Name :- Shivam Indrabhan Borse

Roll No :- 19

Subject: Artificial Neural Network (SL - II)

Class : TE

Branch: AI & DS

Practical – 4

Problem statement : : Write a python Program for Bidirectional Associative Memory with two pairs of vectors.

Code :

```
import numpy as np
import random
import math
```

✓ 0.1s

```
X1 = [1, 1, 1, 1, 1, 1]
X2 = [-1, -1, -1, -1, -1, -1]
X3 = [1, -1, -1, 1, 1, 1]
X4 = [1, 1, -1, -1, -1, -1]

X = np.array([X1, X2, X3, X4])

Y1 = [1, 1, 1]
Y2 = [-1, -1, -1]
Y3 = [-1, 1, 1]
Y4 = [1, -1, 1]

Y = np.array([Y1, Y2, Y3, Y4])

print("X = ", X)
print("\nY = ", Y)
print("\n\nDimensions of X: ", X.shape)
print("\nDimensions of Y: ", Y.shape)
```

✓ 0.0s

```
X =  [[ 1  1  1  1  1  1]
 [-1 -1 -1 -1 -1 -1]
 [ 1 -1 -1  1  1  1]
 [ 1  1 -1 -1 -1 -1]]

Y =  [[ 1  1  1]
 [-1 -1 -1]
 [-1  1  1]
 [ 1 -1  1]]


Dimensions of X:  (4, 6)

Dimensions of Y:  (4, 3)
```

```python
def calcWeight(X, Y):
    return np.dot(X.T, Y)

weight = calcWeight(X, Y)
print('W = ', weight, end = "")

print("\n\nDimensions of Weight Matrix: ",weight.shape)
```
✓ 0.0s

```
W =  [[2 2 4]
 [4 0 2]
 [2 2 0]
 [0 4 2]
 [0 4 2]
 [0 4 2]]

Dimensions of Weight Matrix:  (6, 3)
```

```python
def ForwardBipolarActivation(matrix, weight):
    matrix[matrix > 0] = 1
    matrix[matrix <= 0] = -1
    return np.array(matrix)

def BackwardBipolarActivation(matrix, weight):
    matrix[matrix >= 0] = 1
    matrix[matrix < 0] = -1
    return np.array(matrix)
```
✓ 0.0s

Forward Testing

```python
def forward(Y, weight):
    x = np.dot(Y, weight.T)
    return ForwardBipolarActivation(x, weight)
```
✓ 0.0s

```python
print("\nweight * Y1 = ", forward(Y1, weight), " = X1")
print("\nweight * Y2 = ", forward(Y2, weight), " = X2")
print("\nweight * Y3 = ", forward(Y3, weight), " = X3")
print("\nweight * Y4 = ", forward(Y4, weight), " = X4")

print("\n\nIt is observed that the obtained results match with the original X matrices.\n\nThus forward testing is 100% accurate.")
```
✓ 0.0s

```
weight * Y1 =  [1 1 1 1 1 1]  = X1

weight * Y2 =  [-1 -1 -1 -1 -1 -1]  = X2

weight * Y3 =  [ 1 -1 -1  1  1  1]  = X3

weight * Y4 =  [ 1  1 -1 -1 -1 -1]  = X4


It is observed that the obtained results match with the original X matrices.

Thus forward testing is 100% accurate.
```

backward Testing

```
def backward(X, weight):
    Y = np.dot(weight.T, X)
    return BackwardBipolarActivation(Y, weight)
```

[14] ✓ 0.0s

```
print("\nweight * X1 = ", backward(X1, weight), " = Y1")
print("\nweight * X2 = ", backward(X2, weight), " = Y2")
print("\nweight * X3 = ", backward(X3, weight), " = Y3")
print("\nweight * X4 = ", backward(X4, weight), " = Y4")

print("\n\nIt is observed that the obtained results match with the original Y (target) matrices.\n\nThus backward testing is 100% accurate.")
```

[15] ✓ 0.0s

```
weight * X1 =  [1 1 1]  = Y1

weight * X2 =  [-1 -1 -1]  = Y2

weight * X3 =  [-1  1  1]  = Y3

weight * X4 =  [ 1 -1  1]  = Y4


It is observed that the obtained results match with the original Y (target) matrices.

Thus backward testing is 100% accurate.
```