**Lab Assignment No. 6**

**Code:**

import numpy as np  
  
class NeuralNetwork:  
 def \_\_init\_\_(self, layer\_sizes):  
 self.layer\_sizes = layer\_sizes  
 self.weights = [np.random.randn(y, x) for x, y in zip(layer\_sizes[:-1], layer\_sizes[1:])]  
 self.biases = [np.random.randn(y, 1) for y in layer\_sizes[1:]]  
  
 def sigmoid(self, z):  
 return 1 / (1 + np.exp(-z))  
  
 def forward\_propagation(self, X):  
 activation = X  
 for w, b in zip(self.weights, self.biases):  
 z = np.dot(w, activation) + b  
 activation = self.sigmoid(z)  
 return activation  
  
 def sigmoid\_derivative(self, z):  
 return self.sigmoid(z) \* (1 - self.sigmoid(z))  
  
 def backpropagation(self, X, y):  
 m = X.shape[1]  
 delta\_weights = [np.zeros(w.shape) for w in self.weights]  
 delta\_biases = [np.zeros(b.shape) for b in self.biases]  
 # Forward propagation  
 activation = X  
 activations = [activation]  
 zs = []  
 for w, b in zip(self.weights, self.biases):  
 z = np.dot(w, activation) + b  
 zs.append(z)  
 activation = self.sigmoid(z)  
 activations.append(activation)  
  
 # Backpropagation  
 delta = (activations[-1] - y) \* self.sigmoid\_derivative(zs[-1])  
 delta\_weights[-1] = np.dot(delta, activations[-2].T)  
 delta\_biases[-1] = np.sum(delta, axis=1, keepdims=True)  
 for l in range(2, len(self.layer\_sizes)):  
 delta = np.dot(self.weights[-l+1].T, delta) \* self.sigmoid\_derivative(zs[-l])  
 delta\_weights[-l] = np.dot(delta, activations[-l-1].T)  
 delta\_biases[-l] = np.sum(delta, axis=1, keepdims=True)  
  
 return delta\_weights, delta\_biases  
  
 def train(self, X, y, num\_epochs, learning\_rate):  
 m = X.shape[1]  
 for epoch in range(num\_epochs):  
 delta\_weights, delta\_biases = self.backpropagation(X, y)  
 self.weights = [w - (learning\_rate / m) \* dw for w, dw in zip(self.weights, delta\_weights)]  
 self.biases = [b - (learning\_rate / m) \* db for b, db in zip(self.biases, delta\_biases)]  
  
 def predict(self, X):  
 return self.forward\_propagation(X)

if \_\_name\_\_ == '\_\_main\_\_':  
 layer\_sizes = [2, 4, 1] # Input layer: 2 neurons, Hidden layer: 4 neurons, Output layer: 1 neuron  
 nn = NeuralNetwork(layer\_sizes)  
  
 # Training data  
 X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]]).T  
 y = np.array([[0, 1, 1, 0]])  
  
 # Train the neural network  
 num\_epochs = 10000  
 learning\_rate = 0.1  
 nn.train(X, y, num\_epochs, learning\_rate)  
   
 predictions = nn.predict(X)  
 print("Predictions:")  
 print(predictions)

**Output:**

Predictions:  
[[0.23317429 0.62300796 0.69296428 0.49537369]]

X

array([[0, 0, 1, 1],  
 [0, 1, 0, 1]])

X[0]

array([0, 0, 1, 1])

y[0]

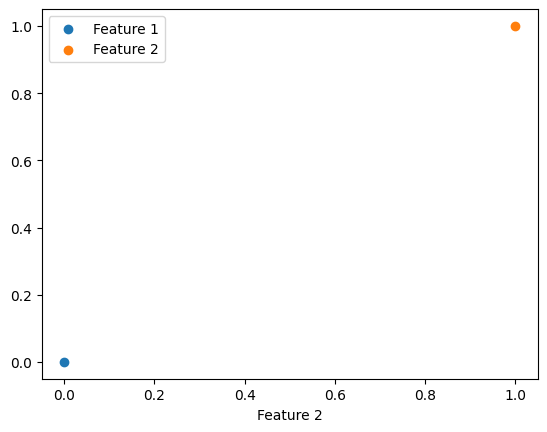
array([0, 1, 1, 0])

x = [(x, y) for x, y in zip(X[:,0], X[:,1])]  
x

[(0, 0), (0, 1)]

x0 = [i for i in X[0] if i == 0]  
x1 = [i for i in X[1] if i == 1]

import matplotlib.pyplot as plt  
plt.scatter(x0[0], x0[1], label="Feature 1")  
plt.scatter(x1[0], x1[1], label="Feature 2")  
plt.xlabel("Feature 1")  
plt.xlabel("Feature 2")  
plt.legend()  
plt.show()



final\_preditions = [1 if predict >= 0.5 else 0 for predict in predictions[0]]

final\_preditions

[0, 1, 1, 0]