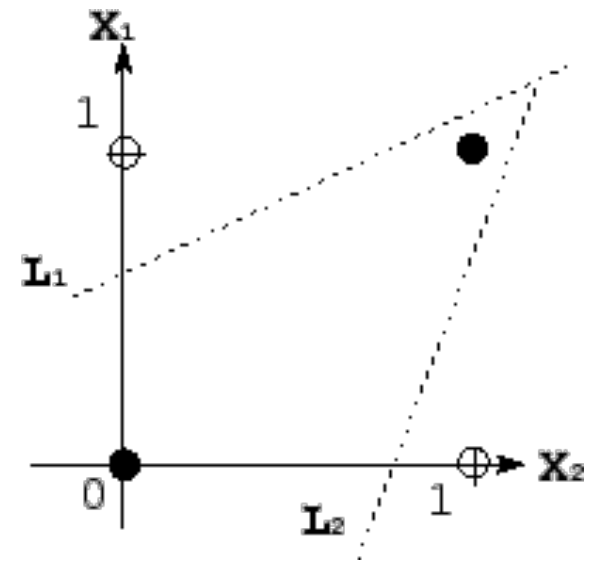
* In the original definition of perceptron inputs typically had only two states: ON and OFF

- The perceptron algorithm converges if examples are linearly separable

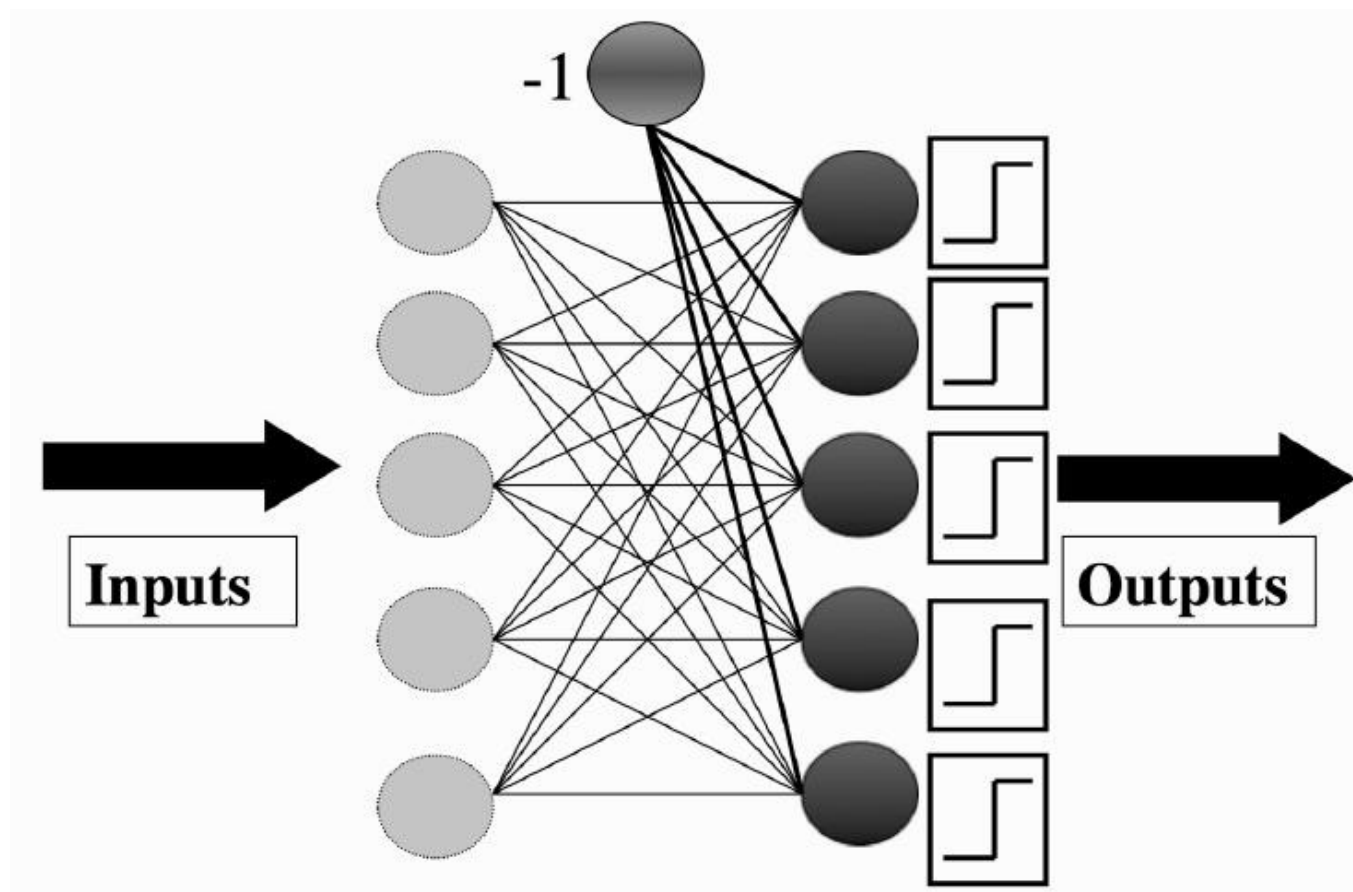


| X_1 | X_2 | Y |
|-------|-------|-----|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

$Y = X_1 \oplus X_2$

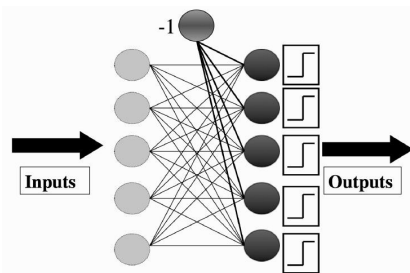
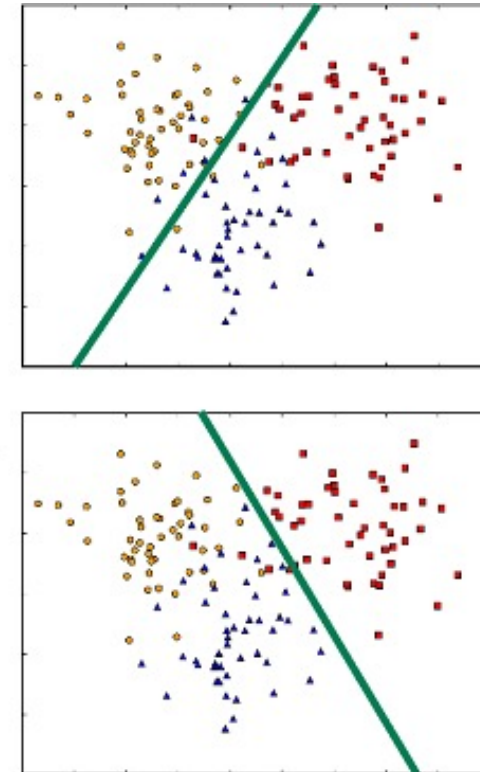
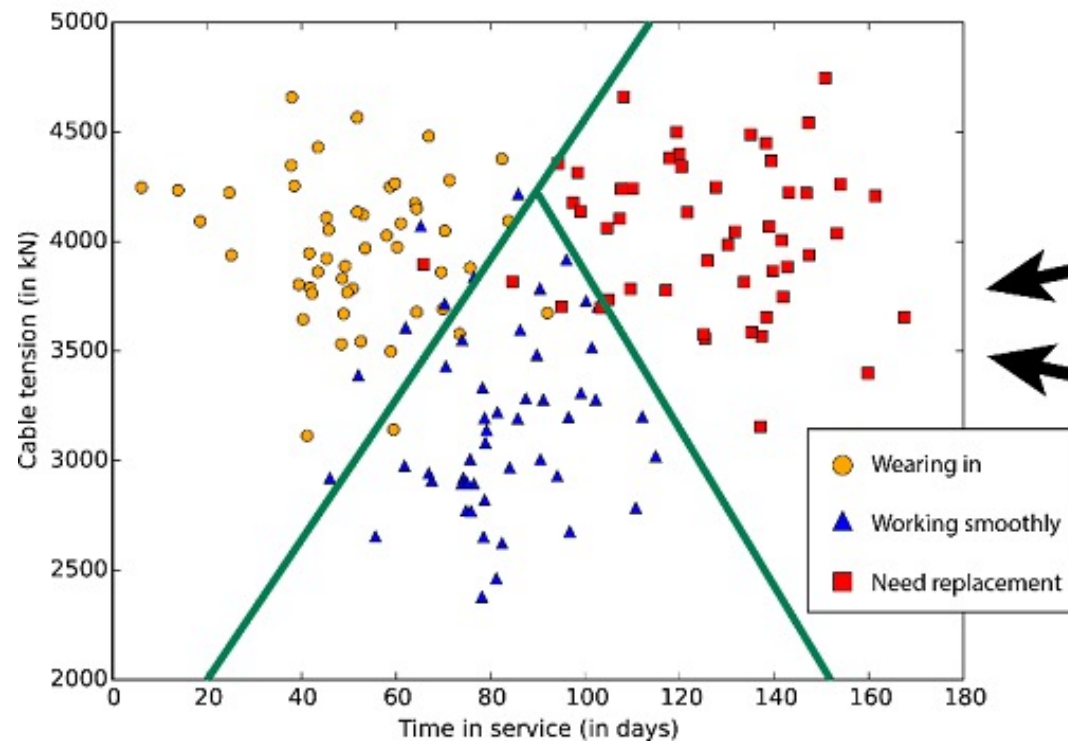


Perceptron with several outputs



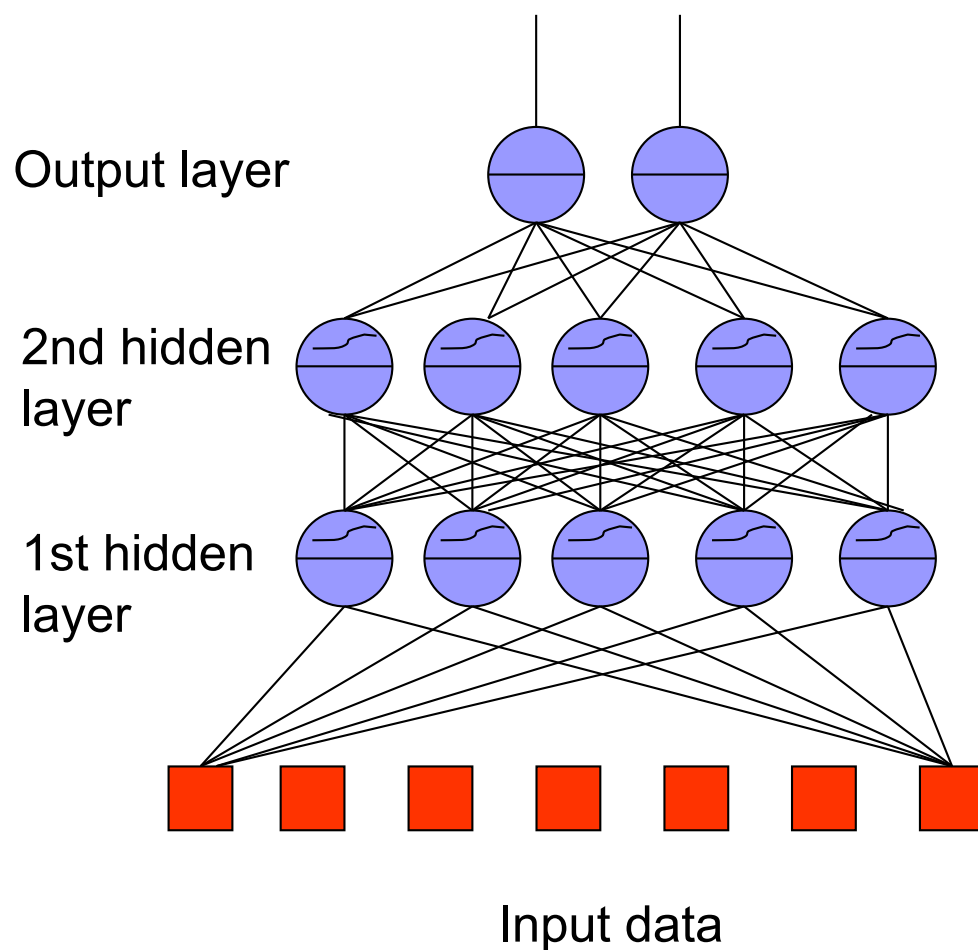
A perceptron network with several outputs (from Marsland Fig. 3.3)

Perceptron with several outputs

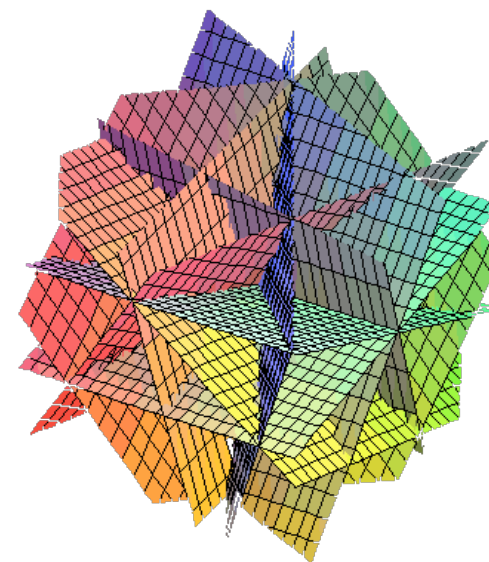


Adding a selector cell per class we have a multilayer perceptron

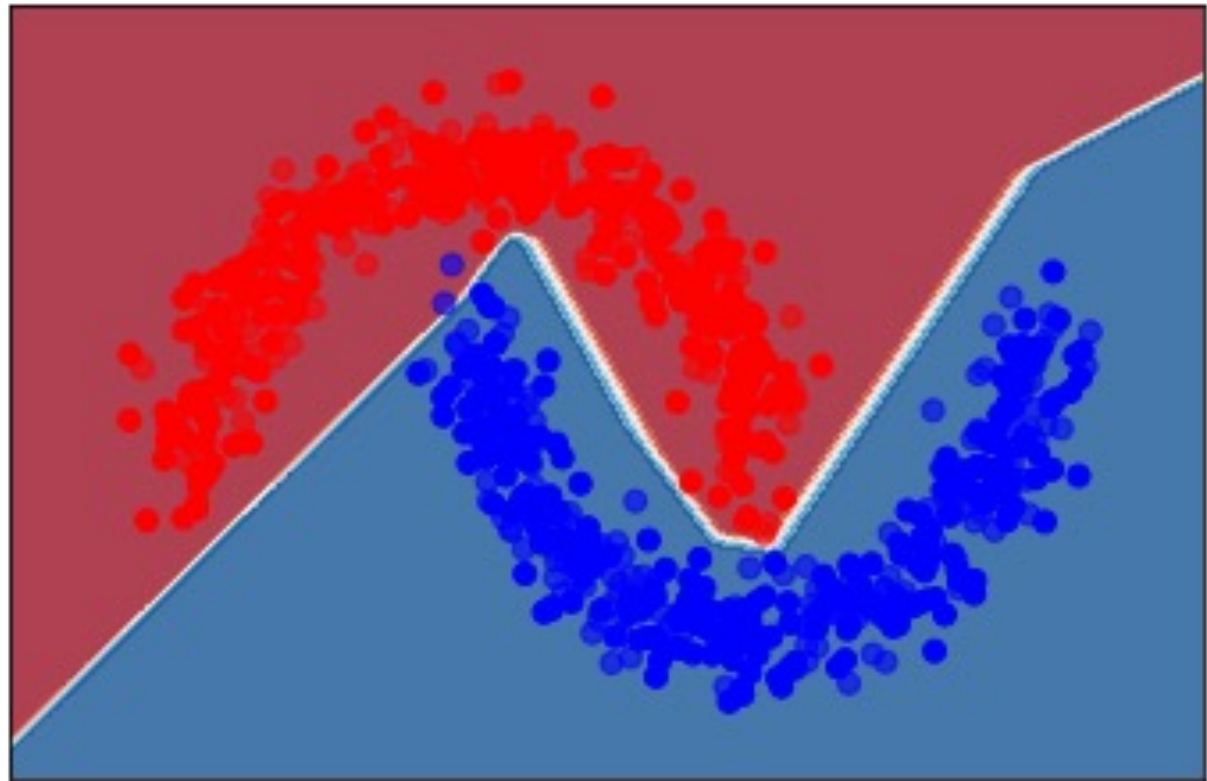
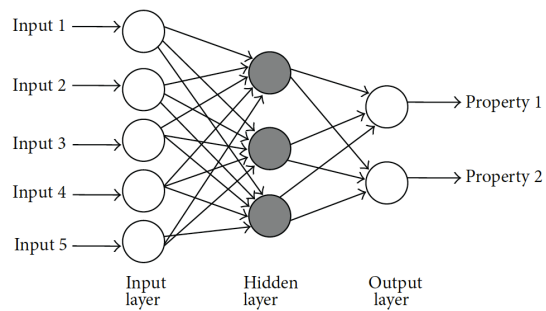
Multi-Layer Perceptron



- One or more hidden layers

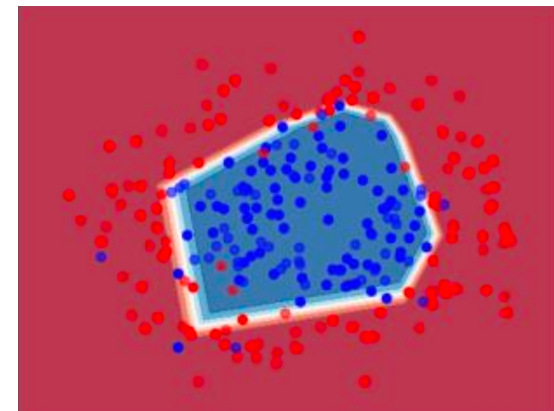
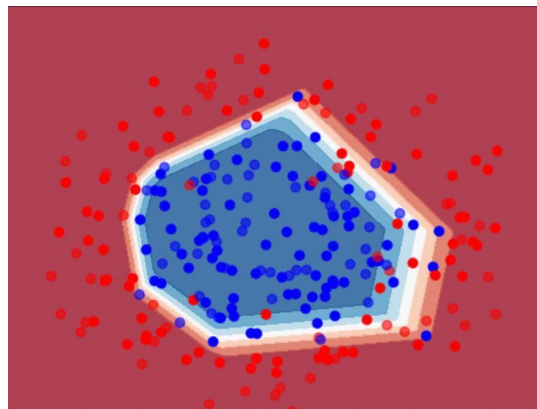
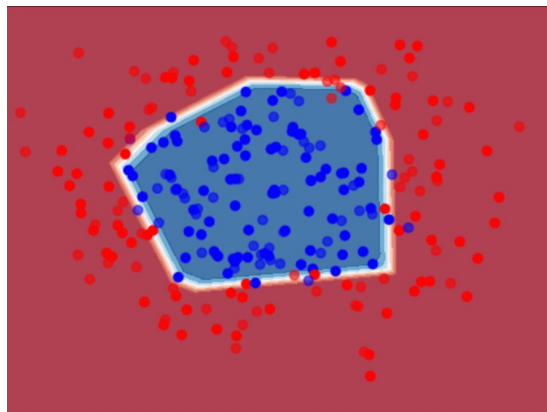
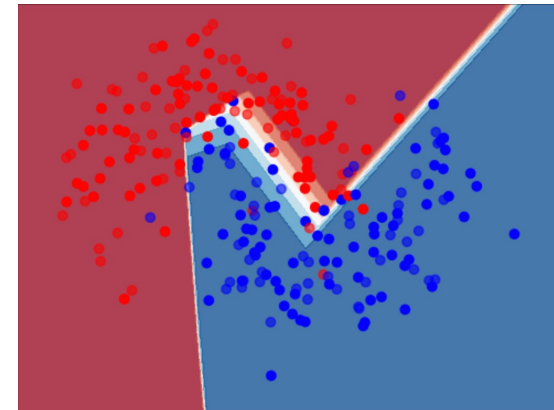
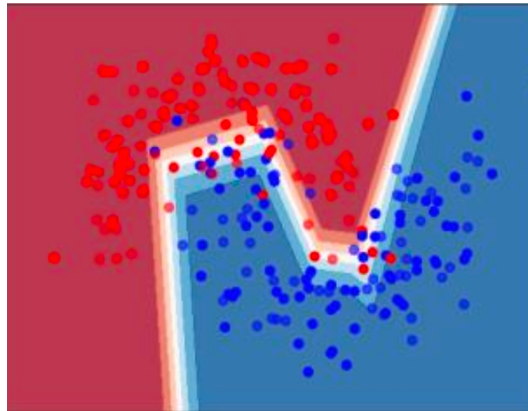
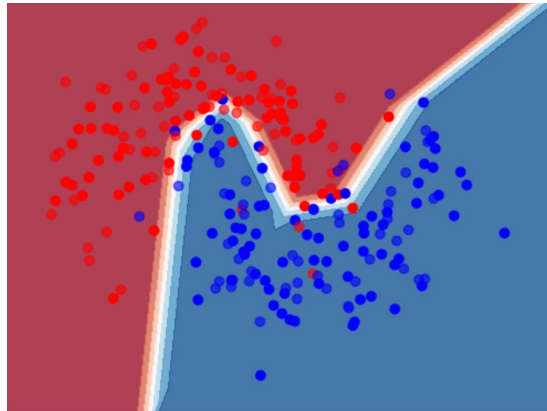


Multi-Layer Perceptron





Multi-Layer Perceptron





Backpropagation

- Backpropagation is a process involved in **training** a neural network.
- It involves taking the error rate of a forward propagation and feeding this loss backward through the neural network layers to fine-tune the weights.
- Neural Networks [online course](#)

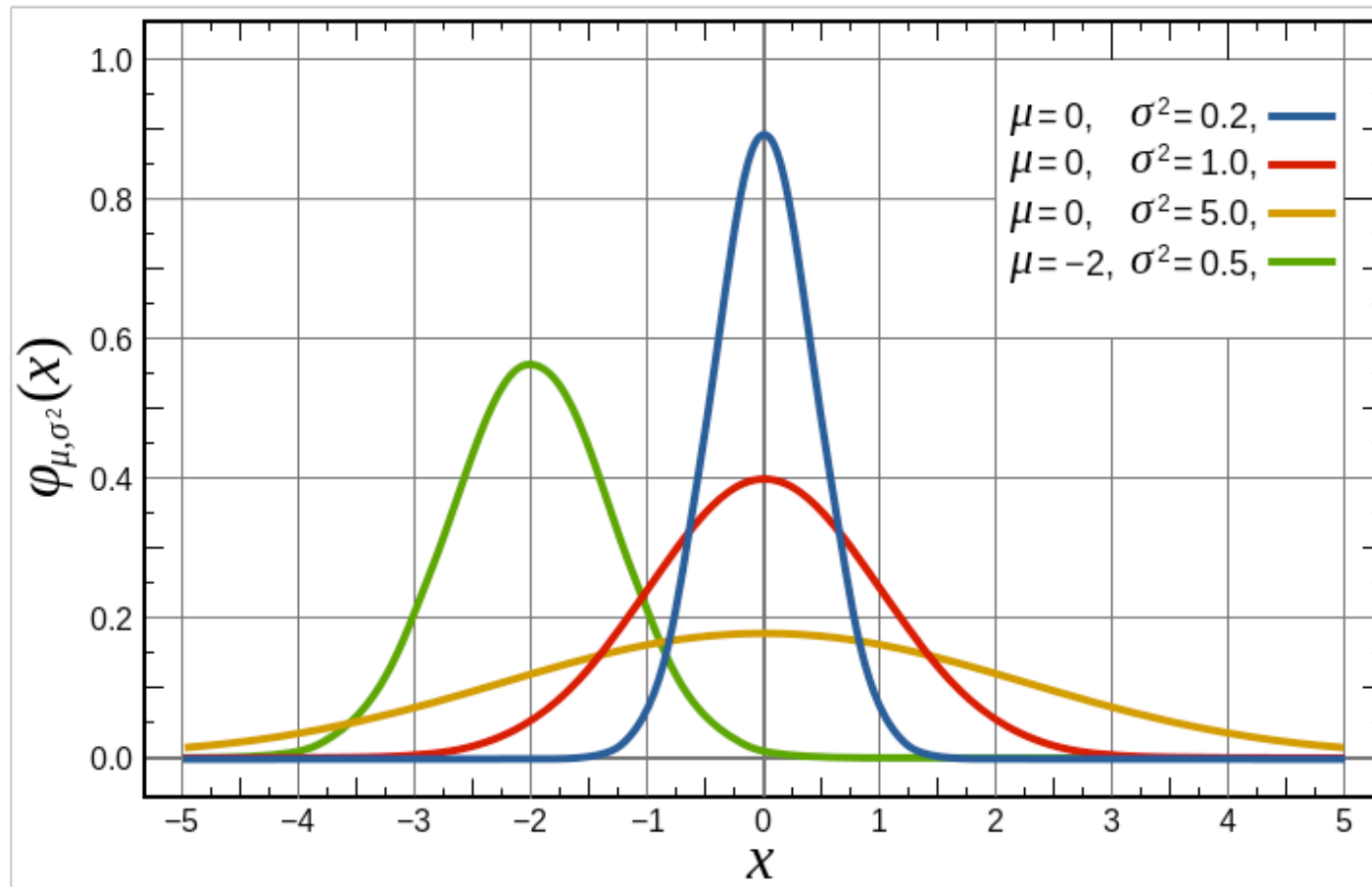
Radial Basis Functions

- A **radial basis function (RBF)** is a real-valued function whose value depends only on the distance from some other point c , called a *center*, $\varphi(\mathbf{x}) = f(||\mathbf{x}-\mathbf{c}||)$
- Any function φ that satisfies the property $\varphi(\mathbf{x}) = f(||\mathbf{x}-\mathbf{c}||)$ is a radial function.
- The distance is usually the Euclidean distance

$$||\mathbf{x} - \mathbf{c}||^2 = \sum_{i=1}^N (x_i - c_i)^2$$

Distance in 2D space:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$



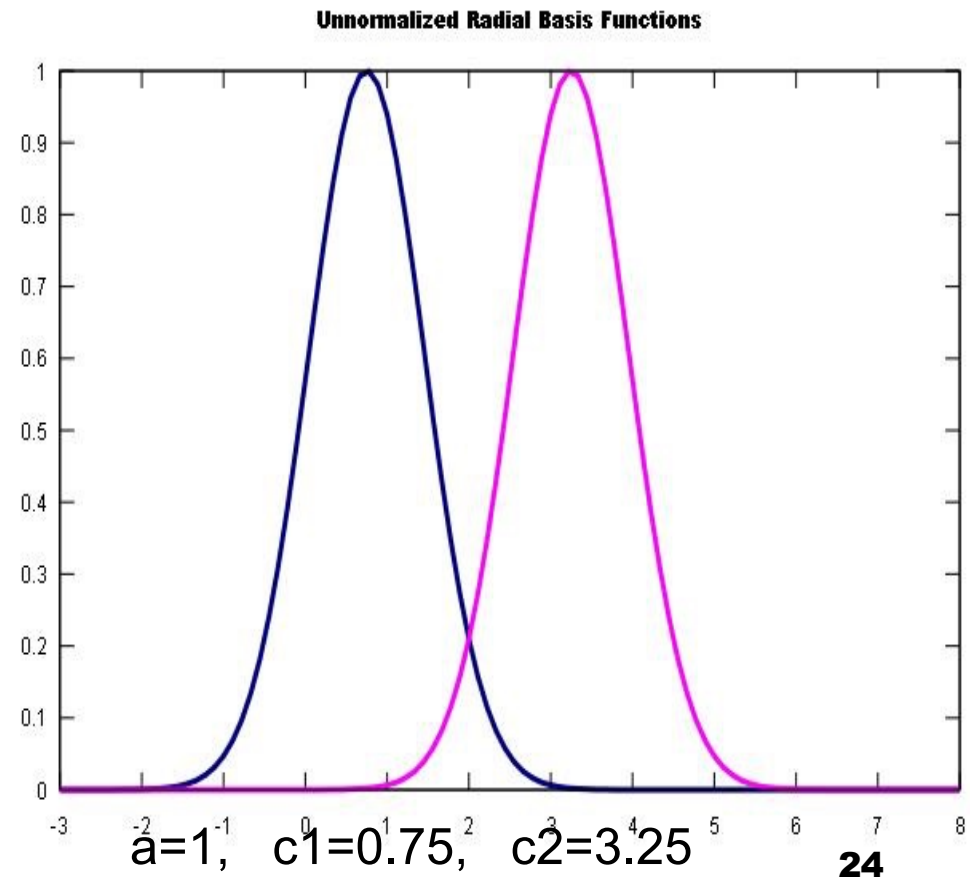
Normalized Gaussian curves with expected value μ and variance σ^2 .
The corresponding parameters are $a = 1/(\sigma\sqrt{2\pi})$, $b = \mu$, $c = \sigma$

$$f(x) = ae^{-\frac{(x-b)^2}{2c^2}} \qquad g(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Radial Basis Functions

- The popular output of radial basis functions is the Gaussian function:

$$\Phi(\|x - c_j\|) = \exp\left(-a \left(\frac{\|x - c_j\|}{\sigma_j}\right)^2\right)$$

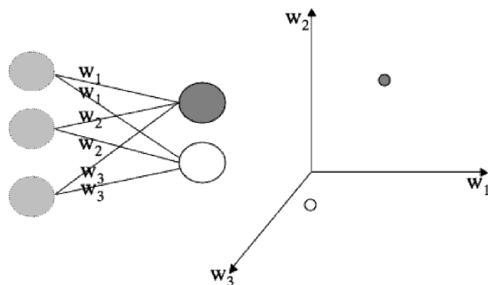


Radial Basis Functions

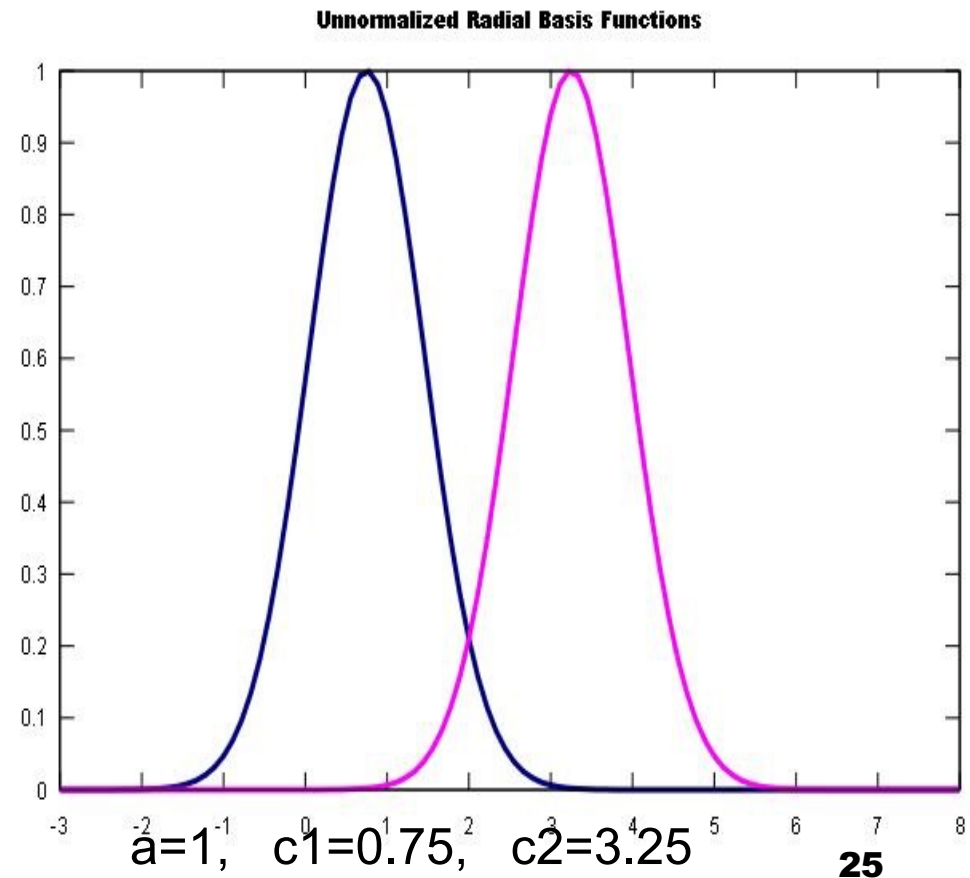
- The popular output of radial basis functions is the Gaussian function:

$$\Phi(\|x - c_j\|) = \exp\left(-a \left(\frac{\|x - c_j\|}{\sigma_j}\right)^2\right)$$

Sometimes the centers are called weights



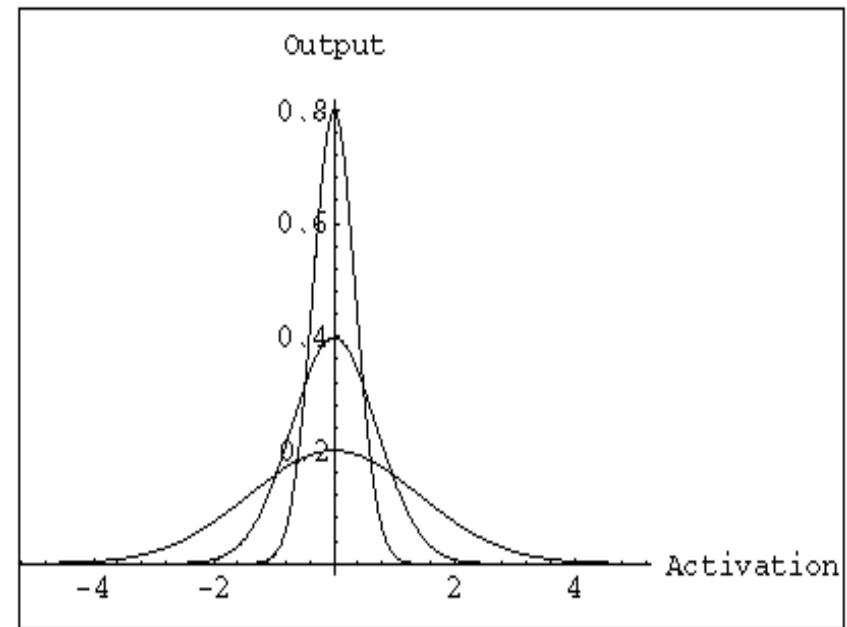
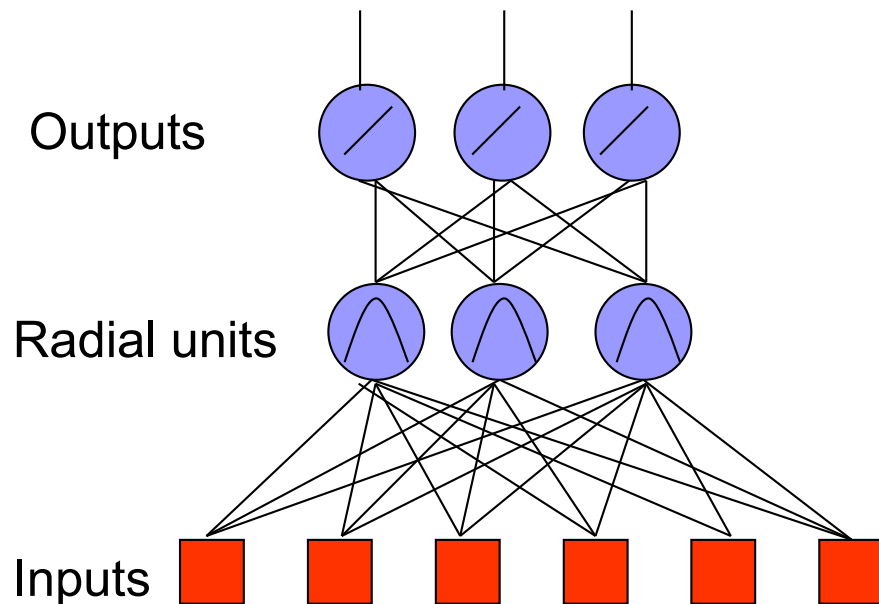
Weight Space = Input Space



Radial Basis Functions Network (RBFN)

■ Features

- One hidden layer
- The activation of a hidden unit is determined by a radial basis function

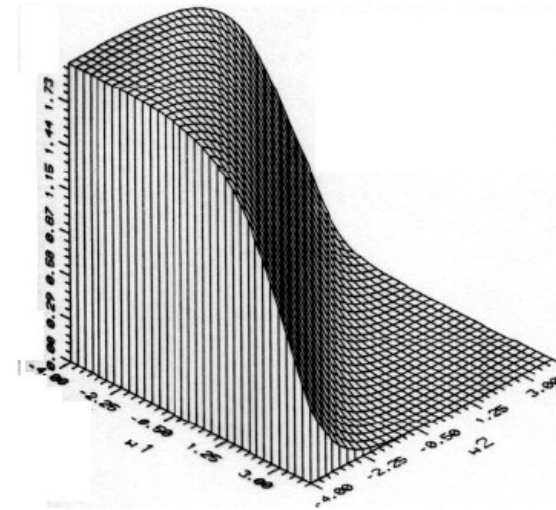


Sigmoidal vs. Gaussian Units

Sigmoidal unit:

$$y_j = \tanh\left(\sum_i w_{ji} x_i\right)$$

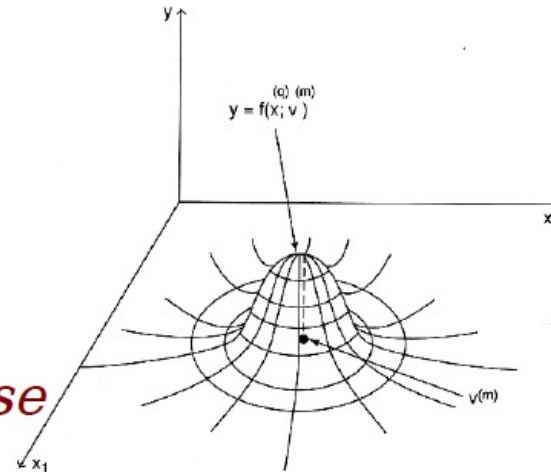
Decision boundary is a hyperplane



Gaussian unit:

$$y_j = \exp\left(\frac{-\|\vec{X} - \vec{\mu}_j\|^2}{\sigma_j^2}\right)$$

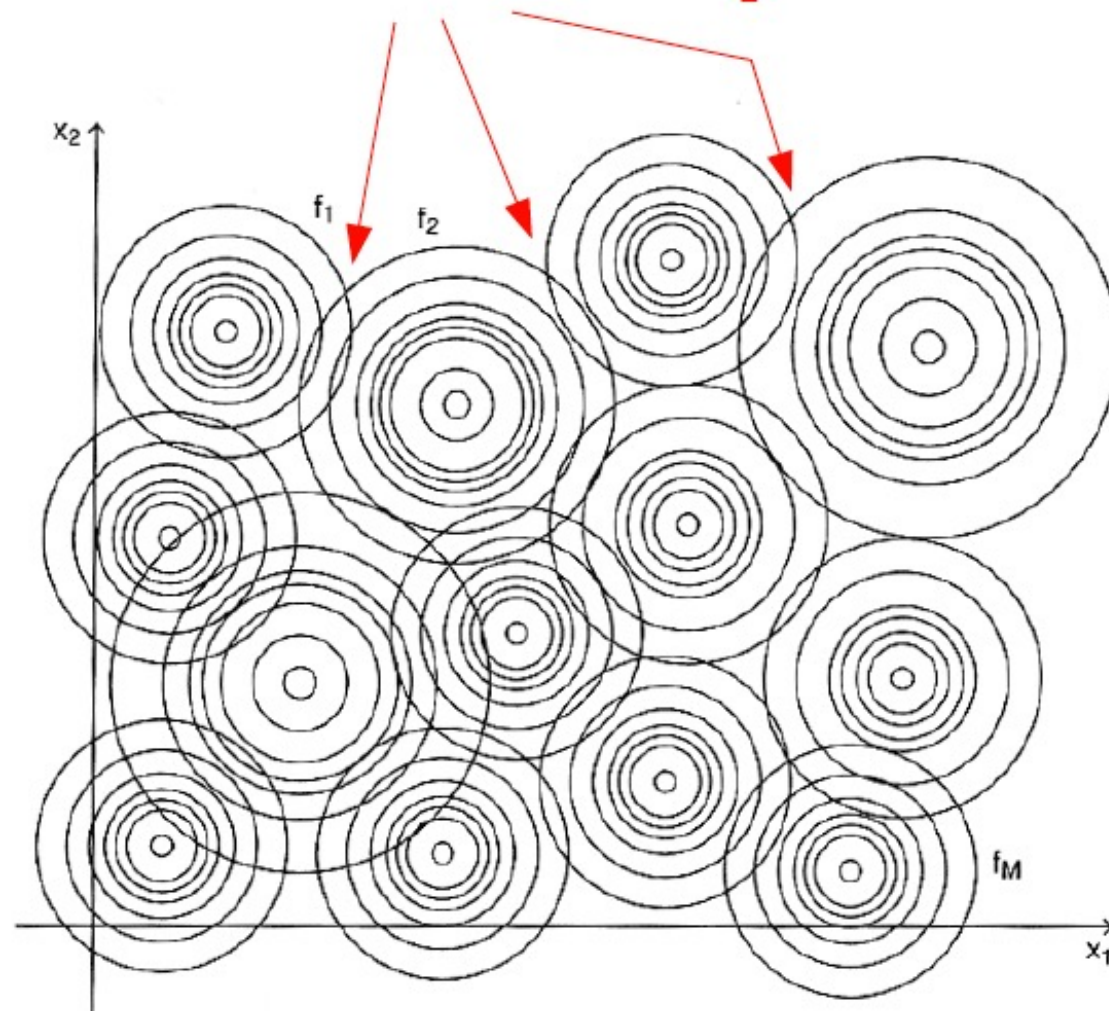
Decision boundary is a hyperellipse




Tiling the Input Space

Receptive fields in Neuroscience

Note: fields overlap



- 
- Generally, the hidden unit function is the Gaussian function
 - The output Layer is linear:

$$s(x) = \sum_{j=1}^K W_j \Phi(\|x - c_j\|)$$

$$\Phi(\|x - c_j\|) = \exp\left(-w_j \left(\frac{\|x - c_j\|}{\sigma_j}\right)^2\right)$$



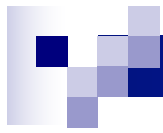
RBFN Learning

- The training is performed by deciding on
 - How many hidden nodes there should be
 - The centers and the sharpness of the Gaussians
- 2 steps (first unsupervised, second supervised)
 - In the 1st stage, the input data set is used to determine the parameters of the RBF
 - In the 2nd stage, RBFs are kept fixed while the second layer weights are learned (Simple BP algorithm like for MLPs or Perceptron)



Summary

- Neural networks are utilized as statistical tools
 - Adjust non linear functions to fulfill a task
 - Need of multiple and representative examples but fewer than in other methods
- Neural networks enable to model complex static phenomena (Feed-Forward) as well as dynamic ones (Recurrent NN)
- NN are good classifiers BUT
 - Good representations of data have to be formulated
 - Training vectors must be statistically representative of the entire input space
 - Unsupervised techniques can help
- The use of NN needs a good comprehension of the problem



Self Organising Feature Maps (SOM)

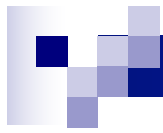
- Used for Unsupervised Learning
- Weights in neurons must represent a class of pattern

one neuron, one class



Four requirements for SOM

- Input pattern presented to all neurons and each produces an output.
- Output: measure of the match between input pattern and pattern stored by neuron.
- A competitive learning strategy selects neuron with largest response.
- A method of reinforcing the largest response

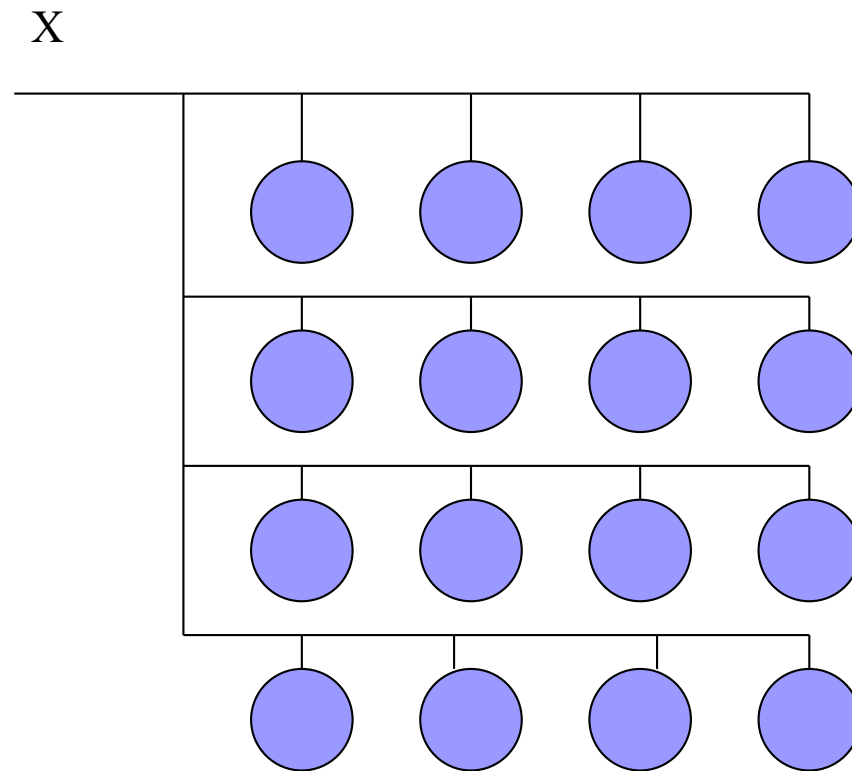


Architecture

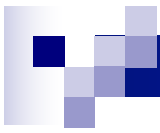
- The Kohonen network (named after Teuvo Kohonen) is a self-organising network proposed in the 1980s
- Neurons are usually arranged on a 2-dimensional grid
- Inputs are sent to all neurons
- There are no connections between neurons



Architecture



Kohonen network



Output value

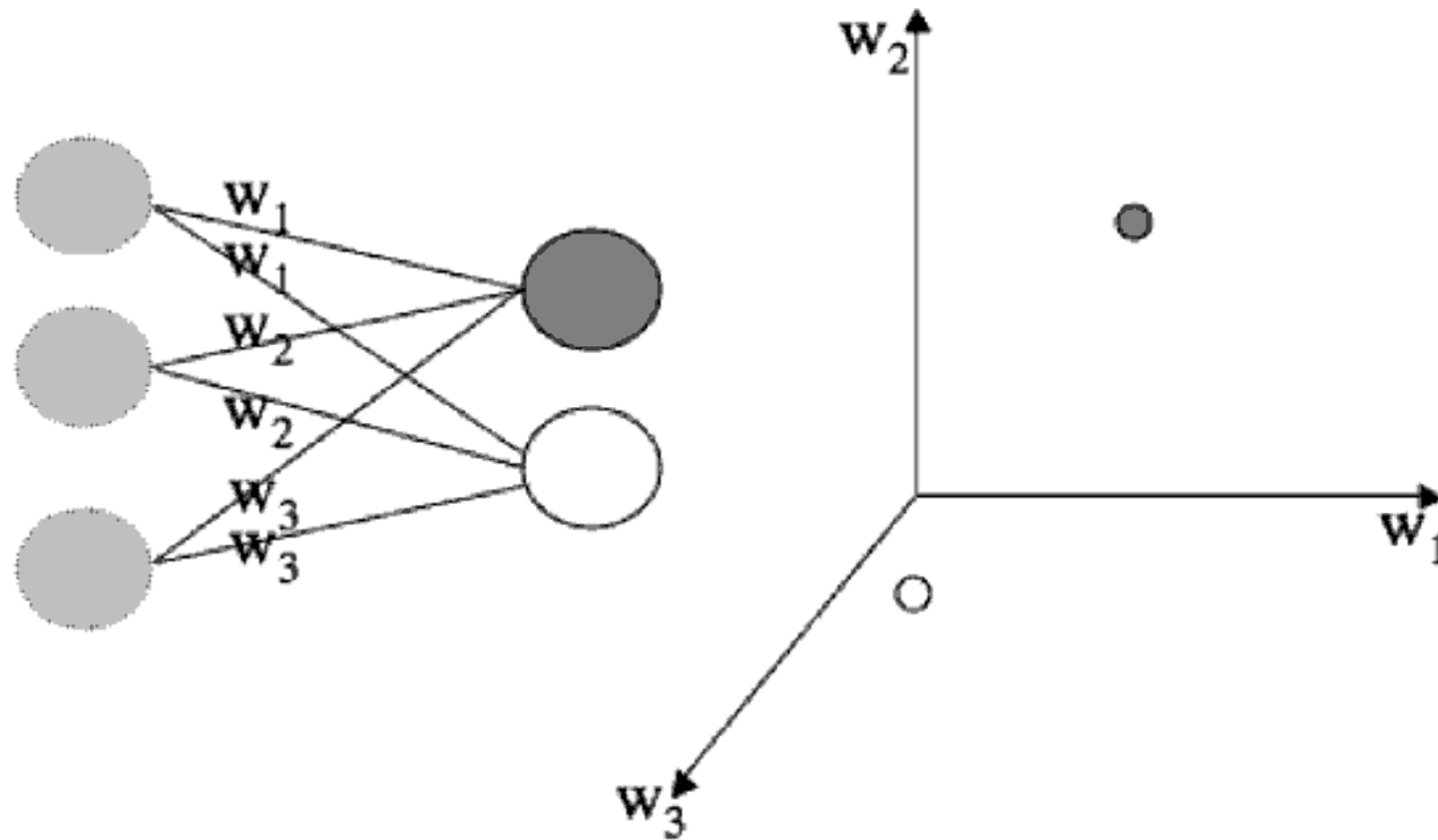
- The output of each neuron is the weighted sum
- There is no threshold or bias
- Input values and weights are normalized



“Winner takes all”

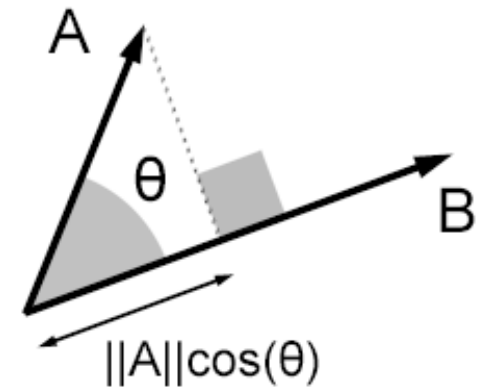
- Initially the weights in each neuron are random
- Input values are sent to all the neurons
- The outputs of each neuron are compared
- The “winner” is the neuron with the largest output value

Weight Space = Input Space



The linear combination is a dot product

$$\begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} \cdot \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix} = a_x \cdot b_x + a_y \cdot b_y + a_z \cdot b_z$$



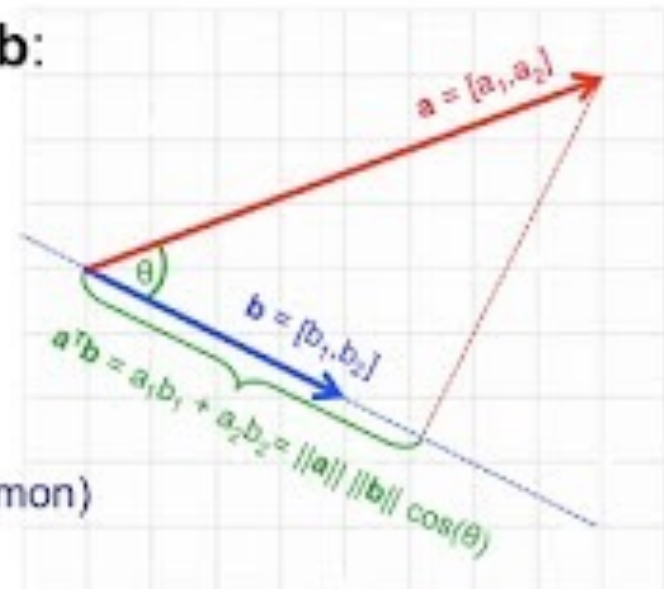
- Similarity of document vectors **a** and **b**:

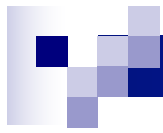
- $\mathbf{a} = [a_1 \ a_2 \ \dots \ a_d]$, $\mathbf{b} = [b_1 \ b_2 \ \dots \ b_d]$

- $\mathbf{a}^T \mathbf{b} = a_1 b_1 + \dots + a_d b_d = \sum_i a_i b_i$

- Geometrically:

- length of projection of **a** onto **b**
 - highest if **a, b** point in the same direction
 - zero if **a, b** are orthogonal (no words in common)
 - cosine of the angle between **a** and **b**

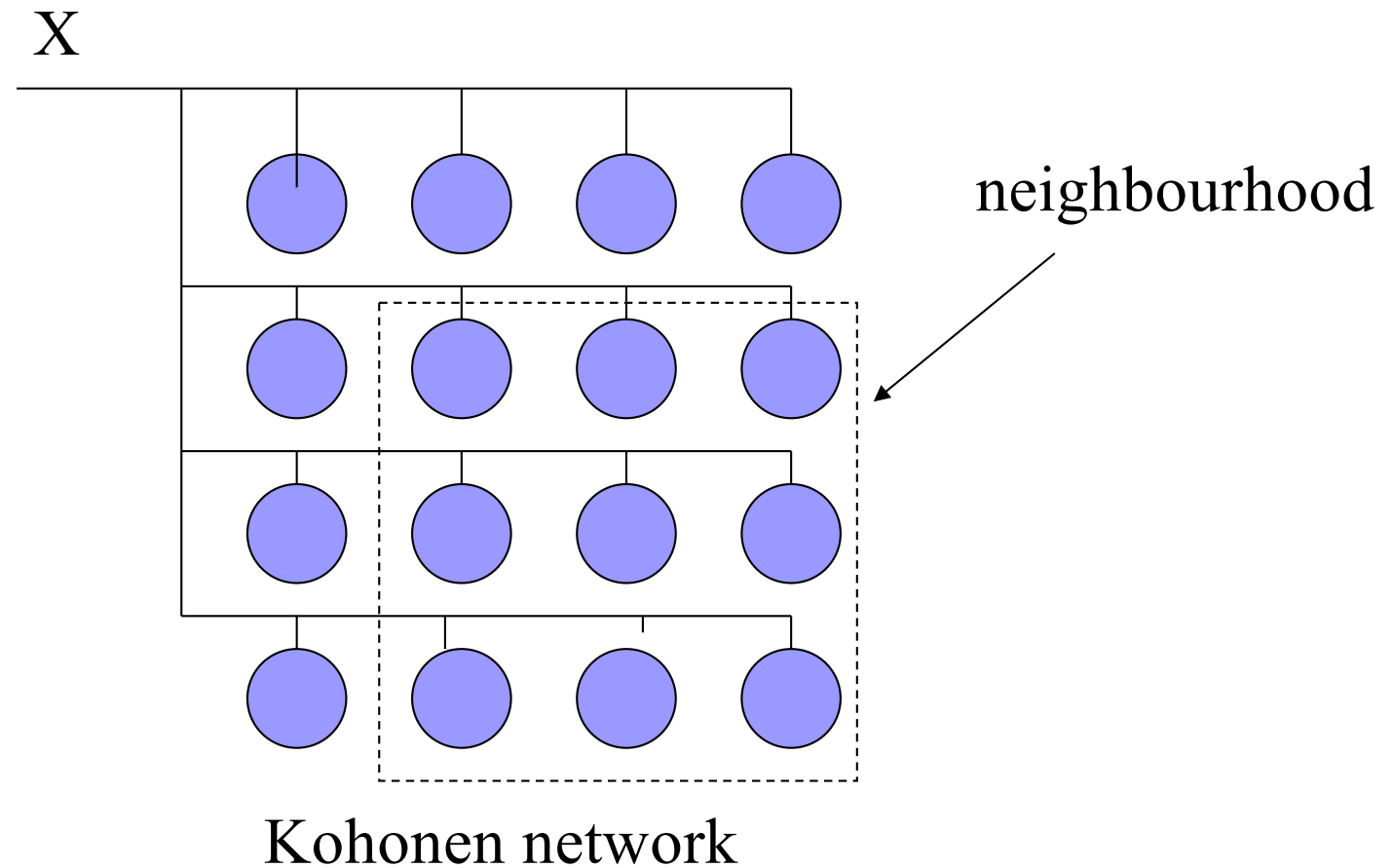




Training

- Having found the winner, the weights of the winning neuron are adjusted
- Weights of neurons in a surrounding neighbourhood are also adjusted

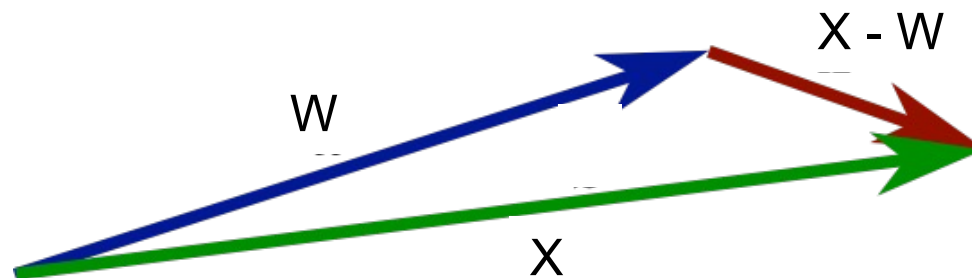
Neighbourhood

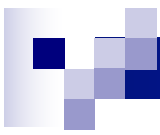


Training

- As training progresses the neighbourhood gets smaller
- Weights are adjusted according to the following formula:

$$w' = w + \alpha(x - w)$$





Weight adjustment

- The learning coefficient (alpha) starts with a value of 1 and gradually reduces to 0
- This has the effect of making big changes to the weights initially, but no changes at the end
- The weights are adjusted so that they more closely resemble the input patterns



Weight adjustment details

$$\mathbf{W}_v(s + 1) = \mathbf{W}_v(s) + \Theta(u, v, s) \alpha(s)(\mathbf{D}(t) - \mathbf{W}_v(s))$$

Where:

- s is the step index,
- t an index into the training sample,
- u is the index of the best matching unit for $\mathbf{D}(t)$,
- $\alpha(s)$ is a monotonically decreasing learning coefficient
- $\mathbf{D}(t)$ is the input vector
- $\Theta(u, v, s)$ is the neighborhood function which gives the distance between the neuron u and the neuron v in step s



Example

- A Kohonen network receives the input pattern 0.6 0.6 0.6.
- Two neurons in the network have weights 0.5 0.3 0.8 and -0.6 -0.5 0.6.
- Which neuron will have its weights adjusted and what will the new values of the weights be if the learning coefficient is 0.4?



Answer

$$w' = w + \alpha(x - w)$$

The weighted sums are 0.96 and -0.3 so the first neuron wins.
($0.6 \times 0.3 + 0.6 \times 0.5 + 0.6 \times 0.8 = 0.18 + 0.30 + 0.48 = 0.96$)

The weights become:

$$w1 = 0.5 + 0.4 \times (0.6 - 0.5)$$

$$w1 = 0.5 + 0.4 \times 0.1 = 0.5 + 0.04 = 0.54$$

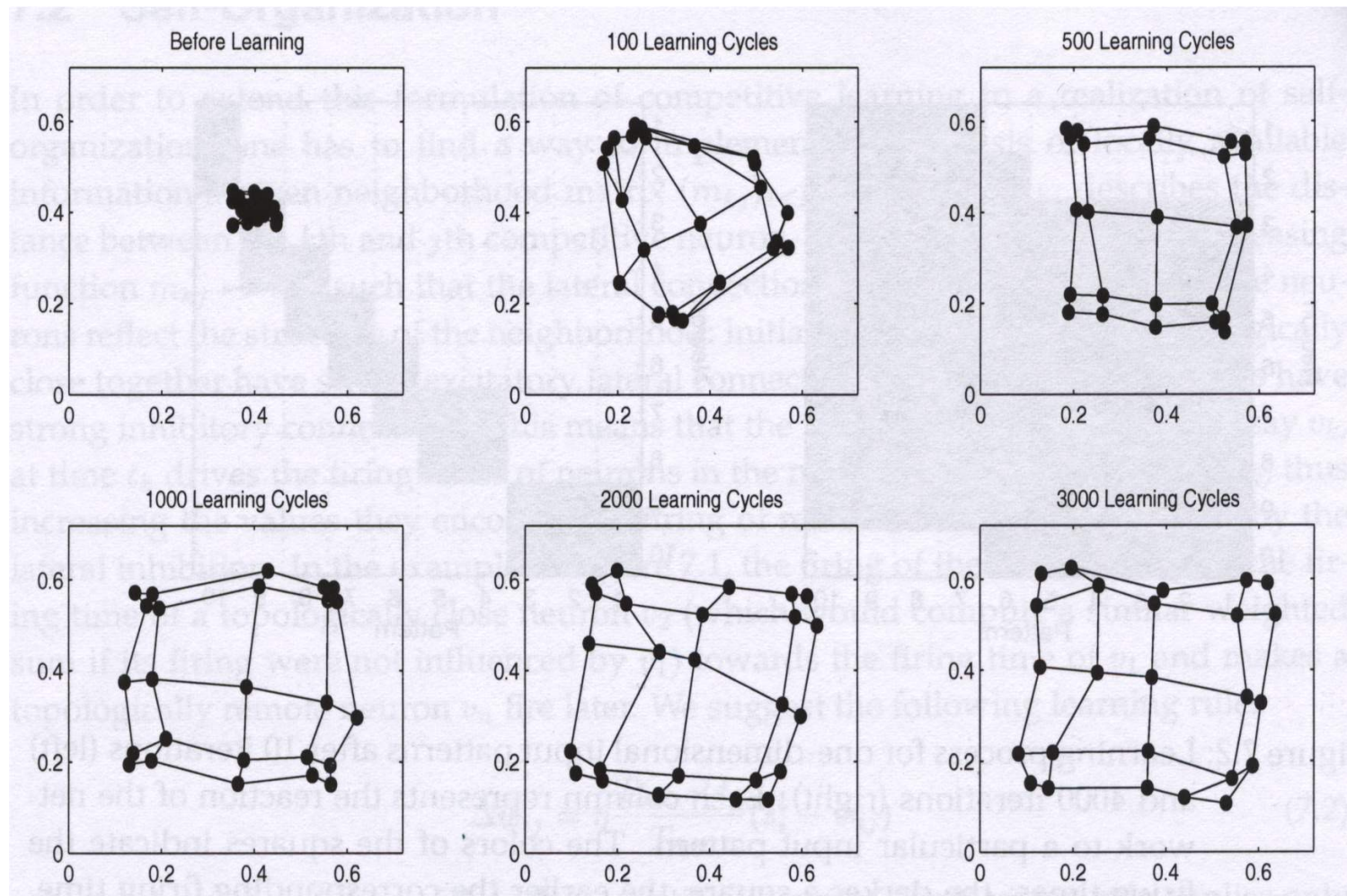
$$w2 = 0.3 + 0.4 \times (0.6 - 0.3)$$

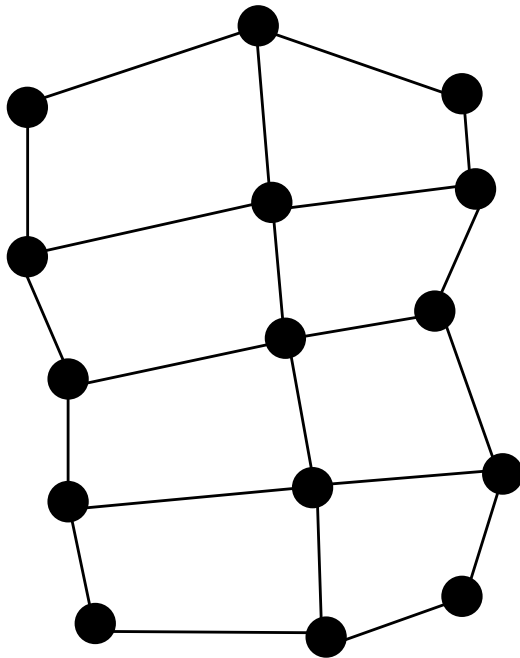
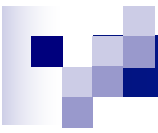
$$w2 = 0.3 + 0.4 \times 0.3 = 0.3 + 0.12 = 0.42$$

$$w3 = 0.8 + 0.4 \times (0.6 - 0.8)$$

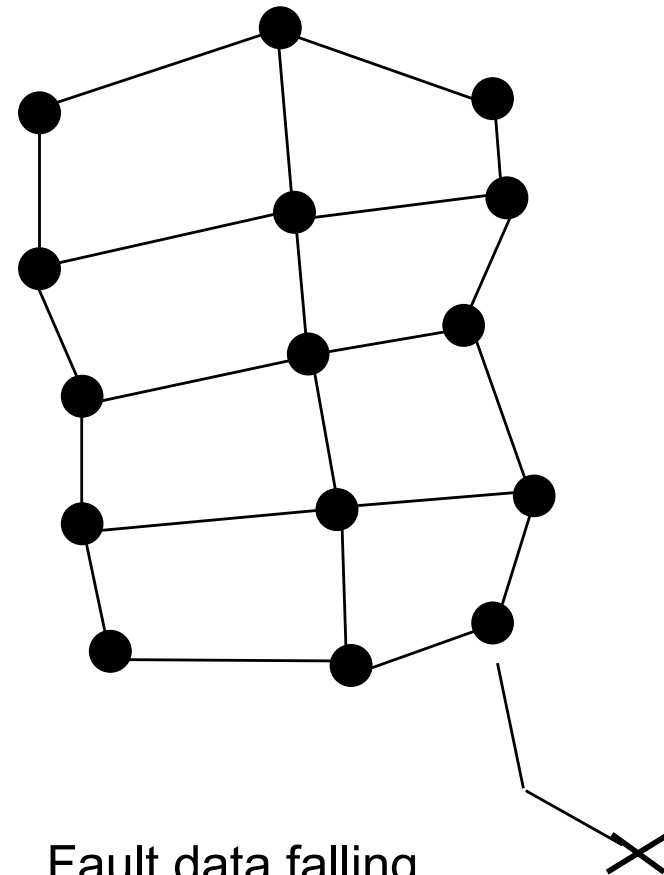
$$w3 = 0.8 - 0.4 \times 0.2 = 0.8 - 0.08 = 0.72$$

Visualizing a SOM





Kohonen network
representing
'Normal' space



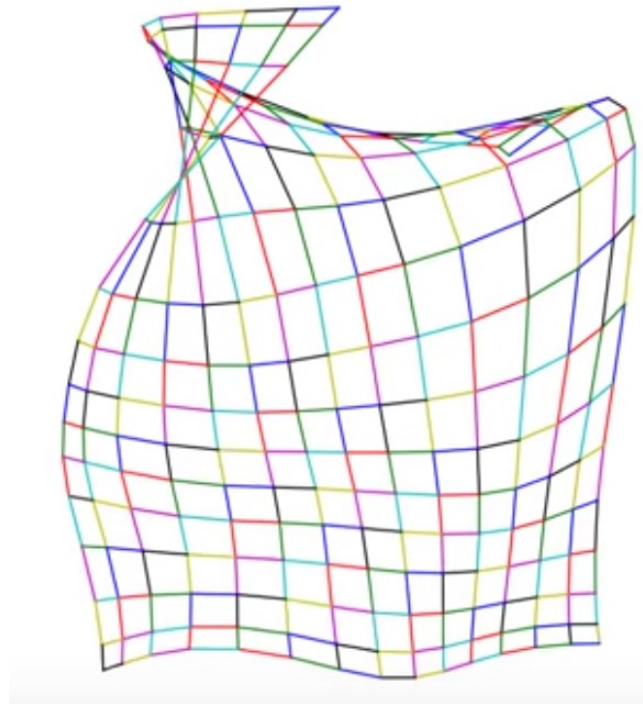
Fault data falling
outside
'Normal' space



Summary

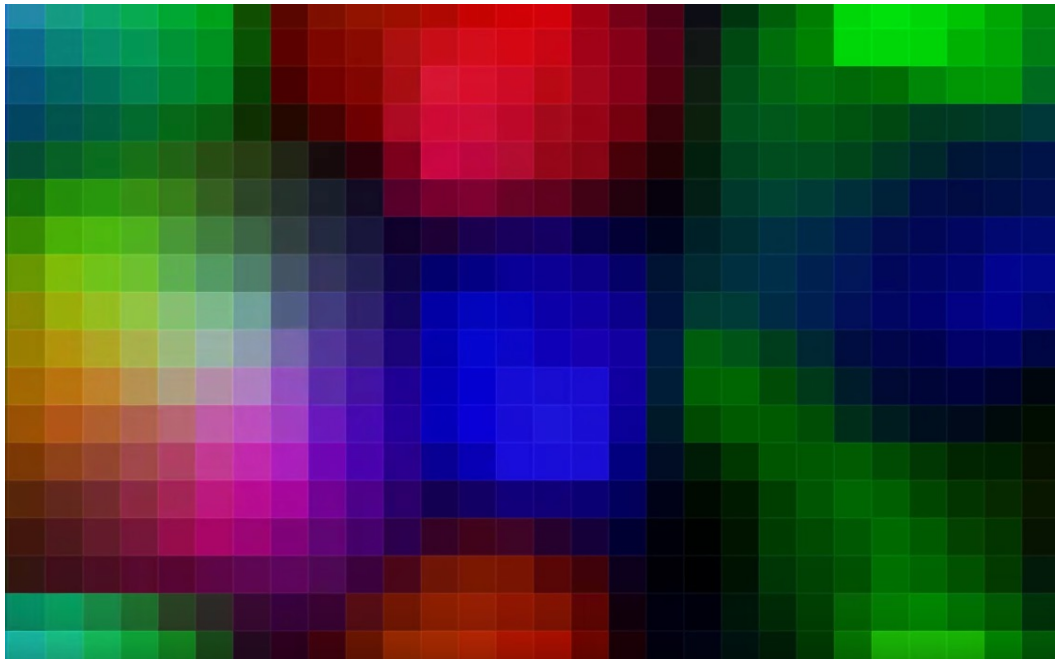
- The Kohonen network is self-organising
- It uses unsupervised training
- All the neurons are connected to the input
- A winner takes all mechanism determines which neuron gets its weights adjusted
- Neurons in a neighbourhood also get adjusted

Visualizing a SOM



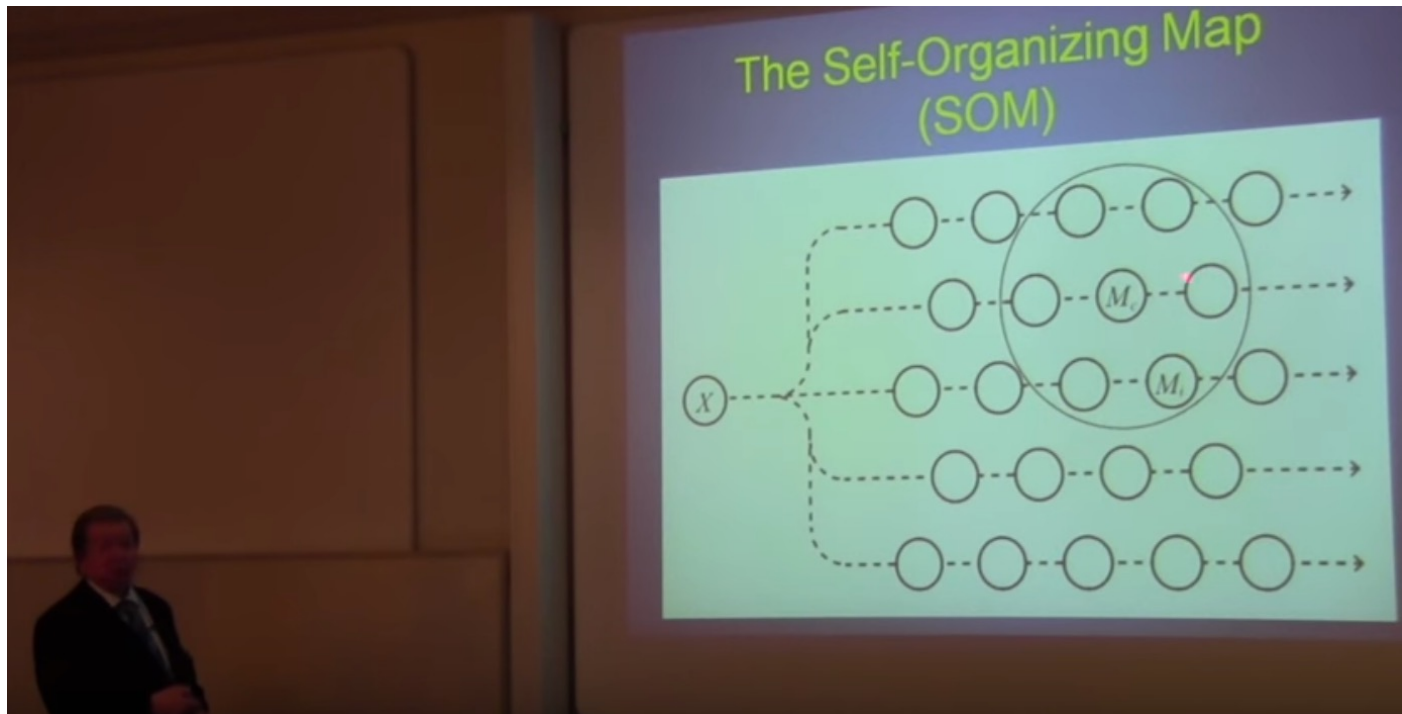
<https://youtu.be/lixbH1gDhsg>

Visualizing a SOM



<https://youtu.be/dASyjPQtbS8>

Prof. Kohonen explains SOMs



<https://youtu.be/iWPhGKniTew>

Prof. Teuvo Kohonen explains Self-Organizing Maps in May 2014

Watch from 12:05 to 16:05, he still uses his classical slides!