Aim: Write a python program for Bidirectional Associative Memory with two pairs of vectors.

Software for Python: Jupyter Notebook

Theory:

Bidirectional Associative Memory (BAM) is a supervised learning model in Artificial Neural Network. This is hetero-associative memory, for an input pattern, it returns another pattern which is potentially of a different size. This phenomenon is very similar to the human brain. Human memory is necessarily associative. It uses a chain of mental associations to recover a lost memory like associations of faces with names, in exam questions with answers, etc.

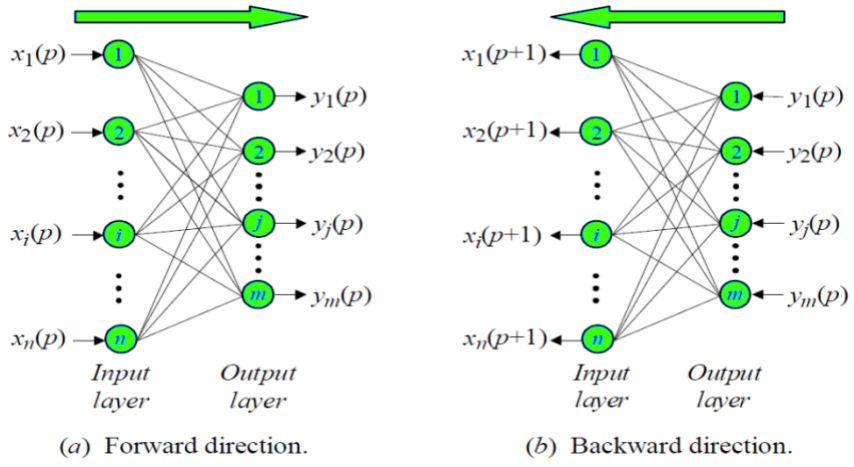
In such memory associations for one type of object with another, a Recurrent Neural Network (RNN) is needed to receive a pattern of one set of neurons as an input and generate a related, but different, output pattern of another set of neurons.

Why BAM is required?

The main objective to introduce such a network model is to store hetero-associative pattern pairs. This is used to retrieve a pattern given a noisy or incomplete pattern.

BAM Architecture:

When BAM accepts an input of n-dimensional vector X from set A then the model recalls m-dimensional vector Y from set B. Similarly, when Y is treated as input, the BAM recalls X.



(a) Forward direction. (b) Backward direction.

Fig.1.BAM Architecture

Algorithm:

1.Storage (Learning): In this learning step of BAM, weight matrix is calculated between M pairs of patterns (fundamental memories) are stored in the synaptic weights of the network following the equation.

$$W = \sum_{m=1}^{M} X_m Y_m^T$$

2.Testing: We have to check that the BAM recalls perfectly Ym for corresponding Xm and recalls Xm for corresponding Ym. Using,

$$Y_m = \text{sign}\left(W^T X_m\right), \quad m = 1.2, \ldots, M$$

$$X_m = \text{sign}\left(W Y_m\right), \quad m = 1.2, \ldots, M$$

All pairs should be recalled accordingly.

3.Retrieval: For an unknown vector X (a corrupted or incomplete version of a pattern from set A or B) to the BAM and retrieve a previously stored association:

$$X \neq X_m, \quad m = \mathbf{1}, \mathbf{2}, \ldots, M$$

Initialize the BAM:
$X(0) = X, \quad p = 0$
Calculate the BAM output at iteration p
$Y(p) = \text{sign}\left[W^T X(p)\right]$
Update the input vector X(p):
$X(p+1) = \text{sign}[WY(p)]$
Repeat the iteration until convergence, when input and output remain unchanged.

Procedure:
This code implements a Bidirectional Associative Memory (BAM) neural network.

First, it defines two input/output pairs, input1/output1 and input2/output2. It then defines a weight matrix W, which is the sum of the outer products of the two input/output pairs. The weight matrix W is used to associate the input vectors with the output vectors.

The code then defines two activation functions: f and g. The function f returns 1 if the input is greater than 0, and -1 otherwise. The function g returns 1 if the input is greater than or equal to 0, and -1 otherwise. These functions are used to threshold the output of the network.

The BAM function takes an input vector as input and uses the weight matrix W to compute an output vector. The output vector is then thresholded using the g function to produce a new input vector. This new input vector is then thresholded using the f function to produce a new output vector. This process is repeated until the input/output pairs converge to a stable state.

Finally, the code tests the BAM function by applying it to an input vector and printing the input vector, the output vector, and the new input vector produced by the BAM function.

Conclusion: Thus, we have studied Bidirectional Associative Memory (BAM) with two pairs of vectors using Python Programming.

Program Code and Output

In [1]:

```
import numpy as np

# Define input/output vectors
input1 = np.array([1, 0, 1, 0])
output1 = np.array([0, 1])

input2 = np.array([1, 1, 0, 0])
output2 = np.array([1, 0])

# Define weight matrix
W = np.outer(output1, input1) + np.outer(output2, input2)

# Define activation functions
def f(x):
    return 1 if x > 0 else -1

def g(x):
    return 1 if x >= 0 else -1

# Define bidirectional associative memory function
def BAM(input_vec):
    y = np.dot(W, input_vec)
    output_vec = np.array([g(x) for x in y])
    x = np.dot(W.T, output_vec)
    input_vec = np.array([f(x) for x in x])
    return input_vec, output_vec

# Test the BAM function
input_vec = np.array([1, 0, 0, 1])
input_vec_new, output_vec = BAM(input_vec)

print("Input vector: ", input_vec)
print("Output vector: ", output_vec)
print("New input vector: ", input_vec_new)
```

```
Input vector:  [1 0 0 1]
Output vector:  [1 1]
New input vector:  [ 1  1  1 -1]
```