Terna Engineering College

Department of Artificial Intelligence and Data Science

Program : Sem VI

Course: Machine Learning Lab

Experiment No.08

PART A

(PART A: TO BE REFERRED BY STUDENTS)

A.1 Aim: To implement Hebbian Learning using Python.

A.2 Theory:

Hebbian Learning Rule:

Hebbian Learning Rule, also known as the Hebb Learning Rule, was proposed by Donald O Hebb. It is one of the first and 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(input vector) : t(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:

$$w_i(new) = w_i(old) + x_i y$$

$$b(new) = b(old) + y$$

Importance of Hebbian Learning :

Hebbian learning is an unsupervised method, requiring no other information that the activates such as labels or error signals. It is based on the fundamental principle of biological learning.

Hebbian Learning can strengthen the neural response that is done by an input. This can be useful if the response made is appropriate to the situation, but can be counterproductive if a different response is more appropriate.

Limitations of Hebbian Learning:

- Hebb's principle does not cover all forms of synaptic long-term plasticity.
- Hebb did not give any rules for inhibitory synapses.
- He also did not make predictions for any anti-casual spike sequences.

e.g.

Let us implement logical AND function with bipolar inputs using Hebbian Learning.

The Activation function used here will be Bipolar Sigmoidal Function so the range is [-1,1].

X1 and X2 are inputs, b is the bias taken as 1, and the target value is the output of logical AND operation over inputs.

Input	**Input**	**Bias**	**Target**
X1	**X2**	**b**	**y**
1	1	1	1
1	-1	1	-1
-1	1	1	-1
-1	-1	1	-1

Step 1: Initially, the weights are set to zero and the bias is also set to zero.

$w = [\ 0\ 0\ 0\ ]^T$ and $b = 0$

Step 2:

Set input vector $Xi = S_i$ 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 : $w(new) = w(old) + x_1y_1 = [\ 0\ 0\ 0\ ]^T + [\ -1\ -1\ 1\ ]^T\ .\ [\ -1\ ] = [\ 1\ 1\ -1\ ]^T$

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

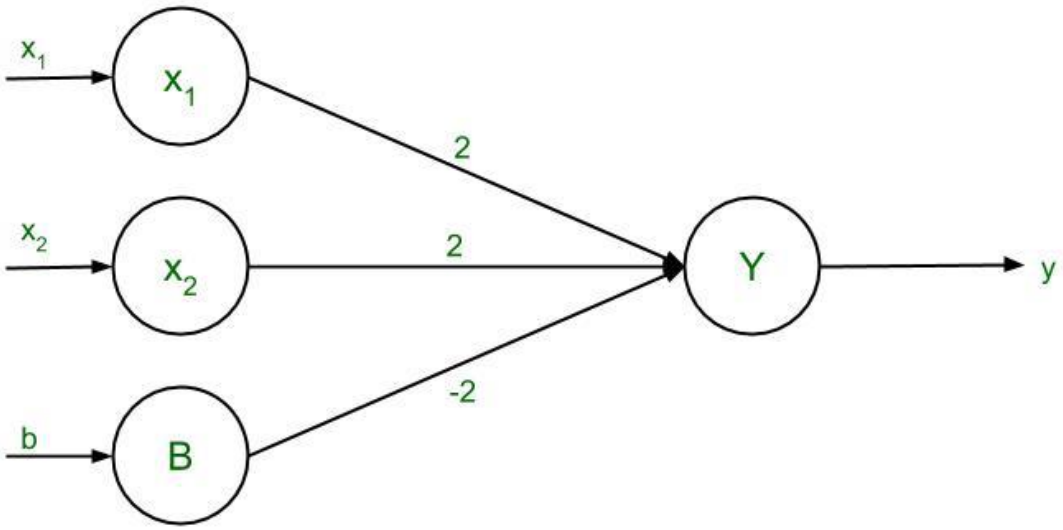
Second iteration : $(new) = [\ 1\ 1\ -1\ ]^T + [\ -1\ 1\ 1\ ]^T\ .\ [\ -1\ ] = [\ 2\ 0\ -2\ ]^T$

Third iteration : $w(new) = [\ 2\ 0\ -2]^T + [\ 1\ -1\ 1\ ]^T\ .\ [\ -1\ ] = [\ 1\ 1\ -3\ ]^T$

Fourth iteration : $w(new) = [\ 1\ 1\ -3]^T + [\ 1\ 1\ 1\ ]^T\ .\ [\ 1\ ] = [\ 2\ 2\ -2\ ]^T$

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

Testing the network :



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 :

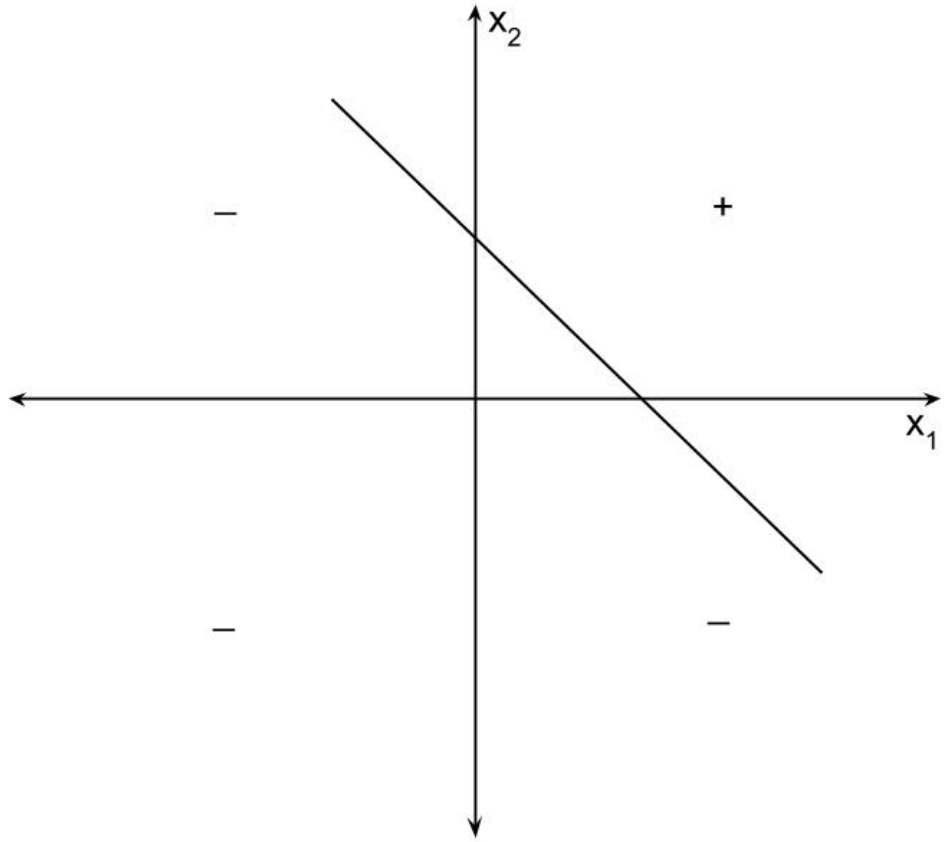
$2x_1 + 2x_2 – 2b = y$

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, $x_2 = -x_1 + 1$



PART B

(PART B: TO BE COMPLETED BY STUDENTS)

(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Black board access available)

Roll. No. A12	Name: Sufiyan Khan
Class: TE – AI & DS	Batch: A1
Date of Experiment:	Date of Submission: 24-03-24
Grade:	

B.1 Input and Output:

```
class HebbianNetwork:
    def __init__(self, input_size, output_size, learning_rate=1):
        self.input_size = input_size
        self.output_size = output_size
        self.learning_rate = learning_rate
        self.weights = [[0.0] * output_size for _ in range(input_size)]

    def train(self, input_data):
        for i in range(self.input_size):
            for j in range(self.output_size):
                self.weights[i][j] += self.learning_rate * input_data[i] * input_data[j]

    def predict(self, input_data):
        output = [0.0] * self.output_size
        for j in range(self.output_size):
            for i in range(self.input_size):
                output[j] += input_data[i] * self.weights[i][j]
        return output

if __name__ == "__main__":
    # Create a Hebbian network with 3 input neurons and 1 output neuron
    network = HebbianNetwork(input_size=3, output_size=3)

    # Train the network with sample data
    training_data = [
        [-1, 0, 1],  # Input pattern 1
        [0.5, -1.5, 2.5],  # Input pattern 2
        [0, 1, 2]   # Input pattern 3
    ]
    for data in training_data:
        network.train(data)

    # Test the network with a new input pattern
    test_input = [1, 0, 1]
    predicted_output = network.predict(test_input)
    print("Predicted output:", predicted_output)
```

```
Predicted output: [1.5, -2.5, 11.5]
```

B.2 Conclusion:

Thus we have successfully implemented Heebian Learning using Python to recognize how to improve the weights of nodes of a network.