

**SUBJECT: Machine learning and Data Science**

**NAME : YASH DATTATRAYA DESAI**

**CLASS : DIV 3**

**SEMESTER/YEAR : 7TH (BE)**

**ROLL NO : C43414**

**EXPERIMENT NO- 03**

**TITLE:** Implementation of basic logic gates.

**AIM:** To implement basic logic gates using Hebbnet Neural Networks.

**SOFTWARES:**

SOFTWARE	VERSION
Jupyter Notebook	V5.1

**THEORY:**

**Hebbian Learning Rule with Implementation of AND Gate**

Hebbian Learning Rule, also known as Hebb Learning Rule, was proposed by Donald O Hebb. It is one of the first and also easiest learning rules in the neural network. It is used for pattern classification. It is a single layer neural network, i.e. it has one input layer and one output layer. The input layer can have many units, say n. The output layer only has one unit. Hebbian rule works by updating the weights between neurons in the neural network for each training sample.

**Hebbian Learning Rule Algorithm :**

1. Set all weights to zero,  $w_i = 0$  for  $i=1$  to  $n$ , and bias to zero.
2. For each input vector,  $S(\text{input vector}) : t(\text{target output pair})$ , repeat steps 3-5.
3. Set activations for input units with the input vector  $X_i = S_i$  for  $i = 1$  to  $n$ .
4. Set the corresponding output value to the output neuron, i.e.  $y = t$ .
5. Update weight and bias by applying Hebb rule for all  $i = 1$  to  $n$ :

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

$$b(\text{new}) = b(\text{old}) + y$$

**Implementing AND Gate :**

INPUT				TARGET	
	$x_1$	$x_2$	$b$		$y$
$X_1$	-1	-1	1	$Y_1$	-1
$X_2$	-1	1	1	$Y_2$	-1
$X_3$	1	-1	1	$Y_3$	-1
$X_4$	1	1	1	$Y_4$	1

*Truth Table of AND Gate using bipolar sigmoidal function*

There are 4 training samples, so there will be 4 iterations. Also, the activation function used here is Bipolar Sigmoidal Function so the range is  $[-1,1]$ .

**Step 1 :**

Set weight and bias to zero,  $w = [0\ 0\ 0]^T$  and  $b = 0$ .

**Step 2 :**

Set input vector  $X_i = S_i$  for  $i = 1$  to 4.

$$X_1 = [-1\ -1\ 1]^T$$

$$X_2 = [-1\ 1\ 1]^T$$

$$X_3 = [1\ -1\ 1]^T$$

$$X_4 = [1\ 1\ 1]^T$$

**Step 3 :**

Output value is set to  $y = t$ .

**Step 4 :**

Modifying weights using Hebbian Rule:

First iteration –

$$w(\text{new}) = w(\text{old}) + x_1 y_1 = [0\ 0\ 0]^T + [-1\ -1\ 1]^T \cdot [-1] = [1\ 1\ -1]^T$$

For the second iteration, the final weight of the first one will be used and so on.

Second iteration –

$$w(\text{new}) = [1\ 1\ -1]^T + [-1\ 1\ 1]^T \cdot [-1] = [2\ 0\ -2]^T$$

Third iteration –

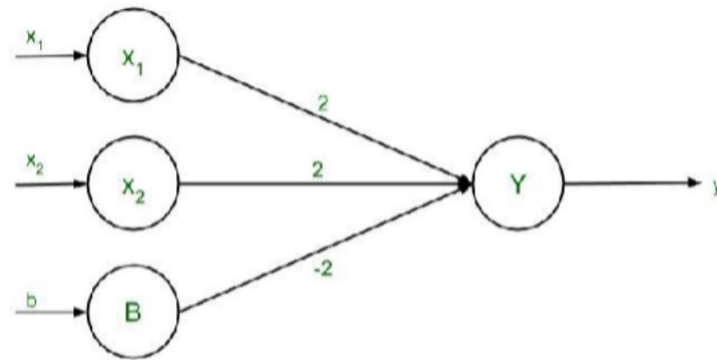
$$w(\text{new}) = [2\ 0\ -2]^T + [1\ -1\ 1]^T \cdot [-1] = [1\ 1\ -3]^T$$

Fourth iteration –

$$w(\text{new}) = [1\ 1\ -3]^T + [1\ 1\ 1]^T \cdot [1] = [2\ 2\ -2]^T$$

So, the final weight matrix is  $[2\ 2\ -2]^T$

**Testing the network :**



*The network with the final weights*

For  $x_1 = -1, x_2 = -1, b = 1, Y = (-1)(2) + (-1)(2) + (1)(-2) = -6$

For  $x_1 = -1, x_2 = 1, b = 1, Y = (-1)(2) + (1)(2) + (1)(-2) = -2$

For  $x_1 = 1, x_2 = -1, b = 1, Y = (1)(2) + (-1)(2) + (1)(-2) = -2$

For  $x_1 = 1, x_2 = 1, b = 1, Y = (1)(2) + (1)(2) + (1)(-2) = 2$

The results are all compatible with the original table.

### **Decision Boundary :**

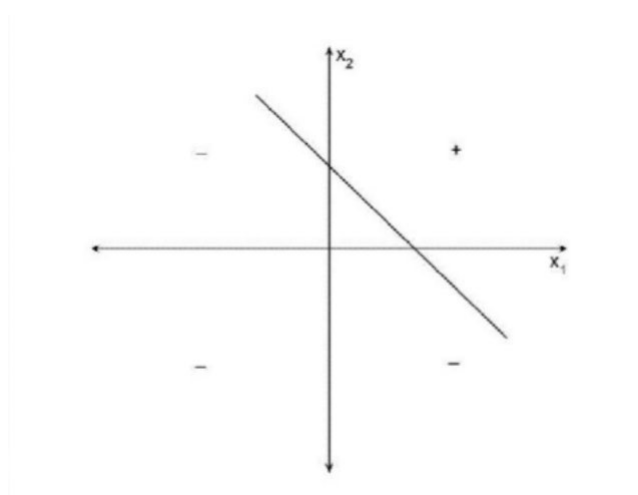
$$2x_1 + 2x_2 - 2b = y$$

Replacing  $y$  with  $0, 2x_1 + 2x_2 - 2b = 0$

Since bias,  $b = 1$ , so  $2x_1 + 2x_2 - 2(1) = 0$

$$2(x_1 + x_2) = 2$$

The final equation,  $x_2 = -x_1 + 1$



*Decision Boundary of AND Function*

Implementation of Artificial Neural Network for AND Logic Gate with 2-bit Binary Input  
Artificial Neural Network (ANN) is a computational model based on the biological neural networks of animal brains. ANN is modeled with three types

Smt. Kashibai Navale College of Engineering Vadgaon Pune, Computer Dept

---

of layers: an input layer, hidden layers (one or more), and an output layer. Each layer comprises nodes (like biological neurons) are called Artificial Neurons. All nodes are connected with weighted edges (like synapses in biological brains) between two layers. Initially, with the forward propagation function, the output is predicted. Then through backpropagation, the weight and bias to the nodes are updated to minimizing the error in prediction to attain the convergence of cost function in determining the final output. AND logical function truth table for 2-bit binary variables, i.e, the input vector and the corresponding output.

X1	X2	Y
0	0	0
0	1	0
1	0	0
1	1	1

### APPROACH:

- Step1:** Import the required Python libraries
- Step2:** Define Activation Function: Sigmoid Function
- Step3:** Initialize neural network parameters (weights, bias) and define model hyperparameters (number of iterations, learning rate)
- Step4:** Forward Propagation
- Step5:** Backward Propagation
- Step6:** Update weight and bias parameters
- Step7:** Train the learning model
- Step8:** Plot Loss value vs Epoch
- Step9:** Test the model performance

### CONCLUSION:

We implemented the logic gates using Hebbnet Neural Network and got the outputs respectively.