Objective : To implement a Multi-Layer Perceptron (MLP) using NumPy that can learn the XOR Boolean function.

Description of the model : The MLP consists of an input layer with 2 neurons, a hidden layer with 2 neurons, and an output layer with 1 neuron.

The activation function used is the **sigmoid function**.

The network is trained using backpropagation with **gradient descent**.

Python Implementation :

```
import numpy as np

def sigmoid(x):

    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):

    return x * (1 - x)

X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

y = np.array([[0], [1], [1], [0]])

input_neurons = 2

hidden_neurons = 2

output_neurons = 1

np.random.seed(42)

weights_input_hidden = np.random.uniform(-1, 1, (input_neurons, hidden_neurons))

weights_hidden_output = np.random.uniform(-1, 1, (hidden_neurons, output_neurons))

bias_hidden = np.random.uniform(-1, 1, (1, hidden_neurons))

bias_output = np.random.uniform(-1, 1, (1, output_neurons))

learning_rate = 0.5
```

```
epochs = 10000

for epoch in range(epochs):

    hidden_layer_input = np.dot(X, weights_input_hidden) + bias_hidden

    hidden_layer_output = sigmoid(hidden_layer_input)

    output_layer_input = np.dot(hidden_layer_output, weights_hidden_output) + bias_output

    output = sigmoid(output_layer_input)

    error = y - output

    d_output = error * sigmoid_derivative(output)

    error_hidden_layer = d_output.dot(weights_hidden_output.T)

    d_hidden_layer = error_hidden_layer * sigmoid_derivative(hidden_layer_output)

    weights_hidden_output += hidden_layer_output.T.dot(d_output) * learning_rate

    bias_output += np.sum(d_output, axis=0, keepdims=True) * learning_rate

    weights_input_hidden += X.T.dot(d_hidden_layer) * learning_rate

    bias_hidden += np.sum(d_hidden_layer, axis=0, keepdims=True) * learning_rate

print("Final Output:")

print(output)
```

Description of code :

- The model initializes random weights and biases.
- The forward pass computes activations for the hidden and output layers.
- The error is computed, and gradients are backpropagated to update weights.
- Training continues for 10,000 epochs with a learning rate of 0.5.

Performance Evaluation :

The final output approximates the XOR truth table:

Input: [0,0] -> Output ≈ 0

Input: [0,1] -> Output ≈ 1

Input: [1,0] -> Output ≈ 1

Input: [1,1] -> Output ≈ 0

The network successfully learns the XOR function.