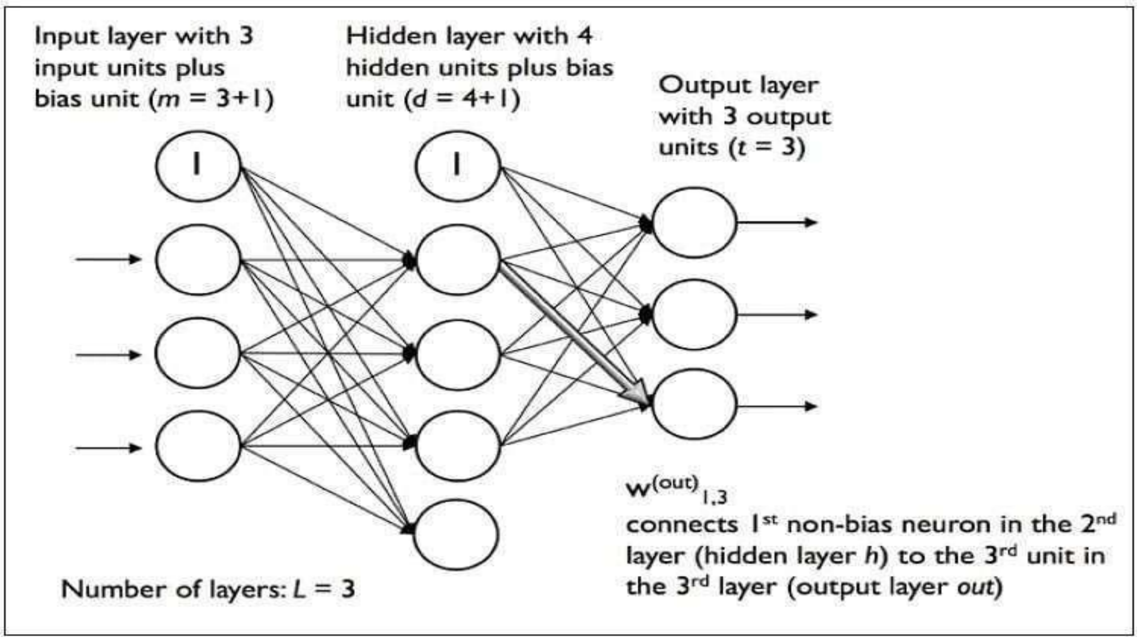
Name : OM RAJESH TIWARI	
Roll no : 29	
Experiment No. 2	
Implement Multilayer Perceptron algorithm to simulate XOR gate	
Date of Performance:31/07/23	
Date of Submission:8/8/23	

Aim: Implement Multilayer Perceptron algorithm to simulate XOR gate.
perceptorn

Objective: Ability to perform experiments on different architectures of multilayer

Theory:

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



of
A fully connected multi-layer neural network is called a Multilayer Perceptron (MLP).

At has 3layers 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 this figure, 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 a neural 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 deep learning method input and output layers 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 a set 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 MLP uses backpropagation for training the network. MLP is a

Code :

```
# importing Python library
import numpy as np

if v >= 0:
# define Unit Step Function
def unitStep(v):

        return 1
    else:
        return 0

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

# wNOT 0.5
# NOT Logic Function
    = -1,     =
```

```
def NOT_logicFunction(x):
bNOT   0.5
                    = -1

                   =

        return perceptronModel(x, wNOT, bNOT)


def AND_logicFunction(x):
# AND Logic Function

# here w1 = wAND1 = 1,

                                   -1.5



                           [1, 1

                    = -1

        return perceptronModel(x, w, bAND)

def OR_logicFunction(x):
# OR Logic Function

        = 1,    = 1,      =0.5



                           [1, 1

                 =

        return perceptronModel(x, w, bOR)

def XOR_logicFunction(x):
# XOR Logic Function

# with AND, OR and NOT

# function calls in sequence



        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

test2 = np.array([0, 1])
# testing the Perceptron Model

test1 = np.array([0, 0])

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

Conclusion:

In conclusion, the implementation of the Multilayer Perceptron (MLP) algorithm to simulate the XOR gate showcases the synergy between network architecture and the backpropagation algorithm in solving non-linear classification problems. The XOR gate simulation, a classic example of a problem that cannot be solved by a single-layer perceptron, underscores the significance of introducing hidden layers and utilizing backpropagation to achieve accurate results