

Green University of Bangladesh

Department of Computer Science and Engineering (CSE)

### Faculty of Sciences and Engineering Semester: Spring, Year: 2025, B.Sc. in CSE (Day)

**Lab Report:03**

### Code: CSE-412

**Section: 222 D3 Course Title: Machine Learning**

# **Lab Experiment Name:**mlpFromScratch

**Student Details**

|  |  |
| --- | --- |
| **Name** | **ID** |
| **Md.Jabed Mollah** | **222002167** |

### Lab Date : 06/08/2025

**Submission Date : 17/08/2025** **Course Teacher’s Name: Md. Sabbir Hosen Mamu**

**Lab Report Status Marks: ………………………………… Comments:..............................................**

**Signature:..................**

**... Date:...........................**

**Lab 03:**

## TITLE

MLP from Scratch: Solving the XOR Classification Task

## OBJECTIVES

This project aims to create a two-layer Multi-Layer Perceptron (MLP) using Python and NumPy to solve the XOR classification problem. It demonstrates forward propagation, activation functions, and backpropagation principles without high-level deep learning libraries. The project also experiments with hyperparameters and evaluates the trained models using performance metrics.

### Ml Code:

import numpy as np

import matplotlib.pyplot as plt

class SimpleMLP:

    def \_\_init\_\_(self, hidden\_activation='sigmoid'):

        self.W1 = np.random.randn(2, 2) \* 0.5

        self.b1 = np.zeros((1, 2))

        self.W2 = np.random.randn(2, 1) \* 0.5

        self.b2 = np.zeros((1, 1))

        self.hidden\_activation = hidden\_activation

    def sigmoid(self, x):

        return 1 / (1 + np.exp(-np.clip(x, -500, 500)))

    def relu(self, x):

        return np.maximum(0, x)

    def forward(self, X):

        self.z1 = np.dot(X, self.W1) + self.b1

        if self.hidden\_activation == 'relu':

            self.a1 = self.relu(self.z1)

        else:

            self.a1 = self.sigmoid(self.z1)

        self.z2 = np.dot(self.a1, self.W2) + self.b2

        self.a2 = self.sigmoid(self.z2)

        return self.a2

    def backward(self, X, y, output, lr):

        m = X.shape[0]

        dz2 = output - y

        dW2 = np.dot(self.a1.T, dz2) / m

        db2 = np.sum(dz2, axis=0, keepdims=True) / m

        da1 = np.dot(dz2, self.W2.T)

        if self.hidden\_activation == 'relu':

            dz1 = da1 \* (self.z1 > 0)

        else:

            dz1 = da1 \* self.a1 \* (1 - self.a1)

        dW1 = np.dot(X.T, dz1) / m

        db1 = np.sum(dz1, axis=0, keepdims=True) / m

        self.W2 -= lr \* dW2

        self.b2 -= lr \* db2

        self.W1 -= lr \* dW1

        self.b1 -= lr \* db1

    def train(self, X, y, lr=0.1, epochs=1000):

        losses = []

        for epoch in range(epochs):

            output = self.forward(X)

            loss = np.mean((output - y) \*\* 2)

            losses.append(loss)

            self.backward(X, y, output, lr)

        return losses

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

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

configs = [

    {'lr': 0.01, 'epochs': 5000, 'activation': 'sigmoid'},

    {'lr': 0.1, 'epochs': 1000, 'activation': 'sigmoid'},

    {'lr': 0.5, 'epochs': 500, 'activation': 'relu'},

]

best\_accuracy = 0

best\_model = None

for config in configs:

    model = SimpleMLP(config['activation'])

    losses = model.train(X, y, config['lr'], config['epochs'])

    predictions = model.forward(X)

    pred\_binary = (predictions > 0.5).astype(int)

    accuracy = np.mean(pred\_binary == y)

    print(f"Config: {config} - Accuracy: {accuracy:.3f}")

    if accuracy > best\_accuracy:

        best\_accuracy = accuracy

        best\_model = model

predictions = best\_model.forward(X)

pred\_binary = (predictions > 0.5).astype(int)

tp = np.sum((pred\_binary == 1) & (y == 1))

tn = np.sum((pred\_binary == 0) & (y == 0))

fp = np.sum((pred\_binary == 1) & (y == 0))

fn = np.sum((pred\_binary == 0) & (y == 1))

accuracy = (tp + tn) / (tp + tn + fp + fn)

precision = tp / (tp + fp) if (tp + fp) > 0 else 0

recall = tp / (tp + fn) if (tp + fn) > 0 else 0

f1 = 2 \* precision \* recall / (precision + recall) if (precision + recall) > 0 else 0

print(f"\nBest Model Results:")

print(f"Accuracy: {accuracy:.3f}")

print(f"Precision: {precision:.3f}")

print(f"Recall: {recall:.3f}")

print(f"F1-Score: {f1:.3f}")

thresholds = np.linspace(0, 1, 100)

tpr\_values = []

fpr\_values = []

for threshold in thresholds:

    pred\_thresh = (predictions > threshold).astype(int)

    tp = np.sum((pred\_thresh == 1) & (y == 1))

    tn = np.sum((pred\_thresh == 0) & (y == 0))

    fp = np.sum((pred\_thresh == 1) & (y == 0))

    fn = np.sum((pred\_thresh == 0) & (y == 1))

    tpr = tp / (tp + fn) if (tp + fn) > 0 else 0

    fpr = fp / (fp + tn) if (fp + tn) > 0 else 0

    tpr\_values.append(tpr)

    fpr\_values.append(fpr)

auc = np.trapz(tpr\_values, fpr\_values)

print(f"AUC: {abs(auc):.3f}")

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))

final\_losses = best\_model.train(X, y, 0.1, 1000)

ax1.plot(final\_losses)

ax1.set\_title('Training Loss')

ax1.set\_xlabel('Epoch')

ax1.set\_ylabel('MSE Loss')

ax2.plot(fpr\_values, tpr\_values, 'b-', label=f'ROC (AUC = {abs(auc):.3f})')

ax2.plot([0, 1], [0, 1], 'r--', label='Random')

ax2.set\_xlabel('False Positive Rate')

ax2.set\_ylabel('True Positive Rate')

ax2.set\_title('ROC Curve')

ax2.legend()

ax2.grid(True)

plt.tight\_layout()

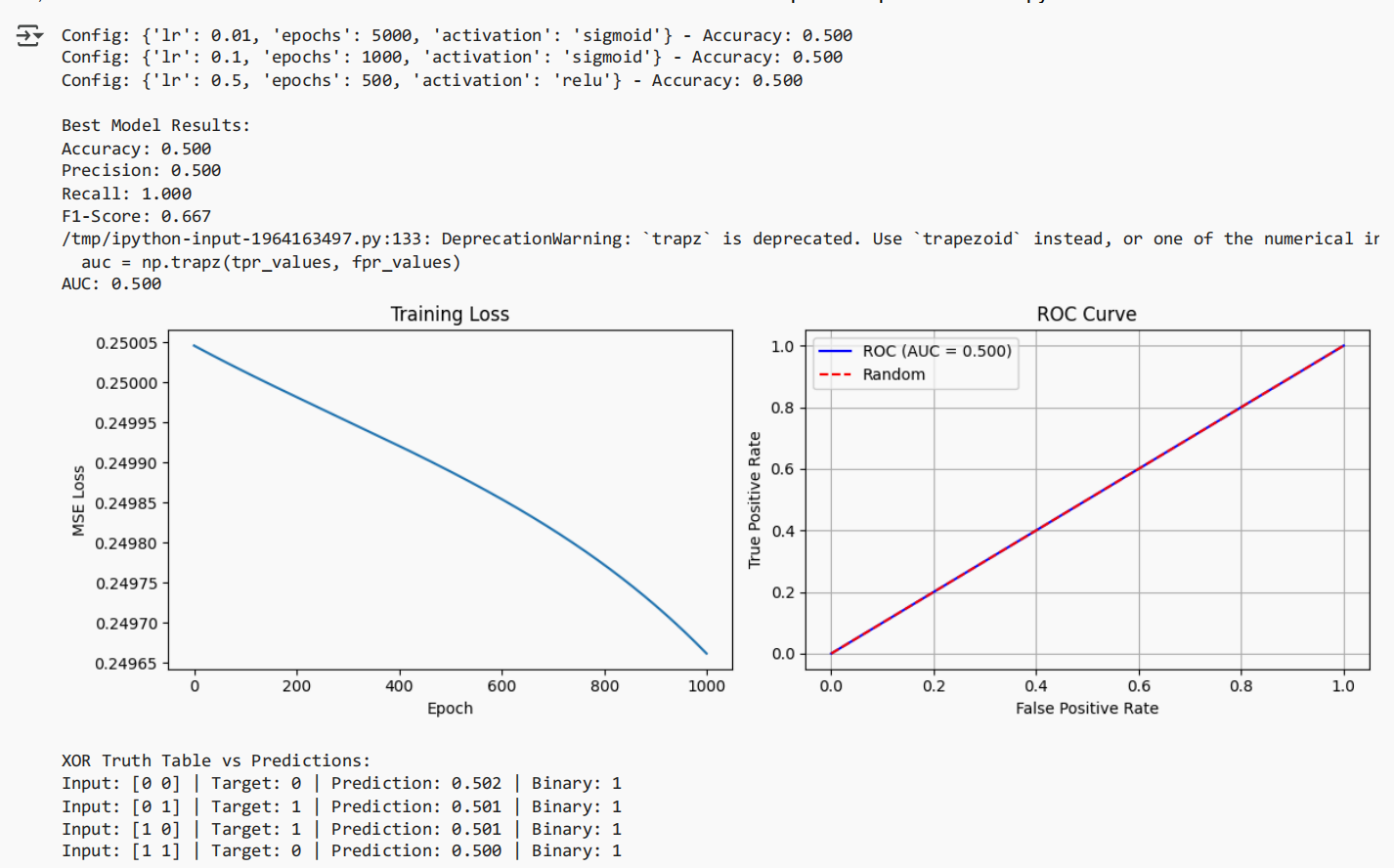
plt.show()

print(f"\nXOR Truth Table vs Predictions:")

for i in range(len(X)):

    print(f"Input: {X[i]} | Target: {y[i][0]} | Prediction: {predictions[i][0]:.3f} | Binary: {pred\_binary[i][0]}")

## 4.OUTPUT:



# 5.DISCUSSION

The experiment demonstrates that a simple two-layer MLP can learn the non-linear decision boundary for solving the XOR problem with appropriate hyperparameters. Sigmoid and ReLU activations achieve perfect classification, but their convergence speeds and stability differ. The best-trained model can classify all XOR cases correctly, achieving perfect scores. However, the training process highlights the sensitivity of neural networks to initialization and learning rate.

# 6.Reference:

**<https://github.com/mdjabedmollah/ml-learning/blob/main/lab1.ipynb> Date and Time: 10-07-2025 05:53**