import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)

y = np.array(([92], [86], [89]), dtype=float)

X = X/np.amax(X,axis=0)

y = y/100

def sigmoid (x):

    return 1/(1 + np.exp(-x))

def derivatives\_sigmoid(x):

    return x \* (1 - x)

epoch=5

lr=0.1

inputlayer\_neurons = 2 #number of features in data set

hiddenlayer\_neurons = 3 #number of hidden layers neurons

output\_neurons = 1 #number of neurons at output layer

#weight and bias initialization

wh=np.random.uniform(size=(inputlayer\_neurons,hiddenlayer\_neurons)) # random samples from uniform distribution.2\*3 , weights from ip layer to hidden layer

bh=np.random.uniform(size=(1,hiddenlayer\_neurons)) # bias from ip layer to hidden layer

wout=np.random.uniform(size=(hiddenlayer\_neurons,output\_neurons)) # weight from hidden to output

bout=np.random.uniform(size=(1,output\_neurons)) #bias from hidden to output.

#draws a random range of numbers uniformly of dim x\*y

for i in range(epoch):

    #Forward Propogation

    hinp1=np.dot(X,wh)

    hinp=hinp1 + bh

    hlayer\_act = sigmoid(hinp)

    outinp1=np.dot(hlayer\_act,wout)

    outinp= outinp1+bout

    output = sigmoid(outinp)

    #Backpropagation

    EO = y-output #target-obrerved op

    outgrad = derivatives\_sigmoid(output)

    d\_output = EO \* outgrad

    EH = d\_output.dot(wout.T)

    hiddengrad = derivatives\_sigmoid(hlayer\_act)#how much hidden layer wts contributed to error

    d\_hiddenlayer = EH \* hiddengrad

    wout += hlayer\_act.T.dot(d\_output) \*lr   # dotproduct of nextlayererror and currentlayerop

    wh += X.T.dot(d\_hiddenlayer) \*lr

    print ("-----------Epoch-", i+1, "Starts----------")

    print("Input: \n" + str(X))

    print("Actual Output: \n" + str(y))

    print("Predicted Output: \n" ,output)

    print ("-----------Epoch-", i+1, "Ends----------\n")

print("Input: \n" + str(X))

print("Actual Output: \n" + str(y))

print("Predicted Output: \n" ,output)