# Rajshahi University of Engineering & Technology

### Heaven’s Light is Our Guide

**Course Code**: CSE 4204

**Course Title**: Sessional based on CSE 4203

**Experiment no.**: 03


**Submitted by:**
**Name**: Ayan Sarkar
**Roll**: 1903162
**Section**: C
**Department**: CSE

**Submitted to:**
Md. Mazharul Islam
Lecturer,
Department of CSE,
RUET

**Problem:** Multilayer Perceptron (MLP) for Solving the XOR Problem

**Experiment Description:** The experiment involves designing, implementing, and training an MLP to classify XOR inputs. Key steps include:

**Data Preparation:** Using the XOR dataset with

- inputs X=[[0,0],[0,1],[1,0],[1,1]]
- outputs y=[[0],[1],[1],[0]].

**Network Architecture:**

- Input Layer: 2 neurons (for two features).
- Hidden Layer: 2 neurons with sigmoid activation.
- Output Layer: 1 neuron with sigmoid activation.

**Training:**

- Forward Propagation: Compute outputs using weights and biases.
- Backpropagation: Update weights and biases using gradient descent, with the derivative of the sigmoid computed inline as $\sigma(x)(1-\sigma(x))$.

**Dataset Characteristic:**

- Samples: 4 (all possible binary combinations of two inputs).
- Features: 2 (binary values: 0 or 1).
- Labels: Binary (0 or 1).
- Non-Linearity: The dataset is not linearly separable, necessitating a multi-layer network.

**Code:**

```
import numpy as np
import matplotlib.pyplot as plt

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Input dataset (XOR problem)
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])

```

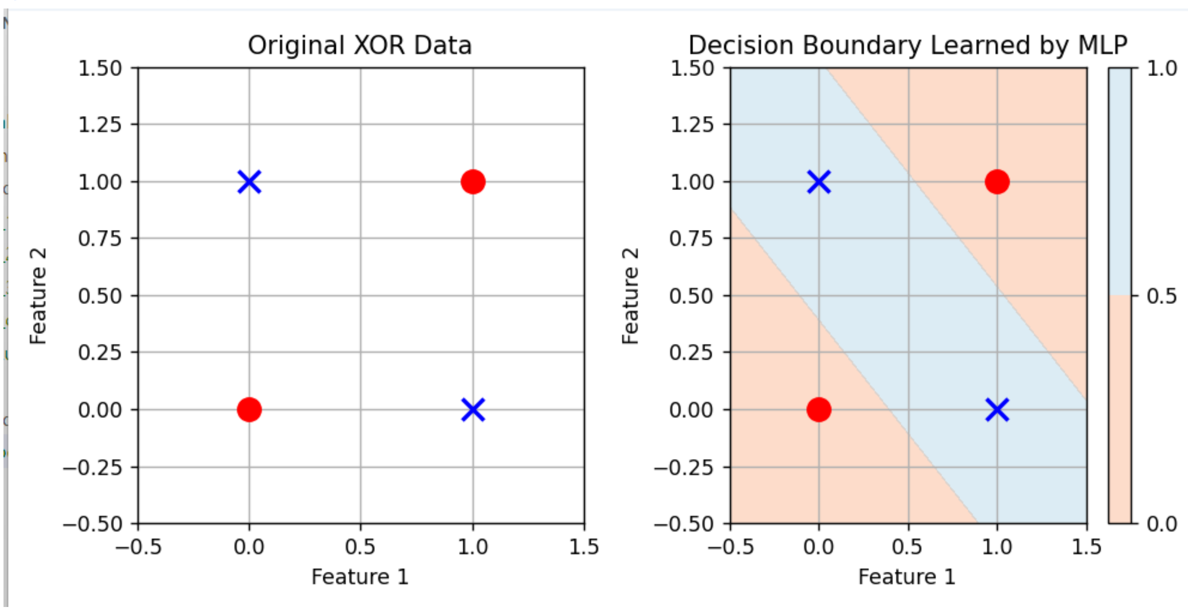
```python
# 1. Visualize Original XOR Data
plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
for i in range(len(X)):
    if y[i] == 0:
        marker = 'ro'  # Red circles for class 0
    else:
        marker = 'bx'  # Blue crosses for class 1
    plt.plot(X[i, 0], X[i, 1], marker, markersize=10, markeredgewidth=2)
plt.title("Original XOR Data")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.grid(True)
plt.xlim(-0.5, 1.5)
plt.ylim(-0.5, 1.5)
```

```python
input_neurons = X.shape[1]
hidden_neurons = 2
output_neurons = 1

np.random.seed(42)
weights_input_hidden = np.random.uniform(size=(input_neurons, hidden_neurons))
weights_hidden_output = np.random.uniform(size=(hidden_neurons, output_neurons))

bias_hidden = np.random.uniform(size=(1, hidden_neurons))
bias_output = np.random.uniform(size=(1, output_neurons))

learning_rate = 0.1
epochs = 10000
```

```python
for epoch in range(epochs):
    hidden_input = np.dot(X, weights_input_hidden) + bias_hidden
    hidden_output = sigmoid(hidden_input)
    output_input = np.dot(hidden_output, weights_hidden_output) + bias_output
    predicted_output = sigmoid(output_input)

    error = y - predicted_output
    d_predicted = error * predicted_output * (1 - predicted_output)
    weights_hidden_output += hidden_output.T.dot(d_predicted) * learning_rate
    bias_output += np.sum(d_predicted, axis=0, keepdims=True) * learning_rate

    error_hidden = d_predicted.dot(weights_hidden_output.T)
    d_hidden = error_hidden * hidden_output * (1 - hidden_output)
    weights_input_hidden += X.T.dot(d_hidden) * learning_rate
    bias_hidden += np.sum(d_hidden, axis=0, keepdims=True) * learning_rate
```

```python
# 2. Visualize Decision Boundary
plt.subplot(1, 2, 2)

xx, yy = np.meshgrid(np.linspace(-0.5, 1.5, 100), np.linspace(-0.5, 1.5, 100))
grid = np.c_[xx.ravel(), yy.ravel()]

hidden_layer = sigmoid(np.dot(grid, weights_input_hidden) + bias_hidden)
predictions = sigmoid(np.dot(hidden_layer, weights_hidden_output) + bias_output)
Z = predictions.reshape(xx.shape)

plt.contourf(xx, yy, Z, levels=[0, 0.5, 1], cmap=plt.cm.RdYlBu, alpha=0.3)
plt.colorbar()

for i in range(len(X)):
    if y[i] == 0:
        marker = 'ro'
    else:
        marker = 'bx'
    plt.plot(X[i, 0], X[i, 1], marker, markersize=10, markeredgewidth=2)

plt.title("Decision Boundary Learned by MLP")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.grid(True)
plt.tight_layout()
plt.show()

# Print final predictions
print("\nFinal Predictions:")
print(predicted_output)
```

### Visualization:



**Result:**

- **Training Error:** Decreased from ~0.5 to ~0.0004 over 10,000 epochs.
- **Final Predictions:**

```
[[0.06028315]
 [0.94448041]
 [0.94437509]
 [0.05996173]]
```

- **Accuracy**: 100% on the training set.

**Conclusion:** The Multilayer Perceptron (MLP) successfully solved the XOR problem, demonstrating its capability to model non-linear decision boundaries through the inclusion of a hidden layer and the Backpropagation algorithm. By training on the XOR dataset, the MLP achieved 100% accuracy, with predictions closely matching the expected outputs, and reduced the mean absolute error to near-zero values over 10,000 epochs.