Design a Hebb net to implement logical AND function:-

→ The training data for the AND function :-

Inputs			Target
x_1	x_2	b	y
1	1	1	1
1	-1	1	-1
-1	1	1	-1
-1	-1	1	-1

→ Initially the weight & bias are set to zero, i.e.

$$w_1 = w_2 = b = 0$$

→ First inpt $[x_1\ x_2\ b] = [1\ 1\ 1]$ and target = 1 [i.e, $y = 1$].

→ setting the inital weights as old weights and applying the Hebb rule, we get :-

$$W_i(new) = w_i(old) + \Delta w_i$$

→ Here $\Delta w_i = x_i y$

$$\Delta w_1 = x_1 y = 1 \times 1 = 1$$

$$\Delta w_2 = x_2 y = 1 \times 1 = 1$$

$$\Delta b = y = 1$$

→ So, $w_1(new) = w_1(old) + \Delta w_1$

$= 0 + 1 = 1$

$w_2(new) = w_2(old) + \Delta w_2$

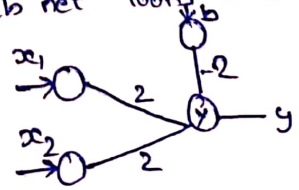
$= 0 + 1 = 1$

$b(new) = b(old) + \Delta b$

$= 0 + 1 = 1$

Inputs			weight changes			weights		
x_1	x_2	b	y	Δw_1	Δw_2	Δb	w_1	w_2	b
							(0	0	0)
1	1	1	1	1	1	1	1	1	1
1	-1	1	-1	-1	1	-1	0	2	0
-1	1	1	-1	1	-1	-1	1	1	-1
-1	-1	1	-1	1	1	-1	2	2	-2

∴ The Hebb net looks like



-> second input $[x_1\ x_2\ b] = [1\ -1\ 1]$ and target $y = -1$

-> The weight change here is :-

$\Delta w_1 = x_1 y = 1 \times -1 = -1$

$\Delta w_2 = x_2 y = -1 \times -1 = 1$

$\Delta b = \Delta b = \Delta y = -1$

$[\Delta w_1, \Delta w_2, \Delta b] = [-1, 1, -1]$

-> so, $w_1(new) = w_1(old) + \Delta w_1$

$= 1 + (-1) = 0$

$w_2(new) = w_2(old) + \Delta w_2 \Rightarrow 1 + 1 = 2$

$b(new) = b(old) + \Delta b \Rightarrow 1 + (-1) = 0$

$[w_1, w_2\ b] = [0\ 2\ 0]$

-> Third input $[x_1\ x_2\ b] = [-1\ 1\ 1]$ and target i.e, $y = -1$.

-> The weight change here is :-

$\Delta w_1 = x_1 y = -1 \times -1 = 1$

$\Delta w_2 = x_2 y = 1 \times -1 = -1$

$\Delta b = y = -1$

$[\Delta w_1, \Delta w_2, \Delta b] = [1\ -1\ -1]$

-> so, $w_1(new) = w_1(old) + \Delta w_1 \Rightarrow 0 + 1 = 1$

$w_2(new) = w_2(old) + \Delta w_2 \Rightarrow 2 + (-1) = 1$

$b(new) = b(old) + \Delta b \Rightarrow 0 + (-1) = -1$

$[w_1, w_2\ b] = [1\ 1\ -1]$

-> Fourth input $[x_1\ x_2\ b] = [-1\ -1\ 1]$ and target i.e, $y = -1$

-> The weight change here is :-

$\Delta w_1 = x_1 y = -1 \times -1 = 1$

$\Delta w_2 = x_2 y = -1 \times -1 = 1$

$\Delta b = y = -1$

$[\Delta w_1\ \Delta w_2\ \Delta b] = [1\ 1\ -1]$

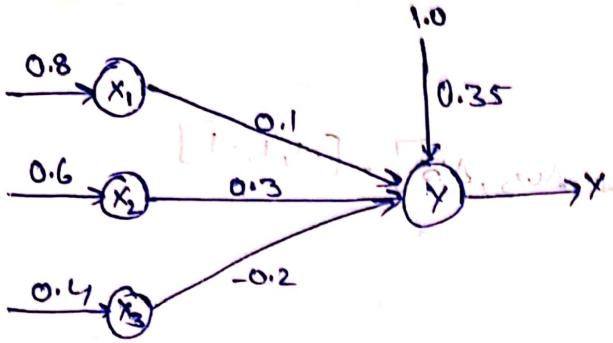
-> So, $w_1(new) = w_1(old) + \Delta w_1 \Rightarrow 1 + 1 = 2$

$w_2(new) = w_2(old) + \Delta w_2 \Rightarrow 1 + 1 = 2$

$b(new) = b(old) + \Delta b \Rightarrow -1 + (-1) = -2$

$[w_1, w_2\ b] = [2\ 2\ -2]$

* Sigmoid Activation Function solved example:-



Binary sigmoid activation function

$$Y = f(y_{in}) = \frac{1}{1+e^{-y_{in}}}$$

Bipolar sigmoid activation function

$$Y = f(y_{in}) = \frac{2}{1+e^{-y_{in}}} - 1$$

-> Here $Y_{in} = b\sum_{i=1}^{n} \omega_i x_i$

$$Y_{in} = b + x_1\omega_1 + x_2\omega_2 + x_2\omega_3$$

$$= 0.35 + (0.8)(0.1) + (0.6)(0.3) + (0.4)(-0.2)$$

$$= 0.53.$$

-> Binary sigmoid activation function : $Y = f(y_{in}) = \frac{1}{1+e^{-y_{in}}} = \frac{1}{1+e^{-(0.53)}} = \underline{0.625}$

-> Bipolar sigmoid activation function : $y = f(y_{in}) = \frac{2}{1+e^{-y_{in}}} - 1 = \frac{2}{1+e^{-(0.53}} - 1 = \underline{0.259}$

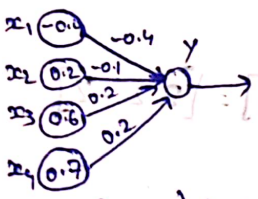
-> Threshold Function : $f(v) = \begin{cases} 0 & \text{if } v < 0 \\ 1 & \text{if } v \geq 0 \end{cases}$ [Here $v = y_{in}$]

-> Linear Function : $f(v) = v, \forall v$

-> Piecewise Linear Function : $f(v) = \begin{cases} 1, & v \geq 0.5 \\ v, & -0.5 < v < 0.5 \\ 0, & v \leq -0.5 \end{cases}$

-> Rectified Linear Unit (ReLU) : $f(v) = \text{Max}(0, v)$

-> Leaky Rectified Linear unit (Leaky-ReLU) : $f(v) = \text{Max}(\alpha v, v)$ where $0 < \alpha < 1$



$(y_{in}) = (-0.4)(-0.4) + (-0.1)(0.2) + (0.2)(0.6) + (0.2)(0.7) = \underline{0.4}$

linear = 0.4 ; Piecewise linear function = $\underline{0.4}$

ReLU $\Rightarrow f(v) = \text{Max}(0, v) \Rightarrow \text{Max}(0, 0.4) \Rightarrow \underline{0.4}$

$(\alpha = 0.4)$ Leaky ReLU $\Rightarrow f(v) = \text{Max}(\alpha v, v) = \text{Max}((0.4)(0.4), 0.4) = M(0.16, 0.4) = \underline{0.4}$

Sigmoid $(\alpha = 1) \Rightarrow y = f(v) = \frac{1}{1+e^{-\alpha v}} = \frac{1}{1+e^{-(0.4)}} = \underline{0.59}$

Softmax Activation function: $f(v_i) = \frac{e^{v_i}}{\sum_{i=1}^{p} e^{v_i}} = \frac{1.49182}{e^{(0.1)} + e^{(-0.1)} + e^{(0.2)} + e^{(0.4)}} = \frac{1.491}{4.431} = \underline{0.336}$

Implement AND function using perceptron networks for bipolar inputs and targets :-

→ In case of perceptron learning rule, the learning signal is the difference between the calculated output and actual (target) output of a neuron.

→ The output "y" is obtained on the basis of the net input calculated and activation function being applied over the net input.

$$y_{in} = b + \sum_{i=1}^{n} x_i w_i$$

→ $y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > \theta \\ 0 & \text{if } -\theta \le y_{in} \le \theta \\ -1 & \text{if } y_{in} < -\theta \end{cases}$ Here θ is threshold

→ weights are updated using formula -

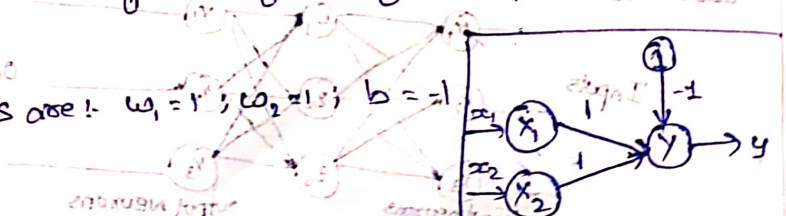
if $y \neq t$, then $w(new) = w(old) + \alpha t x$ (here α is learning rate)

else, we have : $w(new) = w(old)$

$\therefore \Delta w_1 = \alpha t x_1 \; ; \; \Delta w_2 = \alpha t x_2 \; ; \; \Delta b = \alpha t$

Input		Target	Net input	Calculated output	Weight changes			Weights		
x_1	x_2	(t)	(y_{in})	(y)	Δw_1	Δw_2	Δb	w_1	w_2	b
								(0	0	0)
EPOCH-1										
1	1	1	0	0	1	1	1	1	1	1
1	-1	-1	1	1	-1	1	-1	0	2	0
-1	1	-1	2	1	1	-1	1	-1	1	1
-1	-1	-1	-3	-1	0	0	0	-1	1	1
EPOCH-2 (or 2nd iteration)										
1	1	1	1	1	0	0	0	-1	1	1
1	-1	-1	-1	-1	0	0	0	-1	1	1
-1	1	-1	-1	-1	0	0	0	-1	1	1
-1	-1	-1	-3	-1	0	0	0	-1	1	1

$\therefore$ for $\alpha = 1$, the final weights & bias are :- $w_1 = 1 \; ; \; w_2 = 1 \; ; \; b = 1$



* Artificial Neural Network : Introduction:

-> An artifical neural network (ANN) may be defined as an information - processing model that is inspired by the way biological nervous systems, such as the brain, process information.

-> This model tries to replicate only the most basic functions of the brain.

-> An ANN is composed of a large number of highly interconnected processing units (neurons) working in unison to solve specific problems.

-> Like human being, the ANN learn by examples.

-> An ANN is configured for a specific application, such as spam classification, Face Recognition, Pattern recognition through a learning process.

-> Each neuron is connected with the other by a connection link.

-> Each connection link is associated with weights which contains the information about the input signal.

-> This information is used to solve a particular problem by the neural network.

-> ANN have the ability to learn, recall, identify & generalize training patterns or data similar to human brain.

-> Thus, the ANN Processing elements are called neurons or artificial neurons.

-> Each neuron has an internal state of its own.

-> This internal state is called the activation level of neuron.

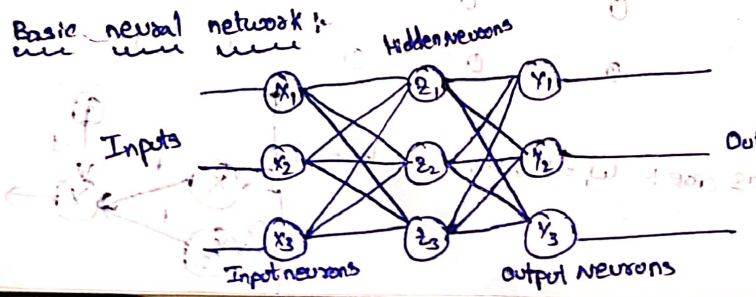
-> The activation signal of a neuron is transmitted to other neurons.

-> Remember, a neuron can send only one signal at a time, which can be transmitted to several other neurons.

Neural Networs :-

Huge distributed information - processing system, made up of simpler inter-connected processing units, called as neurons (or) nodes which work together collectively.

Basic neural network :-



* Application of Neural Networks :-

-> Control systems → Signal processing

-> Image, voice, Pattern recognition

-> DBMS

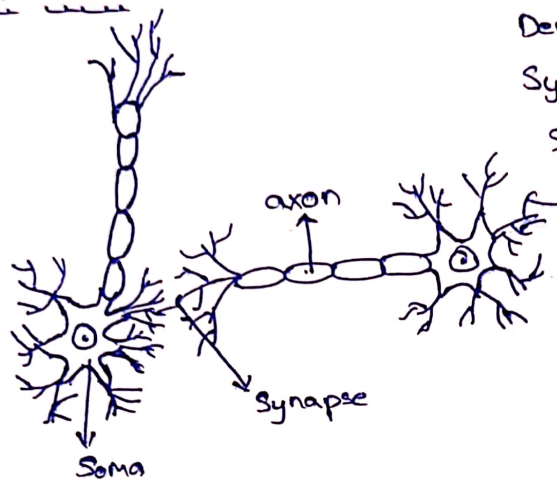
-> Data mining & processing

-> medical Imaging & diagnosis

-> Traffic control

-> weather prediction & forecast.

* Biological Neuron :-



Dendrite : receives signals from other neurons

Synapse : Point of connection to other neurons

Soma : Process the information.

Axon : transimt the output of neutron

BNN (Human)	ANN (Computer)
-> Cell	-> Neuron / node
Dendrites / synapse	weights
Soma	Net input
Axon	Output
-> Parallel processing i.e, more than 1 at a time	-> Single time signal Single processing
-> Slow (Comparatively)	-> Fast
-> Unreliable (can forget years ago topic)	-> Reliable
-> Dynamic infrastructure	-> Static infrastructure.

-> A network is said to be feedforward network if no neuron in the output layer is an input to a node in the same (or) preceding layer.

-> The hidden layers are not in contact with external enviroment. With more no of hidden layers, the output response is more efficient. The nodes in the prev layers are connected to the nodes in the next layers.

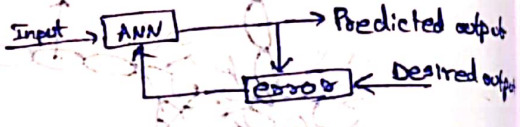
-> In a single layer recurrent network, the feedback forms a closed loop

* Supervised learning :-

-> The learning here is performed with the help of the teacher.

-> Input vector along with ~~out~~ target vector is known as the training pair.

-> eg :- Classification problems, Regression problems



* UNsupervised learning :-

-> Learning without teacher; Tadpole learn to swim.

-> during training process, network receives input patterns and organize it to form clusters ; eg: clustering.

* ~~Re~~Semi - Supervised learning :-

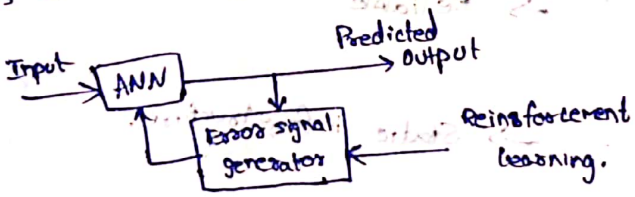
-> both labelled & unlabelled data is given. (sample of labelled data is small compositive to unlabelled data)

-> initall training is done using labelled and then it will be applied to greater no of unlabelled data

* Reinforcement learning :-

-> Learning is done by interaction with the environment. similar to supervised learning.

-> agent receives feedback in the form of rewards or penalties based on its actions, and its goal is to learn a policy that maximizes its cumulative reward over time.



Applications :-

web search; Computational biology ; Finance ; E-Commerce ; space exploration; robotics ; debugging ; Intelligent video surveillance ; social networking.

Learning Rules:-

i) Error correction:-

In this type of learning; The network adjusts its weights in response to the difference between its predicted output & desired output. This is done until the Predicted output equals to desired output.

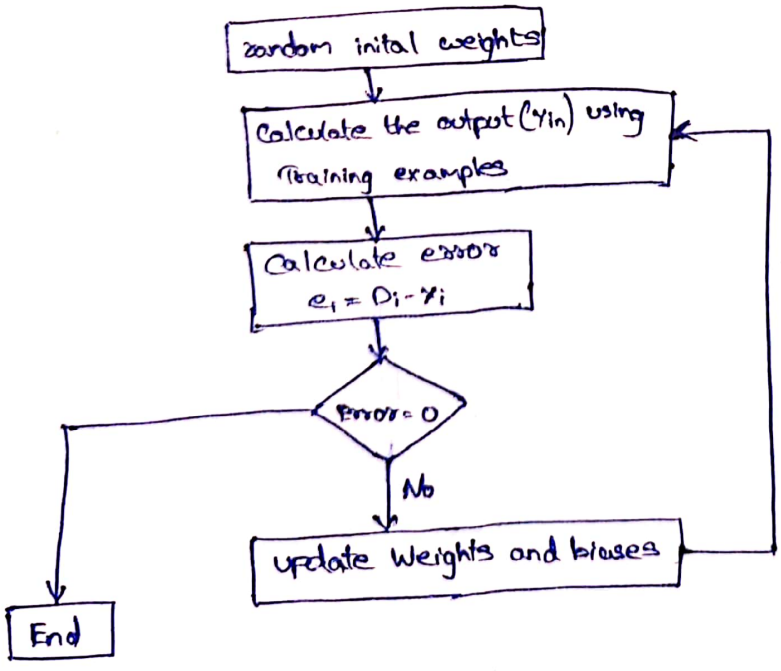
ii) Memory-based learning:- This type of learning involves storing the training data examples in the network & using them on new data. (handwritten recog.)

iii) Hebbian learning:- "Cell that fire together, wire together", in other words when two neurons are activated, the strength of the connection b/w them increases, and vice versa.

iv) Competitive learning:- neurons in the network compete with each other to become activated in response to particular input. The winner then adjust its weight to better recognize the input.

v) Boltzmann learning:- This type of learning is a unsupervised learning based on the Boltzmann distribution from statistical physics. Learns by sampling from this distribution and adjusting the weights to maximize the probability of generating the correct outputs.

* Perception learning algorithm flow chart:-



- To get the correct weights, many training 'epochs' are used with suitable learning rate α.
- This can't be used to solve non-linearly separable like (XOR)

2 kinds of learning in ANN :-

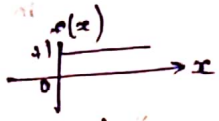
(gradient decent (or) back prop.

i) Parameter learning : It updates the connecting weight in a neural net.

ii) Structure learning : It focuses on the change in network structure (which includes the number of processing elements as well as their connection types.)

Activation Functions :- (here θ represent threshold value)

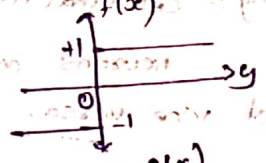
i) Identity function : $f(x) = x$ for all x



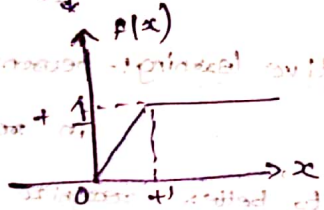
ii) binary step function : $f(x) = \begin{cases} 1 & \text{if } x \geq \theta \\ 0 & \text{if } x < \theta \end{cases}$

iii) bipolar step function :

$$f(x) = \begin{cases} 1 & \text{if } x \geq \theta \\ -1 & \text{if } x < \theta \end{cases}$$



iv) Ramp function : $f(x) = \begin{cases} 1 & \text{if } x > 1 \\ x & \text{if } 0 \leq x \leq 1 \\ 0 & \text{if } x < 0 \end{cases}$



v) Sigmoidal functions : widely used in back-propagation nets. & of two types :

i) Binary sigmoid function : $f(x) = \frac{1}{1+e^{-\lambda x}}$ (0 to 1 range) {λ is steepness parameter}

ii) Bipolar sigmoid function : $f(x) = \frac{2}{1+e^{-\lambda x}} - 1$ (-1 to 1 range)

(or) $y = f(y_{in}) = \frac{1}{1+e^{-y_{in}}}$ for Binary sigmoid activation function

$y = f(y_{in}) = \frac{2}{1+e^{-y_{in}}} - 1$ for Bipolar sigmoid activation function.

where: $y_{in} = b + \sum_{i=1}^{n} x_i w_i$ (for eg, $n = 3$ then, $y_{in} = b + x_1 w_1 + x_2 w_2 + x_3 w_3$)

Mcculloch-Pitts Neuron:-

since the firing of the output neuron is based upon the threshold, the activation function here is defined as:-

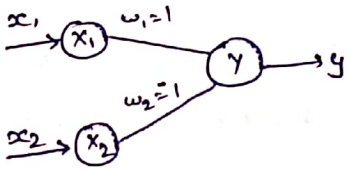
$$f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} \geq \theta \\ 0 & \text{if } y_{in} < \theta \end{cases}$$

Here, the threshold value should also satisfy :- $\theta > nw - p$ (n → no. of neurons; w → +ve weight; p → negative weight)

Implementing ANDNOT function using Mcculloch-pitts Neuron:-

x_1	x_2	y
1	1	-1
1	-1	1
-1	1	-1
-1	-1	-1

assume $w_1 = 1$ & $w_2 = 1$



now, $y_{in} = x_1 w_1 + x_2 w_2$

$(1,1)$, $y_{in} = 1(1) + 1(1) = 2$ → only this can't be fired in this so update weights.

$(1,-1)$, $y_{in} = 1(1) + 1(-1) = 0$

$(-1,1)$, $y_{in} = 1(-1) + 1(1) = 0$

$(-1,-1)$, $y_{in} = 1(-1) + 1(-1) = -2$

assume: $w_1 = 1$ & $w_2 = -1$

$(1,1)$, $y_{in} = 1(1) + (1)(-1) = 0$

$(1,-1)$, $y_{in} = 1(1) + (-1)(-1) = 2$

$(-1,1)$; $y_{in} = 1(-1) + 1(-1) = -2$

$(-1,-1)$; $y_{in} = -1(1) + (-1)(-1) = 0$

now by using $w_1 = 1$ & $w_2 = -1$; we can only fire for $x_1 = 1$ & $x_2 = -1$

so the θ will be : $\theta \geq 2$ (threshold is θ)

Implementing AND function using Mcculloch-pitts Neuron :-

x_1	x_2	y
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

assume $w_1 = 1$ & $w_2 = 1$

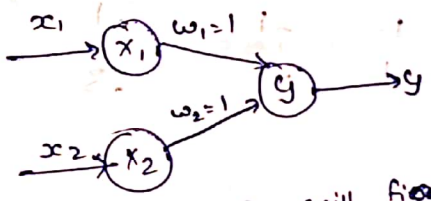
now, $y_{in} = x_1 w_1 + x_2 w_2$

$(1,1)$, $y_{in} = 1(1) + 1(1) = 2$

$(1,-1)$, $y_{in} = 1(1) + 1(-1) = 0$

$(-1,1)$, $y_{in} = 1(-1) + 1(1) = 0$

$(-1,-1)$, $y_{in} = 1(-1) + 1(-1) = -2$



here for $\theta \geq 2$ will fire only the required x_1 & x_2.

$\therefore y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} \geq 2 \\ 0 & \text{if } y_{in} < 2 \end{cases}$

here in mcculloch-pitts Neuron, we will change the assume weights until only the desired and required x_1, x_2 should be fired.

AND using perceptron:-

x_1	x_2	T
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

$$y_{in} = b + \sum_{i=1}^{n} x_i w_i \quad ; \quad y = f(y_{in}) = \begin{cases} 1 & \text{if } y > 0 \\ 0 & \text{if } y = 0 \\ -1 & \text{if } y < 0 \end{cases}$$

$$= b + x_1 w_1 + x_2 w_2$$

$$w_i(new) = w_i(old) + \Delta w_i$$

$$\Delta w_i = \alpha x_i t \cdot (\alpha = 1)$$

$$\Delta w_1 = \alpha t x_1$$

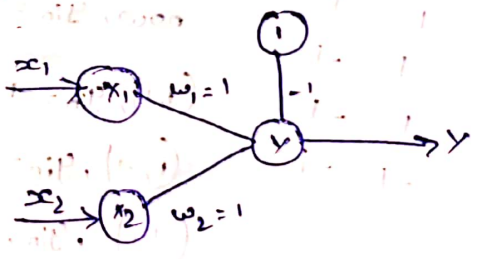
$$\Delta w_2 = \alpha t x_2$$

$$\Delta b = \alpha t$$

Inputs		(t)	net input	Calculated output	weight changes			weights		
x_1	x_2	Target	(y_{in})	(y)	Δw_1	Δw_2	Δb	w_1	w_2	b
Epoch 1								(0	0	0)
1	1	1	0	0	1	1	1	1	1	1
1	-1	-1	1	1	-1	1	-1	0	2	0
-1	1	-1	2	1	1	-1	-1	1	1	-1
-1	-1	-1	-1	-1	0	0	0	1	1	-1

Epoch 2

1	1	1	1	1	0	0	0	1	1	-1
1	-1	-1	-1	-1	0	0	0	1	1	-1
-1	1	-1	-1	-1	0	0	0	1	1	-1
-1	-1	-1	-3	-1	0	0	0	1	1	-1



Adaline for OR function:-

x_1	x_2	t
1	1	1
1	-1	1
-1	1	1
-1	-1	-1

→ Initially, the weights are set to $w_1 = w_2 = b = 0.1$ and learning rate = 0.1 {& 0.2 All for AND NOT}

→ Calculate net input $\Rightarrow y_{in} = b + \sum_{i=1}^{n} x_i w_i$

→ Then compute $(t - y_{in})$

→ Then calculate change in weights : $\Delta w_1 = \alpha (t - y_{in}) x_1$

$\Delta w_2 = \alpha (t - y_{in}) x_2$

$\Delta b = \alpha (t - y_{in})$

→ Then update weights : $w_i(new) = w_i(old) + \Delta w_i$

→ Then Calculate error : $E = (t - y_{in})^2$

→ Calculate the sum of errors i.e ΣE in each epoch and check that the ΣE of epoch is decreasing by Calculating for First 2 Epoch

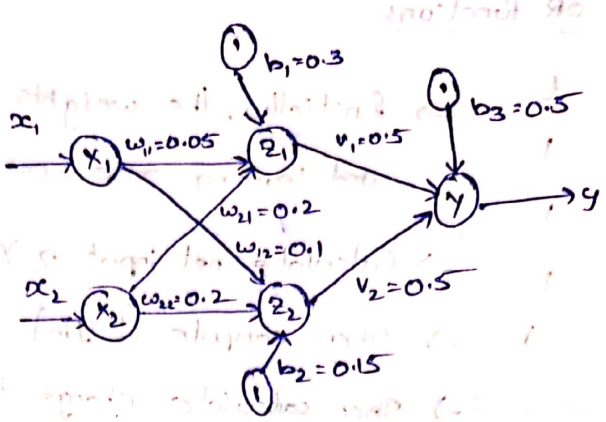
$Y_{in} = b + \Sigma x_i w_i$ $\quad (t - y_{in})$ $\quad \Delta w_1 = \alpha (t - y_{in}) x_1$, $\Delta w_2 = \alpha (t - y_{in}) x_2$, $\Delta b = \alpha (t - y_{in})$ $\quad$ here $\alpha = 0.1$

Inputs			Target	Net input	error	weight changes			weights			Error
x_1	x_2	1	t	(y_{in})	$(t - y_{in})$	Δw_1	Δw_2	Δb	w_1 (0.1	w_2 0.1	b 0.1)	$(t - y_{in})^2$
1	1	1	1	0.3	0.7	0.07	0.07	0.07	0.17	0.17	0.17	0.49
1	-1	1	1	0.17	0.83	0.083	-0.083	0.083	0.253	0.087	0.253	0.69
-1	1	1	1	0.087	0.913	-0.0913	0.0913	0.0913	0.1617	0.1783	0.3443	0.83
-1	-1	1	-1	0.0043	-1.0043	0.1004	0.1004	-0.1004	0.2621	0.2787	0.2439	1.01

implementing XOR using Madaline

-) :

x_1	x_2	Target
1	1	-1
1	-1	1
-1	1	1
-1	-1	-1



-) First calculate the net input to the hiddenlayer z_{in1}; z_{in2} i.e $\begin{cases} z_{in1} = b_1 + x_1 w_{11} + x_2 w_{21} \\ z_{in2} = b_2 + x_1 w_{21} + x_2 w_{21} \end{cases}$

-) output of the hiddenlayer can be calculated by applying ~~hidden~~ activation function :-

$$z_i = f(z_{in}) = \begin{cases} 1 & \text{if } z_{in} \geq 0 \\ -1 & \text{if } z_{in} < 0 \end{cases}$$

-) Now calculate the net input to the output : $y_{in} = b_3 + z_1 v_1 + z_2 v_2$

-) ~~Calculate~~ apply activation function over net input y_{in} to calculate the output y:

-) if $t \neq y$, perform weight updation by updating ~~weights & bias of hidden layer~~

$$w_{ij}(new) = w_{ij}(old) + \alpha (t - z_{inj}) x_i$$

$$b_j(new) = b_j(old) + \alpha (t - z_{inj})$$

Inputs			Target										
x_1	x_2	1	(t)	z_{in1}	z_{in2}	z_1	z_2	y_{in}	y	w_{11}	w_{21}	b_1	w_{22}	b_2

BAM :- is a supervised learning model.

-> This can be used for pattern recognition and associative recall task

-> The network consists of two layers ; input layer and output layer with each layer containing equal number of neurons.

-> The connection b/w i/p & o/p is bidirectional, so the information can flow bidirectionally.

-> During the training process, the network adjusts its weights between neurons by presented i/p, o/p pairs.

-> Once the network is trained, it can be used to recall an output pattern from the given input pattern and vice versa.

-> BAM is mainly used to retrive a pattern from the given noisy or incomplete pattern.

Adaline (Adaptive linear neuron) :- (used for regression, classification & pattern recognition)

-> It is a single layered neural network with a linear activation function.

-> used for handling linearly separable classes and adjust its weights iteratively to minimize the error between the predicted and desired. using gradient descent algorithm.

Madaline (multiple Adaptive linear neuron) :-

-> multi-layer nn that consists of multiple Adalin units in layers.

-> used in pattern recognition where goal is to classify into multiple categories.

-> desined to handle non linear separable classes

Associative memory network :-

is a type of ANN that is designed to store and retrive Patterns from memory.

Auto-associative memory :- → stores & retrives patterns by comparing it with itself.
→ used for pattern completion.
→ Consists of single layered neurons that are fully connected.
→ Present with set of inputs during training and weights b/w neuron adj using Hebb

Hetro-associative memory :- → " " " " " Comparing with other patterns
→ used for pattern recognition.
→ Consists of multiple layers of neurons
→ Present with set of input & oupt Patterns during traing and Hebb is used to adj weight

Hopfield learning :-

→ Is a type of unsupervised learning which is also known as content addresable memory in which neurons are arranged in specific pattern to form recurrent neural network. and adjust the weights of the network to minimize the energy

Discrete Hopfield network :-

→ is a fully interconnected single-layer feedback network.
→ it is also a symmetric weighted network that takes two valued inputs :
binary (0 or 1) (or) bipolar (1 or -1)
→ usese discrete functions to minimize the energy of the network. used for discrete Problems.

Continous Hopfield network :-

→ A discrete H.f.n can be modified to C.H.N by assuming time as a Continous variable.
→ This is a type of recurrent neural network used for pattern recognition & optimization Problem.
→ uses Continues functions to minimize the energy of the network and update their values.
→ limitation - Can't store large no/of patterns & get stuck in local minima

Hebb, Hofield, Boltzman, Compitative → unsupervised learning

Delta learning rule :- (supervised)

-) is a gradient descent method that adjust the weights in order to minimize the diff b/w actual output and desired output.

-) used in single layer perception, and can also used in back propagation.

-) find error b/w actual o/p and desired o/p and use this error to adjust weights

credit assignment problem is a fundamental challenge in ANN, Particularly in deep neural networks with multiple layers of interconnected neurons.

uses backpropagation & reinforment learning

Properties of ANN :-

i) Non linearity (Can model nonlinear relationship b/w input & output variables)
ii) Adaptability (used for real time processing & continuous learning),
iii) Distributed computation (simple processing units works in parallel)
iv) Generalization (to predictions on new, unseen data)
v) Robustness (Can tolerate noise and errors in input data)
vi) Scalability (Can handle large amount of data / Can increase complexity of model)