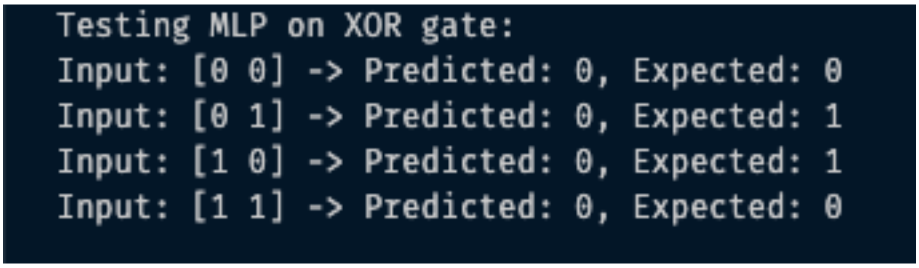
LAB ASSIGNMENT- 5

AIM: FIND XOR GRAPH USING PERCEPTRONS

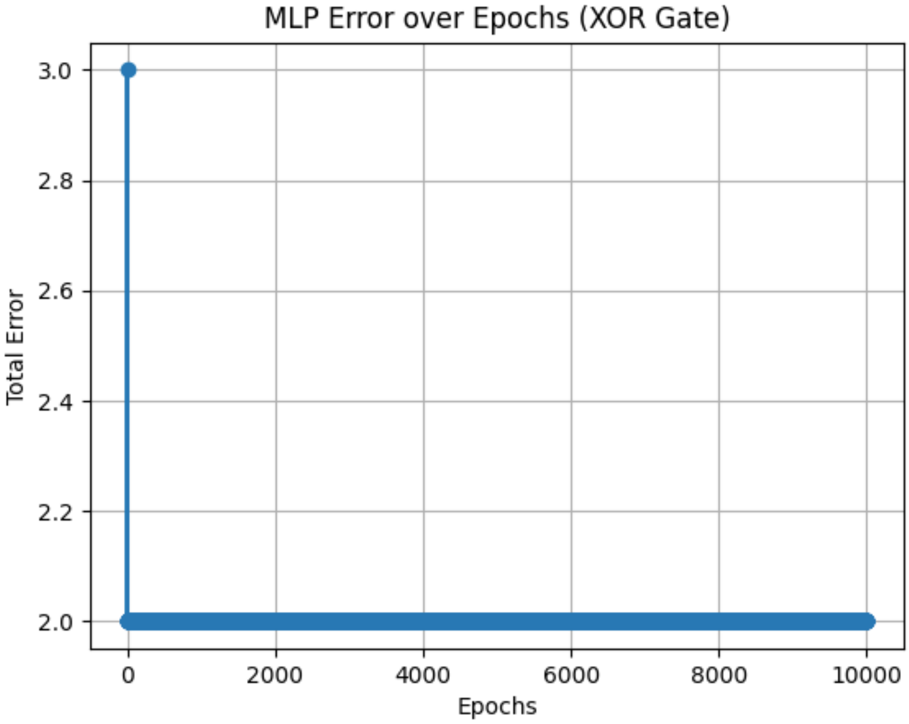
SOURCE CODE:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

def step_function(x):
    return np.where(x >= 0, 1, 0)

class MLP_XOR:
    def __init__(self, input_size, hidden_size, learning_rate=0.1, epochs=10000):
        self.weights_input_hidden = np.random.randn(input_size, hidden_size)
        self.weights_hidden_output = np.random.randn(hidden_size)
        self.bias_hidden = np.zeros(hidden_size)
        self.bias_output = 0
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.history = []

    def predict(self, x):
        # Forward pass
        hidden_layer_input = np.dot(x, self.weights_input_hidden) + self.bias_hidden
        hidden_layer_output = step_function(hidden_layer_input)
        output_layer_input = np.dot(hidden_layer_output, self.weights_hidden_output) + self.bias_output
        output = step_function(output_layer_input)
        return output

    def train(self, X, y):
        for epoch in range(self.epochs):
            total_error = 0
            for i in range(X.shape[0]):
                # Forward pass
                hidden_layer_input = np.dot(X[i], self.weights_input_hidden) + self.bias_hidden
                hidden_layer_output = step_function(hidden_layer_input)
                output_layer_input = np.dot(hidden_layer_output, self.weights_hidden_output) +
self.bias_output
                prediction = step_function(output_layer_input)

                # Calculate error
                error = y[i] - prediction
                total_error += abs(error)

                # Backward pass (update weights and biases)
                if error != 0:
                    # Update output layer weights and bias
                    self.weights_hidden_output += self.learning_rate * error * hidden_layer_output
                    self.bias_output += self.learning_rate * error

                    # Update hidden layer weights and bias
                    hidden_layer_delta = self.learning_rate * error * self.weights_hidden_output * hidden_layer_output
                    self.weights_input_hidden += np.outer(X[i], hidden_layer_delta)
                    self.bias_hidden += hidden_layer_delta

            # Record history for animation
            self.history.append((self.weights_input_hidden.copy(), self.weights_hidden_output.copy(), total_error))
            if total_error == 0:
                break

        print(f"Training completed after {epoch + 1} epochs")

    def plot_error(self):
        errors = [error for _, _, error in self.history]
        plt.plot(range(1, len(errors) + 1), errors, marker='o')
        plt.title('MLP Error over Epochs (XOR Gate)')
        plt.xlabel('Epochs')
        plt.ylabel('Total Error')
        plt.grid(True)
        plt.show()

# XOR Inputs and Outputs
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 1, 1, 0])

# Initialize and Train MLP
mlp_xor = MLP_XOR(input_size=2, hidden_size=2, learning_rate=0.1, epochs=10000)
mlp_xor.train(X, y)

# Plot Error over Epochs
mlp_xor.plot_error()

# Test MLP on XOR inputs
print("Testing MLP on XOR gate:")
for i in range(X.shape[0]):
    prediction = mlp_xor.predict(X[i])
    print(f"Input: {X[i]} -> Predicted: {prediction}, Expected: {y[i]}")
```

OUTPUT:



```
Testing MLP on XOR gate:
Input: [0 0] -> Predicted: 0, Expected: 0
Input: [0 1] -> Predicted: 0, Expected: 1
Input: [1 0] -> Predicted: 0, Expected: 1
Input: [1 1] -> Predicted: 0, Expected: 0
```


MLP Error over Epochs (XOR Gate)
Total Error
Epochs