Artificial Intelligence AI 2002 Lecture 17

Ms. Mahzaib Younas

Lecturer Department of Computer Science

FAST NUCES CFD

Unsupervised Learning

Unsupervised Learning

- In **unsupervised learning**, the agent learns patterns in the input even though **no explicit feedback** is supplied.
- **Unsupervised learning** occurs when no classifications are given and the *learner must discover categories and regularities in the data*.
- The most general example of unsupervised learning task is **clustering**:
  - potentially useful clusters developed from the input examples.
- For example, **a taxi agent** might gradually develop a concept of “good traffic days” and “bad traffic days”.

Clustering

K-means Clustering

- K-means is a partitioning clustering algorithm
- Let the set of data points (or instances) *D* be

$$\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\},$$

  where
  - $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ir})$ is a vector in a real-valued space $X \subseteq R^r$, and
  - r is the number of attributes (dimensions) in the data.

- The *k*-means algorithm partitions the given data into *k* clusters.
  - Each cluster has a cluster **center**, called **centroid**.
  - *k* is specified by the user

K-means Clustering

- Basic Algorithm:

```
1: Select $K$ points as the initial centroids.
2: **repeat**
3:     Form $K$ clusters by assigning all points to the closest centroid.
4:     Recompute the centroid of each cluster.
5: **until** The centroids don't change
```

Stopping/Convergence Criterion

1. No (or minimum) re-assignments of data points to different clusters,
2. No (or minimum) change of centroids, or
3. Minimum decrease in the **sum of squared error** (SSE),

$$SSE = \sum_{j=1}^{k} \sum_{\mathbf{x} \in C_j} dist(\mathbf{x}, \mathbf{m}_j)^2$$

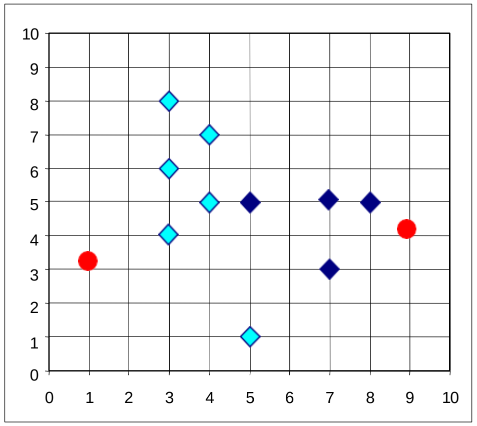
- C_j is the jth cluster, $\mathbf{m}_j$ is the centroid of cluster C_j (the mean vector of all the data points in C_j), and $dist(\mathbf{x}, \mathbf{m}_j)$ is the distance between data point $\mathbf{x}$ and centroid $\mathbf{m}_j$.

K-means Clustering--- Details

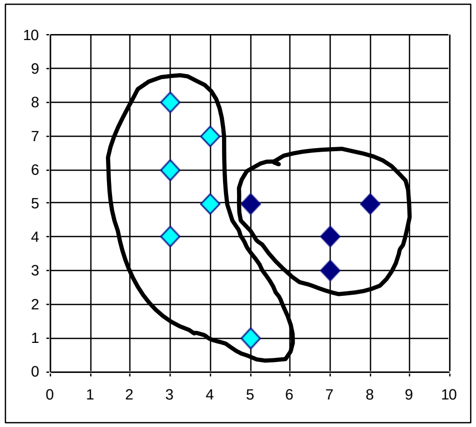
- *Initial centroids are often chosen randomly*.
  - Clusters produced vary from one run to another.
- The **centroid** is (typically) the mean of the points in the cluster.
- **‘Closeness’** is measured by Euclidean distance, cosine similarity, correlation, etc.
- K-means will converge for common similarity measures mentioned above.

- Most of the convergence happens in the first few iterations.
  - Often the stopping condition is changed to ‘Until relatively few points change clusters’

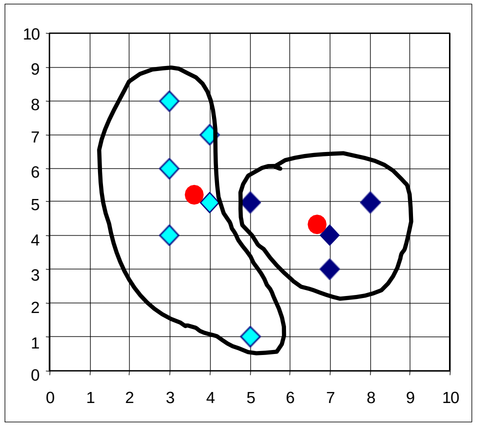
K-means Clustering Example



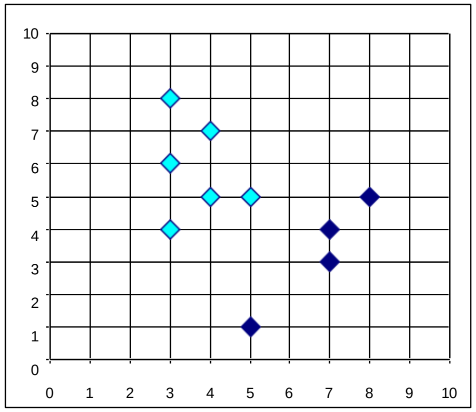
Assign each objects to most similar center



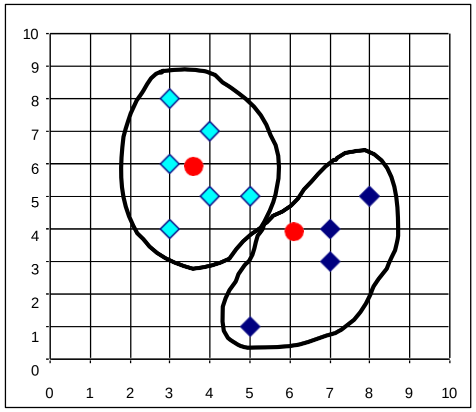
Update the cluster means



reassign



Update the cluster means



reassign

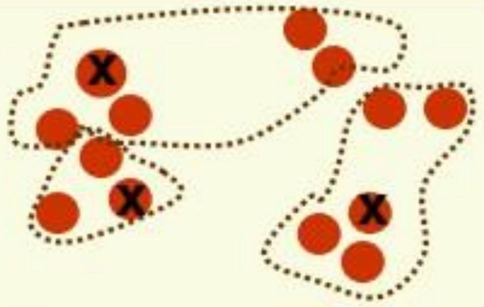
K=2

Arbitrarily choose K object as initial cluster center

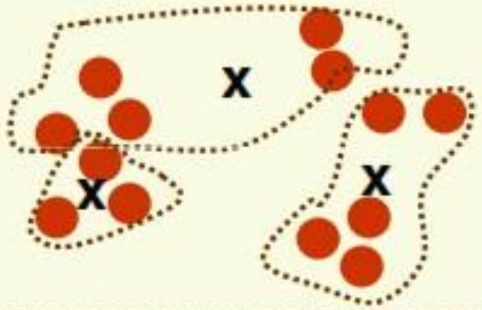
K-means Clustering

k = 3

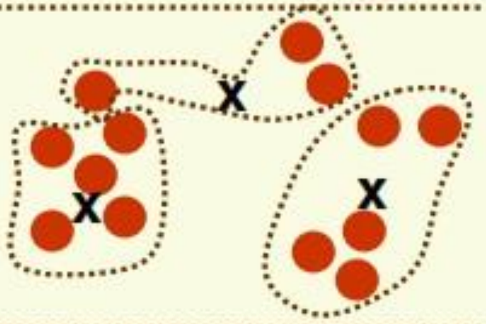
1. Initialize
   - pick ***k*** cluster centers arbitrary
   - assign each example to closest center



2. compute sample means for each cluster



3. reassign all samples to the closest mean



4. if clusters changed at step 3, go to step 2

K-means Clustering

- Pre-processing
  - Normalize the data
  - Eliminate outliers

- Post-processing
  - **Eliminate small clusters** that may represent outliers
  - **Split ‘loose’ clusters**, i.e., clusters with relatively high SSE
  - **Merge clusters that are ‘close’** and that have relatively low SSE

Distance Function

- Most commonly used functions are
  - Euclidean distance and
  - Manhattan (city block) distance
- We denote distance with: *dist*($\mathbf{x}_i$, $\mathbf{x}_j$), where $\mathbf{x}_i$ and $\mathbf{x}_j$ are data points (vectors)
- They are special cases of Minkowski distance. q is positive integer.

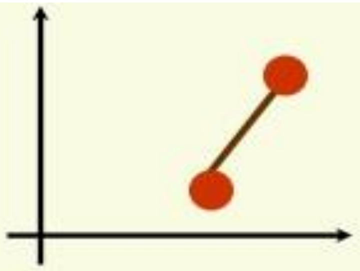
$$d(i,j) = \sqrt[q]{\underbrace{|x_{i1} - x_{j1}|}_{\text{1st dimension}}^{q} + \underbrace{|x_{i2} - x_{j2}|}_{\text{2nd dimension}}^{q} + \ldots + \underbrace{|x_{ip} - x_{jp}|}_{\text{pth dimension}}^{q}}$$

Distance (dissimilarity) Measures

Euclidean distance

$$d(x_i, x_j) = \sqrt{\sum_{k=1}^{d} \left(x_i^{(k)} - x_j^{(k)}\right)^2}$$

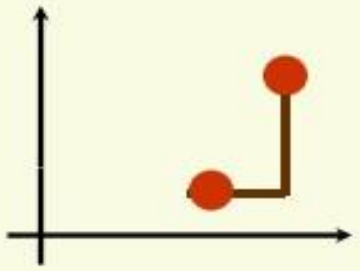
- translation invariant



Manhattan (city block) distance

$$d(x_i, x_j) = \sum_{k=1}^{d} \left| x_i^{(k)} - x_j^{(k)} \right|$$

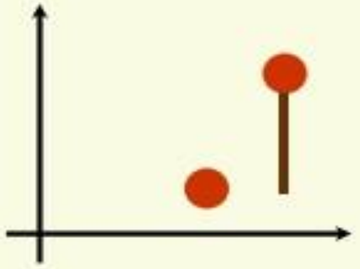
- approximation to Euclidean distance, cheaper to compute



Chebyshev distance

$$d(x_i, x_j) = \max_{1 \le k \le d} | x_i^{(k)} - x_j^{(k)} |$$

- approximation to Euclidean distance, cheapest to compute



K-means Clustering

- Time complexity for K-means clustering is

$$O(\, n \times K \times I \times d \,)$$

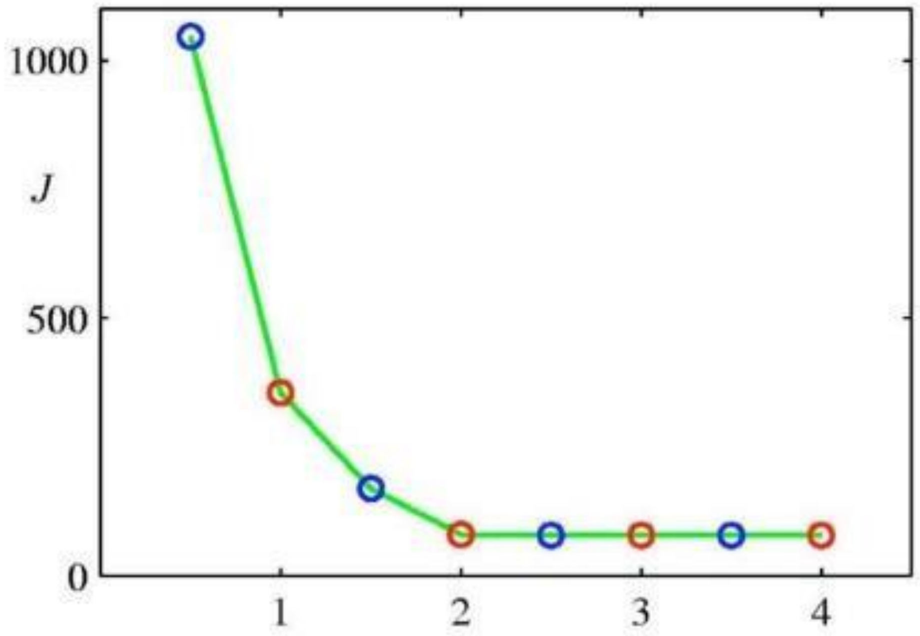
  - n = number of points,
  - K = number of clusters,
  - I = number of iterations,
  - d = number of attributes
- The storage required is

$$O((n + K)d)$$

  - n = number of points,
  - K = number of clusters,
  - d = number of attributes

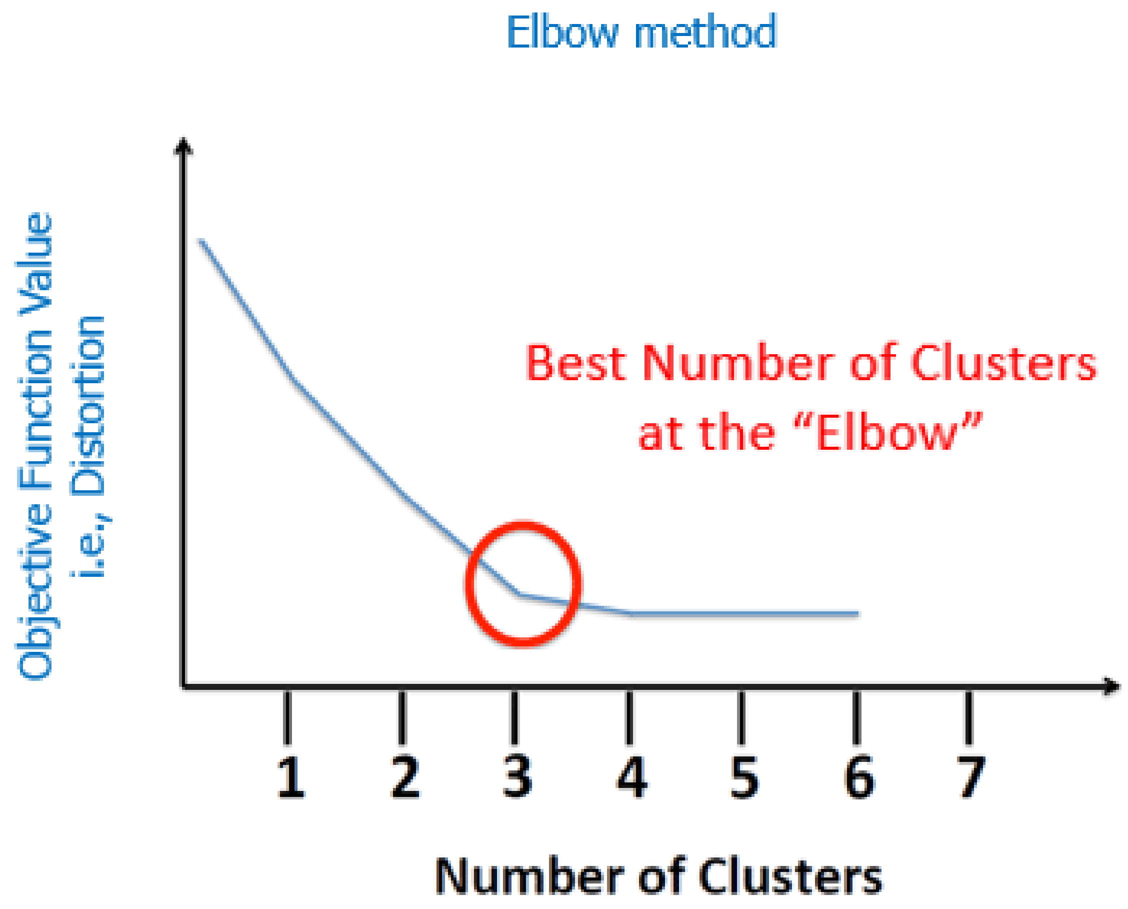
The Value of K

- One way to select K for the K-means algorithm is to try different values of K, plot the K-means objective versus K, and look at the "elbow-point" in the plot



- For the above plot, $K = 2$ is the elbow point

The Value of K



Limitations in K-means Clustering

Limitations of K-means

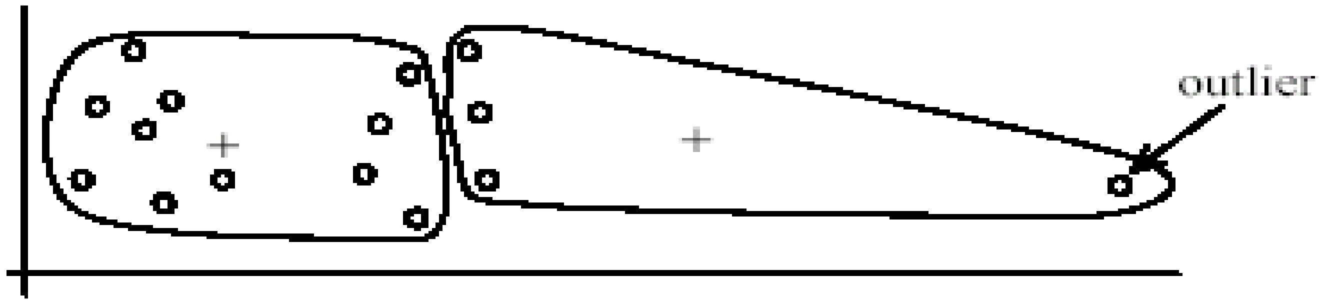
- K-means has problems when the data contains outliers

- *The K-means algorithm is very sensitive to the **initial seeds**.*

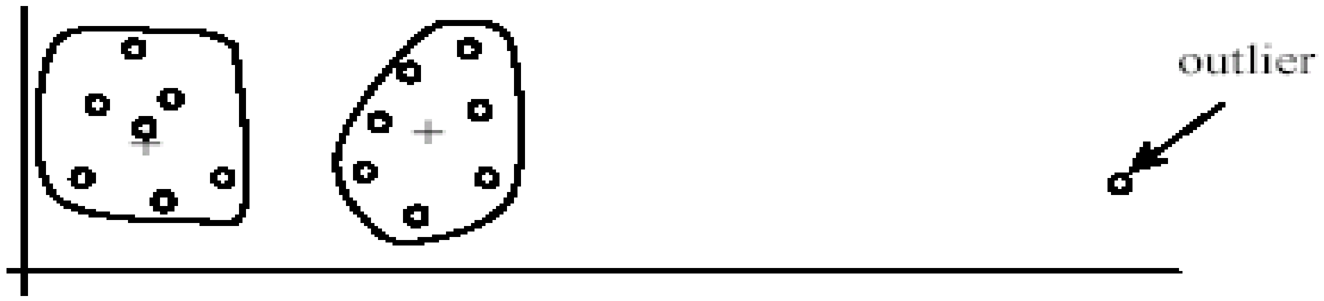
- K-means has problems when clusters are of different
  - Sizes
  - Densities
  - Non-globular shapes

Limitations of K-means

- K-means has problems when the data contains outliers



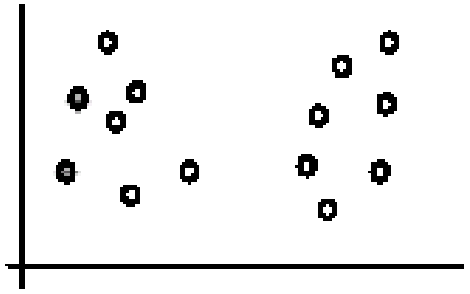
(A): Undesirable clusters



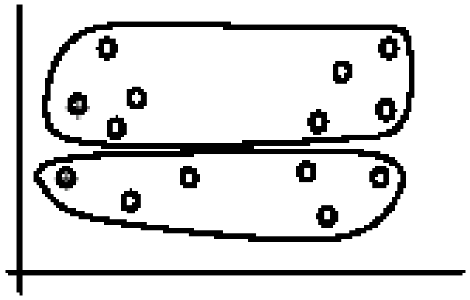
(B): Ideal clusters

Limitations of K-means

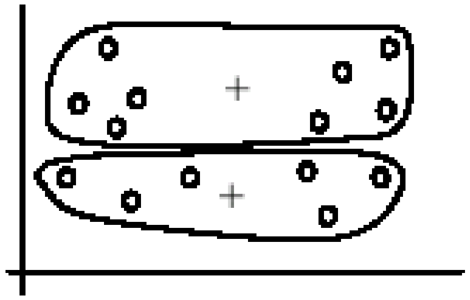
- The algorithm is sensitive to initial seeds



(A). Random selection of seeds (centroids)



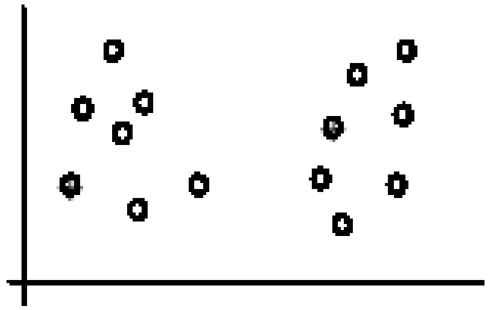
(B). Iteration 1

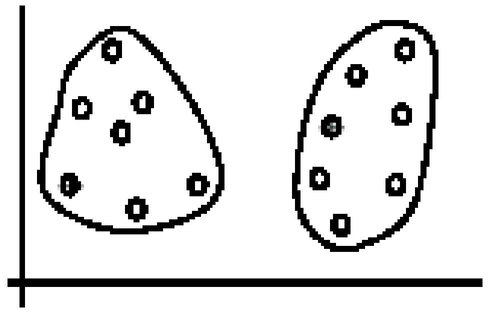


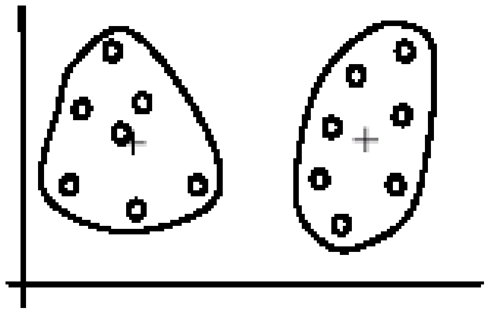
(C). Iteration 2

Limitations of K-means

- The algorithm is sensitive to initial seeds

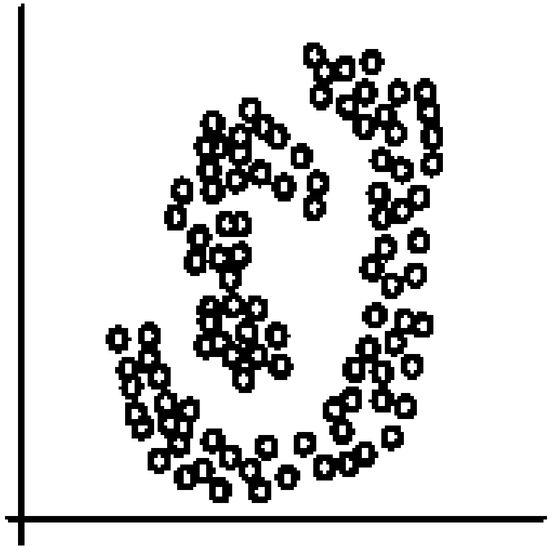

(A). Random selection of *k* seeds (centroids)

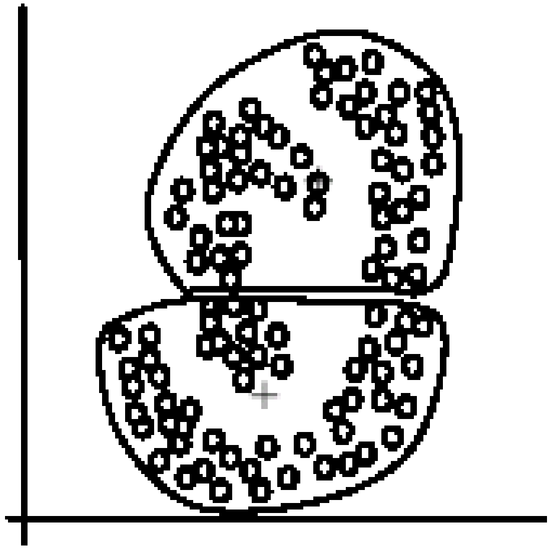

(B). Iteration 1


(C). Iteration 2

Limitations of K-means

- The *k*-means algorithm is not suitable for discovering clusters that are not hyper-ellipsoids (or hyper-spheres).


(A): Two natural clusters


(B): *k*-means clusters

K-Medoids Clustering

K-Medoids Clustering

- The k-means algorithm is sensitive to outliers!
  - Since an object with an extremely large value may substantially distort the distribution of the data.

K-Medoids:

- Instead of taking the mean value of the object in a cluster as a reference point, *medoids can be used*, which is the most centrally located object in a cluster.

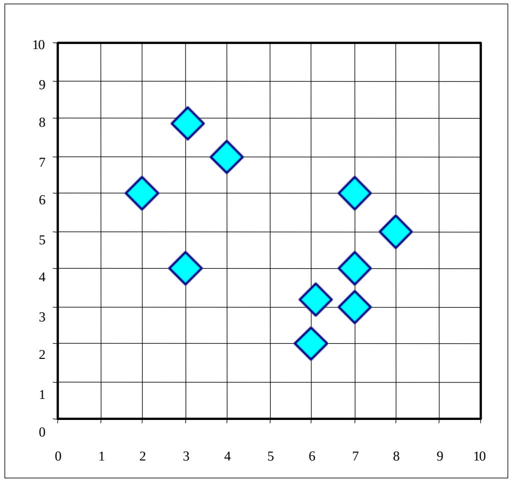
K-Medoids Clustering

- Find *representative* objects, called **medoids**, in the clusters

- ***PAM*** **(Partitioning Around Medoids, 1987)**
  - starts from an initial set of medoids and iteratively replaces one of the medoids by one of the non-medoids if it improves the total distance of the resulting clustering

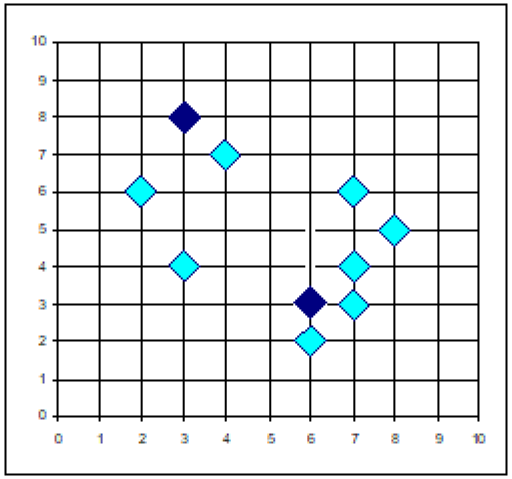
  - *PAM* works effectively for small data sets, but does not scale well for large data sets

K-Medoids Clustering



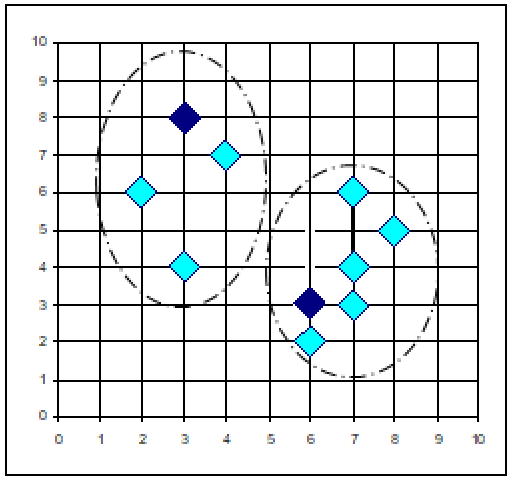
K=2

Arbitrary choose k object as initial medoids

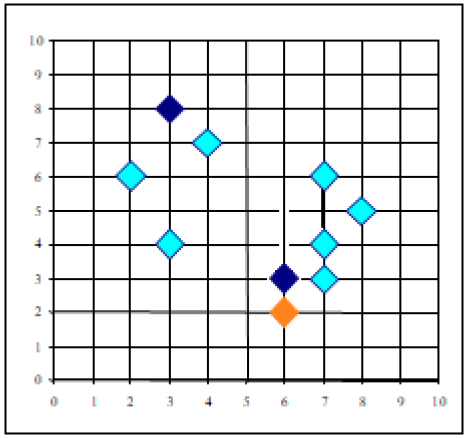


Assign each remaining object to nearest medoids

Total Cost = 20

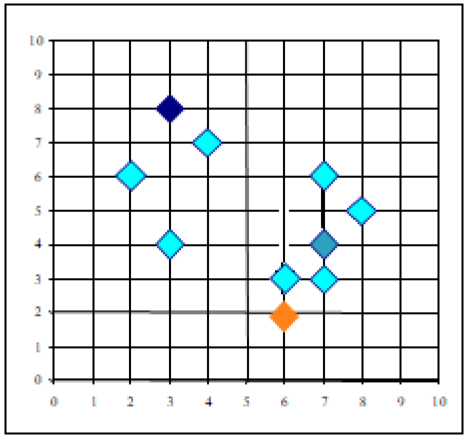


Randomly select a nonmedoid object,O_{ramdom}



Compute total cost of swapping

Total Cost = 26



Swapping O and O_{ramdom}

If quality is improved.

Do loop

Until no change

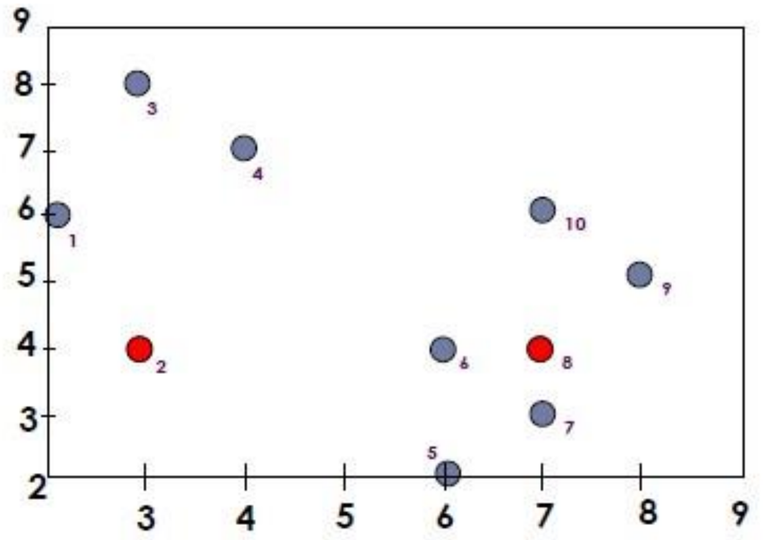
K-Medoids Clustering

- Use real object to represent the cluster
  1. Select ***k*** representative objects arbitrarily
  2. For each pair of non-selected object $\boldsymbol{h}$ and selected object $\boldsymbol{i}$, calculate the **total swapping cost** $\boldsymbol{TC_{ih}}$
  3. For each pair of $\boldsymbol{i}$ and $\boldsymbol{h}$,
     - If $TC_{ih} < 0$, $\boldsymbol{i}$ is replaced by $\boldsymbol{h}$
     - Then assign each non-selected object to the most similar representative object
  4. repeat steps 2-3 until there is no change

K-Medoids Clustering

Data Objects

	A_1	A_2
O_1	2	6
O_2	3	4
O_3	3	8
O_4	4	7
O_5	6	2
O_6	6	4
O_7	7	3
O_8	7	4
O_9	8	5
O_{10}	7	6



Goal: create two clusters

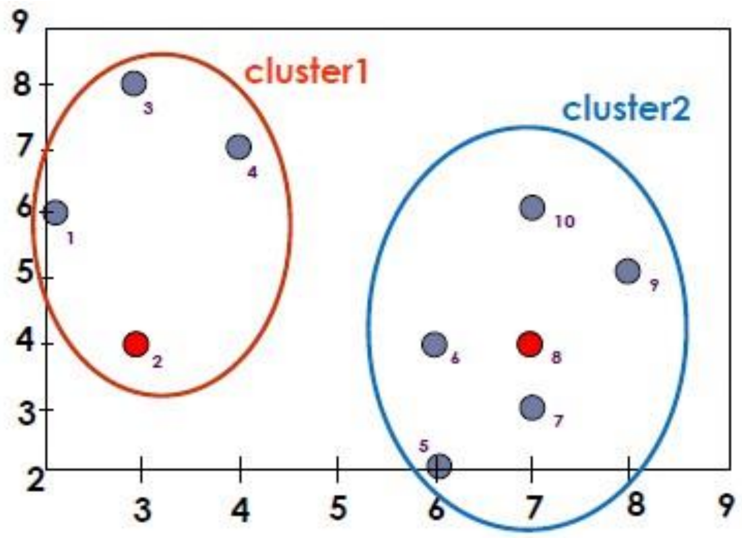
Choose randmly two medoids

$O_2 = (3,4)$
$O_8 = (7,4)$

K-Medoids Clustering

Data Objects

	A_1	A_2
O_1	2	6
O_2	3	4
O_3	3	8
O_4	4	7
O_5	6	2
O_6	6	4
O_7	7	3
O_8	7	4
O_9	8	5
O_{10}	7	6



→Assign each object to the closest representative object

→Using L1 Metric (Manhattan), we form the following clusters

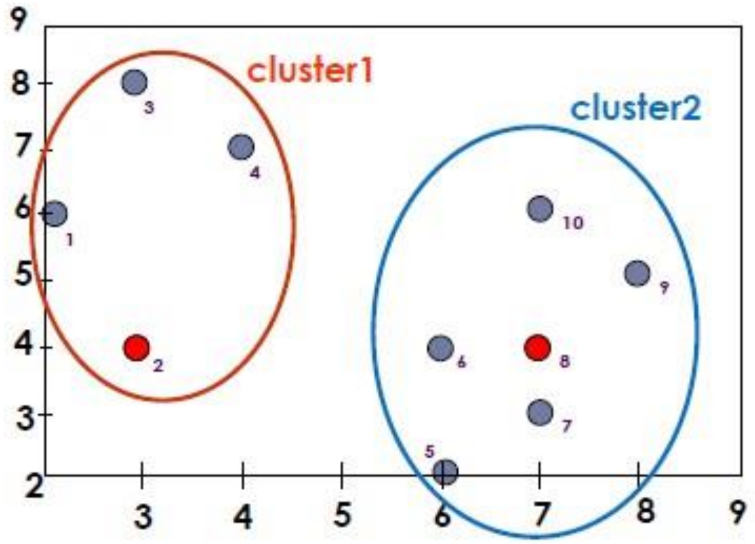
Cluster1 = $\{O_1, O_2, O_3, O_4\}$

Cluster2 = $\{O_5, O_6, O_7, O_8, O_9, O_{10}\}$

K-Medoids Clustering

Data Objects

	A_1	A_2
O_1	2	6
O_2	3	4
O_3	3	8
O_4	4	7
O_5	6	2
O_6	6	4
O_7	7	3
O_8	7	4
O_9	8	5
O_{10}	7	6



→Compute the absolute error criterion **[for the set of Medoids (O2,O8)]**

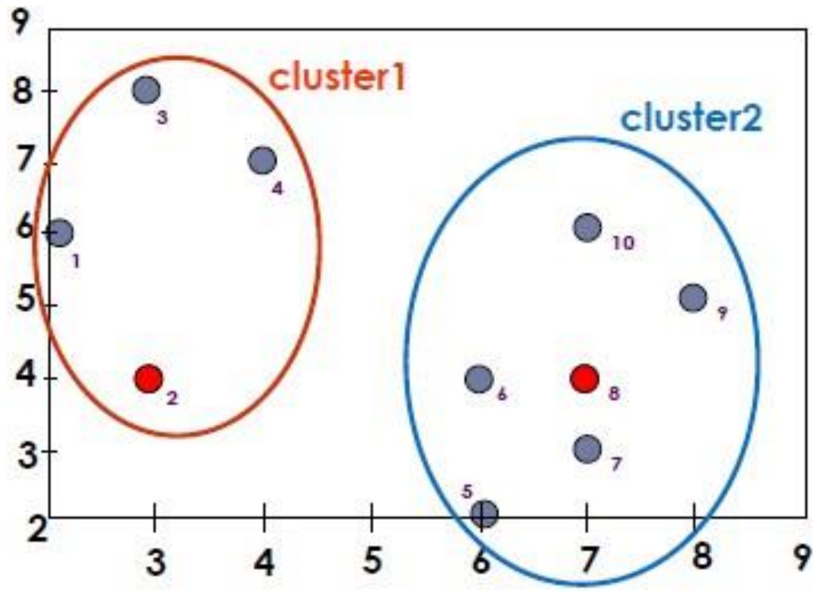
$$E=\sum_{i-1}^{k}\sum_{p\in C_i}|p-o_i|=|o_1-o_2|+|o_3-o_2|+|o_4-o_2|$$

$$+|o_5-o_8|+|o_6-o_8|+|o_7-o_8|+|o_9-o_8|+|o_{10}-o_8|$$

K-Medoids Clustering

Data Objects

	A_1	A_2
O_1	2	6
O_2	3	4
O_3	3	8
O_4	4	7
O_5	6	2
O_6	6	4
O_7	7	3
O_8	7	4
O_9	8	5
O_{10}	7	6



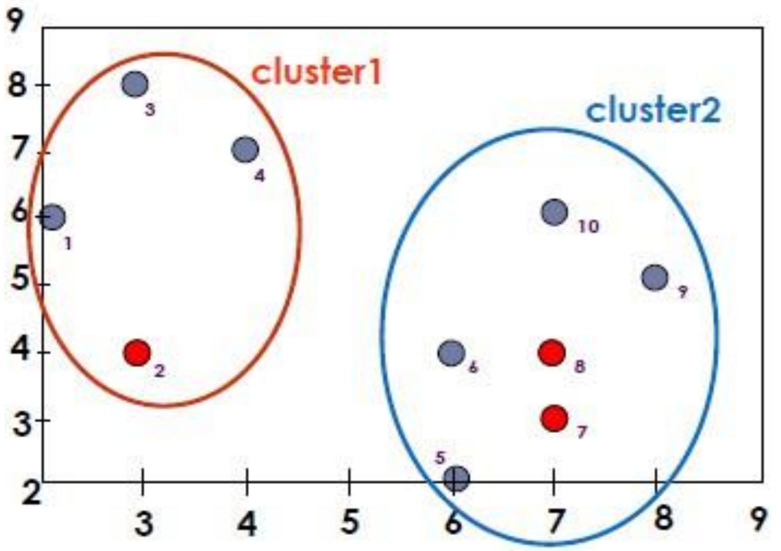
→The absolute error criterion **[for the set of Medoids (O2,O8)]**

$$E = (3+4+4)+(3+1+1+2+2) = 20$$

K-Medoids Clustering

Data Objects

	A_1	A_2
O_1	2	6
O_2	3	4
O_3	3	8
O_4	4	7
O_5	6	2
O_6	6	4
O_7	7	3
O_8	7	4
O_9	8	5
O_{10}	7	6



→Choose a random object O_7

→Swap O8 and O7

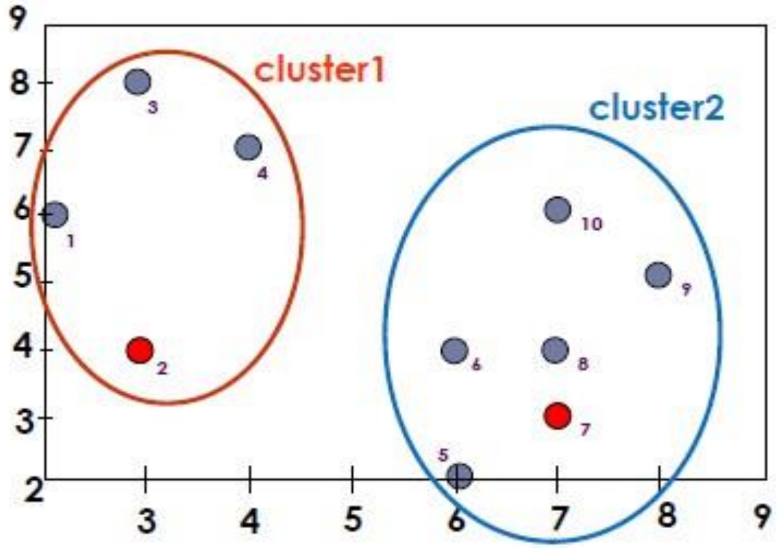
→Compute the absolute error criterion **[for the set of Medoids (O2,O7)]**

$$E = (3+4+4)+(2+2+1+3+3) = 22$$

K-Medoids Clustering

Data Objects

	A_1	A_2
O_1	2	6
O_2	3	4
O_3	3	8
O_4	4	7
O_5	6	2
O_6	6	4
O_7	7	3
O_8	7	4
O_9	8	5
O_{10}	7	6



→Compute the cost function

Absolute error [for O_2,O_7] – Absolute error [O_2,O_8]

$$S = 22 - 20$$

S> 0 ⇒ it is a bad idea to replace O_8 by O_7

K-Medoids Clustering

- *PAM is more robust than k-means in the presence of noise and outliers* because a medoid is less influenced by outliers or other extreme values than a mean
- PAM works efficiently for small data sets but **does not scale well** for large data sets.
- $O(k(n-k)^2)$ for each iteration
  - where n is # of data points,
  - k is # of clusters

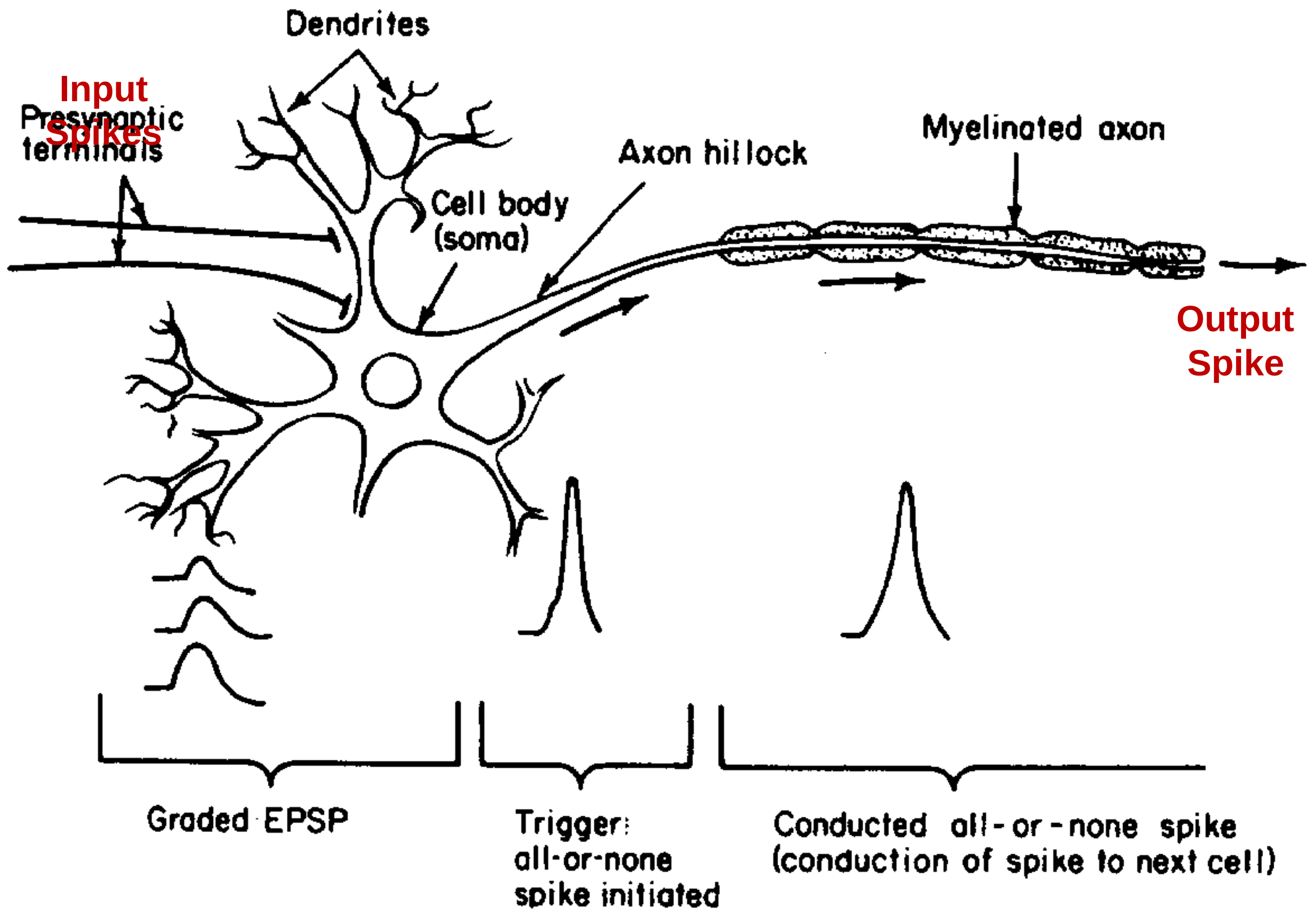
Artificial Neural Network

Biological Inspiration

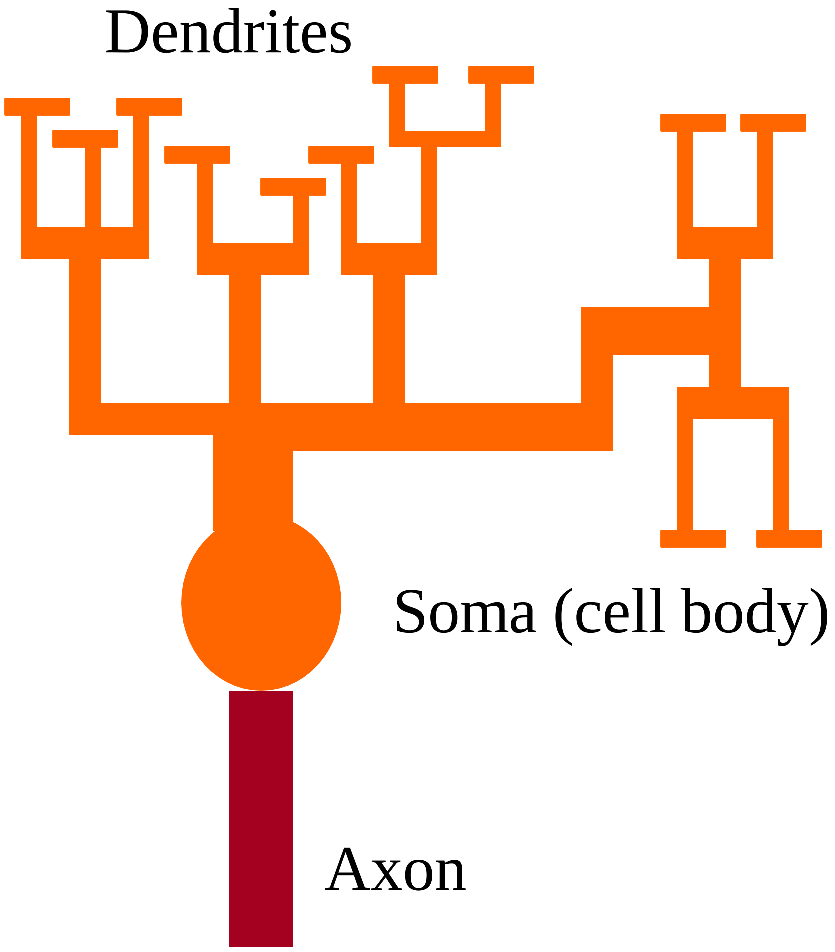
- Animals are able to **react adaptively to changes** in their external and internal environment, and they use their **nervous system** to perform these behaviours.

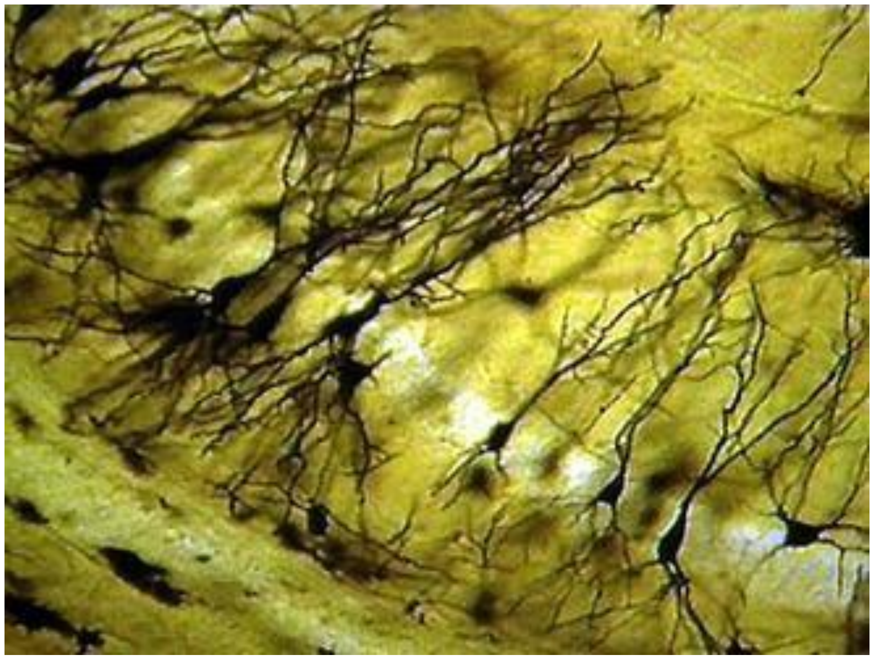
- An appropriate model/simulation of the nervous system should be able to produce similar responses and behaviours in artificial systems.

Biological Inspiration



Biological Inspiration





Biological Inspiration

Four Parts of Typical Nerve Cell:

- **Dendrites:**

  accepts the inputs

- **Soma:**

  process the inputs

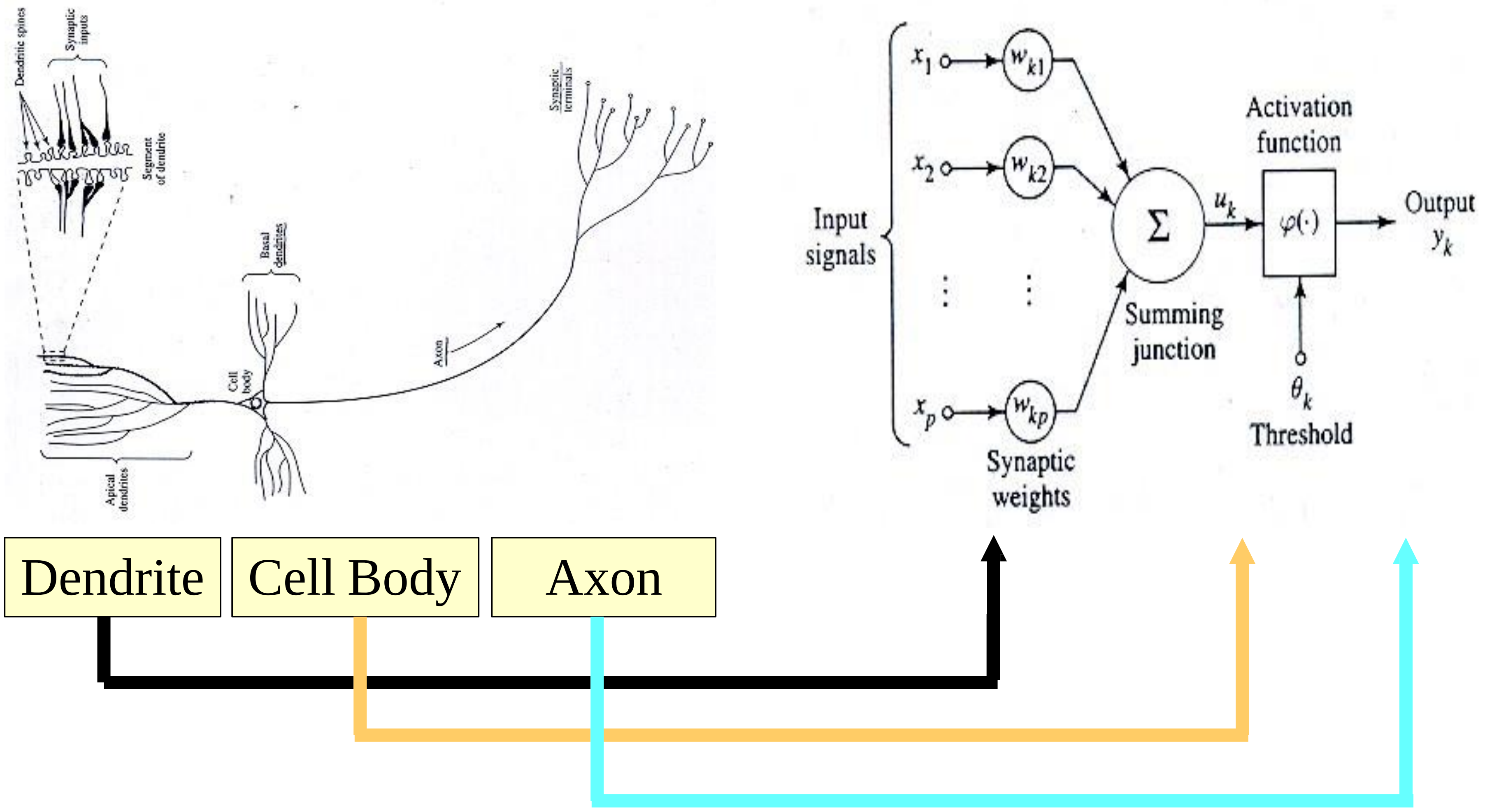
- **Axon:**

  turns the process input into outputs

- **Synapses:**

  the electromechanical contact between the neurons

Biological Inspiration



Perceptron

Perceptron

- A simplest type of ANN system is based on a unit called a **perceptron**.
- A perceptron
  - takes a **vector of real-valued inputs**,
  - calculates a linear combination of these inputs,
  - then **outputs** a 1 if the result is greater than some ***threshold*** and -1 otherwise.
- More precisely, given inputs $\boldsymbol{x_1}$ through $\boldsymbol{x_n}$ the output $\boldsymbol{o(x_1, \dots, x_n)}$ computed by the perceptron is

$$o(x_1, \dots, x_n) = \begin{cases} 1 & \text{if} \quad w_0 + w_1 x_1 +, \dots, +w_n x_n > 0 \\ -1 & \text{otherwise} \end{cases}$$

Perceptron

$$o\left(x_1,\ldots,x_n\right) = \begin{cases} 1 & \text{if } \; w_0 + w_1x_1 + ,\ldots, + w_nx_n > 0 \\ -1 & \text{otherwise} \end{cases}$$

- where each $\boldsymbol{w_i}$ is a real-valued constant, or weight,
  - that determines the contribution of input $\boldsymbol{x_i}$ to the perceptron
  - output.
- The quantity $(\boldsymbol{w_0})$ is a threshold
  - the weighted combination of inputs $\boldsymbol{w_1x_1 + \ldots + w_nx_n}$ must
  - exceed in order for the perceptron to output a 1.

Perceptron

- We may imagine an *additional constant input* $\boldsymbol{x_0} = \mathbf{1}$, allowing to write the above inequality as,

$$\sum_{i=0}^{n} w_i x_i > 0$$

or in **vector form** as

$$o(\mathbf{x}) = \begin{cases} 1 & \text{if} \quad \mathbf{w}.\mathbf{x} > 0 \\ -1 & \text{otherwise} \end{cases}$$

$$\mathbf{x} = \vec{x}$$

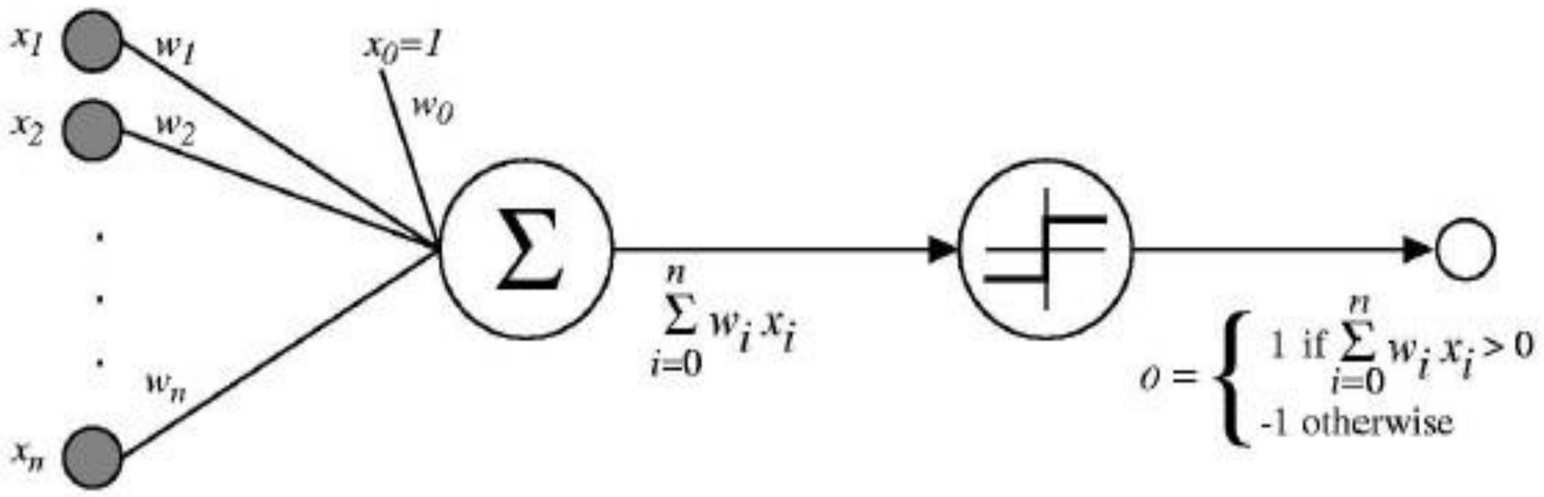
$$sgn(y) = \begin{cases} 1 & \text{if} \quad y > 0 \\ -1 & \text{otherwise} \end{cases}$$

Perceptron

- Learning a perceptron involves choosing values for the weights $w_0, \ldots, w_n$.
- Therefore, the space H of candidate hypotheses considered in perceptron learning is the *set of all possible real-valued weight vectors*

$$H = \left\{\vec{w} \mid \vec{w} \in \Re^{(n+1)}\right\}$$

Perceptron



x_1
x_2
x_n
w_1
w_2
w_n
$x_0=1$
w_0
Σ
$\sum_{i=0}^{n} w_i x_i$
$o = \begin{cases} 1 \text{ if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 \text{ otherwise} \end{cases}$

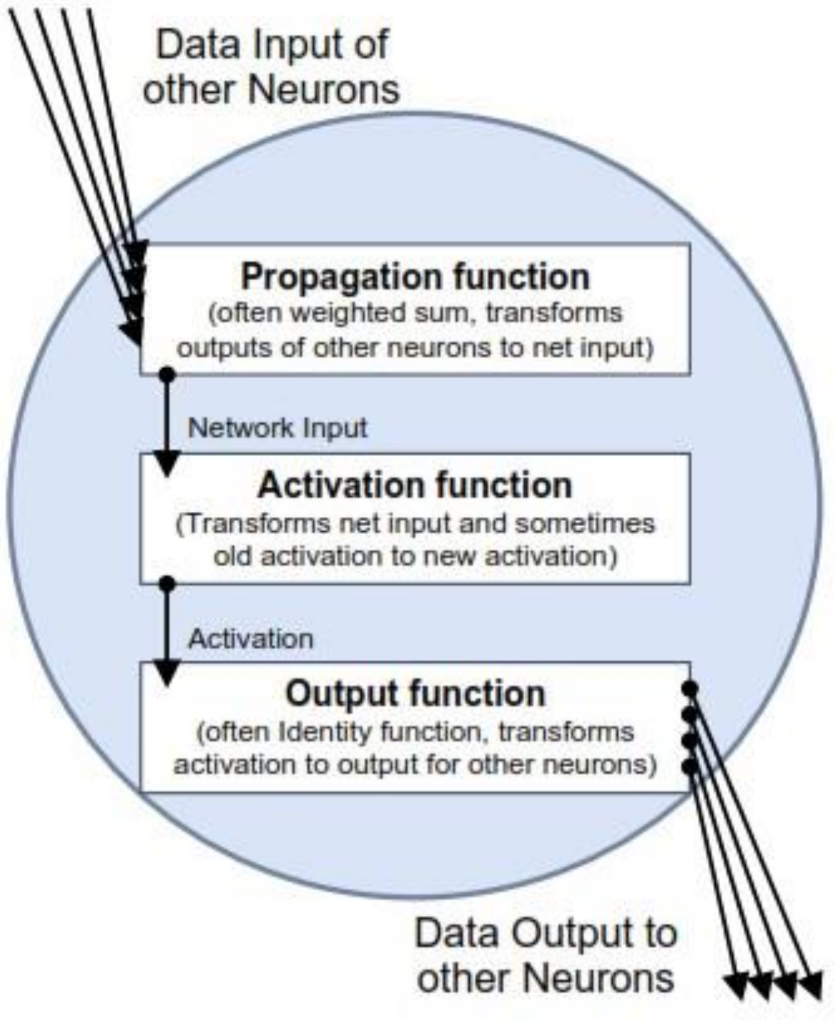
$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Neural Network Components

Neural Network Components

- A ***neural network*** is a sorted **triple** $(\boldsymbol{N}, \boldsymbol{V}, \boldsymbol{w})$ with two sets N, V and a function w,
  - whereas N is the set of *neurons* and
  - V is a sorted set $\{(i, j) | i, j \ \in \ N\}$ whose elements are called ***connections*** between neuron i and neuron j.
- The function $w : \ V \ \rightarrow \ R$ defines the ***weights***, where as $w(i, j)$,
  - The weight of the connection between neuron i and neuron j, is shortly referred to as $w_{i,j}$.

Neural Network Components


Data Input of other Neurons
Propagation function
(often weighted sum, transforms outputs of other neurons to net input)
Network Input
Activation function
(Transforms net input and sometimes old activation to new activation)
Activation
Output function
(often Identity function, transforms activation to output for other neurons)
Data Output to other Neurons

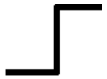
Input Neuron

- An ***input neuron*** is an ***identity neuron***. It exactly forwards the information received.
- Input neuron only forwards data
- Thus, it represents the **identity function**, which can be indicated by the symbol /
- The input neuron is represented by the symbol ⊘

Binary Neuron

- **Information processing neurons** process the input information somehow, *i.e.* do not represent the identity function.
- A **binary neuron** sums up all inputs by using the weighted sum as propagation function, which is illustrate by the sigma sign.

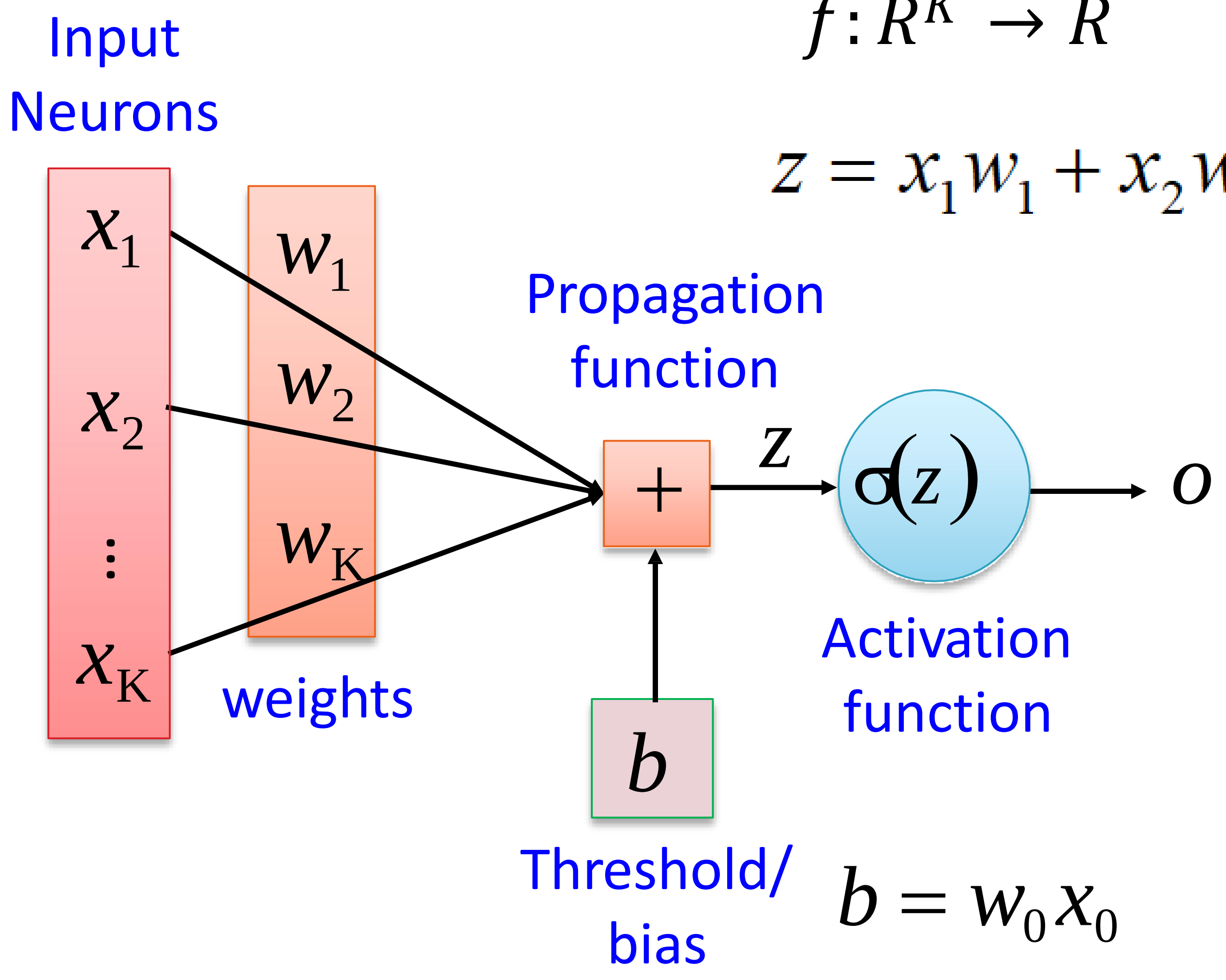
$\boxed{?}$

- The activation function of the neuron is also binary threshold function, which can be illustrated by 

Neural Network Components

$$f: R^K \to R$$

$$z = x_1 w_1 + x_2 w_2 + x_K w_K + b$$



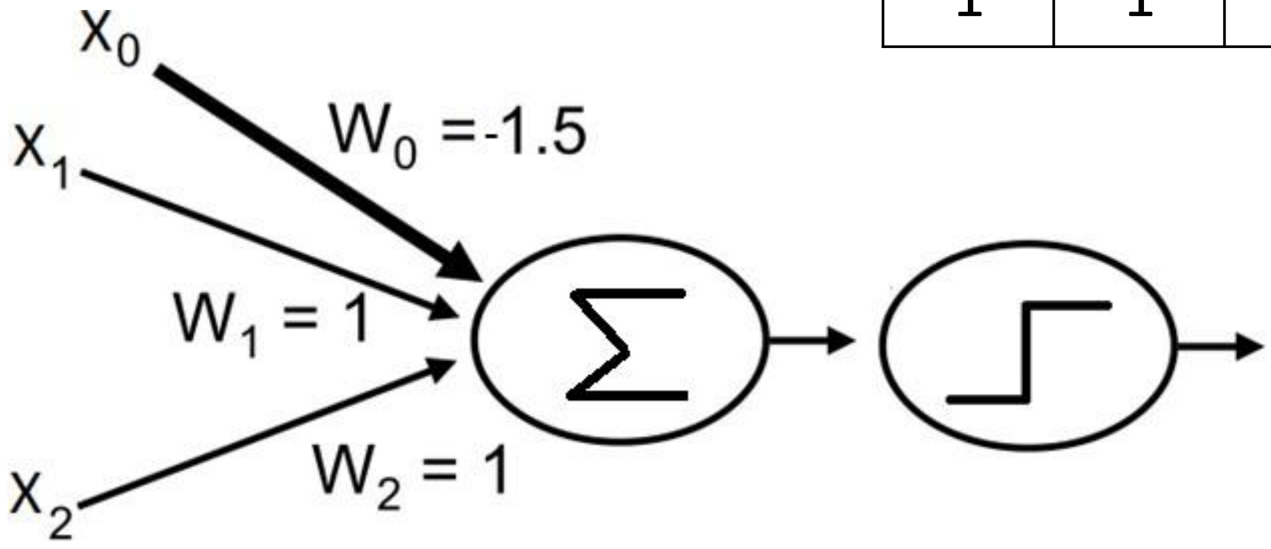
$$b = w_0 x_0$$

AND Function

X_1	X_2	Y
0	0	0
0	1	0
1	0	0
1	1	1

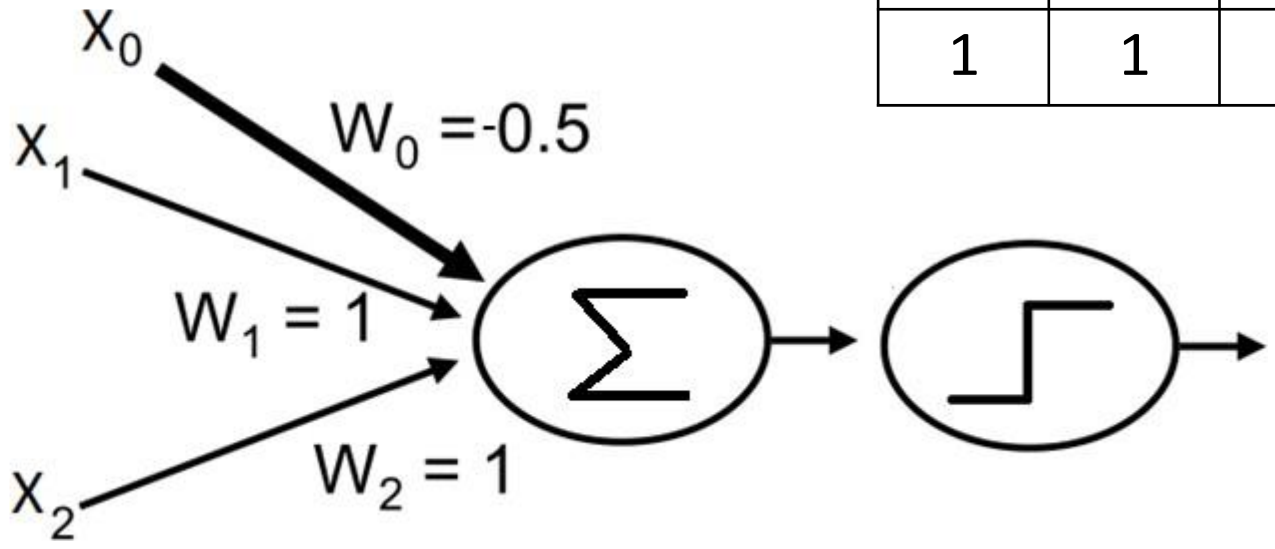
AND Function

X_1	X_2	Y
0	0	0
0	1	0
1	0	0
1	1	1

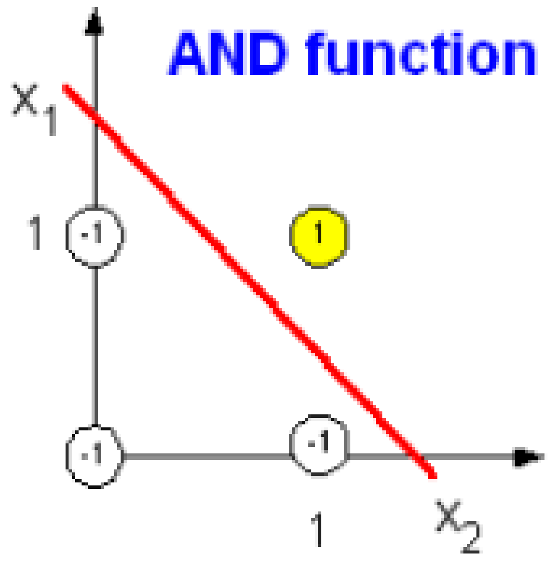


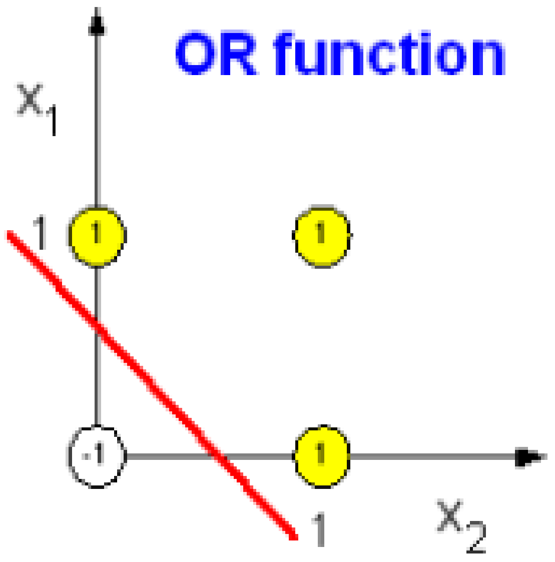
OR Function

X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	1



AND OR Function





Perceptron Training Rule

Perceptron Training Rule

- **How to learn the weights for a single perceptron.**
  - ❑ Begin with random weights,
  - ❑ Iteratively apply the perceptron to each training example,
  - ❑ Modifying the perceptron weights whenever it misclassifies an example.
  - ❑ This process is repeated, iterating through the training examples as many times as needed until the perceptron classifies all training examples correctly.
  - ❑ Weights are modified at each step according to the perceptron training rule.

Perceptron Training Rule

- The ***perceptron training rule,*** which revises the weight w_i associated with input x_i according to the rule:

$$w_i \leftarrow w_i + \Delta w_i$$

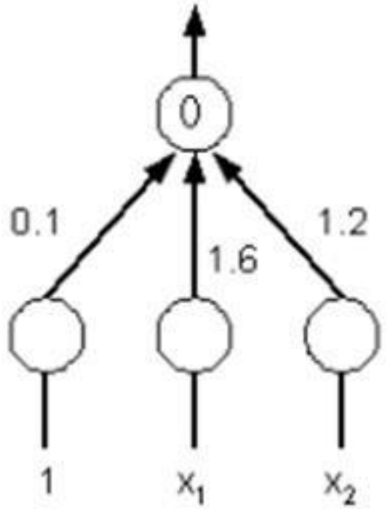
where

$$\Delta w_i = \eta(t - o)x_i$$

Where:

- t is target value
- o is perceptron output
- η is small constant (e.g., 0.1) called *learning rate*

= 0 . 5



using these updated weights:

$x_1 = 1, x_2 = 1$:	$0.1*1 + 1.6*1 + 1.2*1$	$= 2.9 \rightarrow 1$	OK
$x_1 = 1, x_2 = -1$:	$0.1*1 + 1.6*1 + 1.2*-1$	$= 0.5 \rightarrow 1$	WRONG
$x_1 = -1, x_2 = 1$:	$0.1*1 + 1.6*-1 + 1.2*1$	$= -0.3 \rightarrow -1$	OK
$x_1 = -1, x_2 = -1$:	$0.1*1 + 1.6*-1 + 1.2*-1$	$= -2.7 \rightarrow -1$	OK

$$w_i \leftarrow w_i + \Delta w_i$$

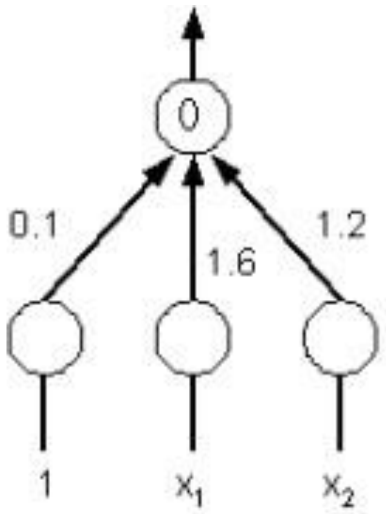
$$\Delta w_i = \eta(t - o)x_i$$

=

0

.

5



using these updated weights:

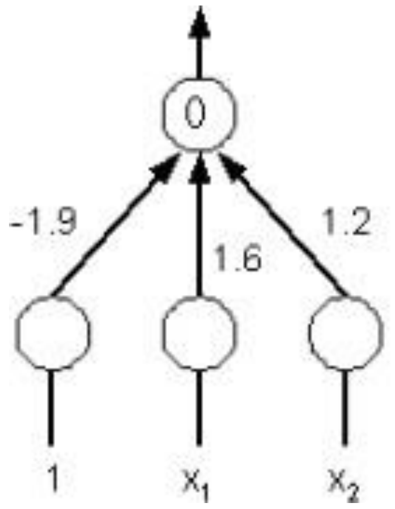
$x_1 = 1, x_2 = 1$:	$0.1*1 + 1.6*1 + 1.2*1$	$= 2.9 \rightarrow 1$		OK
$x_1 = 1, x_2 = -1$:	$0.1*1 + 1.6*1 + 1.2*-1$	$= 0.5 \rightarrow 1$		WRONG
$x_1 = -1, x_2 = 1$:	$0.1*1 + 1.6*-1 + 1.2*1$	$= -0.3 \rightarrow -1$		OK
$x_1 = -1, x_2 = -1$:	$0.1*1 + 1.6*-1 + 1.2*-1$	$= -2.7 \rightarrow -1$		OK

new weights: $w_0 = 0.1 - 1 = -0.9$
$w_1 = 1.6 - 1 = 0.6$
$w_2 = 1.2 + 1 = 2.2$

$$w_i \leftarrow w_i + \Delta w_i$$
$$\Delta w_i = \eta(t - o)x_i$$

Perceptron Training Rule

training set: $x_1 = 1, x_2 = 1 \rightarrow 1$
$x_1 = 1, x_2 = -1 \rightarrow -1$
$x_1 = -1, x_2 = 1 \rightarrow -1$
$x_1 = -1, x_2 = -1 \rightarrow -1$



using these updated weights:
$x_1 = 1, x_2 = 1$: $-1.9*1 + 1.6*1 + 1.2*1 = 0.9 \rightarrow 1$ OK
$x_1 = 1, x_2 = -1$: $-1.9*1 + 1.6*1 + 1.2*-1 = -1.5 \rightarrow -1$ OK
$x_1 = -1, x_2 = 1$: $-1.9*1 + 1.6*-1 + 1.2*1 = -2.3 \rightarrow -1$ OK
$x_1 = -1, x_2 = -1$: $-1.9*1 + 1.6*-1 + 1.2*-1 = -4.7 \rightarrow -1$ OK

DONE!

Perceptron Training Rule

Example:

- The training rule will increase *w*, if $(t - o)$, η and x_i are all positive.
    - if $x_i = 0.8$, $\eta = 0.1$, $t = 1$, and $o = -1$, then the weight update will be

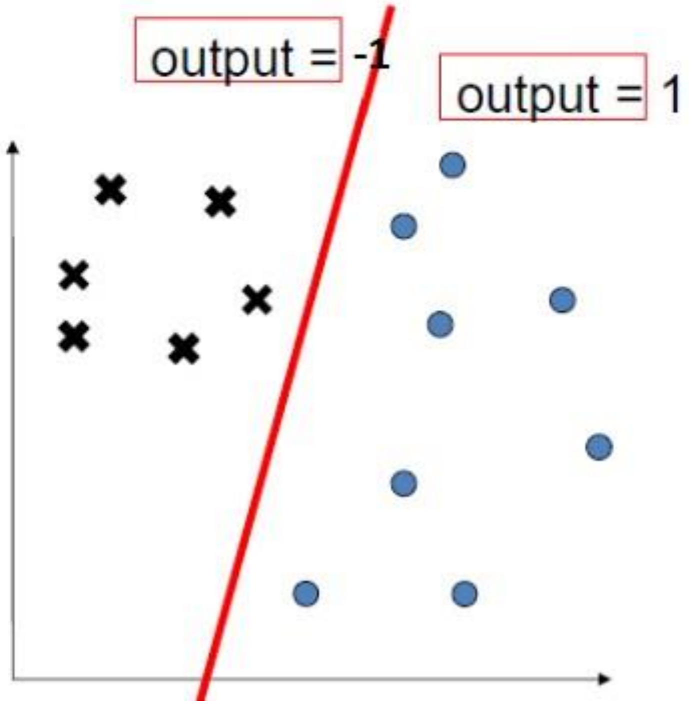
$$\Delta w_i = \eta(t - o)x_i = 0.1(1 - (-1))0.8 = 0.16.$$

- On the other hand,
    - if $x_i = 0.8$, $\eta = 0.1$, $t = -1$ and $o = 1$, then weights associated with positive x_i will be decreased rather than increased.

$$\Delta w_i = \eta(t - o)x_i = 0.1(-1 - (1))0.8 = -0.16.$$

Perceptron Training Rule

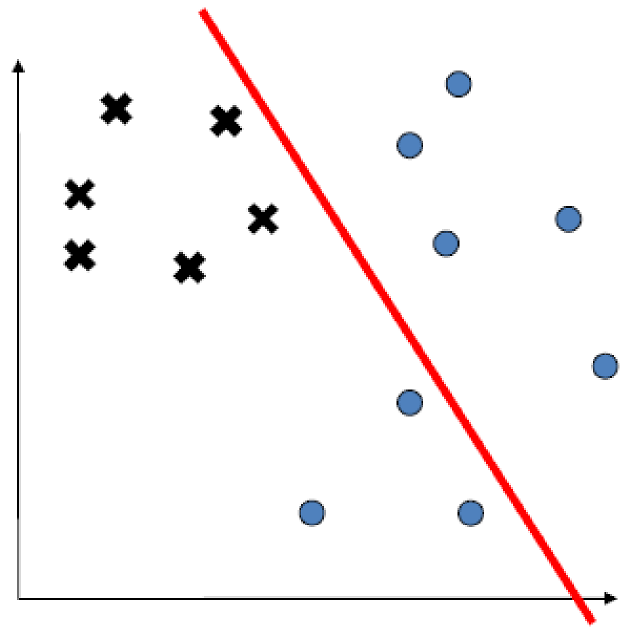
$$\begin{cases} \sum_{i=1}^{M} w_i x_i > 0 & output = 1 \\ else & output = -1 \end{cases}$$



Perceptron Training Rule

$$\begin{cases} \sum_{i=1}^{M} w_i x_i > 0 & output = 1 \\ else & output = -1 \end{cases}$$

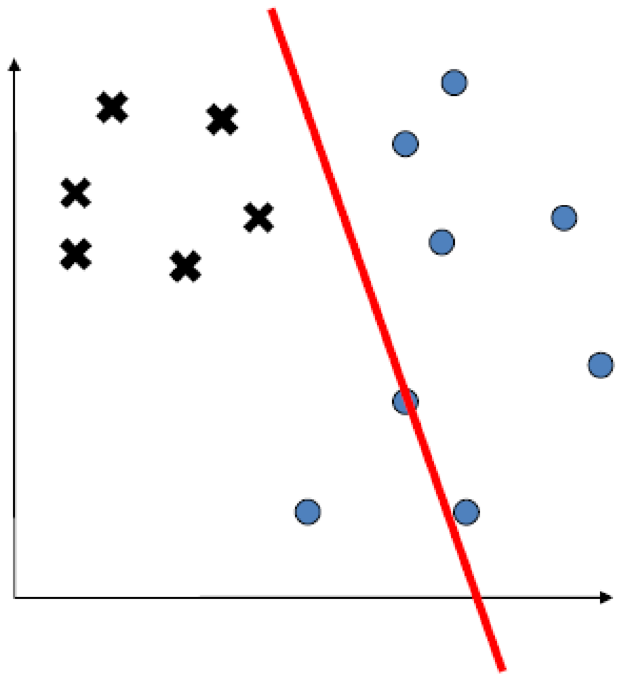
$$w_1 = 1, w_2 = 0.2, w_0 = 0.05$$



Perceptron Training Rule

$$\begin{cases} \sum_{i=1}^{M} w_i x_i > 0 & output = 1 \\ else & output = -1 \end{cases}$$

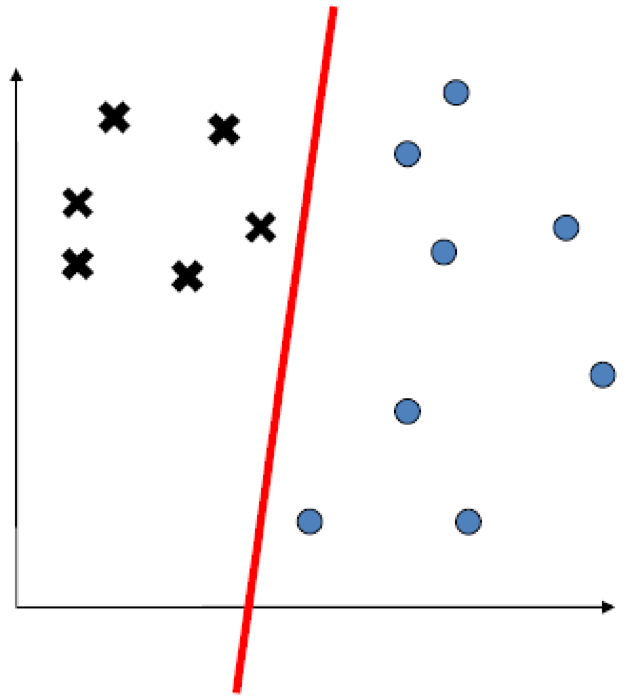
$$w_1 = 2.1, w_2 = 0.2, w_0 = 0.05$$



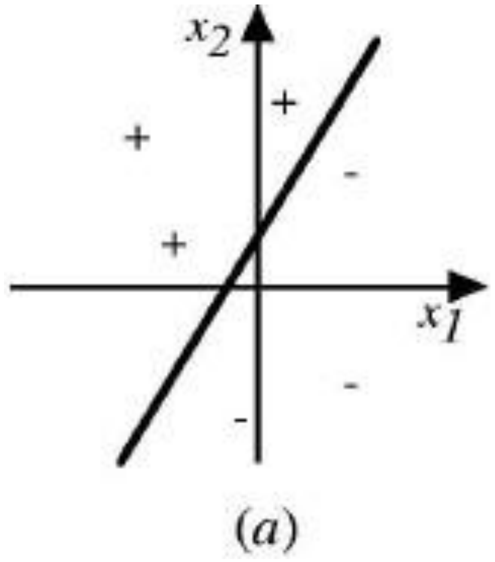
Perceptron Training Rule

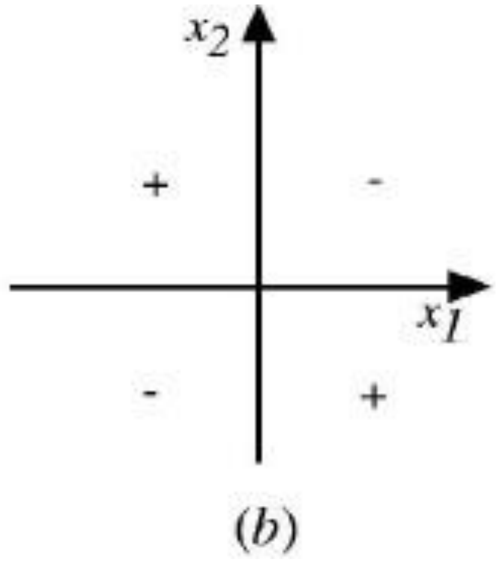
$$\begin{cases} \overset{M}{\underset{i=1}{\square}} \; w_i x_i > 0 & output = 1 \\ else & output = -1 \end{cases}$$

$$w_1 = -0.8, w_2 = 0.03, w_0 = 0.05$$



Perceptron





The **decision surface** represented by a **two-input perceptron x_1** and **x_2**. ***(a)*** A set of training examples and the decision surface of a perceptron that classifies them correctly. ***(b)*** A set of training examples that is not linearly separable.

Perceptron Training Rule

- The **perceptron rule** finds a successful weight vector when the training examples are **linearly separable**,
- It fails to converge if the examples are **not linearly separable**.
- The solution is … **Delta Rule** also known as **(Widrow-Hoff Rule)**

Delta Rule

- use ***gradient descent*** to search the hypothesis space of possible weight vectors to find the weights that best fit the training examples.

Reading Material

- **Artificial Intelligence,** A Modern Approach

  Stuart J. Russell and Peter Norvig
  - **Chapter 18.**
- **Machine Learning**

  Tom M. Mitchell
  - **Chapter 4.**