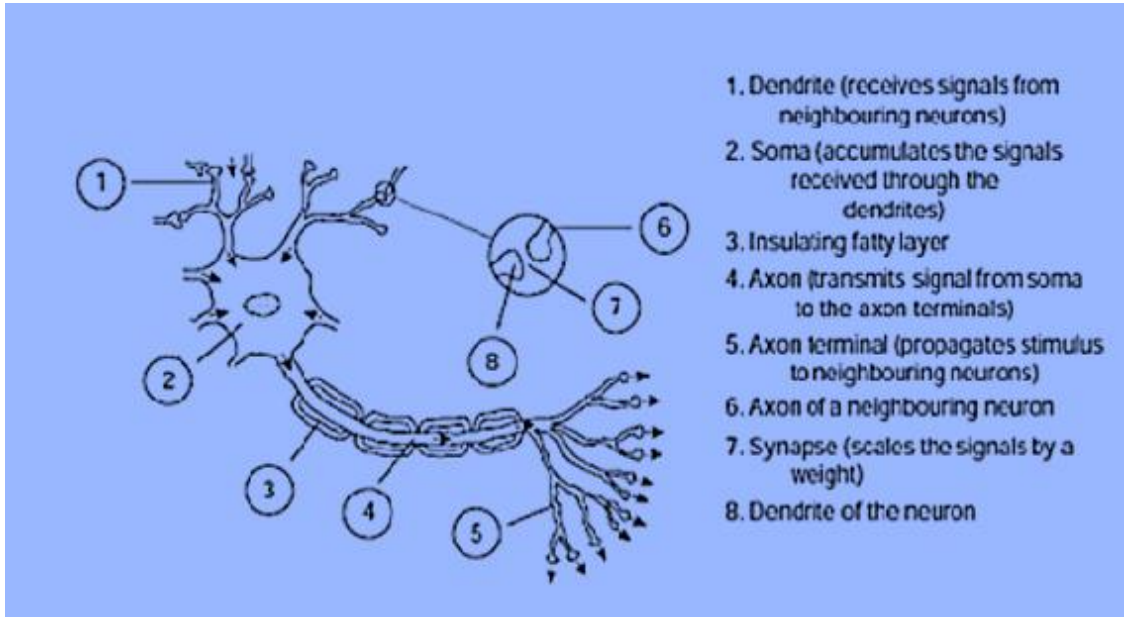
Neural Network

Neural Network: An inspiration from Human Brain

Network of neurons in Human Brain

Structure of a biological neuron



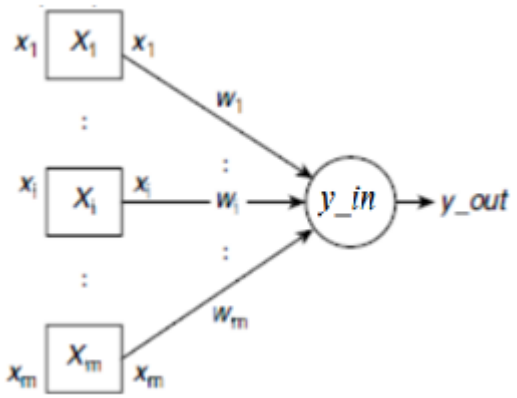
Salient features of a biological neuron

#	Feature
1	The body of the neuron is called the *soma* that acts as a processing element to receive numerous signals through the dendrites simultaneously.
2	The strengths of the incoming signals are modified by the synaptic gaps.
3	The role of the soma, i.e., the processing element of a neuron, is simple. It sums up the weighted input signals and if the sum is sufficiently high, it transmits an output signal through the axon. The output signal reaches the receiving neurons in the neighbourhood through the axon terminals.
4	The weight factors provided by the synaptic gaps are modified over time and experience. This phenomenon, perhaps, accounts for development of skills through practice, or loss of memory due to infrequent recall of stored information.

Artificial Neuron

An artificial neuron is a computational model based on the structure and functionality of a biological neuron.
It consists of a processing output element, a number of inputs and weighted edges connecting each input to the processing element.

Structure of an artificial neuron



x_i The i^{th} input unit.

w_i The weight associated with the interconnection between input unit x_i and the output.

Y_in The total (or net) input to the output

Y_out Signal transmitted by the output. It is known as the activation of Y_in

The net input *Y*_in obtained as

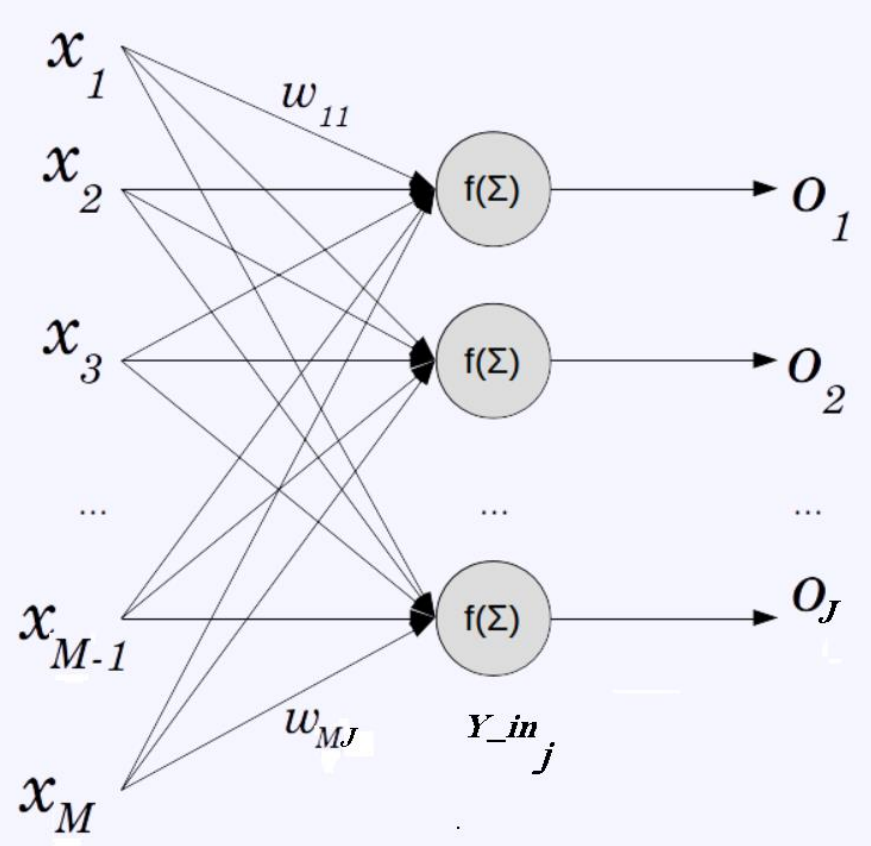
$$y\_in = x_1 w_1 + x_2 w_2 + \cdots + x_m w_m = \sum_{i=1}^{m} x_i w_i$$

The output signal transmitted by Y_in is a function of the net input y_in. Hence

$$y\_out = f(y\_in) = \begin{cases} 1, & if\ y\_in > \theta, \\ 0, & if\ y\_in \le \theta. \end{cases}$$

$$y\_out = f(y\_in) = \begin{cases} 1, & if \sum_{i=1}^{m} x_i w_i > \theta, \\ 0, & if \sum_{i=1}^{m} x_i w_i \le \theta. \end{cases}$$

Structure of a single-layer feed forward ANN



$$y\_in_j = x_1 w_{1j} + x_2 w_{2j} + \cdots + x_{mj} w_{mj} = \sum_{i=1}^{m} x_i w_{mj}$$

In general, the net input y_in_j to the output unit Y_j is given by

$$y\_in_j = [x_1 \ldots x_m] \times \begin{bmatrix} w_{1j} \\ \vdots \\ w_{mj} \end{bmatrix} = X . W^T$$

$$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1j} \\ w_{21} & w_{22} & \cdots & w_{2j} \\ \vdots & \vdots & \cdots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mj} \end{bmatrix}$$

Pattern Classification

Classification is the process of identifying the class to which a given pattern belongs. For example, let us consider the set S of all 3-bit patterns. We may divide the patterns of S into two classes A and B where A is the class of all patterns having more 0s than 1s and B the converse. Therefore

S = {000, 001, 010, 011, 100, 101, 110, 111}

A = {000, 001, 010, 100}

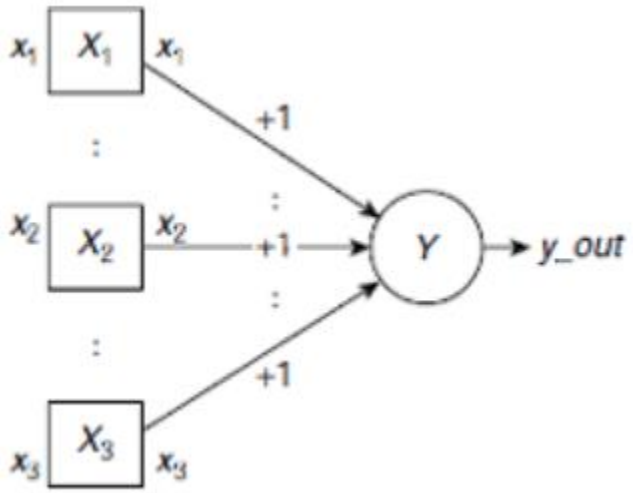
B = {011, 101, 110, 111}

Row #	Pattern			Class
0	0	0	0	*A*
1	0	0	1	*A*
2	0	1	0	*A*
3	0	1	1	*B*
4	1	0	0	*A*
5	1	0	1	*B*
6	1	1	0	B
7	1	1	1	*B*

```
Procedure Classify (x, A, B)
Begin
  n0 = n1 = 0;  /* initialize
  counts */
  /* count 0s and 1s in x */
  For i ← 1 to 3 do
    If the i^th bit is 0 Then
      n0 ++;  Else n1 ++;
    End-if
  End-for
  If n0 > n1  Then Return A;
          Else Return B;
  End-if
End-procedure
```

Solving using O(n) time complexity.

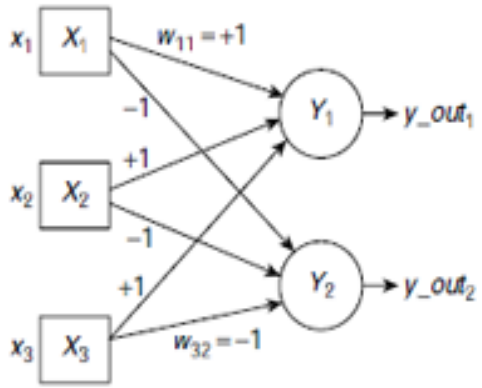
Pattern Classification



$$y\_in = \sum_{i=1}^{3} x_i w_i = x_1 w_1 + x_2 w_2 + x_3 w_3$$
$$= x_1 + x_2 + x_3$$
$$y\_out = f(y\_in) = \begin{cases} 1, & \text{if } y\_in \geq 2, \\ 0, & \text{otherwise.} \end{cases}$$

x1	x2	x3	y_in	y_out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	2	1
1	0	0	1	0
1	0	1	2	1
1	1	0	2	1
1	1	1	3	1

Pattern Classification



$$y\_in_1 = \sum_{i=1}^{3} x_i w_{i1} = x_1 w_{11} + x_2 w_{21} + x_3 w_{31}$$

$$= x_1 + x_2 + x_3$$

$$y\_out_1 = f(y\_in_1) = \begin{cases} 1, & \text{if } y\_in_1 \geq 2, \\ 0, & \text{otherwise.} \end{cases}$$

$$y\_in_2 = \sum_{i=1}^{3} x_i w_{i2} = x_1 w_{12} + x_2 w_{22} + x_3 w_{32}$$

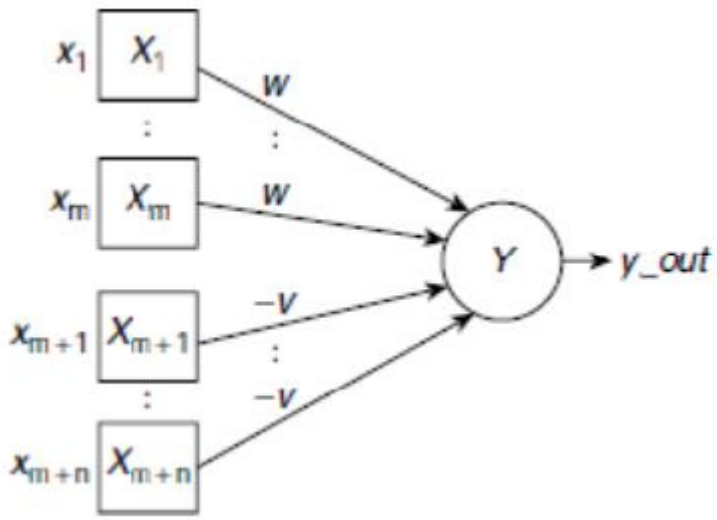
$$= -x_1 - x_2 - x_3$$

$$y\_out_2 = f(y\_in_2) = \begin{cases} 1, & \text{if } y\_in_2 \geq -1, \\ 0, & \text{otherwise.} \end{cases}$$

x1	x2	x3	y_in1	y_in2	y_out1(y_in1>=2 1 else 0)	y_out2(yin2>=-1 1 else 0)
0	0	0	0	0	0	1
0	0	1	1	1	0	1
0	1	0	1	-1	0	1
0	1	1	2	-2	1	0
1	0	0	1	-1	0	1
1	0	1	2	-2	1	0
1	1	0	2	-2	1	0
1	1	1	3	-3	1	0

THE MCCULLOCH–PITTS NEURAL MODEL

Structure of a McCulloch-Pitts neuron



1	There are two kinds of input units, *excitatory*, and *inhibitory*. the excitatory inputs are shown as inputs $X_1, \ldots, X_m$ and the inhibitory inputs are $X_{m+1}, \ldots, X_{m+n}$. The excitatory inputs are connected to the output unit through positively weighted links. Inhibitory inputs have negative weights on their connecting paths to the output unit.
2	All excitatory weights have the same positive magnitude w and all inhibitory weights have the same negative magnitude $-v$.
3	The activation $y\_out = f(y\_in)$ is binary, i.e., either 1 (in case the neuron fires), or 0 (in case the neuron does not fire).
4	The activation function is a binary step function. It is 1 if the net input y_in is greater than or equal to a given threshold value θ, and 0 otherwise.

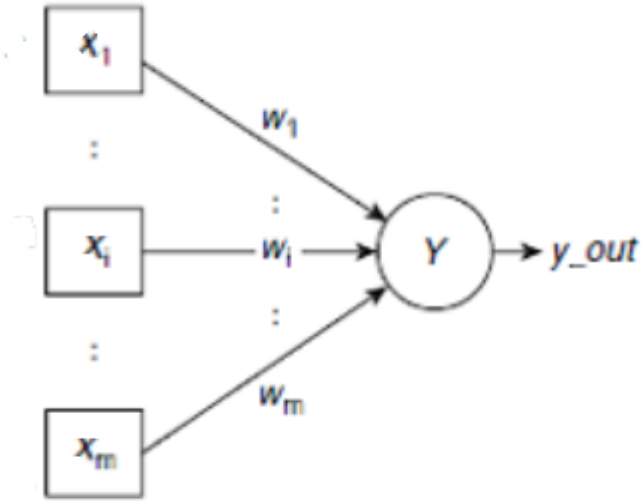
The net input y_in to the neuron Y is given by

$$y\_in = \sum_{i=1}^{m} x_i \times w + \sum_{j=m+1}^{n} x_j \times (-v) = w \sum_{i=1}^{m} x_i - v \sum_{i=m+1}^{n} x_j$$

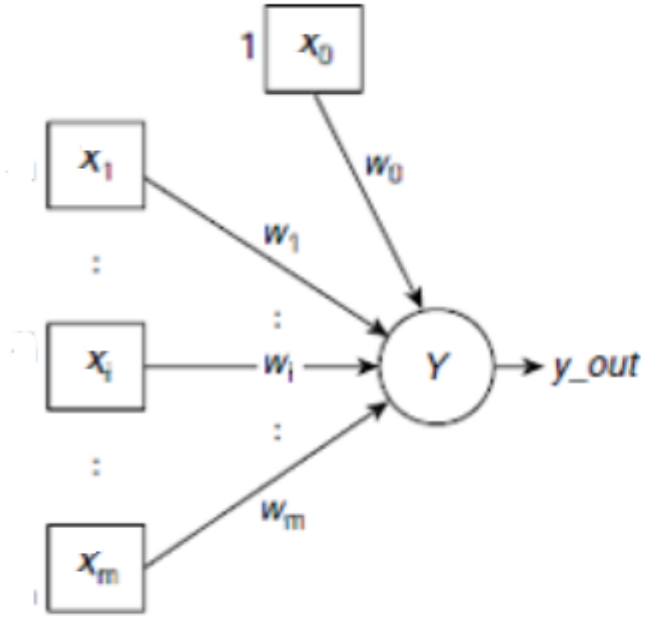
If θ be the threshold value, then the activation of Y, i.e., y_out, is obtained as

$$y\_out = f(y\_in) = \begin{cases} 1, & \text{if } y\_in \geq \theta, \\ 0, & \text{otherwise.} \end{cases}$$

MCCULLOCH–PITTS vs PERCEPTRON



(a) A perceptron without any adjustable threshold



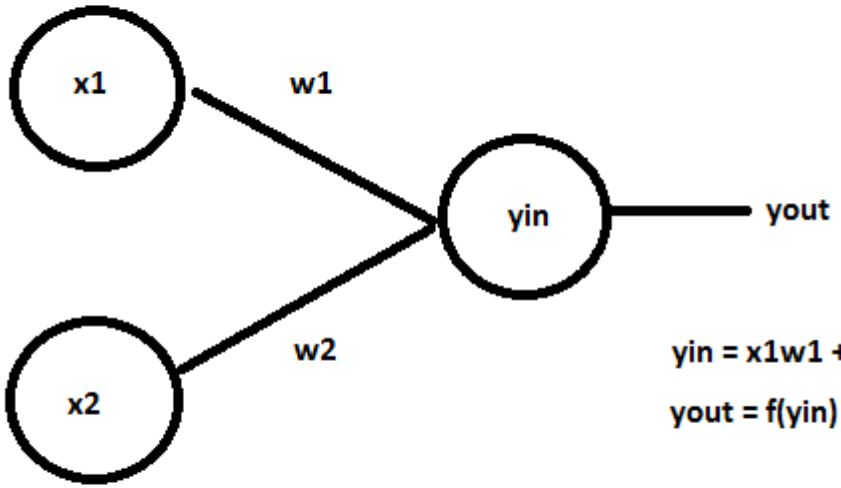
(b) Adjustable threshold as an additional weight

MCCULLOCH–PITTS

Logical OR

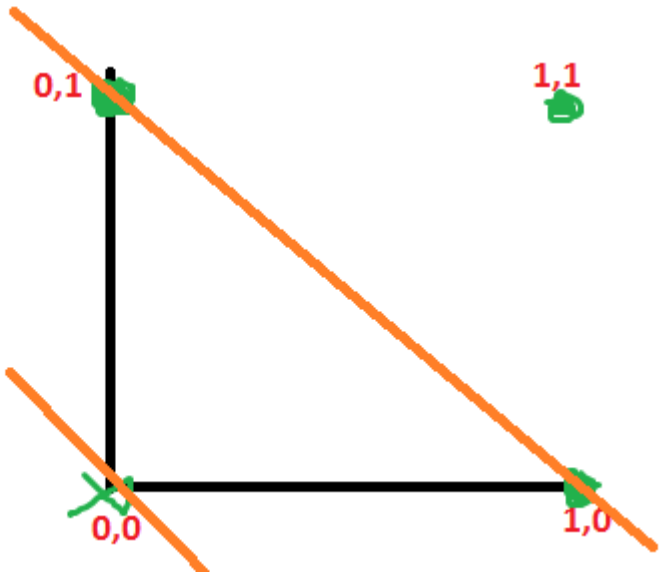
Binary Classification

x1	x2	w1	w2	yin	yout
0	0	1	1	0	0 B
0	1	1	1	1	1 A
1	0	1	1	1	1 A
1	1	1	1	2	1 A



yin = x1w1 + x2w2 = x1+x2

yout = f(yin) = 1 yin > =1
0 yin < 1

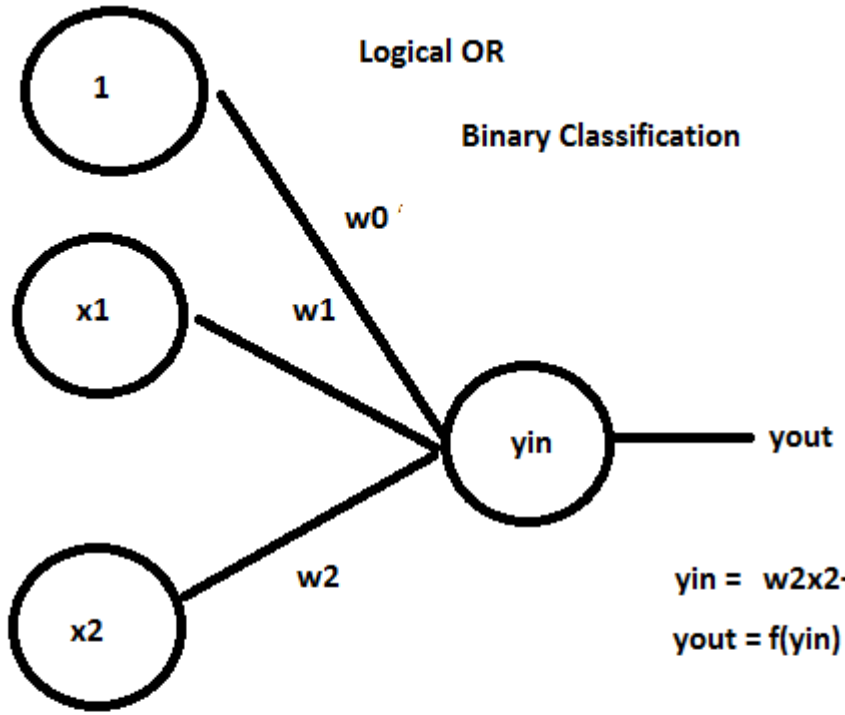


ax+by+c=0

x1+x2=1

x1+x2-1=0

PERCEPTRON



Logical OR

Binary Classification

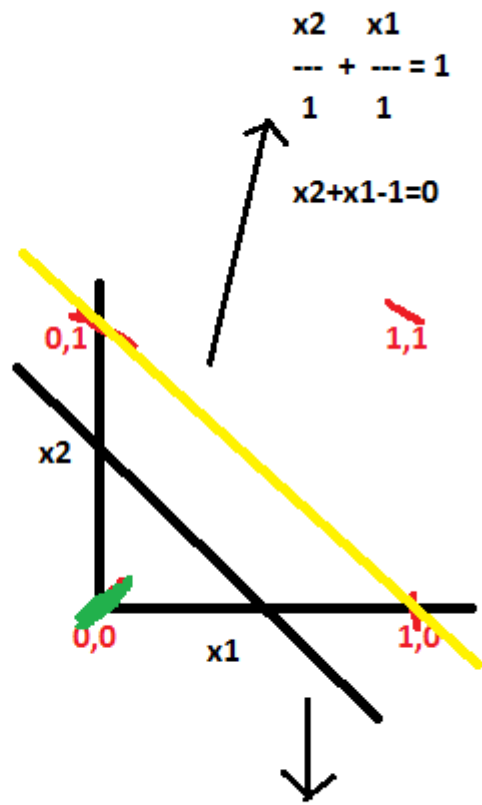
x0	x1	x2	w0	w1	w2	yin	yout'
1	0	0	-1	1	1	-1	0
1	0	1	-1	1	1	0	1
1	1	0	-1	1	1	0	1
1	1	1	-1	1	1	1	1

$yin = w2x2+w1x1+w0x0 = x2+x1-1$

$yout = f(yin) = 1 \quad yin >= 0$

$0 \quad yin < 0$

Linear Separability



$ax+by+c = 0$

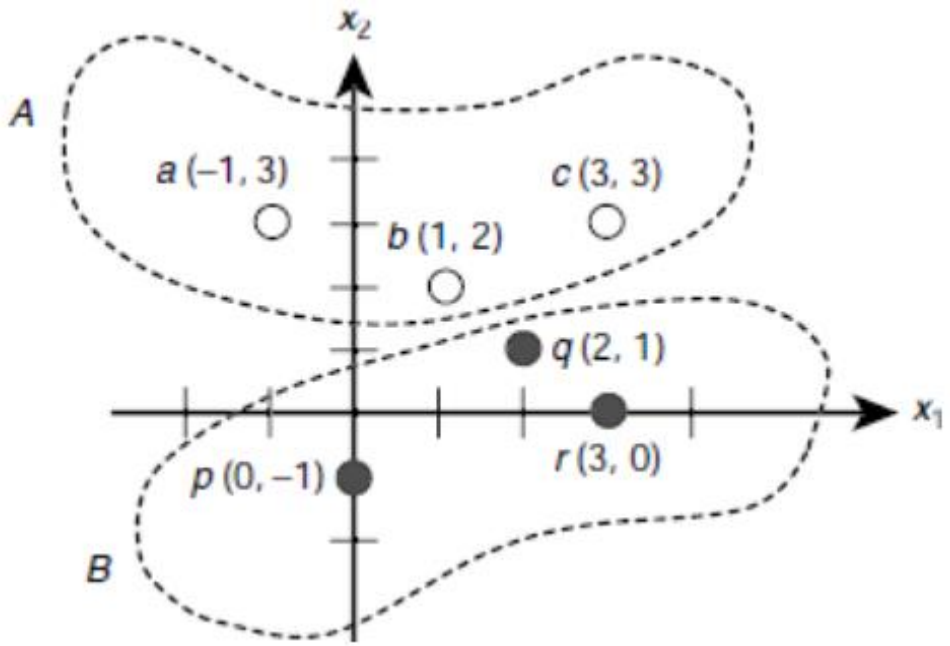
$w2x2+w1x1+w0x0=0$

$w2x2+w1x1+w0 = 0$

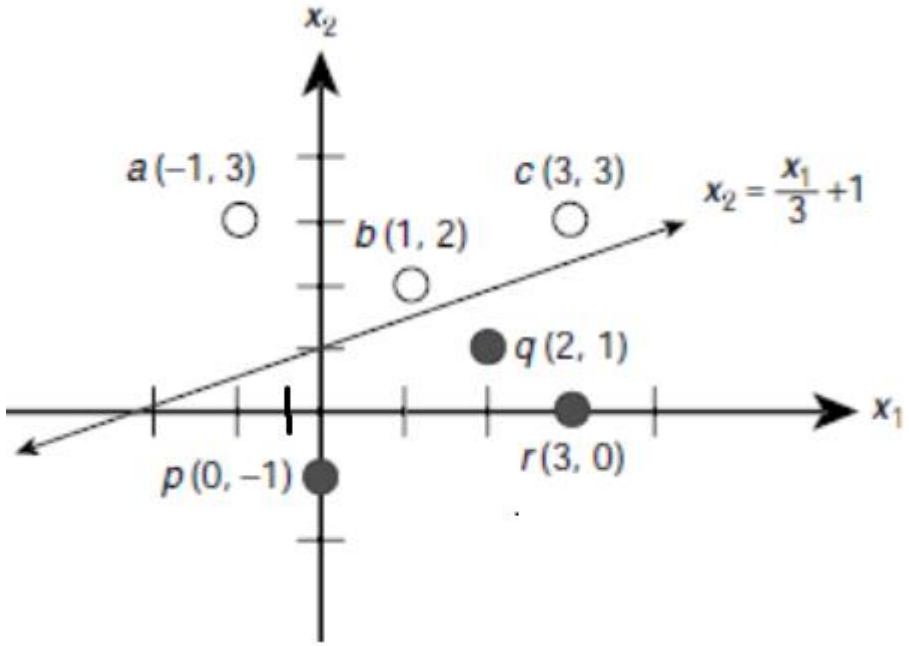
Logical AND ????

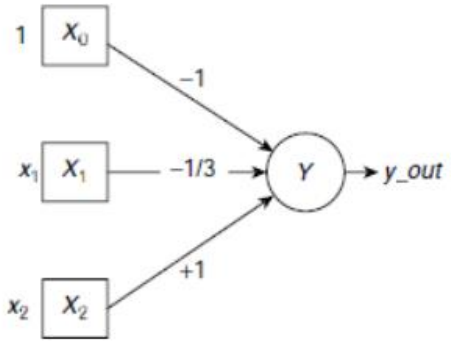
Pattern Classification

Let us consider, for example, two sets of points on a two-dimensional Cartesian plane A = {a, b, c} ={(–1, 3), (1, 2), (3, 3)}, and B = {p, q, r} = {(0, –1), (2, 1), (3, 0)}.



Pattern Classification



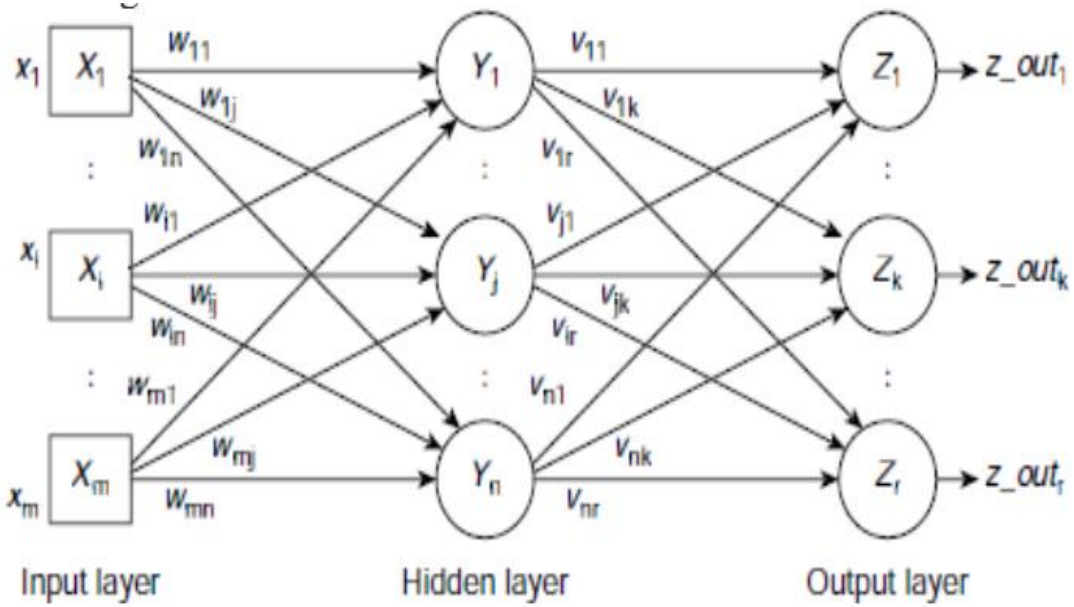


$$y\_in = \sum_{i=0}^{2} x_i w_i = -1 - \frac{x_1}{3} + x_2$$

$$y\_out = \begin{cases} 1, & \text{if } y\_in > 0, \\ 0, & \text{otherwise} \end{cases}$$

x0	x1	x2	w0	w1	w2	yin	yout
1	-1	3	-1	-1/3	1	7/3	A
1	1	2	-1	-1/3	1	2/3	A
1	3	3	-1	-1/3	1	1	A
1	0	-1	-1	-1/3	1	-2	B
1	2	1	-1	-1/3	1	-2/3	B
1	3	0	-1	-1/3	1	-2	B

Multilayer Feed Forward ANNs



The expressions for the net inputs to the hidden layer units and the output units are obtained as

$$Y\_in = X \cdot W^T$$

$$Z\_in = Y\_out \cdot V^T$$

where

$X = [x_1, x_2, \ldots, x_m]$, X is the input vector,

$Y\_in = [y\_in_1, y\_in_2, \ldots, y\_in_n]$, is the net input vector to the hidden layer,

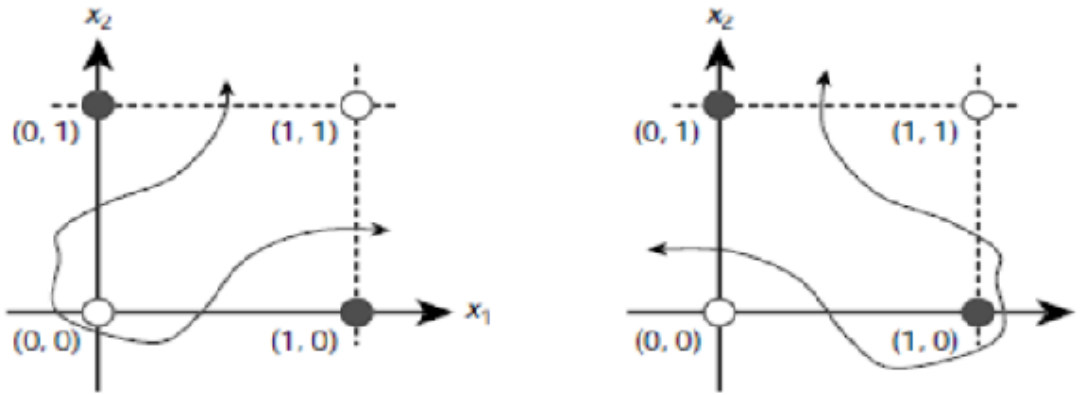
$Z\_in = [z\_in_1, z\_in_2, \ldots, z\_in_r]$, is the net input vector to the output layer,

$Y\_out = [y\_out_1, y\_out_2, \ldots, y\_out_n]$, is the output vector from the hidden layer,

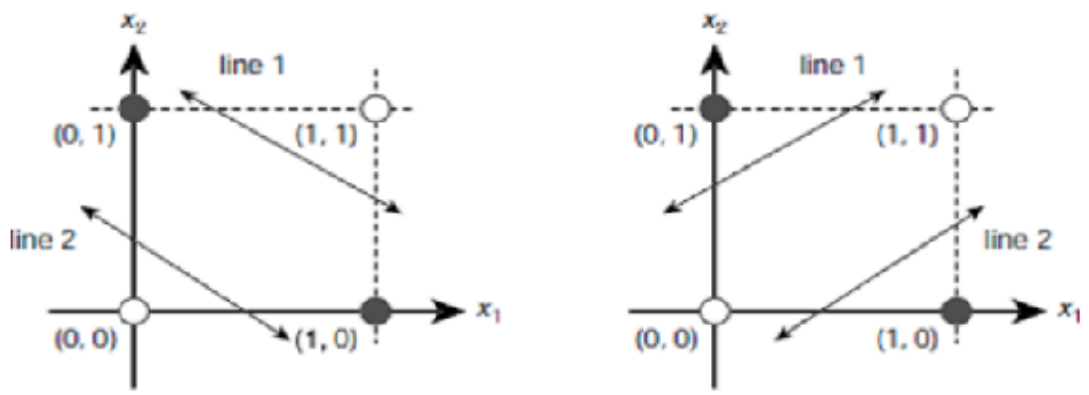
W and V are the weight matrices for the interconnections between the input layer, hidden layer, and output layer respectively.

$$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{bmatrix} \text{ and } V = \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1r} \\ v_{21} & v_{22} & \cdots & v_{2r} \\ \vdots & \vdots & \cdots & \vdots \\ v_{n1} & v_{n2} & \cdots & v_{nr} \end{bmatrix}$$

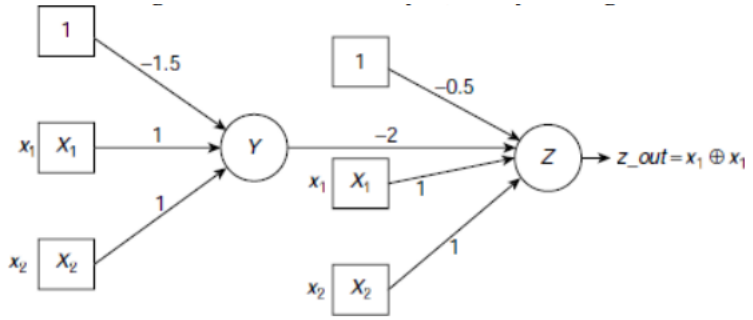
Non Linear Separability



Solving the XOR problem with curved decision surface



Solving the XOR problem with two decision lines

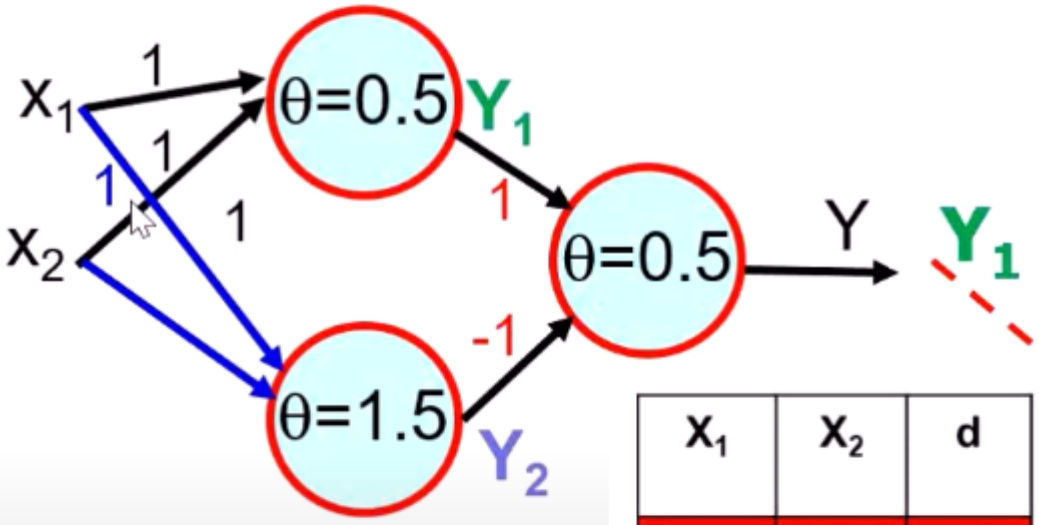


Zin = x0*v0 + x1*v1 + x2*v2 + Yout*wv

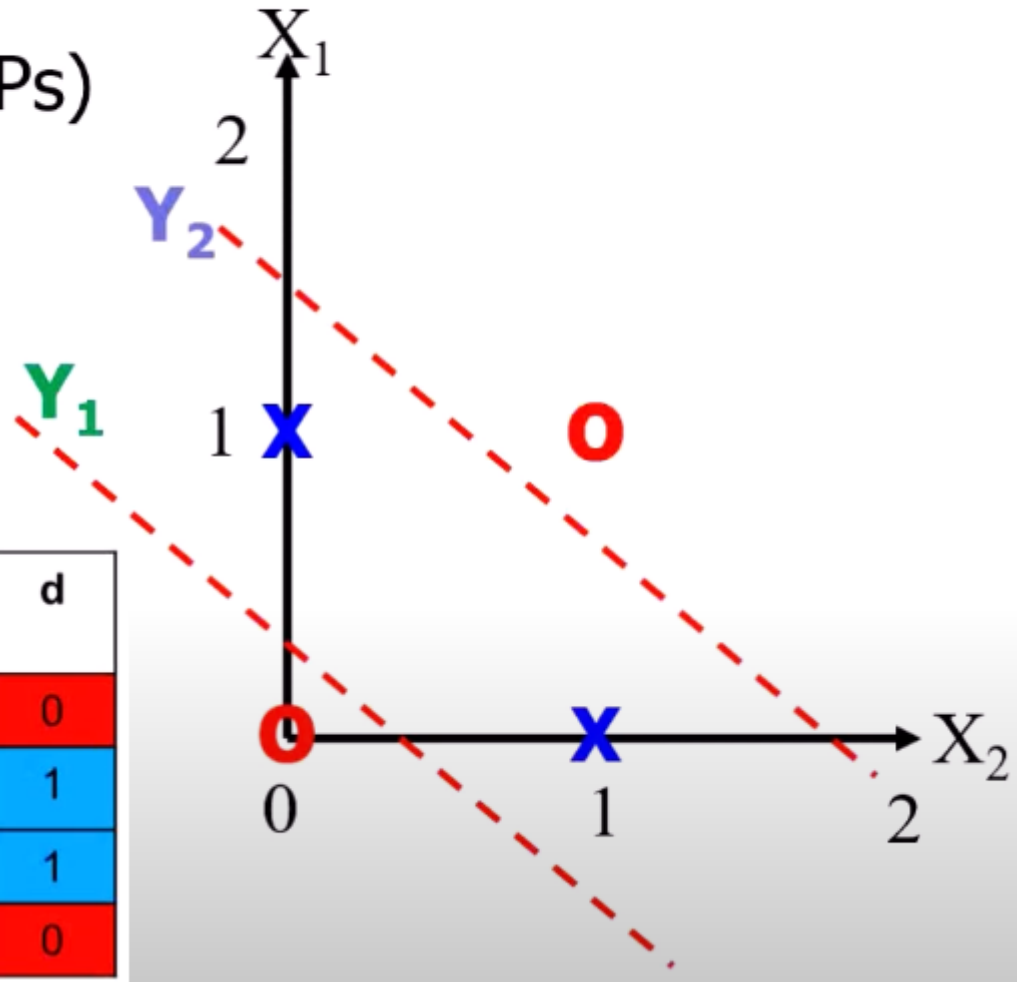
x0	x1	x2	w0	w1	w2	Yin	Yout	wv	v0	v1	v2	Zin	Zout
1	0	0	-1.5	1	1	-1.5	0	-2	-0.5	1	1	-0.5	0
1	0	1	-1.5	1	1	-0.5	0	-2	-0.5	1	1	+0.5	1
1	1	0	-1.5	1	1	-0.5	0	-2	-0.5	1	1	+0.5	1
1	1	1	-1.5	1	1	+0.5	1	-2	-0.5	1	1	-0.5	0

XOR problem revisited

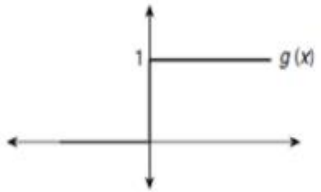
Multi-Layer Perceptrons (MLPs)



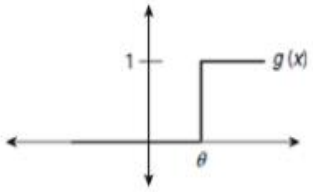
X_1	X_2	d
0	0	0
0	1	1
1	0	1
1	1	0



Activation Function



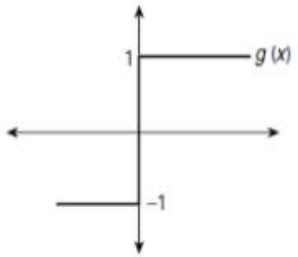
(a) Basic step function



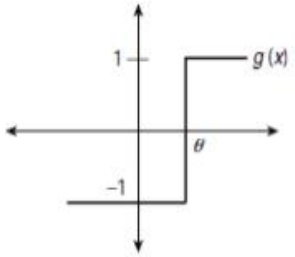
(b) Threshold function function

$$g(x)=\begin{cases}1, & \text{if } x>0,\\ 0, & \text{otherwise.}\end{cases}$$

$$g(x)=\begin{cases}1, & \text{if } x>\theta,\\ 0, & \text{otherwise.}\end{cases}$$



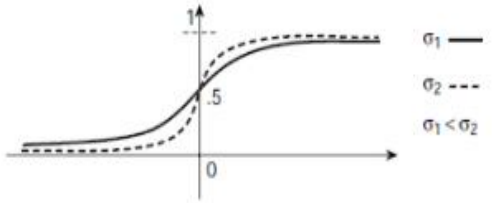
(c) Bipolar step function function



(d) Bipolar threshold function function

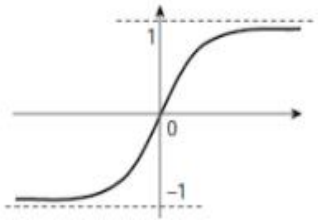
$$g(x)=\begin{cases}1, & \text{if } x>0,\\ -1, & \text{otherwise.}\end{cases}$$

$$g(x)=\begin{cases}+1, & \text{if } x>\theta,\\ -1, & \text{otherwise.}\end{cases}$$



Binary Sigmoid function

$$g(x)=\frac{1}{1+e^{-\sigma x}}$$

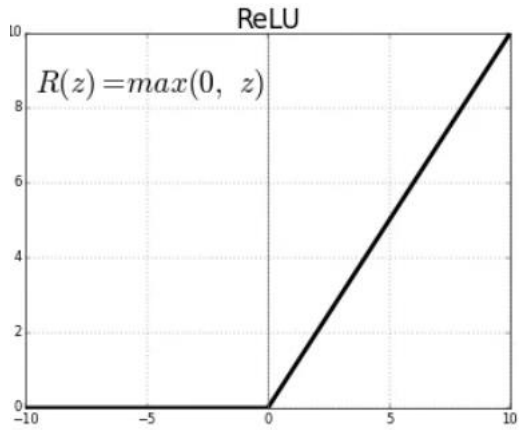


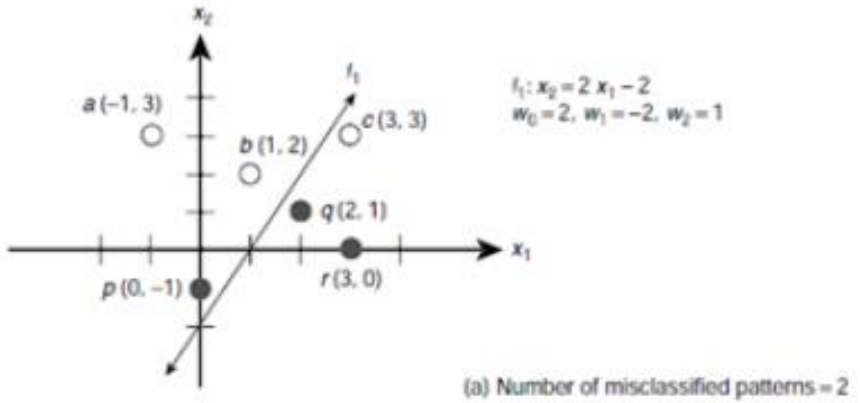
Bipolar Sigmoid function

$$g(x)=\frac{1-e^{-\sigma x}}{1+e^{-\sigma x}}$$

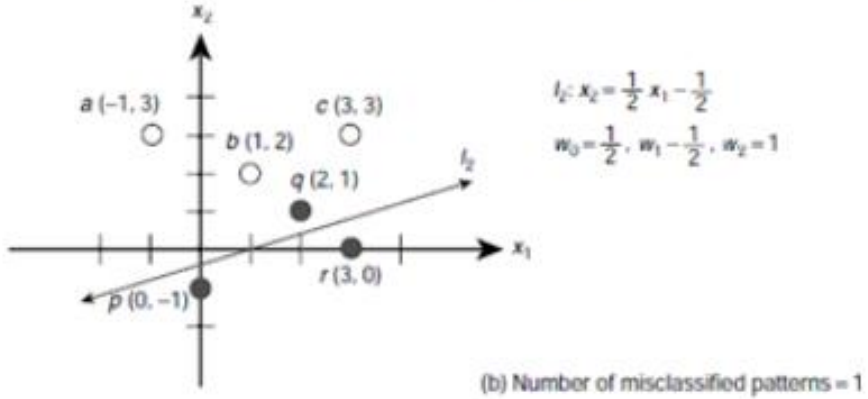
ReLU (Rectified Linear Unit) Activation Function

R(z) is zero when z is less than zero and R(z) is equal to z when z is above or equal to zero.

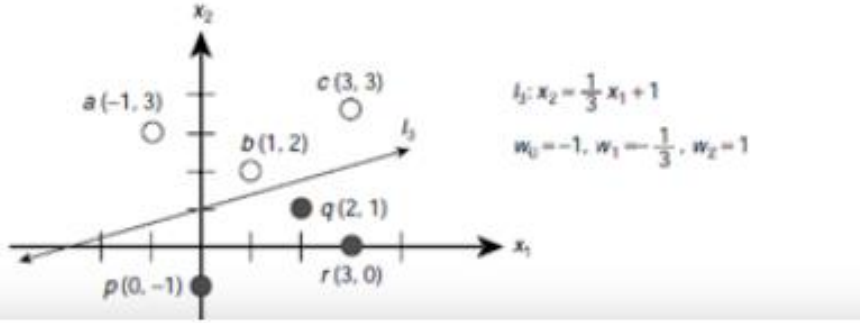




(a) Number of misclassified patterns = 2



(b) Number of misclassified patterns = 1



Learning

- Hebb Net
- Perceptron Learning Rule
- Delta / LMS (Least Mean Square)
- Winner-takes-all

#	Learning Rule	Formula for Δw_i
1	Hebb	$\Delta w_i = x_i \times t$
2	Perceptron	$\Delta w_i = \eta \times (t - y\_out) \times x_i$,
3	Delta/LMS/Widrow-Hoff	$\Delta w_i = \eta \times (t - y\_in) \times x_i$
4	Winner-takes-all	$\Delta w_{ij} = \eta \times [x_i - w_{ij}(old)]$

Hebb Rule

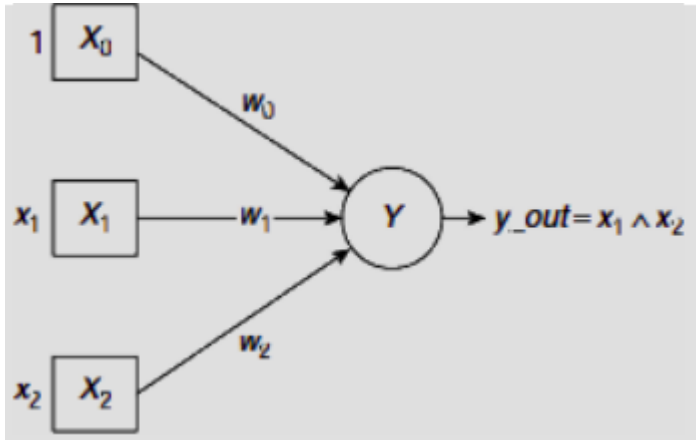
The Hebb rule is one of the earliest learning rules for ANNs.
According to this rule the weight adjustment is computed

```
HebbNet
[Δ w(k)= x(k) * t ]   t=target output [Bipolar input & target]

w(k+1) = w(k) + Δw(k) [calculate new weight]
Δw(k)  = x(k) * t     [adjustment of weight]
```

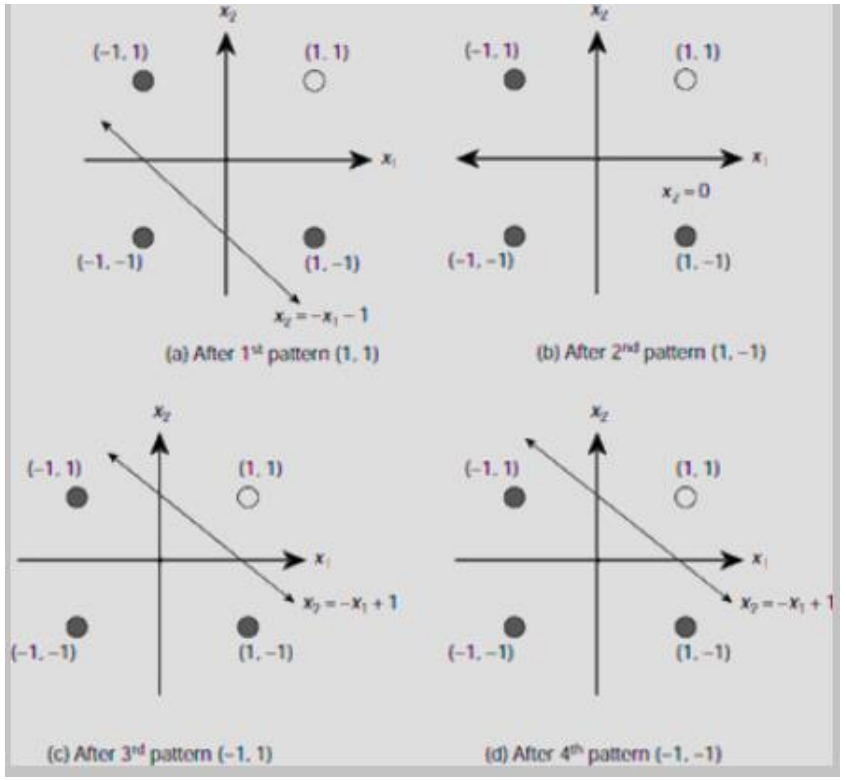
Learning the logical AND function using Hebb Rule

Input Patterns			Output
x_0	x_1	x_2	t
+1	1	1	1
+1	1	−1	−1
+1	−1	1	−1
+1	−1	−1	−1



Logical AND

x0	x1	x2	t	Δw0	Δw1	Δw2	w0	w1	w2	
							0	0	0	
1	1	1	1	1	1	1	1	1	1	
1	1	-1	-1	-1	-1	1	0	0	2	
1	-1	1	-1	-1	1	-1	-1	1	1	x2+x1-1=0
1	-1	-1	-1	-1	1	1	-2	2	2	x2+x1-1=0



Perceptron Learning Rule

The perceptron learning rule is informally stated as:

```
if t!=Yout
   w(k+1) = w(k) + Δw(k)        [calculate new weight]
   Δw(k)  = η*x(k)*t            [adjustment of weight ] η(Learning rate)=[0,1]
else
   w(k+1)= w(k)
```

$$y\_out = \begin{cases} 1, & if\ y\_in > 0 \\ 0, & if\ y\_in = 0 \\ -1 & if\ y\_in < 1 \end{cases}$$

Learning the logical AND function using

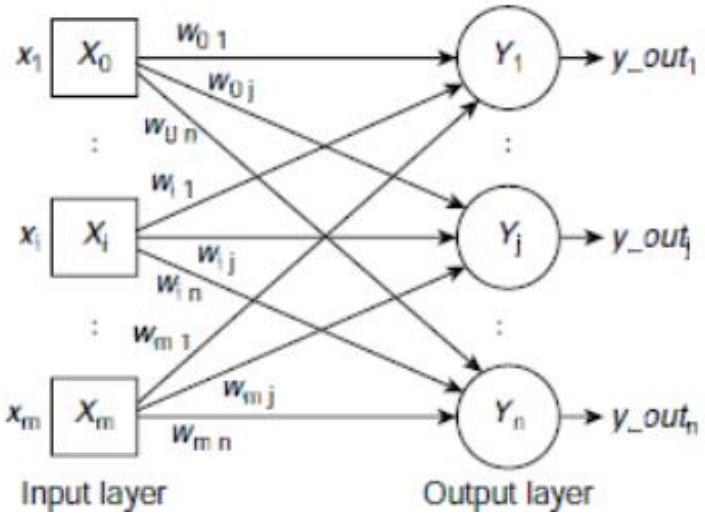
Input Patterns			Output
x_0	x_1	x_2	t
+1	1	1	1
+1	1	−1	−1
+1	−1	1	−1
+1	−1	−1	−1

```
-------------
x0 x1  x2   t  Yin Yout Δw0  Δw1  Δw2  w0  w1  w2    η= 1
                                      0   0   0
1  1   1    1   0   0    1    1    1   1   1   1
1  1  -1   -1   1   1   -1   -1    1   0   0   2
1 -1   1   -1   2   1   -1    1   -1  -1   1   1  x2+x1-1=0
1 -1  -1   -1  -1  -1    0    0    0  -1   1   1
```

There's several variants of this equation, e.g.
Δw(k) = η*x(k)*(t-Yout)

When trying to understand why they are equivalent, notice that the only possible values for y and y' are 1 and -1. So if t=Yout then (t-Yout) is zero and our update rule does nothing. Otherwise, (t-Yout) is equal to either 2 or -2, with the sign matching that of the correct answer (t).
The factor of 2 is irrelevant, because we can tune η to whatever we wish.

Perceptron Learning Rule



The net inputs to the perceptrons are

$$\begin{bmatrix} y\_in_1 \\ \vdots \\ y\_in_j \\ \vdots \\ y\_in_n \end{bmatrix} = \begin{bmatrix} w_{01} & \cdots & w_{i1} & \cdots & w_{m1} \\ \vdots & & \vdots & & \vdots \\ w_{0j} & \cdots & w_{ij} & \cdots & w_{mj} \\ \vdots & & \vdots & & \vdots \\ w_{0n} & \cdots & w_{in} & \cdots & w_{mn} \end{bmatrix} \times \begin{bmatrix} x_0 \\ \vdots \\ x_i \\ \vdots \\ x_m \end{bmatrix}$$

or,

$$Y\_in^{\mathrm{T}} = W^{\mathrm{T}} \times X^{\mathrm{T}}$$

For such an ANN, the adjustment Δw_{ij} of the weight w_{ij} is given by

$$\Delta w_{ij} = \eta \times (t_j - y\_out_j) \times x_i$$

Let us assume the following matrix notations :

$$\Delta W = \begin{bmatrix} \Delta w_{01} & \Delta w_{02} & \cdots & \Delta w_{0n} \\ \Delta w_{11} & \Delta w_{12} & \cdots & \Delta w_{1n} \\ \vdots & \vdots & \vdots & \vdots \\ \Delta w_{m1} & \Delta w_{m2} & \cdots & \Delta w_{mn} \end{bmatrix}, \quad T = \begin{bmatrix} t_1 & \cdots & t_n \end{bmatrix},$$

$$X = \begin{bmatrix} x_0, & \ldots, & x_m \end{bmatrix}, \text{and } Y\_out = \begin{bmatrix} y\_out_1, & \ldots, & y\_out_n \end{bmatrix}.$$

Then the expression of the perceptron learning rule for the architecture becomes

$$[\Delta W]^{\mathrm{T}} = \eta \times [T - Y\_out]^{\mathrm{T}} \times \mathrm{X}$$

Adaline using LMS rule

$$w(k+1) = w(k) + \Delta w(k)$$
$$\Delta w(k) = Eta * (t - Yin) * x(k)$$

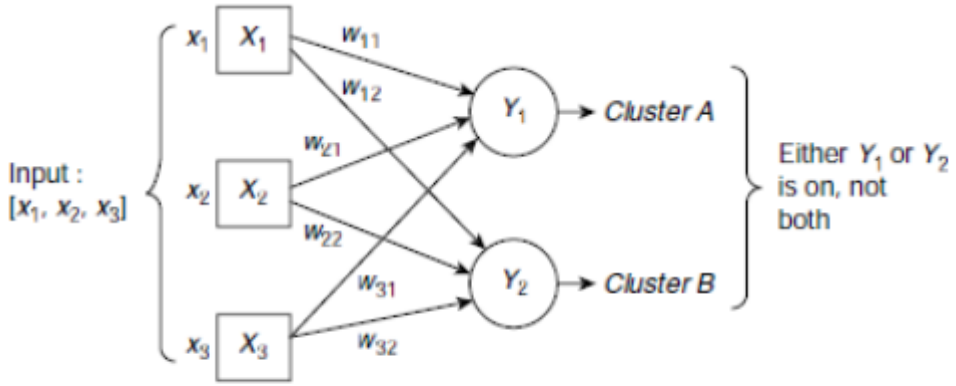
Learning the logical AND function using LMS

#	Input Pattern			Net input	Target output	Error	Weight Adjustments			Weights		
	x_0	x_1	x_2	y_in	t	$t - y\_in$	Δw_0	Δw_1	Δw_2	w_0	w_1	w_2
0										.3	.3	.3
1	1	1	1	.9	1	.1	.02	.02	.02	.32	.32	.32
2	1	1	−1	.32	−1	−1.32	−.264	−.264	.264	.056	.056	.584
3	1	−1	1	.584	−1	−1.584	−.317	.317	−.317	−.261	.373	.267
4	1	−1	−1	−.901	−1	−.099	−.02	.02	.02	−.281	.393	.287

Performance of the net after LMS learning

#	Input Pattern			Net input	Output	Target Output
	x_0	x_1	x_2	$y\_in = \Sigma w_i x_i$	y_out	t
1	1	1	1	.399 > 0	1	1
2	1	1	−1	−.175 < 0	−1	−1
3	1	−1	1	−.387 < 0	−1	−1
4	1	−1	−1	−.961 < 0	−1	−1

Clustering Problem using ANN



3 input 2 output clustering network

1. For each output unit Y_j, $j = 1$ to n, compute the squared Euclidean distance as

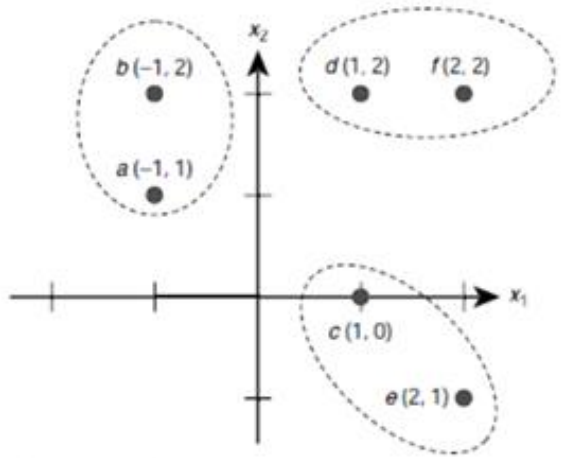
$$D(j) = \sum_{i=1}^{m} (w_{ij} - x_i)^2.$$

m-input n-output clustering network are given

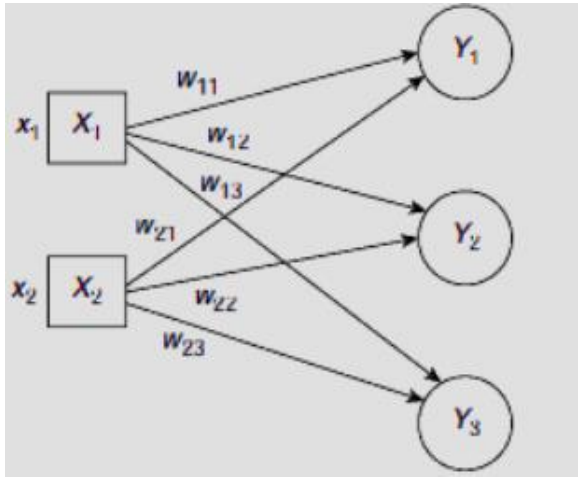
2. Let Y_J be the output unit with the smallest squared Euclidean distance $D(J)$.
3. For all output units within a specified neighbourhood of Y_J, update the weights as follows :

$$w_{ij}(new) = w_{ij}(old) + \eta \times [x_i - w_{ij}(old)]$$

Let us consider a set of six patterns.
Input [S = {a (–1, 1), b (–1, 2), c (1, 0), d (1, 2), e (2, –1), f(2, 2)}]
Output [C1 C2 C3]



A clustering problem showing the expected clusters



the initial distribution of (randomly chosen) weights be

$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} = \begin{bmatrix} 0 & -1 & 2 \\ 2 & 0 & 1 \end{bmatrix}$$

This means that the exemplars, or code vectors, are initially $C_1 = (w_{11}, w_{21}) = (0, 2)$, $C_2 = (w_{12}, w_{22}) = (-1, 0)$, and $C_3 = (w_{13}, w_{23}) = (2, 1)$.

Clusters are formed on the basis of the distances between a pattern and a code vector. For example, to determine the cluster for the pattern a (–1, 1) we need to compute the Euclidean distance between the point a and each code vector C_1, C_2, C_3. The pattern is then clustered with nearest among the three code vectors. Let D_1, D_2, D_3 be the squares of the Euclidean distances between a pattern and C_1, C_2, C_3 respectively. For a (–1, 1) the computations of D_1, D_2, D_3 are as follows.

$$D_1 = (w_{11} - x_1)^2 + (w_{21} - x_2)^2 = (0 + 1)^2 + (2 - 1)^2 = 2$$
$$D_2 = (w_{12} - x_1)^2 + (w_{22} - x_2)^2 = (-1 + 1)^2 + (0 - 1)^2 = 1$$
$$D_3 = (w_{13} - x_1)^2 + (w_{23} - x_2)^2 = (2 + 1)^2 + (1 - 1)^2 = 9$$

Since D_2 is the minimum among these the pattern a (–1, 1) the corresponding exemplar C_2 is declared the winner and the pattern is clustered around the code vector C_2. Similar computations for the pattern b (–1, 2) are given below.

$$D_1 = (w_{11} - x_1)^2 + (w_{21} - x_2)^2 = (0 + 1)^2 + (2 - 2)^2 = 1$$
$$D_2 = (w_{12} - x_1)^2 + (w_{22} - x_2)^2 = (-1 + 1)^2 + (0 - 2)^2 = 4$$
$$D_3 = (w_{13} - x_1)^2 + (w_{23} - x_2)^2 = (2 + 1)^2 + (1 - 2)^2 = 10$$

In this case D_1 is the minimum making C_1 the winner. Table 6.13 summarizes the initial clustering process yielding the clusters $C_1 = \{b, d\}$, $C_2 = \{a\}$ and $C_3 = \{c, e, f\}$ (Fig. 6.43). The minimum distance is indicated by a pair of parentheses.

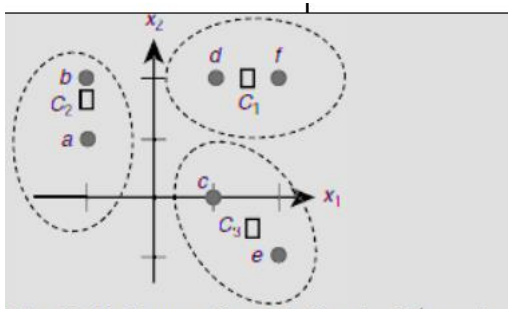


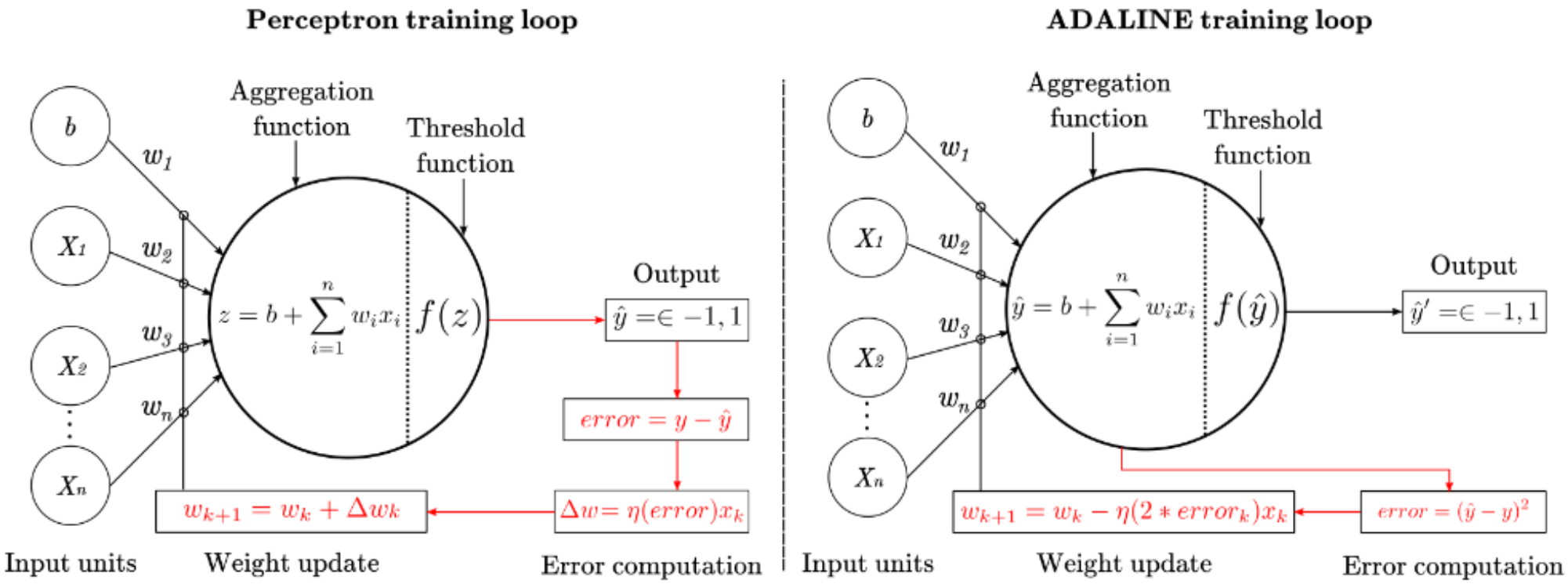
Fig. 6.46. Clusters formed after the 2nd epoch

#	Training Patterns	Squared Euclidean Distance			Winner	New Code Vectors		
		D_1	D_2	D_3		C_1 (w_{11}, w_{21})	C_2 (w_{12}, w_{22})	C_3 (w_{13}, w_{23})
0						(0, 2)	(–1, 0)	(2, 1)
1	a (–1, 1)	2	(1)	9	C_2	(0, 2)	(–1, .5)	(2, 1)
2	b (–1, 2)	(1)	2.25	10	C_1	(–.5, 2)	(–1, .5)	(2, 1)
3	c (1, 0)	6.25	4.25	(2)	C_3	(–.5, 2)	(–1, .5)	(1.5, .5)
4	d (1, 2)	(2.25)	6.25	2.5	C_1	(.25, 2)	(–1, .5)	(1.5, .5)
5	e (2, –1)	12.06	11.25	(2.5)	C_3	(.25, 2)	(–1, .5)	(1.75, –.25)
6	f (2, 2)	(3.06)	11.25	5.13	C_1	(1.13, 2)	(–1, .5)	(1.75, –.25)

1st Epoch

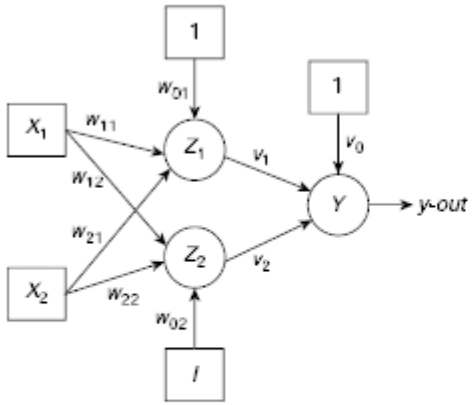
#	Training Patterns	Squared Euclidean Distance			Winner	New Code Vectors		
		D_1	D_2	D_3		C_1 (w_{11}, w_{21})	C_2 (w_{12}, w_{22})	C_3 (w_{13}, w_{23})
0						(1.13, 2)	(–1, .5)	(1.75, –.25)
1	a (–1, 1)	5.54	(.25)	9.13	C_2	(1.13, 2)	(–1, 1.5)	(1.75, –.25)
2	b (–1, 2)	4.54	(.25)	12.63	C_2	(1.13, 2)	(–1, 1.75)	(1.75, –.25)
3	c (1, 0)	4.02	7.06	(.63)	C_3	(1.13, 2)	(–1, 1.75)	(1.38, –.13)
4	d (1, 2)	(.02)	4.06	4.68	C_1	(1.07, 2)	(–1, 1.75)	(1.38, –.13)
5	e (2, –1)	9.86	16.63	(1.14)	C_3	(1.07, 2)	(–1, 1.75)	(1.69, –.57)
6	f (2, 2)	(.86)	9.06	6.7	C_1	(1.56, 2)	(–1, 1.75)	(1.69, –.57)

2nd Epoch


Perceptron training loop
b
w_1
X_1
w_2
X_2
w_3
w_n
X_n
Aggregation function
Threshold function
$z = b + \sum_{i=1}^{n} w_i x_i$
$f(z)$
Output
$\hat{y} = \in -1, 1$
$error = y - \hat{y}$
$\Delta w = \eta(error)x_k$
$w_{k+1} = w_k + \Delta w_k$
Input units
Weight update
Error computation
ADALINE training loop
b
w_1
X_1
w_2
X_2
w_3
w_n
X_n
Aggregation function
Threshold function
$\hat{y} = b + \sum_{i=1}^{n} w_i x_i$
$f(\hat{y})$
Output
$\hat{y}' = \in -1, 1$
$w_{k+1} = w_k - \eta(2 * error_k)x_k$
$error = (\hat{y} - y)^2$
Input units
Weight update
Error computation

MADALINE

Several ADALINEs arranged in a multilayer net is known as Many ADALINES, or Many Adaptive Linear Neurons, or MADALINE in short



two input, one output, one hidden layer with two hidden units MADALINE

In MADALINE (MR-I) algorithm, only the weights of the hidden units are modified during the training and the weights for the inter-connections from the hidden units to the output unit are kept unaltered.

MADALINE MR-I Algorithms

Procedure *MADALINE-MR-I-Learning*

Step 1. Initialize v_0, v_1, v_2 with 0.5 and other weights w_{01}, w_{11}, w_{12}, w_{02}, w_{12} and w_{22} by small random values. All bias inputs are set to 1.

Step 2. Set the learning rate h to a suitable value.

Step 3. For each bipolar training pair $s : t$, do Steps 4-6

Step 4. Activate the input units: $x_1 = s_1$, $x_2 = s_2$, all biases are set to 1 permanently.

Step 5. Propagate the input signals through the net to the output unit Y.

5.1 Compute net inputs to the hidden units.

$$z\_in_1 = 1 \times w_{01} + x_1 \times w_{11} + x_2 \times w_{21}$$

$$z\_in_2 = 1 \times w_{02} + x_1 \times w_{12} + x_2 \times w_{22}$$

5.2 Compute activations of the hidden units z_out_1 and z_out_2 using the bipolar step function

$$z\_out = \begin{cases} 1, & \text{if } z\_in \geq 0 \\ -1, & \text{if } z\_in < 0. \end{cases}$$

5.3 Compute net input to the output unit

$$y\_in = 1 \times v_0 + z\_out_1 \times v_0 + z\_out_2 \times v_2$$

5.4 Find the activation of the output unit y-out using the same activation function as in Step 5.2, *i.e.*,

$$y\_out = \begin{cases} 1, & \text{if } y\_in \geq 0 \\ -1, & \text{if } y\_in < 0. \end{cases}$$

Step 6. Adjust the weights of the hidden units, if required, according to the following rules:

i) If ($y\_out = t$) then the net yields the expected result. Weights need not be updated.

ii) If ($y\_out \neq t$) then apply one of the following rules whichever is applicable.

Case I: $t = 1$

Find the hidden unit z_j whose net input z_in_j is closest to 0. Adjust the weights attached to z_j according to the formula

$$w_{ij}\ (\text{new}) = w_{ij}\ (\text{old}) + h \times (1 - z\_in_j) \times x_i, \text{ for all } i.$$

Case II: $t = -1$

Adjust the weights attached to those hidden units z_j that have positive net input.

$$w_{ij}\ (\text{new}) = w_{ij}\ (\text{old}) + h \times (-1 - z\_in_j) \times x_i, \text{ for all } i.$$

Step 7. Test for stopping condition. It can be any one of the following:

i) No change of weight occurs in Step 6.

ii) The weight adjustments have reached an acceptable level.

iii) A predefined number of iterations have been carried out.

If the stopping condition is satisfied then stop. Otherwise go to Step 3.

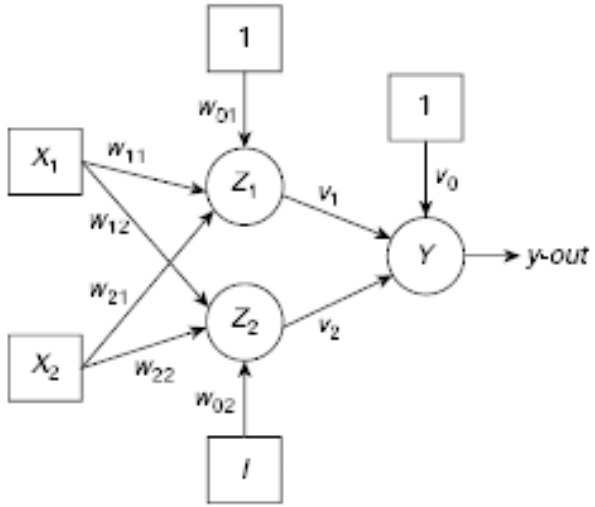
MADALINE training for the XOR function)

Bipolar training set for XOR function

x_0	x_1	x_2	t
1	1	1	−1
1	1	−1	1
1	−1	1	1
1	−1	−1	−1

η = 0.5

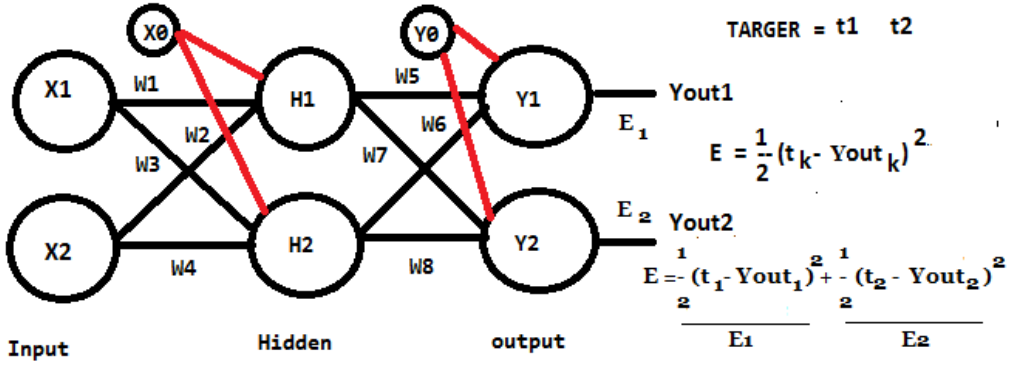
$V_0 = V_1 = V_2 = 0.5$



#	x_0	x_1	x_2	t	z_in_1	z_in_2	z_out_1	z_out_2	y_in	y_out	w_{01}	w_{11}	w_{21}	w_{02}	w_{12}	w_{22}
0											.2	.3	.2	.3	.2	.1
1	1	1	1	−1	.7	.6	1	1	1.5	1	−.65	−.55	−.65	−.5	−.6	−.7
2	1	1	−1	1	−.55	−.4	−1	−1	−.5	−1	−.65	−.55	−.65	.2	.1	−1.4
3	1	−1	1	1	−.75	−1.3	−1	−1	−.5	−1	.23	−1.43	.13	.2	.1	−1.4
4	1	−1	−1	−1	1.56	1.5	1	1	1.5	1	−1.05	−.15	1.41	−1.05	1.35	−.15
Epoch #1																
0											−1.05	−.15	1.41	−1.05	1.35	−.15
1	1	1	1	−1	.21	.15	1	1	1.5	1	−1.66	−.76	.8	−1.63	.77	−.73
2	1	1	−1	1	−3.22	−.13	−1	−1	−.5	−1	−1.66	−.76	.8	−2.07	.33	−.29
3	1	−1	1	1	−.1	−1.45	−1	−1	−.5	−1	−2.11	−.31	.35	−2.07	.33	−.29
4	1	−1	−1	−1	−2.15	−2.11	−1	−1	−.5	−1					-	
Epoch #2																
0											−2.11	−.31	.35	−2.07	.33	−.29
1	1	1	1	−1	−2.07	−2.03	−1	−1	−.5	−1						
2	1	1	−1	1	−2.77	−1.45	−1	−1	−.5	−1	−2.11	−.31	.35	−.84	1.56	−1.52
3	1	−1	1	1	−1.45	−3.92	−1	−1	−.5	−1	−.88	−1.54	.88	−.84	1.56	−1.52
4	1	−1	−1	−1	−.22	−.88	−1	−1	−.5	−1						
Epoch #3																
0											−.88	−1.54	.88	−.84	1.56	−1.52
1	1	1	1	−1	−1.54	−.8	−1	−1	−.5	−1						
2	1	1	−1	1	−3.3	2.24	−1	1	.5	1						
3	1	−1	1	1	1.54	−3.92	1	−1	.5	1						
4	1	−1	−1	−1	−.22	−.88	−1	−1	−.5	−1						
Epoch #4																

THE BACKPROPAGATION ALGORITHM

Backpropagation is a supervised learning method where the net repeatedly adjusts its interconnection weights on the basis of the error, or deviation from the target output, in response to the training patterns.



TARGER = t1 t2

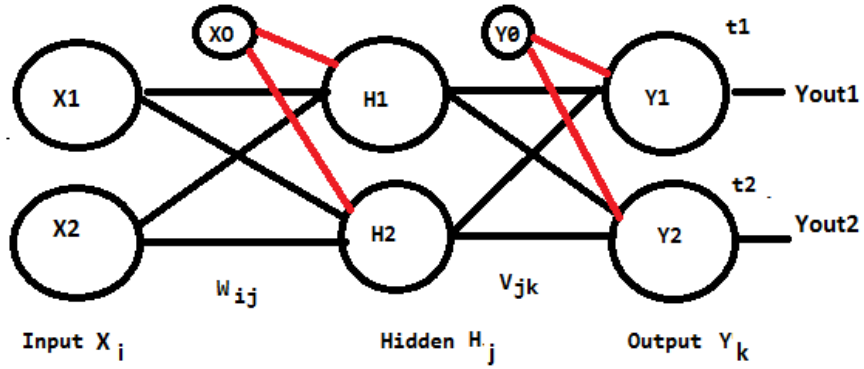
E_1

E_2

$E = \frac{1}{2}(t_k - Yout_k)^2$

$E = \frac{1}{2}(t_1 - Yout_1)^2 + \frac{1}{2}(t_2 - Yout_2)^2$

E_1 E_2



minimize the total error

$$\frac{dE}{dw_5} = \frac{dE}{dYout_1} * \frac{dYout_1}{dy_1} * \frac{dY_1}{dw_5}$$

$$= -(t_1 - Yout_1) * Yout_1(1 - Yout_1) * Hout_1$$

$$Yout_1 = f(Y_1) = \frac{1}{1 + e^{-Y_1}}$$

$$Y_1 = Hout_1 * w_5 + Hout_2 * w_6$$

$$\frac{dE}{dV_{jk}} = -(t_k - Yout_k) * Yout_k(1 - Yout_k) * Hout_j \quad \Delta V_{jk}$$

$$V'_{jk} = V_{jk} + \Delta V_{jk} * \eta$$

$$\frac{dE}{dw_1} = \frac{dE}{dHout_1} * \frac{dHout_1}{dH_1} * \frac{dH_1}{dw_1}$$

$Hout_1(1 - Hout_1)$ X_1

$$\frac{dE}{dHout_1} = \frac{dE_1}{dHout_1} + \frac{dE_2}{dHout_1}$$

$$\frac{dE_1}{dHout_1} = \frac{dE_1}{dY_1} * \frac{dY_1}{dHout_1} \quad (w_5)$$

$$\frac{dE_1}{dY_1} = \frac{dE_1}{dYout_1} * \frac{dYout_1}{dy_1} \quad -(t_1 - Yout_1) \quad (Yout_1(1 - Yout_1))$$

$$= -(t_1 - Yout_1) * Yout_1(1 - Yout_1) * w_5 * Hout_1(1 - Hout_1) * X_1$$

$$= -(t_2 - Yout_2) * Yout_2(1 - Yout_2) * w_6$$

$$\frac{dE}{dHout_j} = \sum_k -(t_k - yout_k) * Yout_k(1 - Yout_k) * V_{jk}$$

$$W'_{ij} = W_{ij} + \Delta W_{ij} * \eta$$

$$\frac{dE}{dW_{ij}} = \sum_k -(t_k - yout_k) * Yout_k(1 - Yout_k) * V_{jk} \; Hout_j(1 - Hout_j) * X_i \quad \Delta W_{ij}$$