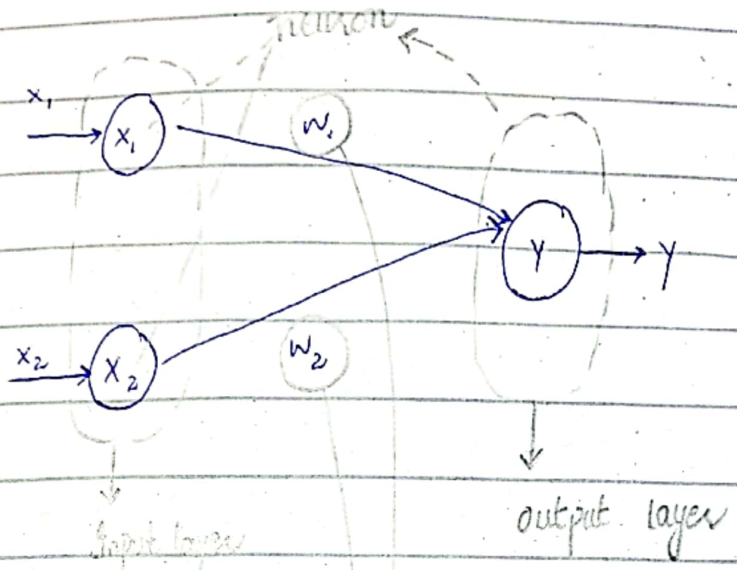
26-12-23
Tuesday

Artificial Neural Network



These weights are adjusted to get required output. If we don't get required output from certain weights we will re adjust them to get required output.

For weight adjustment (Gradient decent or back Propagation etc)

$$y_{in} = x_1 w_1 + x_2 w_2 \qquad y = f(y_{in})$$

↓

activation function: it may vary.

Learning

↳ Capability to learn.

(1) Parameter Learning : It updates the connecting weights in a neural net. (GD, BackP)

(2) Structure Learning : It focuses on the changing in network structure (which includes the number of processing elements as well as their connection types.)

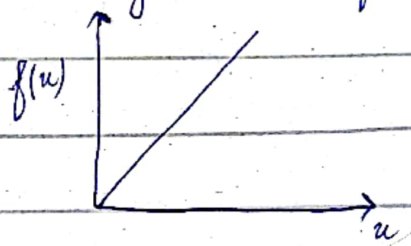
Activation functions

(1) Identity function : It is a linear function.

$$f(u) = u \quad \text{for all } u$$

The output here remains the same as input.

The input layer uses the identity activation function.

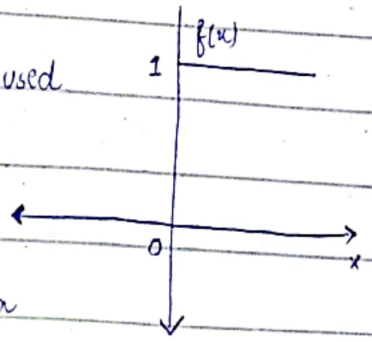


(2) Binary step function :

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

θ → threshold value

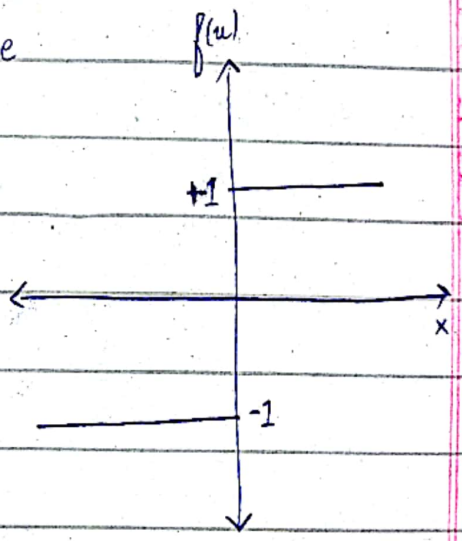
This function is most widely used in single layer nets. to convert the net input to an output that is a binary (1 or 0).



(3) Bipolar step function :

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

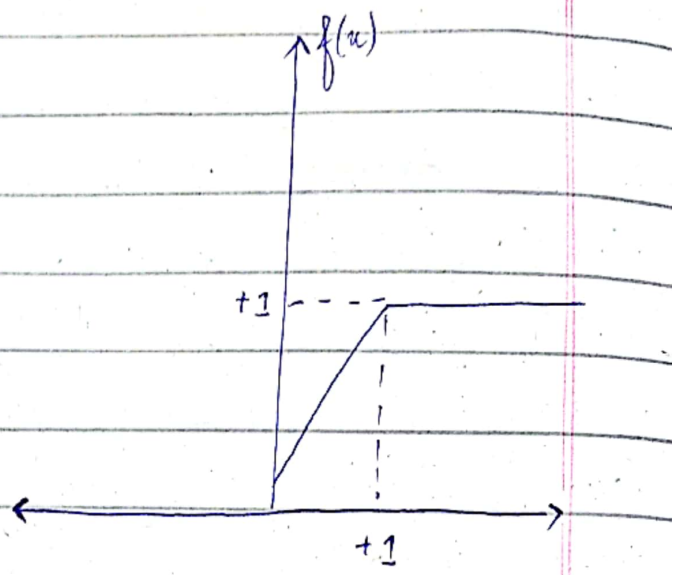
where θ represent the threshold value.



This function is also used in single layer nets to convert the net input to an output that is bipolar (+1 or -1).

(4) Ramp function:

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



(5) Sigmoidal functions:

The sigmoidal functions are widely used in back-propagation nets because of the relationship between the value of the derivative at that point which reduces the computational burden during training.

Sigmoidal functions are of 2 types:

(1) Binary sigmoid function

(2) Bipolar sigmoid function

(5.1) Binary Sigmoid function:

It is also termed as Logistic sigmoid function or unipolar sigmoid function. It is defined as:

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

Where λ is the steepness parameter. The derivative of this function is

$$f'(u) = \lambda f(u)\left[1 - f(u)\right]$$

Here the range of the sigmoid function is from 0 to 1.

(5.2) Bipolar sigmoid function:

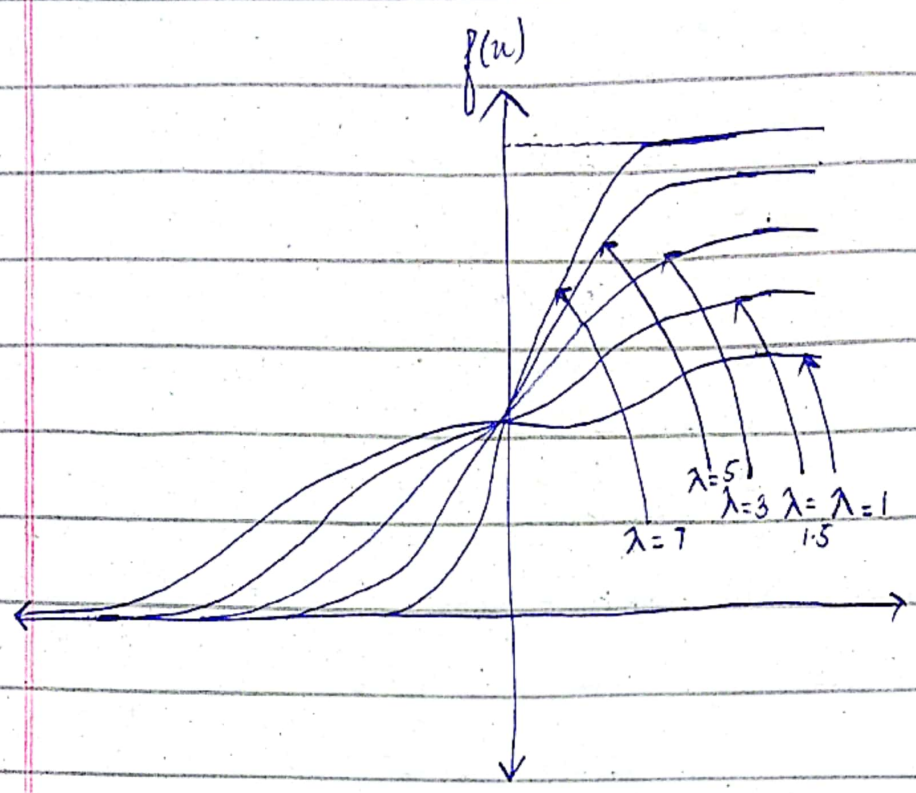
$$f(u) = \frac{2}{1+e^{-\lambda u}} - 1 = \frac{1-e^{-\lambda u}}{1+e^{-\lambda u}}$$

where λ is the steepness parameter.

Range of function $\longrightarrow$ -1 & +1

Derivative:

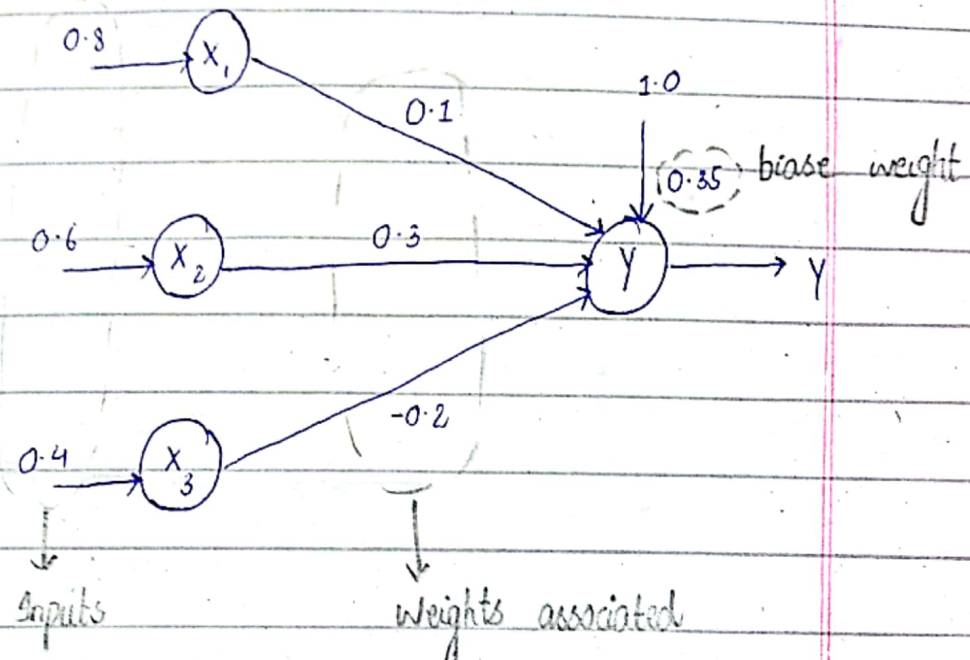
$$f'(u) = \frac{\lambda}{2}\left[1 + f(u)\right]\left[1 - f(u)\right]$$



λ is steepness.

Sigmoid Activation function

Numerical



(1) Binary Sigmoid :

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

y_{in} is the net input to that particular neuron.

(2) Bipolar:

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

The net input to the output neuron is :

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

(n → total no. of input neurons.)

$$= b + x_1 w_1 + x_2 w_2 + x_3 w_3$$

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

$$= 0.35 + 0.08 + 0.18 - 0.08 = \boxed{0.53}$$

Put in function :

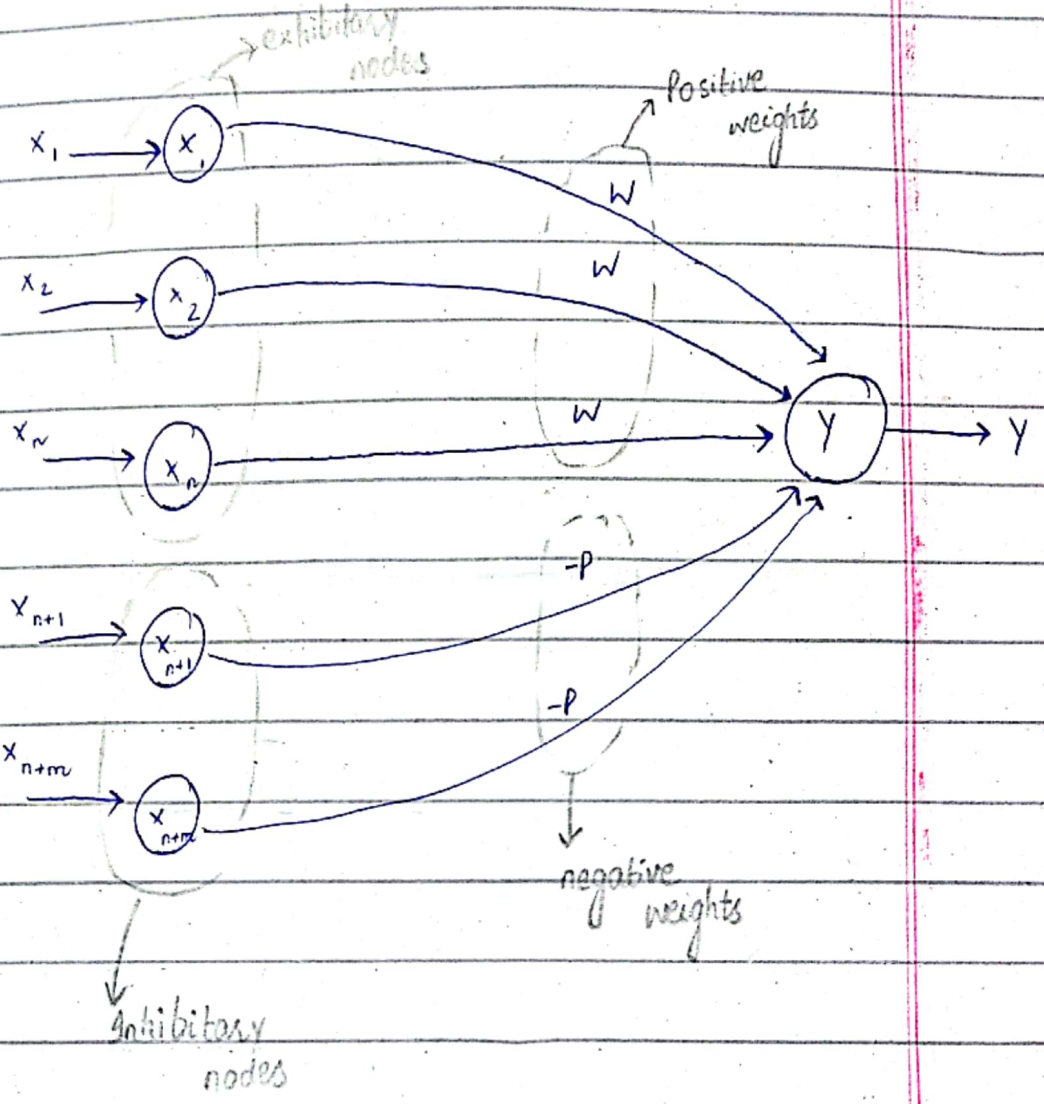
(1) Binary sigmoid Activation function :

$$y = f(y_{in}) = \frac{1}{1+e^{-y_{in}}} = \frac{1}{1+e^{-0.53}} = \boxed{0.625}$$

(2) Bipolar sigmoid Activation function :

$$y = f(y_{in}) = \frac{2}{1+e^{-y_{in}}} - 1 = \frac{2}{1+e^{-0.53}} - 1 = \boxed{0.259}$$

MC Culloch - Pitts Neuron



Since the firing of the output neuron is based upon the threshold, the activation function here is defined as

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

θ → threshold value

y_{in} → net input

The threshold value should satisfy the condition:

$\theta > nW - p$ → negative weight

n → no. of neurons

W → positive weight

Implement AND

x_1	x_2	y
1	1	1
1	0	0
0	1	0
0	0	0

↳ MP neuron doesn't have any algorithm so we have to do the analysis then we have to identify the weights as well as the threshold value.

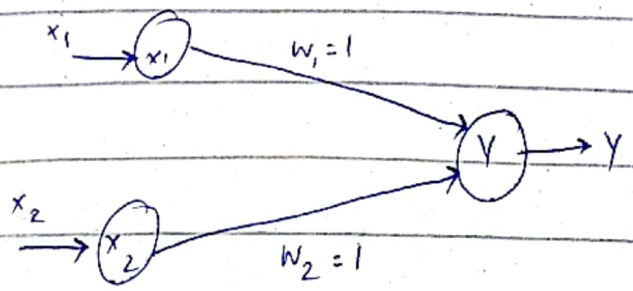
↳ Such that in this case the neuron will only fire when both the inputs are at high(1)

dition:

Hence,

first assume the weights be

$w_1 = 1$ and $w_2 = 1$



Check for all the input conditions in the truth table

$(1,1)$, $y_{in} = x_1 w_1 + x_2 w_2 = 1 \times 1 + 1 \times 1 = 2$

$(1,0)$, $y_{in} = x_1 w_1 + x_2 w_2 = 1 \times 1 + 0 \times 1 = 1$ | 1 | Net inputs at (Y)

$(0,1)$, $y_{in} = x_1 w_1 + x_2 w_2 = 0 \times 1 + 1 \times 1 = 1$

$(0,0)$, $y_{in} = x_1 w_1 + x_2 w_2 = 0 \times 1 + 0 \times 1 = 0$

↳ Now we have to find the threshold value so that neuron fires only when $x_1 = 1$ and $x_2 = 1$.

↓

0 should be greater than equal to (2)

Also be calculated :

$$\theta \geq nw - p$$

Here,

$n = 2$, $w = 1$ and $p = 0$

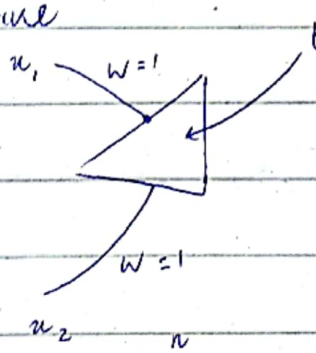
Substitute :

$$\theta \geq 2 \times 1 - 0 \Rightarrow \theta \geq 2$$

Thus the output of neuron Y can be written as ;

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

6 in lecture



$$\sum_{i=1}^{n} x_i w \geq t$$

$$x_1 w + x_2 w \geq t$$

for eg: $x_1, x_2 = 0, 1, 0$

$$0 + 0 \geq 2$$

$$0 \ngeq 2$$

$x_1, x_2 = 1, 1$

$1 \times 1 + 1 \times 1 \geq 2$

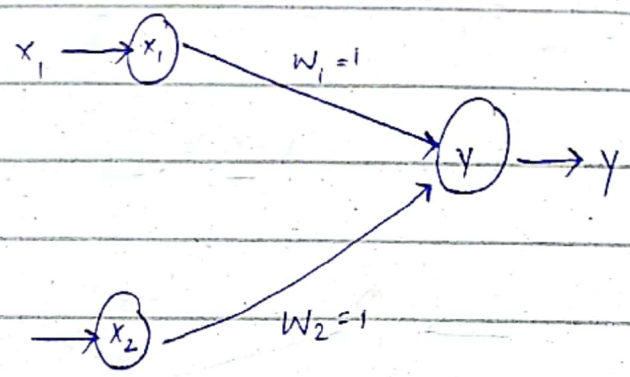
$1 + 1 \geq 2$

$2 \geq 2$

Implement AND NOT

x_1	x_2	y
0	0	0
0	1	0
1	0	1
1	1	0

fixing Condition ←



Initially weights are set to (1,1).

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

$(1,1), y_{in} = 1\times1 + 1\times1 = 2$

$(1,0), y_{in} = 1\times1 + 0\times1 = 1$

$(0,1), y_{in} = 0\times1 + 1\times1 = 1$

$(0,0), y_{in} = 0\times1 + 0\times1 = 0$

From the calculated net inputs, it is not possible to fire the neuron for inputs (1,0) only. Hence, these weights are not suitable.

↳ Modify the weights to $w_1 = 1$ & $w_2 = -1$

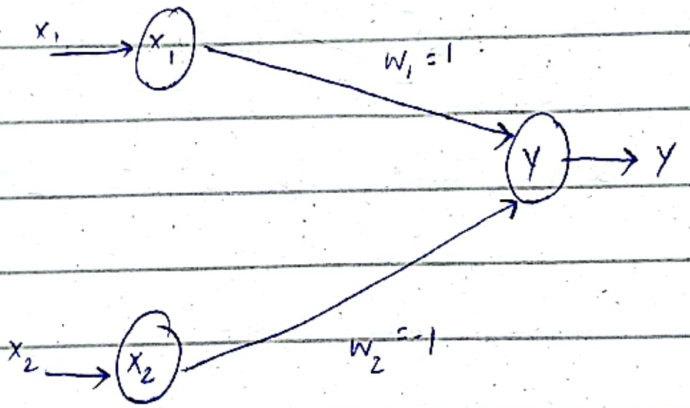
$$y_{in} = x_1 w_1 + x_2 w_2$$

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

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

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

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



From the calculations of net inputs, now it is possible to fire the neuron for inputs (1,0) only by fixing the threshold of (1) ie, $\theta \geq 1$ for Y unit.

Implement XOR function

	x_1	x_2	y
	0	0	0
fixing	0	1	1
	1	0	1
	1	1	0

↳ XOR is a non-linearly ~~set~~ separable function So it can't be implemented by simple logical function.

$y =$

eq: $x_1 \bar{x}_2 + \bar{x}_1 x_2$

$y = Z_1 + Z_2$

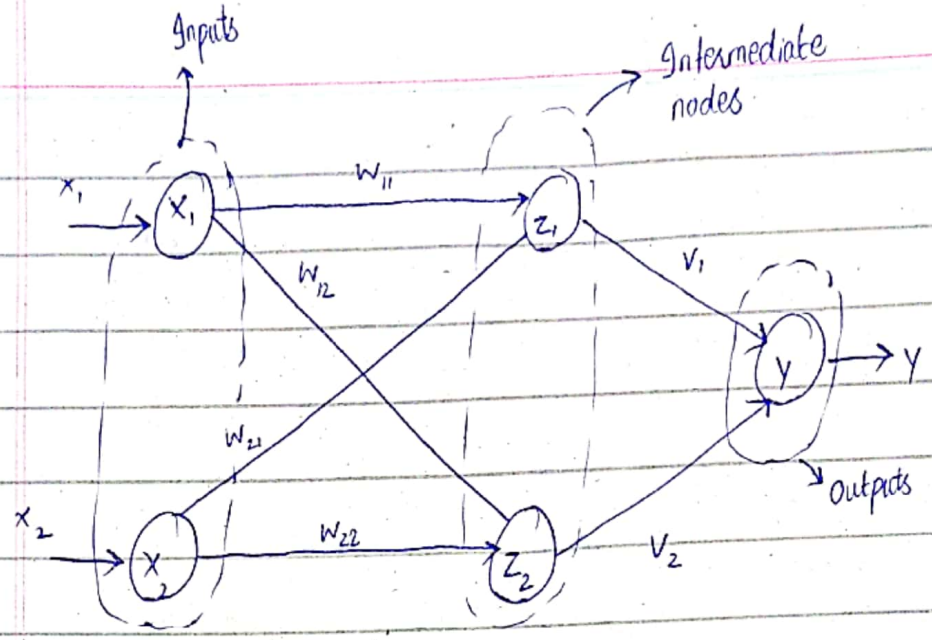
where,

$Z_1 = x_1 \bar{x}_2$ (function 1)

$Z_2 = \bar{x}_1 x_2$ (function 2)

$y = Z_1 (OR) Z_2$

↳ A single layer net is ~~not~~ sufficient to represent the XOR function. We need to add an intermediate layer is necessary.



First function : $Z_1 = x_1 \bar{x}_2$

x_1	x_2	Z_1
0	0	0
0	1	0
1	0	1
1	1	0

$w_{11}, w_{21} = 1$ → this will not work.

Calculate net inputs,

$(0,0)$, $Z_{in} = 0\times1 + 0\times1 = 0$

$(0,1)$, $Z_{in} = 0\times1 + 1\times1 = 1$

$(1,0)$, $Z_{in} = 1\times1 + 0\times1 = 1$

$(1,1)$, $Z_{in} = 1\times1 + 1\times1 = 2$

{ if $\theta = 1$ then it will fire in all three cases which we don't want.

Hence, it is not possible for us to obtain function Z_1 using these weights.

Now, assume

$W_{11} = 1, W_{21} = -1$ → will work

Calculate net inputs,

$(0,0)$, $Z_{1in} = 0 \times 1 + 0 \times -1 = 0$

$(0,1)$, $Z_{1in} = 0 \times 1 + 1 \times -1 = -1$

$(1,0)$, $Z_{1in} = 1 \times 1 + 0 \times -1 = 1$

$(1,1)$, $Z_{1in} = 1 \times 1 + 1 \times -1 = 0$

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

Second function = $Z_2 = \bar{x_1} x_2$

x_1	x_2	Z_2
0	0	0
0	1	1
1	0	0
1	1	0

$W_{12} = W_{22} = 1$ → Not work

Calculate ;

$(0,0)$, $z_{2in} = 0\times1 + 0\times1 = 0$

$(0,1)$, $z_{2in} = 0\times1 + 1\times1 = 1$

$(1,0)$, $z_{2in} = 1\times1 + 0\times1 = 1$

$(1,1)$, $z_{2in} = 1\times1 + 1\times1 = 2$

Hence, it is not possible to obtain function z_2 using these weights.

Assume weights, $w_{12} = -1$, $w_{22} = 1$ → will work

$(0,0)$, $z_{2in} = 0\times-1 + 0\times1 = 0$

$(0,1)$, $z_{2in} = 0\times-1 + 1\times1 = 1$

$(1,0)$, $z_{2in} = 1\times-1 + 0\times1 = -1$

$(1,1)$, $z_{2in} = 1\times-1 + 1\times1 = 0$

$\theta = 1$ & $w_{12} = -1$, $w_{22} = 1$

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

Third function $y = z_1$ (OR) z_2

x_1	x_2	y	z_1	z_2
0	0	0	0	0
0	1	1 ✓	0	1
1	0	1 ✓	1	0
1	1	0	0	1

$$y_{in} = z_1 v_1 + z_2 v_2$$

Assume, $v_1 = v_2 = 1$

Calculate net inputs;

$(0,0)$, $y_{in} = 0 \times 1 + 0 \times 1 = 0$ ✓ X

$(0,1)$, $y_{in} = 0 \times 1 + 1 \times 1 = 1$ ✓ ✓

$(1,0)$, $y_{in} = 1 \times 1 + 0 \times 1 = 1$ ✓ ✓

$(0,0)$, $y_{in} = 0 \times 1 + 0 \times 1 = 0$ ✓ X

↳ if i assume $0 = 0$

↳ if i assume $0 = 1$ → this is what we want

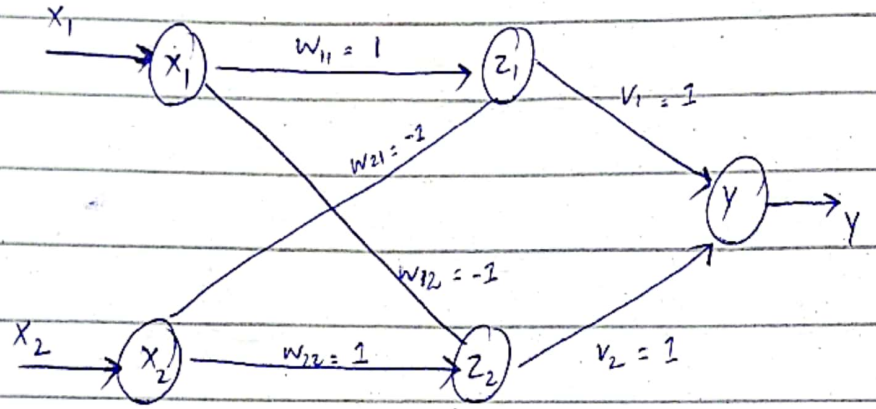
so final 0 value = (1) & $v_1 = v_2 = (1)$

This is how final XOR logic function looks like.

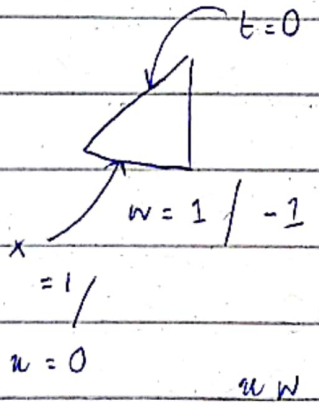
$W_{11} = 1$, $W_{21} = -1$

$W_{12} = -1$, $W_{22} = 1$

$V_1 = V_2 = 1$



NOT



X	Y
1	0
0	1

$u\,w \geq 0$

$(1)(1) \geq 0$

$1 \geq 0$ (Wrong because we don't want to fire when $u = 1$. So we have to change weight).

$(0)(-1) \geq 0$	$(1)(-1) \geq 0$
✓ $0 \geq 0$	$-1 \not\geq 0$ ✓

Hebbian Net Rule

Implement Logical AND function:

Inputs			Target
x_1	x_2	(b: bias)	y
1	1	1	1
1	-1	1	-1
-1	1	1	-1
-1	-1	1	-1

Initially the weights and bias are set to "0" then are modified afterwards for each condition.

$$w_1 = w_2 = b = 0$$

How to modify weights?

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

$$\Delta w_i = (x)(y) \rightarrow \text{Associated target}$$

input

First input $[x_1, x_2, b] = [1\ 1\ 1]$ and target = 1

Setting the initial weights as old weights and applying the Hebb rule, we get

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

$$\Delta w_i = x_i y$$

$$\Delta w_1 = 1 \times 1 = 1$$
$$\Delta w_2 = 1 \times 1 = 1$$
$$\Delta b = y = 1$$

$$w_1(new) = w_1(old) + \Delta w_1 = 0 + 1 = 1$$
$$w_2(new) = w_2(old) + \Delta w_2 = 0 + 1 = 1$$
$$b(new) = b(old) + \Delta b = 0 + 1 = 1$$

Second input $[x_1, x_2, b] = [1, -1, 1]$ & $y = -1$

$$\Delta w_1 = 1 \times -1 = -1$$
$$\Delta w_2 = -1 \times -1 = 1$$
$$\Delta b = y = -1$$

$$w_1(new) = w_1(old) + \Delta w_1 = 1 + (-1) = 0$$
$$w_2(new) = w_2(old) + \Delta w_2 = +1 + 1 = 02$$
$$b(new) = b(old) + \Delta b = 1 - 1 = 0$$

Third input $[x_1, x_2, b] = [-1, 1, 1]$ & $y = -1$

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

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

$\Delta b = y = -1$

$w_1(new) = w(old) + \Delta w_1 = 0 + 1 = 1$

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

$b(new) = b_{(old)} + \Delta b = 0 - 1 = -1$

Fourth input $[x_1, x_2, b] = [-1, -1, 1]$ & $y = -1$

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

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

$\Delta b = y = -1$

$w_1(new) = w(old) + \Delta w_1 = 1 + 1 = 2$

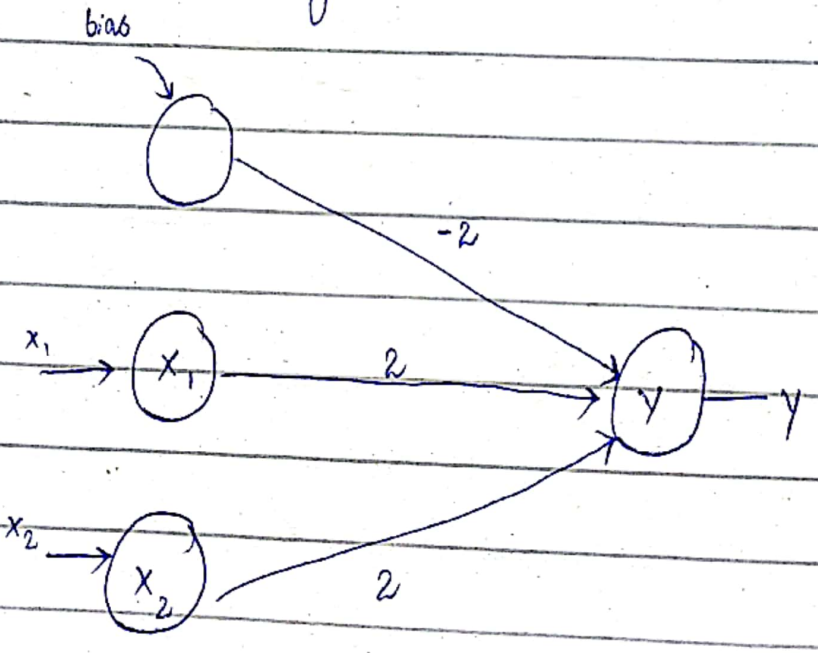
$w_2(new) = w_2(old) + \Delta w_2 = 1 + 1 = 2$

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

Inputs				Weight Changes			Weights		
x_1	x_2	b	y	Δw_1	Δw_2	Δb	w_1 (0)	w_2 (0)	b (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

↓ final weights

Heb Network for AND



Perceptron Learning Rule

testing:

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

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

training:

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

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

↳ Weight Adjustments

if $y \neq t$ then

$$w_i(new) = w_i(old) + \alpha t x_i$$

$$b(new) = b(old) + \alpha t$$

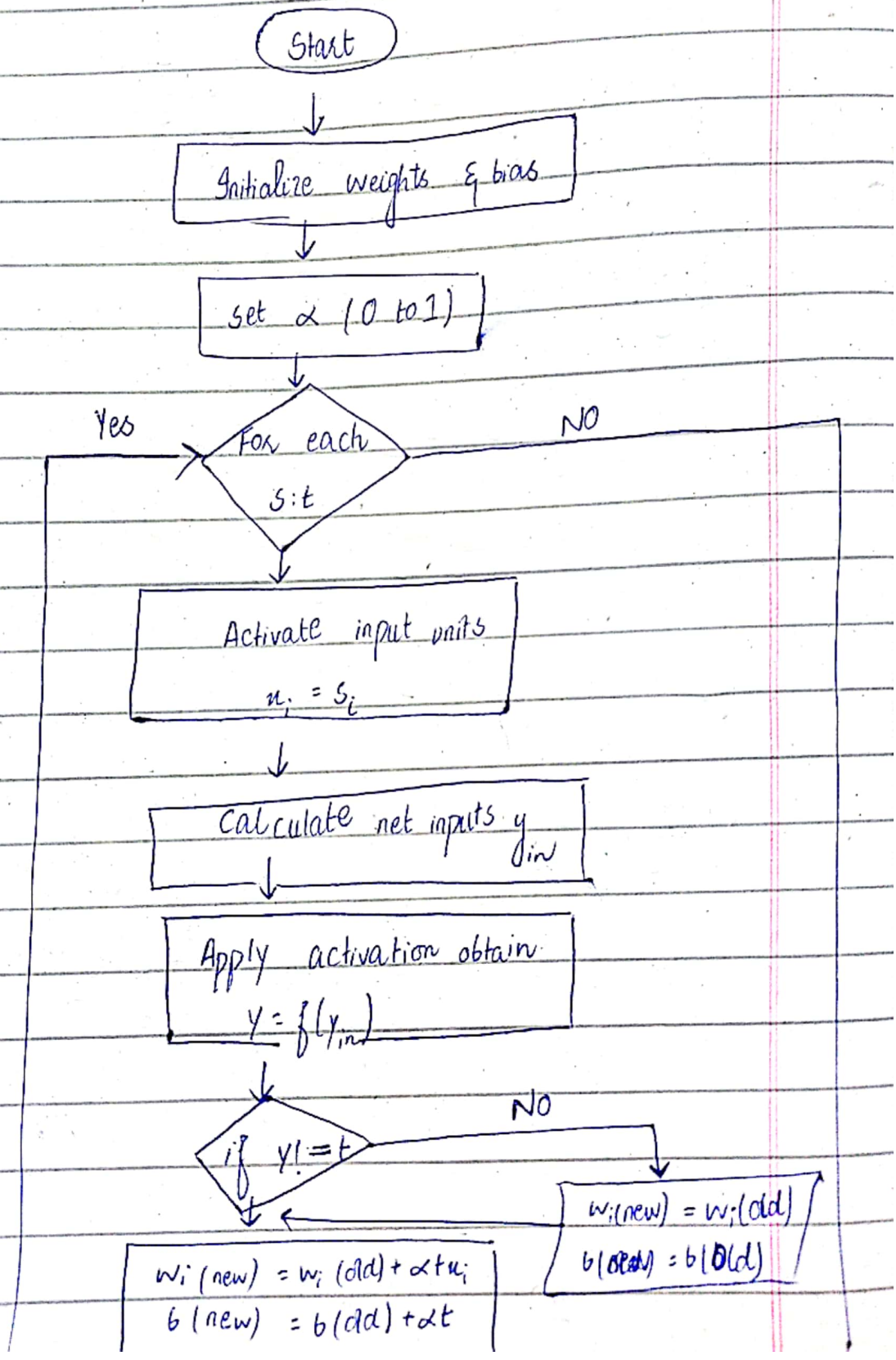
else:

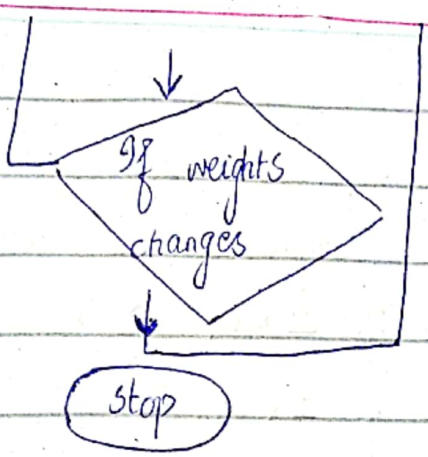
$$w_i(new) = w_i(old)$$

$$b(new) = b(old)$$

↳ α is the learning rate $\alpha (0 < \alpha \leq 1)$

↳ Train the network untill there is no weight change.



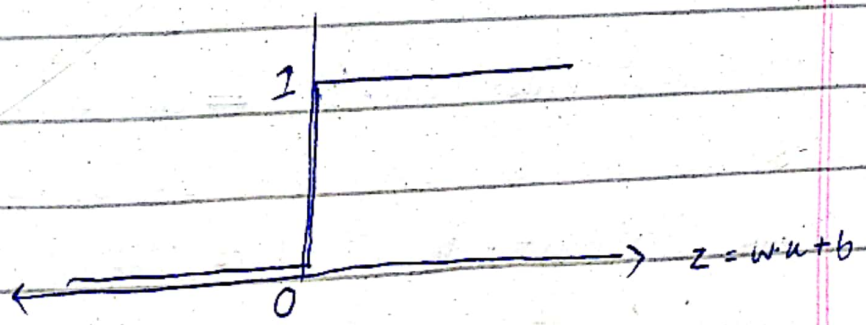


Activation function of a perceptron

Let : $Z = w.u + b$

Activation : $a = f(z) = \begin{cases} 0 & \text{if } z \leq 0 \\ 1 & \text{if } z > 0 \end{cases}$

Activation function of a perceptron is a step function.



Book

Perceptron :

Similar to McCulloch-pitts neuron

Perceptron is a single layer network.

$$\begin{cases} 1 \text{ if } \sum x_i w_i \geq \text{threshold } (t) \\ -1 \text{ if } \sum x_i w_i < t \end{cases}$$

Procedure → perceptron undergoes learning, correct result is assigned to it, change weights to reduce error.

Δw_i be the adjustment of weight.

$$\Delta w_i = \underbrace{C}_{\text{learning constant}} \left(\underbrace{d}_{\text{desired output}} - \underbrace{\text{sign}}_{\text{step function}} \left(\underbrace{\sum x_i w_i}_{\text{output value}} \right) \right) x_i$$

As, output values can be ~~all~~ only -1 or 1 so difference can only be 0, 2 or -2 & Δw_i can either be 0 or 2cx in case output is -1 while we desire 1. If actual value is 1 and desired is -1, decrement weight by $2c x_i$.

Perception did not work on data that is not linearly separable. eg (XOR)

11.2.21 - Example

Each class R_i has a discriminant func g_i measuring its membership in that region.

Boundary Case;

$$g_i(x) = g_j(x) \text{ or } g_i(x) - g_j(x) = 0$$

If linearly sepesable, then discriminant function seperats region into a straight line. This function can be min distance from the cartesian center of each of region.

Linear function computation:

$$f(net) = f(w_1x_1 + w_2x_2 + w_3 \underbrace{x\,1}_{\text{bias}}), \text{ where}$$

$f(x)$ can be $+1, -1$, this thresholding is called linear bipolar thresholding.

Bias serves to shift the $f(x)$ to horizontal axis.

$$[.75, .5, -.6]$$

$$f(net)^1 = f(.75 \times 1 + .5 \times 1 - .6 \times 1)$$
$$= f(0.65)$$
$$= 1$$

$$W^2 = W^1$$

$$f(net)^2 = f(.75 \times 9.4 + .5 \times 6.4 - .6 \times 1)$$
$$= f(9.65)$$
$$= (1)$$

↓ This is not our required output so we have to reduce error.

Adjust weight (W^3)

Learning Constant

$$W^t = W^{t-1} + (C)\left(d^{t-1} - \text{Sign}\left(W^{t-1} \times X^{t-1}\right)\right) X^{t-1}$$

$$W^3 = W^2 + C\left(d^2 - \text{sign}\left(W^2 X^2\right)\right) X^2$$
$$= W^2 + \underbrace{0.2}_{\text{let}} (-1-1) X^2$$

$$= \begin{bmatrix} 0.75 \\ 0.50 \\ -0.60 \end{bmatrix} - 0.4 \begin{bmatrix} 9.4 \\ 6.4 \\ 1.0 \end{bmatrix} = \begin{bmatrix} -3.01 \\ -2.06 \\ -1.00 \end{bmatrix}$$

$$\begin{bmatrix} 0.75 \\ 0.50 \\ -0.60 \end{bmatrix} - \begin{bmatrix} 3.76 \\ 2.56 \\ 0.4 \end{bmatrix} = \begin{bmatrix} -3.01 \\ -2.06 \\ -1.00 \end{bmatrix}$$

$$f(net)' = f(-3.01 \times 1 + (-2.06)(1) - (1.00)(1))$$

$$= f(-3.01 - 2.06 - 1)$$

$$= f(-6.07)$$

Now we consider the third data point with the newly adjusted weights:

$$f(net)^3 = f((-3.01 \times 2.5 + (-2.06)(2.1) + (-1.00)(1)) =$$

$$= f(-7.525 - 4.326 - 1)$$

$$= f(-12.851)$$

$$= \boxed{-1}$$

This is not required so we have to adjust weights again.

video.

AND Gate - Perceptron Training Rule

A	B	A ∧ B
1	1	1
1	0	0
0	1	0
0	0	0

$w_1 = 1.2$, $w_2 = 0.6$ Threshold = 1 & $C = 0.5$

(1) $w_i x_i = 1.2 \times 1 + 0.6 \times 1 = 1.2 + 0.6 = 1.8$

$= (1)$

(2) $w_i x_i = 1.2 \times 1 + 0.6 \times 0 = 1.2 + 0 = 1.2$

$= (1) \times$ wrong

Weight Adjustment

$$w^3 = w^{3-1} + C\left(d^{3-1} - \text{sign}\left(x_i w_i\right)\right) x_i$$

$$= \begin{bmatrix} 1.2 \\ 0.6 \end{bmatrix} + 0.5 \left(0 - 1\right) \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1.2 \\ 0.6 \end{bmatrix} + (-0.5) \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1.2 \\ 0.6 \end{bmatrix} - \begin{bmatrix} 0.5 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.7 \\ 0.6 \end{bmatrix}$$

Modified weights

$w_1 = 0.7$, $w_2 = 0.6$

(1) $w_i x_i = 0.7 \times 1 + 0.6 \times 1 = 0.7 + 0.6 = 1.3$

$= 1$ ✓

(2) $w_i x_i = 0.7 \times 0 + 0.6 \times 1 = 0 + 0.6 = 0.6$

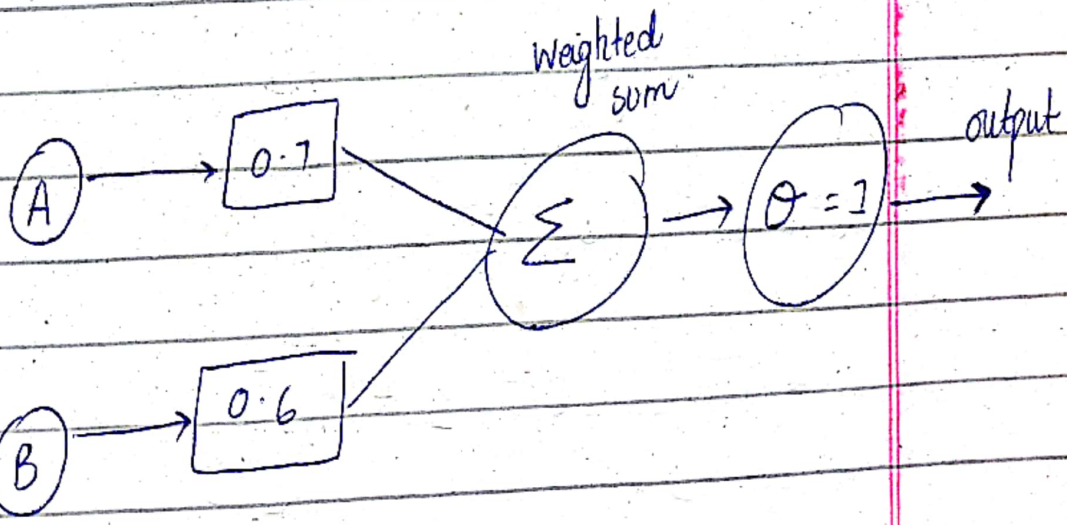
$= 0$ ✓

(3) $w_i x_i = 0.7 \times 1 + 0.6 \times 0 = 0.7 + 0 = 0.7$

$= 0$ ✓

(4) $w_i x_i = 0.7 \times 0 + 0.6 \times 0 = 0 + 0 = 0$

$= 0$ ✓



OR Gate - Perceptron

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

$w_1 = 0.6$ & $w_2 = 0.6$ threshold = 1 & $c = 0.5$

(1) $w_i x_i = 0 \times 0.6 + 0 \times 0.6 = 0 = \boxed{0}$ ✓

(2) $w_i x_i = 0 \times 0.6 + 1 \times 0.6 = 0 + 0.6 = 0.6 = \boxed{0}$ X

Weight Adjustment

$$W^3 = W^{3-1} + c\left(d^{3-1} - \text{sign}\left(w_i x_i\right)\right) x_i$$

$$= \begin{bmatrix} 0.6 \\ 0.6 \end{bmatrix} + 0.5\,(1 - 0) \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.6 \\ 0.6 \end{bmatrix} + 0.5 \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.6 \\ 0.6 \end{bmatrix} + \begin{bmatrix} 0 \\ 0.5 \end{bmatrix} = \begin{bmatrix} 0.6 \\ 1.1 \end{bmatrix}$$

$w_1 = 0.6$, $w_2 = 1.1$

(1) $w_i x_i = 0.6 \times 0 + 1.1 \times 0 = 0 + 0 = 0 = \boxed{0}$ ✓

(2) $w_i x_i = 0.6 \times 0 + 1.1 \times 1 = 0 + 1.1 = 1.1 = \boxed{1}$ ✓

(3) $w_i x_i = 0.6 \times 1 + 1.1 \times 0 = 0.6 + 0 = 0.6 = \boxed{0}$ ✗

Weight Adjustment

$$W^4 = W^{4-1} + C\left(d^{4-1} - \text{sign}(x_i w_i)\right) x_i$$

$$= \begin{bmatrix} 0.6 \\ 1.1 \end{bmatrix} + 0.5\left(1 - 0\right)\begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0.6 \\ 1.1 \end{bmatrix} + 0.5 \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0.6 \\ 1.1 \end{bmatrix} + \begin{bmatrix} 0.5 \\ 0 \end{bmatrix} = \begin{bmatrix} 1.1 \\ 1.1 \end{bmatrix}$$

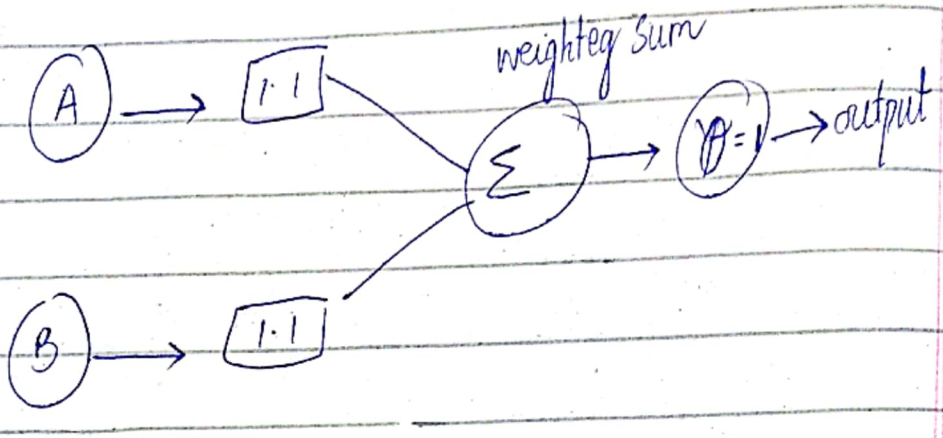
$w_1 = 1.1$, $W_2 = 1.1$

(1) $w_i x_i = 1.1 \times 0 + 1.1 \times 0 = 0 + 0 = 0 = \boxed{0}$ ✓

(2) $w_i x_i = 1.1 \times 0 + 1.1 \times 1 = 0 + 1.1 = 1.1 = \boxed{1}$ ✓

(3) $w_i x_i = 1.1 \times 1 + 1.1 \times 0 = 1.1 + 0 = 1.1 = \boxed{1}$ ✓

(4) $w_i x_i = 1.1 \times 1 + 1.1 \times 1 = 1.1 + 1.1 = 2.2 = \boxed{1}$ ✓



XOR

We can implement it using one / single perceptron.

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

~~$y = \bar{x}_1 x_2 + x_1 \bar{x}_2$~~

$$y = x_1 \bar{x}_2 + \bar{x}_1 x_2$$

$$y = z_1 + z_2$$

$$z_1 = x_1 \bar{x}_2$$

$$z_2 = \bar{x}_1 x_2$$

$$y = z_1 \text{ OR } z_2$$

$Z_1 = x_1 \bar{x}_2$

x_1	x_2	Y
0	0	0
0	1	0
1	0	1
1	1	0

$w_1 = 1$, $w_2 = 1$ thresh = 1 & C = 1.5

$w_{ij} x_i = (1)(0) + (1)(0) = 0 = \boxed{0}$ ✓

$= (1)(0) + (1)(1) = 1 = \boxed{1}$ ✗

Adjust weights

$$w^t = W_{ij}^{t-1} + C\left(d^{t-1} - \text{sign}(x_i w_i)\right) x_i$$

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

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

$$= \begin{bmatrix} 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 1.5 \end{bmatrix} = \begin{bmatrix} 1 \\ -0.5 \end{bmatrix}$$

$w_1 = 1$, $w_2 = -0.5$

(2) $w_{ij} x_i = (1)(0) + (1)(-0.5) = 0 - 0.5 = -0.5 = \boxed{0}$ ✓

(3) $= (1)(1) + (0)(-0.5) = 1 = \boxed{1}$ ✓

(4) $= (1)(1) + (1)(-0.5) = 1 - 0.5 = 0.5 = \boxed{0}$ ✓

Second function:

$Z_2 = \bar{x_1} x_2$

x_1	x_2	y
0	0	0
0	1	1
1	0	0
1	1	0

$w_1 = w_2 = 1$ thresh = 1 , $C = 1.5$

① $y_{in} = \sum w_{ij} \cdot x_i = (1)(0) + (1)(0) = 0 = \boxed{0}$ ✓

② $(1)(0) + (1)(1) = 0 + 1 = \boxed{1}$ ✓

③ $(1)(1) + (1)(0) = 1 = \boxed{1}$ ✗

Weight Adjust.

$$w(new) = w^T = W^{T-1} + C\left(d^{t-1} - sign(x_i w_{ij})\right) x_i$$

$$= \begin{bmatrix} 1 \\ 1 \end{bmatrix} + 1.5 \left(0 - 1\right) \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 \\ 1 \end{bmatrix} - 1.5 \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 1.5 \\ 0 \end{bmatrix} = \begin{bmatrix} -0.5 \\ 1 \end{bmatrix}$$

$w_1 = 0.5$, $w_2 = 1$ thresh = 1 $c = 1.5$

(1) $(0)(-0.5) + (0)(1) = 0 + 0 = \boxed{0}$ ✓

(2) $(0)(-0.5) + (1)(1) = 1 = \boxed{1}$ ✓

(3) $(1)(-0.5) + (0)(1) = -0.5 = \boxed{0}$ ✓

(4) $(1)(-0.5) + (1)(1) = -0.5 + 1 = 0.5 = \boxed{0}$ ✓

Third function = $Y = Z_1$ or Z_2

X_1	X_2	Z_1	Z_2	Y
0	0	0	0	0
0	1	0	1	1
1	0	1	0	1
1	1	0	0	0

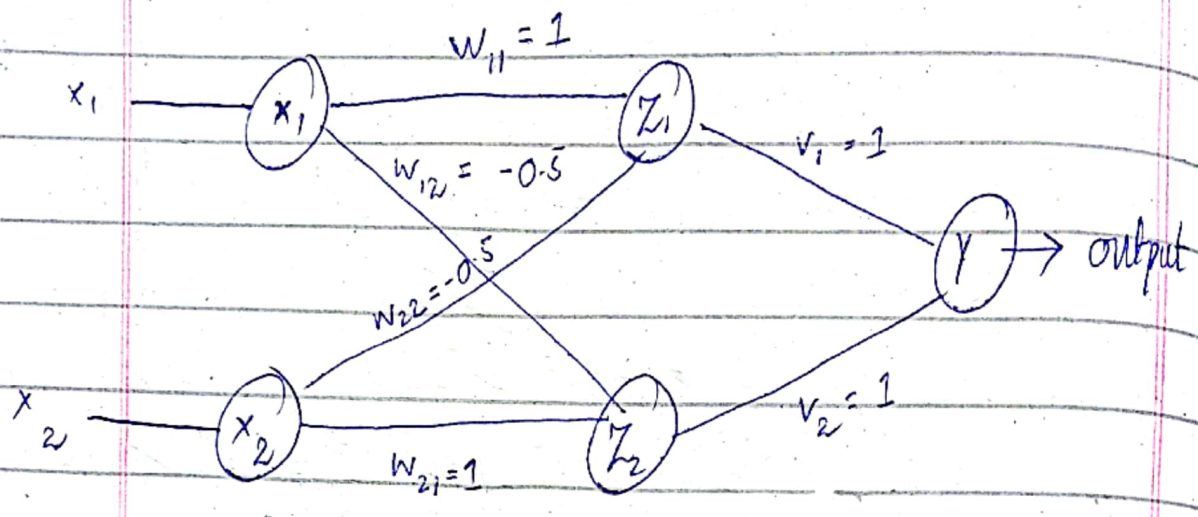
$v_1 = v_2 = 1$ thresh = 1 & $c \geq 0.5$

(1) $y_{in} = w_{ij} x_i = (1)(0) + (1)(0) = 0 = \boxed{0}$ ✓

(2) $= (0)(1) + (1)(1) = 1 = \boxed{1}$ ✓

(3) $(1)(1) + (0)(1) = 1 = \boxed{1}$ ✓

(4) $(0)(1) + (0)(1) = 0 = \boxed{0}$ ✓



Gradient Descent and Delta Rule

: Perceptron rules find a successful weights vector when the training examples are linearly separable, but it can fail to converge if the examples are not linearly seperable.

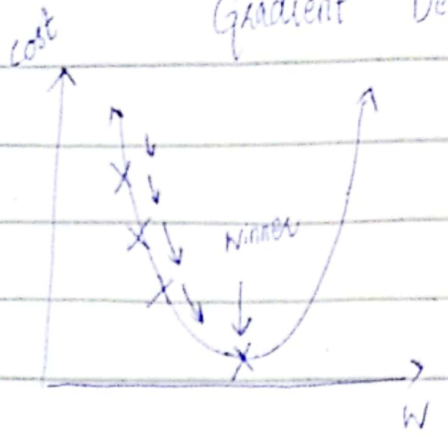
: A second training rule, called the delta rule, is designed to overcome this difficulty.

: If the training examples are not linearly separable, the delta rule converges towards a best-fit approximation to target concept.

: The key idea behind the delta rule is to use gradient descent to search the hypothesis space of possible weight vectors to find the weights that best fit the training examples.

: This rule is important because gradient descent provides the basis for the Backpropagation algorithm, which can learn network with many interconnected units.

Gradient Descent



Training error of a hypothesis:

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

dif/

where D is the set of training eg. t_d is the target output for training example d, and o_d is the output of the linear unit for training example d.

∴ This vector derivative is called the gradient of E with respect to $\vec{w}$, written as $\nabla E(\vec{w})$.

$$\nabla E(\vec{w}) = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n} \right]$$

$$\vec{w} = \vec{w} + \Delta \vec{w}$$

$$\Delta \vec{w} = -\eta \nabla E(\vec{w})$$

The training rule can also be written in its components form.

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

where

$$\Delta w_i = (-\eta) \frac{\partial E}{\partial w_i}$$

η: learning constant — یہ جتنا کم ہوگا اتنا اچھی بات ہے

eg: $\eta = 0.001$

Derivation

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial w_i} (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_{d \in D} 2 (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d)$$

(o_d: actual output, $y = w_i x_i$)

$$= \sum_{d \in D} (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x}_d)$$

(t_d: constant; all weights w are constant with respect to w_i but weight w_i with respect to w_i will be -1 as a result)

$$\frac{\partial E}{\partial w_i} = \sum_{d \in D} (t_d - o_d)(-x_{id})$$

Substitute the values

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

(-ve , -ve → +)

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d) x_{id}$$

1→ The key practical difficulties in applying gradient descent are :-

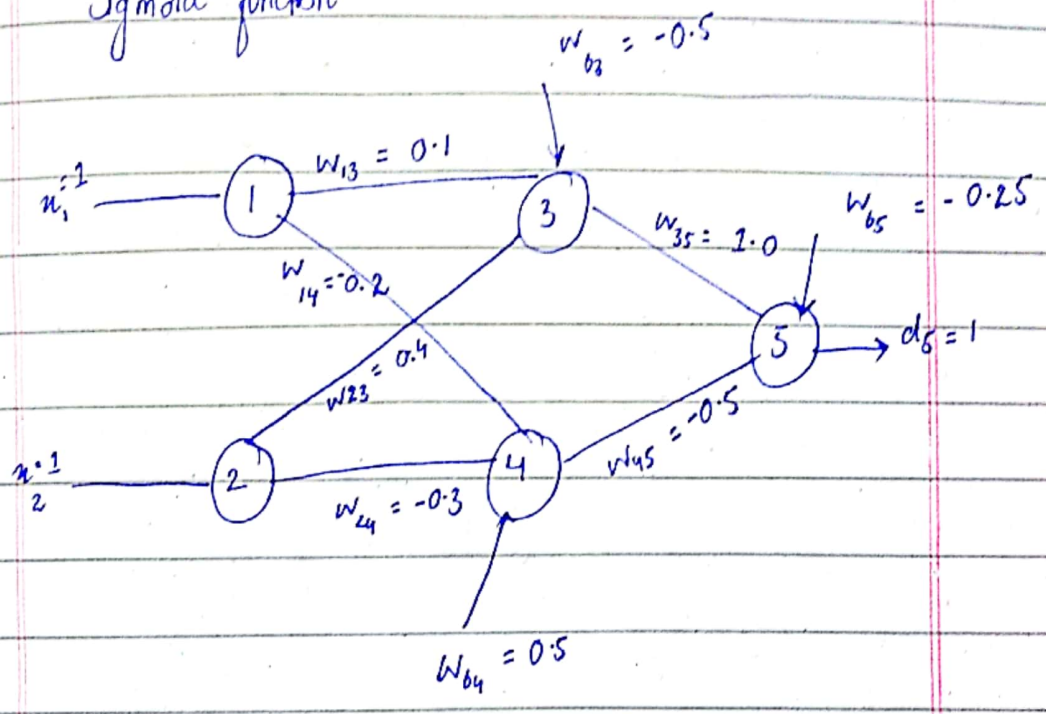
(1) Converging to a local minimum can sometimes be quite slow (i.e it can require many thousands of gradient descent steps)

(2) If there are multiple local minima in the error surface, then there is no guarantee that the procedure will find the global minimum.

Numerical Back Propagation

$\Delta W_{35} = ?$, $\delta_4 = ?$, $c = 0.1$, $\beta = 0.9$

Sigmoid function



$$O_3 = x_i W_i = x_1 W_{13} + x_2 W_{23} + W_{b3}$$

$$= (1)(0.1) + (1)(0.4) - 0.5$$

$$= (0.1) + (0.4) - 0.5 = 0$$

$$O_3 = \frac{1}{1+e^{-net}} = \frac{1}{1+e^{0}} = \boxed{0.5}$$

$$O_4 = x_1 W_{14} + x_2 W_{24} + W_{b4} = (1)(-0.2) + (1)(-0.3) + 0.5$$

$$= -0.2 - 0.3 + 0.5 = 0$$

$$O_4 = \frac{1}{1+e^{-net}} = \frac{1}{1+e^{0}} = \boxed{0.5}$$

$$O_5 = O_3 W_{35} + O_4 W_{45} + W_{65}$$

$$= (0.5)(1.0) + (0.5)(-0.5) + (-0.25)$$

$$= (0.5) - (0.25) = (0.25)$$

$$= 0.25 - 0.25 = 0$$

$$O_5 = \frac{1}{1+e^{-net}} = \frac{1}{1+e^{0}} = \boxed{0.5}$$

$$\delta_5 = e_5 \cdot O_5 (1 - O_5)$$

$$= (1 - 0.5) \cdot 0.5 (1 - 0.5)$$

$$= (0.5) \times (0.25)$$

$$= \boxed{0.125}$$

$$\delta_4 = \delta_5 W_{45} \cdot O_4 (1 - O_4)$$

$$= (0.125)(-0.5) \cdot 0.5 (1 - 0.5)$$

$$= -0.0625 \times 0.25$$

$$= \boxed{-0.015625}$$

$$\Delta W_{35} = C \delta_5 O_3$$

$$= (0.1)(0.125)(0.5)$$

$$= 0.00625$$

∴ If update required:

$$\Delta W_{35}(new) = W_{35}(old) + \Delta W_{35}$$

$$= (1.0)(0.00625)$$

$$= \boxed{0.00625}$$