Batch: A1 Roll No.: 16010120015

Experiment No. 05

Grade: A / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

Title: To Study and implement HEBBS learning rule

Objective: To write a program to implement HEBBS learning rule for the given data.

Expected Outcome of Experiment:

CO3: Understand perceptron's and counter propagation networks

Books/ Journals/ Websites referred:

- J.S.R.Jang, C.T.Sun and E.Mizutani, "Neuro-Fuzzy and Soft Computing", PHI, 2004, Pearson Education 2004.
- Davis E.Goldberg, "Genetic Algorithms: Search, Optimization and Machine Learning", Addison Wesley, N.Y., 1989.
- S. Rajasekaran and G.A.V.Pai, "Neural Networks, Fuzzy Logic and Genetic Algorithms", PHI, 2003.
- http://library.thinkquest.org/C007395/tqweb/history.html

Pre Lab/ Prior Concepts:

Neural networks, sometimes referred to as connectionist models, are parallel-distributed models that have several distinguishing features-

1) A set of processing units;

2) An activation state for each unit, which is equivalent to the output of the unit;

3) Connections between the units. Generally, each connection is defined by a weight w_{jk} that
determines the effect that the signal of unit *j* has on unit *k;*
4) A propagation rule, which determines the effective input of the unit from its external inputs;
5) An activation function, which determines the new level of activation based on the effective
input and the current activation;
6) An external input (bias, offset) for each unit;
7) A method for information gathering (learning rule);
8) An environment within which the system can operate, provide input signals and, if necessary, error signals.

Hebbian learning rule and algorithm

or Hebbian learning rule comes under Artificial Neural Network (ANN) which is an architecture of a large number of interconnected elements called neurons. These neurons process the input received to give the desired output. The nodes or neurons are linked by inputs(x1,x2,x3…xn), connection weights(w1,w2,w3…wn), and activation functions(a function that defines the output of a node).

In layman's term, a neural network trains itself with known examples and solves various problems that are unknown or difficult to be solved by humans!!

In Hebb network, "When an axon of cell A is near enough to excite cell B and repeatedly or permanently takes place in firing it, some growth process or metabolic changes takes place in one or both the cells such that A's efficiency, as one of the cells firing B, is increased."

In this, if 2 interconnected neurons are ON simultaneously then the weight associated with these neurons can be increased by the modification made in their synaptic gaps(strength). The weight update in the Hebb rule is given by;

*ith value of w(new) = ith value of w(old) + (ith value of x * y)*

Implementation Details:

Hebbian learning rule:

```
import numpy as np
import TableIt

x = [[-1,-1],[-1,1],[1,-1],[1,1]]

learning_rate = float(input("Enter learning rate: "))
bias = float(input("Enter bias: "))

weights = []
for i in range(2):
    weights.append(float(input(f"Enter weight{i+1}: ")))

choice = int(input("Enter 1)AND Gate 2)OR Gate "))
target = []
if choice == 1:
    target = [-1,-1,-1,1]
elif choice == 2:
    target = [-1,1,1,1]

print(f">>Inputs: {x}")
print(f">Initial Weights: {weights}")
print(f">Initial Bias: {bias}")
print(f">Learning Rate: {learning_rate}")
print(f">Target: {target}\n")

n = int(input("Enter Number of Epochs: "))
for t in range(n):
    print(f">>Epoch {t+1}")
    for i in range(4):
        v = np.dot(weights,x[i])
        net = v+bias
        out =np.sign((1.0 - np.exp(-net))/(1.0 + np.exp(-net)))
        for k in range(2):
            weights[k] += learning_rate * out * x[i][k]
```

```
        bias += out*learning_rate
        print(f"\t>Input {i+1}: weights:{weights} bias:{bias}")
print(f"\n>>Final Weight: {weights}, Final Bias: {bias}\n")

output = []

for i in range(4):
    v = np.dot(weights,x[i])
    net = v+bias
    f_net = (1.0 - np.exp(-net)) / (1.0 + np.exp(-net))
    output.append(np.sign(f_net))

print(">>Output:")
table = [
    ["X1","X2","Expected Output","Output"],
]
for i in range(4):
    table.append([x[i][0],x[i][1],target[i],output[i]])
TableIt.printTable(table, useFieldNames=True)

if np.array_equal(output,target):
    print('Follows Expected Gate Configuration')
else:
    print('Doesnt Follow Expected Gate Configuration')
```

Output:

```
>>Inputs: [[-1, -1], [-1, 1], [1, -1], [1, 1]]
>Initial Weights: [2.0, 3.0]
>Initial Bias: 1.0
>Learning Rate: 0.001
>Target: [-1, -1, -1, 1]

>>Epoch 1
      >Input 1: weights:[2.001, 3.001] bias:0.999
      >Input 2: weights:[2.0, 3.002] bias:1.0
      >Input 3: weights:[1.999, 3.0029999999999997] bias:0.999
```

```
      >Input 4: weights:[2.0, 3.0039999999999996] bias:1.0
>>Epoch 2
      >Input 1: weights:[2.001, 3.0049999999999994] bias:0.999
      >Input 2: weights:[2.0, 3.0059999999999993] bias:1.0
      >Input 3: weights:[1.999, 3.0069999999999992] bias:0.999
      >Input 4: weights:[2.0, 3.007999999999999] bias:1.0
>>Epoch 3
      >Input 1: weights:[2.001, 3.008999999999999] bias:0.999
      >Input 2: weights:[2.0, 3.009999999999999] bias:1.0
      >Input 3: weights:[1.999, 3.010999999999999] bias:0.999
      >Input 4: weights:[2.0, 3.0119999999999987] bias:1.0
>>Epoch 4
      >Input 1: weights:[2.001, 3.0129999999999986] bias:0.999
      >Input 2: weights:[2.0, 3.0139999999999985] bias:1.0
      >Input 3: weights:[1.999, 3.0149999999999983] bias:0.999
      >Input 4: weights:[2.0, 3.0159999999999982] bias:1.0
>>Epoch 5
      >Input 1: weights:[2.001, 3.016999999999998] bias:0.999
      >Input 2: weights:[2.0, 3.017999999999998] bias:1.0
      >Input 3: weights:[1.999, 3.018999999999998] bias:0.999
      >Input 4: weights:[2.0, 3.019999999999998] bias:1.0
>>Epoch 6
      >Input 1: weights:[2.001, 3.0209999999999977] bias:0.999
      >Input 2: weights:[2.0, 3.0219999999999976] bias:1.0
      >Input 3: weights:[1.999, 3.0229999999999975] bias:0.999
      >Input 4: weights:[2.0, 3.0239999999999974] bias:1.0
>>Epoch 7
      >Input 1: weights:[2.001, 3.0249999999999972] bias:0.999
      >Input 2: weights:[2.0, 3.025999999999997] bias:1.0
      >Input 3: weights:[1.999, 3.026999999999997] bias:0.999
      >Input 4: weights:[2.0, 3.027999999999997] bias:1.0
>>Epoch 8
      >Input 1: weights:[2.001, 3.028999999999997] bias:0.999
      >Input 2: weights:[2.0, 3.0299999999999967] bias:1.0
      >Input 3: weights:[1.999, 3.0309999999999966] bias:0.999
      >Input 4: weights:[2.0, 3.0319999999999965] bias:1.0
>>Epoch 9
      >Input 1: weights:[2.001, 3.0329999999999964] bias:0.999
      >Input 2: weights:[2.0, 3.0339999999999963] bias:1.0
      >Input 3: weights:[1.999, 3.034999999999996] bias:0.999
      >Input 4: weights:[2.0, 3.035999999999996] bias:1.0
>>Epoch 10
```

```
      >Input 1: weights:[2.001, 3.0369999999999996] bias:0.999
      >Input 2: weights:[2.0, 3.037999999999996] bias:1.0
      >Input 3: weights:[1.999, 3.0389999999999957] bias:0.999
      >Input 4: weights:[2.0, 3.0399999999999956] bias:1.0

>>Final Weight: [2.0, 3.0399999999999956], Final Bias: 1.0

>>Output:
+-----------------------------------------------------------------------+
| X1              | X2              | Expected Output | Output          |
+-----------------+-----------------+-----------------+-----------------+
| -1              | -1              | -1              | -1.0            |
| -1              | 1               | -1              | 1.0             |
| 1               | -1              | -1              | -1.0            |
| 1               | 1               | 1               | 1.0             |
+-----------------------------------------------------------------------+
Doesnt Follow Expected Gate Configuration
```

```
>>Inputs: [[-1, -1], [-1, 1], [1, -1], [1, 1]]
>Initial Weights: [2.0, 3.0]
>Initial Bias: 1.0
>Learning Rate: 0.001
>Target: [-1, -1, -1, 1]

>>Epoch 1
    >Input 1: weights:[2.001, 3.001] bias:0.999
    >Input 2: weights:[2.0, 3.002] bias:1.0
    >Input 3: weights:[1.999, 3.0029999999999997] bias:0.999
    >Input 4: weights:[2.0, 3.0039999999999996] bias:1.0
>>Epoch 2
    >Input 1: weights:[2.001, 3.0049999999999994] bias:0.999
    >Input 2: weights:[2.0, 3.0059999999999993] bias:1.0
    >Input 3: weights:[1.999, 3.0069999999999992] bias:0.999
    >Input 4: weights:[2.0, 3.007999999999999] bias:1.0
>>Epoch 3
    >Input 1: weights:[2.001, 3.008999999999999] bias:0.999
    >Input 2: weights:[2.0, 3.009999999999999] bias:1.0
    >Input 3: weights:[1.999, 3.010999999999999] bias:0.999
    >Input 4: weights:[2.0, 3.0119999999999987] bias:1.0
>>Epoch 4
    >Input 1: weights:[2.001, 3.0129999999999986] bias:0.999
    >Input 2: weights:[2.0, 3.0139999999999985] bias:1.0
    >Input 3: weights:[1.999, 3.0149999999999983] bias:0.999
    >Input 4: weights:[2.0, 3.0159999999999982] bias:1.0
>>Epoch 5
    >Input 1: weights:[2.001, 3.016999999999998] bias:0.999
    >Input 2: weights:[2.0, 3.017999999999998] bias:1.0
    >Input 3: weights:[1.999, 3.018999999999998] bias:0.999
    >Input 4: weights:[2.0, 3.019999999999998] bias:1.0
>>Epoch 6
    >Input 1: weights:[2.001, 3.0209999999999977] bias:0.999
    >Input 2: weights:[2.0, 3.0219999999999976] bias:1.0
    >Input 3: weights:[1.999, 3.0229999999999975] bias:0.999
    >Input 4: weights:[2.0, 3.0239999999999974] bias:1.0
>>Epoch 7
    >Input 1: weights:[2.001, 3.0249999999999972] bias:0.999
    >Input 2: weights:[2.0, 3.025999999999997] bias:1.0
    >Input 3: weights:[1.999, 3.026999999999997] bias:0.999
    >Input 4: weights:[2.0, 3.027999999999997] bias:1.0
>>Epoch 8
    >Input 1: weights:[2.001, 3.028999999999997] bias:0.999
    >Input 2: weights:[2.0, 3.0299999999999967] bias:1.0
    >Input 3: weights:[1.999, 3.0309999999999966] bias:0.999
    >Input 4: weights:[2.0, 3.0319999999999965] bias:1.0
>>Epoch 9
    >Input 1: weights:[2.001, 3.0329999999999964] bias:0.999
    >Input 2: weights:[2.0, 3.0339999999999963] bias:1.0
    >Input 3: weights:[1.999, 3.034999999999996] bias:0.999
    >Input 4: weights:[2.0, 3.035999999999996] bias:1.0
>>Epoch 10
    >Input 1: weights:[2.001, 3.036999999999996] bias:0.999
    >Input 2: weights:[2.0, 3.037999999999996] bias:1.0
    >Input 3: weights:[1.999, 3.0389999999999957] bias:0.999
    >Input 4: weights:[2.0, 3.0399999999999956] bias:1.0

>>Final Weight: [2.0, 3.0399999999999956], Final Bias: 1.0

>>Output:
+-----------------------------------------------------------------------+
| X1              | X2              | Expected Output | Output          |
+-----------------+-----------------+-----------------+-----------------+
| -1              | -1              | -1              | -1.0            |
| -1              | 1               | -1              | 1.0             |
| 1               | -1              | -1              | -1.0            |
| 1               | 1               | 1               | 1.0             |
+-----------------------------------------------------------------------+
Doesnt Follow Expected Gate Configuration
```

Conclusion: Thus, we have successfully implemented Hebbian learning algorithm of Neural Network.

Post Lab Descriptive Questions :

1. Compare the Hebbian learning and competitive learning. .

Coding	Not implemented by LTP itself: an "emergent property" of circuits	Depends on encoding of temporal intervals, stimulus properties, and stimulus relationships
Necessary CS-US relation	Close temporal contiguity	Contingency
Form of learned output	Recapitulative: When stimulus recurs, output recurs	Anticipatory: Learned behavior usually differs from behavior during learning
Critical ISI	1–100 ms	None: Rate of conditioning is inversely proportional to ISI/ITI
Effect of ITI	The longer the ITI, the weaker the LTP	The longer the ITI relative to ISI, the faster and stronger the learning
Induction kinetics	Expression requires tens of seconds to minutes	Behavioral expression is immediate, <1 s after induction
Acquisition function	Requires repetition	Often complete within single trial

2. Find the weights after one iteration for hebbian learning of a single neuron network. Start with initial weights W=[1,-1] and Inputs as X1=[1,-2] X2=[2,3], x3[1,-1] and C=1.
   i. Use bipolar binary activation function
   ii. Use continuous activation function

Given: Initial weight $W = [1, -1]$

Inputs $X_1 = [1, -2]$; $X_2 = [2, 3]$; $X_3 = [1, -1]$

$C = 1$.

To find: weights after 2 Epochs. by using bipolar activation funcn.

Formulae: $net_i = w_i^t x_i$

$O_i = f(net_i) = sgn(net_i)$

$\Delta w_i = C O_i X$

Solution:

* Epoch 1:

Step 1 $X = X_1$

$$net_1 = w_1^t x_1 = [1 \ -1] \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

$$= 1 + 2$$

$$= 3$$

$$O_1 = f(net_1) = sgn(net_1) = sgn(3)$$

$$= 1$$

$$\Delta w_1 = C \times O_1 \times x_1$$

$$= 1 \times 1 \times \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

$$= \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

$$W_2 = W_1 + \Delta W_1 = \begin{bmatrix}1\\-1\end{bmatrix} + \begin{bmatrix}1\\-2\end{bmatrix} = \begin{bmatrix}2\\-3\end{bmatrix}$$

Step 2 : $X = X_2$.

$$net_2 = W_2^t X_2 = [2\ -3]\begin{bmatrix}2\\3\end{bmatrix} = 4 - 9 = -5$$

$$O_2 = f(net_2) = sgn(-5) = -1$$

$$\Delta W_2 = C \times O_2 \times X_2 = 1 \times -1 \times \begin{bmatrix}2\\3\end{bmatrix} = \begin{bmatrix}-2\\-3\end{bmatrix}$$

$$W_3 = W_2 + \Delta W_2 = \begin{bmatrix}2\\-3\end{bmatrix} + \begin{bmatrix}-2\\-3\end{bmatrix} = \begin{bmatrix}0\\-6\end{bmatrix}$$

Step 3 : $X = X_3$

$$net_3 = W_3^t X_3 = [0\ -6]\begin{bmatrix}1\\-1\end{bmatrix} = 6$$

$$O_3 = f(net_3) = sgn(6) = 1$$

$$\Delta W_3 = C \times O_3 \times X_3 = 1 \times 1 \times \begin{bmatrix}1\\-1\end{bmatrix} = \begin{bmatrix}1\\-1\end{bmatrix}$$

$$W_4 = W_3 + \Delta W_3 = \begin{bmatrix}0\\-6\end{bmatrix} + \begin{bmatrix}1\\-1\end{bmatrix} = \begin{bmatrix}1\\-7\end{bmatrix}$$

* Epoch 2

Step 1 : $X = X_1$

$$net_4 = W_4^t X_4 = [1\ -7]\begin{bmatrix}1\\-2\end{bmatrix} = 15$$

$$O_4 = f(net_4) = sgn(15) = 1$$

$$\Delta W_4 = C \times O_4 \times X_1 = 1 \times 1 \times \begin{bmatrix}1\\-2\end{bmatrix} = \begin{bmatrix}1\\-2\end{bmatrix}$$

$$W_5 = W_4 + \Delta W_4 = \begin{bmatrix}1\\-7\end{bmatrix} + \begin{bmatrix}1\\-2\end{bmatrix} = \begin{bmatrix}2\\-9\end{bmatrix}$$

Step 2 : $X = X_2$

$$net_5 = W_3^t X = [2\ -9]\begin{bmatrix}2\\3\end{bmatrix} = -23$$

$$O_5 = f(net_5) = sgn(-23) = -1$$

$$\Delta W_5 = C \times O_3 \times X = 1 \times -1 \times \begin{bmatrix}2\\3\end{bmatrix} = \begin{bmatrix}-2\\-3\end{bmatrix}$$

$$W_6 = W_5 + \Delta W_5 = \begin{bmatrix}2\\-9\end{bmatrix} + \begin{bmatrix}-2\\-3\end{bmatrix} = \begin{bmatrix}0\\-12\end{bmatrix}$$

Step 3 : $X = X_3$

$net_6 = w_6^t X$

$= [0 \quad -12] \begin{bmatrix} 1 \\ -1 \end{bmatrix}$

$= 12$

$O_6 = f(net_6) = sgn(12) = 1$

$\Delta w_6 = c \cdot O_6 \cdot X$

$= (1)(1) \begin{bmatrix} 1 \\ -1 \end{bmatrix}$

$= \begin{bmatrix} 1 \\ -1 \end{bmatrix}$

$w_7 = w_6 + \Delta w_6$

$= \begin{bmatrix} 0 \\ -12 \end{bmatrix} + \begin{bmatrix} 1 \\ -1 \end{bmatrix}$

$= \begin{bmatrix} 1 \\ -13 \end{bmatrix}$

Thus after 2 epochs the final weights are $\begin{bmatrix} 1 \\ -13 \end{bmatrix}$

Date: 15/10/2022 **Signature of faculty in-charge**