**DL Experiment 1**

**Khushali Kaku(25)**

import numpy as np

def sigmoid(x):

return 1 / (1 + np.exp(-x))

def sigmoid\_derivative(x):

return x \* (1 - x)

class XORPerceptron:

def \_\_init\_\_(self, input\_dim):

self.input\_dim = input\_dim

self.weights = np.random.rand(input\_dim)

self.bias = np.random.rand()

def predict(self, inputs):

return sigmoid(np.dot(inputs, self.weights) + self.bias)

def train(self, inputs, labels, learning\_rate=0.1, epochs=10000):

for epoch in range(epochs):

for input\_data, label in zip(inputs, labels):

prediction = self.predict(input\_data)

error = label - prediction

adjustment = learning\_rate \* error \* sigmoid\_derivative(prediction)

self.weights += adjustment \* input\_data

self.bias += adjustment

if \_\_name\_\_ == "\_\_main\_\_":

# XOR data and labels

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

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

# Create and train the XOR perceptron

xor\_perceptron = XORPerceptron(input\_dim=2)

xor\_perceptron.train(inputs, labels)

# Test the perceptron

print("Input: [0, 0], Predicted Output:", xor\_perceptron.predict([0, 0]))

print("Input: [0, 1], Predicted Output:", xor\_perceptron.predict([0, 1]))

print("Input: [1, 0], Predicted Output:", xor\_perceptron.predict([1, 0]))

print("Input: [1, 1], Predicted Output:", xor\_perceptron.predict([1, 1]))

Output -

