Experiment - 4

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

Aim: To Build an Artificial Neural Network by implementing the Back-propagation algorithm.

Algorithm/Procedure:

1. Create a feed-forward network with n_i inputs, n_{hidden} hidden units, and n_{out} output units.
2. Initialize all network weights to small random numbers
3. Until the termination condition is met, Do

   For each (x, t), in training examples, Do

   Propagate the input forward through the network:

   1. Input the instance x, to the network and compute the output o_u of every unit u in the network. Propagate the errors backward through the network
   2. For each network unit k, calculate its error term δ_k
   3. For each network unit h, calculate its error term δh
   4. Update each network weight wji

Program:

```
import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)
X = X/np.amax(X,axis=0) #maximum of X array longitudinally
y = y/100

#Sigmoid Function
def sigmoid (x):
   return 1/(1 + np.exp(-x))

#Derivative of Sigmoid Function
def derivatives_sigmoid(x):
   return x * (1 - x)

#Variable initialization
epoch=5 #Setting training iterations
lr=0.1 #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))
```

```
#draws a random range of numbers uniformly of dim x*y
for i in range(epoch):
    #Forward Propogation
    hinp1=np.dot(X,wh)
    hinp=hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1=np.dot(hlayer_act,wout)
    outinp= outinp1+bout
    output = sigmoid(outinp)
    #Backpropagation
    EO = y-output
    outgrad = derivatives_sigmoid(output)
    d_output = EO * outgrad
    EH = d_output.dot(wout.T)
    hiddengrad = derivatives_sigmoid(hlayer_act)#how much hidden layer wts contributed to error
    d_hiddenlayer = EH * hiddengrad
    wout += hlayer_act.T.dot(d_output) *lr   # dotproduct of nextlayererror and currentlayerop
    wh += X.T.dot(d_hiddenlayer) *lr
    print ("-----------Epoch-", i+1, "Starts----------")
    print("Input: \n" + str(X))
    print("Actual Output: \n" + str(y))
    print("Predicted Output: \n" ,output)
    print ("-----------Epoch-", i+1, "Ends ---------\n")

print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

Training Examples:

Example	**Sleep**	**Study**	**Expected % in Exams**
1	2	9	92
2	1	5	86
3	3	6	89

Normalize the input

Example	Sleep	Study	Expected % in Exams
1	2/3 = 0.66666667	9/9 = 1	0.92
2	1/3 = 0.33333333	5/9 = 0.55555556	0.86
3	3/3 = 1	6/9 = 0.66666667	0.89

Expected Output:

```
-----------------------------Epoch- 1 Starts----------------------------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.80441703]
 [0.79630703]
 [0.80433472]]
-----------------------------Epoch- 1 Ends----------------------------

-----------------------------Epoch- 2 Starts----------------------------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.80545046]
 [0.79728381]
 [0.8053763 ]]
-----------------------------Epoch- 2 Ends----------------------------

-----------------------------Epoch- 3 Starts----------------------------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.80646432]
 [0.79824242]
 [0.80639814]]
-----------------------------Epoch- 3 Ends----------------------------

-----------------------------Epoch- 4 Starts----------------------------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
```

```
[1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.80745918]
 [0.79918337]
 [0.80740077]]
-----------Epoch- 4 Ends----------

-----------Epoch- 5 Starts----------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.80843554]
 [0.80010715]
 [0.80838472]]
           Epoch- 5 Ends

Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.80843554]
 [0.80010715]
 [0.80838472]]
```

Result: Thus an Artificial Neural Network by implementing the Back-propagation algorithm has been built and tested successfully using appropriate dataset.