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

Training Examples:

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

Training Examples:

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

Inputs=2
Input Layer=2 Neurons

Output=1
Output Layer=1 Neuron

Note:

Number of hidden layers and corresponding result

0 - Only capable of representing linear separable functions or decisions.

1 - Can approximate any function that contains a continuous mapping from one finite space to another

2 - Can represent an arbitrary decision boundary to arbitrary accuracy with rational activation functions and can approximate any smooth mapping to any accuracy.

More than 2 - Additional layers can learn complex representations but rare scenarios.

Number of neurons in hidden layers

Too few neurons in the hidden layers will result in underfitting, cannot model complicated data sets.

Too many neurons in the hidden layers may result in overfitting, works well on the training dataset and bad on other data. Also, increases the time it takes to train the network.

Rule of thumb guidelines:

- In order to secure the ability of the network to generalize the number of nodes has to be kept as low as possible.
- The number of hidden neurons should be between the size of the input layer and the size of the output layer.
- The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer.
- The number of hidden neurons should be less than twice the size of the input layer.

WARNING: Ultimately, the selection of an architecture for your neural network will come down to trial and error.

Hence for this program we consider:

Inputs=2
Input Layer=2 Neurons

Hence, we use one hidden layer, No. of neurons in hidden layer=3

Output=1
Output Layer=1 Neuron

```python
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))
```

```python
#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
    bout += np.sum(d_output, axis=0,keepdims=True) *lr
    wh += X.T.dot(d_hiddenlayer) *lr
    bh += np.sum(d_hiddenlayer, axis=0,keepdims=True) *lr
print("Input: \n" ,X)
print("Actual Output: \n" ,y)
print("Predicted Output: \n" ,output)
```