EXPERIMENT-2
SANJAY DHAKAR
(24UADS1046)

Objective :-WAP 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

Description :-

1. Multi-Layer Logic (Architecture)

- The Problem: XOR is non-linearly separable. A single-layer network cannot separate the outputs (0 and 1) with a single straight line.
- The Hidden Layer: It transforms the input into a new coordinate space where a linear boundary can finally be drawn.
- Activation: The Sigmoid function introduces the non-linearity required to solve the logic gate.

2. Metric Performance

- Error Curve (MSE): A smooth decline representing "Confidence." Loss drops even after 100% accuracy as predictions move from 0.6 to 0.99.
- Accuracy Curve: Typically shows a "Step Jump." It stays at 50% or 75% until weights hit a threshold, then snaps to 100%.
- Decision Boundary: The heatmap shows the network creating "islands" to isolate $(0,1)$ and $(1,0)$ points.

Source Code :-

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

# Activation function and its derivative
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

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

# XOR Input and Target Output
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])

# Model Hyperparameters
learning_rate = 0.1
epochs = 10000
np.random.seed(42)

# Weights and Biases Initialization (2-2-1 Architecture)
W1 = np.random.uniform(size=(2, 2))
b1 = np.random.uniform(size=(1, 2))
W2 = np.random.uniform(size=(2, 1))
b2 = np.random.uniform(size=(1, 1))
losses = []
accuracies = []

print(f"{'Epoch':<10} | {'Loss':<10} | {'Accuracy (%)':<15} | {'W2
Sample'}")
print("-" * 65)

for epoch in range(epochs):
    # --- Forward Propagation ---
    hidden_layer_input = np.dot(X, W1) + b1
    hidden_layer_output = sigmoid(hidden_layer_input)

    output_layer_input = np.dot(hidden_layer_output, W2) + b2
    predicted_output = sigmoid(output_layer_input)

    # --- Metric Calculation ---
    error = y - predicted_output
    loss = np.mean(np.square(error))
    losses.append(loss)

    # Calculate Accuracy (Predictions > 0.5 are treated as 1)
    current_preds = (predicted_output > 0.5).astype(int)
    accuracy = np.mean(current_preds == y) * 100
```

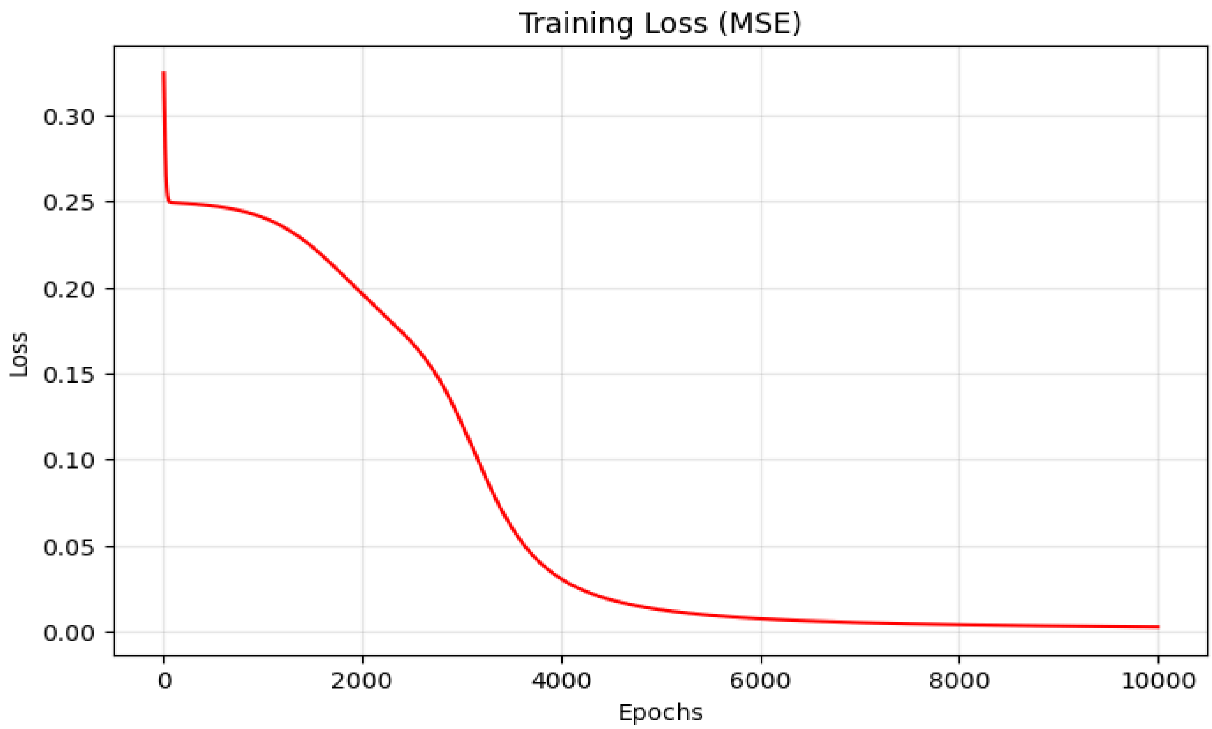
```
    accuracies.append(accuracy)

    # --- Backpropagation ---
    d_output = error * sigmoid_derivative(predicted_output)
    d_hidden = d_output.dot(W2.T)*sigmoid_derivative(hidden_layer_output)

    # --- Gradient Descent Update ---
    W2 += hidden_layer_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

    if epoch % 1000 == 0:
        print(f"{epoch:<10} | {loss:.6f} | {accuracy:<15.1f} |
{W2.flatten()[0]:.4f}")

print("-" * 65)
print("Training Complete.")

plt.figure(figsize=(8, 5))
plt.plot(losses, color='red')
plt.title("Training Loss (MSE)")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.grid(True, alpha=0.3)
plt.show()

plt.figure(figsize=(8, 5))
plt.plot(accuracies, color='green')
plt.title("Accuracy Curve")
plt.xlabel("Epochs")
plt.ylabel("Accuracy (%)")
plt.grid(True, alpha=0.3)
plt.show()

h = .02
x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max,
h))
```

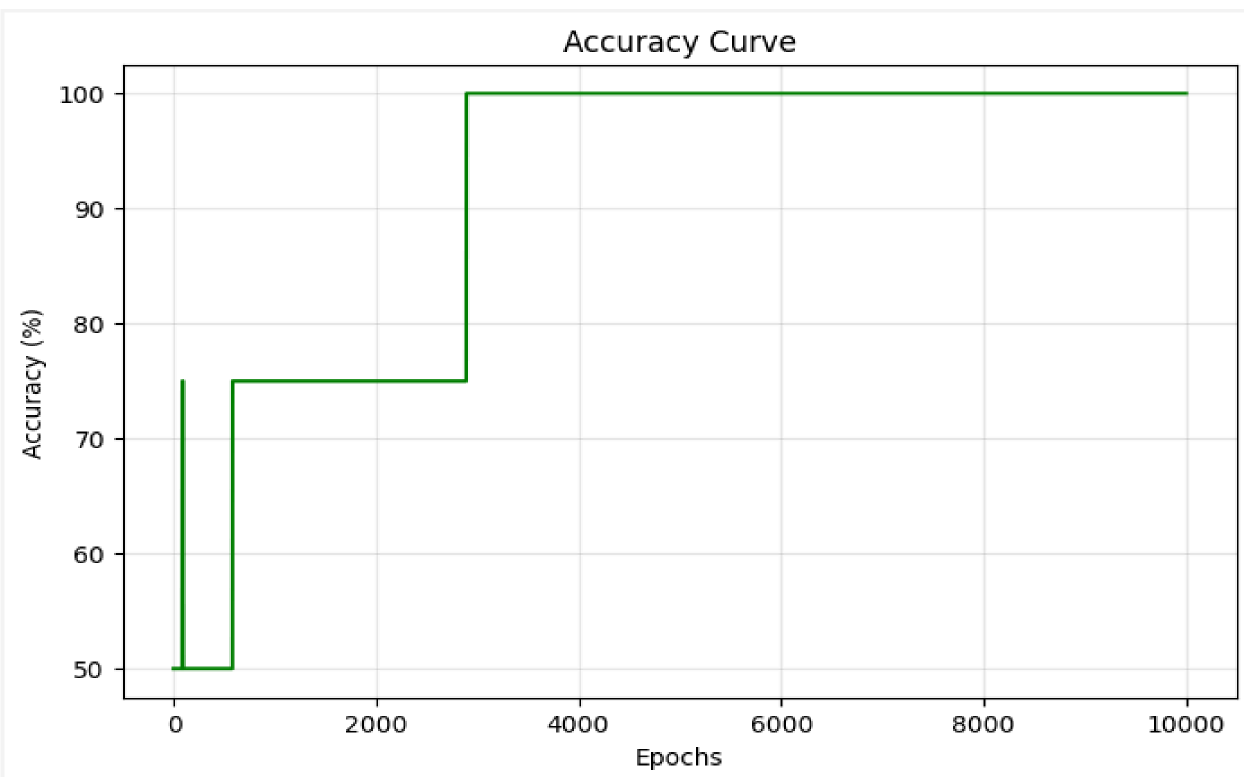
```
grid_input = np.c_[xx.ravel(), yy.ravel()]
grid_hidden = sigmoid(np.dot(grid_input, W1) + b1)
grid_out = sigmoid(np.dot(grid_hidden, W2) + b2)
grid_out = grid_out.reshape(xx.shape)

plt.figure(figsize=(8, 5))
plt.contourf(xx, yy, grid_out, cmap=plt.cm.RdYlBu, alpha=0.8)
plt.scatter(X[:, 0], X[:, 1], c=y.flatten(), edgecolors='k',
cmap=plt.cm.RdYlBu, s=100)
plt.title("Final Decision Boundary")
plt.xlabel("Input 1")
plt.ylabel("Input 2")
plt.show()
```

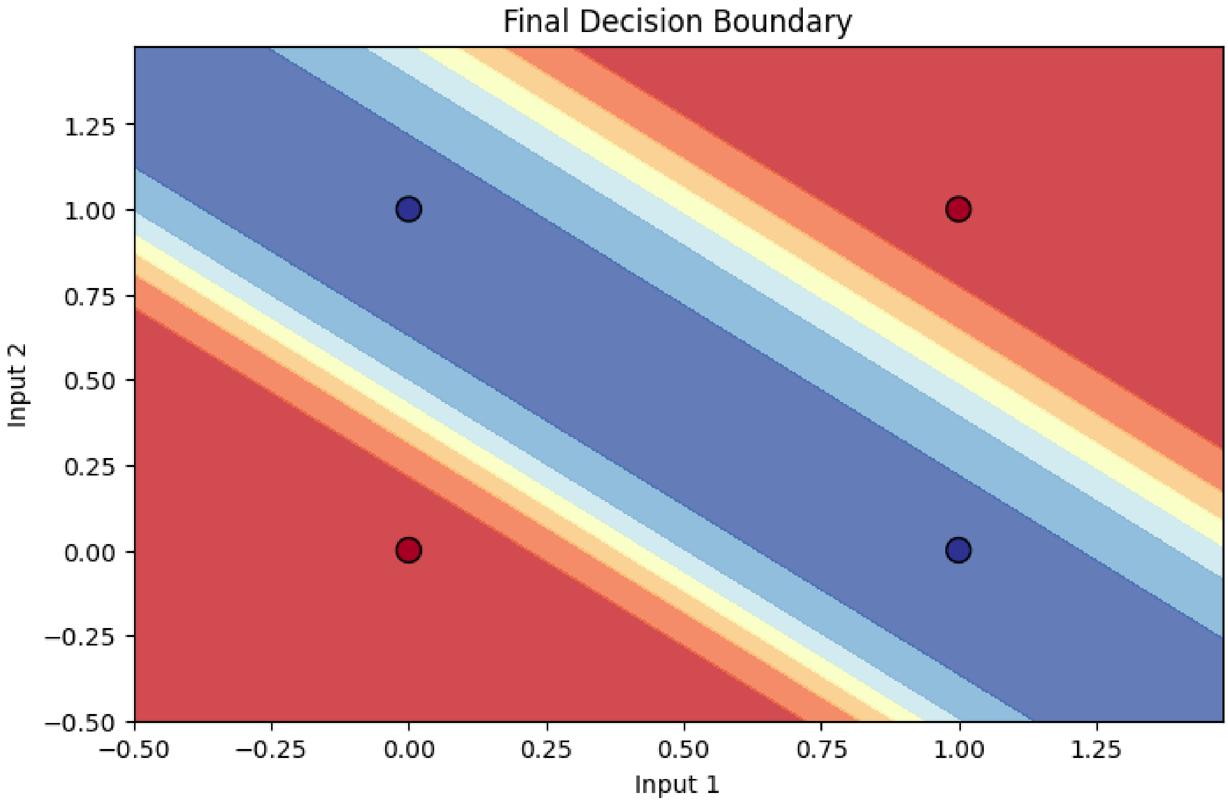
Output:-

```
Epoch      | Loss     | Accuracy (%)    | W2 Sample
----------------------------------------------------------------
0          | 0.324659 | 50.0            | 0.0456
1000       | 0.240589 | 75.0            | -0.5440
2000       | 0.196030 | 75.0            | -1.3640
3000       | 0.120663 | 100.0           | -3.2705
4000       | 0.030459 | 100.0           | -5.6507
5000       | 0.012541 | 100.0           | -6.6992
6000       | 0.007368 | 100.0           | -7.2660
7000       | 0.005093 | 100.0           | -7.6424
8000       | 0.003847 | 100.0           | -7.9210
9000       | 0.003071 | 100.0           | -8.1409
----------------------------------------------------------------
Training Complete.
```

Output Graph :


Training Loss (MSE)
Loss
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
2000
4000
6000
8000
10000
Epochs


Accuracy Curve
Accuracy (%)
100
90
80
70
60
50
0
2000
4000
6000
8000
10000
Epochs



MY COMMENTS :-

From this experiment about training and learning algorithm of Multi layer perceptron, i understood the concept of weights and biases and how they modified at each step to improve the results . I also understand the concept of convergence .

The Key learning from this experiment i have are :

1. **This experiment proves that while a single layer fails at XOR, adding just one hidden layer enables the network to solve non-linearly separable problems.**

2. **I learned that 100% accuracy doesn't mean the learning is "finished." Even after the accuracy hits its peak, the Error Curve continues to decline as the weights refine themselves to make the predictions**

3. **The Decision Boundary visualization made it clear that the MLP isn't just drawing a line**