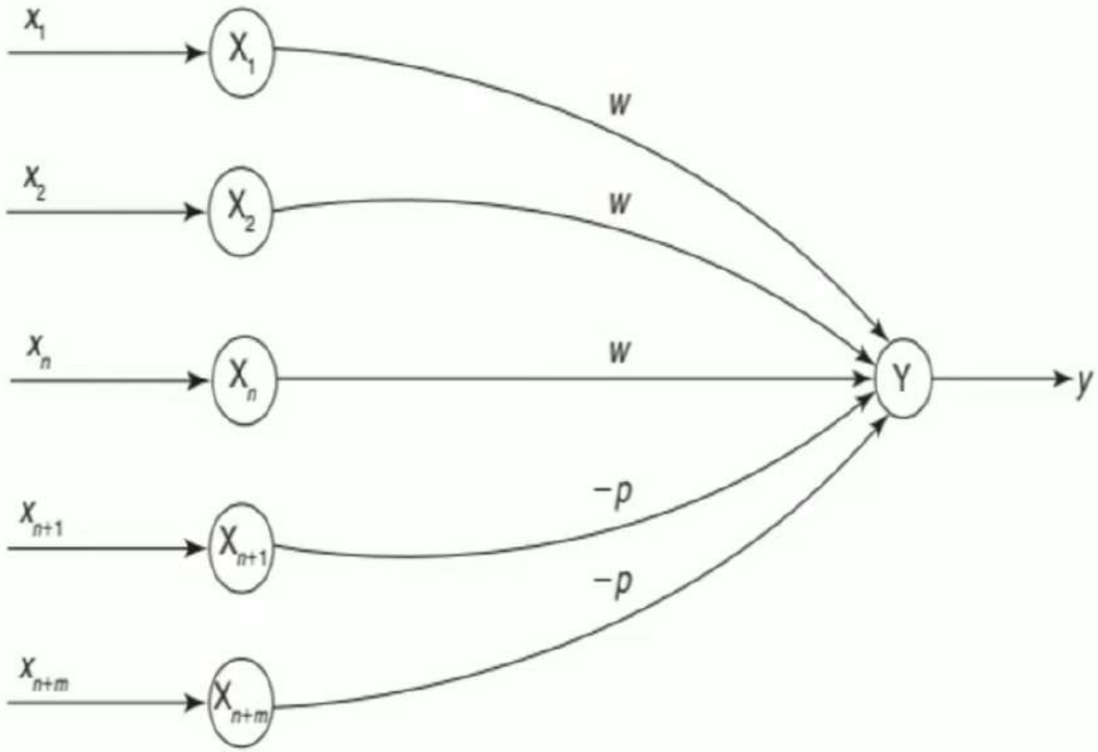
Implement AND function using McCulloch–Pitts Neuron

- The McCulloch–Pitts neuron was the earliest neural network discovered in 1943.
- It is usually called as M–P neuron.
- Since the firing of the output neuron is based upon the threshold, the activation function here is defined as

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

- The threshold value should satisfy the following condition: $\theta > nw - p$



Implement AND function using McCulloch–Pitts Neuron

- Consider the truth table for AND function
- The M–P neuron has no particular training algorithm
- In M-Pneuron, only analysis is being performed.
- Hence, assume the weights be w1 = 1 and w2 = 1.

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

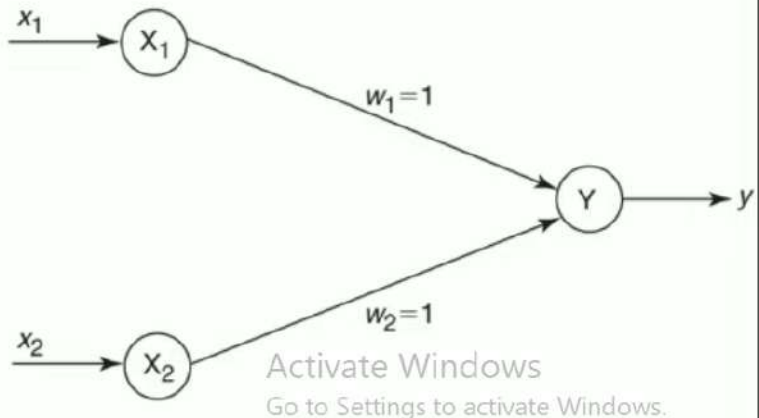
$$(1, 1),\ y_{in} = x_1 w_1 + x_2 w_2 = 1 \times 1 + 1 \times 1 = 2$$

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

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

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

Threshold value is set equal to 2 (θ = 2).



Implement AND function using McCulloch–Pitts Neuron

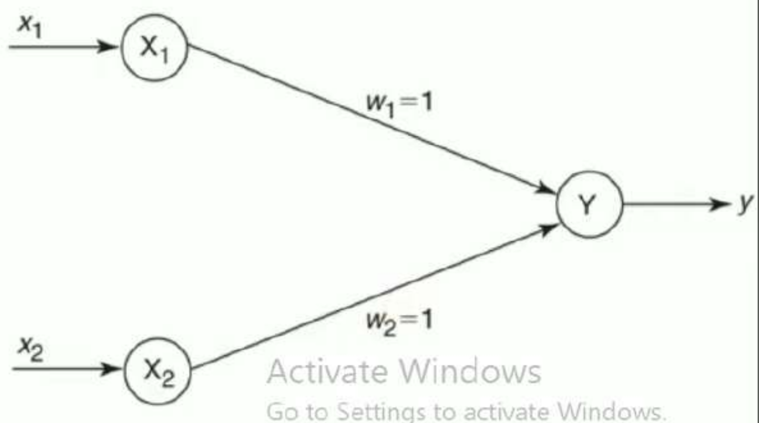
- This can also be obtained by

$$\theta \geq nw - p$$

- Here, n = 2, w = 1 (excitatory weights) and p = 0 (no inhibitory weights).
- Substituting these values in the above-mentioned equation we get $\theta \geq 2 \times 1 - 0 \Rightarrow \theta \geq 2$
- Thus, the output of neuron Y can be written

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

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



Implement ANDNOT function using McCulloch–Pitts Neuron

- Consider the truth table for ANDNOT function
- The M–P neuron has no particular training algorithm
- In M-P neuron, only analysis is being performed.
- Hence, assume the weights be w1 = 1 and w2 = 1.

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

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

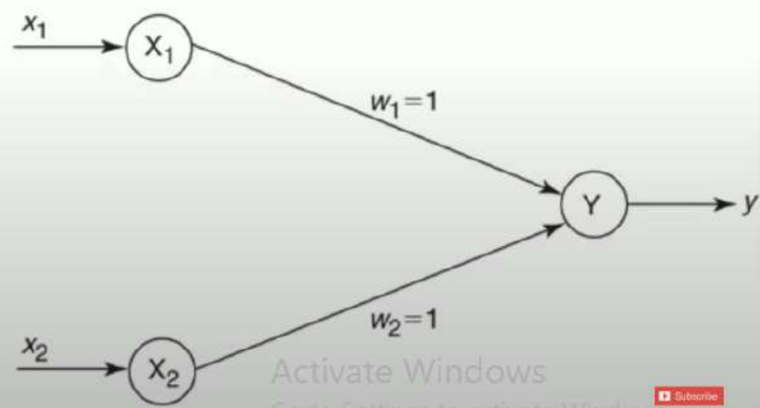
$$(1, 1),\ y_{in} = 1 \times 1 + 1 \times 1 = 2$$

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

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

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

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



Implement ANDNOT function using McCulloch–Pitts Neuron

- Consider the truth table for ANDNOT function
- The M–P neuron has no particular training algorithm
- In M-P neuron, only analysis is being performed.
- Hence, assume the weights be w1 = 1 and w2 = -1.

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

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

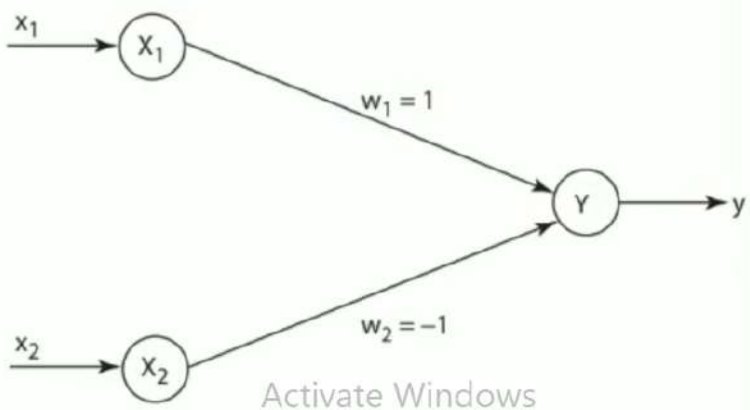
$$(1, 1),\ y_{in} = 1 \times 1 + 1 \times -1 = 0$$

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

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

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

From the calculated net inputs, now it is possible to fire the neuron for input (1, 0) only by fixing a threshold of 1, i.e., θ ≥ 1 for Y unit.



Implement XOR function using McCulloch–Pitts Neuron

- Consider the truth table for XOR function
- The M–P neuron has no particular training algorithm
- In M-P neuron, only analysis is being performed.
- XOR function cannot be represented by simple and single logic function; it is represented as

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

$$y = x_1\overline{x_2} + \overline{x_1}x_2$$

Implement XOR function using McCulloch–Pitts Neuron

$$y = x_1 \overline{x_2} + \overline{x_1} x_2$$

$$y = z_1 + z_2$$

where

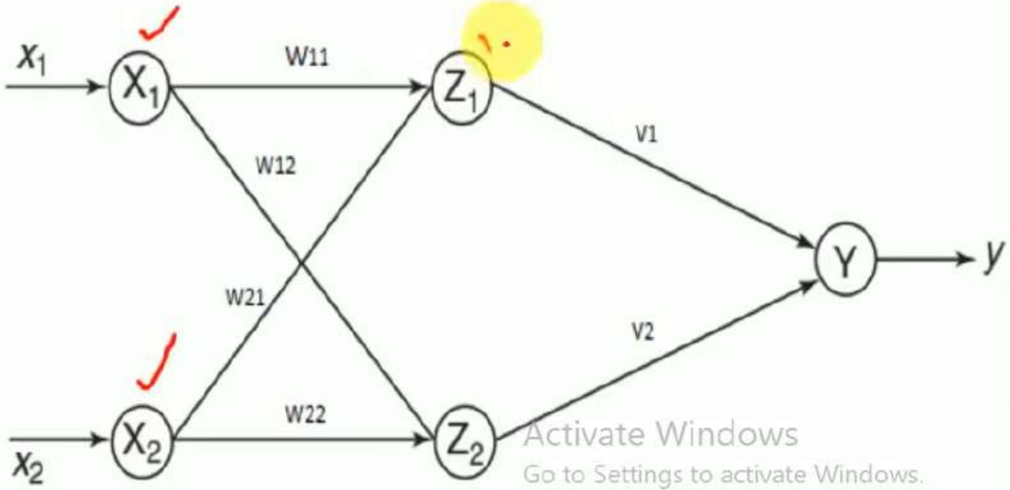
$$z_1 = x_1 \overline{x_2} \qquad \text{(function 1)}$$

$$z_2 = \overline{x_1} x_2 \qquad \text{(function 2)}$$

$$y = z_1 \text{ (OR) } z_2 \qquad \text{(function 3)}$$

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

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



Implement XOR function using McCulloch–Pitts Neuron

- First function $z_1 = x_1 \overline{x_2}$
- The truth table for function z_1

x_1	x_2	z_1
0	0	0
0	1	0
1	0	1
1	1	0

- Assume the weights are initialized to

$$w_{11} = w_{21} = 1$$

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

- Calculate the net inputs,

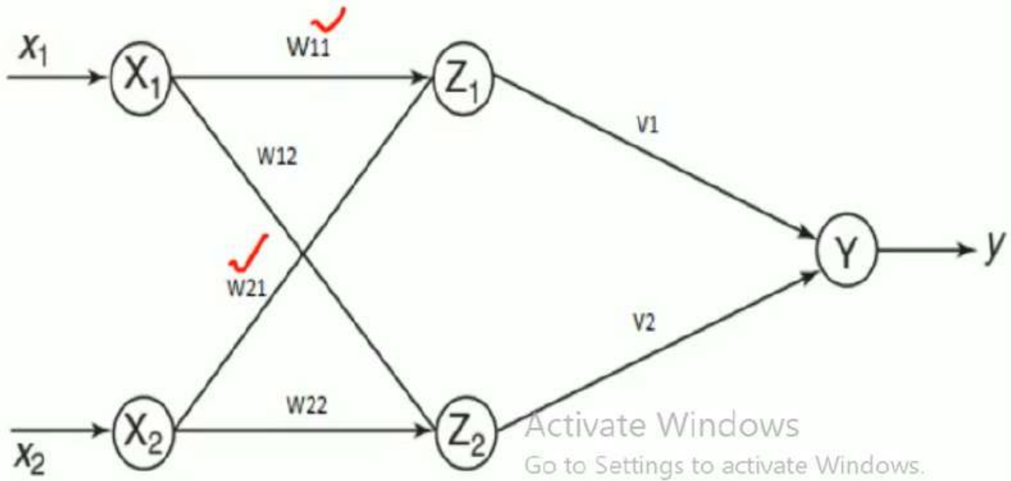
$$(0, 0), z_{1in} = 0 \times 1 + 0 \times 1 = 0$$

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

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

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

- Hence, it is not possible to obtain function z_1 using these weights.



Implement XOR function using McCulloch–Pitts Neuron

- First function $z_1 = x_1 \overline{x_2}$
- The truth table for function z_1
- Assume the weights are initialized to

$$w_{11} = 1; \quad w_{21} = -1$$

x_1	x_2	z_1
0	0	0
0	1	0
1	0	1
1	1	0

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

- Calculate the net inputs,

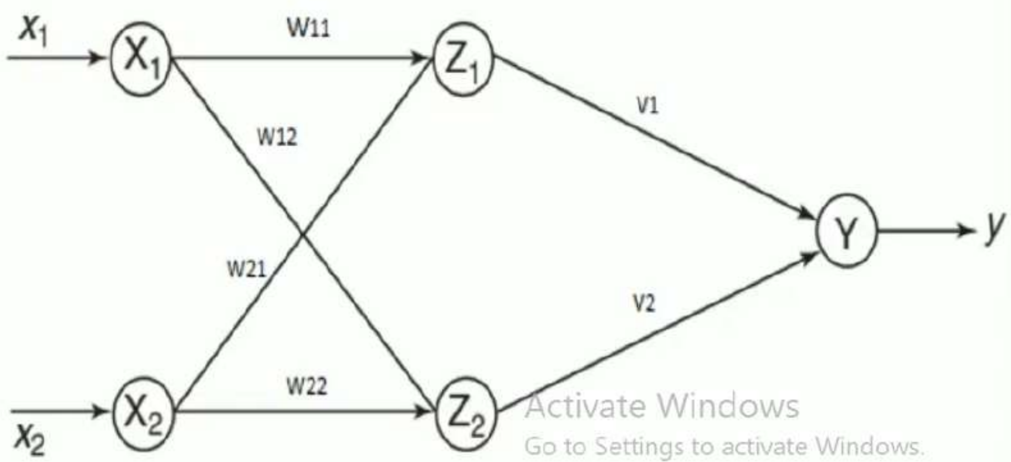
$$(0, 0), z_{1in} = 0 \times 1 + 0 \times -1 = 0$$

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

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

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

- If the θ=1 then the neuron fires.
- Hence $w_{11} = 1; \quad w_{21} = -1$



Implement XOR function using McCulloch–Pitts Neuron

- Second function $z_2 = \overline{x_1} x_2$
- The truth table for function z_2
- Assume the weights are initialized to

$$w_{12} = w_{22} = 1$$

- Calculate the net inputs,

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

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

$(1, 0), z_{2in} = 1 \times 1 + 0 \times 1 = 1$ ✓ x

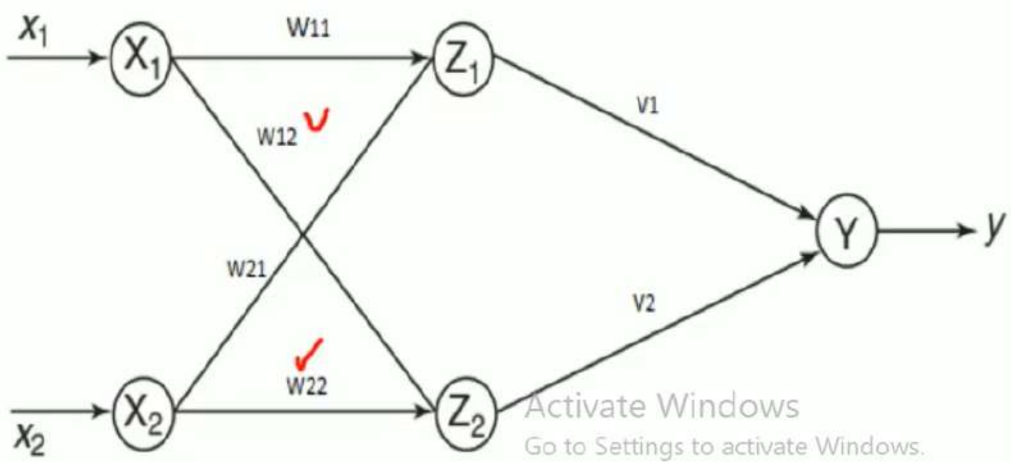
$(1, 1), z_{2in} = 1 \times 1 + 1 \times 1 = 2$ ✓ ✓

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

$\theta = 1, 2$

x_1	x_2	z_2
0	0	0
✓0	1	1
1	0	0
1	1	0

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



Implement XOR function using McCulloch–Pitts Neuron

- Second function $z_2 = \overline{x_1} x_2$
- The truth table for function z_2

$\theta = 1$

x_1	x_2	z_2
0	0	0
0	1	1 ✓
1	0	0
1	1	0

- Assume the weights are initialized to

$$w_{12} = -1; \quad w_{22} = 1$$

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

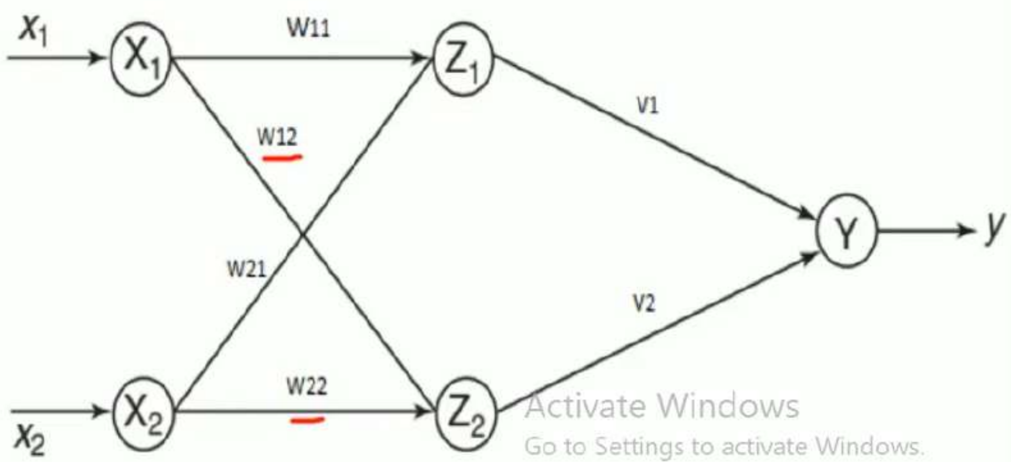
- Calculate the net inputs,

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

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

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

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



- If the θ=1 then the neuron fires.
- Hence $w_{12} = -1; \quad w_{22} = 1$

Implement XOR function using McCulloch–Pitts Neuron

- Third function $y = z_1 \text{ (OR) } z_2$
- The truth table for function y

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	0

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

$\theta = 0, 1$

- Assume the weights are initialized to

$$v_1 = v_2 = 1$$

- Calculate the net inputs,

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

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

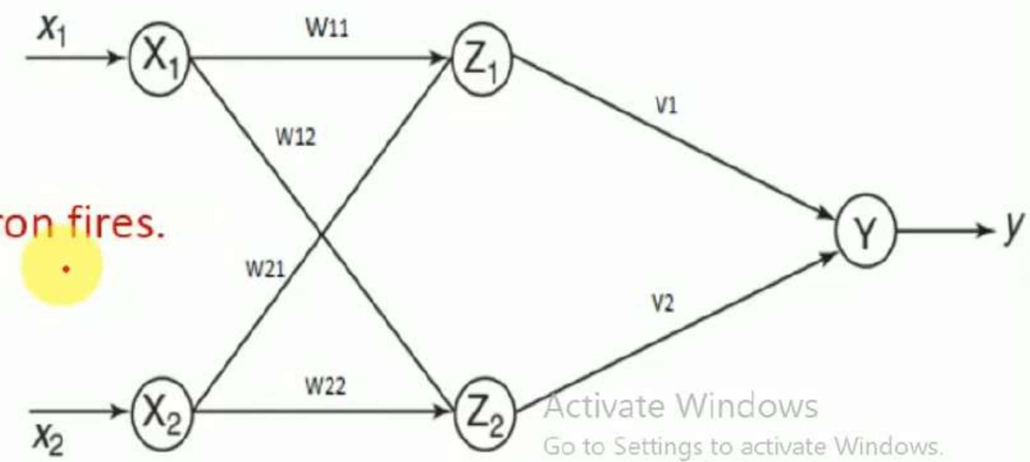
$(0, 1), y_{in} = 0 \times 1 + 1 \times 1 = 1$

If the θ=1 then the neuron fires.

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

Hence $v_1 = v_2 = 1$

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

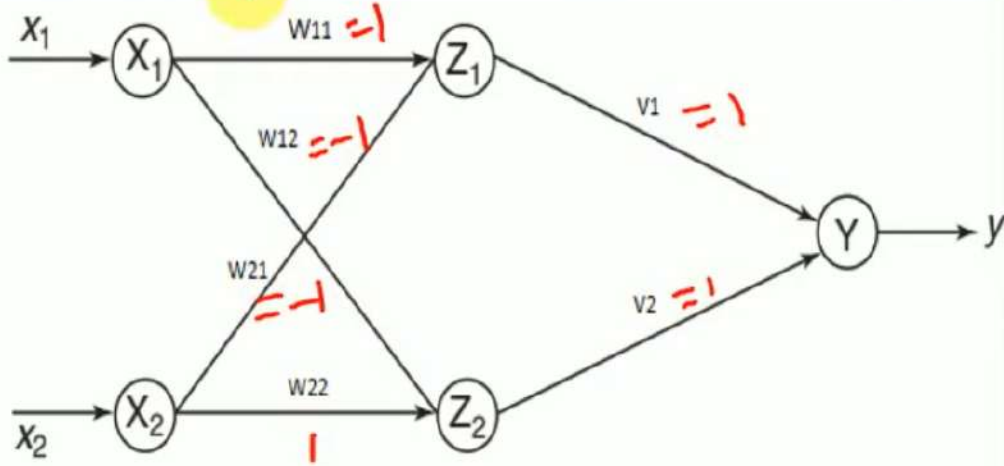


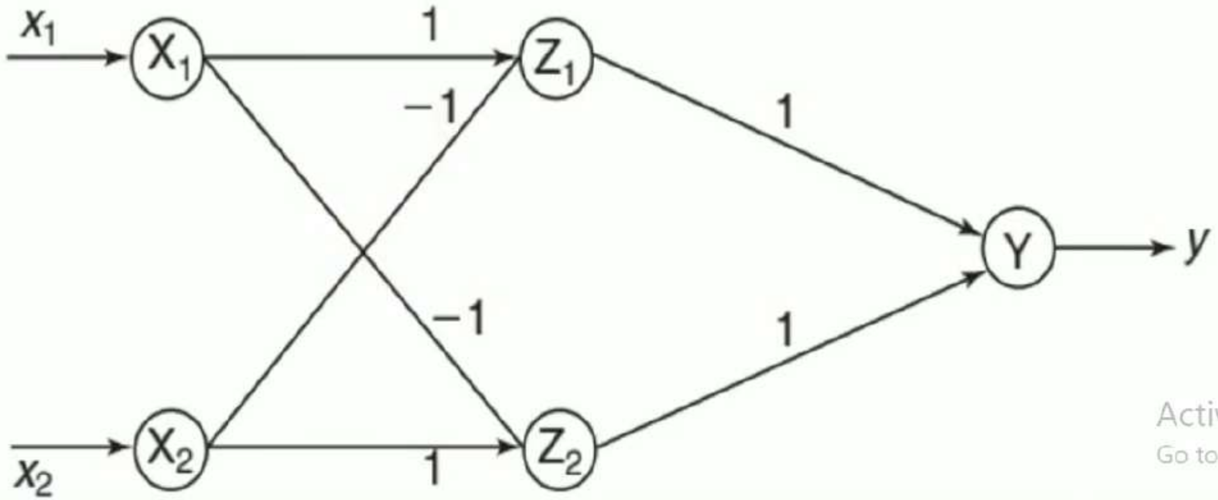
Implement XOR function using McCulloch–Pitts Neuron

$$w_{11} = 1; \quad w_{21} = -1$$

$$w_{12} = -1; \quad w_{22} = 1$$

$$v_1 = v_2 = 1$$





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

Design a Hebb Net to implement logical 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 weights and bias are set to zero, i.e.,

$$w_1 = w_2 = b = 0$$

- First input [x1 x2 b] = [111] and target = 1 [i.e., y = 1]:
- Setting the initial weights as old weights and applying the Hebb rule, we get

$$w_i(\text{new}) = w_i(\text{old}) + \Delta w_i$$

$$\Delta w_i = x_i y$$

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

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

$$\Delta b = y = 1$$

$$w_1(\text{new}) = w_1(\text{old}) + \Delta w_1 = 0 + 1 = 1$$

$$w_2(\text{new}) = w_2(\text{old}) + \Delta w_2 = 0 + 1 = 1$$

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

Design a Hebb Net to implement logical AND function

- Second input [x1 x2 b] = [1 –1 1] and y = –1:
- The weight change here is

$$\Delta w_1 = x_1 y = 1 \times -1 = -1$$
$$\Delta w_2 = x_2 y = -1 \times -1 = 1$$
$$\Delta b = y = -1$$

- The new weights here are

$$w_1(\text{new}) = w_1(\text{old}) + \Delta w_1 = 1 - 1 = 0$$
$$w_2(\text{new}) = w_2(\text{old}) + \Delta w_2 = 1 + 1 = 2$$
$$b(\text{new}) = b(\text{old}) + \Delta b = 1 - 1 = 0$$

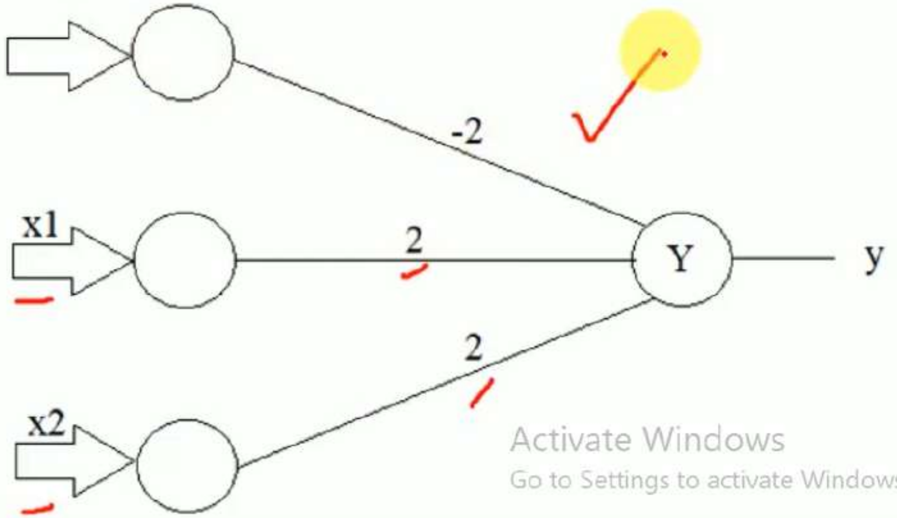
Inputs			Target
x_1	x_2	b	y
1	1	1	1
1	−1	1	−1
−1	1	1	−1
−1	−1	1	−1

Design a Hebb Net to implement logical AND function

- Similarly, by presenting the third and fourth input patterns, the new weights can be calculated.

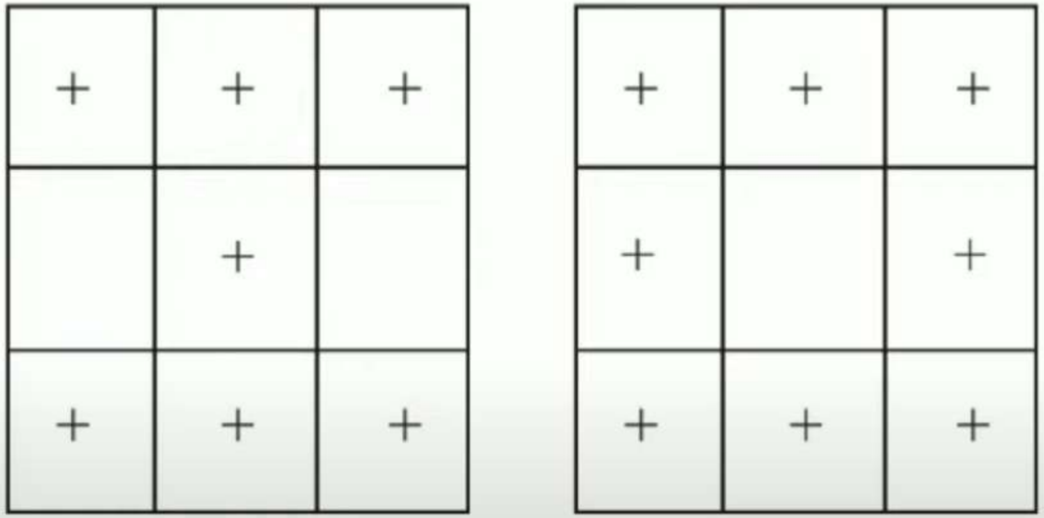
Inputs			Target
x_1	x_2	b	y
1	1	1	1
1	−1	1	−1
−1	1	1	−1
−1	−1	1	−1

Inputs				Weight changes			Weights		
x_1	x_2	b	y	Δw_1	Δw_2	Δb	w_1 (0	w_1 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



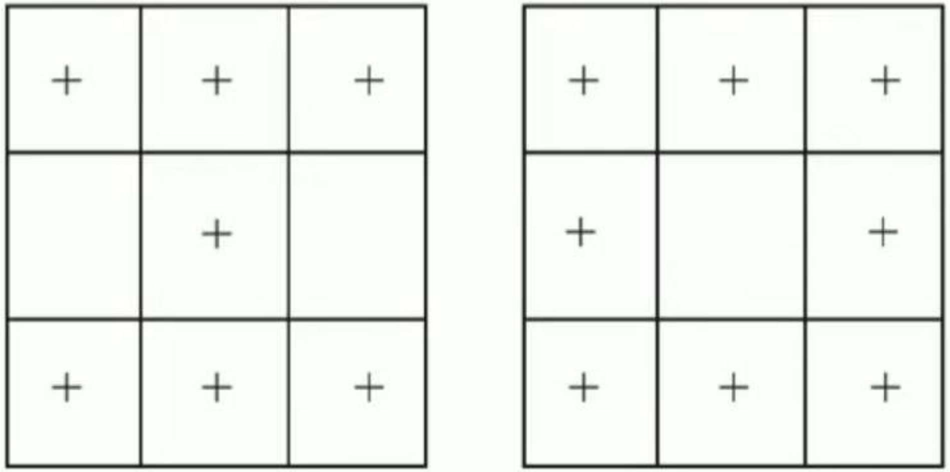
Hebb Net Solved Numerical Example - 2

- Using the Hebb rule, find the weights required to perform the following classifications of the given input patterns shown in Figure.
- The pattern is shown as 3 × 3 matrix form in the squares.
- The "+" symbols represent the value "1" and empty squares indicate "–1"



'I' 'O'

Hebb Net Solved Numerical Example - 2



'I' 'O'

Pattern	Inputs										Target
	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	b	y
I	1	1	1	−1	1	−1	1	1	1	1	1
O	1	1	1	1	−1	1	1	1	1	1	−1

Hebb Net Solved Numerical Example - 2

- Set the initial weights and bias to zero, i.e.,

$$w_1 = w_2 = w_3 = w_4 = w_5$$
$$= w_6 = w_7 = w_8 = w_9 = b = 0$$

- Presenting first input pattern (I), we calculate change in weights:

$$w_i(\text{new}) = w_i(\text{old}) + \Delta w_i \qquad [\Delta w_i = x_i y]$$

Pattern	Inputs										Target
	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	b	y
I	1	1	1	-1	1	-1	1	1	1	1	1
0	1	1	1	1	-1	1	1	1	1	1	-1

$$\Delta w_i = x_i y, \quad i = 1 \text{ to } 9$$
$$\Delta w_1 = x_1 y = 1 \times 1 = 1$$
$$\Delta w_2 = x_2 y = 1 \times 1 = 1$$
$$\Delta w_3 = x_3 y = 1 \times 1 = 1$$
$$\Delta w_4 = x_4 y = -1 \times 1 = -1$$
$$\Delta w_5 = x_5 y = 1 \times 1 = 1$$
$$\Delta w_6 = x_6 y = -1 \times 1 = -1$$
$$\Delta w_7 = x_7 y = 1 \times 1 = 1$$
$$\Delta w_8 = x_8 y = 1 \times 1 = 1$$
$$\Delta w_9 = x_9 y = 1 \times 1 = 1$$
$$\Delta b = y = 1$$

Hebb Net Solved Numerical Example - 2

- Setting the old weights as the initial weights here, we obtain

$$w_i(\text{new}) = w_i(\text{old}) + \Delta w_i$$

$$w_1(\text{new}) = w_1(\text{old}) + \Delta w_1 = 0 + 1 = 1$$

$$w_2(\text{new}) = w_2(\text{old}) + \Delta w_2 = 0 + 1 = 1$$

$$w_3(\text{new}) = w_3(\text{old}) + \Delta w_3 = 0 + 1 = 1$$

$$w_4(\text{new}) = -1, \quad w_5(\text{new}) = 1, \quad w_6(\text{new}) = -1,$$

$$w_7(\text{new}) = 1, \quad w_8(\text{new}) = 1, \quad w_9(\text{new}) = 1,$$

$$b(\text{new}) = 1$$

Pattern	Inputs										Target
	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	b	y
I	1	1	1	−1	1	−1	1	1	1	1	1
O	1	1	1	1	−1	1	1	1	1	1	−1

$$\Delta w_i = x_i y, \quad i = 1 \text{ to } 9$$

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

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

$$\Delta w_3 = x_3 y = 1 \times 1 = 1$$

$$\Delta w_4 = x_4 y = -1 \times 1 = -1$$

$$\Delta w_5 = x_5 y = 1 \times 1 = 1$$

$$\Delta w_6 = x_6 y = -1 \times 1 = -1$$

$$\Delta w_7 = x_7 y = 1 \times 1 = 1$$

$$\Delta w_8 = x_8 y = 1 \times 1 = 1$$

$$\Delta w_9 = x_9 y = 1 \times 1 = 1$$

$$\Delta b = y = 1$$

Hebb Net Solved Numerical Example - 2

- The weights after presenting first input pattern are
- w1= w2 = w3 = w5 = w7 = w8 = w9 = 1
- w4 = w6 = -1
- and b = 1
- Presenting first input pattern (O), we calculate change in weights:

Pattern	Inputs										Target
	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	b	y
I	1	1	1	−1	1	−1	1	1	1	1	1
→ 0	1	1	1	1	−1	1	1	1	1	1	−1

$$w_i(\text{new}) = w_i(\text{old}) + \Delta w_i \qquad [\Delta w_i = x_i y]$$

$$w_1(\text{new}) = w_1(\text{old}) + x_1 y = 1 + 1 \times -1 = 0$$

$$w_2(\text{new}) = w_2(\text{old}) + x_2 y = 1 + 1 \times -1 = 0$$

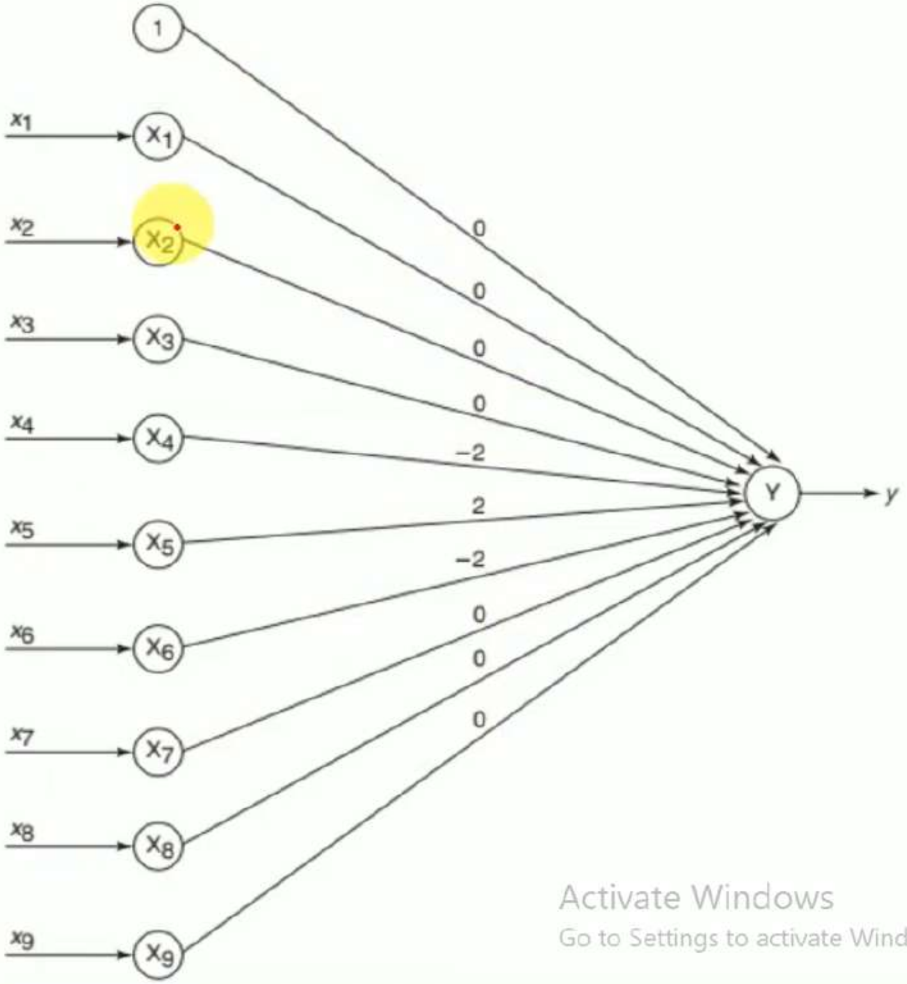
$$w_3(\text{new}) = w_3(\text{old}) + x_3 y = 1 + 1 \times -1 = 0$$

$$w_4(\text{new}) = w_4(\text{old}) + x_4 y = -1 + 1 \times -1 = -2$$

$$w_5(\text{new}) = w_5(\text{old}) + x_5 y = 1 + -1 \times -1 = 2$$

Hebb Net Solved Numerical Example - 2

- The weights after presenting first input pattern are
- w1= w2 = w3 = w7 = w8 = w9 = 0
- w5 = 2
- w4 = w6 = -2
- and b = 0



Perceptron Training Algorithm - Single Output Class

- Initialize the weights and the bias. Also initialize the learning rate ά (0< ά ≤ 1).
- Until the final stopping condition is false.
  - for each training pair indicated by s:t.
    - Set each input unit i = 1 to n: $x_i = s_i$
    - Calculate the output of the network.

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

    - Weight and bias adjustment:

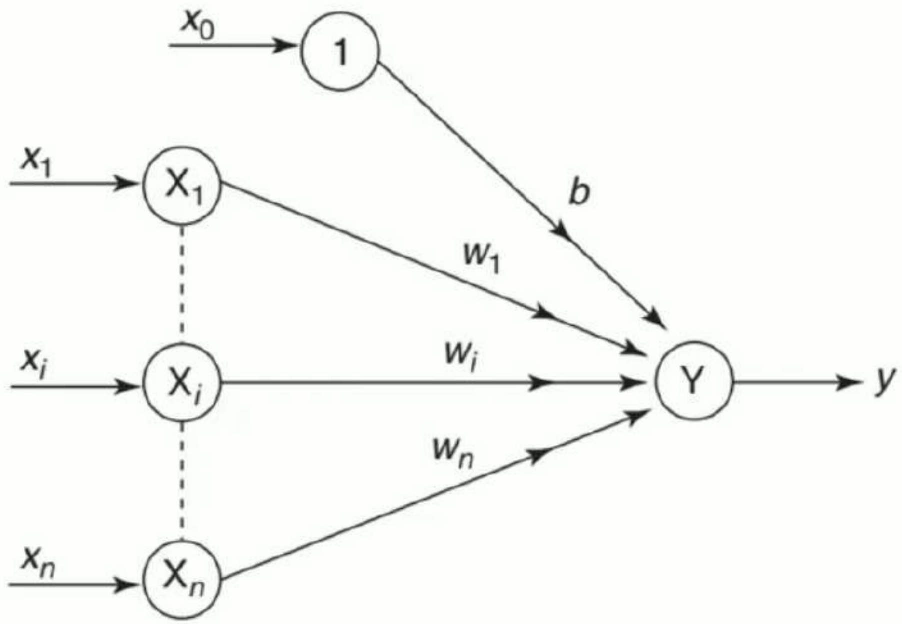
If $y \neq t$, then

$$w_i(\text{new}) = w_i(\text{old}) + \alpha t x_i$$
$$b(\text{new}) = b(\text{old}) + \alpha t$$

else we have

$$w_i(\text{new}) = w_i(\text{old})$$
$$b(\text{new}) = b(\text{old})$$

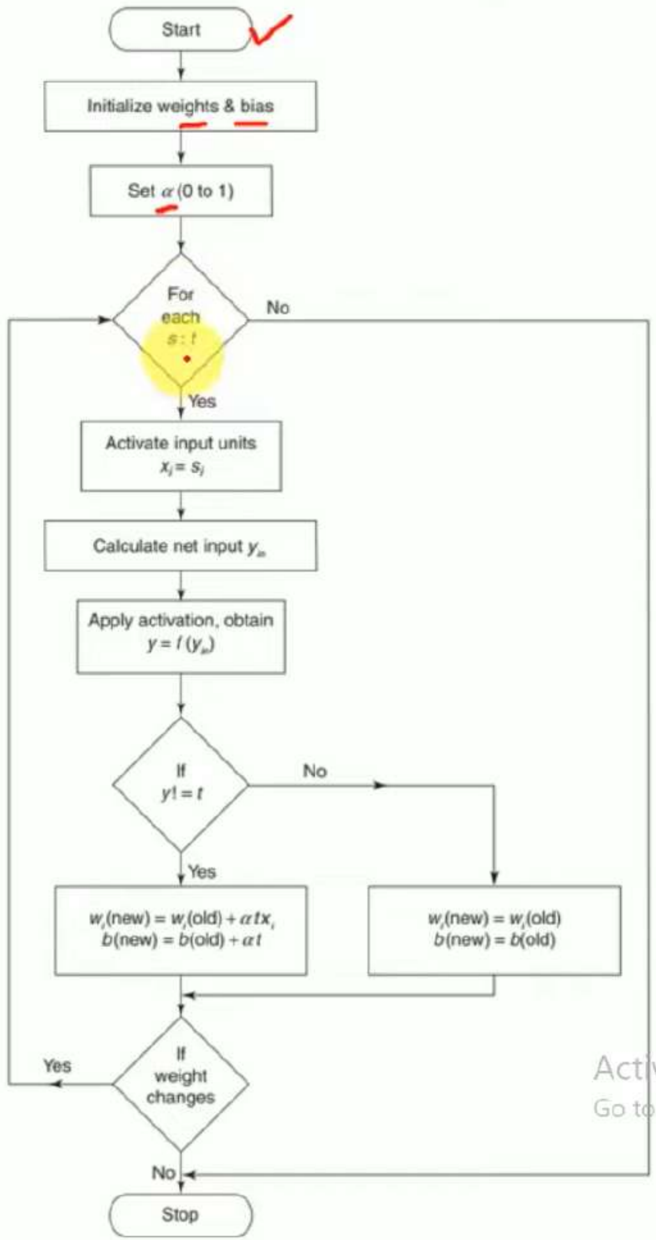
  - Train the network until there is no weight change.



Flowchart of

Perceptron Learning Rule

Single Output Class



Perceptron Learning Rule

- In case of the 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.

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

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

If $y \neq t$, then

$$w(\text{new}) = w(\text{old}) + \alpha t x \quad (\alpha - \text{learning rate})$$

else, we have

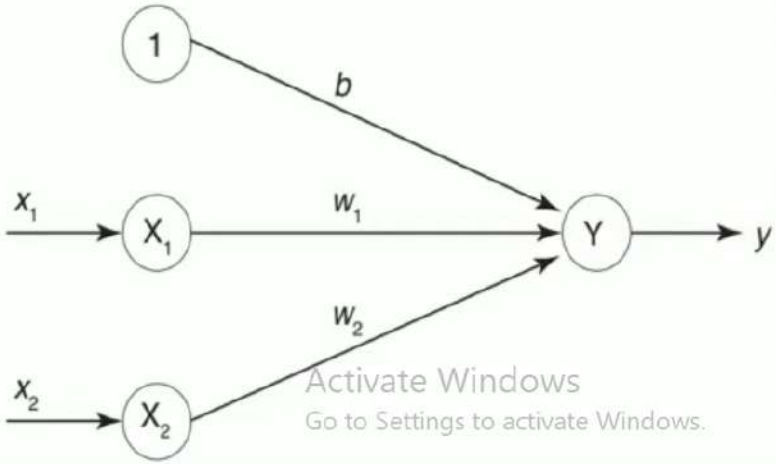
$$w(\text{new}) = w(\text{old})$$

- Weights are updated using the formula

AND function using Perceptron Rule Solved Example

- The perceptron network, which uses perceptron learning rule, is used to train the AND function.
- The input patterns are presented to the network one by one.
- When all the four input patterns are presented, then one epoch is said to be completed.
- The initial weights and threshold are set to zero.
- The learning rate a is set equal to 1.

x_1	x_2	t
1	1	1
1	−1	−1
−1	1	−1
−1	−1	−1



AND function using Perceptron Rule Solved Example

$1 + 1 + (-1)$

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

$0 + 1 \times 0 + 1 \times 0 = 0$

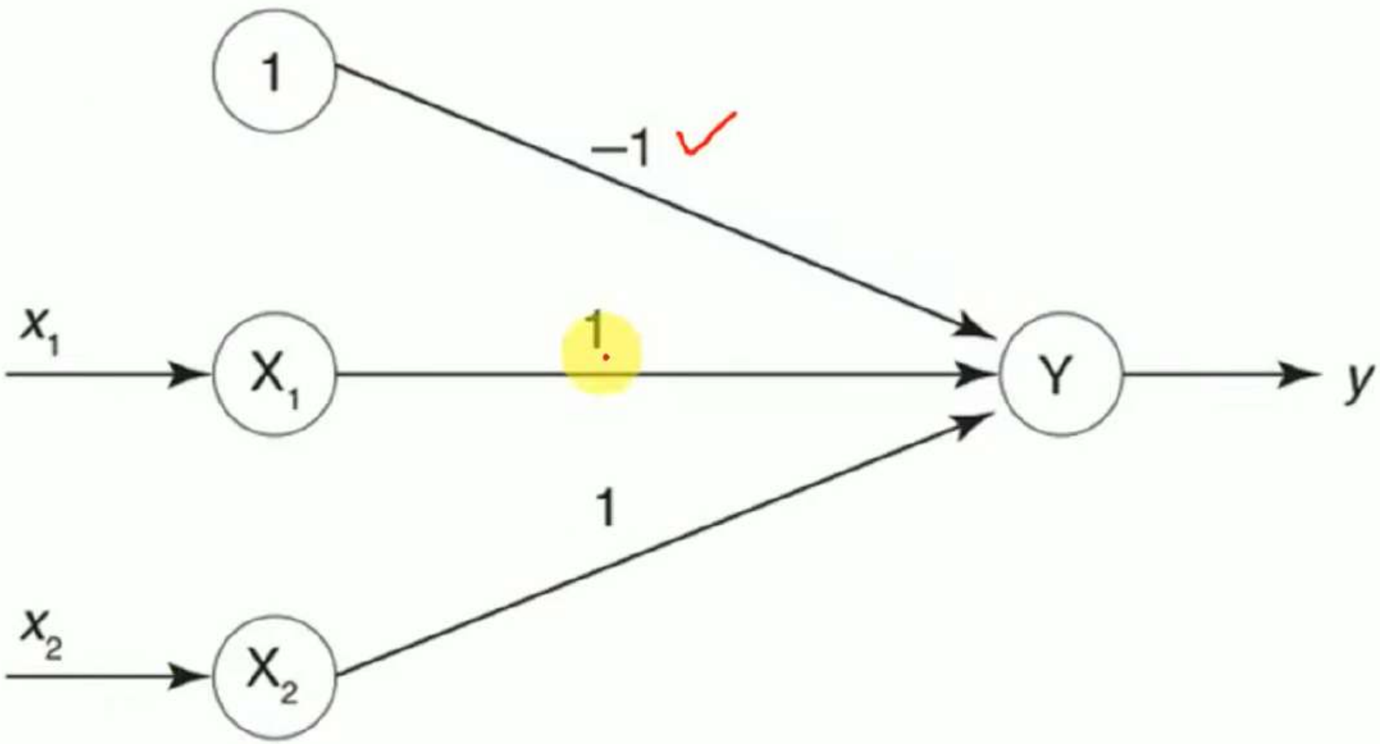
$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > 0 \\ 0 & \text{if } y_{in} = 0 \\ -1 & \text{if } y_{in} < 0 \end{cases}$$

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

$\alpha = 1$

Input		Target	Net input	Calculated output	Weight changes			Weights		
x_1	x_2	(t)	(y_{in})	(y)	Δw_1	Δw_2	Δb	w_1 (0	w_2 0	b 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										
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

AND function using Perceptron Rule Solved Example



Perceptron Network (Rule) Solved Example

- Find the weights required to perform the following classification using perceptron network.
- The vectors (1, 1, 1, 1) and (- 1, 1 - 1, - 1) are belonging to the class 1, vectors (1, 1, 1, - 1) and (1, - 1, - 1, 1) are belonging to the class -1.
- Assume learning rate as 1
- and Initial weights as 0.

Input					Target
x_1	x_2	x_3	x_4	b	(t)
1	1	1	1	1	1
−1	1	−1	−1	1	1
1	1	1	−1	1	−1
1	−1	−1	1	1	−1

Perceptron Network (Rule) Solved Example

$$y_{in} = b + x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4$$

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

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

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

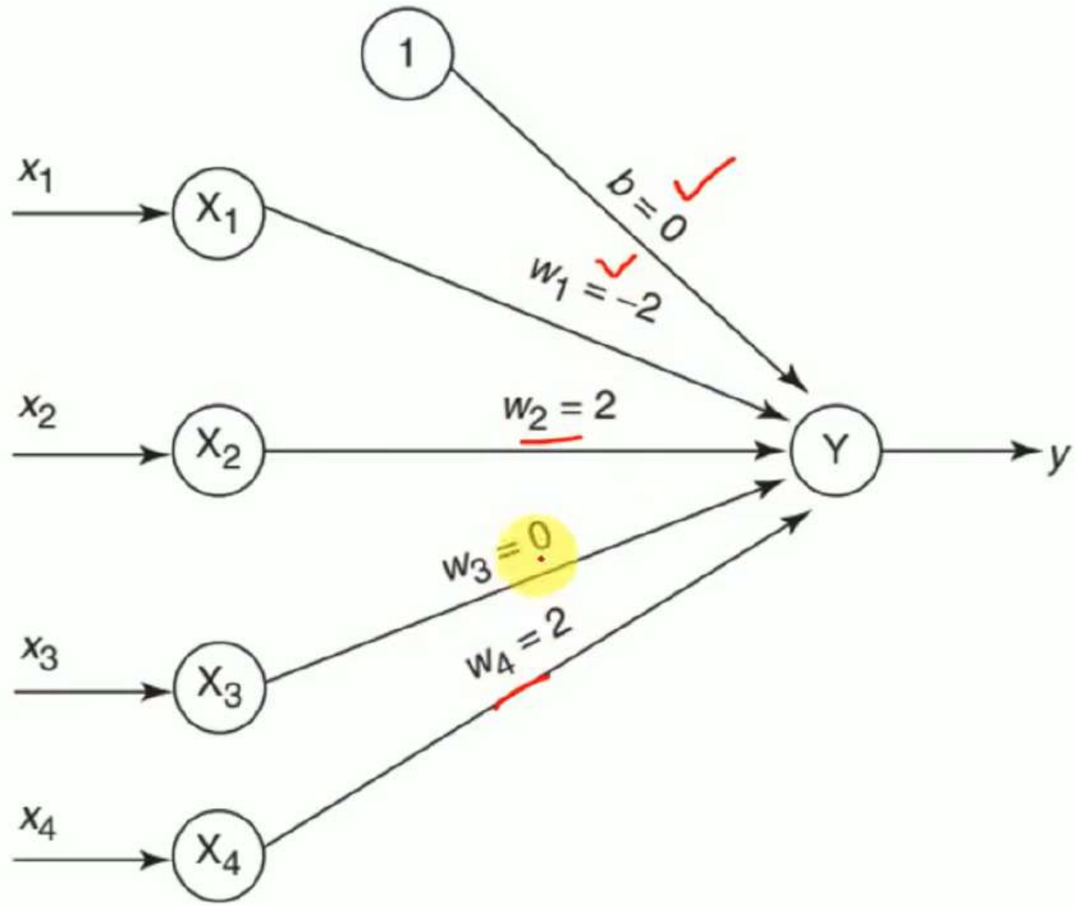
$$\Delta w_3 = \alpha t x_3;$$

$$\Delta w_4 = \alpha t x_4;$$

$$\Delta b = \alpha t$$

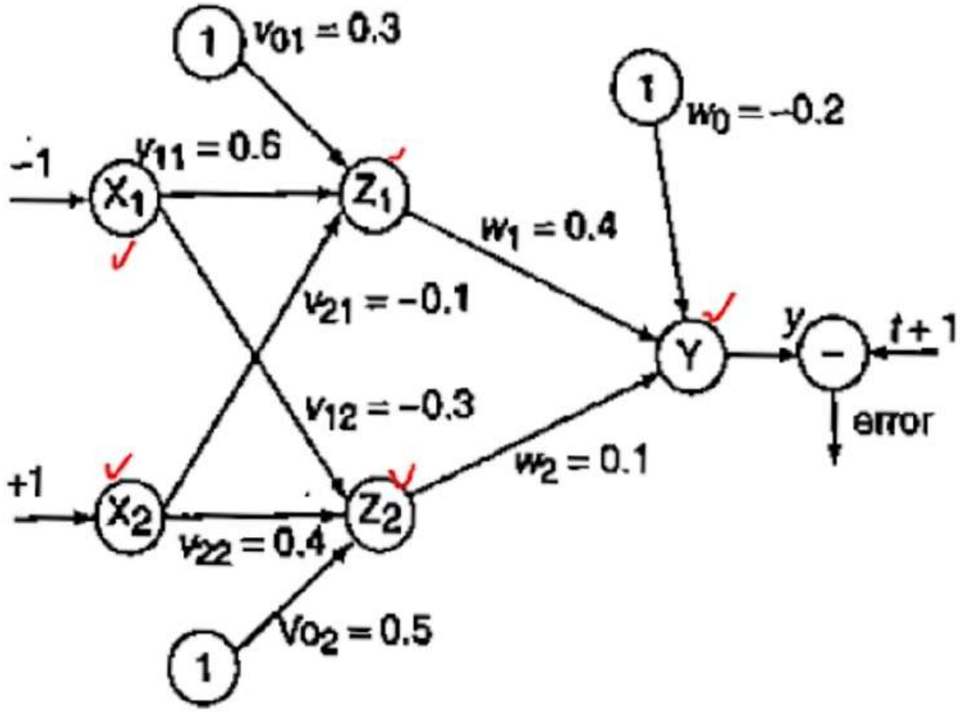
Inputs				Target	Net input	output	Weight changes					Weights				
(x_1	x_2	x_3	x_4	(t)	(y_{in})	(y)	(Δw_1	Δw_2	Δw_3	Δw_4	Δb)	w_1 (0	w_2 0	w_3 0	w_4 0	b 0)
EPOCH-1																
(1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1
(-1	1	-1	-1	1	-1	-1	-1	1	-1	-1	1	0	2	0	0	2
(1	1	1	-1	-1	4	1	-1	-1	-1	1	-1	-1	1	-1	1	1
(1	-1	-1	1	-1	1	1	-1	1	1	-1	-1	-2	2	0	0	0
EPOCH-2																
(1	1	1	1	1	0	0	1	1	1	1	1	-1	3	1	1	1
(-1	1	-1	-1	1	3	1	0	0	0	0	0	-1	3	1	1	1
(1	1	1	-1	-1	4	1	-1	-1	-1	1	-1	-2	2	0	2	0
(1	-1	-1	1	-1	-2	-1	0	0	0	0	0	-2	2	0	2	0
EPOCH-3																
(1	1	1	1	1	2	1	0	0	0	0	0	-2	2	0	2	0
(-1	1	-1	-1	1	2	1	0	0	0	0	0	-2	2	0	2	0
(1	1	1	-1	-1	-2	-1	0	0	0	0	0	-2	2	0	2	0
(1	-1	-1	1	-1	-2	-1	0	0	0	0	0	-2	2	0	2	0

Perceptron Network (Rule) Solved Example



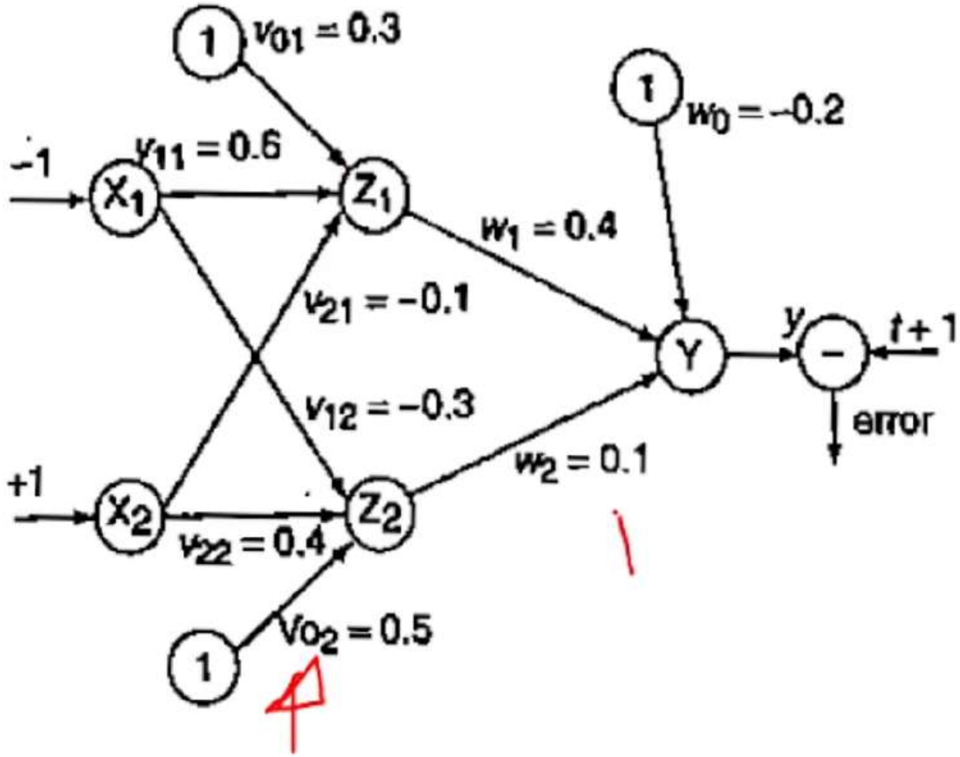
How to Find New Weights – Back Propagation Algorithm

- Using back-propagation network, find the new weights for the figure shown.
- It is presented with the input pattern [-1, 1] and the target output is 1.
- Use a learning rate $\alpha = 0.25$ and binary sigmoidal activation function.



How to Find New Weights – Back Propagation Algorithm

- The new weights are calculated based on the back propagation training algorithm.
- The initial weights are
- $[v_{11}, v_{21}, v_{01}] = [0.6, -0.1, 0.3]$
- $[v_{12}, v_{22}, v_{02}] = [-0.3, 0.4, 0.5]$
- $[w_1, w_2, w_0] = [0.4, 0.1, -0.2]$
- and the learning' rate is $\alpha = 0.25$



How to Find New Weights – Back Propagation Algorithm

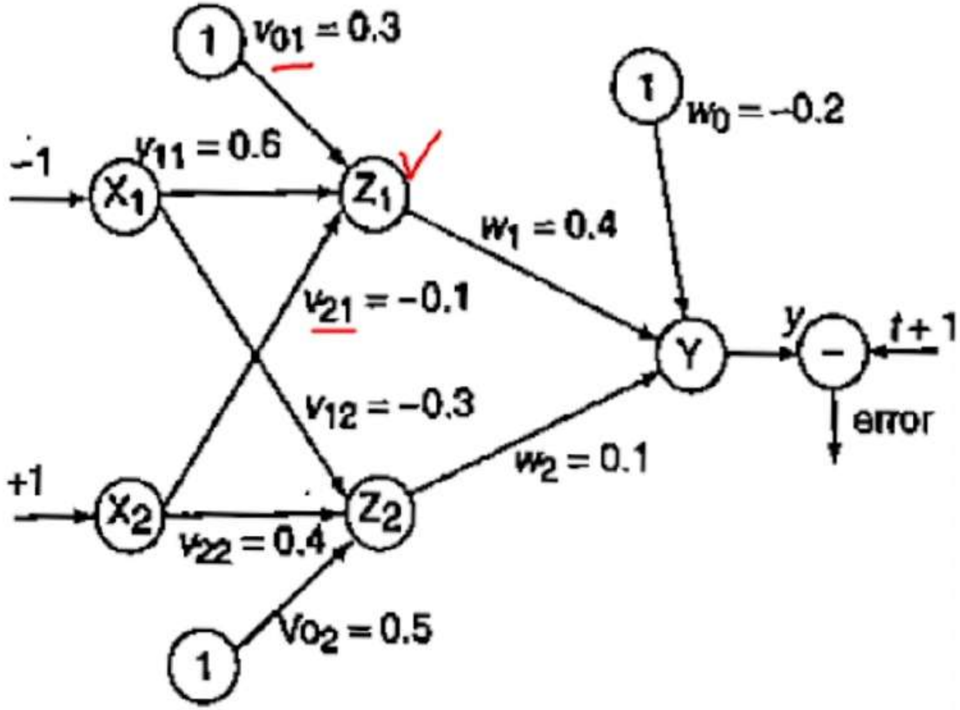
Activation function used is binary sigmoidal activation function and is given by

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

Given the input sample $[x_1, x_2] = [-1, 1]$ and target $t = 1$:

- Calculate the net input: For z_1 layer

$$z_{in1} = v_{01} + x_1 v_{11} + x_2 v_{21}$$
$$= 0.3 + (-1) \times 0.6 + 1 \times -0.1 = -0.4$$



How to Find New Weights –

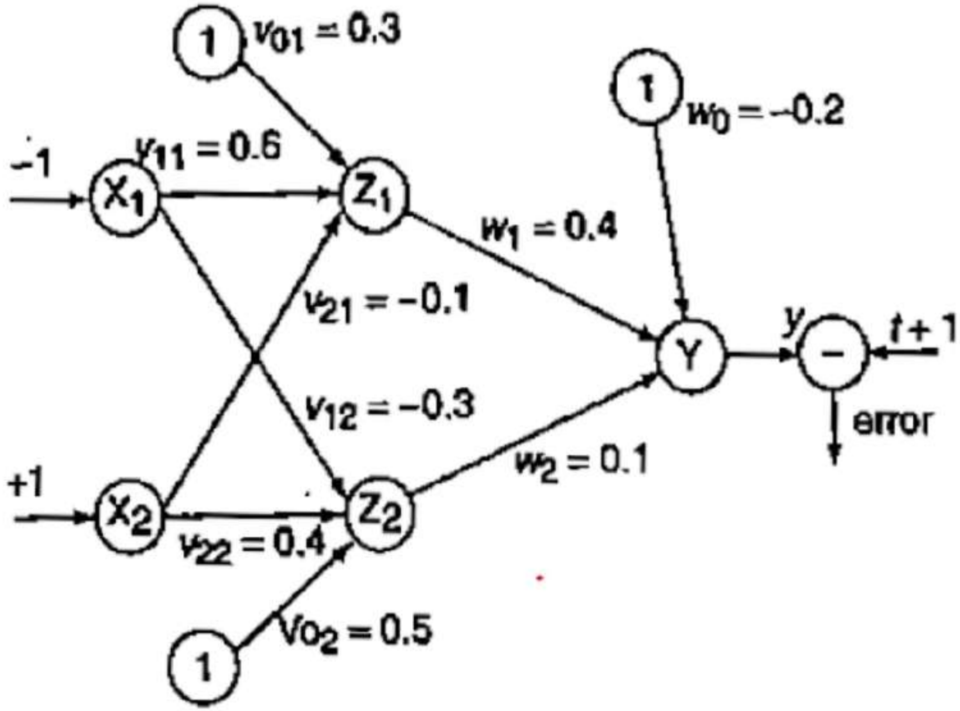
For z_2 layer

$$z_{in2} = v_{02} + x_1 v_{12} + x_2 v_{22}$$
$$= 0.5 + (-1) \times -0.3 + 1 \times 0.4 = 1.2$$

Applying activation to calculate the output, we obtain

$$z_1 = f(z_{in1}) = \frac{1 - e^{-z_{in1}}}{1 + e^{-z_{in1}}} = \frac{1 - e^{0.4}}{1 + e^{0.4}} = -0.1974$$

$$z_2 = f(z_{in2}) = \frac{1 - e^{-z_{in1}}}{1 + e^{-z_{in2}}} = \frac{1 - e^{-1.2}}{1 + e^{-1.2}} = 0.537$$



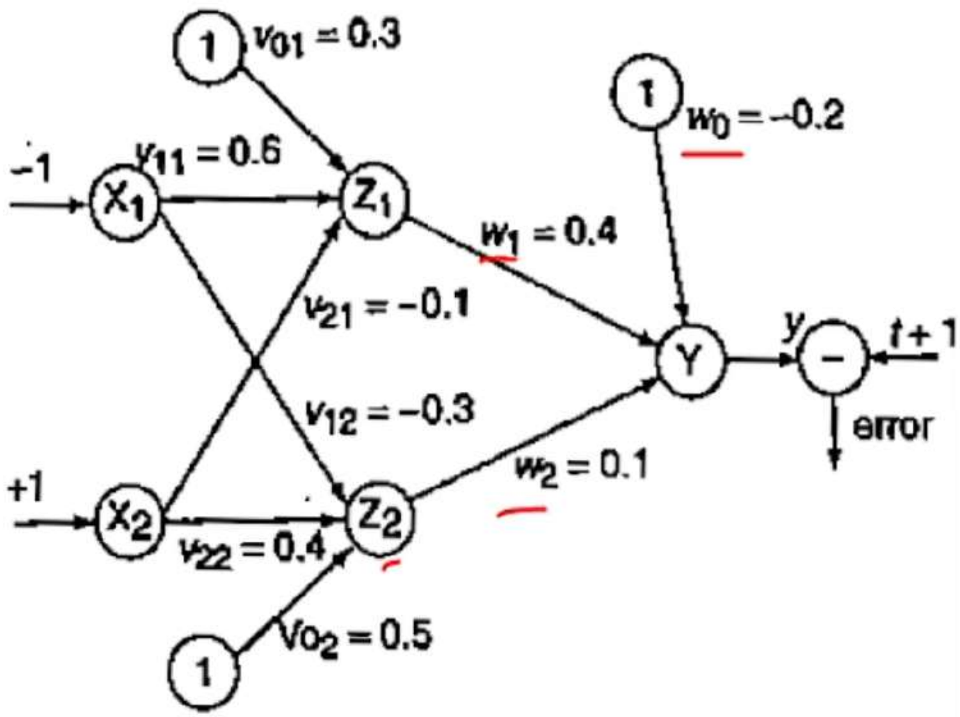
How to Find New Weights – Back Propagation Algorithm

Calculate the net input entering the output layer.
For y layer

$$y_{in} = w_0 + z_1 w_1 + z_2 w_2$$
$$= -0.2 + (-0.1974) \times 0.4 + 0.537 \times 0.1$$
$$= -0.22526$$

Applying activations to calculate the output, we obtain

$$y = f(y_{in}) = \frac{1 - e^{-y_{in}}}{1 + e^{-y_{in}}} = \frac{1 - e^{0.22526}}{1 + e^{0.22526}} = -0.1122$$



How to Find New Weights –

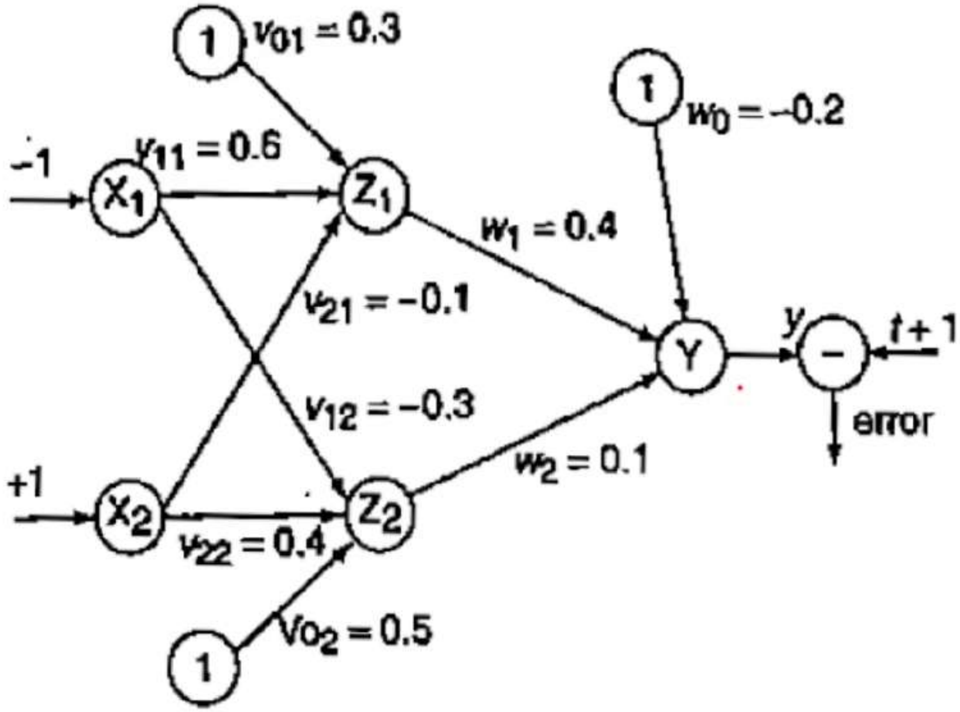
Compute the error portion δ_k:

$$\delta_k = (t_k - y_k) f'(y_{ink})$$

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

Now

$$f'(y_{in}) = 0.5[1 + f(y_{in})]\,[1 - f(y_{in})]$$
$$= 0.5[1 - 0.1122][1 + 0.1122] = 0.4937$$



How to Find New Weights – Back Propagation Algorithm

This implies

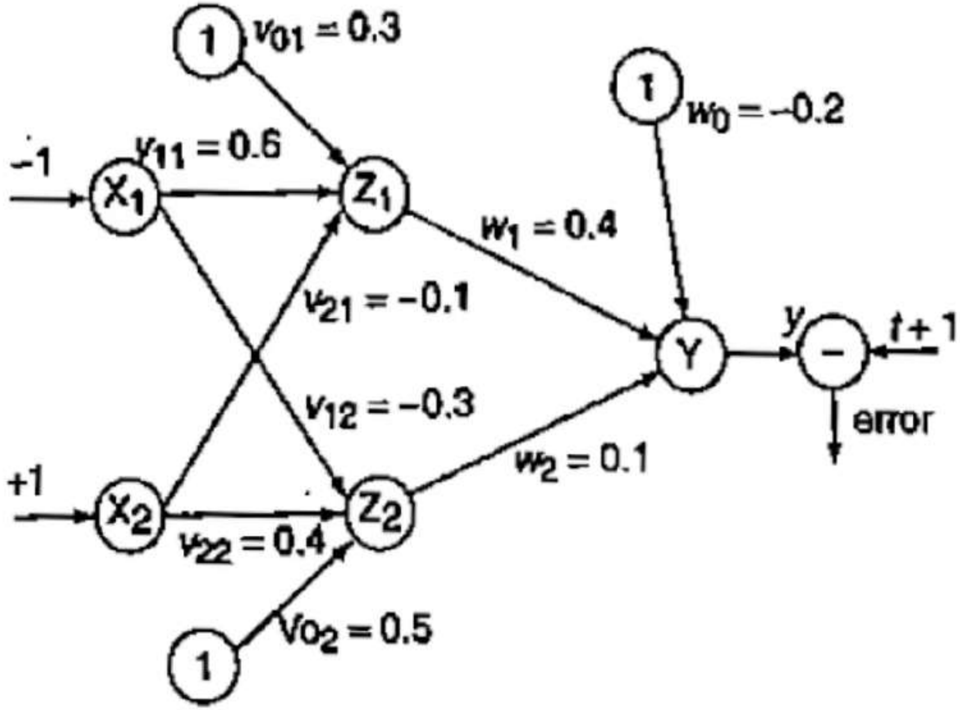
$$\delta_1 = (1 + 0.1122)\,(0.4937) = 0.5491$$

Find the changes in weights between hidden and output layer:

$$\Delta w_1 = \alpha \delta_1 z_1 = 0.25 \times 0.5491 \times -0.1974$$
$$= -0.0271$$

$$\Delta w_2 = \alpha \delta_1 z_2 = 0.25 \times 0.5491 \times 0.537 = 0.0737$$

$$\Delta w_0 = \alpha \delta_1 = 0.25 \times 0.5491 = 0.1373$$



How to Find New Weights – Back Propagation Algorithm

Compute the error portion δ_j between input and hidden layer ($j = 1$ to 2):

$$\delta_j = \delta_{inj} f'(z_{inj})$$

$$\delta_{inj} = \sum_{k=1}^{m} \delta_k w_{jk}$$

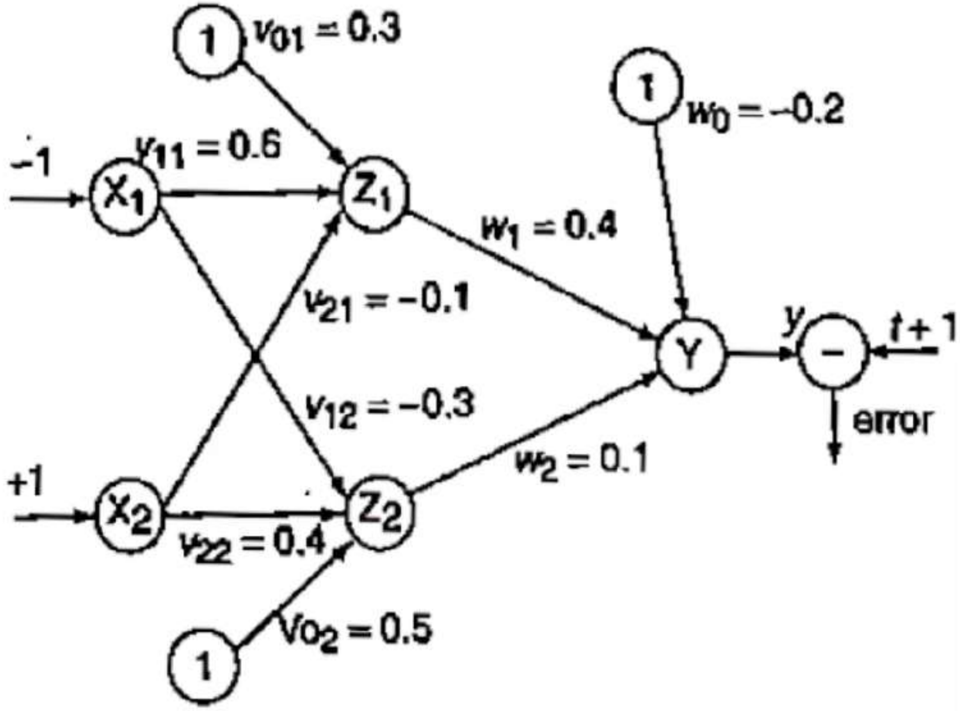
$$\delta_{inj} = \delta_1 w_{j1} \quad [\because \text{only one output neuron}]$$

$$\Rightarrow \delta_{in1} = \delta_1 w_{11} = 0.5491 \times 0.4 = 0.21964$$

$$\Rightarrow \delta_{in2} = \delta_1 w_{21} = 0.5491 \times 0.1 = 0.05491$$

$$\text{Error, } \delta_1 = \delta_{in1} f'(z_{in1}) = 0.21964 \times 0.5 \times (1 + 0.1974)(1 - 0.1974) = 0.1056$$

$$\text{Error, } \delta_2 = \delta_{in2} f'(z_{in2}) = 0.05491 \times 0.5 \times (1 - 0.537)(1 + 0.537) = 0.0195$$



How to Find New Weights – Back Propagation Algorithm

Now find the changes in weights between input and hidden layer:

$$\Delta v_{11} = \alpha \delta_1 x_1 = 0.25 \times 0.1056 \times -1 = -0.0264$$

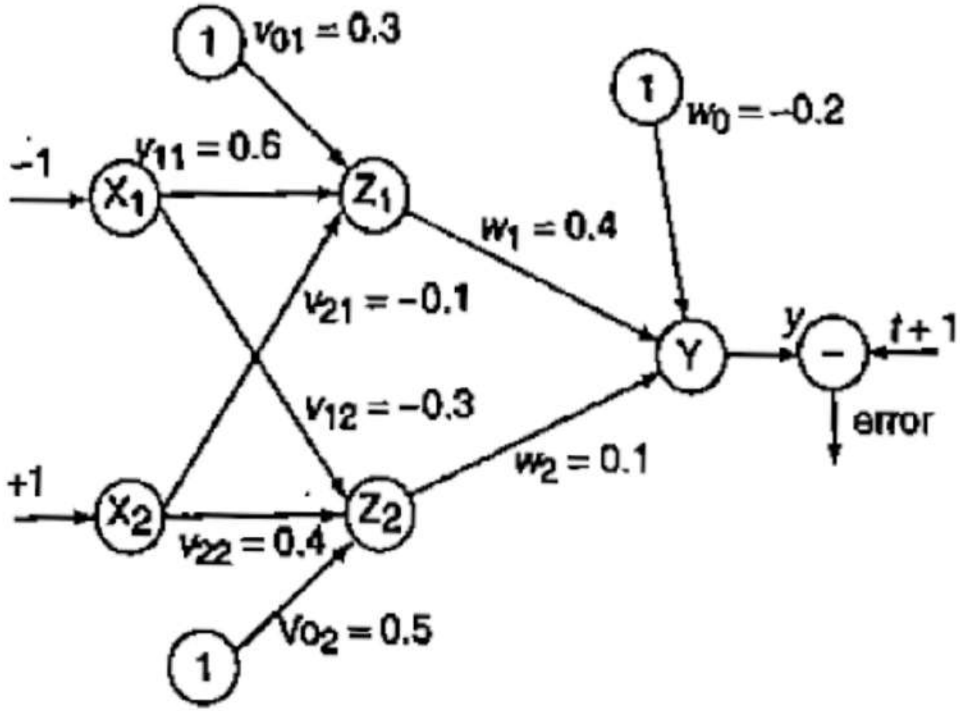
$$\Delta v_{21} = \alpha \delta_1 x_2 = 0.25 \times 0.1056 \times 1 = 0.0264$$

$$\Delta v_{01} = \alpha \delta_1 = 0.25 \times 0.1056 = 0.0264$$

$$\Delta v_{12} = \alpha \delta_2 x_1 = 0.25 \times 0.0195 \times -1 = -0.0049$$

$$\Delta v_{22} = \alpha \delta_2 x_2 = 0.25 \times 0.0195 \times 1 = 0.0049$$

$$\Delta v_{02} = \alpha \delta_2 = 0.25 \times 0.0195 = 0.0049$$



How to Find New Weights – Back Propagation Algorithm

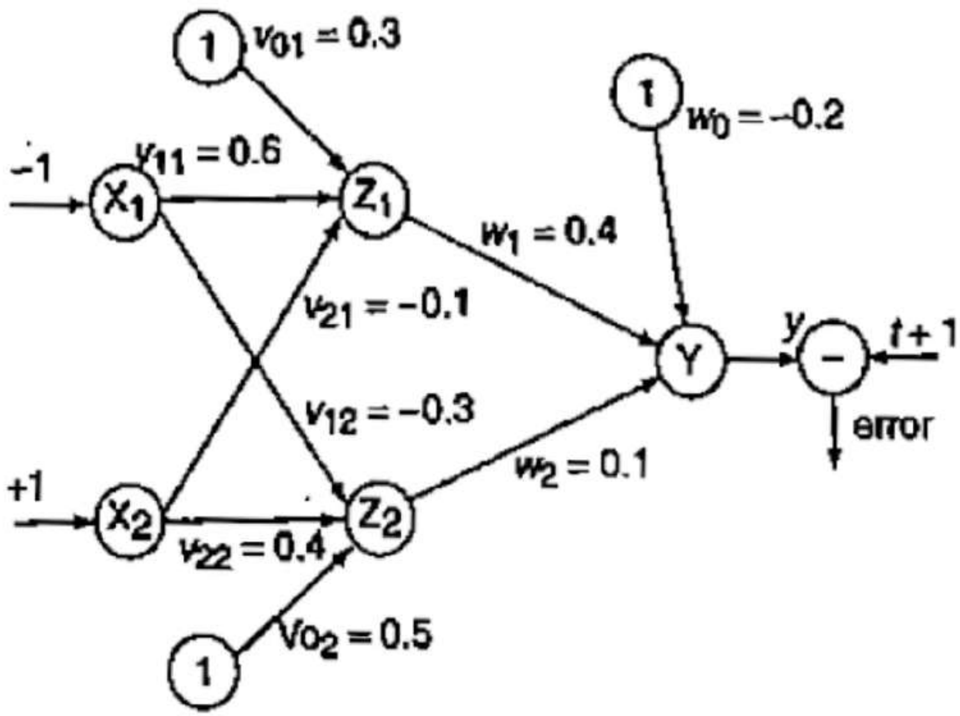
$$v_{11}(\text{new}) = v_{11}(\text{old}) + \Delta v_{11} = 0.6 - 0.0264$$
$$= 0.5736$$

$$v_{12}(\text{new}) = v_{12}(\text{old}) + \Delta v_{12} = -0.3 - 0.0049$$
$$= -0.3049$$

$$v_{21}(\text{new}) = v_{21}(\text{old}) + \Delta v_{21} = -0.1 + 0.0264$$
$$= -0.0736$$

$$v_{22}(\text{new}) = v_{22}(\text{old}) + \Delta v_{22} = 0.4 + 0.0049$$
$$= 0.4049$$

$$w_1(\text{new}) = w_1(\text{old}) + \Delta w_1 = 0.4 - 0.0271$$
$$= 0.3729$$



How to Find New Weights – Back Propagation Algorithm

$$w_2(\text{new}) = w_2(\text{old}) + \Delta w_2 = 0.1 + 0.0737$$
$$= 0.1737$$
$$v_{01}(\text{new}) = v_{01}(\text{old}) + \Delta v_{01} = 0.3 + 0.0264$$
$$= 0.3264$$
$$v_{02}(\text{new}) = v_{02}(\text{old}) + \Delta v_{02} = 0.5 + 0.0049$$
$$= 0.5049$$
$$w_0(\text{new}) = w_0(\text{old}) + \Delta w_0 = -0.2 + 0.1373$$
$$= -0.0627$$