

## PROGRAM 13

**AIM:** Write a program in Python to implement Back-Propagation 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
```

## PROGRAM 13

```
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):
    return math.tanh(x)
def dsigmoid(y):
    return 1 - y**2
def makeMatrix(I, J, fill=0.0):
    m = []
    for i in range(I):
        m.append([fill]*J)
    return m
def randomizeMatrix(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]],
```

## PROGRAM 13

```

[[1,1], [0]]
]
myNN = NN ( 2, 2, 1)
myNN.train(pat)
if __name__ == "__main__":
main()

```

## OUTPUT:

The screenshot displays the Spyder Python IDE interface. The main editor window shows the file `back.py` with the following Python code:

```

55 # we want to find the instantaneous rate of change of ( error with respect to weight
56 # output_delta is defined as an attribute of each output node. It is not the final
57 # To get the final rate we must multiply the delta by the activation of the hidden
58 # This multiplication is done according to the chain rule as we are taking the derivative
59 # of the output node.
60 # dE/dw[j][k] = (t[k] - ao[k]) * s'( SUM( w[j][k]*ah[j] ) ) * ah[j]
61 output_deltas = [0.0] * self.no
62 for k in range(self.no):
63     error = targets[k] - self.ao[k]
64     output_deltas[k] = error * dsigmoid(self.ao[k])
65
66 # update output weights
67 for j in range(self.nh):
68     for k in range(self.no):
69         # output_deltas[k] * self.ah[j] is the full derivative of dError/dweight[j][k]
70         change = output_deltas[k] * self.ah[j]
71         self.wo[j][k] += N*change + M*self.co[j][k]
72         self.co[j][k] = change
73
74 # calc hidden deltas
75 hidden_deltas = [0.0] * self.nh
76 for j in range(self.nh):
77     error = 0.0
78     for k in range(self.no):
79         error += output_deltas[k] * self.wo[j][k]
80     hidden_deltas[j] = error * dsigmoid(self.ah[j])
81
82 #update input weights
83 for i in range(self.ni):
84     for j in range(self.nh):
85         change = hidden_deltas[j] * self.ai[i]
86         #print 'activation',self.ai[i], 'synapse',i,j, 'change',change
87         self.wi[i][j] += N*change + M*self.ci[i][j]
88         self.ci[i][j] = change
89
90 # calc combined error
91 # 1/2 for differential convenience & **2 for modulus
92 error = 0.0
93 for k in range(len(targets)):
94     error = 0.5 * (targets[k]-self.ao[k])**2
95 return error
96
97
98 def weights(self):
99     print('Input weights:')
100     for i in range(self.ni):
101         print (self.wi[i])
102     print()
103     print('Output weights:')
104     for j in range(self.nh):
105         print (self.wo[j])
106     print ('')
107
108 def test(self, patterns):
109     for n in patterns:

```

The IPython console on the right shows the output of running the script:

```

In [2]:
In [2]: runfile('C:/Users/admin/.spyder-py3/back.py', wdir='C:/Users/admin/.spyder-py3')
Combined error 0.33960029597171876
Combined error 0.005804347782481797
Combined error 0.002601213307011534
Combined error 0.0016867154033319899
Combined error 0.0012383865063792165
Combined error 0.0009752734271580881
Combined error 0.0008027801807789076
Combined error 0.0006820177235108346
Combined error 0.0005932187653819327
Combined error 0.0005238373154143884
Combined error 0.00046870901031911995
Combined error 0.0004243707440499534
Combined error 0.00038706606095840503
Combined error 0.00035610581102335867
Combined error 0.0003293302229051458
Combined error 0.0003064937660825221
Combined error 0.0002863502338388085
Combined error 0.0002688362269292783
Combined error 0.0002531466916495717
Combined error 0.00023927819235108606
Inputs: [0, 0] --> [0.9991494359910482] Target [1]
Inputs: [0, 1] --> [0.9893815477575261] Target [1]
Inputs: [1, 0] --> [0.9893589815027886] Target [1]
Inputs: [1, 1] --> [-0.014869572264207365] Target [0]

```

The bottom status bar indicates: Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 162 Column: 11 Memory: 38 %