Name: Subrato Tapaswi	Class/Roll No.: D16AD/60	Grade:

Title of Experiment: Multilayer Perceptron algorithm to Simulate XOR gate.

Objective of Experiment:
The objective of the experiment is to demonstrate a non-linearly separable XOR gate on a multi-layer perceptron network.

Description / Theory:
Multilayer Perceptron-

- A Multilayer Perceptron (MLP) is a type of artificial neural network used in many machine learning tasks like recognizing patterns and making predictions. It's a fundamental concept in deep learning and forms the basis for more advanced neural networks.
- An MLP has three main parts: an input layer, hidden layers, and an output layer. Each part does some math on the data it gets and passes the result to the next part.
- To make an MLP learn, we adjust its internal settings called weights. We do this to make it good at its task. We usually use a method called Gradient Descent for this.
- MLPs are powerful and can learn almost anything if we give them enough resources, like more neurons and layers. But making them learn effectively depends on how we design them and teach them.
- There's a problem called overfitting, where an MLP can be too good at the training data but not good at new, unseen data. To avoid this, we use tricks like regularization and stopping early.
- Before we train an MLP, we have to set some settings like how many layers it has, how many neurons in each layer, and how fast it learns. Figuring out the best settings often involves some trial and error.

- While MLPs have their limits with complex data, they're like the building blocks for more advanced neural networks like CNNs for images and RNNs for sequences.

XOR-

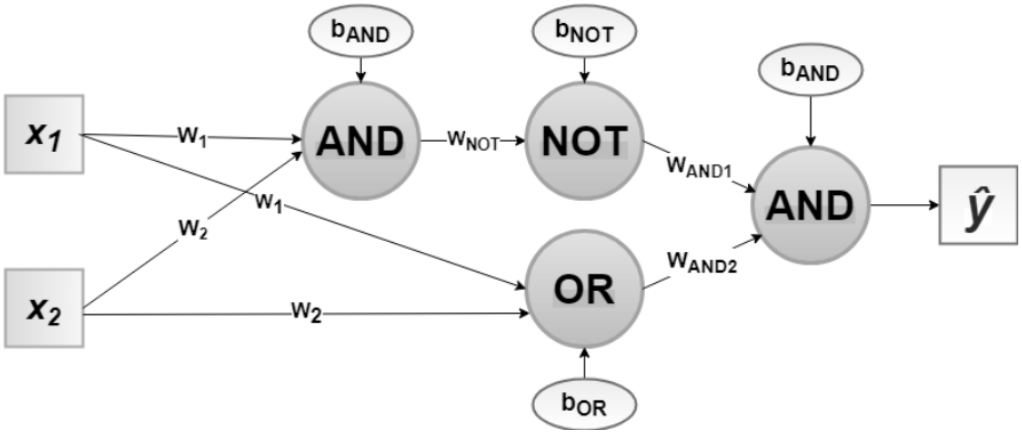
X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

XOR is not linearly separable. But Uni layered perceptrons can only work with linearly separable data. To solve this problem, we add an extra layer to our vanilla perceptron, i.e., we create a Multi Layered Perceptron (or MLP). We call this extra layer the Hidden layer.

XOR can be written as: XOR(x1,x2) = AND(NOT(AND(x1,x2)),OR(x1,x2))

Algorithm/ Pseudo Code / Flowchart

Step1: Now for the corresponding weight vector w:(w1,w2) of the input vector x:(x1,x2) to the AND and OR node
Step2: The output $\hat{y}_1$ from the AND node will be inputted to the NOT node with weight w_{NOT}
Step3: The output $\hat{y}_2$ from the OR node and the output $\hat{y}_3$ from 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.



Source

Weights assumed are:
$w_1 = w_2 = w_{AND1} = w_{AND2} = 1$
$w_{NOT} = -1$

And the bias assumed are:

$b_{AND} = -1.5$
$b_{OR} = -0.5$
$b_{NOT} = 0.5$

Program :

```
def unitStep(v):
        if v >= 0:
                return 1
        else:
                return 0

def perceptronModel(x, w, b):
        v = np.dot(w, x) + b
        y = unitStep(v)
        return y

def NOT_logicFunction(x):
        wNOT = -1
        bNOT = 0.5
        return perceptronModel(x, wNOT, bNOT)

def AND_logicFunction(x):
        w = np.array([1, 1])
        bAND = -1.5
        return perceptronModel(x, w, bAND)


def OR_logicFunction(x):
        w = np.array([1, 1])
        bOR = -0.5
        return perceptronModel(x, w, bOR)

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
```

Output:

```
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 :

In summary, the Multilayer Perceptron (MLP) algorithm has proven its capability in solving complex problems like simulating the XOR gate, which is a classic example of a problem that can't be solved with a single-layer perceptron because it's too simple. However, by using hidden layers and smart activation functions, the MLP can handle this problem and capture the intricate, non-linear connections within it.