1. Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

# BACKPROPAGATION Algorithm

BACKPROPAGATION (*training\_example, ƞ, nin, nout, nhidden )*

*Each training example is a pair of the form (*⃗ ,

*), where (*𝑥 *) is the vector of network*

*input values, (*𝑡 *) and is the vector of target network output values.*

*ƞ is the learning rate (e.g., .05). ni, is the number of network inputs, nhidden the number of units in the hidden layer, and nout the number of output units.*

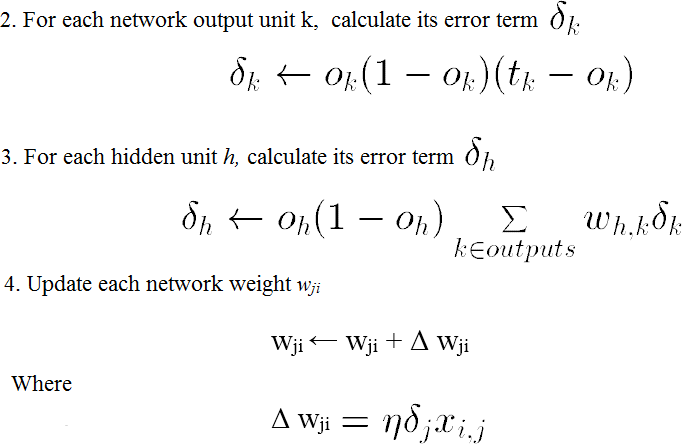
*The input from unit i into unit j is denoted xji, and the weight from unit i to unit j is denoted wji*

* + Create a feed-forward network with ni inputs, nhidden hidden units, and nout output units.
  + Initialize all network weights to small random numbers
  + Until the termination condition is met, Do
    - For each (⃗𝑥 ,

), in training examples, Do

*Propagate the input forward through the network:*

1. Input the instance ⃗ , to the network and compute the output ou of every unit u in the network.

*Propagate the errors backward through the network:*

# 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 |

# 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=5000 #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\_neur ons))

bh=np.random.uniform(size=(1,hiddenlayer\_neurons)) wout=np.random.uniform(size=(hiddenlayer\_neurons,output\_neuron s))

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)

#how much hidden layer wts contributed to error hiddengrad = derivatives\_sigmoid(hlayer\_act) d\_hiddenlayer = EH \* hiddengrad

# dotproduct of nextlayererror and currentlayerop wout += hlayer\_act.T.dot(d\_output) \*lr

wh += X.T.dot(d\_hiddenlayer) \*lr

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

**Output:**

Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output: [[0.92]

[0.86]

[0.89]]

Predicted Output: [[0.89726759]

[0.87196896]

[0.9000671]]