**Name: YATHIN K V**

**USN: 1PE17IS105**

**Class: ISE 7B**

**Machine Learning Laboratory**

Program 4

**Build an Artificial Neural Network by implementing the Back propagation Algorithm and test the same using appropriate data sets.**

**Back propagation Algorithm:**

1.Load data set

2. Assign all network inputs and output

3.Initialize all weights with small random numbers, typically between -1 and 1

repeat

for every pattern in the training set

Present the pattern to the network

for each layer in the network

for every node in the layer

1. Calculate the weight sum of the inputs to the node

2. Add the threshold to the sum

3. Calculate the activation for the node

end

end

// Propagate the errors backward through the network

for every node in the output layer

calculate the error signal

end

for all hidden layers

for every node in the layer

1. Calculate the node's signal

2. Update each node's weight in the network

end

end

// Calculate Global Error

Calculate the Error Function

end

while ((maximum number of iterations < than specified) AND (Error Function is > than specified))

• Input layer with two inputs neurons

• One hidden layer with two neurons

• Output layer with a single neuron

**PROGRAM:**

import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)# featues (hrs slept/hrs studied)

y = np.array(([92], [86], [89]), dtype=float)#label marks obtained

X = X/np.amax(X,axis=0) # maximum of X array longitudinally used to normalize

y = y/100

#Sigmoid Function to transfer neuron activation during forward propagation

def sigmoid (x):

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

#Derivative of Sigmoid Function to Calculate the derivative of an neuron output

def derivatives\_sigmoid(x):

return x \* (1 - x)

#intialize network

epoch=7000 #Setting training iterations

lr=0.2 #Setting learning rate

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))

bh=np.random.uniform(size=(1,hiddenlayer\_neurons))

wout=np.random.uniform(size=(hiddenlayer\_neurons,output\_neurons))

bout=np.random.uniform(size=(1,output\_neurons))

#Forward Propagation

for i in range(epoch):

hinp=np.dot(X,wh)+bh # this function returns a dot product of two arrays

hlayer\_act = sigmoid(hinp)

outinp=np.dot(hlayer\_act,wout)+bout

output = sigmoid(outinp)

#Backpropagation

#Error at output layer

EO = y-output # error at the output

outgrad = derivatives\_sigmoid(output)

d\_output = EO\* outgrad

#error at hidden layer

EH = d\_output.dot(wout.T) #transpose

hiddengrad = derivatives\_sigmoid(hlayer\_act)

#how much hidden layer wts contributed to error

d\_hiddenlayer = EH \* hiddengrad

#weight update

wout += hlayer\_act.T.dot(d\_output) \*lr

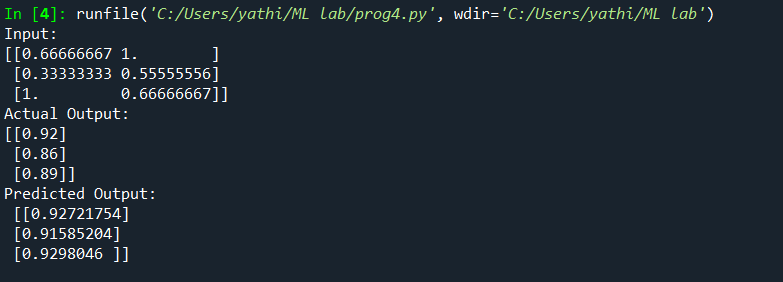
wh += X.T.dot(d\_hiddenlayer) \*lr

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

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

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

**OUTPUT:**

****