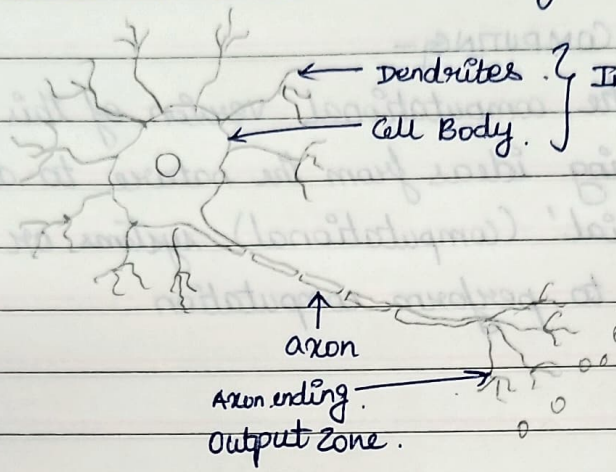
INTRODUCTION TO NATURAL COMPUTING.

NATURE COMPUTING -

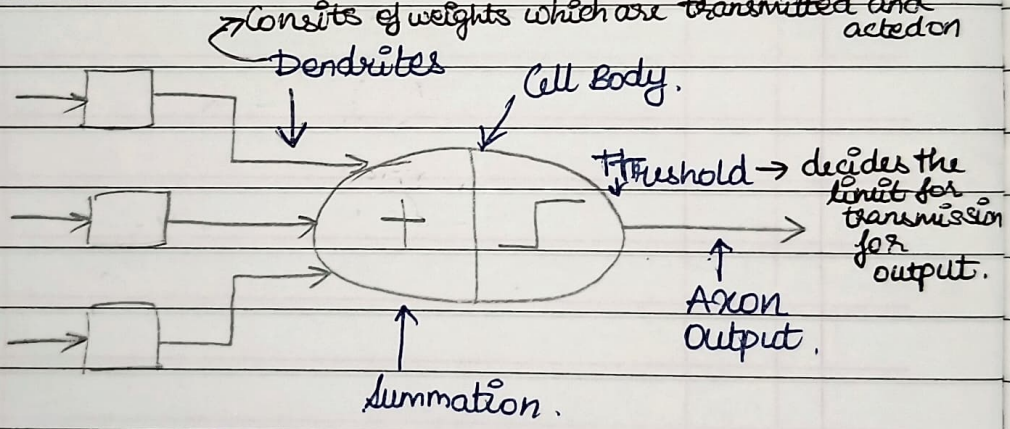
It is the computational version of this process of extracting ideas from the nature to develop 'artificial' (computational) systems or using natural media to perform computation.

ARTIFICIAL NEURAL NETWORKS.

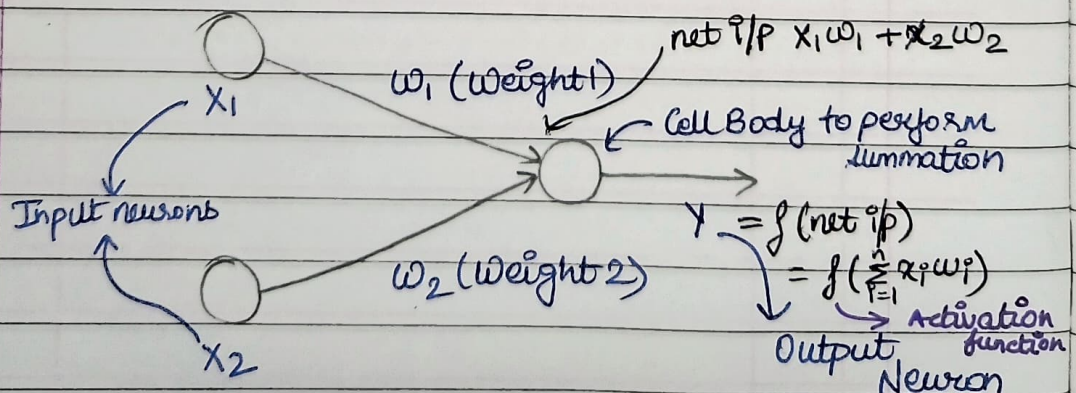
→ Based on neural network of the brain.



* ASSOCIATION OF NEURON WITH ARTIFICIAL NEURON



$$\text{NET I/P} = \sum_{i=1}^{n} x_i w_i$$



Weights can be negative, positive or zero.

* PROCESSING OF AN ARTIFICIAL NEURAL.

→ set of links, describing neurons with weight $w_1, w_2, \ldots, w_m$.

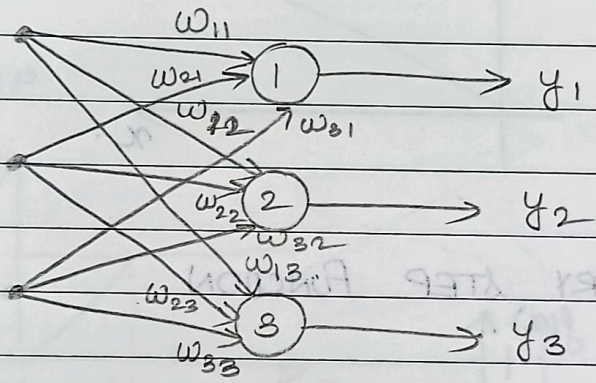
→ Net i/p

$$u = \sum_{j=1}^{m} w_j x_j.$$

→ Activation function

$$= \varphi(u+m).$$

* MULTILAYER ARTIFICIAL NEURAL NETWORK.



$$net_1 = x_1 w_{11} + x_2 w_{21} + x_3 w_{31}$$
$$= \sum_{j=1}^{3} x_j w_{j1}$$

$$net_2 = x_1 w_{12} + x_2 w_{22} + x_3 w_{32}$$
$$= \sum_{j=1}^{3} x_j w_{j2}$$

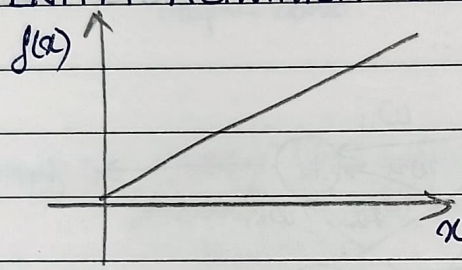
$$net_3 = x_1 w_{13} + x_2 w_{23} + x_3 w_{33}.$$
$$= \sum_{j=1}^{3} x_j w_{j3}.$$

$$y_1 = f(net_1) \quad y_2 = f(net_2) \quad y_3 = f(net_3).$$

* ACTIVATION FUNCTIONS.

DISCRETE	CONTINUOUS.
Hard Limit function	Sigmoidal Activation function.
→ Binary (Unipolar).	→ Binary SAF
→ Bipolar	→ Bipolar SAF.
→ Identity	

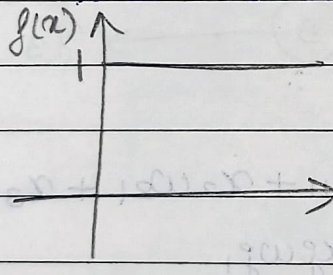
1. IDENTITY ACTIVATION FUNCTION



$y = f(x)$

$f(x) = x$

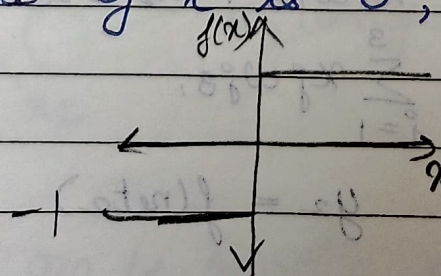
HLF 2. BINARY STEP FUNCTION.



$$f(x) = \begin{cases} 1 ; x > 0 \\ 0 ; x <= 0 \end{cases}$$

HLF 3. BIPOLAR STEP FUNCTION.

If value of x is less than 0, -1
If value of x is greater than 0, $+1$
If value of x is 0, ± 0.

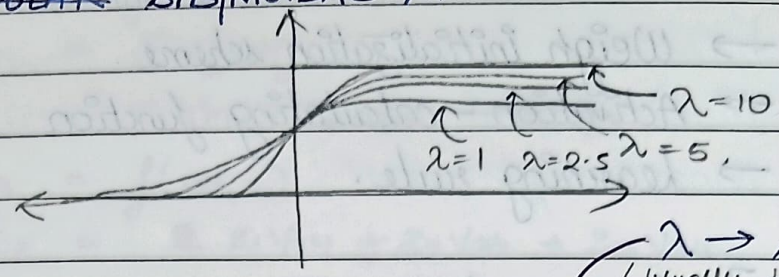


$$f(x) = \begin{cases} 1 ; x > 0 \\ -1 ; x \leq 0 \end{cases}$$

~~BAF~~ 4. BINARY ~~BIPOLAR~~ SIGMOIDAL AF

$$f(x) = \frac{1}{1+e^{(-\lambda x)}}$$

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

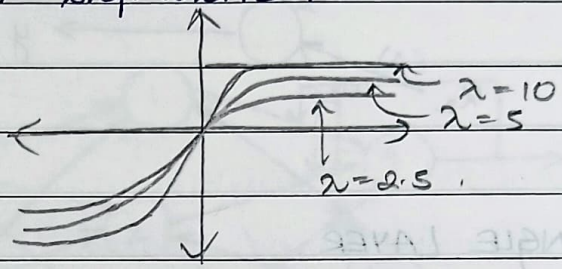


λ → slope.
(usually take 1 initially found by experimentation)
learning grade

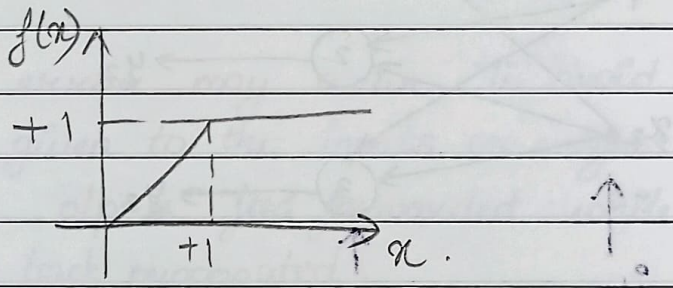
~~BAF~~ 5. ~~BINARY~~ BIPOLAR SIGMOIDAL AF

$$f(x) = \frac{2}{1+e^{(-\lambda x)}} - 1$$

$$f(net) = \frac{2}{1+e^{(-\lambda net)}} - 1$$



6. RAMP.



* CONSTRUCTING ANN.

Process for creating an ANN.

1. DETERMINE THE NETWORK PROPERTIES.
   → Network topology.
   → Types of connectivity.
   → Order of connection.
   → Weight range.

2. DETERMINE THE NODE PROPERTIES.
   → Activation range.

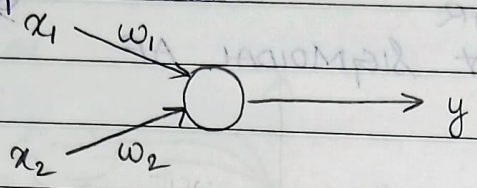
3. DETERMINE THE SYSTEM DYNAMICS.

→ Weigh initialization scheme.

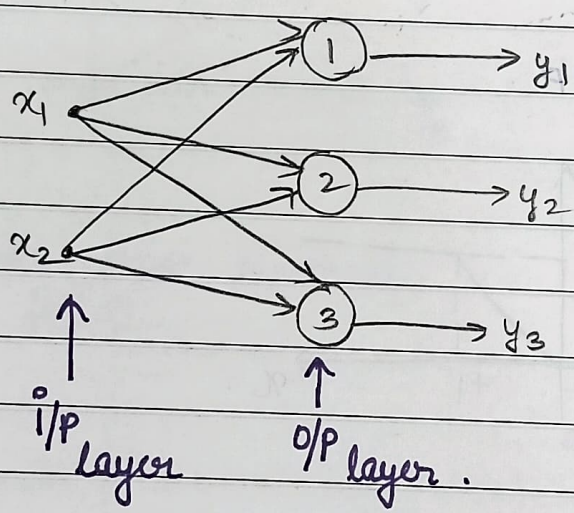
→ Activation - calculating function

→ Learning rule.

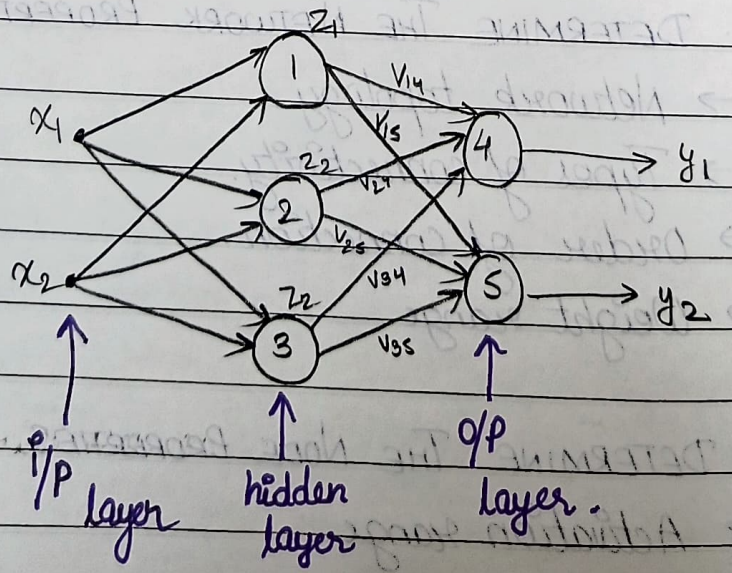
1] SINGLE NEURON



2] SINGLE LAYER.



3] MULTILAYER



$\therefore z_1 = f(net_1)$

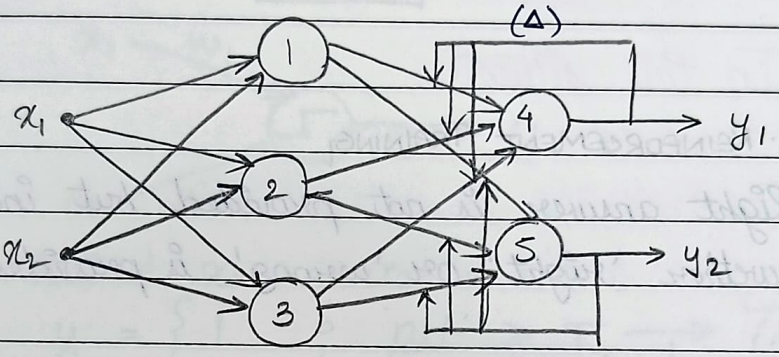
$net_1 = \sum x_i w_{ij}$

$= x_1 w_{11} + x_2 w_{21}$ ~~$+ x_3 w_{31}$~~.

$\therefore y_1 = f(net_2)$

$net_2 = z_1 v_{14} + z_2 v_{24} + z_3 v_{34}$.

4] FEEDBACK NETWORK.



An error may occur. To avoid it, feedback is given to the inputs or weights.
The o/p is fed forwarded while the error is back propogated.
This corruption of network is done to decrease the error created.
This is done until desired output is recieved or error has been significantly reduced.

Things to consider

1] Type of learning → Supervised
→ Unsupervised
→ Reinforcement.

2] Learning rules → Usually gives N/w topology

3] Network Architecture (topology).

4] Activation rule.

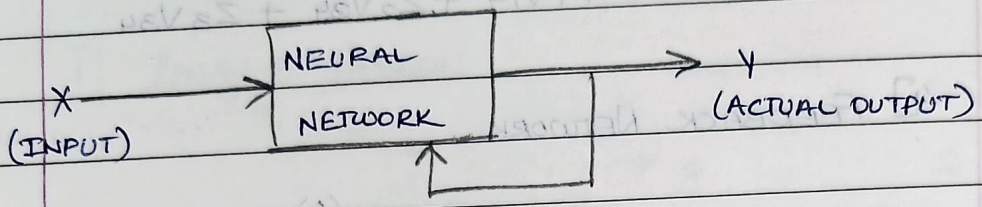
Neural N/w is like a black box as it cannot be determined which node has learned but we can see that we get the desired O/p.

Page No.: YOUVA
Date:

* TRAINING PROCESS.

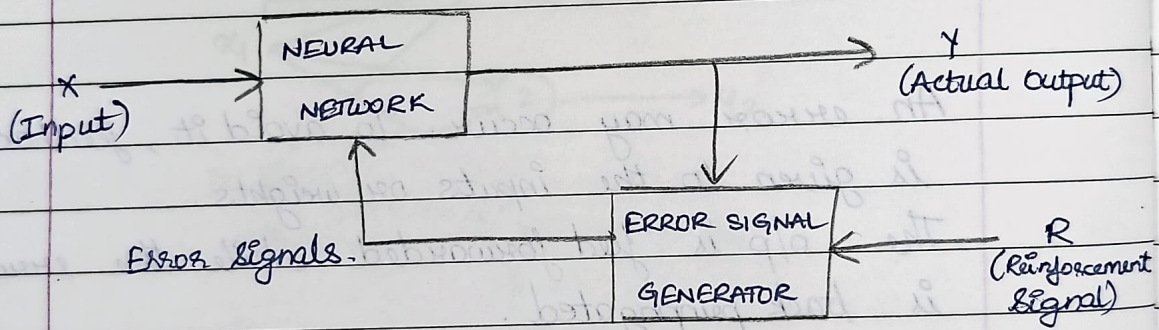
1. UNSUPERVISED TRAINING.

Most similar input vector is assigned to the same output unit.



2. REINFORCEMENT TRAINING

Right answer is not provided but indication of wether 'right' or 'wrong' is provided.

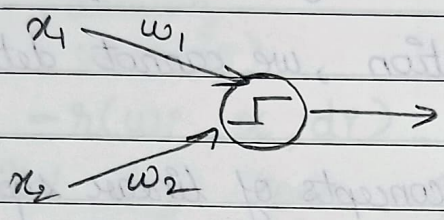


* SALIENT FEATURES OF ANN

1. Adaptive learning.
2. Self organization
3. Real time operation.
4. Fault tolerance via redudant information coding.
5. Massive parallelism
6. Learning and generalizing ability.
7. Distributed representation.

* McCULLOCH – PITTS NEURON.

→ It is a threshold deciding model.
→ Neurons are sparsely & randomly connected.
→ Firing state is binary (1 = firing & 0 = not firing).
→ All but one neuron ~~is~~ are excitatory.
→ One inhibitory neuron connects to all other neurons.
→ It functions to regulate network activity.



$$net = x_1 w_1 + x_2 w_2.$$

$$y = \begin{cases} 1 & ; \ net \geq T \\ 0 & ; \ net < T \end{cases} \rightarrow \text{Threshold.}$$

Now consider

$w_1 = 1$ $\quad$ $w_2 = 1$.

y → desired	x_1	x_2	net
0	0	0	0
1	0	1	1
1	1	0	1
1	1	1	2

↳ By performing OR on x_1 & x_2.

∴ Threshold should be 1.

$$y = \begin{cases} 1 & ; \ net \geq 1 \\ 0 & ; \ net < 1 \end{cases}$$

Now, let y have the o/p of on performing XOR on x_1 & x_2

$w_1 = 1$ $w_2 = 1$.

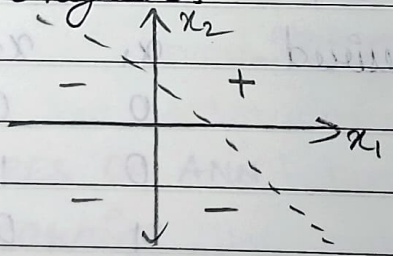
x_1	x_2	y	net	if $w_1 = -1$ & $w_2 = 1$
0	0	0	0	0
0	1	1	1	1
1	0	1	1	-1
1	1	0	2	0.

In this situation, we cannot determine the threshold.

It fails the concepts of linear superability. as we can see in the above XOR model.

Linear seperability.

→ Concept wherin the seperation of the input space into regions based on whether the network response is positive or negative.



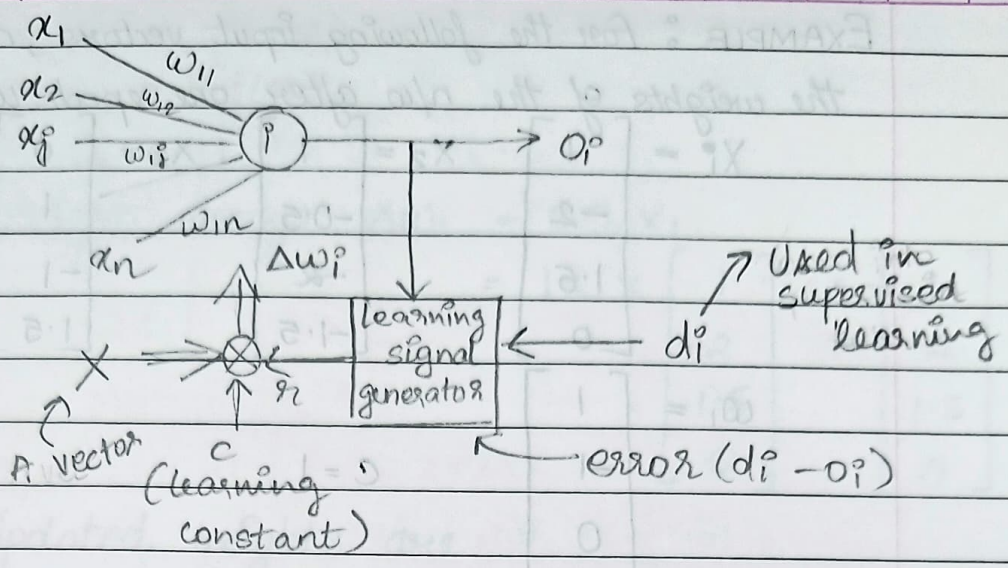
* LEARNING RULE

1. HEBBIAN LR
2. PERCEPTRONE LR
3. DELTA LR.
4. WIDROW HOFF LR.
5. CORRELATION LR.
6. WINNER TAKE ALL LR
7. OUTSTAR LR

- Type of learning
- Type of neuron
- Initial weights
- Neurons / Layer of neurons
- Weight updation formula

These are necessary for each.

GIST OF NEURAL N/W



$$r = r(w_i, x, d_i)$$

In the inital stage some weights are given. The result is O_i is given to learning signal generator and is checked with the desired or output d_i. If it is not like the desired output, the learning signal generator, we allow the neuron to learn the weights.

1. HEBBIAN LR

→ Unsupervised learning
→ Neuron can be discrete / continuous.
→ The initial weights are closed to zero (very small
→ It can be applied to a single neuron or a layer of neurons.
→ Weight updation formula.

$$\Delta w = c O x$$

$$\Delta w_{ij} = c O_i x_j$$

$c \rightarrow$ 'learning' constant.
$O_i \rightarrow$ i^{th} neuron.
$x_j \rightarrow$ j^{th} input neuron.

$$\Delta w_{ij} = c \cdot f(w_i x) x_j$$

$$\therefore w_{n+1} = w_n + \Delta w_n \quad ; n = 1, 2, \ldots$$

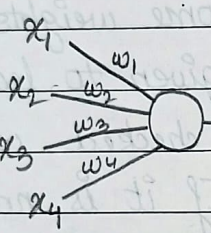
Epoch – Going through all vectors once to update the weights.

EXAMPLE : For the following input vectors, calculate the weights of the n/w after one epoch. using HLR.

$$X_1 = \begin{bmatrix} 1 \\ -2 \\ 1.5 \\ 0 \end{bmatrix} \quad X_2 = \begin{bmatrix} 1 \\ -0.5 \\ -2 \\ -1.5 \end{bmatrix} \quad X_3 = \begin{bmatrix} 0 \\ 1 \\ -1 \\ 1.5 \end{bmatrix}$$

$$W_1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix} \qquad C = 1$$

SOLUTION :



~~STEP 1~~ Let the neuron be bipolar binary polar.

$$f(net) = \begin{cases} 1 & ; net \geq 0 \\ -1 & ; net \leq 0 \end{cases}$$

$$f(net) = sgn(net)$$

STEP 1 – $X_1 = \begin{bmatrix} 1 \\ -2 \\ 1.5 \\ 0 \end{bmatrix} \quad W_1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix}$

$$net = W_1^T X_1$$

$$= \begin{bmatrix} 1 & -1 & 0 & 0.5 \end{bmatrix} \begin{bmatrix} 1 \\ -2 \\ 1.5 \\ 0 \end{bmatrix} = [3]$$

$O_1 = f(net)$

$= f(3)$

$O_1 = 1$

Weight change $\Delta w_i = c O_1 X_1$

$$= 1 \times 1 \times \begin{bmatrix} 1 \\ -2 \\ 1.5 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \\ 1.5 \\ 0 \end{bmatrix}$$

Updated weights are

$$w_2 = w_1 + \Delta w_1$$

$$= \begin{bmatrix} 2 \\ -3 \\ 3.5 \\ 0.5 \end{bmatrix}$$

STEP 2 - $X_2 = \begin{bmatrix} 1 \\ -0.5 \\ -2 \\ -1.5 \end{bmatrix}$

$$net_2 = w_2^T X_2 = \begin{bmatrix} 2 & -3 & 1.5 & 0.5 \end{bmatrix} \begin{bmatrix} 1 \\ -0.5 \\ -2 \\ -1.5 \end{bmatrix}$$

$$= -0.25.$$

$O_2 = -1$

Weight change $\Delta w_2^T = c O_2 X_2 = [-1 \quad 0.5 \quad 2 \quad 1.5]$

Updated weights are.

$$w_3^T = w_2 + \Delta w_2$$

$$= [1 \quad -2.5 \quad 3.5 \quad 2]$$

STEP 3 – $X_3^T = [0 \quad 1 \quad -1 \quad 1.5]$

$net_3 = W_3^T X_3$

$$= [1 \quad -2.5 \quad +3.5 \quad 2] \begin{bmatrix} 0 \\ 1 \\ -1 \\ +1.5 \end{bmatrix} = -3$$

$O_3 = -1$

$\Delta W_3^T = CO_3X_3 = [0 \quad -1 \quad 1 \quad -1.5]$

$W_4^T = W_3 + \Delta W_3$

$= [1 \quad -2.5 \quad 3.5 \quad 2] + [0 \quad -1 \quad 1 \quad -1.5]$

$= [1 \quad -3.5 \quad 4.5 \quad 0.5]$.

2) PERCEPTRON LEARNING RULE

→ It is supervised learning.

→ The type of neuron ~~can~~ is Discrete.

→ It can be applied to single neuron or layer of neurons.

→ Any initial weight can be taken.

→ Weight updation formula.

$$\Delta W = C(d-O)X$$

$$W_{n+1} = \Delta W_n + W_n.$$

d → desired output.

EXAMPLE : For the following input vector, find the change in weight after 1 epoch

$$X_1 = \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} \quad X_2 = \begin{bmatrix} 0 \\ 1.5 \\ +0.5 \\ -1 \end{bmatrix} \quad X_3 = \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}$$

$W_1 = [1 \quad -1 \quad 0 \quad 0.5]$

Since $w_3 = w_2$, we can say w_2 is good for x_2.

M T W T F S S | Page No.: | Date: | YOUVA

Assume $c = 0.1$ $d_1 = -1$ $d_2 = -1$ $d_3 = 1$.

SOLUTION :-

Let the neuron be bipolar step function.

$$f(net) = sgn(net).$$

STEP 1 -

$$X_1 = [1 \ -2 \ 0 \ -1]^T \qquad w_1 = [1 \ -1 \ 0 \ 0.5]$$

$$net = w_1 * X_1$$

$$= [1 \ -1 \ 0 \ 0.5] \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}$$

$$= [2.5]$$

$$O_1 = 1$$

$$\Delta w_1 = c(d_1 - O_1) X_1$$

$$= 0.1 (-1 - 1) X [1 \ -2 \ 0 \ -1]$$

$$= \begin{bmatrix} -0.2 \\ 0.4 \\ 0 \\ 0.2 \end{bmatrix}$$

~~STEP 2~~ $w_2 = \begin{bmatrix} 0.8 \\ -0.6 \\ 0 \\ +0.8 \end{bmatrix}^T$

~~net~~

STEP 2 -

$$X_2^T = [0 \ 1.5 \ -0.5 \ -1]^T \qquad w_2 = [0.8 \ -0.6 \ 0 \ 0.7]$$

$$net = w_2 X_2$$

$$= [0.8 \ -0.6 \ 0 \ 0.7] \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix} = [-1.6]$$

$$O_2 = -1$$

$$\Delta w_2 = c(d_2 - o_2)x_2$$

$$= 0.1(-1+1)\begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix}$$

$$= 0.$$

$\therefore w_3 = w_2.$

STEP 3 –

$$x_3 = \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix} \qquad w_3 = [0.8 \quad -0.6 \quad 0 \quad 0.7]$$

$$net = w_3 x_3$$

$$= [0.8 \quad -0.6 \quad 0 \quad 0.7]\begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix} = -2.1$$

$$o_3 = -1$$

$$\Delta w_3 = c(d_3 - o_3)x_3$$

$$= 0.1(+1+1)\begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix} = \begin{bmatrix} -0.2 \\ 0.2 \\ 0.1 \\ -0.2 \end{bmatrix}$$

$$w_4 = w_3 + \Delta w_3$$

$$= [0.6 \quad -0.4 \quad 0.1 \quad 0.5]$$

NOTE.

If we make use of Bipolar step function.

and

$$d \begin{cases} 1 \\ -1 \end{cases}$$ (desired output).

if

$o = d$, $d - o = 0$ ∴ no change.

$o \neq d$, $d = 1 \quad o = -1$ ∴ net $= 2cx$.

$d = -1 \quad o = 1$ ∴ net $= -2cx$.

∴ If there is a weight change, it will be equal to $\pm 2cx$.

3] DELTA LEARNING RULE

→ Supervised learning

→ Type of neuron is continuous (soft limit sigmoidal function)

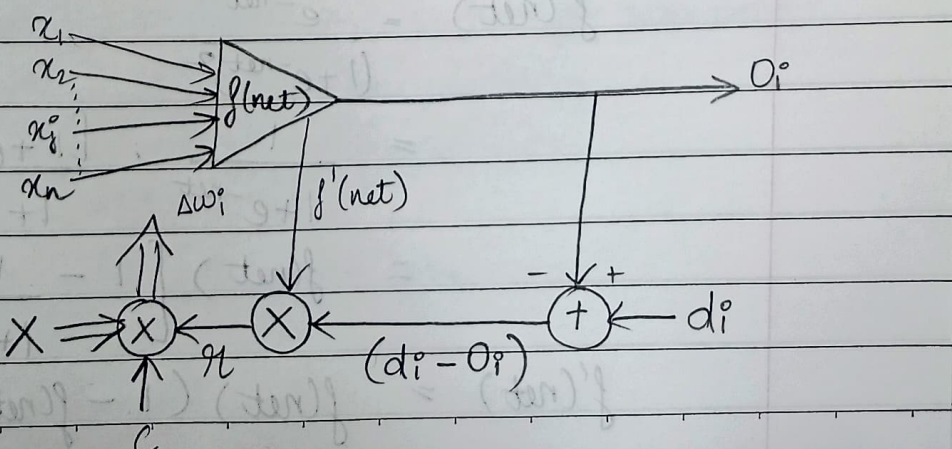
→ Can be applied to single neuron or layer of neurons.

→ Any initial weight.

→ Weight ~~updation~~ updatation formula.

$$\Delta w = c(d - o) f'(net) x.$$

$$w_{n+1} = \Delta w_n + \Delta w_n$$



Page No.: YOUVA
Date:

Error, $E \triangleq \frac{1}{2}(d_i - o_i)^2$

$\triangleq \frac{1}{2}[d_i - f(w_i x)]^2$

Obtain the error gradient vector value by taking partial derivative w.r.t w.

$\nabla E = -2 \times \frac{1}{2}(d_i - f(w_i x))\, f'(w_i x) \cdot x$

$= -(d - f(w_i x))\, f'(w_i x) \cdot X$

The change in the weight requires minimization of this error.

∴ Taking -ve. gradient, we get

$\Delta w = -\eta \nabla E$ where η is +ve constant

$= \eta (d_i - o_i) f'(net_i) X$

Activation function derivative.

1] Unipolar sigmoidal AF

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

let $\lambda = 1$

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

Taking derivative w.r.t. net, we get

$f'(net) = \frac{e^{-net}}{(1+e^{-net})^2}$

$= \frac{1}{1+e^{-net}} \cdot \frac{[1+e^{-net}-1]}{1+e^{-net}}$

$= f(net)\left[1 - \frac{1}{1+e^{-net}}\right]$

$f'(net) = f(net)(1 - f(net))$

2] Bipolar sigmoidal AF...

$$f(net) = \frac{2}{1+e^{-\lambda net}} - 1$$

Let $\lambda = 1$

$$f(net) = \frac{2}{1+e^{-net}} - 1$$

Taking derivative w.r.t net, we get.

$$f'(net) = \frac{1}{2}(1 - f(net)^2)$$

4] WIDROW HOFF LEARNING RULE

→ Supervised learning.
→ Type of neuron can be any.
→ Can be applied to single neuron or layer of neurons.
→ Any initial weight.
→ Weight updatation formula.

$$\Delta w = c(di - wi \cdot x)x$$

$$w_{n+1} = w_n + \Delta w_n.$$

5] CORRELATION LEARNING RULE

→ Supervised learning.
→ Type of neuron can be any
→ Single neuron or layer of neurons.
→ Very small initial weights.
→ Weight updatation formula.

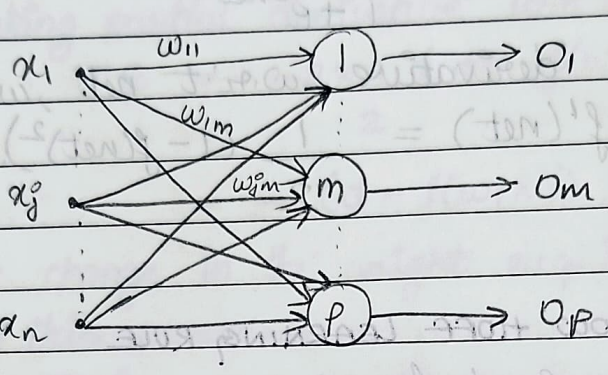
$$\Delta w = cdx$$

$$w_{n+1} = w_n + \Delta w_n.$$

6] WINNER TAKES ALL LEARNING RULE

This is Competative learning

→ Unsupervised learning.
→ Type of neuron is continuous.
→ It needs layer of neurons.
→ initial weights are random.



$$w_m = [w_{1m} \dots w_{jm} \dots w_{nm}]$$

Weight change is done to the winner. → w_{inner}

$$\Delta w_m = \alpha (x - w_m)$$

How to find winner neuron.

1] Maximum inner product.
Maximum net i/p neuron is declared winner.

2] Smallest difference b/w i/p & weight vector.
The difference b/w input x_m & weight w_m vector should be minimum.

7] OUTSTAR LEARNING RULE.

→ Supervised learning.
→ Type of neuron is continuous.
→ It needs layer of neurons.
→ Small initial weight.
→ Weight change formula.

$$\Delta w = B(d - w)$$

* PERCEPTRON TRAINING ALGORITHM.

X → input patterns;
W → initial weights.
neurons → hard limit activation function.

Given $\{x_1 d_1, x_2 d_2, \ldots, x_P d_P\}$
x_i is $(n \times 1)$ & d_i is (1×1) $i = 1, 2, \ldots P$

vector ← $y_i = \begin{bmatrix} x_i \\ 1 \end{bmatrix}$ & → vector.

STEP 1 ; $c > 0$ is chosen.
STEP 2 : w is initialised at small random values.
w is $(n+1) \times 1$ matrix ↗ vector.
Counters and errors are initialised

$$k \leftarrow 1 \quad , p \leftarrow 1 \quad , E \leftarrow 0$$

STEP 3 :

$$y \leftarrow y_p \qquad d \leftarrow d_p$$
$$O \leftarrow \text{sgn}(w'y)$$

STEP 4 :

$$w \leftarrow w + \frac{1}{2} c(d - O) y.$$

STEP 5: Cycle error is computed.

$$E \leftarrow \frac{1}{2}(d-o)^2 + E.$$

STEP 6: If $p < P$ then $p \leftarrow p+1$, $k \leftarrow k+1$ go to step 3.
Else go to step 7

STEP 7: Training cycle is computed.
For $E=0$, trainning session terminated.
Output → weights & k.
$E > 0$, then $E \leftarrow 0$, $p \leftarrow 1$ & enter new training cycle by going to step3

EVOLUTIONARY COMPUTING - GENETIC ALGORITHMS.

→ Technique to solve problems which need optimization.

→ Based on Darwin's theory of evolution.