Multi-layer Perceptrons:

The network consists of a set of sensory units (source nodes) that constitute the input layer, one or more hidden layers of computation nodes & an output layer of computation nodes. The input signal propagates through the network in a forward direction, on a layer-by-layer basis. These neural networks are commonly referred to as multilayer perceptrons (MLPs).

MLPs solve some difficult and diverse problems by training them in a supervised manner with a highly popular algorithm known as the error-back-propagation algorithm. This algorithm is based on the error-correction learning rule.

Basically error back-propagation learning consists of two passes: a forward pass & a backward pass.

In the forward pass an activity pattern (input vector) is applied to the sensory nodes of the network & its effect propagates through the network layer by layer. Finally a set of outputs is produced as the actual response of the network

During the forward pass the synaptic weights of the networks are all fixed.

During the backward pass on the other hand, the synaptic weights of are all adjusted in accordance with an error-correction rule. Specifically the actual response of the network is subtracted from a desired (target) response to produce an error signal.; this error signal is then propagated backward through the network against the direction of synaptic connections - hence the name "error back-propagation"

Back-Propagation Algorithm:

The error signal at the output of neuron j at iteration n is defined by

$$e_j(n) = d_j(n) - y_j(n) \quad \text{neuron } j \text{ is an output node} \quad \text{——— (1)}$$

We define the instantaneous value of the error energy for neuron j as $\left(\frac{1}{2}e_j^2(n)\right)$

The instantaneous value $\mathcal{E}(n)$ of the total error energy is obtained by summing $\frac{1}{2}e_j^2(n)$

Thus $$\mathcal{E}(n) = \frac{1}{2}\sum_{j \in C} e_j^2(n) \quad \text{——— (2)}$$

Where the set C includes all the neurons in the output layer of the Network.

Let N denote the total number of patterns Contained in the training set.

The average squared error energy is obtained by summing $\mathcal{E}(n)$ over all n & then Normalizing w.r.t to the set size N

$$\mathcal{E}_{av} = \frac{1}{N}\sum_{n=1}^{N} \mathcal{E}(n) \quad \text{——— (3)}$$

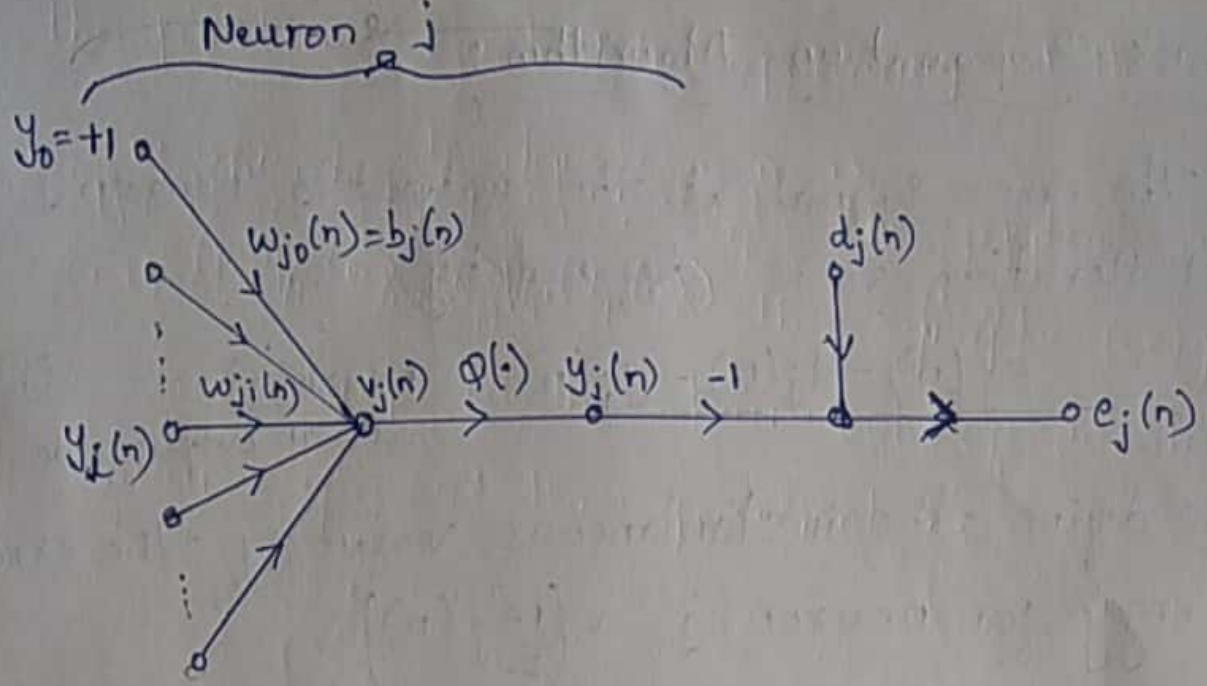


Figure: Signal flow graph highlighting the details of output neuron j.

The induced local field $v_j(n)$ produced at the input of the activation function associated with neuron j is therefore.

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n)\, y_j(n) \quad \text{——} (4)$$

where m is the total number of inputs applied to neuron j

The synaptic weight w_{j0} (corresponding to the fixed input $y_0 = +1$) equals the bias b_j applied to the neuron

The function signal $y_j(n)$ appearing at the output of neuron j at iteration n is

$$y_j(n) = \varphi_j(v_j(n)) \quad \text{——} (5)$$

The back-propagation algorithm applies a correction $\Delta w_{ji}(n)$ to the synaptic weight $w_{ji}(n)$ which is proportional to the partial derivative $\partial E(n)/\partial w_{ji}(n)$.

According to the chain rule of calculus, we may express this gradient as

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = \frac{\partial E(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \cdot \frac{\partial v_j(n)}{\partial w_{ji}(n)} \quad \text{——} (6)$$

The partial derivative $\partial E(n)/\partial w_{ji}(n)$ represents a sensitivity factor, determining the direction of search in weight space for the synaptic weight w_{ji}.

Differentiating both sides of eqn (2) w.r.t $e_j(n)$ we get.

$$E(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n) \quad \text{——} (2) \rightarrow \text{diff}^{\text{e}} \; \frac{1}{2} \cdot 2 e_j(n)$$

$$\frac{\partial E(n)}{\partial e_j(n)} = e_j(n) \quad \text{——} (7)$$

Differentiating both side of eqn (1) w.r.t $y_j(n)$

$$e_j(n) = d_j(n) - y_j(n)$$

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1 \quad \text{—(8)}$$

differentiating $y_j(n) = \varphi_j(v_j(n))$ w.r.t to $v_j(n)$

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi_j'(v_j(n)) \quad \text{—(9)}$$

differentiating $v_j(n) = \sum w_{ji}(n)\, y_i(n)$ w.r.t $w_{ji}(n)$

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_i(n) \quad \text{—(10)}$$

Use eqn's (7) to (10) in (6)

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} = \frac{\partial \mathcal{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \cdot \frac{\partial v_j(n)}{\partial w_{ji}(n)}$$

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} = -e_j(n)\varphi_j'(n) = -e_j(n)\varphi_j'(v_j(n))\, y_i(n)$$

$$= e_j(n)\varphi_j'(v_j(n))\, y_i(n) \quad \text{—(11)}$$

The correction $\Delta w_{ji}(n)$ applied to $w_{ji}(n)$ is defined by delta rule.

$$\Delta w_{ji}(n) = -\eta \frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} \quad \text{—(12)} \quad \& \quad \Delta w_{ji}(n) = \eta \delta_j(n) y_i(n) \quad \text{—(13)}$$

Where the local gradient $\delta_j(n)$ is defined by

$$\delta_j(n) = -\frac{\partial \mathcal{E}(n)}{\partial v_j(n)} = -\frac{\partial \mathcal{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \quad \text{—(14)}$$

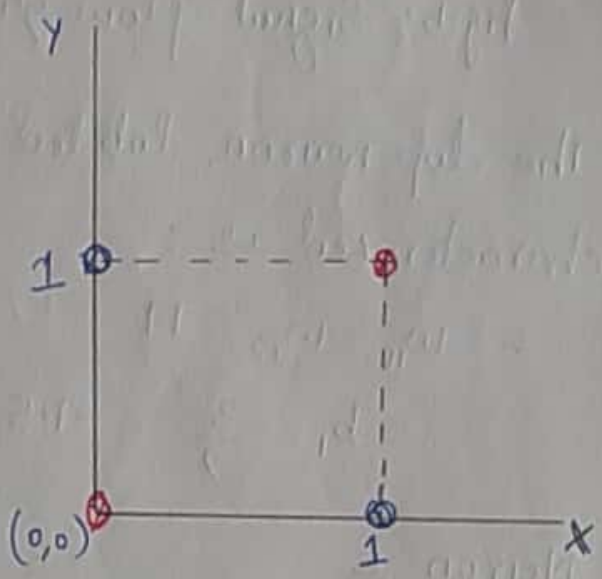
$$= e_j(n)\varphi_j'(v_j(n))$$

XOR Problem:

In single-layer perceptron there are no hidden neurons. Consequently, it cannot classify input patterns that are not linearly separable.

Non-linearly separable patterns are of common occurrence. This situation arises in the Exclusive OR (XOR) problem.

x_1	x_2	y	
0	0	0	⊗
0	1	1	×
1	0	1	×
1	1	0	⊗



⊗ → 0 patterns ⊗ → 1 Patterns

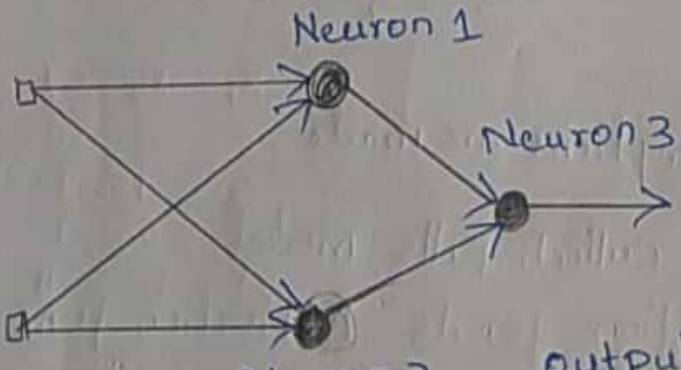


fig: (a) Architectural graph of Network for solving the XOR problem.

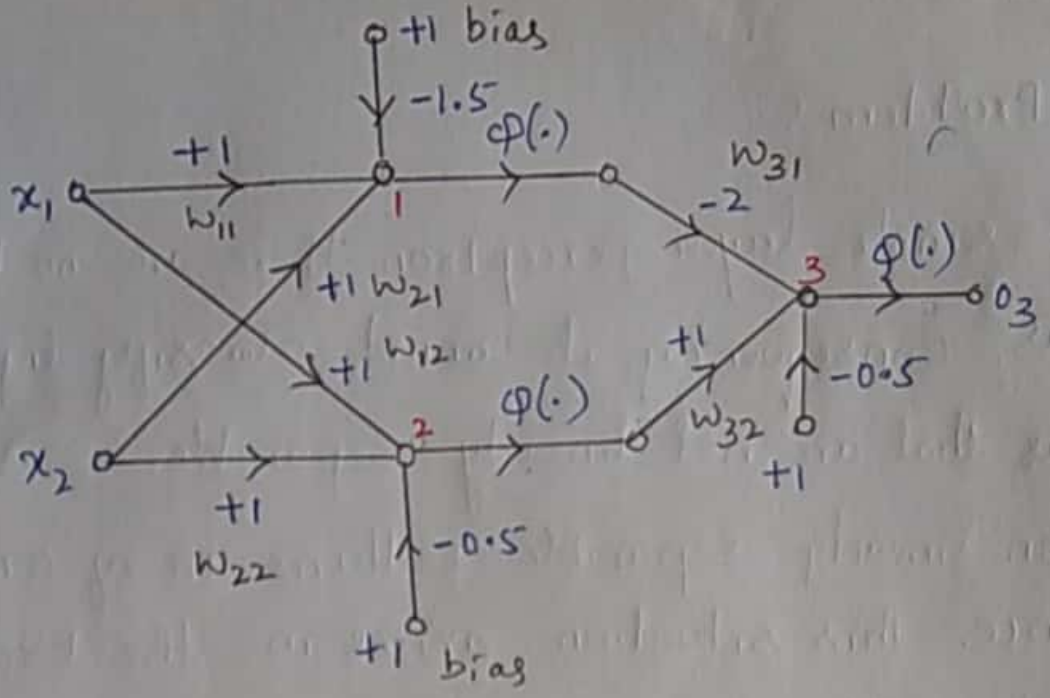


fig 5: Signal-flow graph of the network.

The top neuron, labeled 1 in the hidden layer, is characterized as:

$$w_{11} = w_{12} = +1$$

$$b_1 = -\frac{3}{2} = -1.5$$

Neuron 2

$$w_{21} = w_{22} = +1$$

$$b_2 = \frac{-1}{2} = -0.5$$

neuron 3

$$w_{31} = -2 \quad w_{32} = +1$$

$$b_3 = -\frac{1}{2} = -0.5$$

The following Assumptions are made

* Each Neurons - Mc Culloch pitts model uses threshold function for its Activation func
* Bits 0 and 1 are represented by levels 0 and +1 Respectively.

$X_1=0$ $X_2=0$ $w_{11}=w_{12}=+1$

$-1.5+0(1)+0(1)=0$

$-1.5<0 \Rightarrow$ output 0

X_1	X_2	y
0	0	0
0	1	1
1	0	1
1	1	0

$X_1=0$, $X_2=1$

$0(1)+1(1)-1.5=-0.5<0$

$-0.5<0 \Rightarrow$ output 0

$X_1=1$ $X_2=0$

$1(1)+0(1)-1.5=-0.5$

$-0.5<0 \Rightarrow$ output 0

$X_1=1$ $X_2=1$

$1(1)+1(1)-1.5=2-1.5$

$=0.5>0$ output $=1$

Neuron 1 : AND Gate

Neuron 2 : OR Gate

$X_1=0$ $X_2=0$ $w_{21}=w_{22}=1$

$0(1)+0(1)-0.5$

$=-0.5<0$

$\therefore$ output $=0$

$X_1=1$ $X_2=0$

$1(1)+0(1)-0.5$

$=1-0.5$

$0.5>0$

output $=1$

$X_1=0$ $X_2=1$ $w_{21}=w_{22}=1$

$0(1)+1(1)-0.5=0.5$

$0.5>0$ output $=1$

$X_1=1$ $X_2=1$

$1(1)+1(1)-0.5$

$2+0.5=1.5$

$1.5>0$ output $=1$

Neuron 1	Neuron 2	Neuron 3 (o/p)
0	0	0
0	1	1
1	1	0

$y_1=0 \quad y_2=0$

$0(-2)+0(1)-1(0.5)$

$=-0.5<0$ output=0

$y_1=0 \quad y_2=1$

$0(-2)+1(1)-1(0.5)$

$1-0.5=0.5>0$

output=1

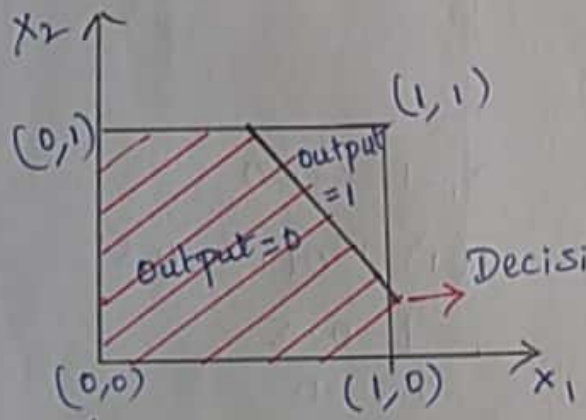
$y_1=1 \quad y_2=1$

$1(-2)+1(1)-1(0.5)$

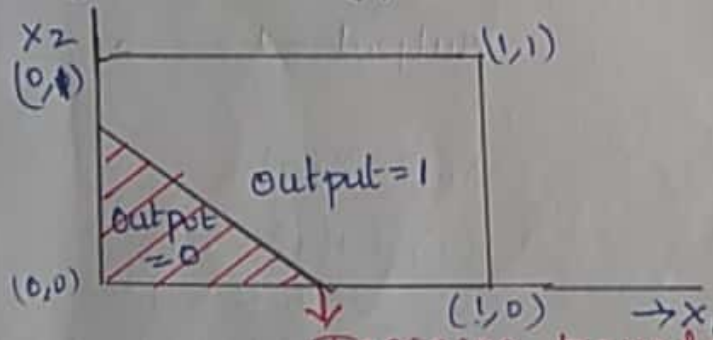
$-2+1-0.5=-1.5<0$ output=1

$x_1=0$	$x_2=0$	$y_1=0$	$y_2=0$	$y_3=0$
$x_1=0$	$x_2=1$	$y_1=0$	$y_2=1$	$y_3=1$
$x_1=1$	$x_2=0$	$y_1=0$	$y_2=1$	$y_3=1$
$x_1=1$	$x_2=1$	$y_1=1$	$y_2=1$	$y_3=0$

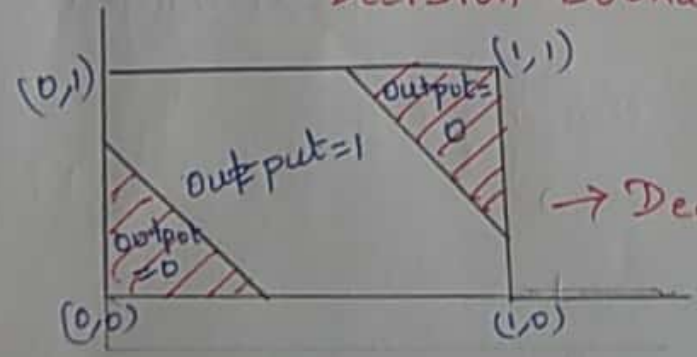
EXOR



Decision Boundary constructed by Neuron 1



Decision boundary constructed by Neuron 2



→ Decision boundary constructed by Complete Neuron.