**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:**

