Introduction to Neural Network

Artificial Neural Network (ANN) : algorithms based on human mind (how human think and solve problem) used to solve variety type of problem.

Area of Neural Network:

1. Signal processing
2. Pattern recognition
3. Control processing
4. Medicine
5. Speech production
6. Speech recognition
7. Business

Theory of Neural Network

- Neural network is an attempt to model the functionality of the brain in simplify manner (capability of thinking remembering and problem solving of the brain)
- These model attempt to achieve good performance via dense interconnection of simplified computational elements.
- Input paths called dendrites
- If input is strong enough it activate firing of neuron produce output through axon

ANN Network

1. Information processing occur ate many simple element called neurons
2. Signal pass between neurons over connection links
3. Each connection has its own weight
4. Each neuron has activation function (non linear) depend on it input to produce output

Neural Network Characterize by

1. Architecture
2. Training (learning)
3. Activation function

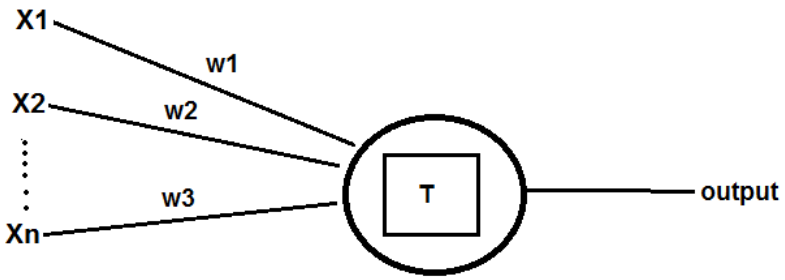
Properties of ANN

1. Parallelism
2. Capacity of adaptation "learning rather than programming"
3. Capacity of generalization

4. No problem definition
5. Abstract and solving problem with noisy
6. Ease of constructing and learning
7. Distributed memory
8. Fault tolerance

McCulloch Pitt Neuron Model

- View neuron start at 1940 (warren McClluch and Walter Pitts
- Network of artificial neurons could c or logical function
- Their model call perceptron 1943
- The figure is



Xi → is input either 0 or 1
Wi → is weight is either +1 or -1
T → is threshold value

Output = 1 if $\sum_{i=1}^{n} Wi\ Xi \geq T$; Output=0 if $\sum_{i=1}^{n} Wi\ Xi < T$

This model is simple but it has substantial computing potential it perform the basic logic operations such as

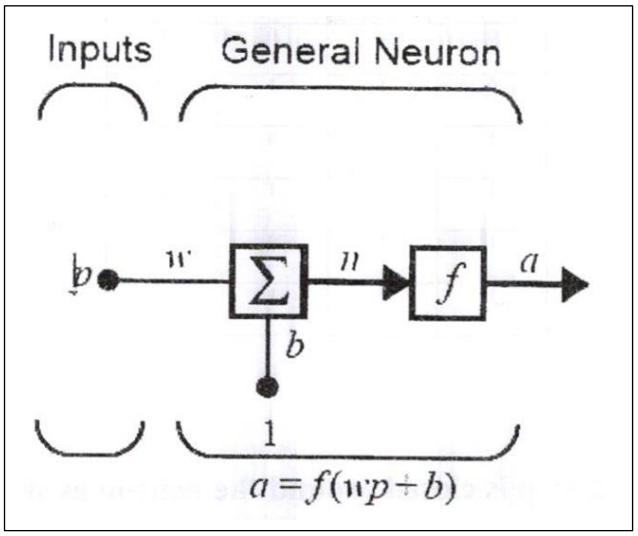
- NOT Gate (inverter)
- OR Gate (3 input)
- AND Gate (3 input)

Neuron Model

Single Input Neurons Model

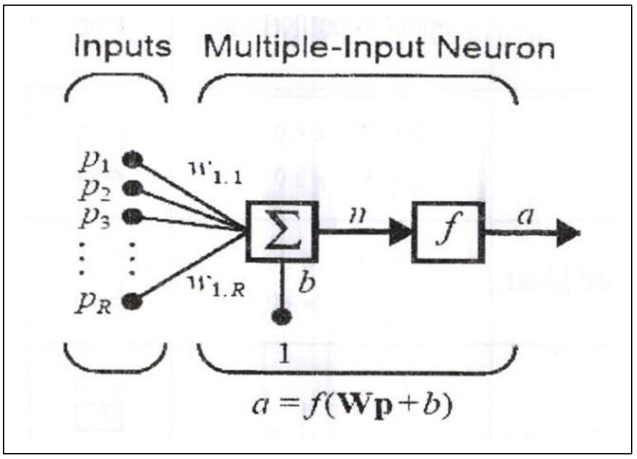
➔ Scalar input p multiply by scalar weight w → produce w*p
➔ Other input 1 multiply by bias b and pass to summer

➔ Then the output go to transfer function (or activation function) (produce scalar neuron output)
➔ F=(wb + b)
➔ w=3 p=2 b=-1.5 f=(3(2)-1.5)=f(4.5)
➔ Bias like neuron except it input is constant 1



Multiple Input Neuron

Neuron could have more than one input (p1, p2, p3, p4,..pr) are each weighted by w11, w12, w13, w14, w1r as in figure below
If there is a bias then
Net = w11 p1+w12 p2+w13 p3+…..w1r pr + b
In matrix for net = wp + b



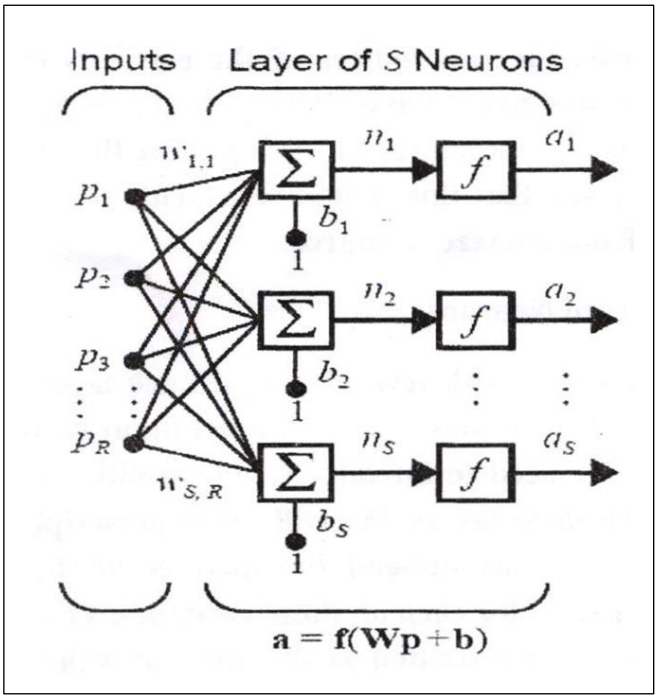
Transfer function

The transfer function in previous figure may be a linear or a nonlinear function of n . A particular transfer function is chosen to satisfy some specification of the problem that the neuron is attempting to solve. Three of the most commonly used functions are discussed below.

Name	Input/Output Relation	Icon	MATLAB Function
Hard Limit	$a = 0 \quad n < 0$ $a = 1 \quad n \geq 0$		hardlim
Symmetrical Hard Limit	$a = -1 \quad n < 0$ $a = +1 \quad n \geq 0$		hardlims
Linear	$a = n$		purelin
Saturating Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n \leq 1$ $a = 1 \quad n > 1$		satlin
Symmetric Saturating Linear	$a = -1 \quad n < -1$ $a = n \quad -1 \leq n \leq 1$ $a = 1 \quad n > 1$		satlins
Log-Sigmoid	$a = \frac{1}{1+e^{-n}}$		logsig
Hyperbolic Tangent Sigmoid	$a = \frac{e^{n}-e^{-n}}{e^{n}+e^{-n}}$		tansig
Positive Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n$		poslin
Competitive	$a = 1$ neuron with max n $a = 0$ all other neurons		compet

Layer of neuron

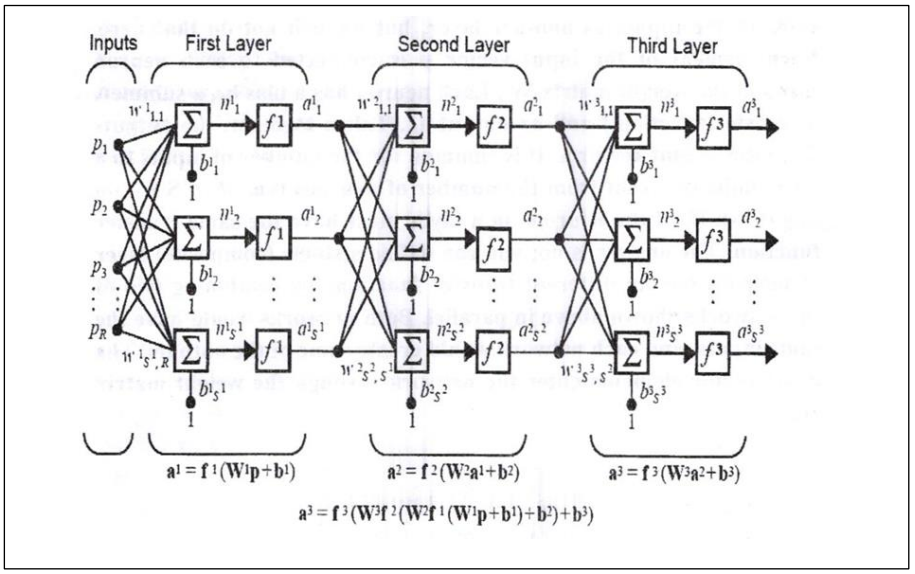
Single layer must has many neuron each has many input



The weight could be represent by matrix notation as show below

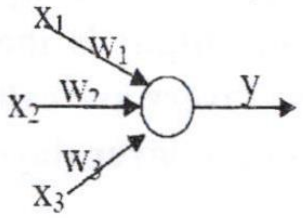
Matrix $W = \begin{matrix} w11 & w12 & w13 \\ w21 & w22 & w23 \\ w31 & w32 & w33 \end{matrix}$

Multiple Layer Neurons



Ex.1 find y for the following neuron if :- $x_1=0.5$, $x_2=1$, $x_3=0.7$

$w_1=0$, $w_2=-0.3$, $w_3=0.6$



Sol

$$net = = X_1W_1 + X_2W_2 + X_3W_3$$

$=0.5*0+1*-0.3+(-0.7*0.6)= -0.72$

1- if f is linear

$y = -0.72$

2- if f is hard limiter (on-off)

$y = -1$

3-if f is sigmoid

$$y = \frac{1}{1+e^{-(-0.72)}} = 0.32$$

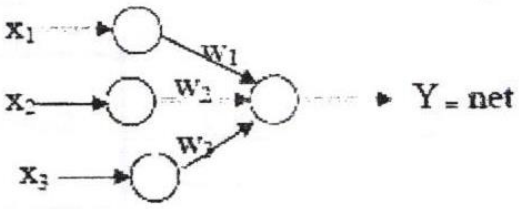
4-if f is tan h

$$y = \frac{e^{-0.72} - e^{0.72}}{e^{-0.72} + e^{+0.72}} = -0.6169$$

5-if f is (TLU) with b=0.6, a=3 then y=-3

Ex.2

The output of a simulated neural using a sigmoid function is 0.5 find the value of threshold when the input $x_1 = 1$, $x_2 = 1.5$, $x_3 = 2.5$. and have initial weights value = 0.2.



Sol

Output = F (net + θ)

$$F(net) = \frac{1}{1+e^{-net}}$$

$$Net = \sum W_i X_i$$

$$= X_1W_1 + X_2W_2 + X_3W_3$$

$$=(1*0.2)+(1.5*0.2)+(2.5*0.2) = 0.2 + 0.30 + 0.50 = 1$$

$$0.5 = \frac{1}{1+e^{-(1+\theta)}}$$

$$0.5\ (1+e^{-(1+\theta)}) = 1$$

$$0.5 + 0.5\ e^{-(1+\theta)} = 1$$

$$0.5\ e^{-(1+\theta)} = 0.5$$

$$e^{-(1+\theta)} = 1$$

$$-(1+\theta) = \ln 1 \Rightarrow -1 - \theta = \theta \Rightarrow -\theta = 1 \Rightarrow \therefore \theta = -1$$

Learning and Adaptation

- Is relatively permanent change in behavior brought about by experience
- By learning rule we mean changing weight

Type of Learning

Supervised learning

- Network output is compared with input in order to compute the error which will be guide the network through changing the weight in order to force the output to close to the target (real data)

Reinforcement

- It is special case of the supervise in which the algorithm give grade to correct the weight in order to make output close to correct target value

Unsupervised Learning

- There is no supervising achieve through the error but it is automatic some thought of classification

Neural network Learning Rule

Hebb learning rule

- Simplest and earlier learning rule is Hebbian learning rule
- He suggest in 1949
- If unit u1 receive an input from unit 2 and both are highly active positive then weight between them should be strength
- If unit u1 receive an input from unit u2 and both are highly active positive then weight between them should be strength otherwise the weight should be decrease
- Unsupervised
- Discrete and continuous transfer function

Example:

Apply Hebbian learning rule on neuron with four inputs. And initial weight $W^1 = [1\ -1\ 0\ 0.5]$, learning constant $c = 1$. The training set is:

$$x_1 = \begin{bmatrix} 1 \\ -2 \\ 1.5 \\ 0 \end{bmatrix}, x_2 = \begin{bmatrix} 1 \\ -0.5 \\ -2 \\ -1.5 \end{bmatrix}, x_3 = \begin{bmatrix} 0 \\ 1 \\ -1 \\ 1.5 \end{bmatrix}$$

Using symmetrical hard limit transfer function:

$$n1=net1=[W^{1t}\, x_1]= \begin{bmatrix} 1 & -1 & 0 & 0.5 \end{bmatrix} \begin{bmatrix} 1 \\ -2 \\ 1.5 \\ 0 \end{bmatrix} = 3$$

$$O1=sgn[3]= 1;\ W^2 = W^1 + c\, O_1 x_1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix} + 1 \times 1 \times \begin{bmatrix} 1 \\ -2 \\ 1.5 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ -3 \\ 1.5 \\ 0.5 \end{bmatrix}$$

$$n2 = net2 = [W^{2t} x_2] = \begin{bmatrix} 2 & -3 & 1.5 & 0.5 \end{bmatrix} \begin{bmatrix} 1 \\ -0.5 \\ -2 \\ -1.5 \end{bmatrix} = -0.25$$

$$O2=sgn\,[-0.25]=-1\ ;\ W^3 = W^2 + c\, O_2 x_2 = \begin{bmatrix} 2 \\ -3 \\ 1.5 \\ 0.5 \end{bmatrix} + 1 \times -1 \times \begin{bmatrix} 1 \\ -0.5 \\ -2 \\ -1.5 \end{bmatrix} = \begin{bmatrix} 1 \\ -2.5 \\ 3.5 \\ 2 \end{bmatrix}$$

$$n3 = net3 = [W^{3t} x_3] = \begin{bmatrix} 1 & -2.5 & 3.5 & 2 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ -1 \\ 1.5 \end{bmatrix} = -3$$

$$O3=sgn\,[-3]=-1\ ;\ W^4 = W^3 + c\, O_3 x_3 = \begin{bmatrix} 1 \\ -2.5 \\ 3.5 \\ 2 \end{bmatrix} + 1 \times -1 \times \begin{bmatrix} 0 \\ 1 \\ -1 \\ 1.5 \end{bmatrix} = \begin{bmatrix} 1 \\ -3.5 \\ 4.5 \\ 0.5 \end{bmatrix}$$

Using log sigmoid transfer function:

$$n1=net1=[W^{1t}x_1]=\begin{bmatrix}1 & -1 & 0 & 0.5\end{bmatrix}\begin{bmatrix}1\\-2\\1.5\\0\end{bmatrix}=3$$

$$O_1=f(n_1)=f(net_1)=\frac{1}{1+e^{-n1}}=\frac{1}{1+e^{-3}}=0.953$$

$$W^2=W^1+c\,O_1x_1=\begin{bmatrix}1\\-1\\0\\0.5\end{bmatrix}+1\times 0.953\times\begin{bmatrix}1\\-2\\1.5\\0\end{bmatrix}=\begin{bmatrix}1.953\\-2.906\\1.4295\\0.5\end{bmatrix}$$

$$n2=net2=[W^{2t}x_2]=\begin{bmatrix}1.953 & -2.906 & 1.4295 & 0.5\end{bmatrix}\begin{bmatrix}1\\-0.5\\-2\\-1.5\end{bmatrix}=-0.203$$

$$O_2=f(n_2)=f(net_2)=\frac{1}{1+e^{-n2}}=\frac{1}{1+e^{0.203}}=0.449$$

$$W^3=W^2+c\,O_2x_2=\begin{bmatrix}1.953\\-2.906\\1.4295\\0.5\end{bmatrix}+1\times 0.449\times\begin{bmatrix}1\\-0.5\\-2\\-1.5\end{bmatrix}=\begin{bmatrix}2.402\\-3.131\\0.5315\\-0.1735\end{bmatrix}$$

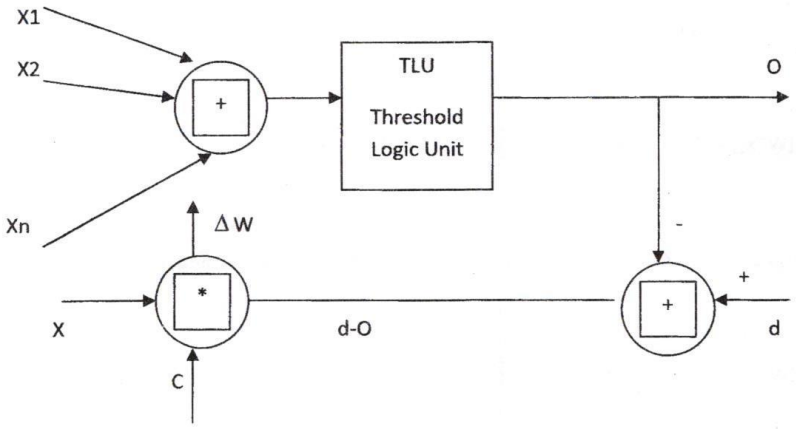
$$n3=net3=[W^{3t}x_3]=\begin{bmatrix}2.402 & -3.131 & 0.5315 & -0.1735\end{bmatrix}\begin{bmatrix}0\\1\\-1\\1.5\end{bmatrix}=-3.923$$

$$O_3=f(n_3)=f(net_3)=\frac{1}{1+e^{-n3}}=\frac{1}{1+e^{3.923}}=0.0194$$

$$W^4 = W^3 + c\, O_3 x_3 = \begin{bmatrix} 2.402 \\ -3.131 \\ 0.5315 \\ -0.1735 \end{bmatrix} + 1 \times 0.0194 \times \begin{bmatrix} 0 \\ 1 \\ -1 \\ 1.5 \end{bmatrix} = \begin{bmatrix} 2.402 \\ -3.112 \\ 0.5121 \\ -0.1444 \end{bmatrix}$$

6.2 Perceptron Learning Rule:

This learning rule is applicable only for neurons with discrete transfer functions, and the weights are adjusted if and only if the output is incorrect. This learning is supervised learning rule as shown in following figure.



At Perceptron learning rule:

$r = d_i - O_i$

Then: $\Delta W = c\,(d_i - O_i)\,X_j$

Example:

Apply Perceptron learning rule on neuron with three inputs. And initial weight:

$W^1 = [1\ -1\ 0\ 0.5]$, learning constant $c = 0.1$. The training sets are:

$$x_1 = \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}; d_1 = -1, x_2 = \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix}; d_2 = -1, x_3 = \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}; d_3 = 1.$$

Using symmetrical hard limit transfer function:

n1=net1=$[W^{1t} x_1]$= $\begin{bmatrix} 1 & -1 & 0 & 0.5 \end{bmatrix} \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} = 2.5$

O1=sgn[2.5]= 1; $d_1 \neq O_1$

$W^2 = W^1 + c\,[d_1 - O_1]\, x_1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix} + 0.1 \times [-1-1] \times \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.8 \\ -0.6 \\ 0 \\ 0.7 \end{bmatrix}$

n2 = net2 = $[W^{2t} x_2]$ = $\begin{bmatrix} 0.8 & -0.6 & 0 & 0.7 \end{bmatrix} \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix} = -1.6$

O2=sgn [-1.6]=-1 ; since $d_2 = O_2 = -1; \therefore d_2 - O_2 = 0; \therefore$ NO need for learning (correction).

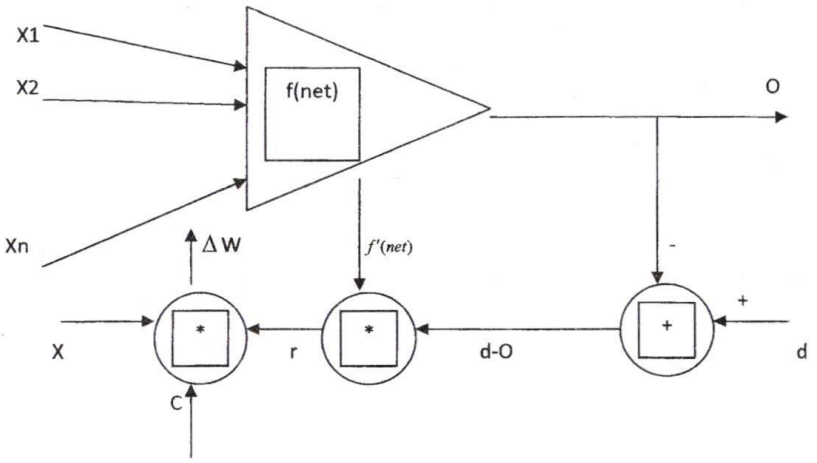
n3 = net3 = $[W^{3t} x_3]$ = $\begin{bmatrix} 0.8 & -0.6 & 0 & 0.7 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix} = -2.1$

O3=sgn [-2.1]=-1 ; $d_3 = 1 \neq O_3$

$W^3 = W^2 + c\,[d3 - O3] x_3 = \begin{bmatrix} 0.8 \\ -0.6 \\ 0 \\ 0.7 \end{bmatrix} + 0.1 \times [1-(-1)] \times \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.6 \\ -0.4 \\ 0.1 \\ 0.5 \end{bmatrix}$

6.3 Delta Learning Rule:

This learning rule is applicable only for neurons with continuous transfer functions. This learning is supervised learning rule as shown in following figure.



At delta learning rule:

$$r = (d_i - f(w_i x)) f'(w_i x)$$

$$f'(w_i x) = \frac{1}{2}(1 - o^2)$$

Then: $\Delta w_i = c(d_i - o_i) f'(net_i) x$

Or: $\Delta w_i = c(d_i - o_i)\frac{1}{2}(1 - o^2) x$

Example:

Apply Delta learning rule on neuron with three inputs. And initial weight: $W^1 = [1\ -1\ 0\ 0.5]$, learning constant c = 0.1. The training sets are:

$$x_1 = \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}; d_1 = -1, x_2 = \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix}; d_2 = -1, x_3 = \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}; d_3 = 1.$$

Using log sigmoid transfer function:

$$n1=net1=[W^{1t}x_1]=\begin{bmatrix}1 & -1 & 0 & 0.5\end{bmatrix}\begin{bmatrix}1\\-2\\0\\-1\end{bmatrix}=2.5$$

$$O_1=f(n_1)=f(net_1)=\frac{1}{1+e^{-n1}}=\frac{1}{1+e^{-2.5}}=0.924$$

$$W^2=W^1+c\,(d_1-O_1)\frac{1}{2}(1-o_1^2)\,x_1=$$

$$\begin{bmatrix}1\\-1\\0\\0.5\end{bmatrix}+0.1\times(-1-0.924)\times 0.5\times(1-(0.924)^2)\times\begin{bmatrix}1\\-2\\0\\-1\end{bmatrix}=\begin{bmatrix}0.986\\-0.972\\0\\0.514\end{bmatrix}$$

$$n2=net2=[W^{2t}x_2]=\begin{bmatrix}0.986 & -0.972 & 0 & 0.514\end{bmatrix}\begin{bmatrix}0\\1.5\\-0.5\\-1\end{bmatrix}=-1.972$$

$$O_2=f(n_2)=f(net_2)=\frac{1}{1+e^{-n2}}=\frac{1}{1+e^{1.972}}=0.122$$

$$W^3=W^2+c\,(d_2-O_2)\frac{1}{2}(1-o_2^2)\,x_2=$$

$$\begin{bmatrix}0.986\\-0.972\\0\\0.514\end{bmatrix}+0.1\times(-1-0.122)\times 0.5\times(1-(0.122)^2)\times\begin{bmatrix}0\\1.5\\-0.5\\-1\end{bmatrix}=\begin{bmatrix}0.986\\-0.889\\-0.028\\0.459\end{bmatrix}$$

$$n3=net3=[W^{3t}x_3]=\begin{bmatrix}0.986 & -0.889 & -0.028 & 0.459\end{bmatrix}\begin{bmatrix}-1\\1\\0.5\\-1\end{bmatrix}=-2.348$$

$$O_3 = f(n_3) = f(net_3) = \frac{1}{1+e^{-n3}} = \frac{1}{1+e^{-2\,348}} = 0.087$$

$W^4 = W^3 + c\,(d_3 - O_3)\frac{1}{2}(1-o_3^2)\,x_3 =$

$$\begin{bmatrix} 0.986 \\ -0.889 \\ -0.028 \\ 0.459 \end{bmatrix} + 0.1 \times (1-0.087) \times 0.5 \times (1-(0.087)^2) \times \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.94 \\ -0.843 \\ -0.005 \\ 0.413 \end{bmatrix}$$

6.4 Widrow-Hoff (Least Mean Square) Learning Rule:

This learning rule is applicable for neurons without transfer functions. This learning is supervised learning rule. This learning rule can be considered a special case of Delta learning rule assuming that:

$$f(w_i^t x) = w_i^t x;\ f'(w_i^t x) = 1$$

Then: $r = d_i - w_i x$

i.e.:

$$\Delta w_i = c(d_i - w_i x)x$$

Example:

Perform two training steps using Widrow-Hoff learning rule? Assume the following training data:

$$x_1 = \begin{bmatrix} 2 \\ 0 \\ -1 \end{bmatrix}; d_1 = -1; x_2 = \begin{bmatrix} 1 \\ -2 \\ -1 \end{bmatrix}; d_2 = 1; w^1 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}; c = 0.25.$$

$$n1 = net1 = [w^{1t} x_1] = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \\ -1 \end{bmatrix} = 1$$

$$w^2 = w^1 + c(d_1 - net_1)x_1 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} + (0.25)(-1-1)\begin{bmatrix} 2 \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1.5 \end{bmatrix}$$

$$n2 = net2 = [w^{2t} x_2] = \begin{bmatrix} 0 & 0 & 1.5 \end{bmatrix}\begin{bmatrix} 1 \\ -2 \\ -1 \end{bmatrix} = -1.5$$

$$w^3 = w^2 + c(d_2 - net_2)x_2 =$$

$$= \begin{bmatrix} 0 \\ 0 \\ 1.5 \end{bmatrix} + (0.25)(1+1.5)\begin{bmatrix} 1 \\ -2 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.625 \\ -1.25 \\ 0.875 \end{bmatrix}$$

Notes:
The performance of learning procedure depend on many factor such as

1. The choice of error function
2. The net architecture.
3. Type of node and possible restrictions on the value of the weight.
4. An activation function

The coverage of the net depend on the :

1. Training set
2. The initial conditions
3. Learning algorithm