Experiment - 4

Build an Artificial Neural Network by implementing the Back propogation algorithm and test the same using appropriate dataset.

```
import random
from math import exp
from random import imporrd seed
def initialize network (n_input, n= hidden, n_output)
network = list()
hidden_layer = [{'weights': [random.uniform (-0.5, 0.5) for i in
                range (n_input + 1)]} for i in range (n_hidden)]
network.append (hidden layer)
output layer = [{'weights': [random.uniform (-0.5, 0.5) for i in range
              (n_hidden + 1)]} for i in range (n_outputs)]
network. append (output_layer)
i = 1
print ("\n the initialized Neural Network:\n")
for layers in network:
    j = 1
    for sub in layer:
        print("\n layer[%d] Node[%d]:\n"% (i,j),sub)
        j = j+1
        i = i+1
    return network
```

Number of inputs

2

Number of outputs

2

The initialised neural network

layer [1] [Node [1]]:

{ 'weights' [0.456034271889, 0.44787, -0.443448632]}

Layer [1] [Node [2]]:

{'weights': [-0.41512800484, 0.33549887812, 0.2359649890]

Layer [2] [Node [1]]:

{'weights': [0.16973040144, -0.19918635424, 0.105944166]}

Layer [2] [Node [2]]:

{'weigws' : [0.10680873, 0.0812040017, - 341617129 72]}

```
def activate (weights ,inputs):
  activation = weight [-1] :
     activation += weights [i]* inputs[i]
     return activation

def transfer (activation):
     return 1.0/ (1.0 + exp (- activation))

def forward_propogate (network, rows):
     input = row
     for layer in network :
      next input = [ ]
      for neuron in layer:
          activation = activate (neuron['weights'], inputs]
          return inputs

def transfer_derivative (output):
     return output* (1.0- output)

def backpropogate_error (network ,encepted) :
     for i in reversed (range (len (network)))):
          layer =network [i]
          errors = list()
          if i != len (network]-1 :
             for j in range (len (layer))] :
                error = 0.0
               for neuron in network [i+1]:
```

Network Training Begins:

```
epoch = 0     Irate = 0.500     error = 5.278
epoch = 1     Irate = 0.500     error = 5.122
epoch = 2     Irate = 0.500     error = 5.006
epoch = 3     Irate = 0.500     error = 4.875
epoch = 4     Irate = 0.500     error = 4.700
epoch = 5     Irate = 0.500     error = 4.966
epoch = 6     Irate = 0.500     error = 4.176
epoch = 7     Irate = 0.500     error = 3.838
epoch = 8     Irate = 0.500     error = 3.469
epoch = 10    Irate = 0.500     error = 3.084
epoch = 11    Irate = 0.500     error = 2.716
epoch = 12    Irate = 0.500     error = 2.367
epoch = 13    Irate = 0.500     error = 4.059
epoch = 14    Irate = 0.500     error = 1.780
epoch = 15    Irate = 0.500     error = 1.61
epoch = 16    Irate = 0.500     error = 1.546
epoch = 17    Irate = 0.500     error = 1.349
epoch = 18    Irate = 0.500     error = 1.045
epoch = 19    Irate = 0.500     error = 0.9229
```

```
            error += (neuron['weights'][j]* neuron['delta'
        error . append (error)
else
    for j in range (len (layer)):
        neuron = layer (j)
        errors. append (expected[j] - neuron ['output'])

 for (i in range (len (layer)):
        neuron = layer (j)
        neuron['delta'] = error (j]* trasfer_deravition(neu
          ['output'])

def train_network (network, train, l_rate, n epoch, n_outputs
   print ("\n Network Training Begins : \n")
   for epoch inrange (n_epoch):
       sum_error = 0
   for row in train:
    outputs = forward_propogate (network, row)
    expected = [for i range (n_outputs)]
    expected [row [-1]] = 1
 print (" \n Network training : \n")
 seed(2)
 dateset = [[2.7810836, 2.550537, 0.0030, 0],
            [1.46548 9372, 2.36, 0], [3.39 65616 80, 4.400293
          [1.38807019, 1.856220, 0], [3.06407232, 3.005
           0] [7.627531214, 2.759, 1], [5.33244, 2.088
            1], [6.922596, 1.771063, 1], [7.67, 3.57]
```

Network

L[1]:

[0864245008, 64347664, -0·8497601716670761, 0·86682945]

output:

0·92455881915836

delta: 0·003645382569274

L[2]:

[-1.293402410111027, 1·7193632371811, 0·21253321507]

output: 0.929558796583884

Delta: 0·0004928559973881506]

```
print ("\n The input dataset \n", dataset)
network = initialize network (n_input, 2, n_outputs)
train_network (network, dataset, 0.5, 20, n_output)
print("\n find neural network")
i=1
for layer in network
j=1
for sub in layer:
    print ("\n layer %. (i,j), sub)
j =j+1
i = i+1
def activate ( weights, inputs):
      activation = weights [-1]
for i in range (len (weight -1)):
     activation += weights [i] * inputs [i]
return activation

def transfer (activation)
   return 1.0/(1.0+ exp (-activation))

for row in dataset
    prediction = predict (network, row)
print ('expected=%d, col = %d', %[row[-1], predictio
```

expected =0 , Got=0
expected =0 , Got =0
expected=0 , Got =0
expected-0 , Got = 0
expected=0 , Got = 0
expected=1 , Got = 1
expected =1 , Got= 1
expected=1 , Got = 1
expected=1 , Got= 1
expected=1 , Got =1

✓