Neural Network Laboratory Record

B.E. (AI & DS) – VI Semester

Name: Rahul Raj

Roll No: 24UADS1041

Experiment Title: Implementation of Multi-Layer Perceptron for XOR Problem

1. Objective

To implement a multi-layer perceptron (MLP) network with one hidden layer using NumPy in Python. Demonstrate that it can learn the XOR Boolean function.

2. Introduction

A Multi-Layer Perceptron (MLP) is a feed-forward artificial neural network consisting of an input layer, one or more hidden layers, and an output layer. Unlike a single-layer perceptron, an MLP can solve non-linear problems using hidden neurons and non-linear activation functions such as sigmoid.

The XOR function is a classic example of a non-linearly separable problem. A single perceptron cannot solve it, but an MLP can learn it by forming a non-linear decision boundary.

3. Dataset Used

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0

4. Python Implementation

```
# ------------------------------

# MLP for XOR Problem using NumPy

# ------------------------------


import numpy as np

import matplotlib.pyplot as plt
```

```
# -------------------------------
# 1 XOR Dataset
# -------------------------------
X = np.array([
    [0, 0],
    [0, 1],
    [1, 0],
    [1, 1]
])

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

# -------------------------------
# 2 Activation Function
# -------------------------------
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

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

# -------------------------------
# 3 Initialize Network Parameters
# -------------------------------
np.random.seed(42)

input_size = 2
hidden_size = 4
output_size = 1
```

```
# Weights
W1 = np.random.randn(input_size, hidden_size)
b1 = np.zeros((1, hidden_size))

W2 = np.random.randn(hidden_size, output_size)
b2 = np.zeros((1, output_size))

# -------------------------------
# 4 Hyperparameters
# -------------------------------
learning_rate = 0.1
epochs = 10000
losses = []

# 5 Training MLP (Backpropagation)
# -------------------------------
for epoch in range(epochs):
    # Forward Pass
    hidden_input = np.dot(X, W1) + b1
    hidden_output = sigmoid(hidden_input)

    final_input = np.dot(hidden_output, W2) + b2
    output = sigmoid(final_input)

    # Compute Loss (MSE)
    loss = np.mean((y - output)**2)
    losses.append(loss)

    # Backpropagation
```

```python
    d_output = (y - output) * sigmoid_derivative(output)
    d_hidden = d_output.dot(W2.T) * sigmoid_derivative(hidden_output)

    # Update Weights and Biases
    W2 += hidden_output.T.dot(d_output) * learning_rate
    b2 += np.sum(d_output, axis=0, keepdims=True) * learning_rate

    W1 += X.T.dot(d_hidden) * learning_rate
    b1 += np.sum(d_hidden, axis=0, keepdims=True) * learning_rate

    # Print loss every 1000 epochs
    if epoch % 1000 == 0:
        print(f"Epoch {epoch}, Loss: {loss:.6f}")

# -------------------------------
# 6 Final Predictions
# -------------------------------
print("\nFinal Predictions (after training):")
print(output)

# -------------------------------
# 7 Plot Loss Curve
# -------------------------------
plt.plot(losses)
plt.title("Loss vs Epochs")
plt.xlabel("Epochs")
plt.ylabel("Mean Squared Error")
plt.show()

# -------------------------------
```

```
# 8 Plot Decision Boundary
# ------------------------------
def plot_decision_boundary():
    x_min, x_max = -0.5, 1.5
    y_min, y_max = -0.5, 1.5
    h = 0.01
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    grid = np.c_[xx.ravel(), yy.ravel()]

    hidden = sigmoid(np.dot(grid, W1) + b1)
    out = sigmoid(np.dot(hidden, W2) + b2)
    Z = out.reshape(xx.shape)

    plt.contourf(xx, yy, Z, levels=[0, 0.5, 1], alpha=0.3, colors=['#FFAAAA','#AAAAFF'])
    plt.scatter(X[:,0], X[:,1], c=y.flatten(), edgecolors='k', s=100)
    plt.title("Decision Boundary")
    plt.xlabel("Input 1")
    plt.ylabel("Input 2")
    plt.show()

plot_decision_boundary()
```

5. Output

Epoch 0, Loss: 0.283190

Epoch 1000, Loss: 0.245226

Epoch 2000, Loss: 0.212412

Epoch 3000, Loss: 0.150331

Epoch 4000, Loss: 0.057156

Epoch 5000, Loss: 0.020929

Epoch 6000, Loss: 0.010685

Epoch 7000, Loss: 0.006679

Epoch 8000, Loss: 0.004683

Epoch 9000, Loss: 0.003527

Final Predictions (after training):

[[0.03730284]

[0.9491398]

[0.94480964]

[0.06425255]]

6. Conclusion

The Multi-Layer Perceptron successfully learned the XOR Boolean function. The decreasing loss over epochs indicates stable convergence. This experiment demonstrates that hidden layers enable neural networks to solve non-linear problems.

7. Graphical Results

Figure 1: Loss Curve

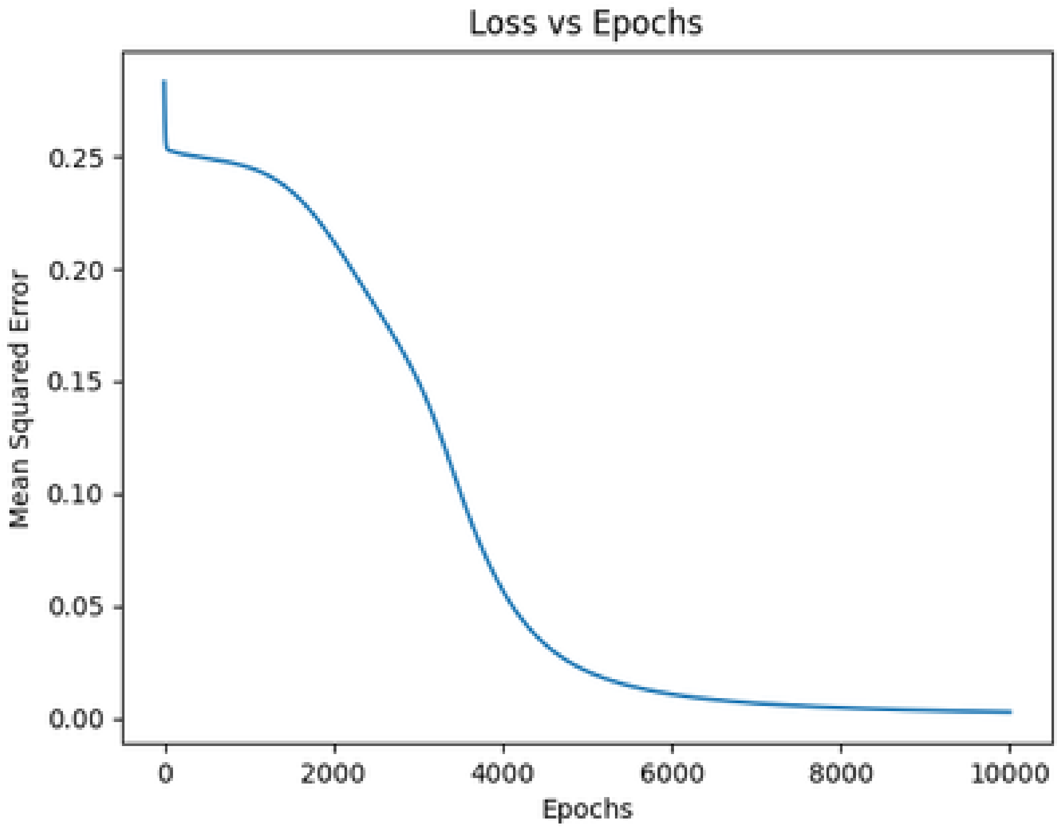

Figure 2: Decision Boundary