Name: Hirenkumar vyas
Roll no: 32
Branch: BE_AI & DS

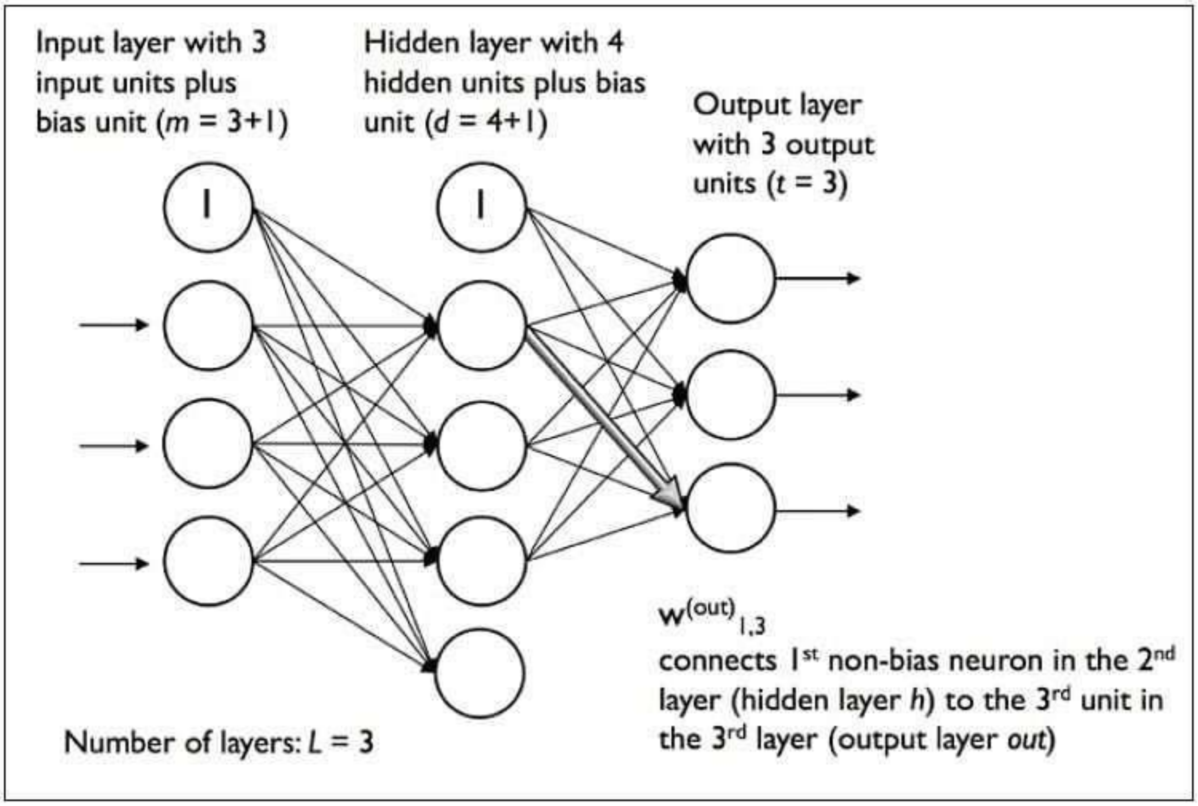
Experiment No. 2
Implement Multilayer Perceptron algorithm to simulate XOR gate
Date of Performance:
Date of Submission:

Aim: Implement Multilayer Perceptron algorithm to simulate XOR gate.

Objective: Ability to perform experiments on different architectures of multilayer perceptron.

Theory:

A multilayer artificial neuron network is an integral part of deep learning. And this lessonwill help you with an overview of multilayer ANN along with overfitting and underfitting.



A fully connected multi-layer neural network is called a Multilayer Perceptron (MLP).
At has 3 layers including one hidden layer. If it has more than 1 hidden layer, it is called a deep ANN. An MLP is a typical example of a feedforward artificial neural network. In thisfigure, the ith activation unit in the lth layer is denoted as ai(l).

The number of layers and the number of neurons are referred to as hyperparameters of aneural network, and these need tuning. Cross-validation techniques must be used to find ideal values for these.
The weight adjustment training is done via backpropagation. Deeper neural networks are better at processing data. However, deeper layers can lead to vanishing gradient problems. Special algorithms are required to solve this issue.

A multilayer perceptron (MLP) is a feed forward artificial neural network that generates aset of outputs from a set of inputs. An MLP is characterized by several layers of input nodes connected as a directed graph between the input nodes connected as a directed graph between the input and output layers. MLP uses backpropagation for training the network. MLP is a deep learning method.

Program: import
numpy as np 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_logicFun
ction(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

test1 = np.array([0, 0])
test2 = np.array([0, 1])
test3 = np.array([1, 0])
test4 = np.array([1, 1])

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

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

Conclusion:
In this experiment, we successfully implemented a Multilayer Perceptron algorithm to simulate the XOR gate, a problem that cannot be solved using a single-layer perceptron due to its nonlinear nature. The MLP architecture, consisting of an input layer, a hidden layer, and an output layer, allowed us to learn the underlying pattern of the XOR gate. Through training with backpropagation, the model adjusted its weights and biases to accurately predict the XOR outputs for different input combinations.