Chapter 3

Knowledge discovery from web data

PROFESSORS

Juan D. Velásquez
Víctor Rebolledo L.

Outline

1. The KDD Process
2. Data sources and cleaning
3. Data consolidation and information repositories
4. Data mining
5. **Tools for mining data**
6. Using data mining to extract knowledge
7. Validation of the extracted knowledge
8. Mining the web

Section 3.5

- Tools for Data Mining

Tools for Data Mining

- Artificial Neural Networks
- Self-Organizing Feature Maps
- K-Means
- Decisions Trees
- Bayesian network
- K-Nearest Neighbor
- Support Vector Machines

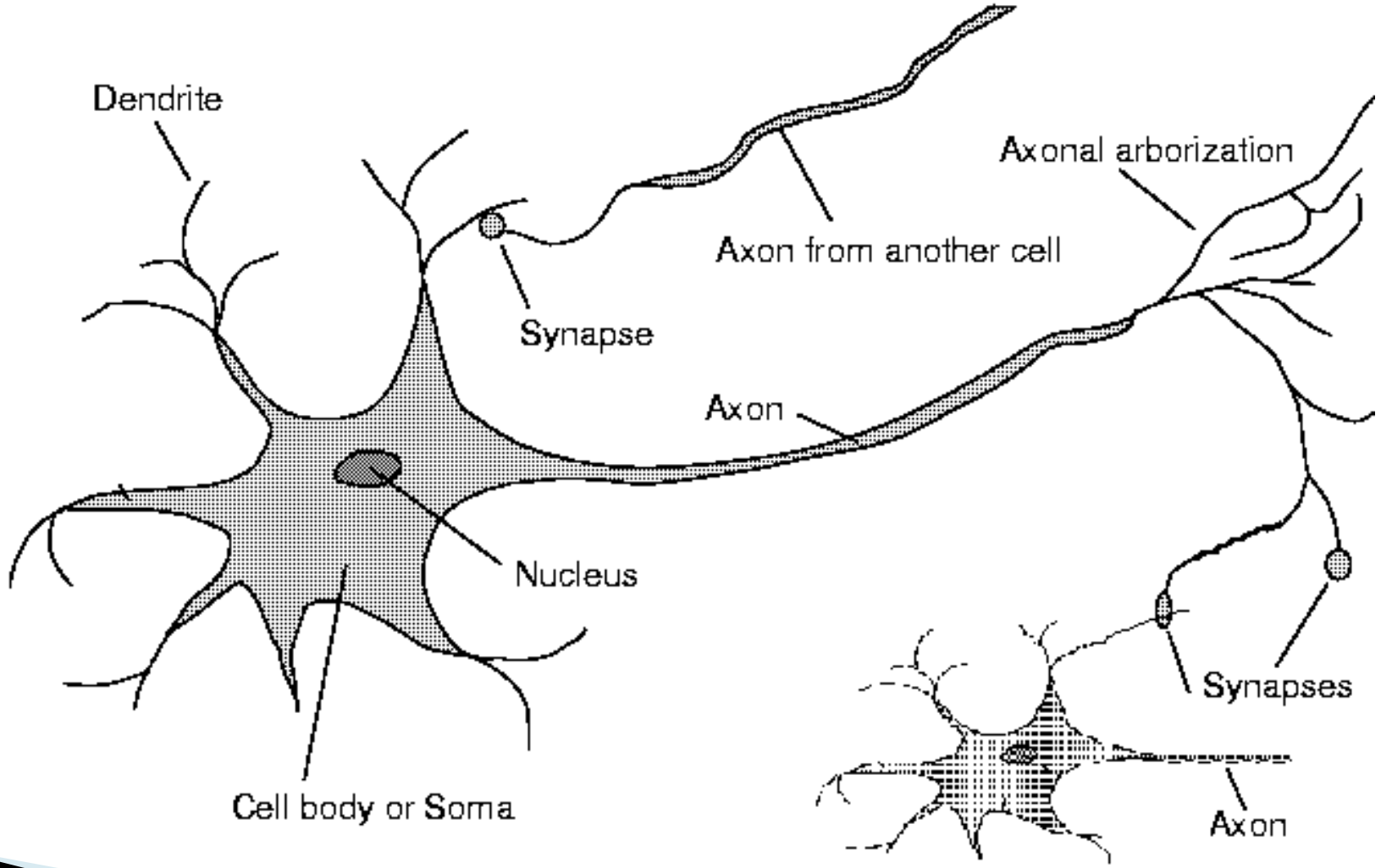
Artificial Neural Networks

Inspired on a biological model ...

What is connectionism/neural networks?

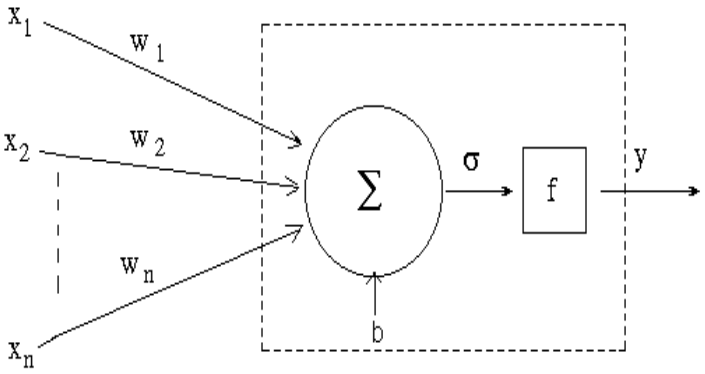
- A **simplified model** of how natural neural systems work. Neural Networks (NNs) **simulate natural** information processing tasks from **human brain**.
- A NN model consists of **neurons and connections between neurons** (synapses).
- Characteristics of Human Brain:
  - **Contains 10^{11} neurons and 10^{15} connections**
  - **Each neuron may connect to other 10,000 neurons.**
  - **Human can perform a task of picture naming in about 500 miliseconds**

Biological Neural Networks

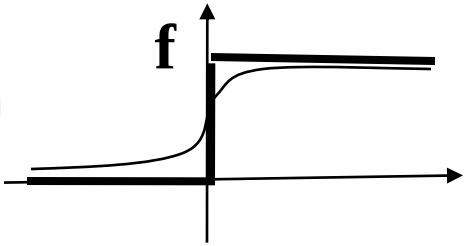


(Picture from G. Kendall, lect. notes Univ. of Nottingham)

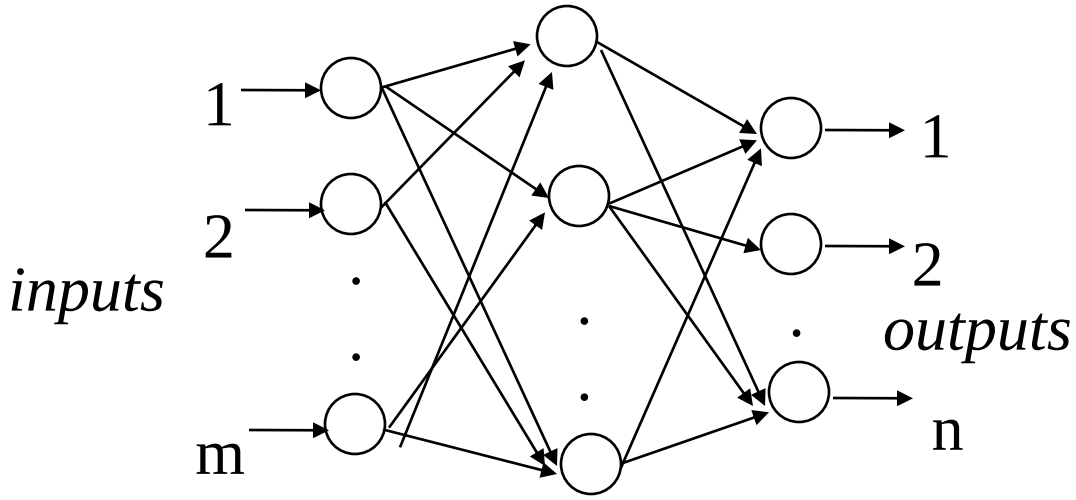
What is an artificial neural network?



$$\sigma = \sum_{i=1}^{n} w_i x_i + b$$



$$y = f(\sigma)$$



$$f : \Re^m \rightarrow \Re^n$$

Artificial neural nets (ANNs):

An **Artificial Neural Network** is an **interconnected assembly** of simple processing elements, **units or nodes** (neurons), whose functionality is inspired by the **functioning of the natural neuron from brain**.

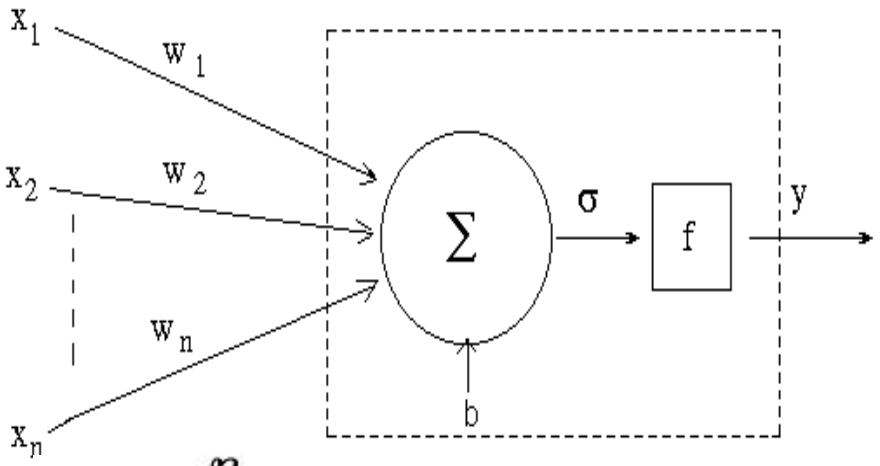
The processing ability of the neural network is stored in **the inter-unit connection** strengths, **or weights**, obtained by a **process of learning from a set of training patterns**.

Artificial neural nets (ANNs):

- The **units** (individual neurons) **operate only locally** on the inputs they receive via connections.

- ANNs undergo some sort of **"training"** whereby the connection **weights are adjusted on the basis of presented data**. In other words, **ANNs "learn" from examples** *(as children learn to recognize dogs from examples of dogs)* and exhibit some **generalization capability** *beyond the training data* (for other data than those included in the training set).

Formal neuron (McCulloch & Pitts)

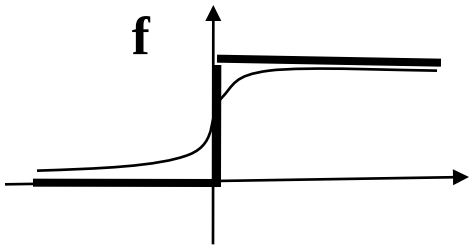


A single neuron has 6 components:

1. Input “x”
2. Weights “w”
3. Bias “b” (Threshold = -b)
4. Activation function “f”
5. Input function σ
6. Output “y”

$$\sigma = \sum_{i=1}^{n} w_i x_i + b$$

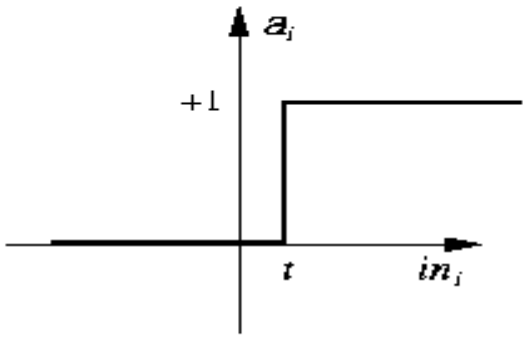
$$y = f(\sigma)$$



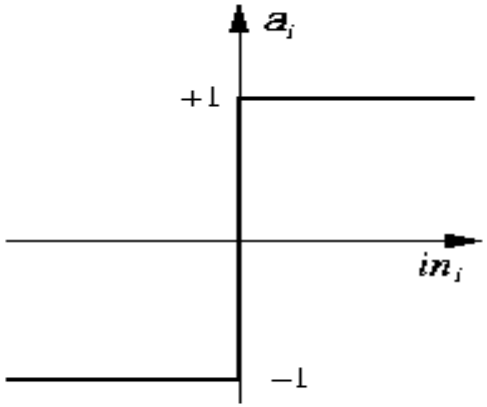
$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

McCulloch & Pitts (1943) recognised as the designers of the first neuron (and neural network) model

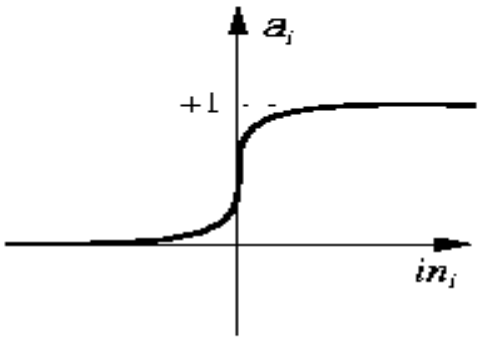
Formal neuron – Activation Functions



Step function



Sign function



Sigmoid function

$$Step_t(x) = \begin{cases} 1 & : \quad x \geq t \\ 0 & : \quad x < t \end{cases}$$

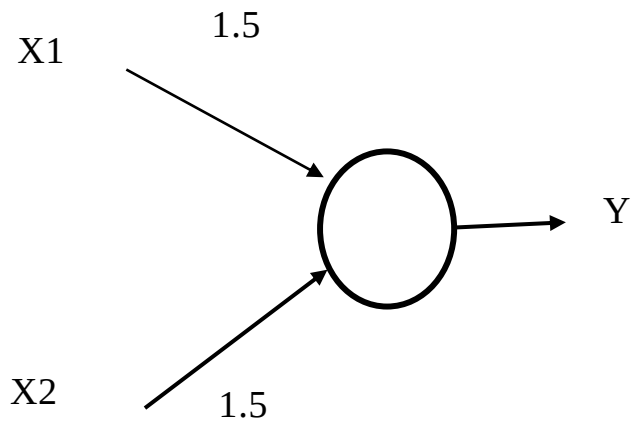
$$Sign(x) = \begin{cases} +1 & : \quad x \geq 0 \\ -1 & : \quad x < 0 \end{cases}$$

$$Sigmoid(x) = \frac{1}{1 + e^{-kx}}$$

$$Id(x) = x$$

$$Lin(x) = k * x$$

A neuron which implements OR function:



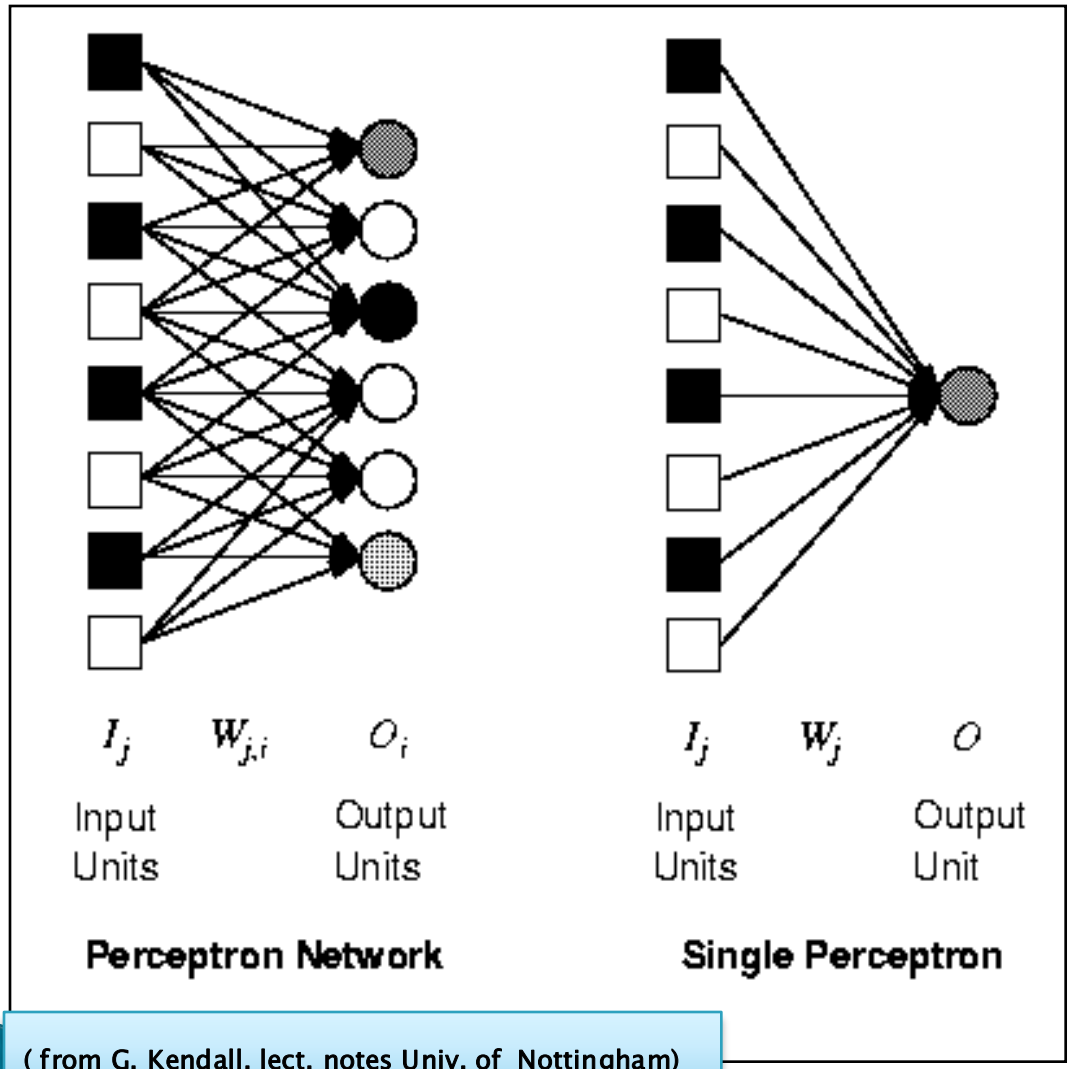
OR

X1	X2	Y
1	1	1
1	0	1
0	1	1
0	0	0

Threshold = 1

Bias = -1 (Threshold = - Bias)

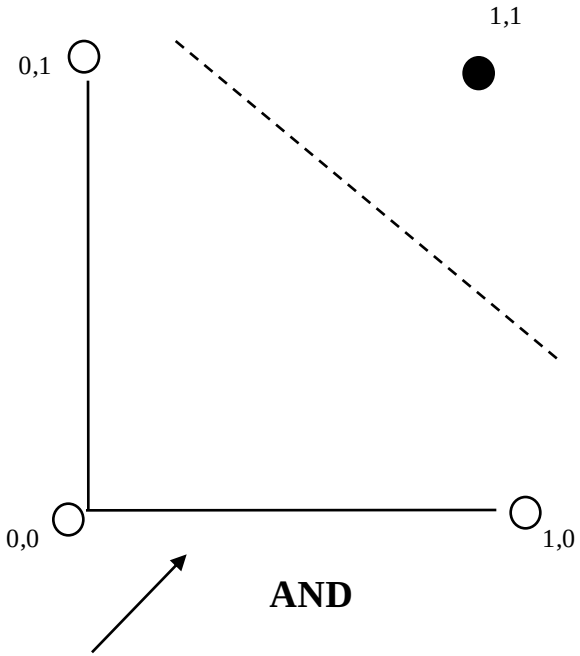
Perceptron

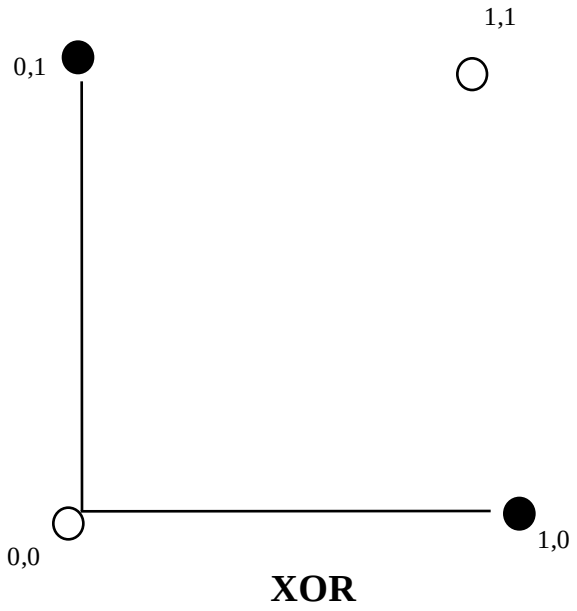


(from G. Kendall, lect. notes Univ. of Nottingham)

- Synonym for **Single-Layer, Feed-Forward Network**
- **First Studied in the 50's (Rosenblatt)**
- Other networks were known about but the **Perceptron** was the only **one capable of learning** and thus all **research was concentrated** in this area

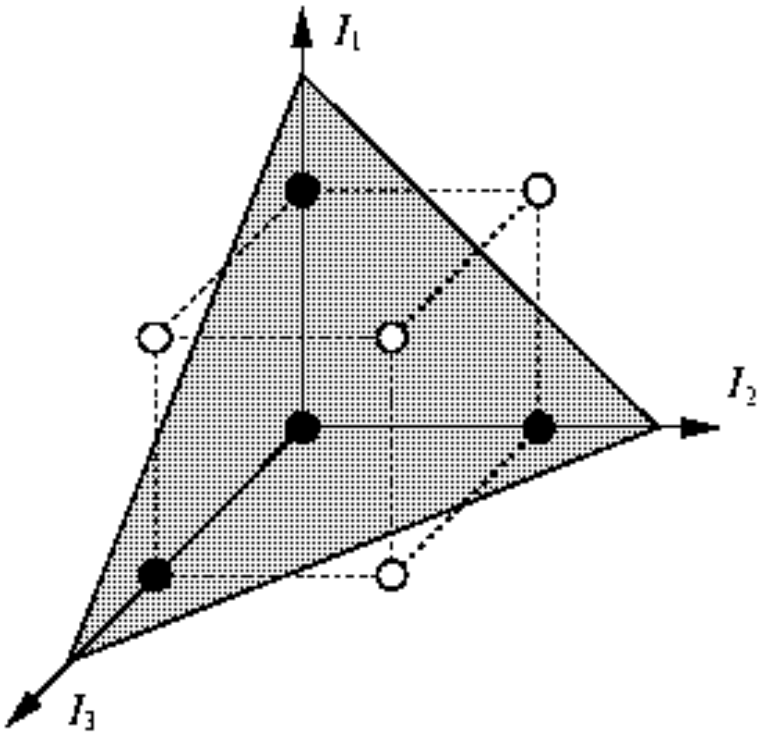
What can perceptrons learn?



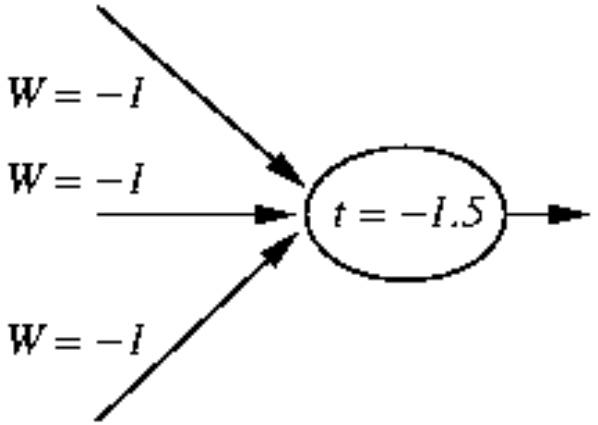


- Functions that can be separated in this way are called ***Linearly Separable*** (XOR is not Linearly Separable)
- **A PERCEPTRON can learn (represent) only Linearly Separable functions.**

What can perceptrons represent?



(a) Separating plane

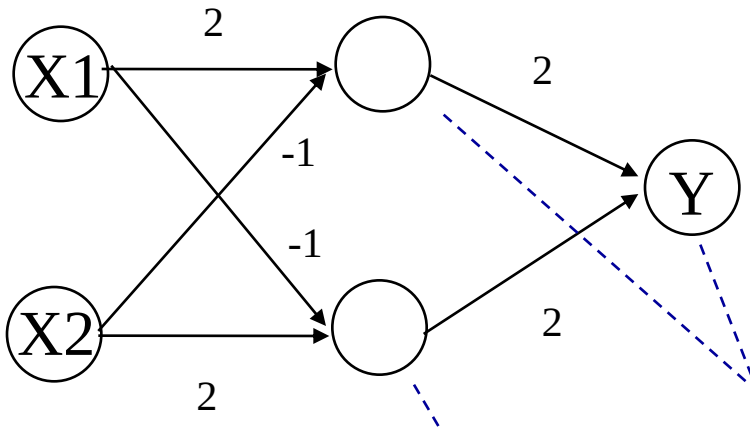


(b) Weights and threshold

Linear Separability is also possible in more than 3 dimensions – but it is harder to visualize

XOR problem – not linearly separable

One neuron layer is not enough, we should introduce an **intermediate (hidden) layer.**



Threshold for all nodes = 1.5

XOR

X1	X2	Y
1	1	0
1	0	1
0	1	1
0	0	0

$$Y = X_1 \text{ XOR } X_2 = (X_2 \text{ AND NOT } X_1) \text{ OR } (X_1 \text{ AND NOT } X_2)$$

Training a perceptron

AND

X1	X2	D
1	1	1
1	0	0
0	1	0
0	0	0

Training Dataset $\{ (\underline{x}(i), d(i)), i=1,\ldots,p\}$

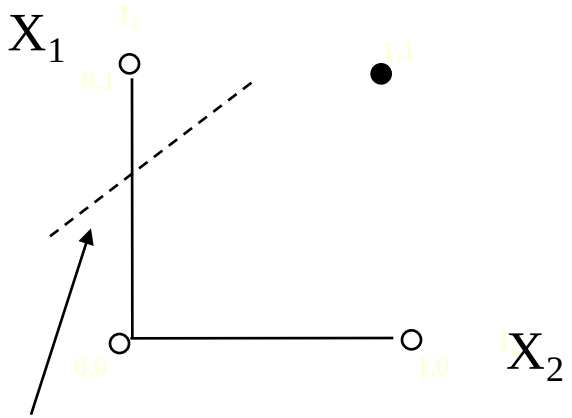
$p = 4$

Training set $= \{ ((1,1),1), ((1,0),0), ((0,1),0), ((0,0),0) \}$

The training technique is called Perceptron Learning Rule.

Perceptron Learning

After weight initialization
(First Epoch)



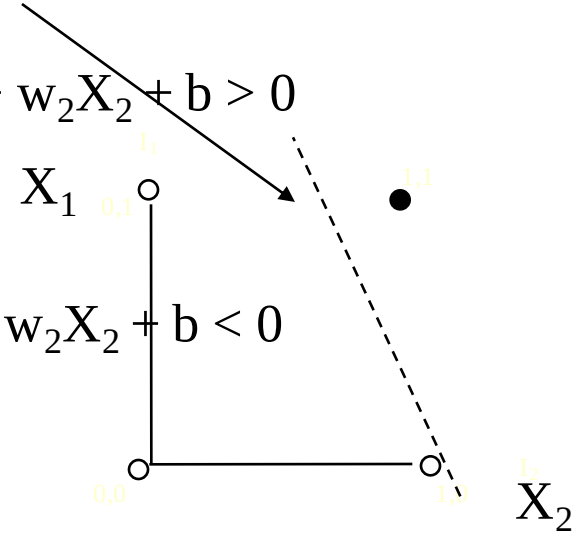
Separation line

$w_1X_1 + w_2X_2 + b = 0$

$w_1X_1 + w_2X_2 + b > 0$

$w_1X_1 + w_2X_2 + b < 0$

At Convergence



Training a Perceptron:
Main Idea

- Vectors from the training set are presented to the **Perceptron network one after another** (cyclic or randomly):

  $(\underline{x}(1), d(1)), (\underline{x}(2), d(2)), \ldots, (\underline{x}(p), d(p)),$
  $(\underline{x}(p+1), d(p+1)), \ldots$

- **If the network's output is correct, no change is made.**
- Otherwise, the **weights and biases are updated** using the ***Perceptron Learning Rule***.
- **An entire pass** through all of the input training vectors is called an **Epoch.**
- When such **an entire pass of the training set** has occurred **without error**, training is **complete**

The Perceptron learning algorithm

1. **Initialize the weights** and **threshold** to ***small random numbers***.
2. At time **step t** present a vector to the neuron inputs and calculate the perceptron **output y(t).**
3. **Update the weights** and **biases** as follows:

$$w_j(t+1) = w_j(t) + \eta(d(t) - y(t))x_j$$

$$b(t+1) = b(t) + \eta(d(t) - y(t))$$

   - d(t) is the desired output
   - y(t) is the computed output
   - t is the step/iteration number
   - η is the gain or step size (Learning Rate), where $0.0 < \eta <= 1.0$
4. Repeat steps 2 and 3 until:
   - The iteration error is less than a user-specified error threshold
   - Or a predetermined number of iterations have been completed.

The **perceptron learning algorithm** developed originally by F. Rosenblatt in the late 1950s.

Example

$\eta = 1$
$Y = f(\sigma) = Id(\sigma)$

t = 0

…

t = 9

t	INPUTS				d(t)	y(t)	E	WEIGHTS				
	x1	x2	x3	x4				b(t)	w1(t)	w2(t)	w3(t)	w4(t)
0								0	0	0	0	0
1	0	0	0	1	-1	0	1	-1	0	0	0	-1
2	1	1	1	0	1	1	2	0	1	1	1	-1
3	1	1	1	1	1	2	0	0	1	1	1	-1
4	0	0	1	1	-1	0	1	-1	1	1	0	-2
5	0	0	0	0	1	-1	2	0	1	1	0	-2
6	0	1	0	1	-1	-1	0	0	1	1	0	-2
7	1	0	0	0	1	1	0	0	1	1	0	-2
8	1	0	1	1	1	-1	2	1	2	1	1	-1
9	0	1	0	0	-1	2	3	0	2	0	1	-1

Training a perceptron

- **Learning only occurs when an error is made**, **otherwise the weights are left unchanged!!.**
- During training, **it is useful to measure the performance of the network** as it attempts **to find the optimal weight set.**
- A **common error measure used is sum-squared errors** (computed over all of the input vector / output vector pairs in the training set):

$$E = \frac{1}{2} \sum_{i=1}^{p} \|d_i(t) - y_i(t)\|^2$$

- where "p" is the **number of input/output vector pairs** in the training set.
- **η - Learning rate** – Dictates how **quickly the network converges.**
- It is set by a matter of experimentation (usually small – e.g. 0.1)

Two types of network training

- **Sequential mode**
  - *on-line or per-pattern*
  - Weights updated **after each pattern is presented** (**Perceptron** is in this class)

- **Batch mode**
  - *off-line or per-epoch*
  - Weights updated **after all patterns are presented**

Learning

- Training from data set, adaptation
  - Extracts **principles from training data** set in order to **generalize to other data**
- The purpose of learning is to **minimize error**:
  - on the **training data set**
  - on the **testing set** (prediction errors)!!!
- Two main types of Neural Network LEARNING:
  - **Supervised learning**
    - have a teacher, telling you what is the **output (target)** for a given input pattern (Perceptron, Delta, Back propagation)
  - **Unsupervised learning**
    - no teacher, learn by itself (SOMs)

Adaline Learning (Delta Rule)

- From it we can better **understand the Perceptron learning rule**, and the more general **BackPropagation learning**
- **Adaline learning** was developed by Widrow and Hoff (1960).
- **ADALINE** is an acronym for **ADAptive LInear Neuron**
  - neurons in the network have *linear activation functions*
- The **Adaline learning rule**
  - also known as the **Delta rule** or the **Widrow-Hoff rule**
  - It is a training rule that **minimizes the output error** using (approximate) **gradient descent method**

Key difference

between Delta rule and Perceptron training rule

- The **Perceptron training rule** converges **after a finite number of iterations** to a **solution that perfectly classifies the training data**, provided the training examples are linearly separable.
- The **Delta rule** converges only **asimptotically** toward the **minimum error solution**, possibly requiring **unbounded time**, but converges regardless of whether the training data are linearly separable or not.
- The Perceptron rule **updates weights based** on the error in the **thresholded Perceptron output,**
- whereas the Delta rule **updates weights based** on the error considering a **linear activation function for neurons.**

Adaline Learning

- After all training pattern vectors xi (i=1,...,p) are presented, the correction to apply to the weights is proportional to the error E(t):

$$E(t) = \frac{1}{2}\sum_{i=1}^{p}[d_i(t) - f(\vec{w}(t)\vec{x}_i)]^2$$

- Our purpose is to find the vector w which minimizes E(t). At each step:

$$w(t+1) = w(t) + \Delta w(t)$$

- In gradient descent techniques:

$$\vec{w}(t+1) = \vec{w}(t) - \eta \nabla E(\vec{w})$$

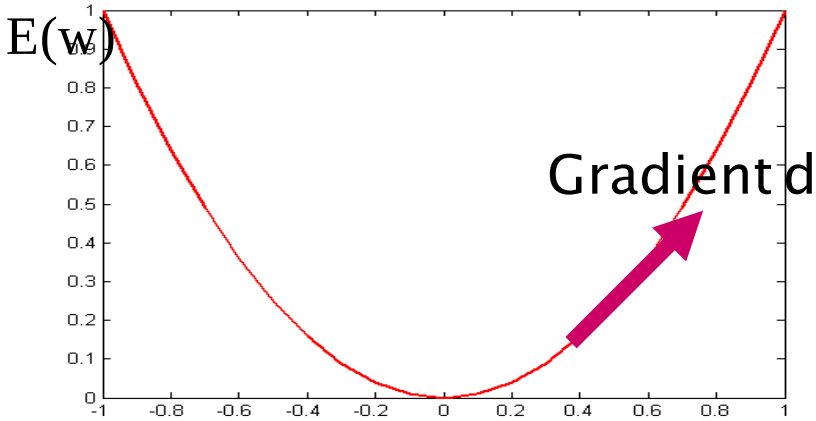
- $\eta > 0$ learning rate

Adaline Learning

- By analogy, **gradient method** can be **compared with a ball rolling down from a hill**:
  - the ball will **roll down and finally stop at the valley**.
  - Gradient direction is the **direction of uphill** (in the Figure – **E(w) one dimensional case**)
- In a gradient descent algorithm, the ball goes in the **opposite direction to the gradient**, i.e., we have

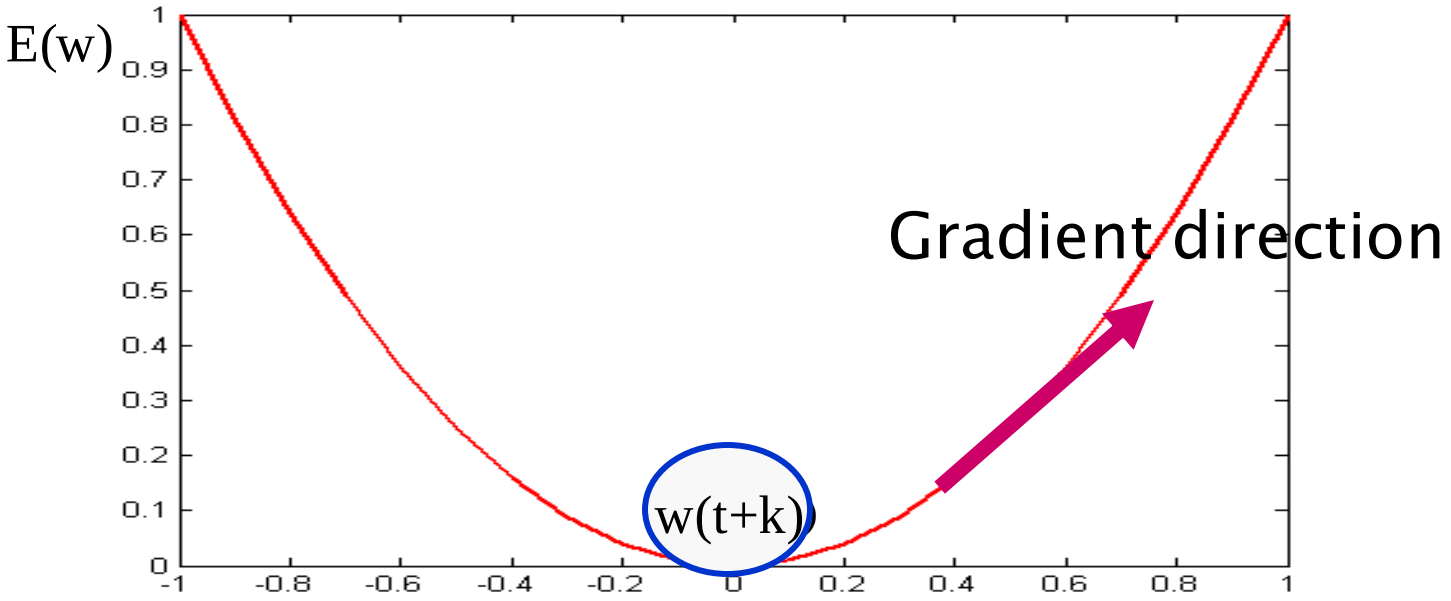
$$\vec{w}(t+1) = \vec{w}(t) - \eta \nabla E(\vec{w})$$

- therefore the ball goes downhill since – E'(w(t)).

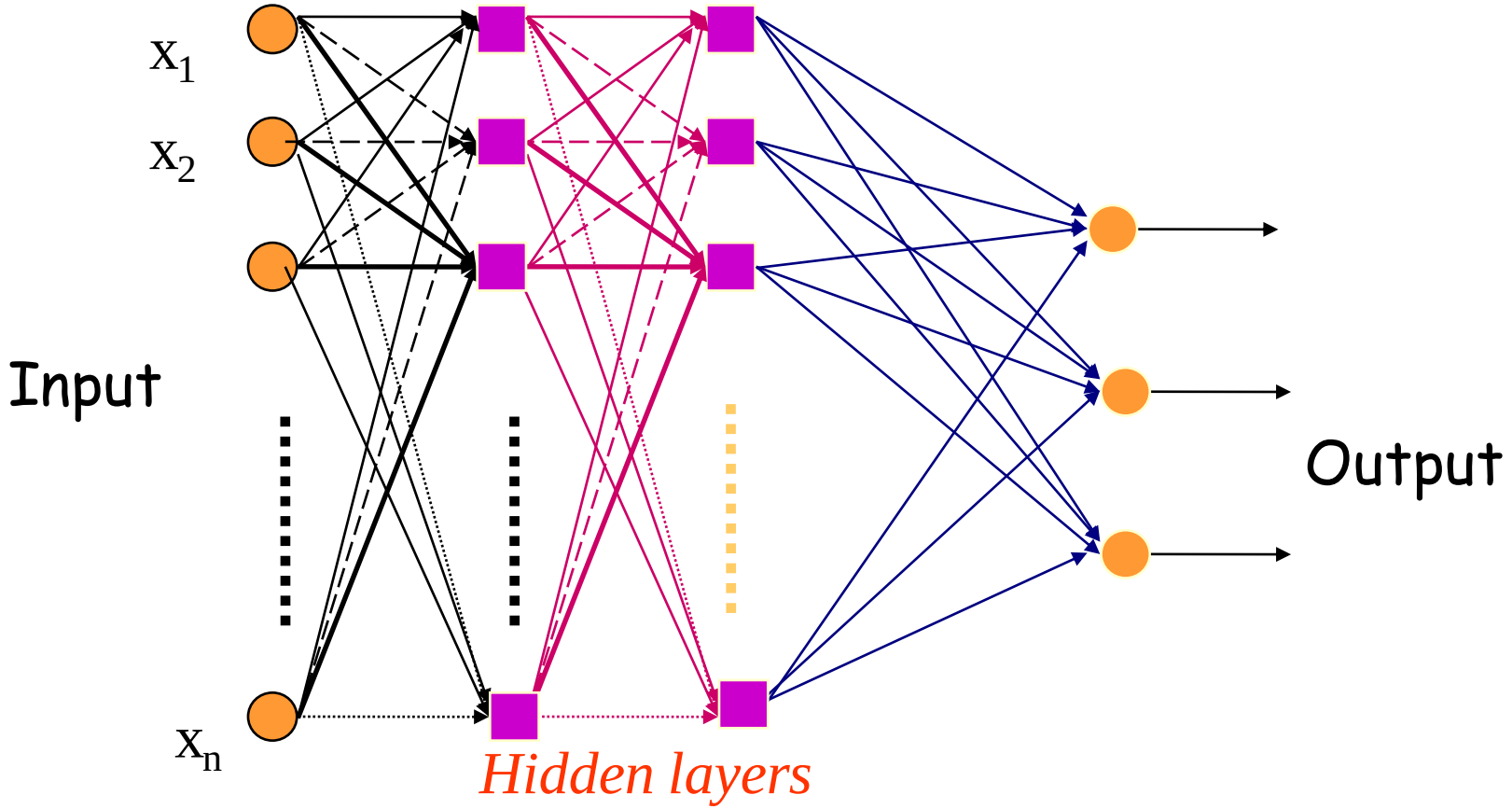


Adaline Learning

- Gradually the ball will stop at a (local or global) **minima where the gradient is zero**



Example of a four-layer neural network



3 proper neuron layers
the first (input layer) is dummy, only transmit the inputs to the next layer

Backpropagation

- Training algorithm for **multilayer neural networks** (or **MLP – Multi-Layer Perceptron**)
- **Supervised learning** algorithm based on gradient descent
- Also named **Generalized Delta Rule** Introduced by Rumelhart, Hinton & Williams (1986) Parker (1982), Werbos (1974)
- Require **differentiable "activation functions" for neurons**, such as **sigmoid** function
- BP neural networks are the **most widely used neural networks**
- **BP networks can learn any non-linear function.**

Summary of BP learning algorithm

```
Set learning rate
Set initial weight values (including biases):  W, b

Loop until stopping criteria satisfied:
   For each of the patterns in the training set
       present an input pattern  to input units
       compute output signal for hidden units
       compute output signal for output units

        present Target response to output units
       compute error signal for this pattern
  Compute an overall error for all the patterns
(e.g. mean squared err )
  Update weights at output layer
  Update weights at hidden layer
   Increment t  to t+1 (t -epoch number)
end loop
```

BP has two phases

- **Forward pass phase**
  - feed-forward propagation of input pattern signals through the network, **from inputs towards the network outputs**

- **Backward pass phase**
  - computes 'error signal' – **propagation of error** (difference between actual and desired output values) backwards through network, **starting from output units towards the input units**

Network training

- Each full presentation of all patterns = **'epoch'**
- Usually **better to randomize order of training** patterns presented for each epoch in order to **avoid correlation between consecutive training pairs being learnt** (**order effects**)
- **Training set shown** repeatedly *until stopping criteria are met*
- **Selecting initial weight values**:
  - **Choice of initial weight values** is important as this decides starting position in weight space. That is, **how far away from global minimum**

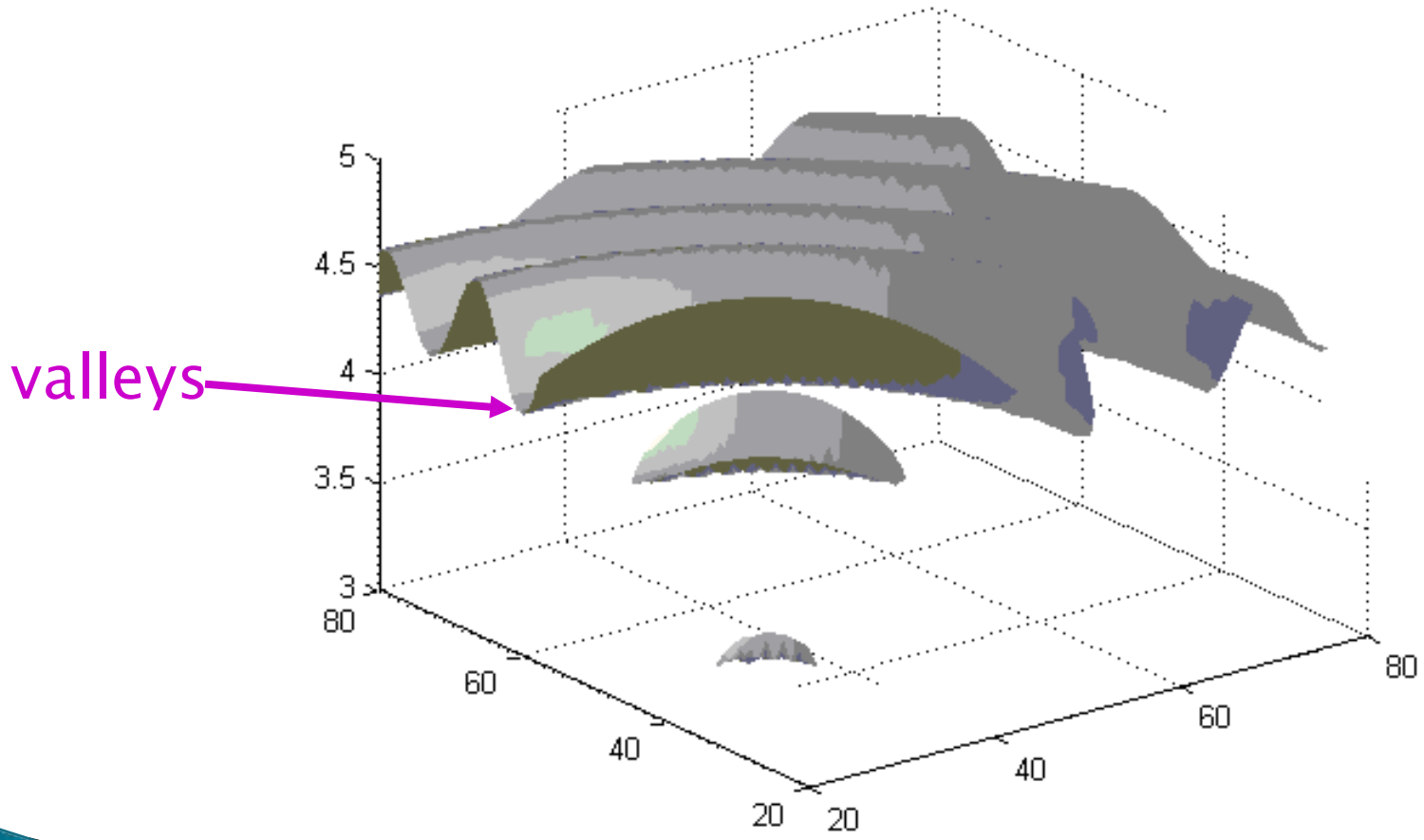
Error function of BP training

- Aim is to **minimise an error function** over all training patterns by **adapting weights** in MLP.
- **Mean squared error** is typically used:

$$E(t) = \frac{1}{2} \sum_{k=1}^{p} (d_k(t) - y_k(t))^2$$

- p : number of training patterns
- In *single layer Perceptron with linear activation functions* (**ADALINE**), the **error function is simple** and described by a **smooth parabolic surface with a single minimum**.

MLP with nonlinear activation functions have *complex error surfaces* (e.g. plateaus, long valleys etc.) with no single minimum



Picture from Jianfeng Feng, lect. notes Univ. of Sussex.

Weights updating in BP

$$w_{ij}(n+1) = w_{ij}(n) + \Delta w_{ij}(n)$$

$$\Delta w_{ij}(n) = \eta \delta_o x_{ij} \quad \text{for } \textbf{output} \text{ layer}$$

$$\Delta w_{ij}(n) = \eta \delta_h x_{ij} \quad \text{for } \textbf{hidden} \text{ layer}$$

- The values δ_o, δ_h are the **gradients at output and respectively hidden layers.** For more details on how to calculate these values please see **"Machine Learning" – Tom Mitchell (1997)**
- The **big problem of BP algorithm**:
  - difficulty to cope with **local minima and find a global minimum.**
- **Few improvements** were reported:
  - variable learning rate, momentum, weight decay, use of modified error functions etc.

Improvement: Momentum

- Method of **reducing problems of instability** while increasing the rate of convergence

- **Modified weight update equation** is:

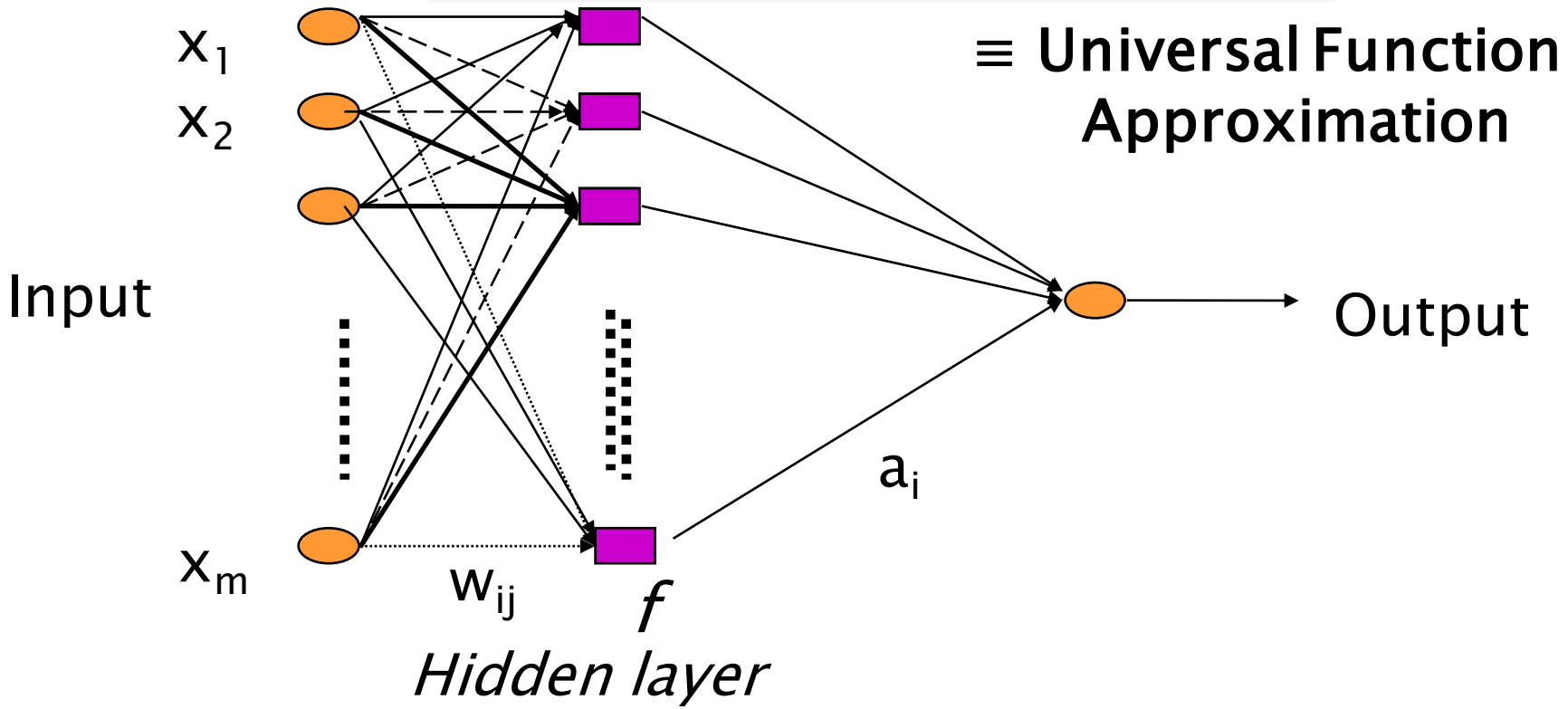
$$w_{ij}(n+1) = w_{ij}(n) + \eta\delta_j(n)x_{ij} + \alpha[w_{ij}(n) - w_{ij}(n-1)]$$

α – Momentum coefficient, 0 <= α < 1

Effect of momentum term

- If **weight changes tend to have the same sign**, *momentum term increases* (gradient decreases) – speed up convergence on shallow gradient
- If **weight changes tend have opposing signs**, *momentum term decreases* and gradient descent slows to reduce oscillations (stabilizes)
- **Can help escape** when being trapped **in local minima**
- ***Increases the convergence speed with a factor of*** $\eta/(1-\alpha)$

Three-layer network (or more)



≡ Universal Function Approximation

Universal Approximation Theorem:
For any given constant $\varepsilon > 0$ and continuous function $h(x_1,\ldots,x_m)$ *with m inputs and n outputs,* there exists a **three layer MLP** (which computes the function H) with m inputs and n outputs with the property

$$|h(x_1,\ldots,x_m) - H(x_1,\ldots,x_m)| < \varepsilon$$

How do I know if network modifications are needed?

- **Low accuracy of training** or test data *indicates that a new hidden layer or more hidden nodes are needed*.
  - if *number of hidden nodes exceeds number of inputs and outputs*, then **add another hidden layer**
  - ***decrease the total hidden nodes by 50%*** *in each successive hidden layer*
  - (e.g., if 10 nodes in first layer, then use 5 in the second layer and 2 in the third layer)
- If NN performs well on the Training set but poorly on Testing set,
  - then it is **treating each record as a special case** and has **"memorized" the data (lost generalization ability – over fitting)**. Then **use fewer hidden nodes or remove a hidden layer**

What is the best architecture for a Neural Network?

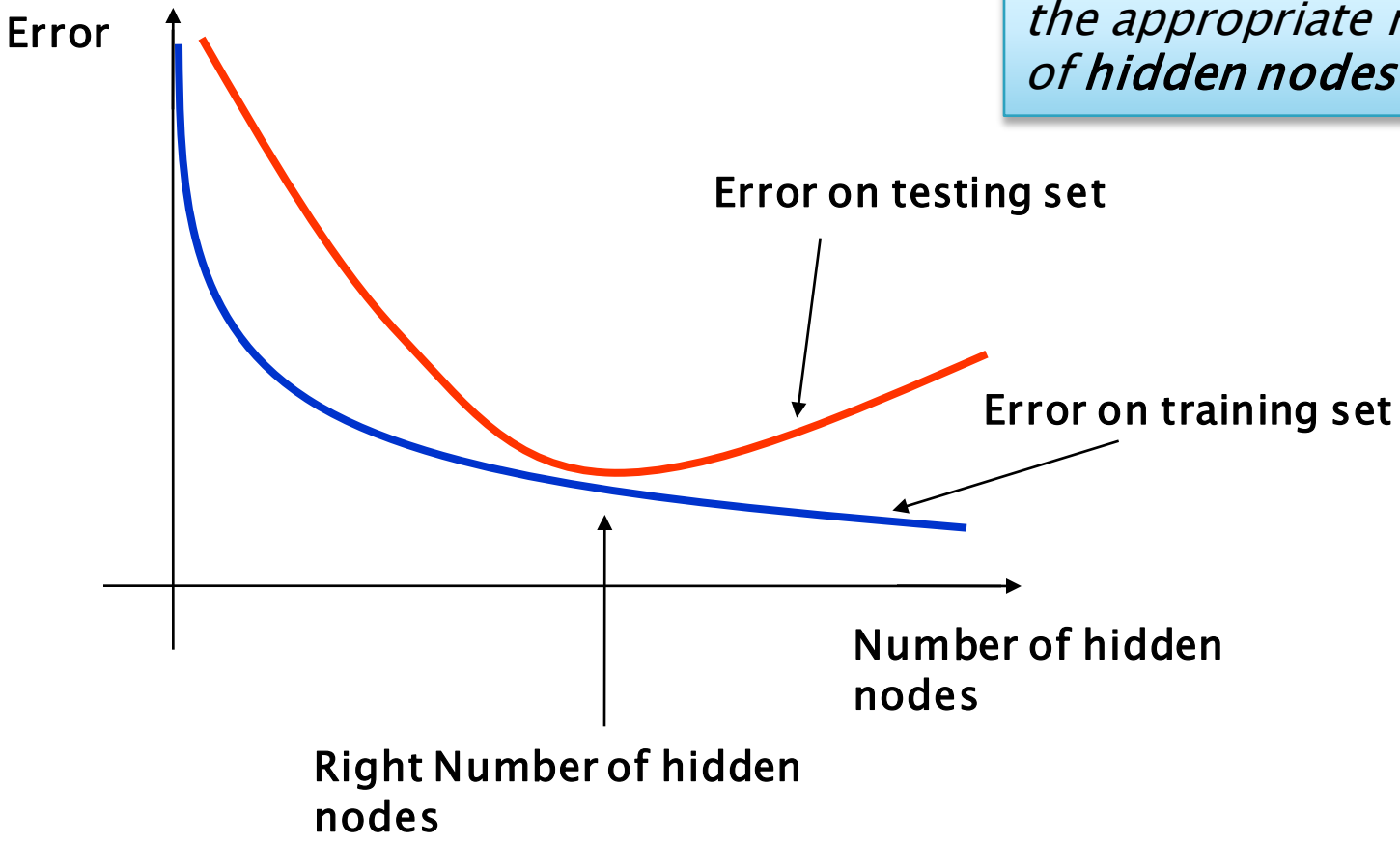
- Divide available data into 2 sets:
  - **Training data set**
    - used to **train the weights and biases of NN**
  - **Testing data set**
    - used to **test the performance** of the trained neural network

*the appropriate number of **hidden nodes***

- If the network **contains more hidden units** (learning parameters) **than necessary** to learn the training set
  - then the **network will memorize the training patterns**
  - and will **exhibit poor classification abilities for data not contained in the training set** (testing set).
  - That means **the network lost generalization ability** – **over fitting.**

What is the best architecture for a Neural Network?

*the appropriate number of **hidden nodes***



Mayor clases of Neural Networks

- Backpropagation Neural Networks
  - supervised learning
- Kohonen Self Organizing Maps
  - unsupervised learning
- Hopfield Neural Networks
  - recurrent neural networks
- Radial Basis Function Neural Networks (RBF)
- Neuro-Fuzzy Networks (NF)
- Others: various architectures of recurrent neural networks,
  - networks with dynamic neurons,
  - networks with competitive learning, etc.

History of NN

- **McCulloch & Pitts (1943)**
  - neural networks and artificial intelligence were born, first well-known model for a biological neuron
- **Hebb(1949)**
  - Hebb learning rule
- **Minsky(1954)**
  - Neural Networks (PhD Thesis)
- **Rosenblatt(1957)**
  - Perceptron networks (Perceptron learning rule)
- **Widrow and Hoff(1959)**
  - Delta rule for ADALINE networks
- **Minsky & Papert(1969)**
  - Criticism on Perceptron networks (problem of linear separability)
- **Kohonen(1982)**
  - Self-Organizing Maps

History of NN (II)

- **Hopfield(1982)**
  - Hopfield Networks
- **Rumelhart, Hinton & Williams (1986)**
  - Back-Propagation algorithm
- **Broomhead & Lowe (1988)**
  - Radial Basis Functions networks (RBF)
- **Vapnik (1990)**
  - Support Vector Machine approach
- **In the '90s**
  - massive interest in neural networks, many NN applications were developed
  - Neuro-Fuzzy networks emerged

In conclusion, Neural Networks:

- **Can learn directly from data.**
  - They exhibit **good learning ability** – better than other AI approaches
- Can learn **from noisy or corrupted data**
- **Parallel** information **processing**
- **Computationally fast once trained**
- **Robustness to partial failure** of the network
- **Useful where data are available** and difficult to acquire symbolic knowledge
- Drawback of NN
  - – knowledge captured by a NN through learning (in weights –real numbers) is not in a **familiar form for human beings**, e.g. if–then rules (**NNs are black box structures**).
- **Over fitting issues.**

Self-Organizing Feature Map (SOFMs)

A competitive learning algorithm for pattern discovery

Competitive learning

- **Neurons compete among themselves to be activated.**
- While in "**Hebbian learning**", several output neurons can be **activated simultaneously**, in **competitive learning, only a single output neuron is active at any time.**
- The **output neuron** that ***wins*** the "competition" is called the ***winner-takes-all*** **neuron.**

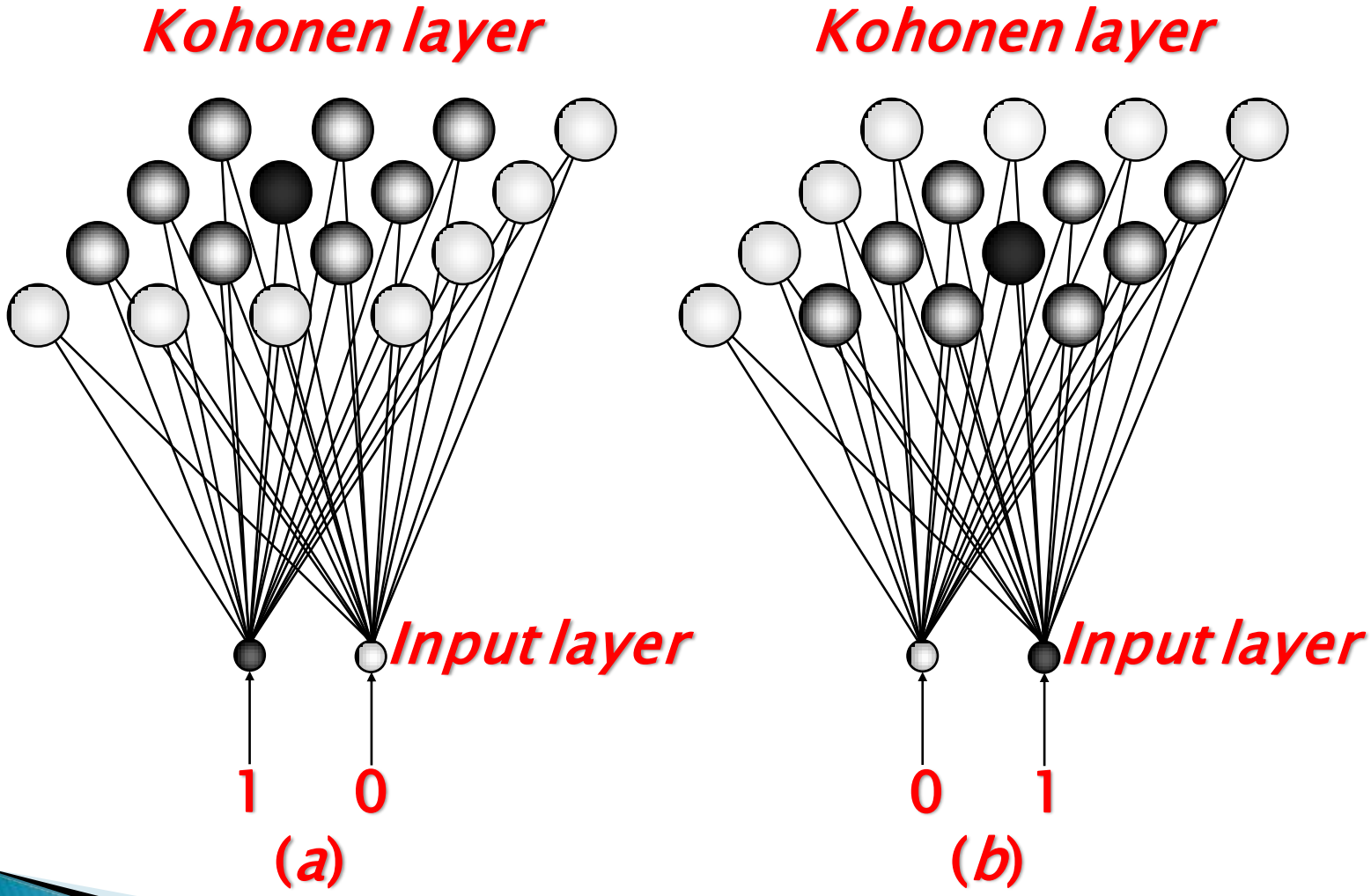
Competitive Learning (2)

- The basic idea of competitive learning was introduced in the early 1970s.
- In the late 1980s, Teuvo Kohonen introduced a special class of artificial neural networks called **Self-Organising feature Maps**.
- These **maps are based on competitive learning.**

What is a Self-Organising feature Map?

- **Our brain is dominated by the cerebral cortex**
  - a very complex structure of billions of neurons and hundreds of billions of synapses.
  - The cortex includes areas that are responsible for different human activities (motor, visual, auditory, somatosensory, etc.), and associated with different sensory inputs.
  - We can say that **each sensory input is mapped into a corresponding area of the cerebral cortex.**
  - ***The cortex is a self-organising computational map in the human brain.***

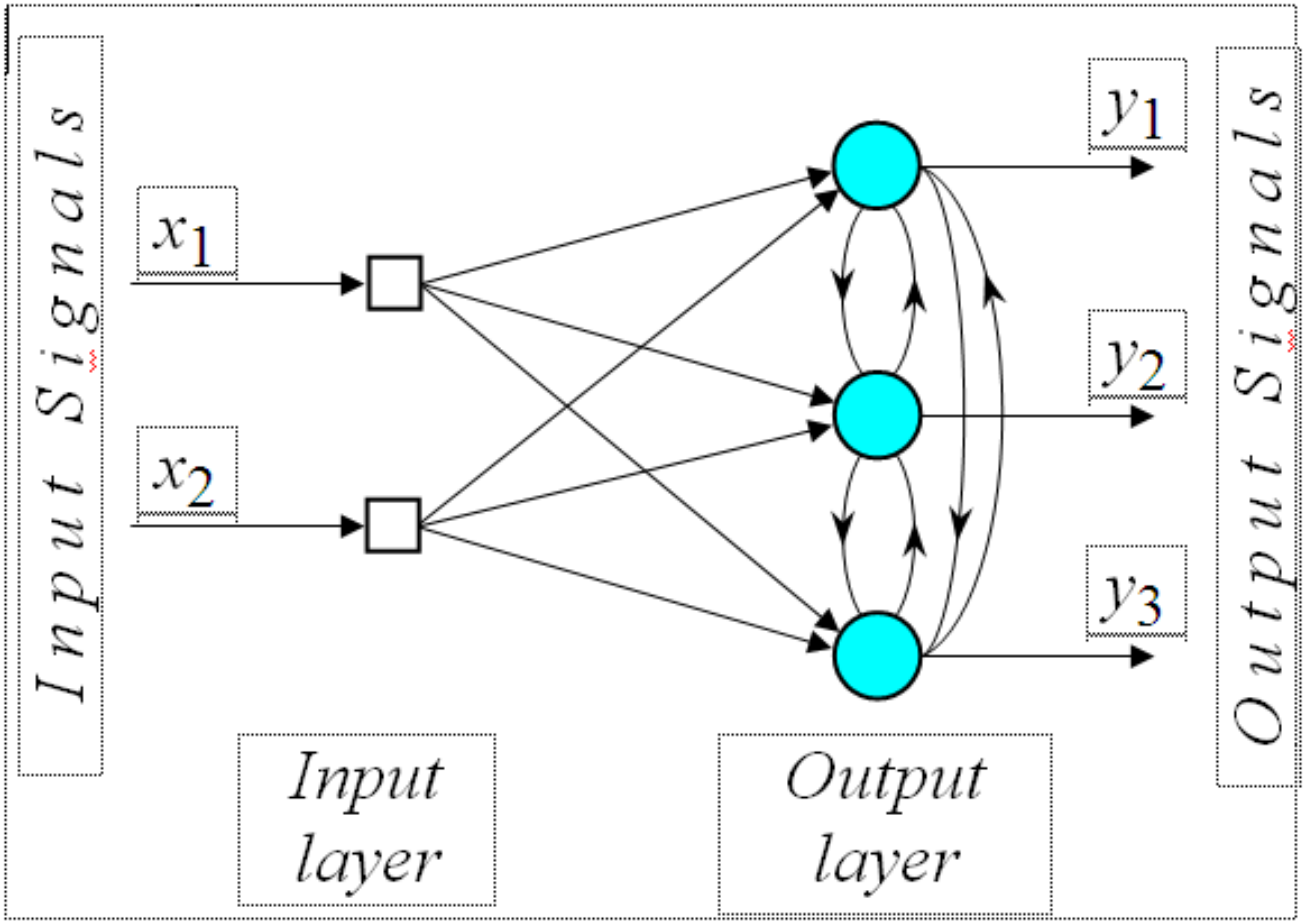
Feature-mapping Kohonen mode



The Kohonen network

- The Kohonen model provides a topological mapping. It places a fixed number of input patterns from the input layer into a higher-dimensional output or Kohonen layer.
- Training in the Kohonen network begins with the winner's neighbourhood of a fairly large size. Then, as training proceeds, the neighbourhood size gradually decreases.

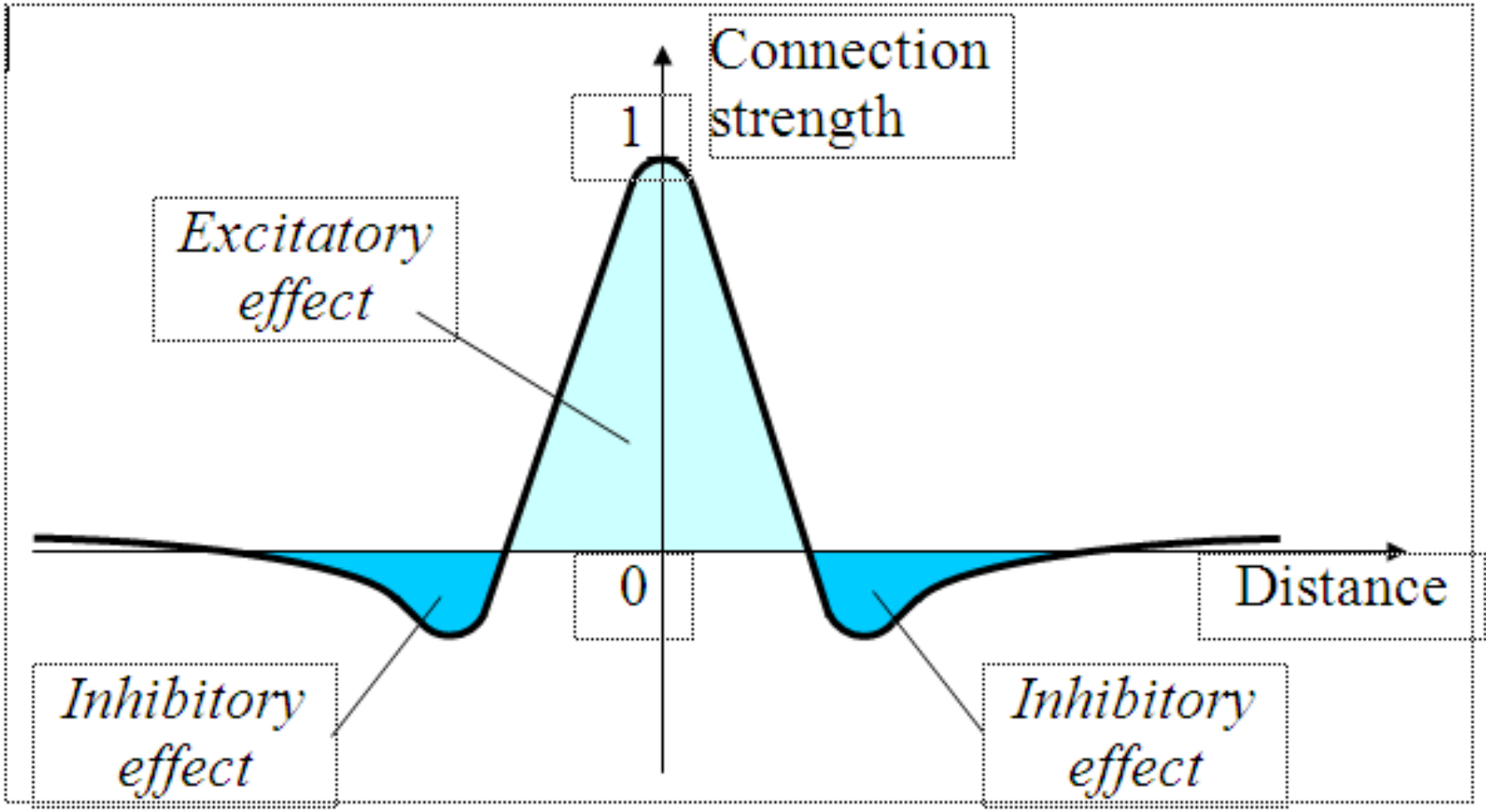
Architecture of the Kohonen Network



Architecture of the Kohonen Network

- The lateral connections are used to create a competition between neurons. The neuron with the largest activation level among all neurons in the output layer becomes the winner. This neuron is the only neuron that produces an output signal. The activity of all other neurons is suppressed in the competition.
- The lateral feedback connections produce excitatory or inhibitory effects, depending on the distance from the winning neuron. This is achieved by the use of a Mexican hat function which describes synaptic weights between neurons in the Kohonen layer.

The Mexican hat function of lateral connection



The Mexican hat function of lateral connection

- In the Kohonen network, a neuron learns by shifting its weights from inactive connections to active ones. Only the winning neuron and its neighbourhood are allowed to learn. If a neuron does not respond to a given input pattern, then learning cannot occur in that particular neuron.
- The competitive learning rule defines the change Δw_{ij} applied to synaptic weight w_{ij} as

$$\Delta w_{ij} = \begin{cases} \alpha(x_i - w_{ij}) & \text{If neuron j wins the competition} \\ 0 & \text{If neuron j loses the competition} \end{cases}$$

where x_i is the input signal and α is the *learning rate* parameter.

- The overall effect of the competitive learning rule resides in **moving the synaptic weight vector Wj** of the winning neuron j towards the input pattern X. The matching criterion is equivalent to the **minimum Euclidean distance between vector**s.
- The **Euclidean distance** between a pair of n-by-1 vectors X and Wj is defined by

$$d = \|X - W_j\| = \left[\sum_{i=1}^{n}(x_i - w_{ij})^2\right]^{1/2}$$

where xi and wij are the ith elements of the vectors X and Wj, respectively.

- To identify the winning neuron, jX, that best matches the input vector X, we should apply the following condition:

$$jx = \min_j \|X - W_j\|, j = 1, 2, \ldots$$

where m is the number of neurons in the Kohonen layer.

- Suppose, for instance, that the 2-dimensional input vector X is presented to the **three-neuron Kohonen network**,

$$jx = \begin{bmatrix} 0.52 \\ 0.12 \end{bmatrix}$$

- The initial weight vectors, Wj, are given by

$$W_1 = \begin{bmatrix} 0.27 \\ 0.81 \end{bmatrix} W_2 = \begin{bmatrix} 0.42 \\ 0.70 \end{bmatrix} W_3 = \begin{bmatrix} 0.43 \\ 0.21 \end{bmatrix}$$

- We find the winning (best-matching) neuron j_X using the minimum-distance Euclidean criterion:

$$
\begin{aligned}
d_1 &= \sqrt{(x_1 - w_{11})^2 + (x_2 - w_{21})^2} \\
d_1 &= \sqrt{(0.52 - 0.27)^2 + (0.12 - 0.81)^2} = 0.73
\end{aligned}
$$

$$
\begin{aligned}
d_2 &= \sqrt{(x_1 - w_{12})^2 + (x_2 - w_{22})^2} \\
d_2 &= \sqrt{(0.52 - 0.42)^2 + (0.12 - 0.70)^2} = 0.59
\end{aligned}
$$

$$
\begin{aligned}
d_3 &= \sqrt{(x_1 - w_{13})^2 + (x_2 - w_{23})^2} \\
d_3 &= \sqrt{(0.52 - 0.43)^2 + (0.12 - 0.21)^2} = 0.13
\end{aligned}
$$

- Neuron 3 is the winner and its weight vector W_3 is updated according to the competitive learning rule.

$$
\begin{aligned}
\Delta w_{13} &= \alpha(x_1 - w_{13}) = 0.1(0.52 - 0.43) = +0.01 \\
\Delta w_{23} &= \alpha(x_2 - w_{23}) = 0.1(0.12 - 0.21) = -0.01
\end{aligned}
$$

- The updated weight vector W_3 at iteration (p + 1) is determined as:

$$W_3(p+1) = W_3(p) + \Delta W_3(p) = \begin{bmatrix} 0.43 \\ 0.21 \end{bmatrix} + \begin{bmatrix} +0.01 \\ -0.01 \end{bmatrix} = \begin{bmatrix} 0.44 \\ 0.20 \end{bmatrix}$$

- The weight vector W_3 of the wining neuron 3 becomes closer to the input vector X with each iteration.

Competitive Learning Algorithm – *Step 1: Initialisation*

- Set **initial synaptic weights to small random values**, say in an interval [0, 1], and assign a **small positive value to the learning rate** parameter α.

Competitive Learning Algorithm – *Step 2: Activation and Similarity Matching*

- **Activate the Kohonen network** by applying the input vector X, and find the **winner-takes-all** (best matching) neuron jX at iteration p, using **the minimum-distance Euclidean criterion**

$$\|X - W_j(p)\| = \{\sum_{i=1}^{n} [x_i - w_{ij}(p)]^2\}^{1/2}$$

where n is the number of neurons in the input layer, and m is the number of neurons in the Kohonen layer.

Competitive Learning Algorithm – *Step 3: Learning*

Update the synaptic weights

$$w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)$$

where $\Delta w_{ij}(p)$ is the weight correction at iteration p.

The weight correction is determined by the competitive learning rule:

$$\Delta w_{ij}(p) = \begin{cases} \alpha[x_i - w_{ij}(p)], & j \in \Lambda_j(p) \\ 0, & j \notin \Lambda_j(p) \end{cases}$$

where α is the *learning rate* parameter, and $\Lambda_j(p)$ is the neighbourhood function centred around the winner-takes-all neuron j_X at iteration p.

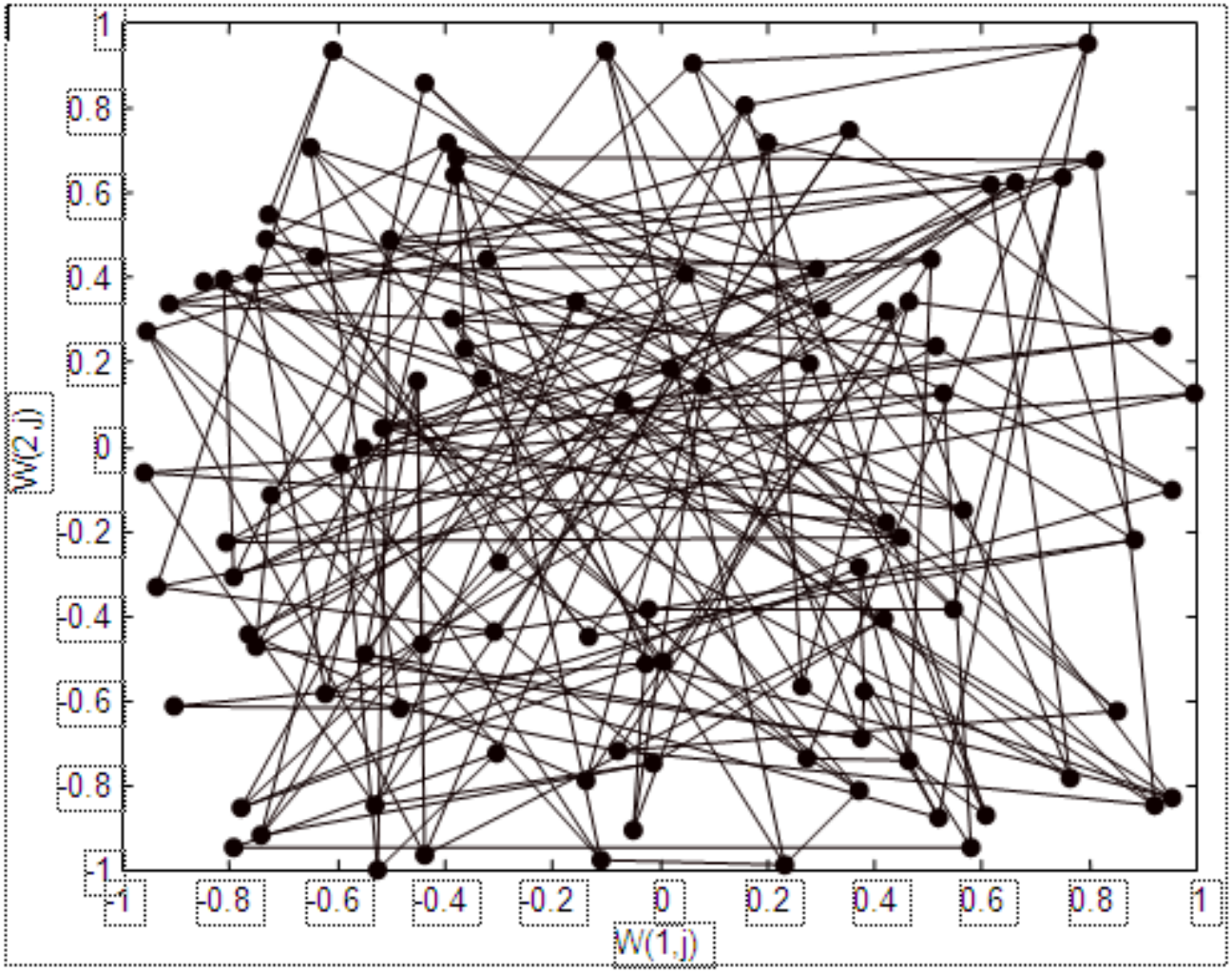
Competitive Learning Algorithm – *Step 4: Iteration*

- Increase iteration p by **one**
- Go back to Step 2 and continue until the **minimum-distance Euclidean criterion** is satisfied, or **no noticeable changes occur in the feature map**.

Competitive learning in the Kohonen network

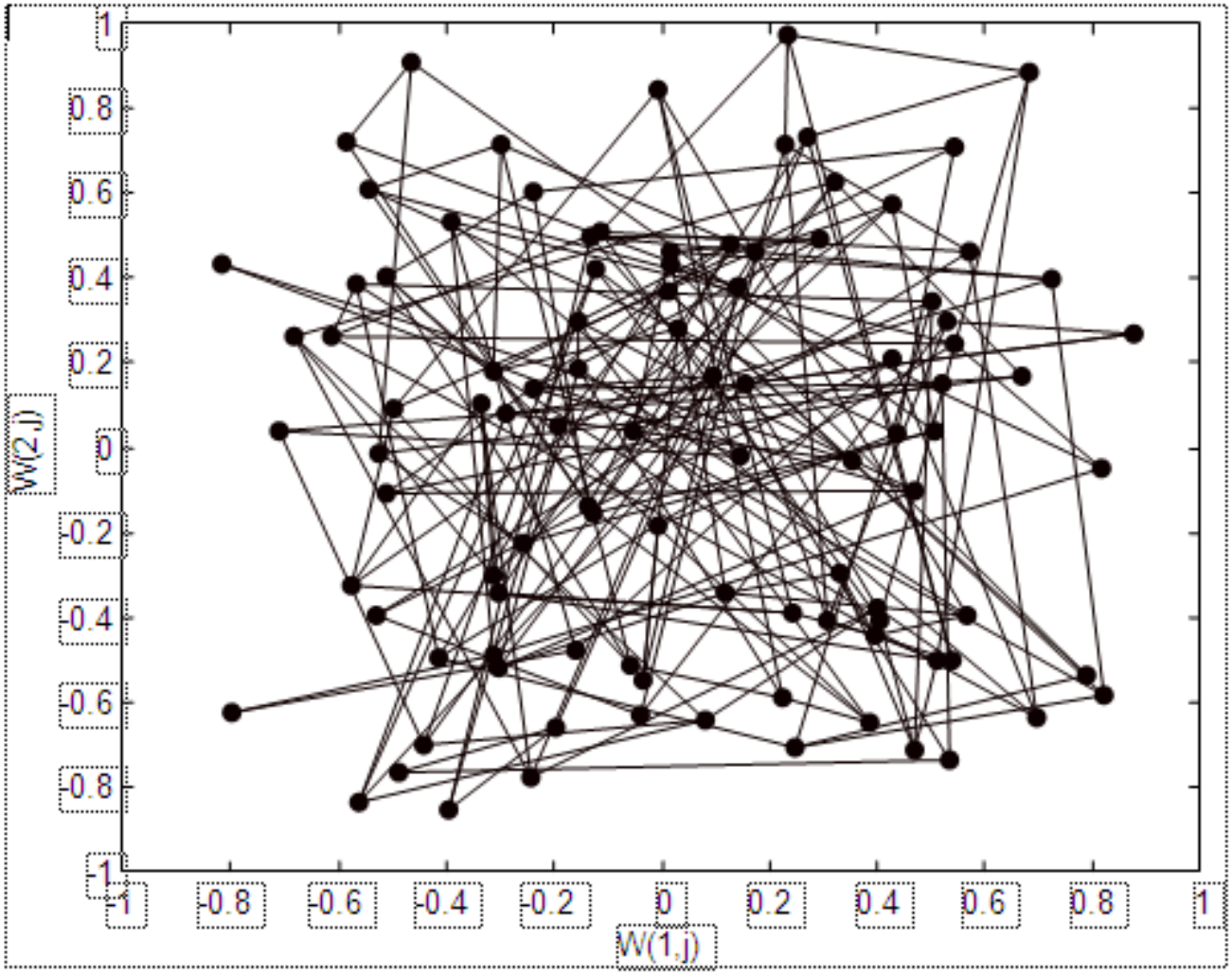
- To illustrate competitive learning, consider the Kohonen network with **100 neurons arranged in the form of a two-dimensional lattice** with 10 rows and 10 columns.
- The network is required to classify **two-dimensional input vectors** each neuron in the network should respond **only to the input vectors occurring in its region.**
- The network is trained with **1000 two-dimensional input vectors** generated **randomly in a square region** in the interval between –1 and +1. The learning rate parameter α is equal to 0.1.

The evolution:
Initial random weights



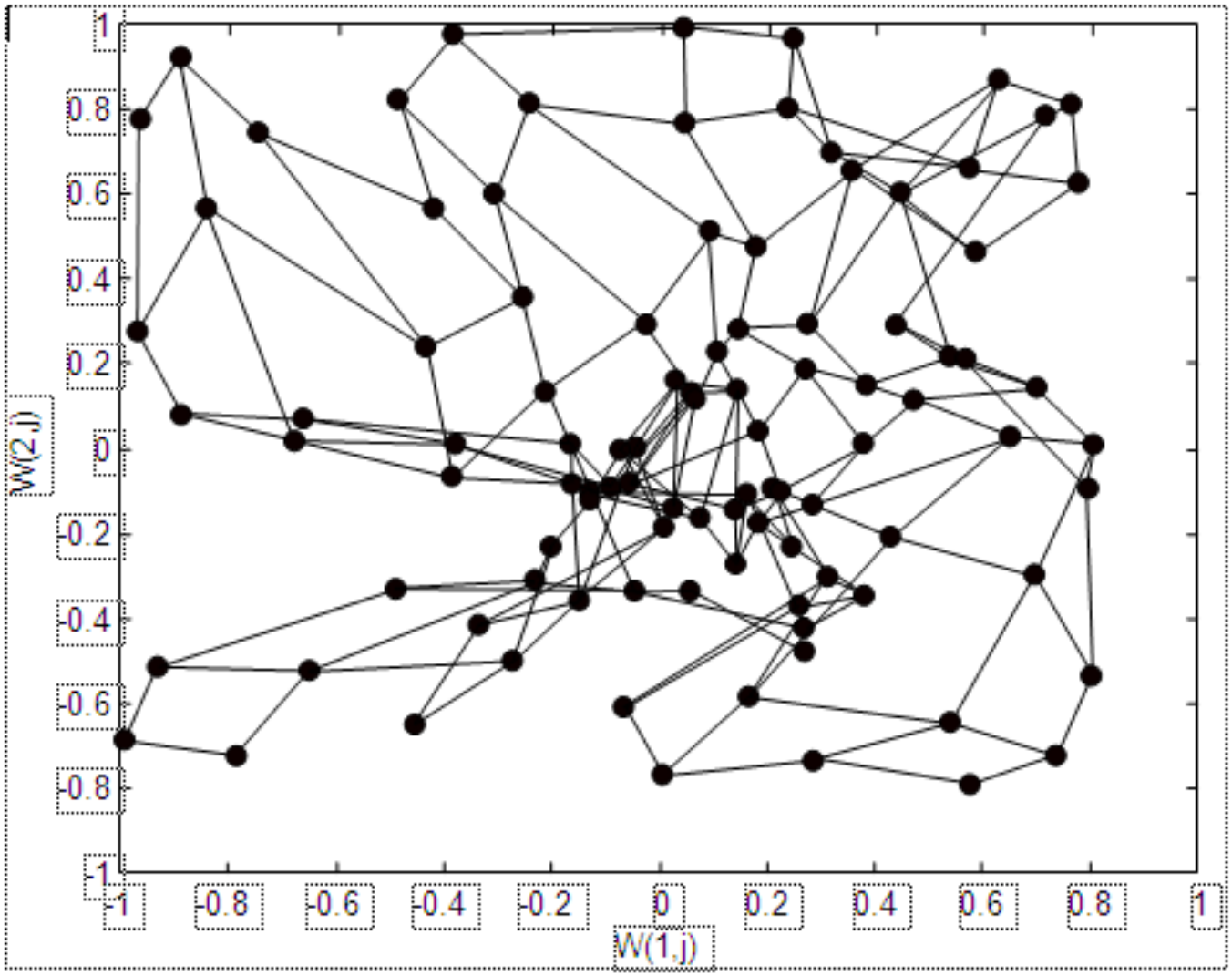
The Evolution:

Network after 100 iterations



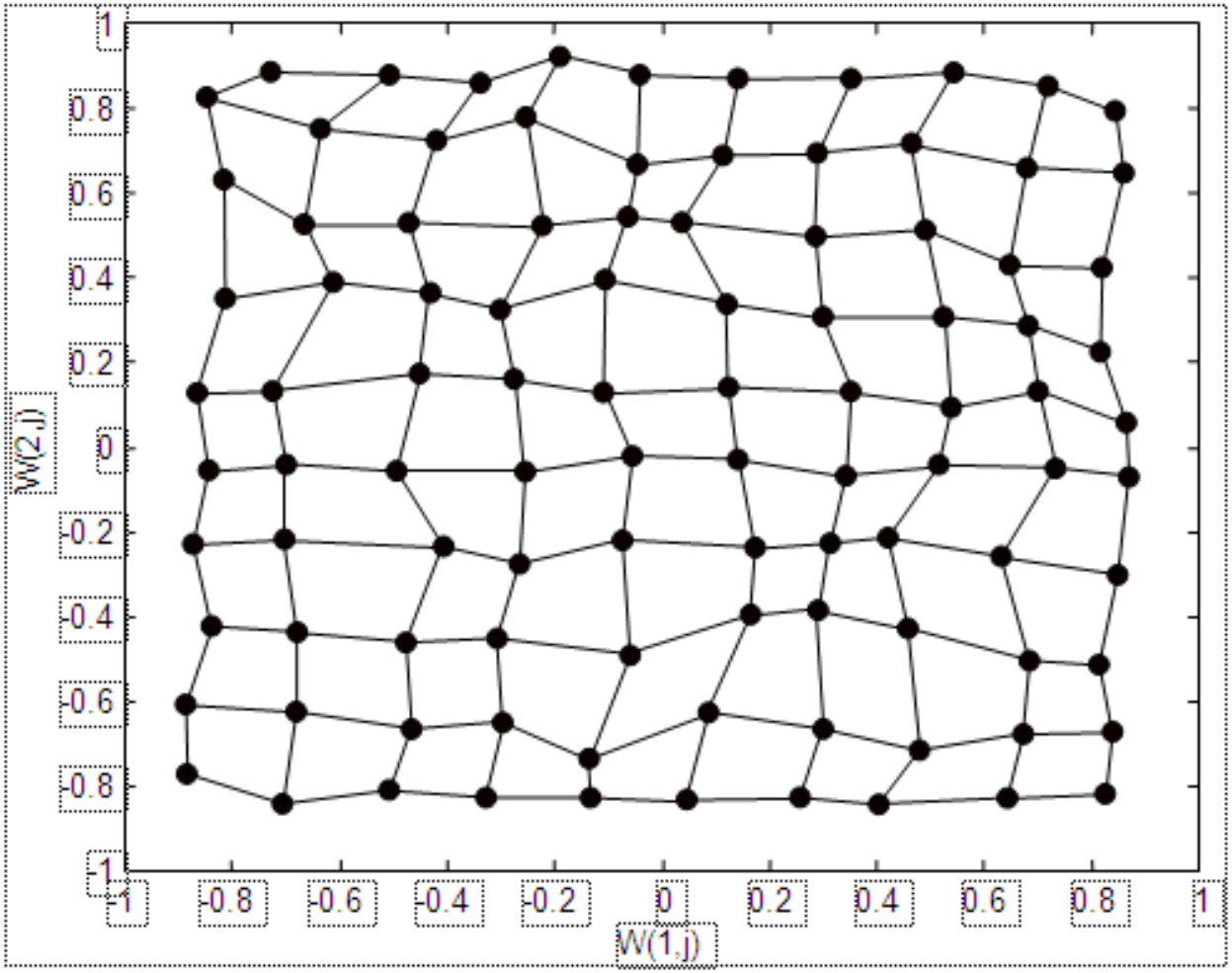
The Evolution:

Network after 1000 iterations

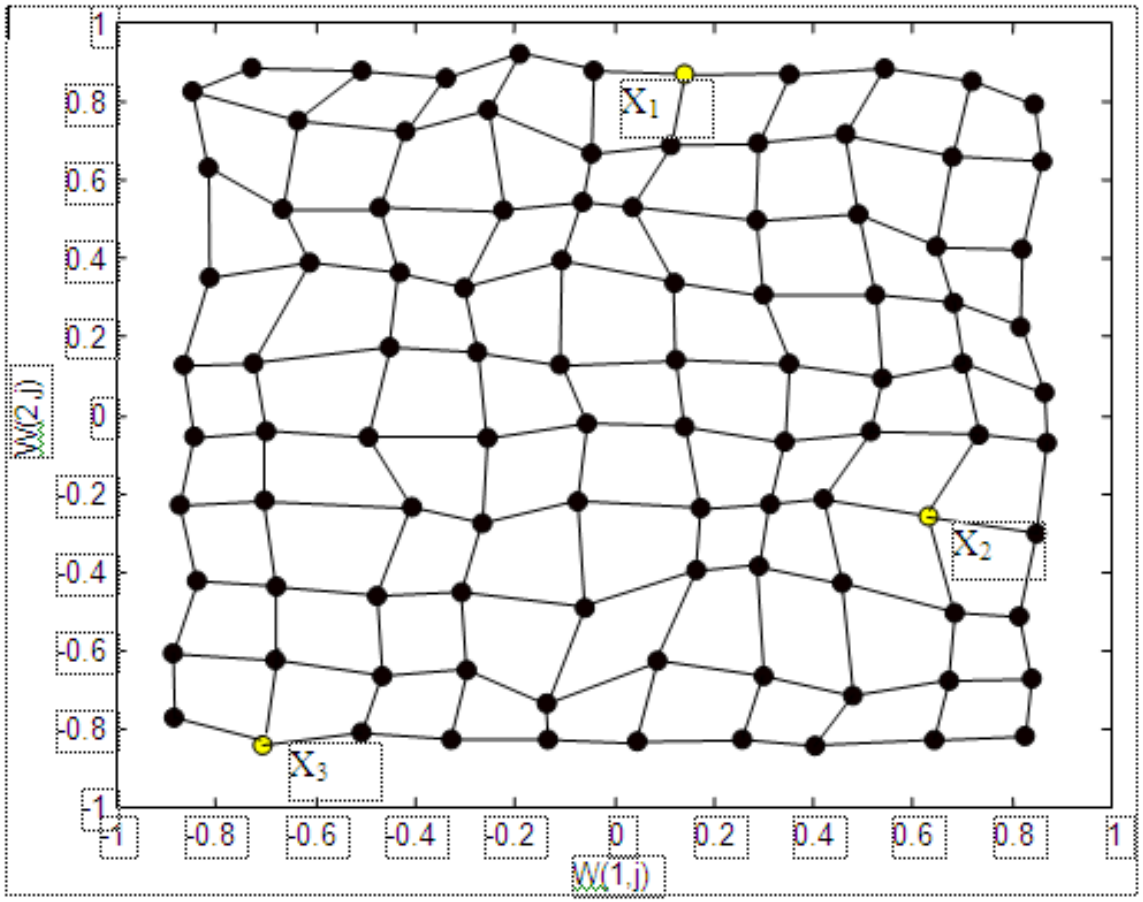


The evolution:

Network after 10,000 iterations



Response of the network for the following input vectors:

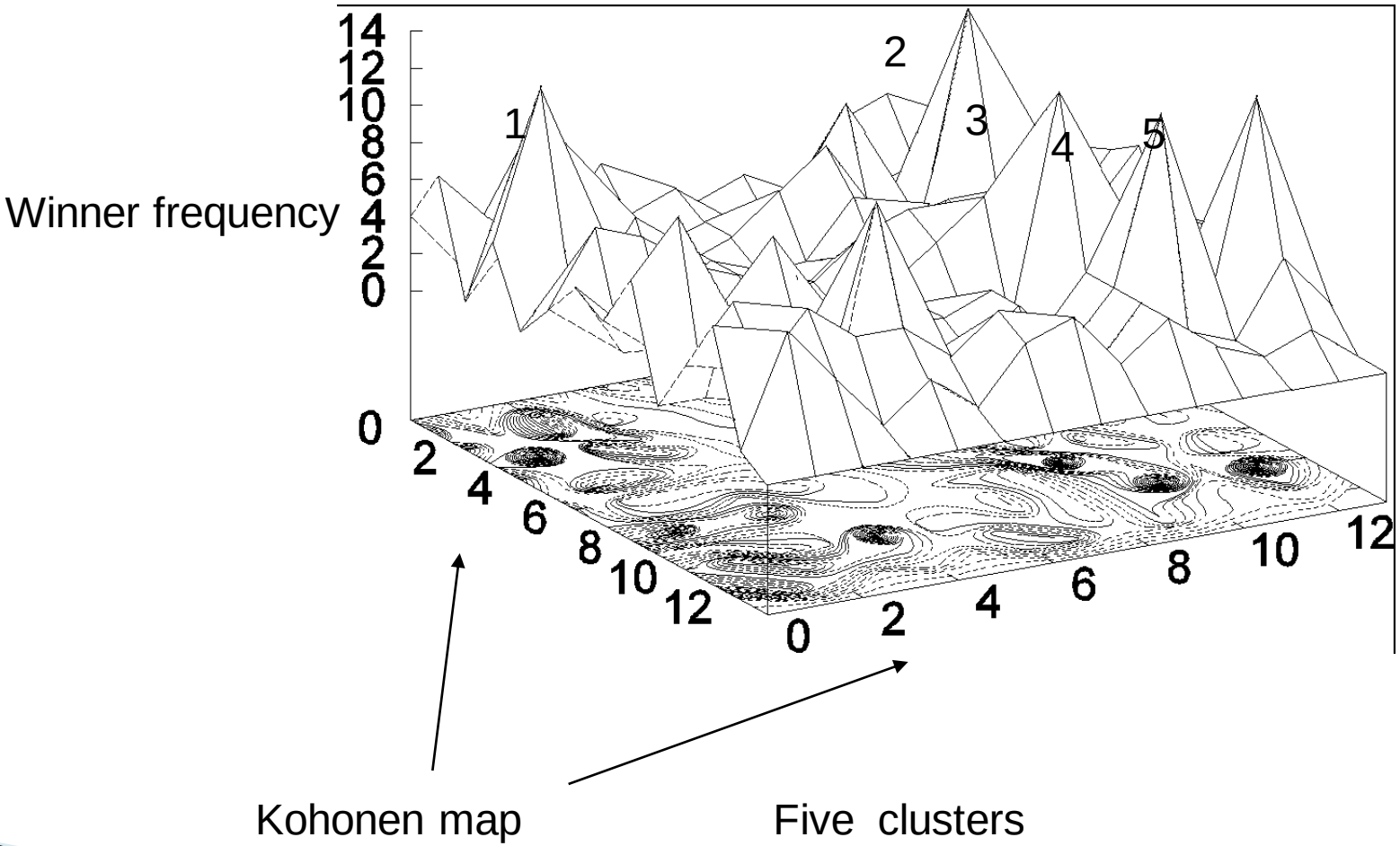


$$\mathbf{X}_1 = \begin{bmatrix} 0.2 \\ 0.9 \end{bmatrix}$$

$$\mathbf{X}_2 = \begin{bmatrix} 0.6 \\ -0.2 \end{bmatrix}$$

$$\mathbf{w}_3 = \begin{bmatrix} -0.7 \\ -0.8 \end{bmatrix}$$

Finally: Cluster extraction



Summary on Neural Networks – What did we learn?

- Different **neural network architectures** and learning algorithms:
- **Supervised learning**
  - Perceptron (Perceptron learning rule)
  - Adaline (Delta rule)
  - Feed forward Multi-Layer Perceptron (Back propagation).
- **Unsupervised learning**
  - Competitive learning
  - Kohonen Self-Organizing Maps

K-means

Classical clustering methods

- Partitioning methods
  - Construct various partitions and then evaluate them by some criterion
  - *k-Means (and EM), k-Medoids*
- Hierarchical methods
  - Create a hierarchical decomposition of the set of objects using some criterion
  - *agglomerative, divisive*
- Model-based clustering methods
  - A model is hypothesized for each of the clusters and the idea is to find the best fit of that model to each other

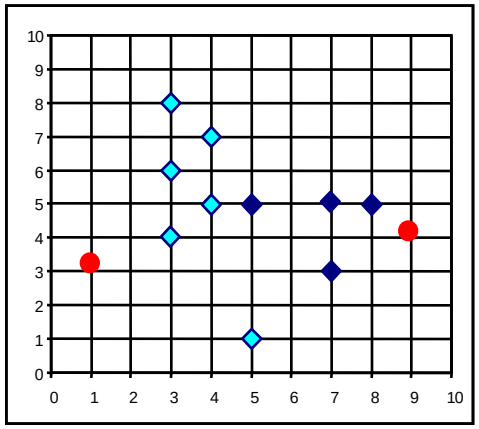
Non-Classical clustering methods

- **Fuzzy clustering algorithms**
  - Fuzzy C-means is the most popular one
- **Neural networks** have been used for clustering
  - Self-Organizing Maps (SOMs - Kohonen, 1984)
  - Adaptive Resonance Theory (ART) networks (Carpenter & Grossberg, 1990)
- **Evolutionary algorithms** based clustering
- **Simulated annealing** based clustering

K-means Algorithm

- First we should specify k
  - the **number of clusters we want to find out**
  - Each cluster will be represented by the **center of the cluster.** Iteratively minimize the objective function (distance to clusters).
- Algorithm:
  1. **Randomly pick k points** (inside the hypervolume containing the pattern set) as the **"centroids"** of the k clusters we want
  2. For each pattern in the data set, **assign the pattern to the cluster** with the **closest centroid**
  3. **Recompute the cluster centroids** using the ***current cluster memberships***
  4. If there is **no (or minimal) change** in the identified clusters between two consecutive iterations **stop, otherwise go to step 2**

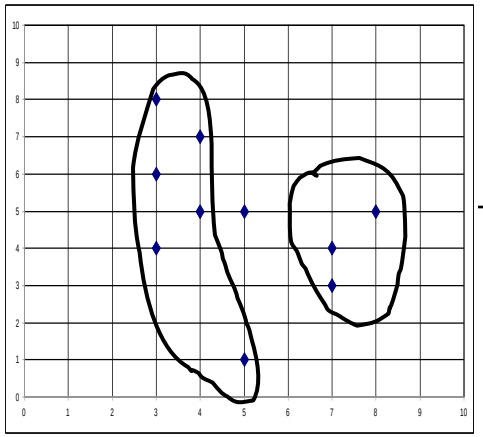
Example of *K-Means* Clustering



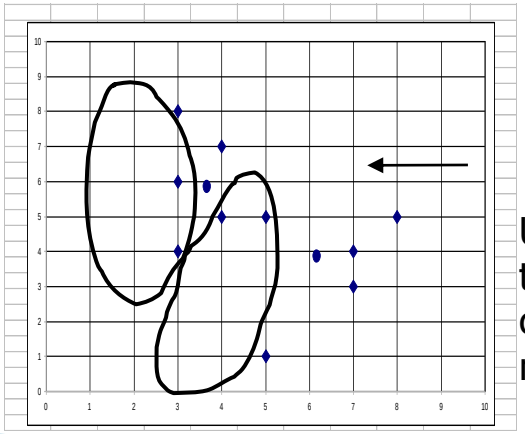
K=2

Arbitrarily choose K object as initial cluster center

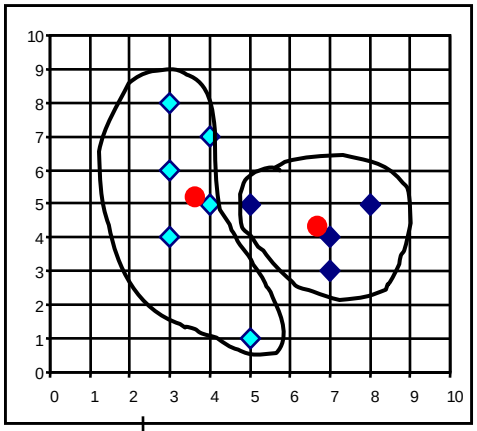
Assign each objects to most similar center



reassign

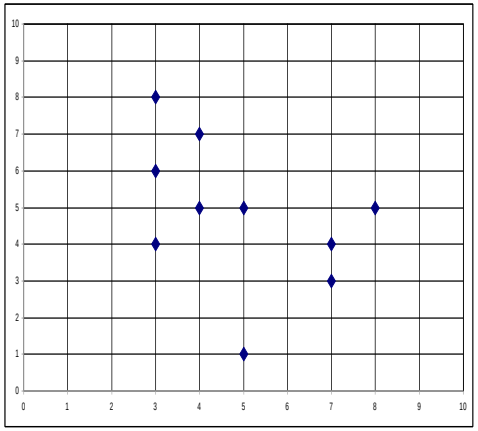


Update the cluster means



Update the cluster means

reassign



K-means Example

- **Objects**: 1, 2, 5, 6,7 (1-dimensional objects)
- We want to find **2 clusters** (k=2). Numerical difference is used as **distance**.
- K-means:
  - Randomly select 5 and 6 as centroids;
  - => Two clusters {1,2,5} and {6,7}; meanC1=8/3, meanC2=6.5
  - => {1,2}, {5,6,7}; meanC1=1.5, meanC2=6
  - => no change.
  - *Aggregate dissimilarity*
    - sum of squared distances between each point (in all clusters) and *its* cluster center--(intra-cluster distance)

$$= 0.5^2 + 0.5^2 + 1^2 + 0^2 + 1^2 = 2.5$$

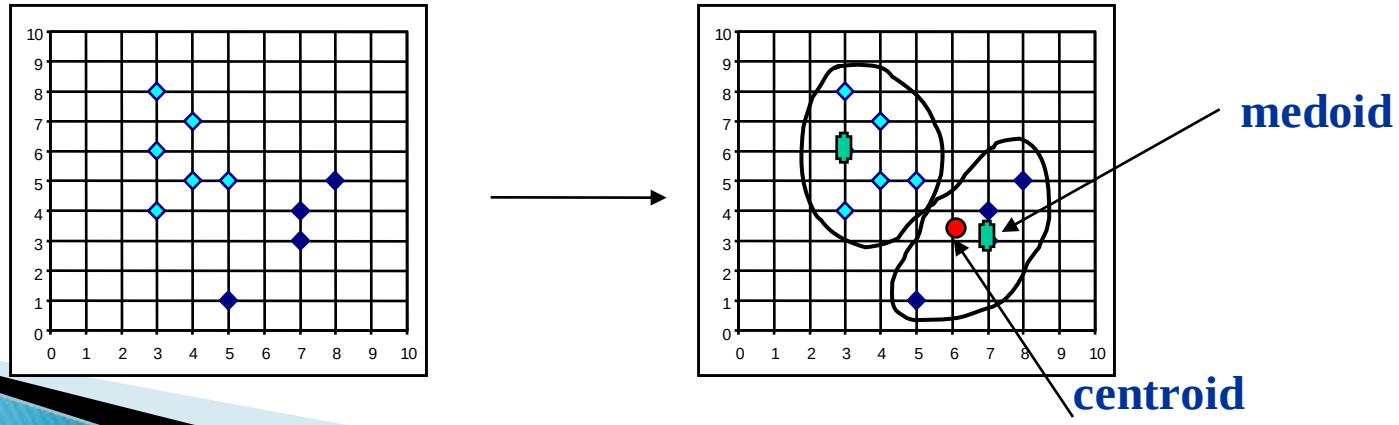
$|1\text{-}1.5|^2$

Problems of K-means

- **We need to specify k in advance**!
  - Solution: May need to try out several k
- Tends to go to local minima that depend on the selection of starting centroids
  - Solution: Run the algorithm with different starting points
- **Assumes clusters are spherical** in vector space (Euclidean topology), could be foils, donuts and others shapes!!
- Sensitive to coordinate changes, weighting, etc.
- K-means is sensitive to "outliers" (does not recognize them)
- an object with an extremely large value (outlier) may **substantially distort** the distribution of the data

Problems of K-means

- Outlier problem can be handled by **K-medoid or neighborhood**-based algorithms
- **K-Medoids method:**
  - Use the **medoids** (the **most centrally located** object in a cluster) instead of computing the **mean** value of the objects in a cluster (centroids) as a reference point for that cluster. See for example PAM (Partitioning Around Medoids) algorithm – (Kaufman & Rousseeuw, 1987 Wiley).



Variants of K-means

- **Recompute the centroids after every change** (or few changes), rather than after all the patterns are re-assigned
  - *Improves the convergence speed*
- Starting centroids (seeds) may determine to converge to local minima, as well as the rate of convergence
  - Use **heuristics** to pick good seeds
  - Run **K-means M times** and **pick the best clustering obtained**