Vivekanand Education Society's
Institute of Technology
Approved by AICTE & Affiliated to University of Mumbai
Artificial Intelligence and Data Science Department

Subject/Odd Sem 2023-23/Experiment 1

Name : Shreya Singh	Class/Roll No. : D16AD/55	Grade :

Title of Experiment : Multilayer Perceptron Algorithm.

Objective of Experiment : Implementation of basic neural network models

Outcome of Experiment : Enabling the model to learn the non-linear relationship between inputs and outputs, providing correct XOR gate behavior

Problem Statement : Multilayer Perceptron algorithm to Simulate XOR Gate.

Theory : The XOR gate is a logical gate that takes two binary inputs (0 or 1) and outputs 1 if exactly one of the inputs is 1, and 0 otherwise. Here is the truth table for the XOR gate:

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

$$XOR(\boldsymbol{x_1}, \boldsymbol{x_2}) = AND(NOT(AND(\boldsymbol{x_1}, \boldsymbol{x_2})), OR(\boldsymbol{x_1}, \boldsymbol{x_2}))$$

Subject/Odd Sem 2023-23/Experiment 1

The XOR gate is interesting because it is not linearly separable, meaning we cannot draw a straight line to separate the inputs that yield output 0 from those that yield output 1. This nonlinearity makes it a good example to demonstrate the capabilities of a Multilayer Perceptron.

A Multilayer Perceptron is a type of artificial neural network that consists of multiple layers of interconnected nodes (neurons). The first layer is the input layer, the last layer is the output layer, and any layers in between are called hidden layers. Each connection between nodes has an associated weight, and each node has an activation function that determines its output based on the weighted sum of its inputs.

Designing the Perceptron Network:

Step1: Now for the corresponding weight vector w:(w1 ,w2) of the input vector x:(x1 ,x2) to the AND and OR node,the associated Perceptron Function can be defined as:

For the AND node:

$$\hat{y}_1 = \Theta\left(w_1 x_1 + w_2 x_2 + b_{\text{AND}}\right)$$

For the OR node:

$$\hat{y}_2 = \Theta\left(w_1 x_1 + w_2 x_2 + b_{\text{OR}}\right)$$

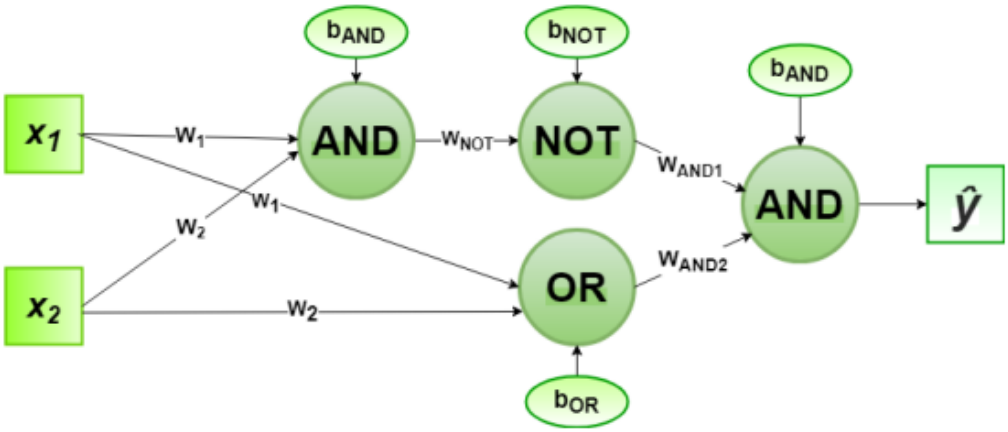
Step2:

$$\hat{y}_3 = \Theta\left(w_{\text{NOT}} \hat{y}_1 + b_{\text{NOT}}\right)$$

Step3: The output ($\hat{y}_2$) from the OR node and the output ($\hat{y}_3$) from the NOT node as mentioned in Step2 will be inputted to the AND node with weight (w_{AND1},w_{AND2}). Then the corresponding output ($\hat{y}$) is the final output of the XOR logic function. The associated Perceptron Function can be defined as:

$$\hat{y} = \Theta\left(w_{AND1}\hat{y}_3 + w_{AND2}\hat{y}_2 + b_{AND}\right)$$

Flowchart



Subject/Odd Sem 2023-23/Experiment 1

Program :

```
[1]  # importing Python library
     import numpy as np
```

```
[2]  # define Unit Step Function
     def Activation(v):
       if v >= 0:
         return 1
       else:
         return 0
```

```
[3]  # design Perceptron Model
     def perceptronModel(x, w, b):
       v = np.dot(w, x) + b
       y = Activation(v)
       return y
```

Subject/Odd Sem 2023-23/Experiment 1

0s

```
# NOT Logic Function
# wNOT = -1, bNOT = 0.5
def NOT_logicFunction(x):
  wNOT = -1
  bNOT = 0.5
  return perceptronModel(x, wNOT, bNOT)

# AND Logic Function
# here w1 = wAND1 = 1,
# w2 = wAND2 = 1, bAND = -1.5
def AND_logicFunction(x):
  w = np.array([1, 1])
  bAND = -1.5
  return perceptronModel(x, w, bAND)

# OR Logic Function
# w1 = 1, w2 = 1, bOR = -0.5
def OR_logicFunction(x):
  w = np.array([1, 1])
  bOR = -0.5
  return perceptronModel(x, w, bOR)
```

Subject/Odd Sem 2023-23/Experiment 1

```
[5]  # XOR Logic Function
     # with AND, OR and NOT
     # function calls in sequence
     def XOR_logicFunction(x):
       y1 = AND_logicFunction(x)
       y2 = OR_logicFunction(x)
       y3 = NOT_logicFunction(y1)
       final_x = np.array([y2, y3])
       finalOutput = AND_logicFunction(final_x)
       return finalOutput
```

```
[6]  # testing the Perceptron Model
     test1 = np.array([0, 1])
     test2 = np.array([1, 1])
     test3 = np.array([0, 0])
     test4 = np.array([1, 0])

     print("XOR({}, {}) = {}".format(0, 1, XOR_logicFunction(test1)))
     print("XOR({}, {}) = {}".format(1, 1, XOR_logicFunction(test2)))
     print("XOR({}, {}) = {}".format(0, 0, XOR_logicFunction(test3)))
     print("XOR({}, {}) = {}".format(1, 0, XOR_logicFunction(test4)))
```

```
XOR(0, 1) = 1
XOR(1, 1) = 0
XOR(0, 0) = 0
XOR(1, 0) = 1
```

Results and Discussions :

We have implemented a basic neural network model , Multilayer Perceptron and have understood the working of the Multilayer Perceptron Algorithm by stimulating XOR Gate.