CS 461
Artificial Intelligence

Dr. Hashim Yasin

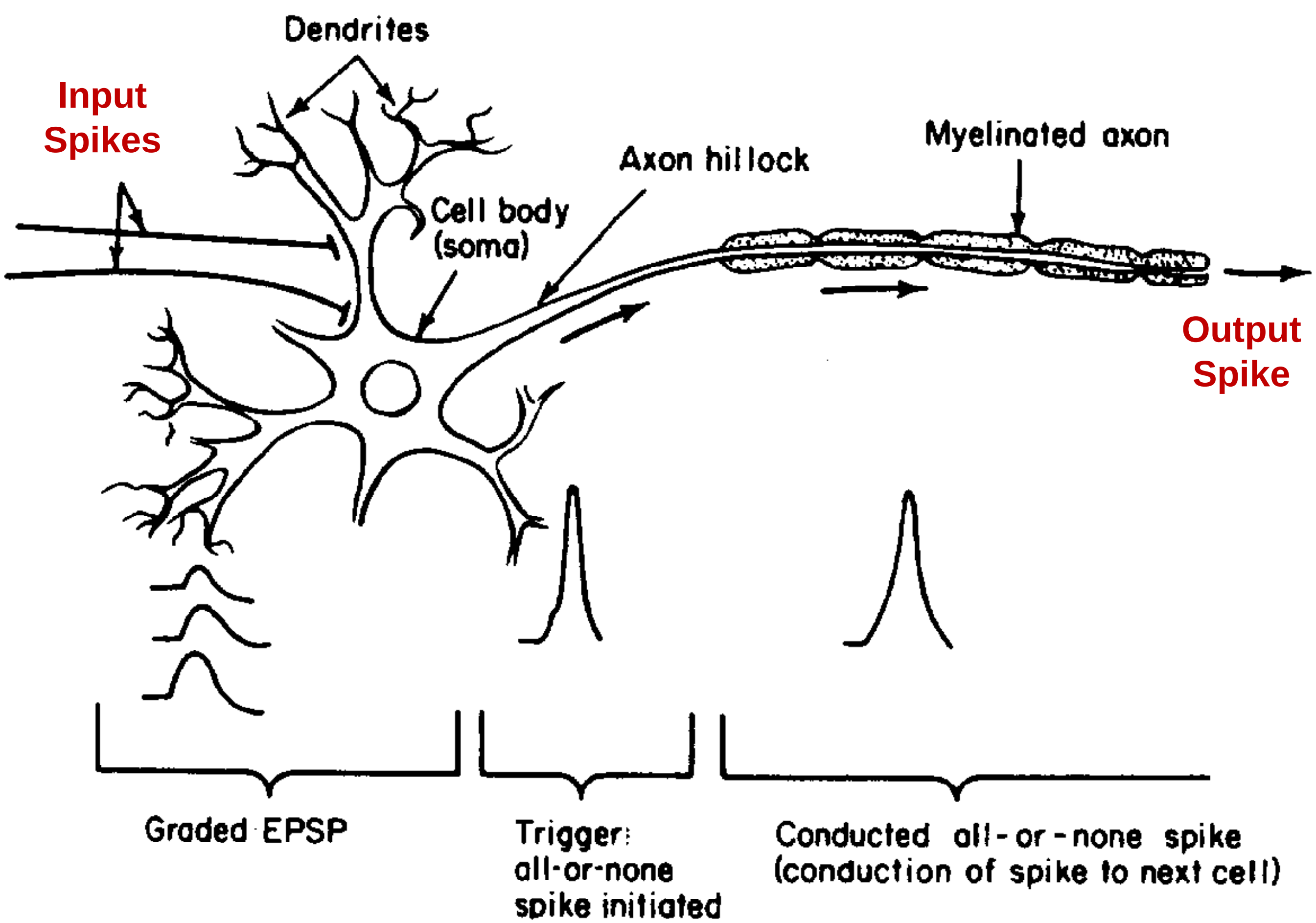
Artificial Neural Network

Biological Inspiration

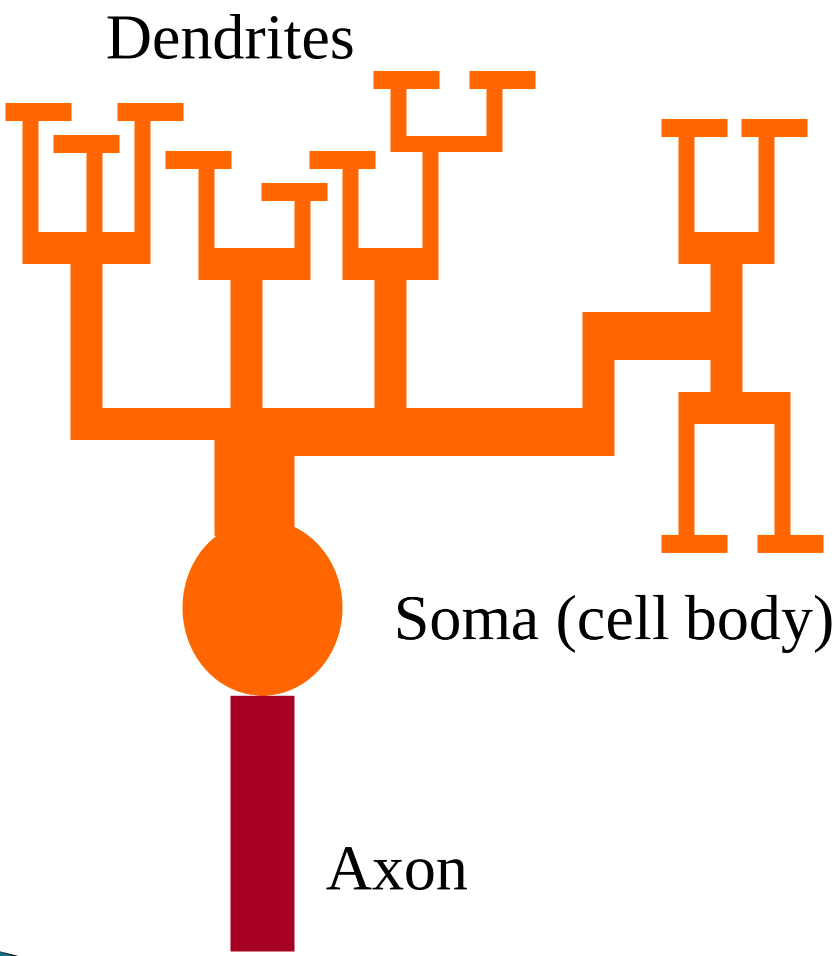
- Animals are able to **react adaptively to changes** in their external and internal environment, and they use their **nervous system** to perform these behaviours.

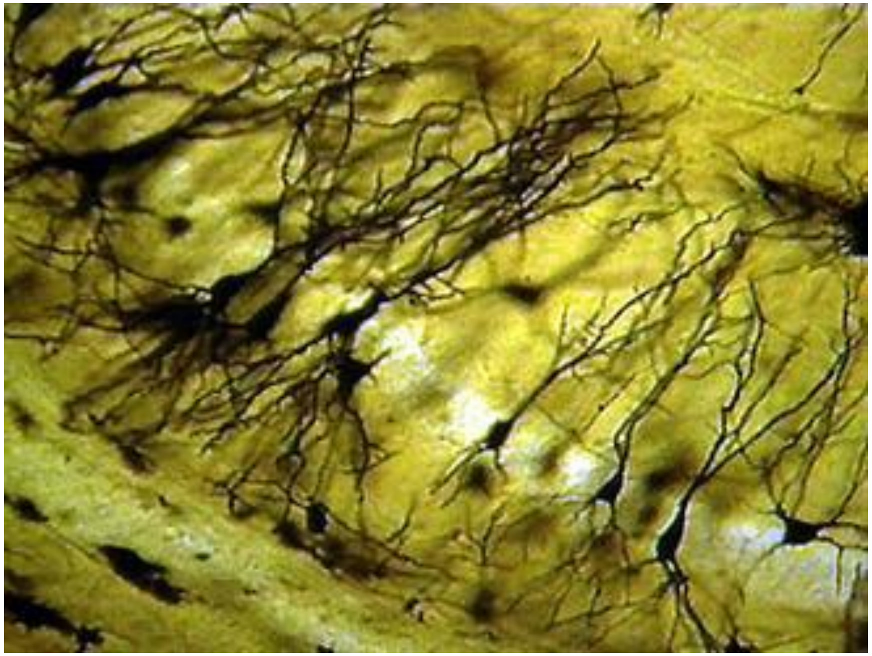
- An appropriate model/simulation of the nervous system should be able to produce similar responses and behaviours in artificial systems.

Biological Inspiration



Biological Inspiration





Biological Inspiration

Four Parts of Typical Nerve Cell:

- **Dendrites:**

  accepts the inputs

- **Soma:**

  process the inputs

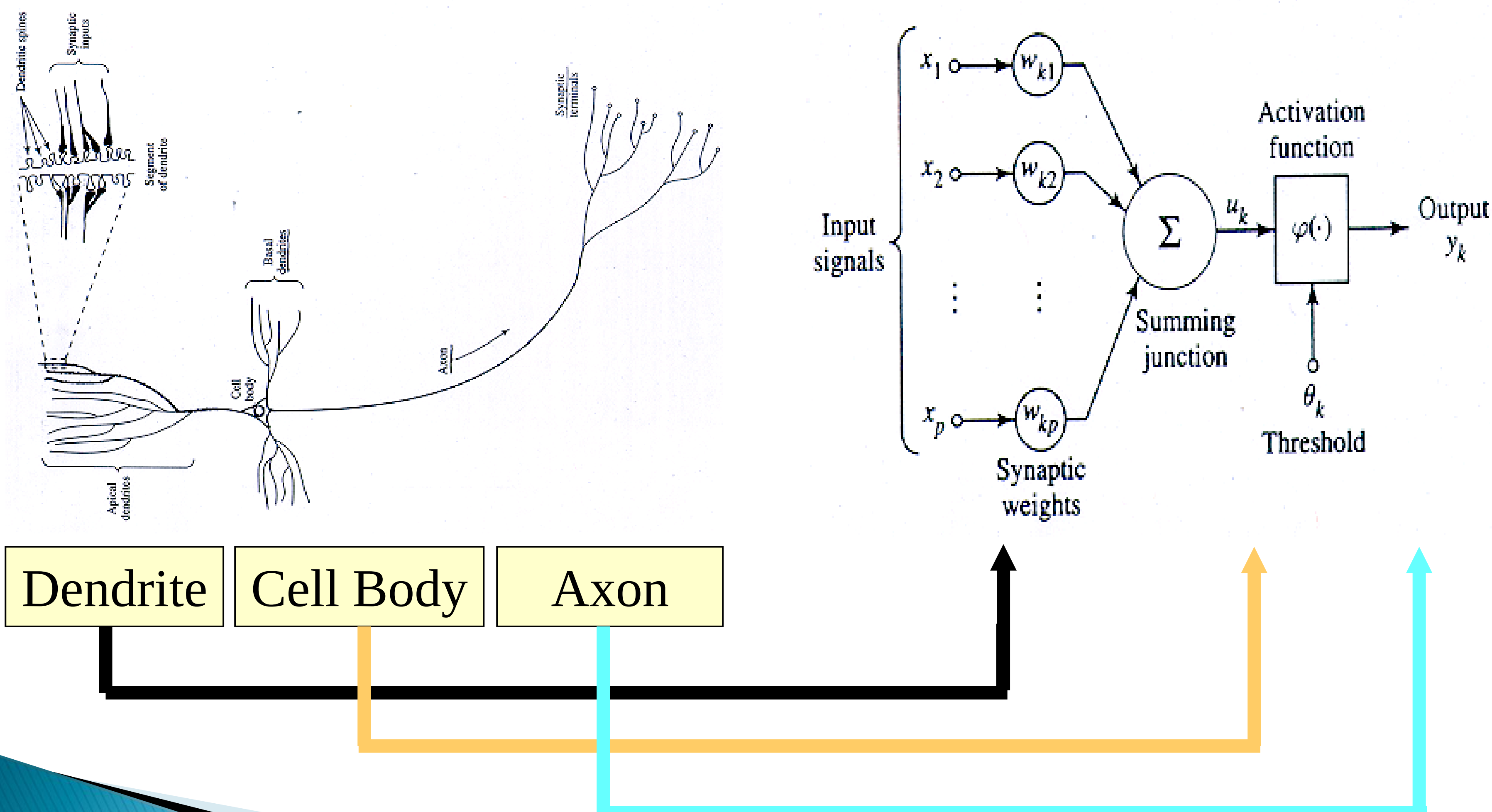
- **Axon:**

  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 $\boldsymbol{o(x_1, \ldots, x_n)}$ computed by the perceptron is

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

Perceptron

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

- 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 $\boldsymbol{w_1 x_1} + \dots + \boldsymbol{w_n x_n}$ must exceed in order for the perceptron to output a 1.

Perceptron

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

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

or in **vector form** as

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

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

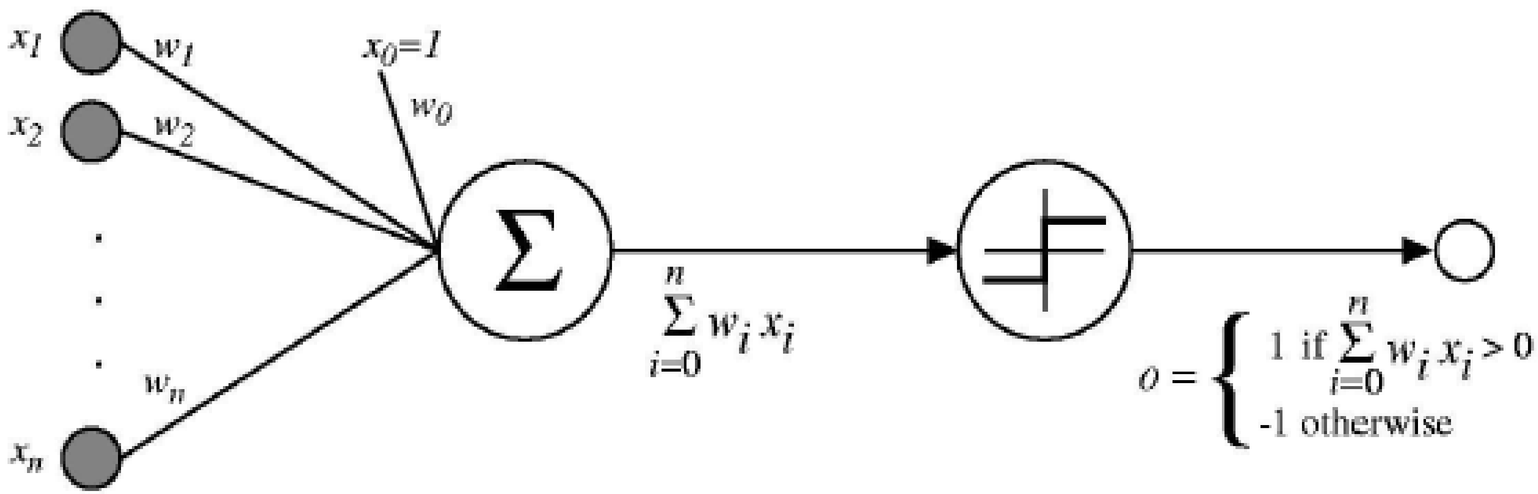
$$sgn(y) = \begin{cases} 1 & \text{if} \quad y > 0 \\ -1 & \text{otherwise} \end{cases}$$

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*

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

Perceptron



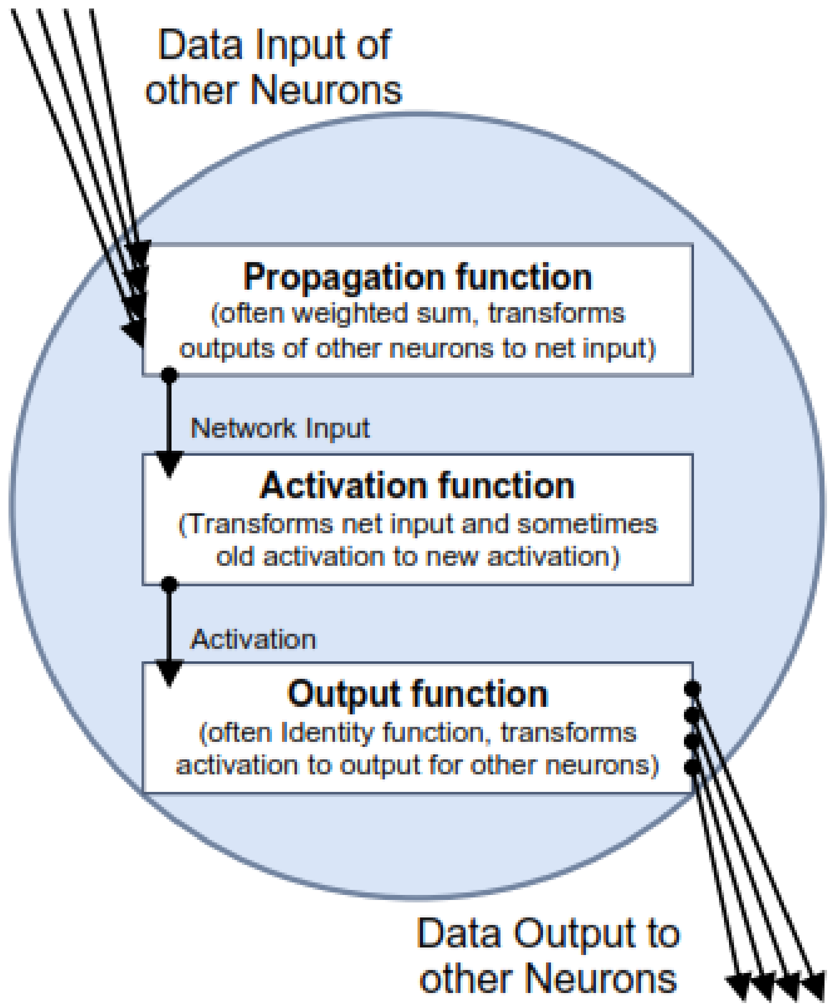
$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Neural Network Components

Neural Network Components

- A ***neural network*** is a sorted **triple** $\boldsymbol{(N, V, w)}$ 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

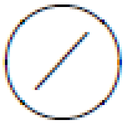


Input Neuron

- 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.

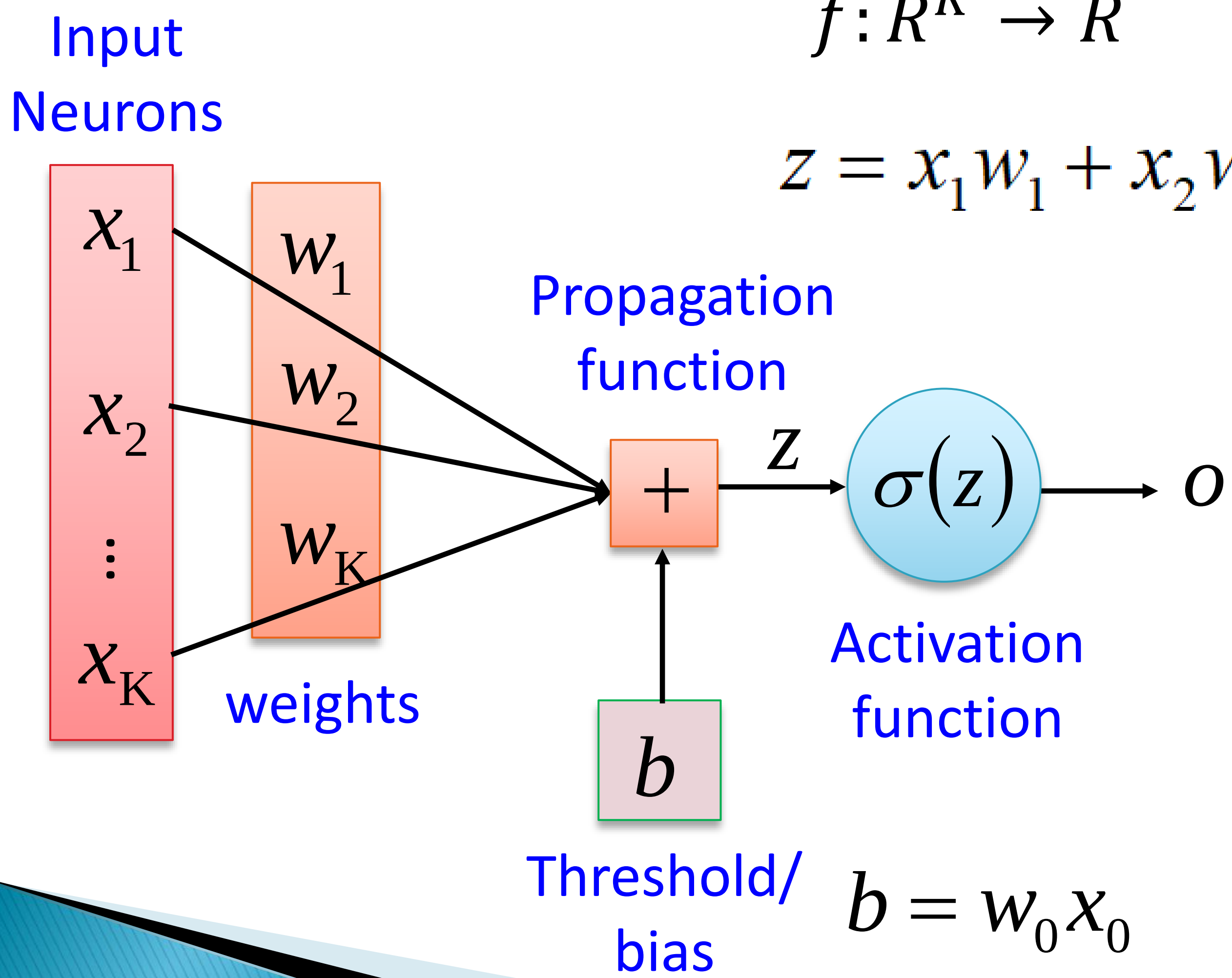
$$\sum$$

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

Neural Network Components

$$f: R^K \rightarrow R$$

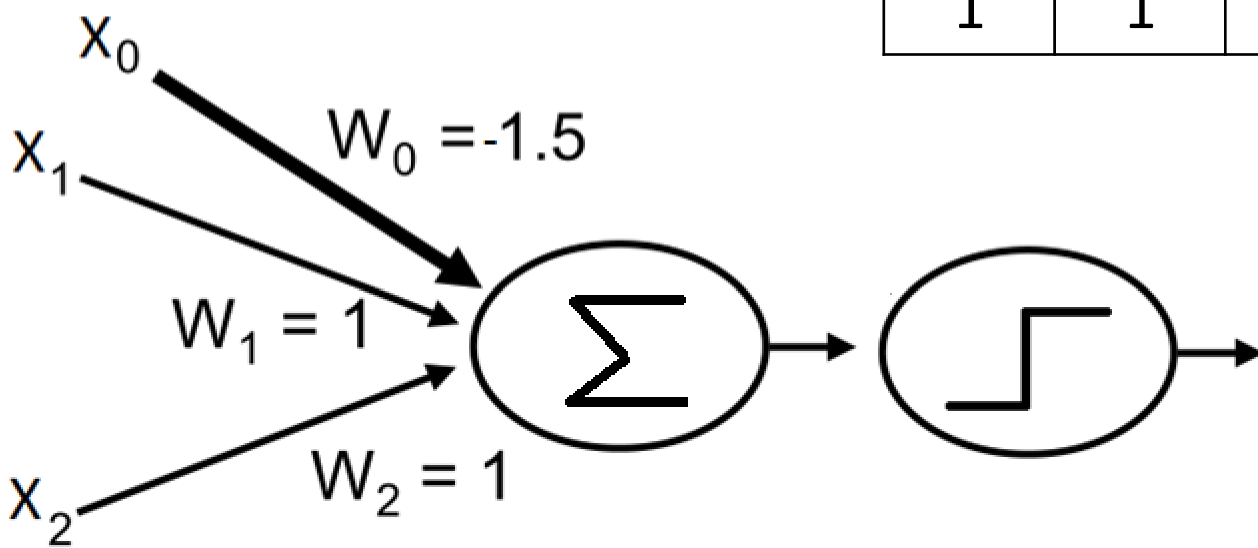
$$z = x_1 w_1 + x_2 w_2 + x_K w_K + b$$



$$b = w_0 x_0$$

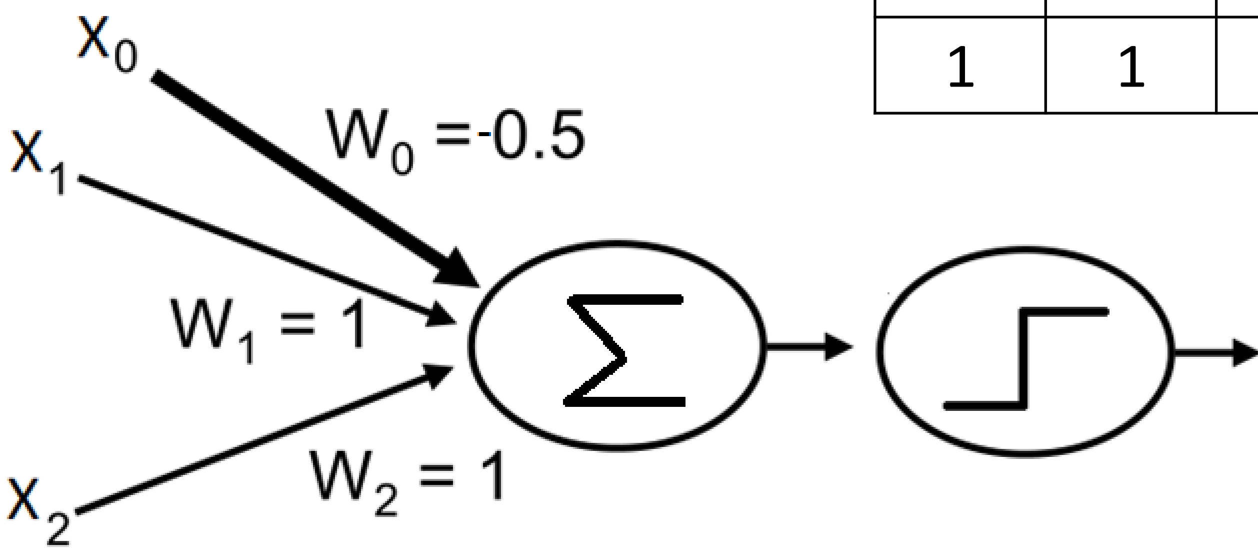
AND Function

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

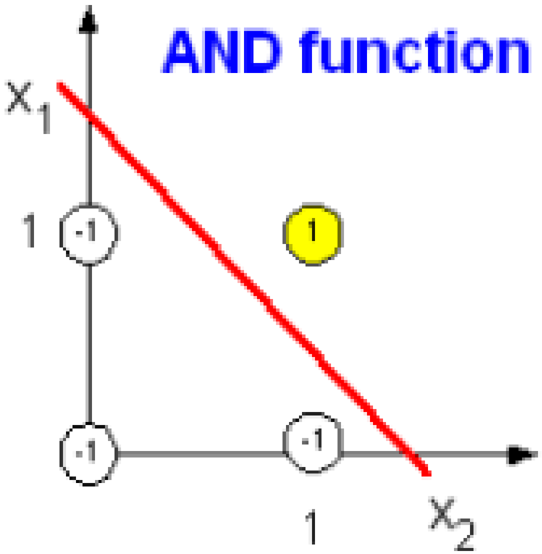


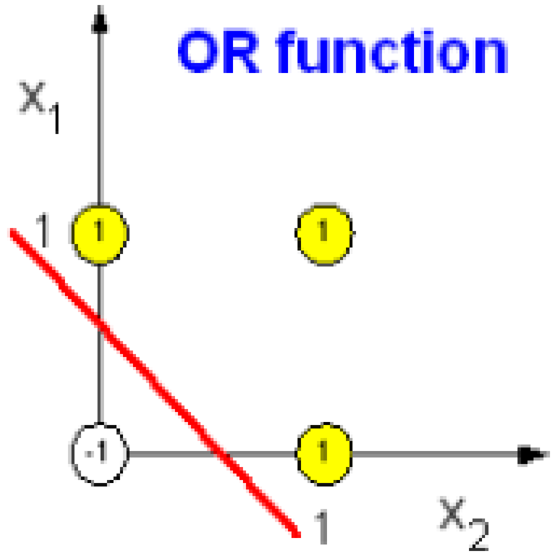
OR Function

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*

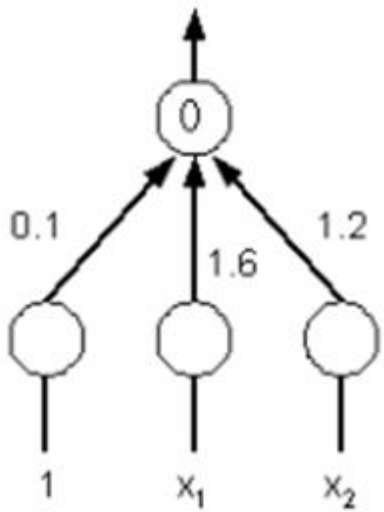
Perceptron Training Rule

training set: $x_1 = 1, x_2 = 1 \rightarrow 1$ $\eta = 0.5$

$x_1 = 1, x_2 = -1 \rightarrow -1$

$x_1 = -1, x_2 = 1 \rightarrow -1$

$x_1 = -1, x_2 = -1 \rightarrow -1$



using these updated weights:

$x_1 = 1, x_2 = 1$:	$0.1*1 + 1.6*1 + 1.2*1$	$= 2.9 \rightarrow 1$	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

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

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

Perceptron Training Rule

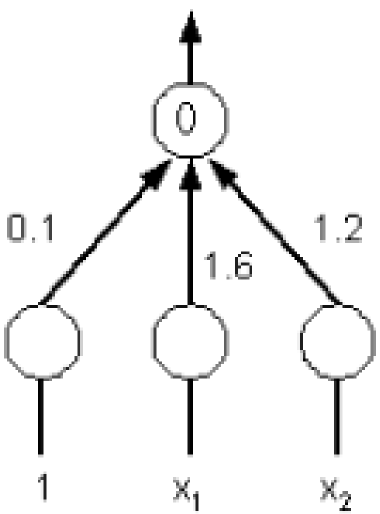
training set:

$x_1 = 1, x_2 = 1 \rightarrow 1$ $\quad \eta = 0.5$

$x_1 = 1, x_2 = -1 \rightarrow -1$

$x_1 = -1, x_2 = 1 \rightarrow -1$

$x_1 = -1, x_2 = -1 \rightarrow -1$



using these updated weights:

$x_1 = 1, x_2 = 1$:	$0.1*1 + 1.6*1 + 1.2*1$	$= 2.9 \rightarrow 1$	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: $w_0 = 0.1 - 1 = -0.9$

$w_1 = 1.6 - 1 = 0.6$

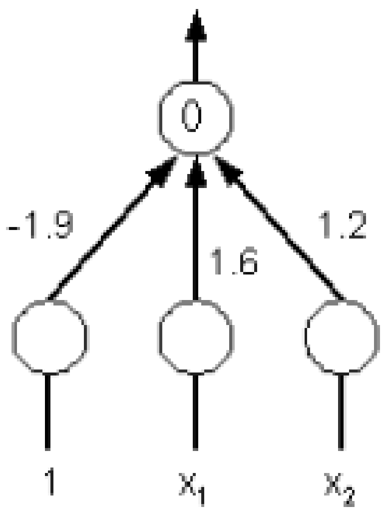
$w_2 = 1.2 + 1 = 2.2$

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

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

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

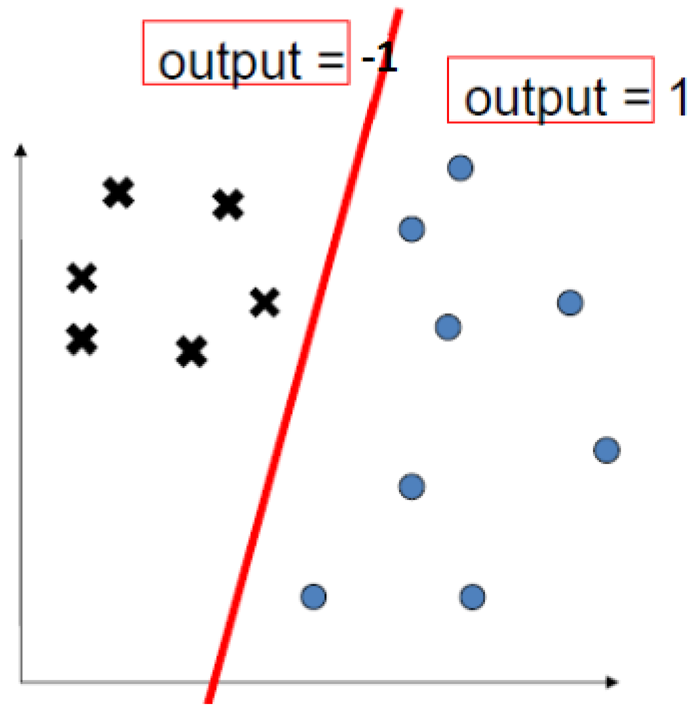
$$\Delta w_i = \eta(t - o)x_i = 0.1(1 - (-1))0.8 = 0.16.$$

- 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

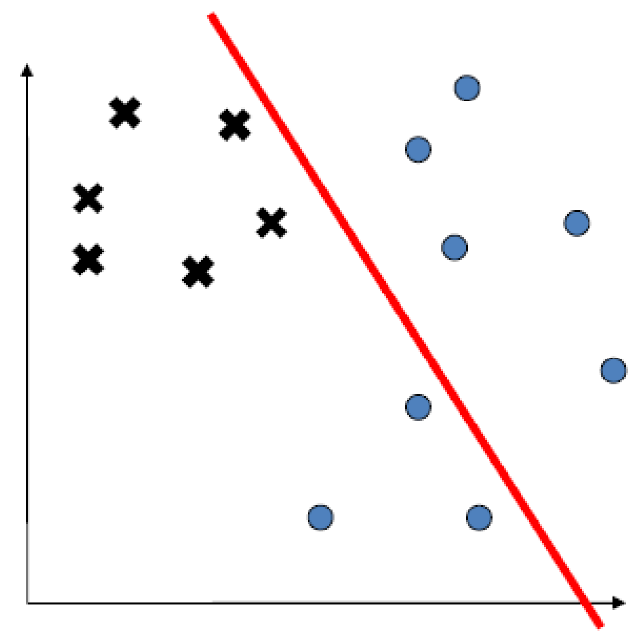
$$\begin{cases} \sum_{i=1}^{M} w_i x_i > 0 & output = 1 \\ else & output = -1 \end{cases}$$



Perceptron Training Rule

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

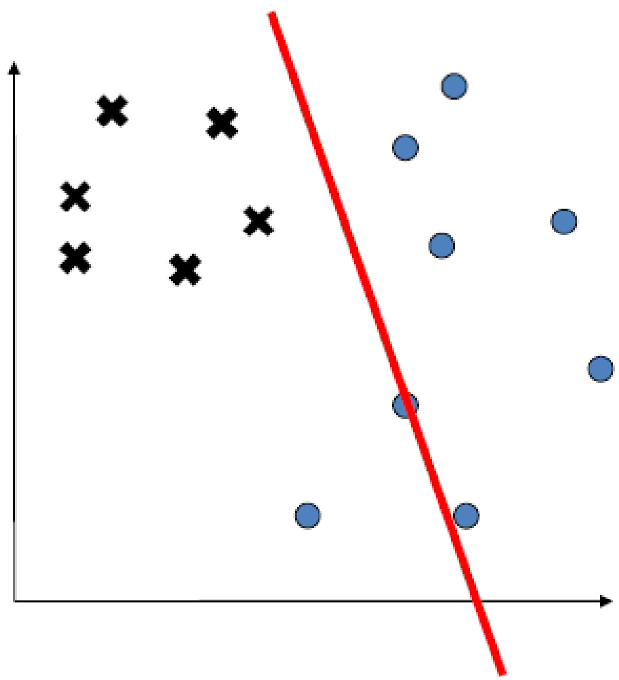
$$w_1 = 1, w_2 = 0.2, w_0 = 0.05$$



Perceptron Training Rule

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

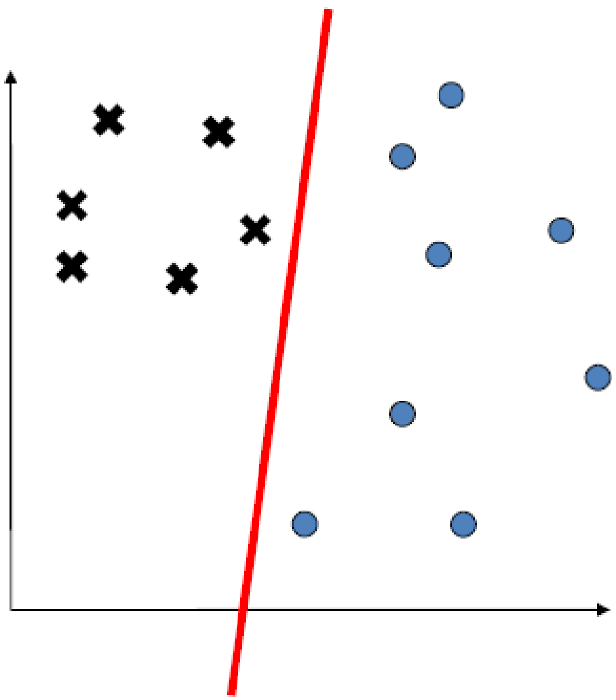
$w_1 = 2.1, w_2 = 0.2, w_0 = 0.05$



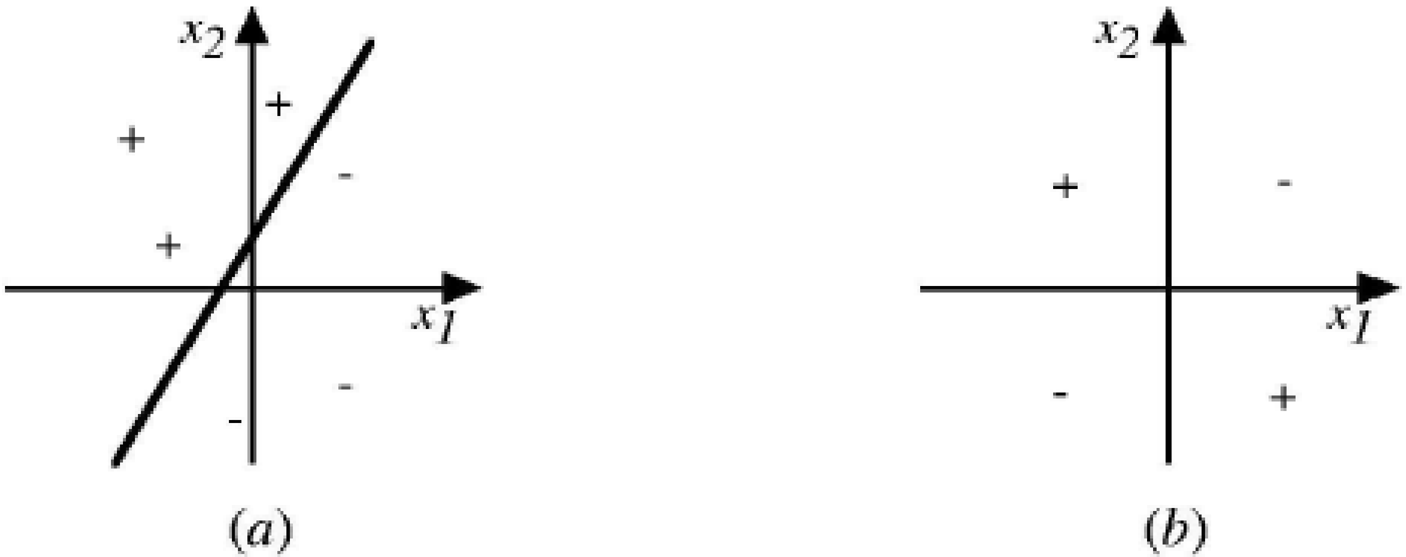
Perceptron Training Rule

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

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



Perceptron



The **decision surface** represented by a **two-input perceptron** $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$.
(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.**