EXPERIMENT NO. 8

AIM: Implement Bidirectional Associative Memory (BAM) Algorithm for the given dataset.

SOFTWARE: Python

LABORATORY OUTCOMES:

- Students will be able to implement Bidirectional Associative Memory Network algorithm for the given dataset.
- Students will be able to compute the Weight Matrix using the BAM Algorithm and Test the BAM model for the input patterns.

THEORY:

The architecture of BAM network consists of two layers of neurons which are connected by directed weighted pare interconnections. The network dynamics involve two layers of interaction. The BAM network iterates by sending the signals back and forth between the two layers until all the neurons reach equilibrium. The weights associated with the network are bidirectional. Thus, BAM can respond to the inputs in either layer.

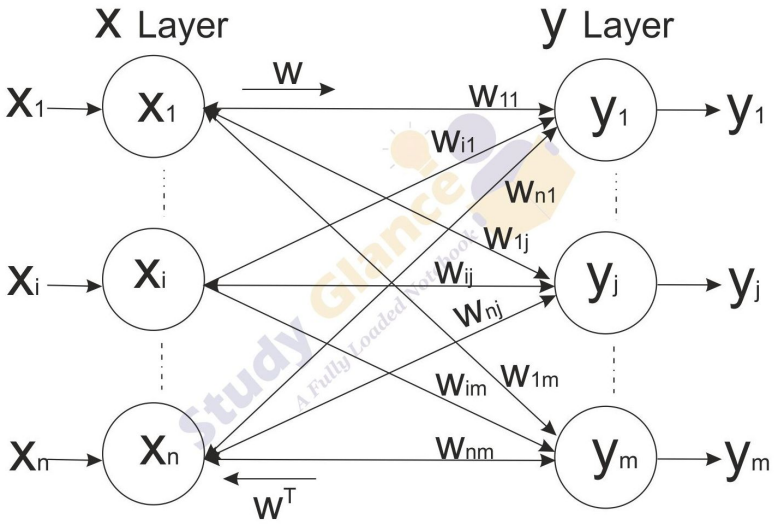

Figure shows a BAM network consisting of n units in X layer and m units in Y layer. The layers can be connected in both directions(bidirectional) with the result the weight matrix sent from the X layer to the Y layer is W and the weight matrix for signals sent from the Y layer to the X layer is W^T. Thus, the Weight matrix is calculated in both directions.

Determination of Weights

Let the input vectors be denoted by s(p) and target vectors by t(p). p = 1, ... , P. Then the weight matrix to store a set of input and target vectors, where

$$s(p) = (s_1(p), .. , s_i(p), ... , s_n(p))$$
$$t(p) = (t_1(p), .. , t_j(p), ... , t_m(p))$$

can be determined by Hebb rule training a1gorithm. In case of input vectors being binary, the weight matrix $W = \{w_{ij}\}$ is given by

$$w_{ij} = \sum_{p=1}^{P} [2S_i(p) - 1][2t_j(p)-1]$$

When the input vectors are bipolar, the weight matrix $W = \{w_{ij}\}$ can be defined as,

$$w_{ij} = \sum_{p=1}^{P} [S_i(p)][t_j(p)]$$

The activation function is based on whether the input target vector pairs used are binary or bipolar.

Testing Algorithm for Discrete Bidirectional Associative Memory:

Step 0: Initialize the weights to store p vectors. Also initialize all the activations to zero.

Step 1: Perform Steps 2-6 for each testing input.

Step 2: Set the activations of X layer to current input pattern, i.e., presenting the input pattern x to X layer and similarly presenting the input pattern y to Y layer. Even though, it is bidirectional memory, at one time step, signals can be sent from only one layer. So, either of the input patterns may be the zero vector

Step 3: Perform Steps 4-6 when the activations are not converged.

Step 4: Update the activations of units in Y layer. Calculate the net input,

$$y_{inj}=\sum_{i=1}^{n} x_i w_{ij}$$

Applying activation function we obtain

$$y_j = f(y_{inj})$$

Send this signal to the X layer.

Step 5: Update the activations of units in X layer. Calculate the net input,

$$x_{ini}=\sum_{j=1}^{m} y_j w_{ij}$$

Applying activation function we obtain

$$x_i = f(x_{ini})$$

Send this signal to the Y layer.

Step 6: Test for convergence of the net. The convergence occurs if the activation vectors x and y reach equilibrium. If this occurs then stop, Otherwise, continue.

PROGRAM:

```
# Import Python Libraries
import numpy as np

# Take two sets of patterns:
# Set A: Input Pattern
x1 = np.array([1, 1, 1, 1, 1, 1]).reshape(6, 1)
x2 = np.array([-1, -1, -1, -1, -1, -1]).reshape(6, 1)
x3 = np.array([1, 1, -1, -1, 1, 1]).reshape(6, 1)
x4 = np.array([-1, -1, 1, 1, -1, -1]).reshape(6, 1)

# Set B: Target Pattern
y1 = np.array([1, 1, 1]).reshape(3, 1)
y2 = np.array([-1, -1, -1]).reshape(3, 1)
y3 = np.array([1, -1, 1]).reshape(3, 1)
y4 = np.array([-1, 1, -1]).reshape(3, 1)
```

```
'''
print("Set A: Input Pattern, Set B: Target Pattern")
print("\nThe input for pattern 1 is")
print(x1)
print("\nThe target for pattern 1 is")
print(y1)
print("\nThe input for pattern 2 is")
print(x2)
print("\nThe target for pattern 2 is")
print(y2)
print("\nThe input for pattern 3 is")
print(x3)
print("\nThe target for pattern 3 is")
print(y3)
print("\nThe input for pattern 4 is")
print(x4)
print("\nThe target for pattern 4 is")
print(y4)

print("\n------------------------------")
'''
# Calculate weight Matrix: W
inputSet = np.concatenate((x1, x2, x3, x4), axis = 1)
targetSet = np.concatenate((y1.T, y2.T, y3.T, y4.T), axis = 0)
print("\nWeight matrix:")
weight = np.dot(inputSet, targetSet)
print(weight)

print("\n------------------------------")

# Testing Phase
# Test for Input Patterns: Set A
print("\nTesting for input patterns: Set A")
def testInputs(x, weight):
  # Multiply the input pattern with the weight matrix
  # (weight.T X x)
  y = np.dot(weight.T, x)
  y[y < 0] = -1
  y[y >= 0] = 1
  return np.array(y)

print("\nOutput of input pattern 1")
print(testInputs(x1, weight))
print("\nOutput of input pattern 2")
print(testInputs(x2, weight))
print("\nOutput of input pattern 3")
```

```
print(testInputs(x3, weight))
print("\nOutput of input pattern 4")
print(testInputs(x4, weight))

# Test for Target Patterns: Set B
print("\nTesting for target patterns: Set B")
def testTargets(y, weight):
  # Multiply the target pattern with the weight matrix
  # (weight X y)
  x = np.dot(weight, y)
  x[x <= 0] = -1
  x[x > 0] = 1
  return np.array(x)

print("\nOutput of target pattern 1")
print(testTargets(y1, weight))
print("\nOutput of target pattern 2")
print(testTargets(y2, weight))
print("\nOutput of target pattern 3")
print(testTargets(y3, weight))
print("\nOutput of target pattern 4")
print(testTargets(y4, weight))
```

OUTPUTS:

```
Weight matrix:
[[4 0 4]
 [4 0 4]
 [0 4 0]
 [0 4 0]
 [4 0 4]
 [4 0 4]]

------------------------------

Testing for input patterns: Set A

Output of input pattern 1
[[1]
 [1]
 [1]]

Output of input pattern 2
[[-1]
 [-1]
 [-1]]

Output of input pattern 3
[[ 1]
 [-1]
```

```
 [ 1]]

Output of input pattern 4
[[-1]
 [ 1]
 [-1]]

Testing for target patterns: Set B

Output of target pattern 1
[[1]
 [1]
 [1]
 [1]
 [1]
 [1]]

Output of target pattern 2
[[-1]
 [-1]
 [-1]
 [-1]
 [-1]
 [-1]]

Output of target pattern 3
[[ 1]
 [ 1]
 [-1]
 [-1]
 [ 1]
 [ 1]]

Output of target pattern 4
[[-1]
 [-1]
 [ 1]
 [ 1]
 [-1]
 [-1]]
```

CONCLUSION:

Bidirectional Associative Memories (BAM) is system that allow to associate pairs of patterns. Once a memory has learned, patterns can be recalled in two directions. BAMs have many applications in pattern recognition and image processing. BAM compute the weight matrix and Test the BAM model for the input patterns.

TEXT/REFERENCE BOOKS:

- "Neural Network a Comprehensive Foundation" By Simon Haykin
- "Introduction to Soft Computing" By Dr. S. N. Shivnandam, Mrs. S. N. Deepa

- "Neural Network: A classroom Approach" By Satish Kumar
- "Neural Network, Fuzzy Logic and Genetic Algorithms" By Rajshekharan S, Vijayalakshmi Pai
- "Neural Network Design" by Hagan Demuth, Beale"Neural Network for Pattern Recognition", Christopher M. Bishop

WEB ADDRESS (URLS):

- https://en.wikipedia.org/wiki/Bidirectional_associative_memory
- https://www.geeksforgeeks.org/bidirectional-associative-memory-bam-implementation-from-scratch/
- https://studyglance.in/nn/display.php?tno=8&topic=Bidirectional-Associative-Memory

----------x----------x----------