**Aim:** Write a program to implement Back Propagation Network.

**Code:**

importnumpy as np

x1 = float(input("Enter X1: "))

print(x1)

x2 = float(input("Enter X2: "))

print(x2)

b1 = float(input("Enter bias 1: "))

b2 = float(input("Enter bias 2: "))

b3 = float(input("Enter bias 3: "))

alpha = float(input("Enter alpha: "))

t = float(input("Enter target: "))

a = [0.6,0.3,-0.1,-0.3,0.4,0.5,0.4,0.1,-0.2]

print('phase 1')

zin1 = float(b1\*a[1]+x1\*a[0]+x2\*a[2])

print('zin1=',zin1)

zp1 = 1/(1+np.exp(-zin1))

print('z1=',zp1)

fzin1= zp1\*(1-zp1)

print('fzin1=',fzin1)

zin2= float(a[3]\*x1+a[4]\*x2+a[5]\*b2)

print('zin2=',zin2)

zp2 = 1/(1+np.exp(-zin2))

print('z2=',zp2)

fzin2= zp2\*(1-zp2)

print('fzin2=',fzin2)

yin=float(zp1\*a[6]+zp2\*a[7]+b3\*a[8])

print('yin=',yin)

y = 1/(1+np.exp(-yin))

print('y=',y)

fyin= y\*(1-y)

print('fyin=',fyin)

print('phase 2')

dell1=(t-y)\*fyin

print('dell1=',dell1)

delta\_w11=alpha\*dell1\*zp1

print('delta\_w11=',delta\_w11)

delta\_w21=alpha\*dell1\*zp2

print('delta\_w21=',delta\_w21)

dellin1=dell1\*a[6]

print('dellin1=',dellin1)

dellin2 = dell1\*a[7]

print('dellin2=',dellin2)

delta1=dellin1\*fzin1

print('delta1=',delta1)

delta2=dellin2\*fzin2

print('delta2=',delta2)

delta\_w01=alpha\*dell1

print('delta\_w01=',delta\_w01)

print('phase 3')

delta\_v11=alpha\*delta1\*x1

print('delta\_v11=',delta\_v11)

delta\_v12=alpha\*delta2\*x1

print('delta\_v12=',delta\_v12)

delta\_v21=alpha\*delta1\*x2

print('delta\_v21=',delta\_v21)

delta\_v22=alpha\*delta2\*x2

print('delta\_v22=',delta\_v22)

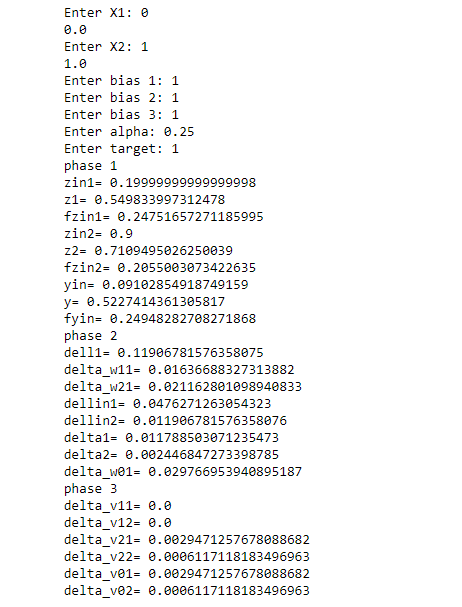
delta\_v01 = alpha\*delta1

print('delta\_v01=',delta\_v01)

delta\_v02 = alpha\*delta2

print('delta\_v02=',delta\_v02)

**Output:**



**AIM:**Write a program in Python to implement Back-Proagation Neural Network.

**CODE:**

import math

import random

import string

class NN:

def \_\_init\_\_(self, NI, NH, NO):

    # number of nodes in layers

    self.ni = NI + 1 # +1 for bias

self.nh = NH

    self.no = NO

    self.ai, self.ah, self.ao = [],[], []

    self.ai = [1.0]\*self.ni

self.ah = [1.0]\*self.nh

    self.ao = [1.0]\*self.no

self.wi = makeMatrix (self.ni, self.nh)

self.wo = makeMatrix (self.nh, self.no)

    # initialize node weights to random vals

randomizeMatrix ( self.wi, -0.2, 0.2 )

randomizeMatrix ( self.wo, -2.0, 2.0 )

    self.ci = makeMatrix (self.ni, self.nh)

    self.co = makeMatrix (self.nh, self.no)

defrunNN (self, inputs):

iflen(inputs) != self.ni-1:

print('incorrect number of inputs')

for i in range(self.ni-1):

      self.ai[i] = inputs[i]

for j in range(self.nh):

sum = 0.0

for i in range(self.ni):

sum +=( self.ai[i] \* self.wi[i][j] )

self.ah[j] = sigmoid (sum)

for k in range(self.no):

sum = 0.0

for j in range(self.nh):

sum +=( self.ah[j] \* self.wo[j][k] )

       self.ao[k] = sigmoid (sum)

return self.ao

defbackPropagate (self, targets, N, M):

output\_deltas = [0.0] \* self.no

for k in range(self.no):

error = targets[k] - self.ao[k]

output\_deltas[k] =  error \* dsigmoid(self.ao[k])

for j in range(self.nh):

for k in range(self.no):

change = output\_deltas[k] \* self.ah[j]

self.wo[j][k] += N\*change + M\*self.co[j][k]

         self.co[j][k] = change

hidden\_deltas = [0.0] \* self.nh

for j in range(self.nh):

error = 0.0

for k in range(self.no):

error += output\_deltas[k] \* self.wo[j][k]

hidden\_deltas[j] = error \* dsigmoid(self.ah[j])

for i in range (self.ni):

for j in range (self.nh):

change= hidden\_deltas[j] \* self.ai[i]

self.wi[i][j] += N\*change + M\*self.ci[i][j]

         self.ci[i][j] = change

error = 0.0

for k in range(len(targets)):

error = 0.5 \* (targets[k]-self.ao[k])\*\*2

return error

def weights(self):

print('Input weights:')

for i in range(self.ni):

print (self.wi[i])

print()

print('Output weights:')

for j in range(self.nh):

print (self.wo[j])

print ('')

def test(self, patterns):

for p in patterns:

inputs = p[0]

print('Inputs:', p[0], '-->', self.runNN(inputs), '\tTarget', p[1])

def train (self, patterns, max\_iterations = 1000, N=0.5, M=0.1):

for i in range(max\_iterations):

for p in patterns:

inputs = p[0]

targets = p[1]

self.runNN(inputs)

error = self.backPropagate(targets, N, M)

if i % 50 == 0:

print('Combined error', error)

self.test(patterns)

def sigmoid (x):

returnmath.tanh(x)

defdsigmoid (y):

return 1 - y\*\*2

defmakeMatrix ( I, J, fill=0.0):

  m = []

for i in range(I):

m.append([fill]\*J)

return m

defrandomizeMatrix ( matrix, a, b):

for i in range ( len (matrix) ):

for j in range ( len (matrix[0]) ):

matrix[i][j] = random.uniform(a,b)

def main ():

pat = [

      [[0,0], [1]],

      [[0,1], [1]],

      [[1,0], [1]],

      [[1,1], [0]]

  ]

myNN = NN ( 2, 2, 1)

myNN.train(pat)

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

main()

**OUTPUT:**

