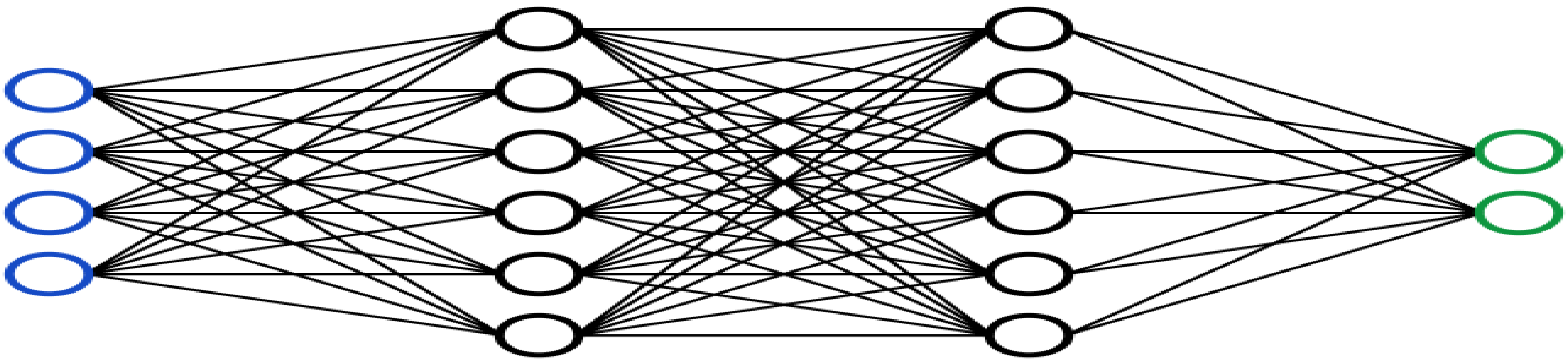


Artificial Neural Network (ANN)

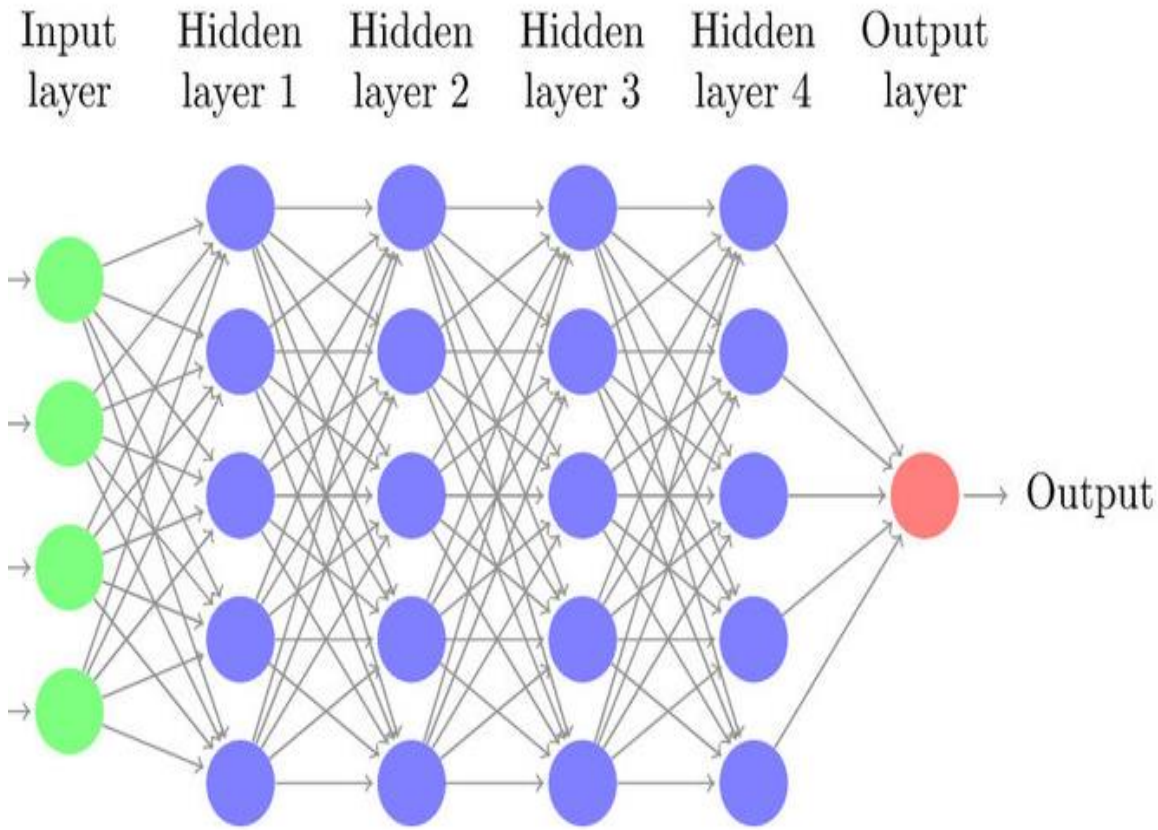
Lecture6

Dina El-Manakhly, Ph. D.

dina_almnakhly@science.suez.edu.eg

Multi-layer perceptron (REVISION)

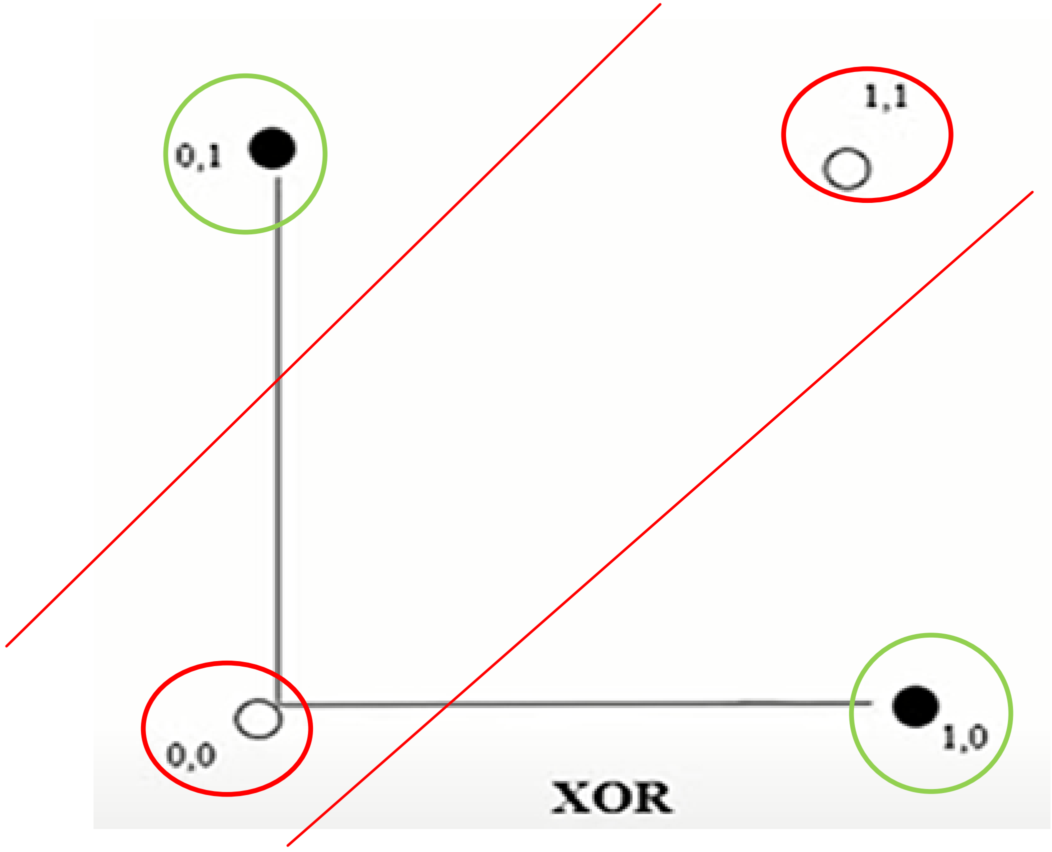
- Multi-layer perception is also known as MLP. It is fully connected dense layers, which transform any input dimension to the desired dimension.
- A multi-layer perceptron has one input layer and for each input, there is one neuron(or node), it has one output layer with a single node for each output and it can have any number of hidden layers and each hidden layer can have any number of nodes.



XOR-Gate with multilayer perceptron (REVISION)

EX-OR (X-OR) Gate Truth Table

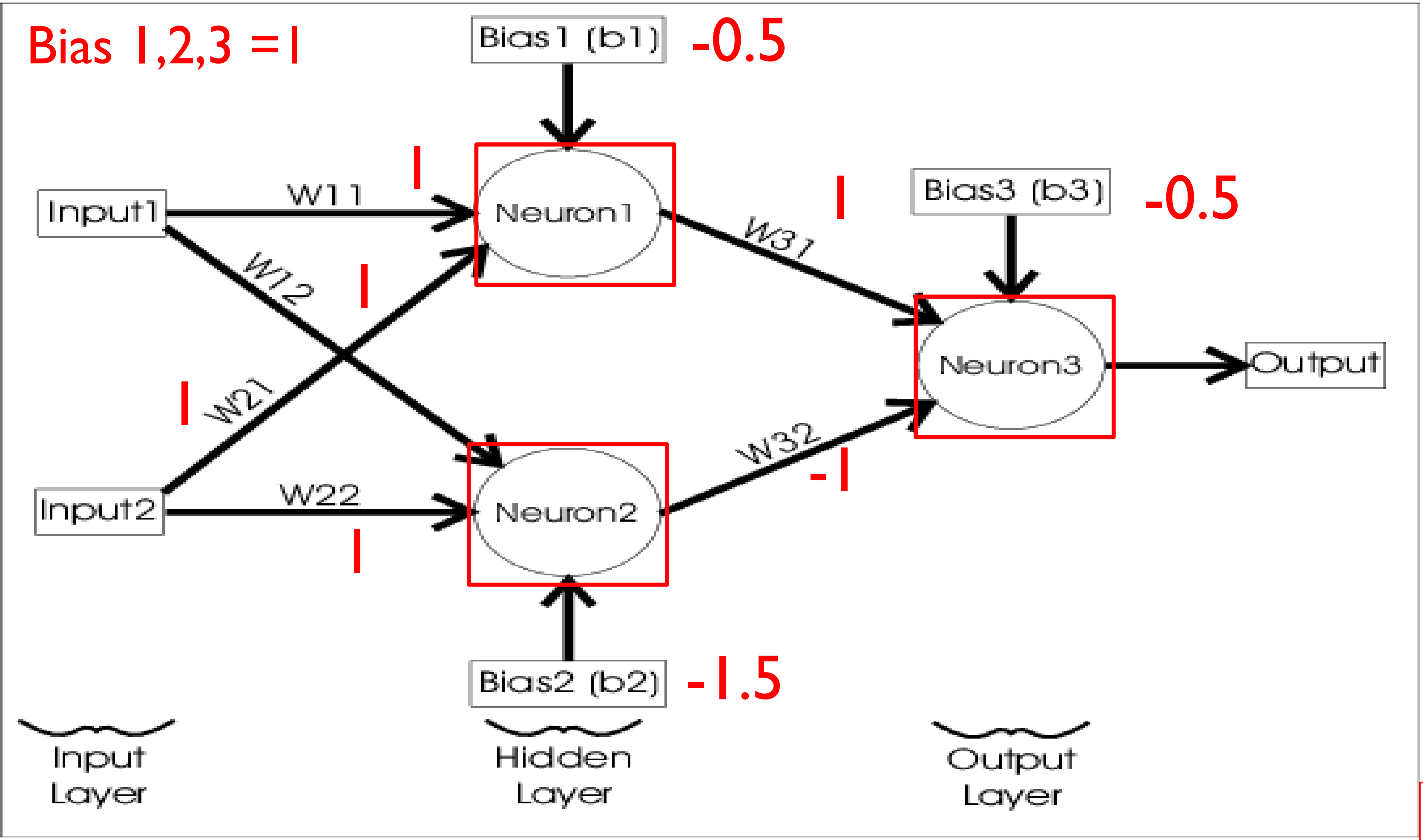
Inputs		Output $X = A \oplus B$
A	B	
0	0	0
0	1	1
1	0	1
1	1	0



Non linearly separable

Calculation of XOR gate output

(Example with true weights)



1- The XOR gate truth table says, if X1 = 0 and X2 =0 ,the output should be 0

2- For hidden layer neuron Neuron 1=
*Input1 * w11 + input2 * w21 + bias1* b1=*
*0*1 + 0*1 + 1*(-0.5) = -0.5*
Stepfunction(-0.5)=0

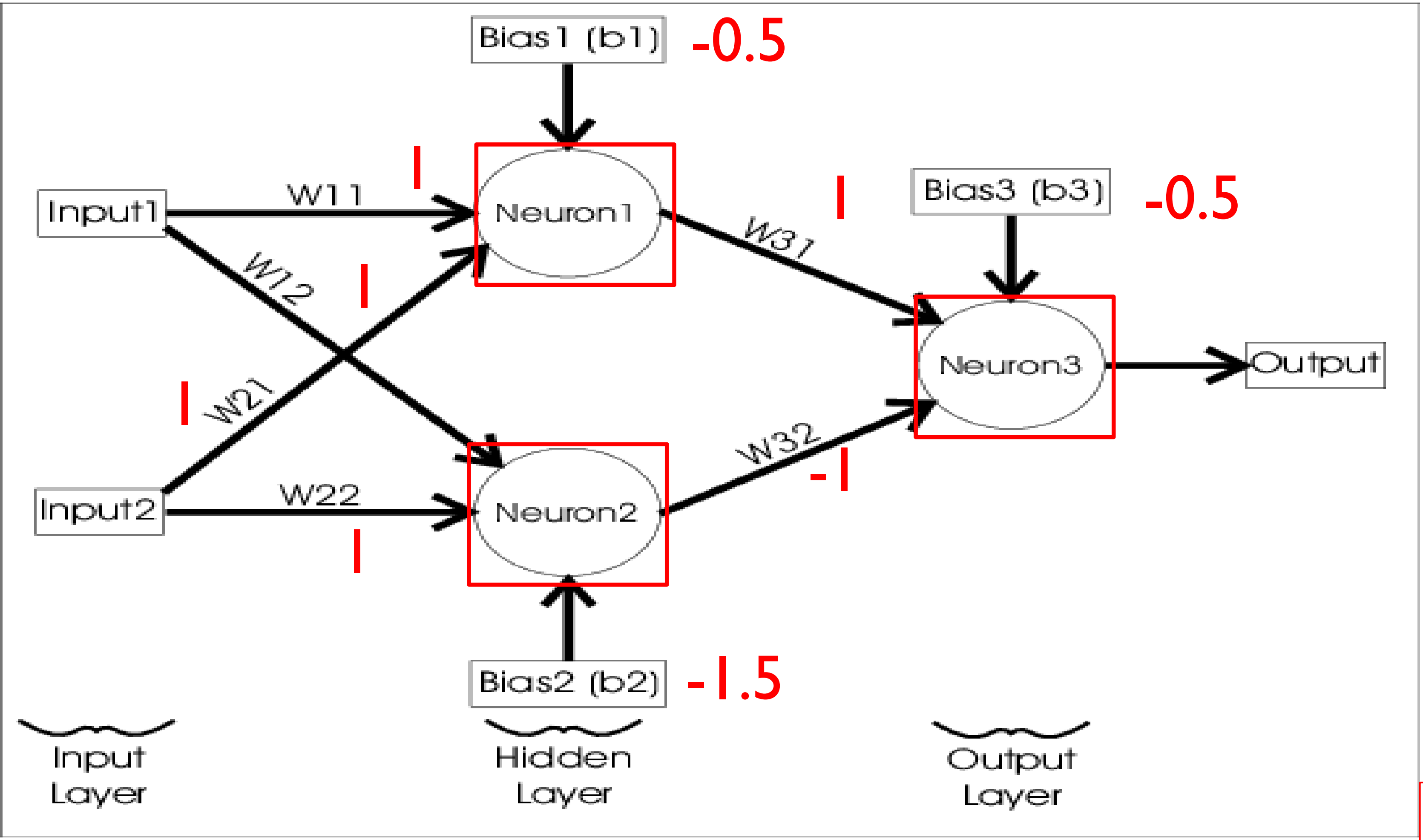
3- For hidden layer neuron Neuron 2=
*Input1 * w12 + input2 * w22 + bias2* b2=*
*0*1 + 0*1 + 1*(-1.5) = -1.5*
Stepfuntion(-1.5)=0

4- For Neuron 3=
*N1 * w31 + N2 * w32 + bias3* b3=*
*0*1 + 0*(-1) + 1*(-0.5) = -0.5*
Stepfunction(-0.5)=0

5- Matched with XOR truth table first row.

Calculation of XOR gate output

(Example with true weights)



1- The XOR gate truth table says, if X1 = 0 and X2 =1 ,the output should be 0

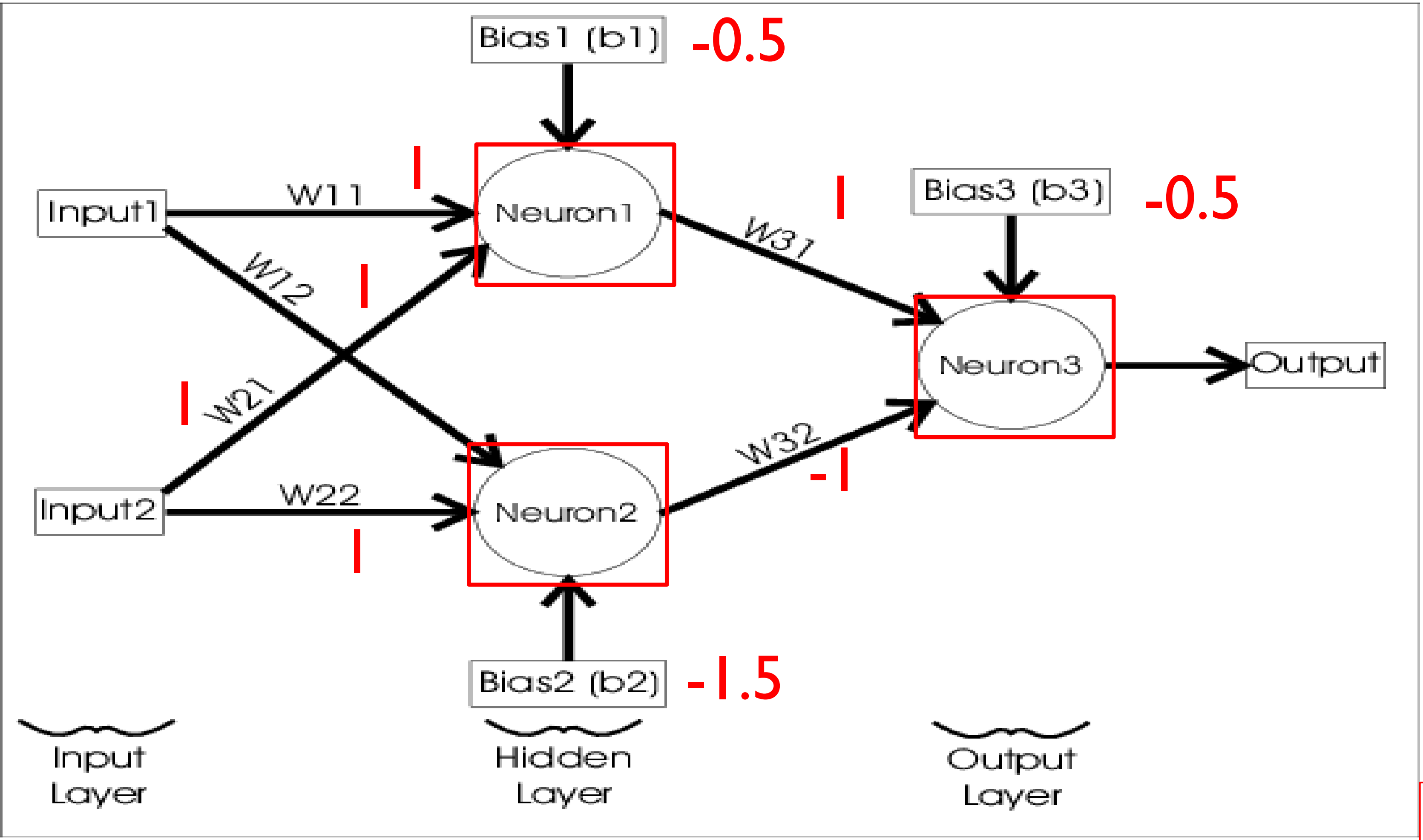
2- For hidden layer neuron Neuron 1=
*Input1 * w11 + input2 * w21 + bias1* b1=*
*0*1 + 1*1 + 1*(-0.5) = 0.5*
Stepfunction(0.5)=1

3- For hidden layer neuron Neuron 2=
*Input1 * w12 + input2 * w22 + bias2* b2=*
*0*1 + 1*1 + 1*(-1.5) = -0.5*
Stepfunction(-0.5)=0

4- For Neuron 3=
*N1 * w31 + N2 * w32 + bias3* b3=*
*1*1 + 0*(-1) + 1*(-0.5) = 0.5*
Stepfunction(0.5)=1

5- Matched with XOR truth table second row.

Calculation of XOR gate output
(Example with true weights)



1- The XOR gate truth table says, if X1 = 1 and X2 =0 ,the output should be 0

2- For hidden layer neuron Neuron 1=
*Input1 * w11 + input2 * w21 + bias1* b1=*
*1*1 + 0*1 + 1*(-0.5) = 0.5*
Stepfunction(0.5)=1

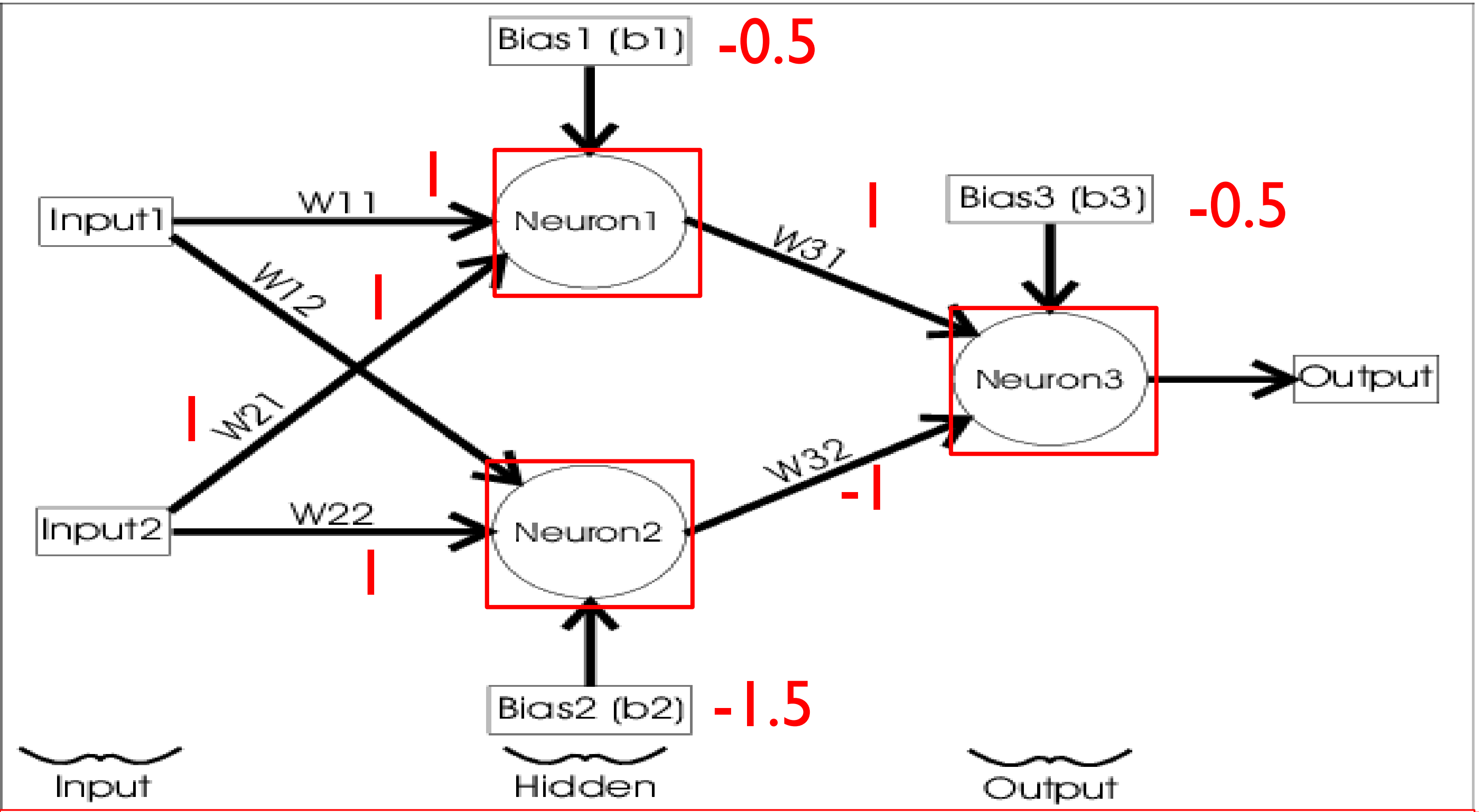
3- For hidden layer neuron Neuron 2=
*Input1 * w12 + input2 * w22 + bias2* b2=*
*1*1 + 0*1 + 1*(-1.5) = -0.5*
Stepfunction(-0.5)=0

4- For Neuron 3=
*N1 * w31 + N2 * w32 + bias3* b3=*
*1*1 + 0*(-1) + 1*(-0.5) = 0.5*
Stepfunction(0.5)=1

5- Matched with XOR truth table third row.

Calculation of XOR gate output

(Example with true weights)



1- The XOR gate truth table says, if X1 = 1 and X2 =1 ,the output should be 0

2- For hidden layer neuron Neuron 1=
Input1 * w11 + input2 * w21 + bias1* b1=
1*1 + 1*1 + 1*(-0.5) = 1.5
Stepfunction(1.5)=1

3- For hidden layer neuron Neuron 2=
Input1 * w12 + input2 * w22 + bias2* b2=
1*1 + 1*1 + 1*(-1.5) = 0.5
Stepfunction(0.5)=1

4- For Neuron 3=
N1 * w31 + N2 * w32 + bias3* b3=
1*1 + 1*(-1) + 1*(-0.5) = -0.5
Stepfunction(-0.5)=0

Finally, the neural network matches the XOR truth table so we don't need to update the given weights.

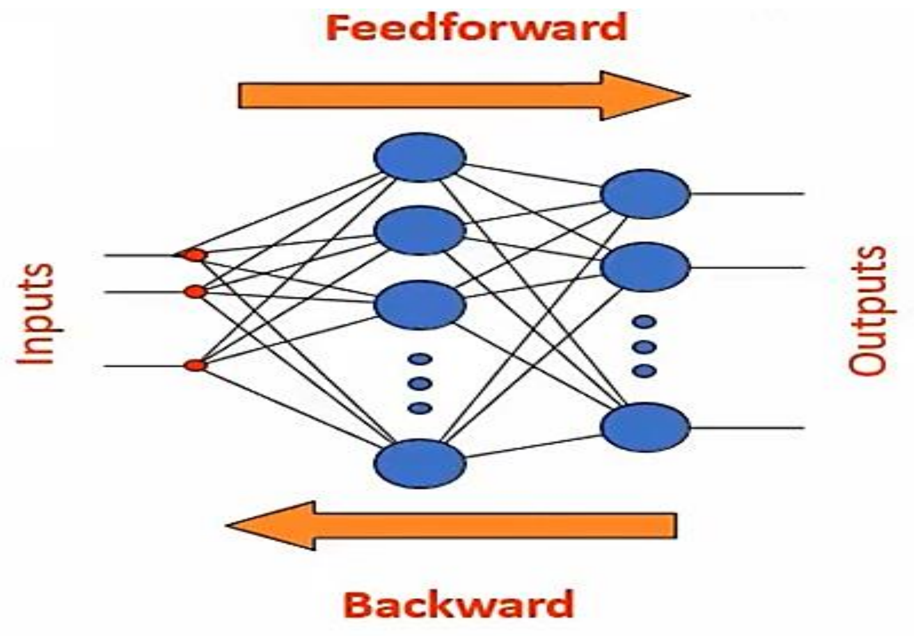
5- Matched with XOR truth table fourth row.

Weight adaptation in MLP

Method: Back propagation

- **Fowrward VS Backword passes**

The Backpropagation algorithm is a sensible approach for dividing the contribution of each weight.



Sum of product

Step1	**Fowrward**	**Input weights**	➡	**SOP**	➡	**Prediction Output**	➡	**Prediction Error**
Step2	**backward**	**Prediction Error**	➡	**Prediction Output**	➡	**SOP**	➡	**Input weights**

Simple example

- **Let us work with a simpler example**

$$y = x^2 z + c$$

How to answer this question: **What is the effect on the output Y given a change in variable X?**

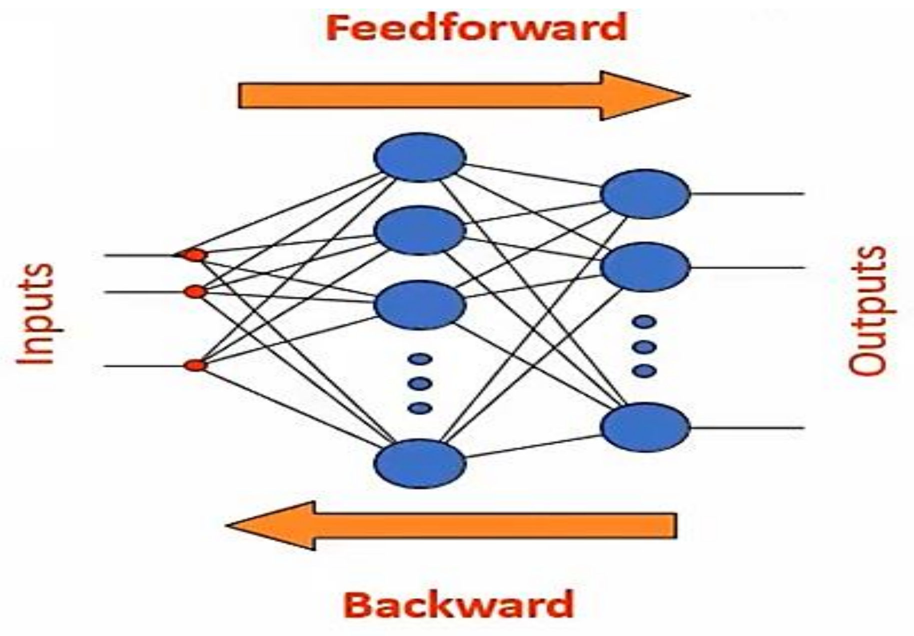
This question is answered using derivatives. Derivative of Y wrt X ($\partial y / \partial x$) will tell us the effect of changing the variable X over the output Y.

Weight adaptation

Second Method: Back propagation

- **Fowrward VS Backword passes**

The Backpropagation algorithm is a sensible approach for dividing the contribution of each weight.



Sum of product

Fowrward	**Input weights**	➡	**SOP**	➡	**Prediction Output**	➡	**Prediction Error**
$\frac{\partial E}{\partial W}$ **backward**	**Prediction Error**	➡	**Prediction Output**	➡	**SOP**	➡	**Input weights**

Chain rule

- Backward Pass: what is the change in prediction Error (E) given the change in weight (W)?
  Get partial derivative of E **w.r.t** W

Prediction Error → **Prediction Output** → **SOP** → **Input weights**

$$E = \frac{1}{2}(d-y)^2 \qquad y = f(s) = \frac{1}{1+e^{-s}} \qquad s = x_1 w_1 + x_2 w_2 + b \qquad w_1, w_2$$

$$\boxed{\frac{\partial E}{\partial W}} = \frac{\partial E}{\partial y} \times \frac{\partial y}{\partial s} \times \frac{\partial s}{\partial w_1}, \frac{\partial s}{\partial w_2}$$

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial y} x \frac{\partial y}{\partial s} x \frac{\partial s}{\partial w_1}$$

$$\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial y} x \frac{\partial y}{\partial s} x \frac{\partial s}{\partial w_2}$$

Weight adaptation

- **Update the Weights**

  In order to update the weights , use the Gradient Descent

$$W_{inew} = W_{iold} + \eta * \frac{\partial E}{\partial W_i}$$

Steps of single epoch

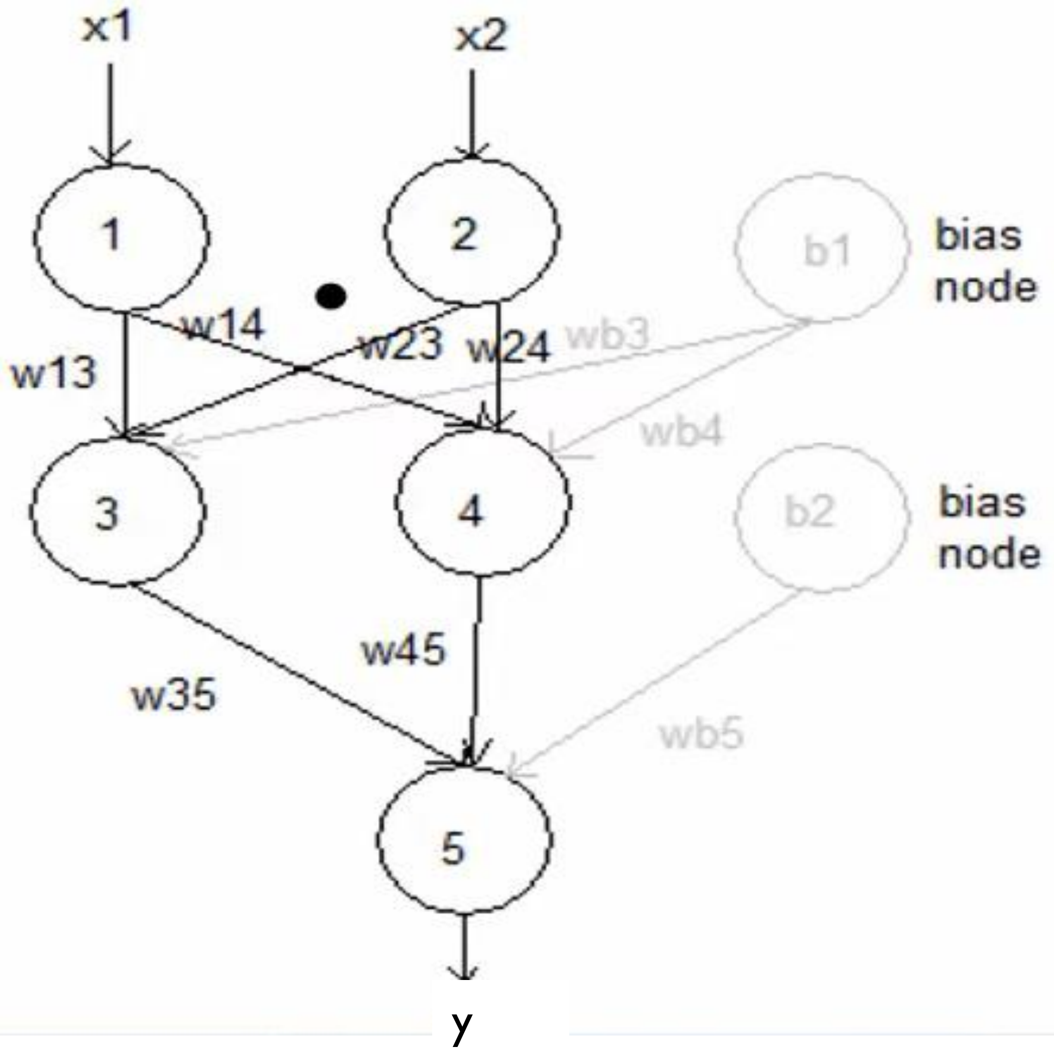
For each pattern

- Forward prop
  - Calculate net_j and o_j for all neurons (except input layer and bias neurons)
  - Calculate specific error (for single pattern)
- Back prop
  - Calculate δ_j for all neurons (except input layer and bias neurons)
  - Calculate $\Delta w_{i,j}$ for all variable weights including bias weights
  - $w_{i,j} \coloneqq w_{i,j} + \Delta w_{i,j}$

Example: XOR Problem

x1	x2	t
0	0	0
0	1	1
1	0	1
1	1	0

Assume learning rate = 0.3

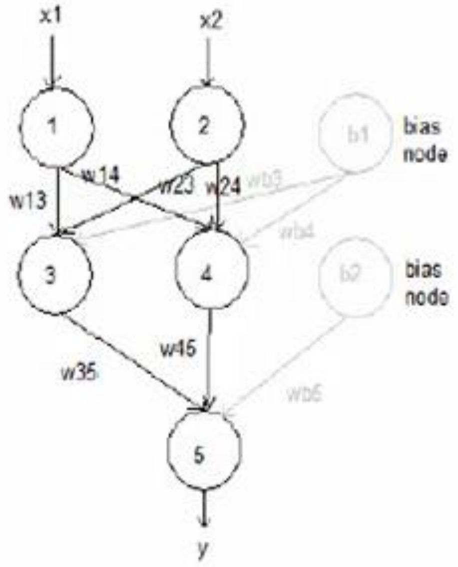


Epoch: 1

Pattern: 1: $x1 = 0, x2 = 0, t = 0$

- Initial weights:

w13 = 0.3	w23 = -0.1	wb3 = 0.2
w14 = -0.2	w24 = 0.2	wb4 = -0.3
w35 = 0.4	w45 = -0.2	wb5 = 0.4



- Forward prop:
  - net3 = w13 * x1 + w23 * x2 + wb3 = 0.3 * 0 – 0.1 * 0 + 0.2 = 0.2
  - o3 = 1/(1 + e^-net3) = 1/(1 + e^-0.2) = 0.5498
  - net4 = w14 * x1 + w24 * x2 + wb4 = -0.2 * 0 + 0.2 * 0 – 0.3 = -0.3
  - o4 = 1/(1 + e^-net4) = 1/(1 + e^0.3) = 0.4256
  - net5 = w35 * o3 + w45 * o4 + wb5 = 0.4 * 0.5498 – 0.2 * 0.4256 + 0.4 = 0.5348
  - y = 1/(1+e^-net5) = 1/(1+e^-0.5348) = 0.6306
- Calculating error:
  - Err_p1 = 0.5 * (0 – 0.6306)^2 = 0.1988

Sigmoid

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

Mean square Error

Actual - predicted

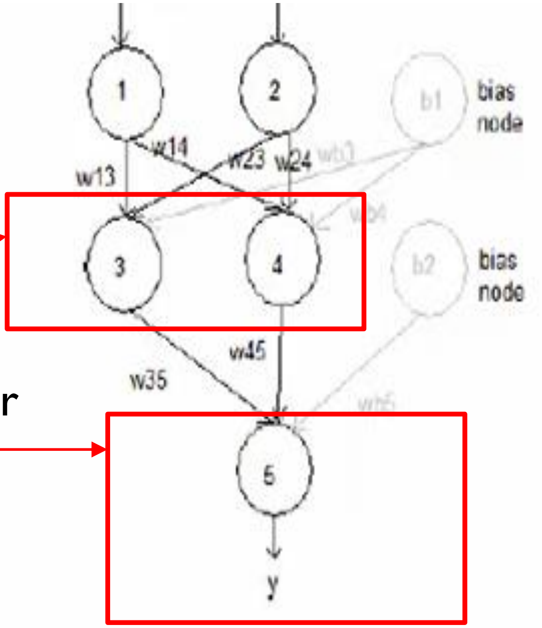
Epoch: 1

Pattern: 1: $x1 = 0, x2 = 0, t = 0$

Back Prop:

1) Finding delta

Delta for only hidden layer and output layer



$\delta 5 = y*(1 - y)*(t - y) = 0.6306 * (1 - 0.6306) * (0 - 0.6306) = - 0.1469$

$\delta 3 = o3(1 - o3)* \delta 5 * w35 = 0.5498 * (1 - 0.5498) * - 0.1469 * 0.4 = - 0.0145$

$\delta 4 = o4(1 - o4)* \delta 5 * w45 = 0.4256 * (1 - 0.4256) * - 0.1469 * -0.2 = 0.0072$

1- If h is output neuron

Delta= $y_{h*}(1-y_h)*(t_h-y_h)$

2- If h is hidden neuron

Delta=$o_h*(1-o_h)* \sum_{l \in L} \text{delta}_l * whl$, L is the next layer

2) Finding new weights

w35 := w35 + η * o3 * δ5 = 0.4 + 0.3 * 0.5498 * - 0.1469 = 0.3758

w45 := w45 + η * o4 * δ5 = -0.2 + 0.3 * 0.4256 * - 0.1469 = -0.2188

wb5 := wb5 + η * 1 * δ5 = 0.4 + 0.3 * 1 * - 0.1469 = 0.3559

w14 := w14 + η * x1 * δ4 = - 0.2 + 0.3 * 0 * 0.0072 = - 0.2

w24 := w24 + η * x2 * δ4 = 0.2 + 0.3 * 0 * 0.0072 = 0.2

wb4 := wb4 + η * 1 * δ4 = - 0.3 + 0.3 * 1 * 0.0072 = - 0.2978

w13 := w13 + η * x1 * δ3 = 0.3 + 0.3 * 0 * - 0.0145 = 0.3

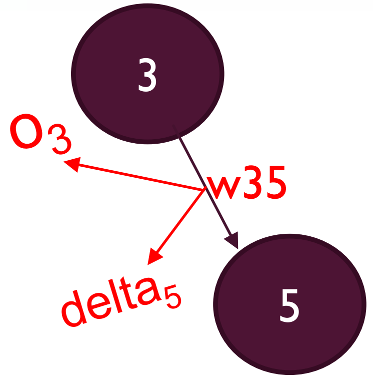
w23 := w23 + η * x2 * δ3 = - 0.1 + 0.3 * 0 * - 0.0145 = - 0.1

wb3 := wb3 + η * 1 * δ3 = 0.2 + 0.3 * 1 * - 0.0145 = 0.1957

$$W_{inew} = W_{iold} + \eta * \frac{\partial E}{\partial W_i}$$

$\boxed{= \text{delta}*\text{o}}$

Example



Epoch: 1

Pattern: 2: $x1 = 0, x2 = 1, t = 1$

- weights:

w13 = 0.3	w23 = - 0.1	wb3 = 0.1957
w14 = - 0.2	w24 = 0.2	wb4 = - 0.2978
w35 = 0.3758	w45 = - 0.2188	wb5 = 0.3559

- Forward prop:
  - net3 = w13 * x1 + w23 * x2 + wb3 = ...
  - o3 = 1/(1 + e^-net3) = ...
  - net4 = w14 * x1 + w24 * x2 + wb4 = ...
  - o4 = 1/(1 + e^-net4) = ...
  - net5 = w35 * o3 + w45 * o4 + wb5 = ...
  - y = 1/(1+e^-net5) = ...
- Calculating error:
  - Err_p2 = 0.5 * (t – y)^2 = ...

Assignment

Epoch: 1

Pattern: 2: $x1 = 0, x2 = 1, t = 1$

Epoch: 1

Pattern: 3: $x1 = 1, x2 = 0, t = 1$

Epoch: 1

Pattern: 4: $x1 = 1, x2 = 1, t = 0$

End of Epoch 1

Total error = Err_p1 + Err_p2 + Err_p3 + Err_p4 = 0.1988 + ...

If Total error <= tolerance (If given): Then stop training

If epoch number = max number of epochs (if given): Then stop training

Otherwise, run another epoch using last weights