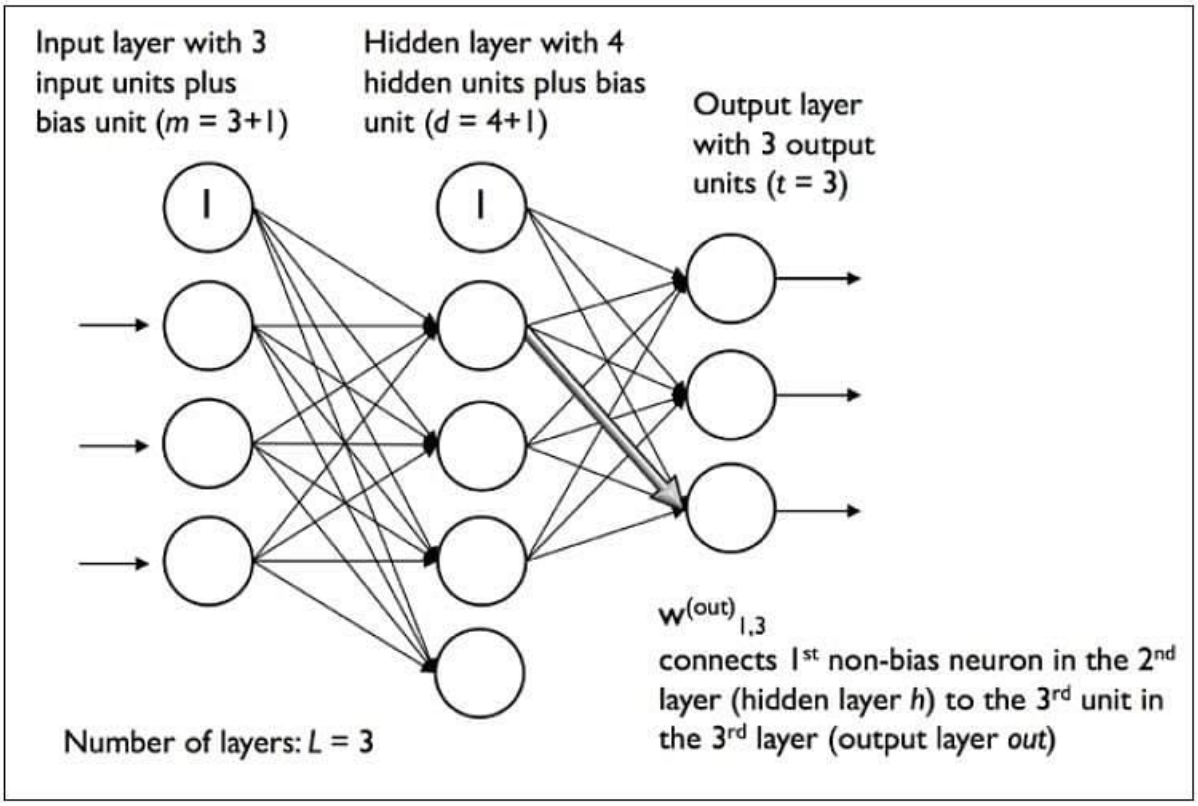
Experiment No. 2
Implement Multilayer Perceptron algorithm to simulate XOR gate
Date of Performance:
Date of Submission:

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:

```
import numpy as np
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
def sigmoid_derivative(x):
    return x * (1 - x)
inputs = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
outputs = np.array([[0], [1], [1], [0]])
input_layer_neurons = 2
hidden_layer_neurons = 2
output_neurons = 1
hidden_weights = np.random.rand(input_layer_neurons, hidden_layer_neurons)
hidden_bias = np.random.rand(1, hidden_layer_neurons)
output_weights = np.random.rand(hidden_layer_neurons, output_neurons)
output_bias = np.random.rand(1, output_neurons)
learning_rate = 0.1
epochs = 10000
for epoch in range(epochs):
    hidden_layer_input = np.dot(inputs, hidden_weights) + hidden_bias
    hidden_layer_output = sigmoid(hidden_layer_input)
```

```
    output_layer_input = np.dot(hidden_layer_output, output_weights) + output_bias
    predicted_output = sigmoid(output_layer_input)
    error = outputs - predicted_output
    d_predicted_output = error * sigmoid_derivative(predicted_output)
    error_hidden_layer = d_predicted_output.dot(output_weights.T)
    d_hidden_layer = error_hidden_layer * sigmoid_derivative(hidden_layer_output)
    output_weights += hidden_layer_output.T.dot(d_predicted_output) * learning_rate
    output_bias += np.sum(d_predicted_output, axis=0, keepdims=True) * learning_rate
    hidden_weights += inputs.T.dot(d_hidden_layer) * learning_rate
    hidden_bias += np.sum(d_hidden_layer, axis=0, keepdims=True) * learning_rate
print("Output after training:")
print(predicted_output)
```

Output:

```
PS D:\Programming language\DSA\Array and String> python .\MLP.py
Output after training:
[[0.04778607]
 [0.49709726]
 [0.94727535]
 [0.50464753]]
```

Conclusion: