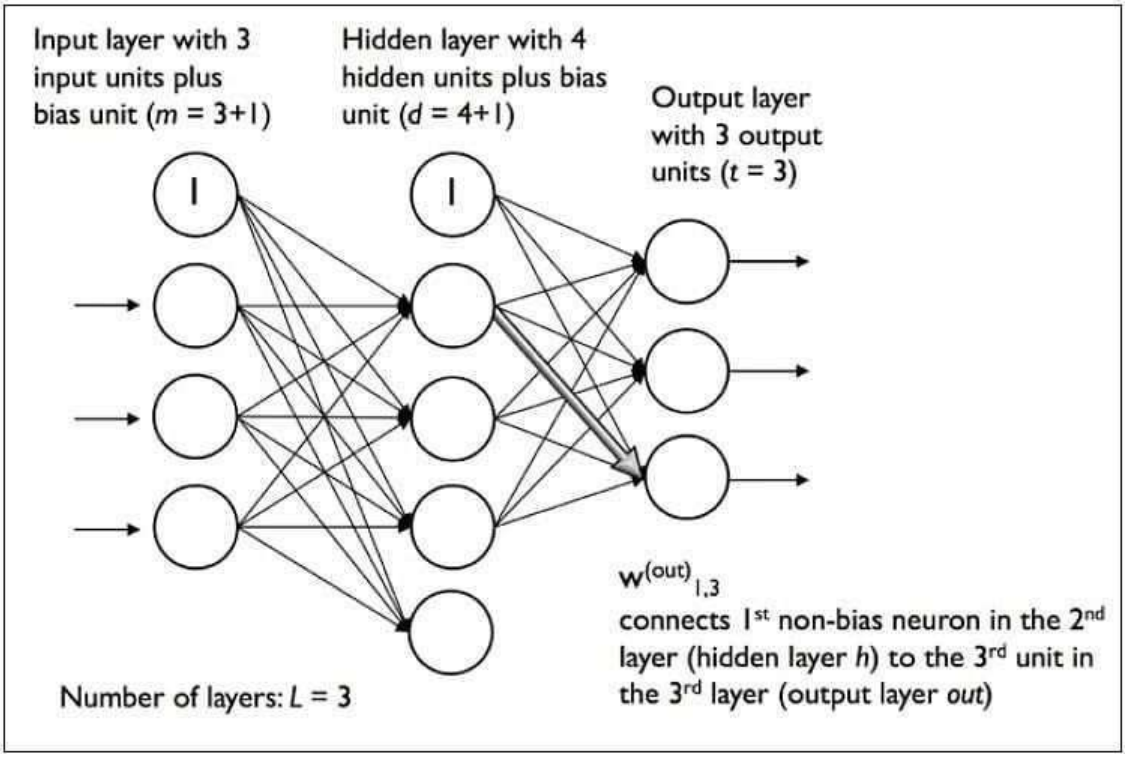
Experiment No. 2
Implement Multilayer Perceptron algorithm to simulate XOR gate
Date of Performance:
Date of Submission:

Aim: Implement Multilayer Perceptron algorithm to simulate XOR gate.

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



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 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 of 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 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 the input and output layers. MLP uses backpropagation for training the network. MLP is a deep learning method.

CODE:

```
library 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_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
 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)))
```

OUTPUT:

```
C:\Users\student>python -u "C:\Users\student\AppData\Local\Temp\tempCodeRunnerFile.python"
XOR(0, 1) = 1
XOR(1, 1) = 0
XOR(0, 0) = 0
XOR(1, 0) = 1
```

Conclusion:

The network and backpropagation algorithm are key components of artificial neural networks (ANNs). A neural network consists of interconnected nodes (neurons) in an input layer, hidden layers, and an output layer, with each connection having a weight that influences signal strength. Backpropagation is a supervised learning technique that trains the network by minimizing the error between predicted and actual outputs. It computes the gradient of the loss function with respect to each weight using the chain rule, propagates the error backward, and updates the weights to reduce the error, typically using gradient descent. This process iterates until the network achieves acceptable accuracy.