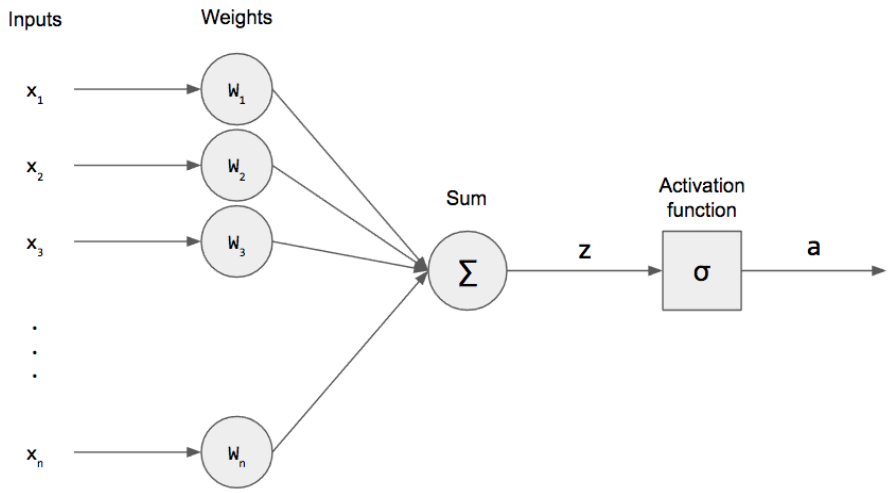
Neural Representation of AND, OR, NOT, XOR and XNOR Logic Gates

(Perceptron Algorithm)



Note: The Perceptron algorithm states that:

Prediction (y `) = 1 if Wx+b > 0 and 0 if Wx+b ≤ 0

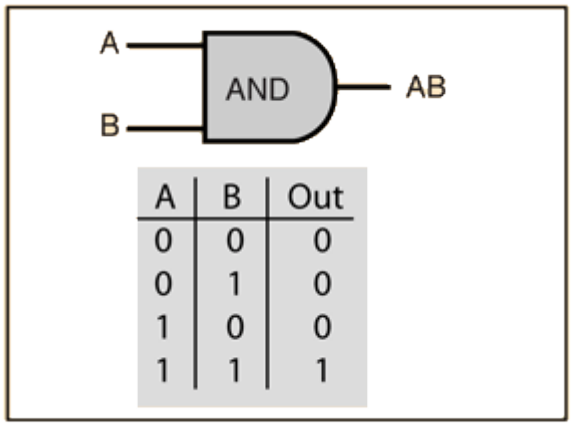
Also, the steps in this method are very similar to how Neural Networks learn, which is as follows;

- Initialize weight values and bias
- Forward Propagate
- Check the error
- Backpropagate and Adjust weights and bias
- Repeat for all training examples

Now that we know the steps, let's get up and running:

AND Gate

From our knowledge of logic gates, we know that an AND logic table is given by the diagram below



A	B	Out
0	0	0
0	1	0
1	0	0
1	1	1

AND Gate

The question is, what are the weights and bias for the AND perceptron?

First, we need to understand that the output of an AND gate is 1 only if both inputs (in this case, x1 and x2) are 1. So, following the steps listed above;

Row 1

- From w1*x1+w2*x2+b, initializing w1, w2, as 1 and b as –1, we get;

x1(1)+x2(1)–1

- Passing the first row of the AND logic table (x1=0, x2=0), we get;

0+0–1 = –1

- From the Perceptron rule, if Wx+b≤0, then y`=0. Therefore, this row is correct, and no need for Backpropagation.

Row 2

- Passing (x1=0 and x2=1), we get;

0+1–1 = 0

- From the Perceptron rule, if Wx+b≤0, then y`=0. This row is correct, as the output is 0 for the AND gate.
- From the Perceptron rule, this works (for both row 1, row 2 and 3).

Row 4

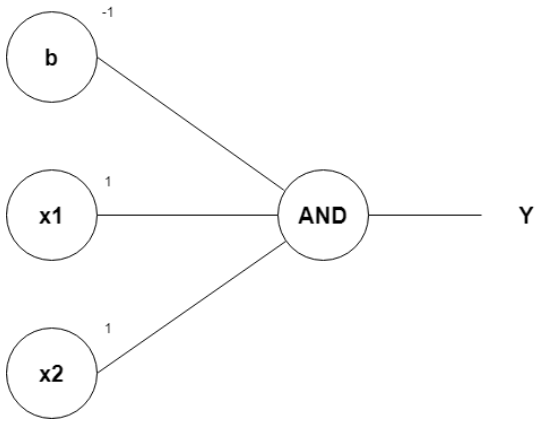
- Passing (x1=1 and x2=1), we get;

1+1–1 = 1

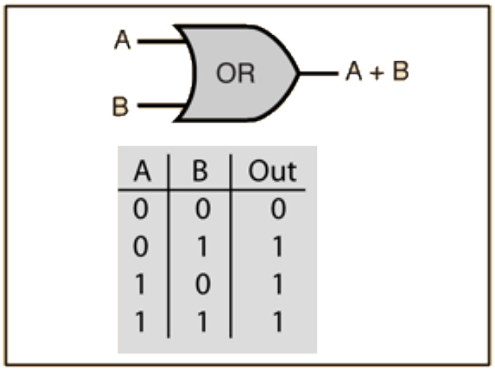
- Again, from the perceptron rule, this is still valid.

Therefore, we can conclude that the model to achieve an AND gate, using the Perceptron algorithm is;

$x1+x2-1$



OR Gate



A	B	Out
0	0	0
0	1	1
1	0	1
1	1	1

OR Gate

From the diagram, the OR gate is 0 only if both inputs are 0.

Row 1

- From w1x1+w2x2+b, initializing w1, w2, as 1 and b as –1, we get;

$x1(1)+x2(1)-1$

- Passing the first row of the OR logic table (x1=0, x2=0), we get;

$0+0-1 = -1$

- From the Perceptron rule, if Wx+b≤0, then y`=0. Therefore, this row is correct.

Row 2

- Passing (x1=0 and x2=1), we get;

$0+1-1 = 0$

- From the Perceptron rule, if Wx+b <= 0, then y`=0. Therefore, this row is incorrect.
- So we want values that will make inputs x1=0 and x2=1 give y` a value of 1. If we change w2 to 2, we have;

$0+2-1 = 1$

- From the Perceptron rule, this is correct for both the row 1 and 2.

Row 3

- Passing (x1=1 and x2=0), we get;

$1+0-1 = 0$

- From the Perceptron rule, if Wx+b <= 0, then y`=0. Therefore, this row is incorrect.
- Since it is similar to that of row 2, we can just change w1 to 2, we have;

$2+0-1 = 1$

- From the Perceptron rule, this is correct for both the row 1, 2 and 3.

Row 4

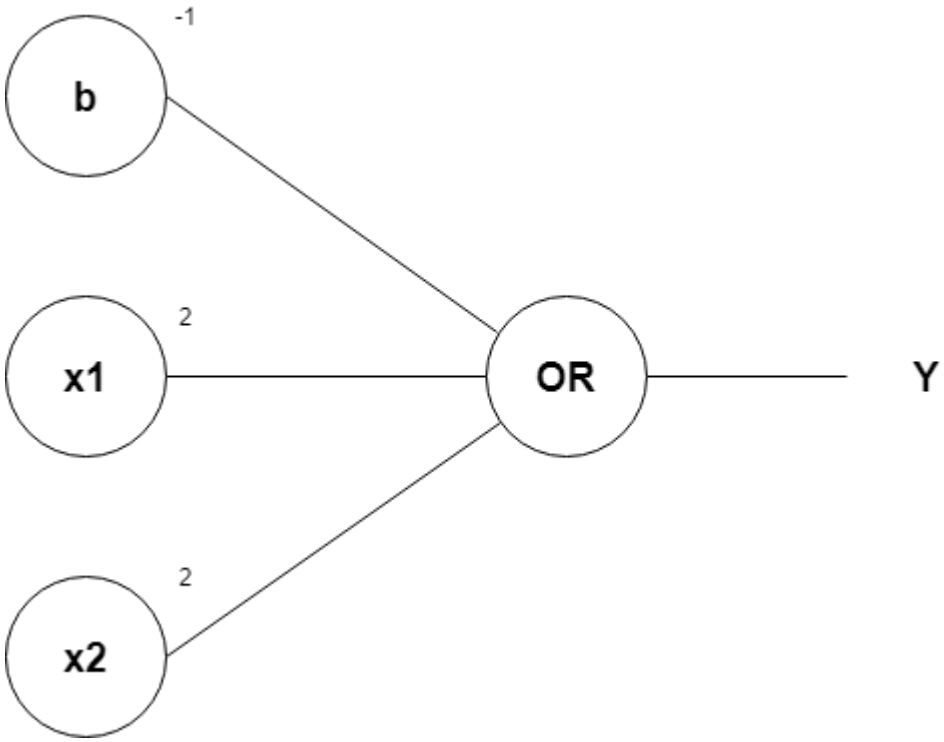
- Passing (x1=1 and x2=1), we get;

$2+2-1 = 3$

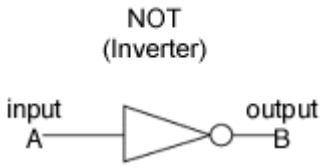
- Again, from the perceptron rule, this is still valid. Quite Easy!

Therefore, we can conclude that the model to achieve an OR gate, using the Perceptron algorithm is;

$2x1+2x2-1$



NOT Gate



A	B
0	1
1	0

NOT Gate

From the diagram, the output of a NOT gate is the inverse of a single input. So, following the steps listed above;

Row 1

- From w1x1+b, initializing w1 as 1 (since single input), and b as –1, we get;

x1(1)–1

- Passing the first row of the NOT logic table (x1=0), we get;

0–1 = –1

- From the Perceptron rule, if Wx+b≤0, then y`=0. This row is incorrect, as the output is 1 for the NOT gate.
- So we want values that will make input x1=0 to give y` a value of 1. If we change b to 1, we have;

0+1 = 1

- From the Perceptron rule, this works.

Row 2

- Passing (x1=1), we get;

1+1 = 2

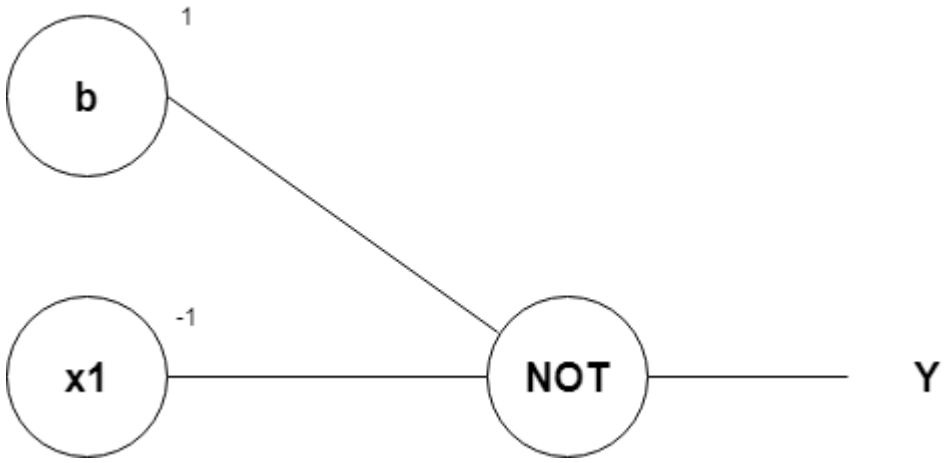
- From the Perceptron rule, if Wx+b > 0, then y`=1. This row is so incorrect, as the output is 0 for the NOT gate.
- So we want values that will make input x1=1 to give y` a value of 0. If we change w1 to −1, we have;

−1+1 = 0

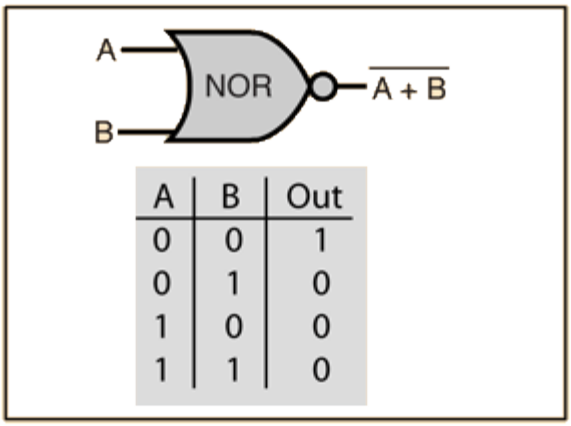
- From the Perceptron rule, if Wx+b ≤ 0, then y`=0. Therefore, this works (for both row 1 and row 2).

Therefore, we can conclude that the model to achieve a NOT gate, using the Perceptron algorithm is;

−x1+1



NOR Gate



A	B	Out
0	0	1
0	1	0
1	0	0
1	1	0

NOR Gate

From the diagram, the NOR gate is 1 only if both inputs are 0.

Row 1

- From w1x1+w2x2+b, initializing w1 and w2 as 1, and b as –1, we get;

x1(1)+x2(1)–1

- Passing the first row of the NOR logic table (x1=0, x2=0), we get;

0+0–1 = –1

- From the Perceptron rule, if Wx+b≤0, then y`=0. This row is incorrect, as the output is 1 for the NOR gate.
- So we want values that will make input x1=0 and x2 = 0 to give y` a value of 1. If we change b to 1, we have;

0+0+1 = 1

- From the Perceptron rule, this works.

Row 2

- Passing (x1=0, x2=1), we get;

0+1+1 = 2

- From the Perceptron rule, if Wx+b > 0, then y`=1. This row is incorrect, as the output is 0 for the NOR gate.
- So we want values that will make input x1=0 and x2 = 1 to give y` a value of 0. If we change w2 to –1, we have;

0–1+1 = 0

- From the Perceptron rule, this is valid for both row 1 and row 2.

Row 3

- Passing (x1=1, x2=0), we get;

1+0+1 = 2

- From the Perceptron rule, if Wx+b > 0, then y`=1. This row is incorrect, as the output is 0 for the NOR gate.
- So we want values that will make input x1=0 and x2 = 1 to give y` a value of 0. If we change w1 to –1, we have;

–1+0+1 = 0

- From the Perceptron rule, this is valid for both row 1, 2 and 3.

Row 4

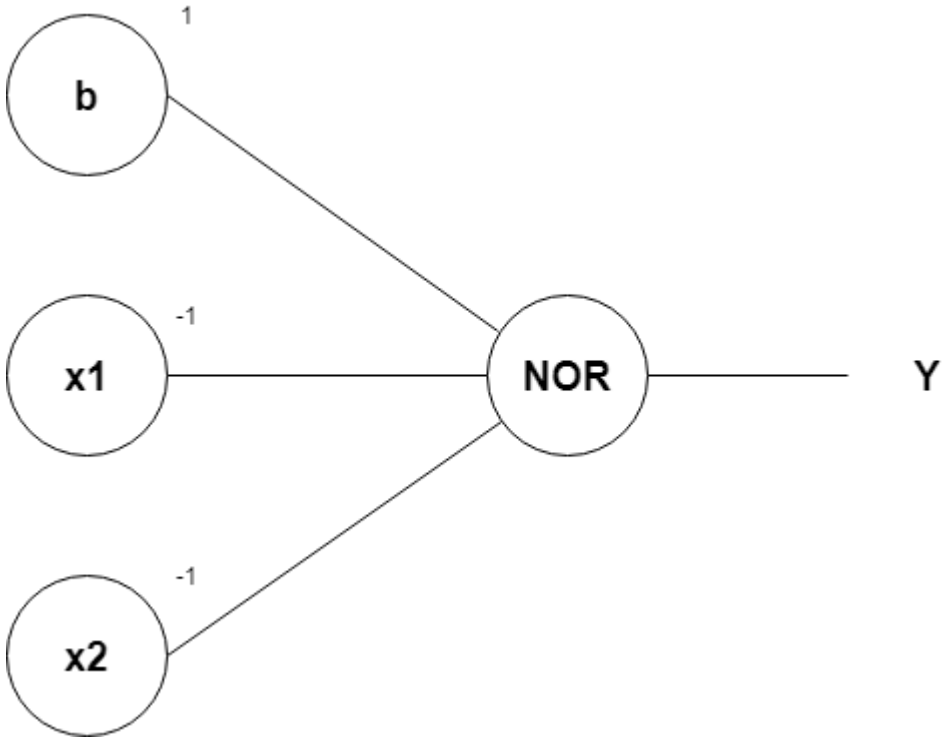
- Passing (x1=1, x2=1), we get;

-1-1+1 = -1

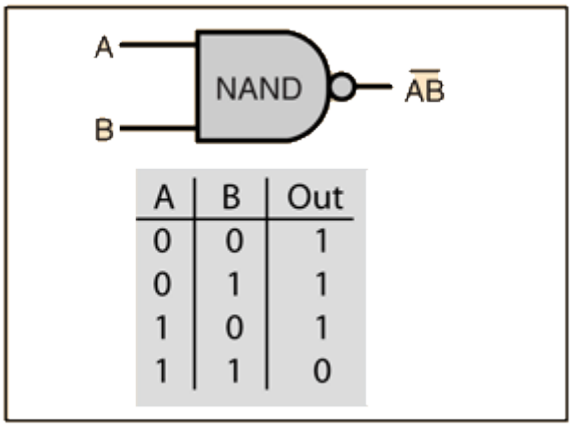
- From the Perceptron rule, this still works.

Therefore, we can conclude that the model to achieve a NOR gate, using the Perceptron algorithm is;

-x1-x2+1



NAND Gate



A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0

From the diagram, the NAND gate is 0 only if both inputs are 1.

Row 1

- From w1x1+w2x2+b, initializing w1 and w2 as 1, and b as -1, we get;

x1(1)+x2(1)-1

- Passing the first row of the NAND logic table (x1=0, x2=0), we get;

0+0-1 = -1

- From the Perceptron rule, if Wx+b≤0, then y`=0. This row is incorrect, as the output is 1 for the NAND gate.
- So we want values that will make input x1=0 and x2 = 0 to give y` a value of 1. If we change b to 1, we have;

0+0+1 = 1

- From the Perceptron rule, this works.

Row 2

- Passing (x1=0, x2=1), we get;

0+1+1 = 2

- From the Perceptron rule, if Wx+b > 0, then y`=1. This row is also correct (for both row 2 and row 3).

Row 4

- Passing (x1=1, x2=1), we get;

1+1+1 = 3

- This is not the expected output, as the output is 0 for a NAND combination of x1=1 and x2=1.

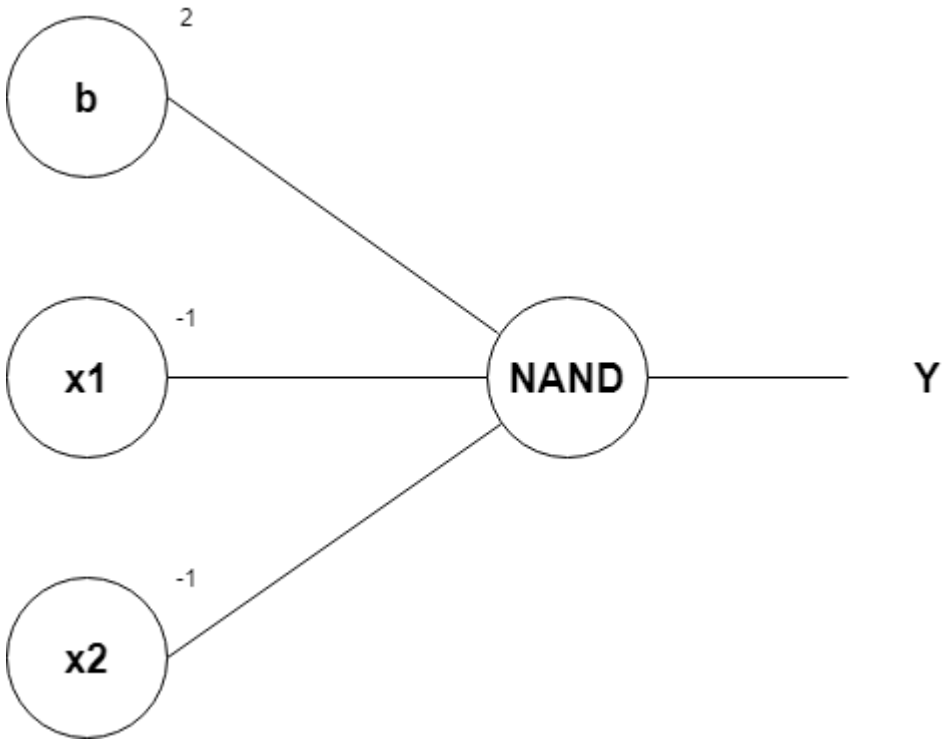
- Changing values of w1 and w2 to -1, and value of b to 2, we get;

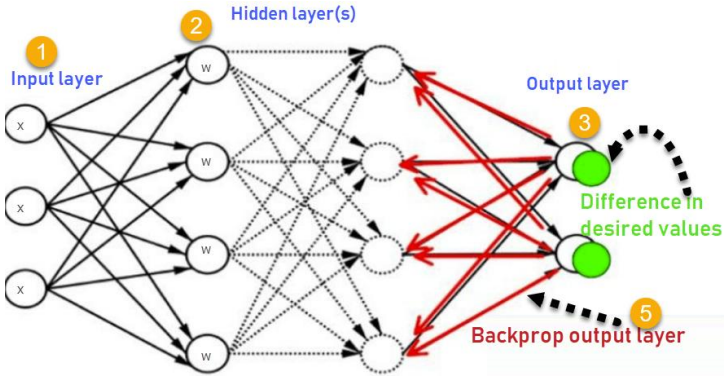
-1-1+2 = 0

- It works for all rows.

Therefore, we can conclude that the model to achieve a NAND gate, using the Perceptron algorithm is;

-x1-x2+2





The Error Backpropagation Training Algorithm (EBPTA), also known as backpropagation, is a widely used training algorithm for feedforward neural networks.

It is a supervised learning algorithm that aims to minimize the difference between the actual and expected output of the network, which is called the error.

The basic idea of backpropagation is to compute the gradient of the error with respect to the weights of the network, and then use this gradient to update the weights in a way that reduces the error.

This is done by propagating the error backwards through the network and computing the derivative of the error with respect to each weight.

The backpropagation algorithm consists of the following steps:

1. Forward Pass: The input vector is fed into the network, and the activation values of each neuron in each layer are computed, starting from the input layer and progressing through the hidden layers to the output layer.
2. Error Computation: The error of the network is computed as the difference between the actual output of the network and the expected output for the given input.
3. Backward Pass: The error is then propagated back through the network from the output layer to the input layer. This is done by computing the error gradient with respect to each weight in each layer, using the chain rule of calculus.
4. Weight Update: The weights in each layer are then updated based on the error gradient computed in the previous step. The update rule typically involves multiplying the gradient by a learning rate, and subtracting the result from the current weight value.
5. Repeat: The above steps are repeated for multiple input/output pairs until the network converges to a solution.

The backpropagation algorithm can be computationally expensive, especially for large networks with many layers and neurons. Therefore, there have been several variations and optimizations of the algorithm proposed over the years to improve its efficiency and convergence properties. Some of these include the use of momentum, adaptive learning rates, and batch training.

Despite its limitations, the backpropagation algorithm remains one of the most widely used and successful training algorithms for neural networks. It has been used in a wide range of applications, including speech recognition, computer vision, and natural language processing.

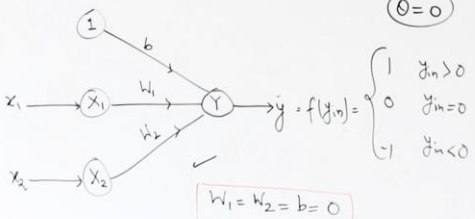
AND Problem

Implement AND function Using PERCEPTRON

Solution:- Truth table for AND function with bipolar Inputs and targets.

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

Perceptron Network



$W_1 = W_2 = b = 0$

Networks for Bipolar Inputs & Targets

First Input Pattern:- $[x_1\ x_2\ t] = [1\ 1\ 1]$

Calculate the Net Input

ie $y_{in} = b + w_1x_1 + w_2x_2$
$= 0 + (0)(1) + (0)(1)$
$= 0$

$\therefore y = f(y_{in}) = 0$

Check $y \neq t$, Hence Weight change is required

$w_i(new) = w_i(old) + \alpha t x_i$ (where α is learning rate) $\alpha = 1$

$w_1(new) = w_1(old) + \alpha t x_1$
$= 0 + (1)(1)(1)$
$= 1$

$w_2(new) = w_2(old) + \alpha t x_2$
$= 0 + (1)(1)(1)$
$= 1$

$b(new) = b(old) + \alpha t$
$= 0 + (1)(1)$
$= 1$

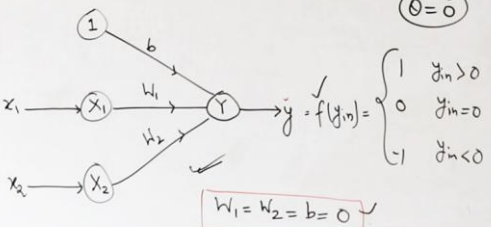
$w_1 = 1$
$w_2 = 1$
$b = 1$

Implement AND function Using PERCEPTRON

Solution:- Truth table for AND function with bipolar Inputs and targets.

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

Perceptron Network



$W_1 = W_2 = b = 0$

Networks for Bipolar Inputs & Targets

Second Input Pattern:- $[x_1\ x_2\ t] = [1\ -1\ -1]$

Calculate the Net Input

ie $y_{in} = b + w_1x_1 + w_2x_2$
$= 1 + (1)(1) + (1)(-1)$
$= 1$

$\therefore y = f(y_{in}) = 1$

Check $y \neq t$, Hence Weight change is required

$w_i(new) = w_i(old) + \alpha t x_i$ (where α is learning rate) $\alpha = 1$

$w_1(new) = w_1(old) + \alpha t x_1$
$= 1 + (1)(-1)(1)$
$= 0$

$w_2(new) = w_2(old) + \alpha t x_2$
$= 1 + (1)(-1)(-1)$
$= 2$

$b(new) = b(old) + \alpha t$
$= 1 + (1)(-1)$
$= 0$

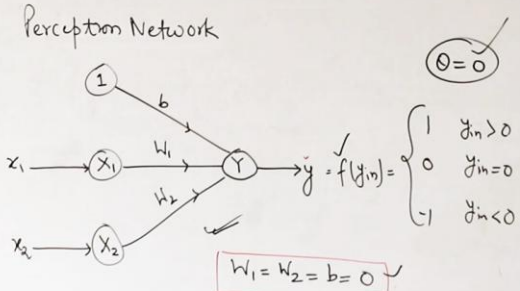
$w_1 = 1$	$w_1 = 0$
$w_2 = 1$	$w_2 = 2$
$b = 1$	$b = 0$

Implement AND function Using PERCEPTRON Networks for Bipolar Inputs & Targets

Solution:- Truth table for AND function with bipolar Inputs and targets.

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

Perceptron Network



$W_1 = W_2 = b = 0$

Third Input Pattern:- $[x_1 \; x_2 \; t] = [-1 \;\; 1 \;\; -1]$

Calculate the Net Input

$$y_{in} = b + W_1 x_1 + W_2 x_2$$
$$= 0 + (0)(-1) + (2)(1)$$
$$= 2$$

$\therefore y = f(y_{in}) = 1$

Check $y \neq t$, Hence weight change is required

$W_i(new) = W_i(old) + \alpha t x_i$ (where α is learning rate)

$\alpha = 1$

$W_1(new) = W_1(old) + \alpha t x_1$
$= 0 + (1)(-1)(-1)$
$= 1$

$W_2(new) = W_2(old) + \alpha t x_2$
$= 2 + (1)(-1)(1)$
$= 1$

$b(new) = b(old) + \alpha t$
$= 0 + (1)(-1)$
$= -1$

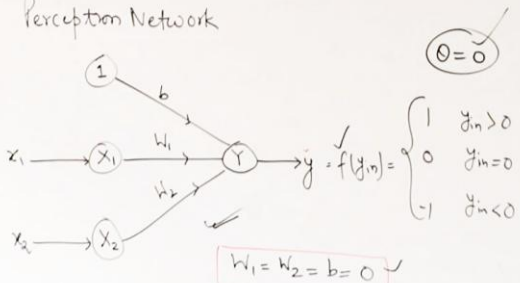
$W_1 = 1$	$W_1 = 0$	$W_1 = 1$
$W_2 = 1$	$W_2 = 2$	$W_2 = 1$
$b = 1$	$b = 0$	$b = -1$

Implement AND function Using PERCEPTRON Networks for Bipolar Inputs & Targets

Solution:- Truth table for AND function with bipolar Inputs and targets.

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

Perceptron Network



$W_1 = W_2 = b = 0$

Fourth Input Pattern:- $[x_1 \; x_2 \; t] = [-1 \;\; -1 \;\; -1]$

Calculate the Net Input

$$y_{in} = b + W_1 x_1 + W_2 x_2$$
$$= (-1) + (1)(-1) + (1)(-1)$$
$$= -3$$

$\therefore y = f(y_{in}) = -1$

Check $y = t$, Hence weight change is not required

(where α is learning rate)

$\alpha = 1$

$W_1 = 1$	$W_1 = 0$	$W_1 = 1$	$W_1 = 1$
$W_2 = 1$	$W_2 = 2$	$W_2 = 1$	$W_2 = 1$
$b = 1$	$b = 0$	$b = -1$	$b = -1$

Table- Training of Perceptron Network

Input x_1	Input x_2	Input 1	Target (t)	Net Input (y_{in})	Calculated Output (y)	Weight Changes ΔW_1	Weight Changes ΔW_2	Weight Changes Δb	Weights W_1	Weights W_2	Weights b
EPOCH-1											
1	1	1	1	0	0	1	1	1	1	1	1
1	-1	1	-1	1	1	-1	1	-1	0	2	0
-1	1	1	-1	2	1	+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	1	0	0	0	1	1	-1
1	-1	1	-1	-1	-1	0	0	0	1	1	-1
-1	1	1	-1	-1	-1	0	0	0	1	1	-1
-1	-1	1	-1	-3	-1	0	0	0	1	1	-1

Now the final weights after 2nd Epoch

$W_1 = 1,\ W_2 = 1,\ b = -1$

Now the Equation for the decision boundary or separating line –

$$b + w_1 x_1 + w_2 x_2 = 0 \quad (\theta = 0)$$

Now $(-1) + 1(x_1) + (1)x_2 = 0$

$\Rightarrow -1 + x_1 + x_2 = 0$

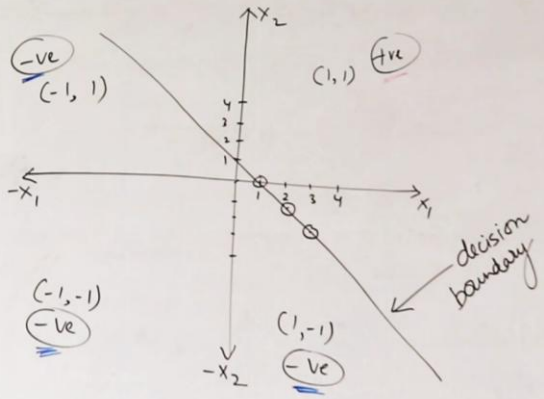
$\Rightarrow \boxed{x_2 = -x_1 + 1}$

$x_1 = 1 \quad x_2 = -1 + 1 = 0$

$x_1 = 2 \quad x_2 = -2 + 1 = -1$

$x_1 = 3 \quad x_2 = -3 + 1 = -2$

Decision boundary for AND in perceptron model



CONCLUSION :- It is clear from above straight line that it separates positive and negative response regions.