**Name: Abdul Moiz Asif**

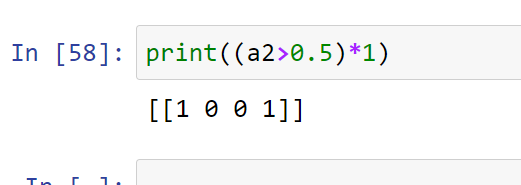
**CMS: 263802**

**Ai lab#5**

**Task#1**

# importing the python libraries  
import numpy as np  
  
# declaring the sigmoid function  
def sigmoid(z):  
 return 1/ (1 + np.exp(-z))  
  
def Initializeparameters(HiddenLayerNeurons, InputLayerNeurons, OuputLayerNeurons):  
 w1 = np.random.randn(HiddenLayerNeurons, InputLayerNeurons)  
 w2 = np.random.randn(OuputLayerNeurons, HiddenLayerNeurons)  
  
 b1 = np.zeros((HiddenLayerNeurons, 1))  
 b2 = np.zeros((OuputLayerNeurons, 1))  
 parameters = {"w1": w1, "w2": w2, "b1": b1, "b2": b2}  
 return parameters  
  
def ForwardPropagation(parameters, x, y):  
 m = x.shape[1]  
 w1 = parameters["w1"]  
 w2 = parameters["w2"]  
 b1 = parameters["b1"]  
 b2 = parameters["b2"]  
  
 z1 = np.dot(w1, x) + b1  
 a1 = sigmoid(z1)  
 z2 = np.dot(w2, a1) + b2  
 a2 = sigmoid(z2)  
 cache = (z1, a1, w1, b1, z2, a2, w2, b2)  
 loss = -np.sum(np.multiply(np.log(a2), y) + np.multiply(np.log(1 - a2), (1 - y))) / m  
 return cache, loss, a2  
  
def backwardPropogation(cache, x, y):  
 (z1, a1, w1, b1, z2, a2, w2, b2) = cache  
 m = x.shape[1]  
 dz2 = a2 - y  
 dw2 = np.dot(dz2, a1.T) / m  
 db2 = np.sum(dz2, axis=1, keepdims=True)  
  
 da1 = np.dot(w2.T, dz2)  
 dz1 = np.multiply(da1, a1 \* (1 - a1))  
 dw1 = np.dot(dz1, x.T) / m  
 db1 = np.sum(dz1, axis=1, keepdims=True) / m  
 gradients = {"dz1": dz1, "dw1": dw1, "db1": db1, "da1": da1, "dz2": dz2,  
 "dw2": dw2, "db2": db2}  
 return gradients  
  
  
def UpdateWeights(parameters, gradients, LR):  
 parameters["w1"] = parameters["w1"] - gradients["dw1"] \* LR  
 parameters["w2"] = parameters["w2"] - gradients["dw2"] \* LR  
 parameters["b1"] = parameters["b1"] - gradients["db1"] \* LR  
 parameters["b2"] = parameters["b2"] - gradients["db2"] \* LR  
 return parameters  
  
  
# x,y for training the model  
x = np.array([[0, 0, 1, 1], [0, 1, 0, 1]]) # XOR input  
y = np.array([[0, 1, 1, 0]]) # XOR output  
HiddenLayerNeurons = 2  
InputLayerNeurons = x.shape[0]  
OuputLayerNeurons = y.shape[0]  
epochs = 100000  
LearningRate = 0.01  
parameters = Initializeparameters(HiddenLayerNeurons, InputLayerNeurons, OuputLayerNeurons)  
for i in range(epochs):  
 cache, loss, a2 = ForwardPropagation(parameters, x, y)  
 gradients = backwardPropogation(cache, x, y)  
 parameters = UpdateWeights(parameters, gradients, LearningRate)  
 loss  
  
x = np.array([[1, 1, 0, 0], [0, 1, 0, 1]])  
\_, loss, a2 = ForwardPropagation(parameters, x, y)  
a2  
  
print((a2 > 0.5) \* 1)

**Output:**

****

**Task#2**

import pandas as pd

import csv

from sklearn.neural\_network import MLPClassifier

#Working on TrainingSet

#reading all file

train\_data = pd.read\_csv("TrainingSet.csv")

#extracting first 4 columns as features

features = train\_data.iloc[:,0:4].values

#column of all plants in traning data

labels = train\_data.iloc[:,4].values

clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden\_layer\_sizes=(100,), random\_state=1)

#fiiting our trainig data into the model

clf.fit(features, labels)

#Working on TestSet1

test\_data = pd.read\_csv("TestSet1.csv")

test\_features= test\_data.iloc[:,0:4].values

#Performming predictions on samples in test\_features

test\_labels = clf.predict(test\_features)

#writing predictions to a csv file

testData = pd.DataFrame(test\_labels)

testData.to\_csv("results.csv", index=False, header=None)

