Experiment 02

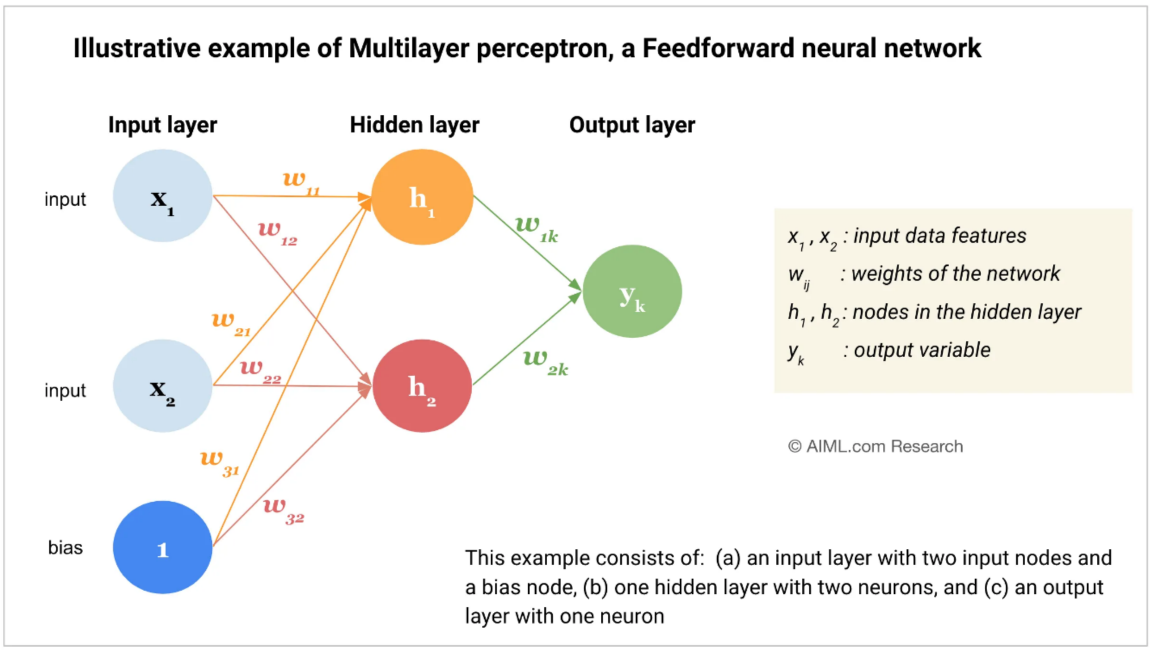
Aim: Implement Multilayer Perceptron (MLP) algorithm to simulate XOR gate.

Apparatus: Laptop / Desktop with Internet Connection, Google Colab / Jupyter Notebook

Theory:

The **Multilayer Perceptron (MLP)** is a class of feedforward artificial neural networks composed of at least three layers: an **input layer**, **one or more hidden layers**, and an **output layer**. Unlike single-layer perceptrons, MLPs can solve **non-linearly separable problems** such as the **XOR function**.

The XOR gate returns True only when the inputs differ. It cannot be implemented using a single-layer perceptron because it's not linearly separable. However, an MLP with **nonlinear activation functions** can model this.



XOR Gate – Truth Table:

Input A	Input B	Output (A XOR B)
0	0	0
0	1	1
1	0	1
1	1	0

Key Components of MLP:

1. **Input Layer** – Takes binary inputs A and B
2. **Hidden Layer** – Learns intermediate representations using activation functions like sigmoid
3. **Output Layer** – Produces final prediction
4. **Weights & Biases** – Learn through backpropagation
5. **Activation Function** – Typically sigmoid, tanh, or ReLU

Implementation Code (Python + NumPy):

```
# Implement Multilayer Perceptron to simulate XOR gate
import numpy as np


# Activation function & its derivative
def sigmoid(x):
    return 1 / (1 + np.exp(-x))


def sigmoid_derivative(x):
    return x * (1 - x)


# XOR dataset
X = np.array([[0,0], [0,1], [1,0], [1,1]])
Y = np.array([[0], [1], [1], [0]])


# Hyperparameters
epochs = 10000
lr = 0.1
input_neurons, hidden_neurons, output_neurons = 2, 2, 1
```

```
# Initialize weights and biases
hidden_weights = np.random.uniform(size=(input_neurons, hidden_neurons))
hidden_bias = np.random.uniform(size=(1, hidden_neurons))
output_weights = np.random.uniform(size=(hidden_neurons, output_neurons))
output_bias = np.random.uniform(size=(1, output_neurons))

# Training loop
for epoch in range(epochs):
    # Forward propagation
    hidden_layer_input = np.dot(X, 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)

    # Backpropagation
    error = Y - predicted_output
    d_predicted = error * sigmoid_derivative(predicted_output)

    error_hidden = d_predicted.dot(output_weights.T)
    d_hidden = error_hidden * sigmoid_derivative(hidden_layer_output)

    # Update weights and biases
    output_weights += hidden_layer_output.T.dot(d_predicted) * lr
    output_bias += np.sum(d_predicted, axis=0, keepdims=True) * lr
```

```
    hidden_weights += X.T.dot(d_hidden) * lr
    hidden_bias += np.sum(d_hidden, axis=0, keepdims=True) * lr


# Display final outputs
print("Final output after training:")
print(predicted_output)
print("\nRounded outputs:")
print(np.round(predicted_output))
```

Output:

```
Final output after training:
[[0.05811025]
 [0.94658224]
 [0.9466309 ]
 [0.05749416]]

Rounded outputs:
[[0.]
 [1.]
 [1.]
 [0.]]
```

Conclusion:

The XOR logic gate was successfully implemented using a **Multilayer Perceptron** with a hidden layer. This experiment demonstrates how MLPs can solve **non-linearly separable problems** using backpropagation and non-linear activation functions.