**Experiment 2 Description**

**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.

**Overview:** This experiment implements a multi-layer perceptron (MLP) to solve the XOR problem using a manually designed neural network without backpropagation or gradient descent.

**Description of the model:** It takes inputs- weights and bias and calculates the weighted sum. Uses step function as activation function.

**Python Implementation:**

import numpy as np

import matplotlib.pyplot as plt

def perceptron(inputs, weights, bias):

return np.where(np.dot(inputs, weights) + bias > 0, 1, 0)

def xor\_mlp(x):

# Generate weights programmatically for hidden layer

hidden\_weights = np.array([[1, -1], [-1, 1], [1, 1], [-1, -1]])

hidden\_biases = np.array([0, 0, -0.5, 0.5])

# Compute hidden layer outputs

hidden\_output = np.array([perceptron(x, w, b) for w, b in zip(hidden\_weights, hidden\_biases)]).T

# XOR perceptron (using first two perceptrons)

xor\_weights = np.array([1, 1, 0, 0])

xor\_bias = 0

# Compute final XOR output

return perceptron(hidden\_output, xor\_weights, xor\_bias)

# XOR truth table inputs

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

Y\_true = np.array([0, 1, 1, 0]) # Expected XOR outputs

Y\_pred = xor\_mlp(X)

# Print results

for i in range(len(X)):

print(f'Input: {X[i]} -> XOR Output: {Y\_pred[i]}')

# Calculate accuracy

accuracy = np.mean(Y\_pred == Y\_true) \* 100

print(f'Accuracy: {accuracy:.2f}%')

# Plot decision boundary

def plot\_decision\_boundary():

x\_min, x\_max = -0.5, 1.5

y\_min, y\_max = -0.5, 1.5

xx, yy = np.meshgrid(np.linspace(x\_min, x\_max, 100), np.linspace(y\_min, y\_max, 100))

grid = np.c\_[xx.ravel(), yy.ravel()]

predictions = xor\_mlp(grid).reshape(xx.shape)

plt.contourf(xx, yy, predictions, alpha=0.6)

plt.scatter(X[:, 0], X[:, 1], c=Y\_true, edgecolors='k', cmap=plt.cm.Paired)

plt.xlabel('X1')

plt.ylabel('X2')

plt.title('XOR Decision Boundary')

plt.show()

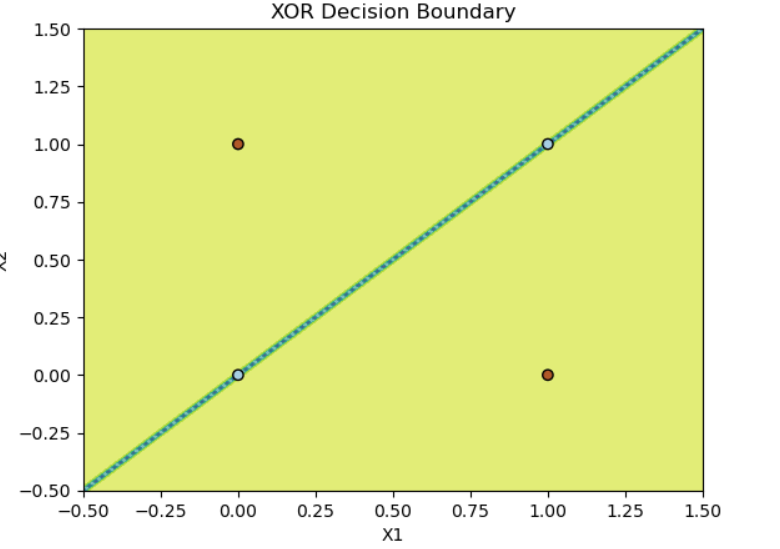
plot\_decision\_boundary()

**Description of code:**

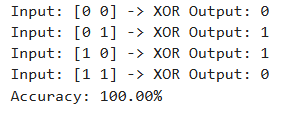
* xor\_mlp(x) function defines a simple **2-layer neural network** to compute XOR.
* Hidden layer has 4 perceptrons with weights and biases
* Hidden layer outputs are computed by applying the perceptron function.
* First two hidden perceptrons contribute to the XOR output.
* Computes the final XOR output using the hidden layer results.

**Performance Evaluation:**

**-Accuracy:** 100.00%



**Output:**

s

**Comments:**

* To improve performance Sigmoid or ReLU fuction can be used.
* Instead of manually defining perceptrons TensorFlow or PyTorch can be used.