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

```
from math import exp
from random import seed
from random import random

def intialize_networks (n_inputs, n_hidden, n_outputs):
    network = list()
    hidden_layer = [{'weights':[random() for i in range
                    (n_inputs+1)]} for i in range(n_hidden)]
    network.append (hidden_layer)
    output_layer = [{'weights':[random() for i in range
                    (n_hidden +1)]} for i in range(n_outputs)]
    network.append(output_layer)
    return network

def activate (weights, inputs):
    activation = weights [-1]
    for i in range (len(weights)-1):
        activation += weight [i]* inputs [i]
    return activation

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

Teacher's Signature : _____

```
def forward_propogate(network, row):
    inputs = row
    for layer in network:
        new_inputs = []
        for neuron in layer:
            activation = activate(neuron['weights'], inputs)
            neuron['outputs'] = transfer(activation)
            new_inputs.append(neuron['outputs'])
        inputs = new_inputs
    return inputs.

def transfer_derivate derivative(output):
    return output * (1.0 - output)

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

Teacher's Signature :

```
        for j in range(len(layer)):
            neuron = layer[i]
            neuron['data'] = errors[j] * transfer_derivate
                                        (neuron['output'])

def update_weights(network, row, l_rate):
    for i in range(len(network)):
        inputs = row[-1]
        if i != 0:
            inputs = [neuron['output'] for neuron in
                      network[i-1]]
        for neuron in network[i]:
            for j in range(len(inputs)):
                neuron['weights'][j] += l_rate *
                                neuron['delta'] * inputs[j]
            neuron['weights'][-1] += l_rate * neuron['delta']

def train_network(network, train, l_rate, n_epoch, n_outputs):
    for epoch in range(n_epoch):
        sum_error = 0
        for row in train:
            outputs = forward_propogate(network, row)
            expected = [0 for i in range(n_outputs)]
            expected[row[-1]] = 1
            sum_error += sum([(expected[i] - outputs[i])
                        **2 for i in range(len(expected))])
            back_propogate_error(network, expected)
            update_weights(network, row, l_rate)
```

Teacher's Signature : _____

```
        print('>epoch=%d, lrate=%.3f, error=%.3f'
                        % (epoch, l_rate, sum_error))
seed(1)
dataset = [
    [2.7810836, 2.550537003, 0],
    [1.465489372, 2.362125076, 0],
    [3.396561688, 4.400293529, 0],
    [1.38807019, 1.850220317, 0],
    [3.06407232, 3.005305973, 0],
    [7.627531214, 2.759262235, 0],
    [5.332441248, 2.088626775, 1],
    [6.922596716, 1.77106367, 1],
    [8.675418651, -0.242068655, 1],
    [7.673756466, 3.508563011, 1]]
n_inputs = len(dataset[0]) - 1
n_outputs = len(set([row[-1] for row in dataset]))

network = initialize_networks(n_inputs, 2, n_outputs)
train_network(network, dataset, 0.5, 20, n_outputs)
for layer in network:
    print(layer)
```

Teacher's Signature : _____

output :-

```
>epoch =0 , lrate = 0.500 , error = 6.350
>epoch = 1 , lrate = 0.500 , error = 5.531
>epoch = 2 , lrate = 0.500 , error = 5.021
>epoch = 3 , lrate = 0.500 , error = 4.951
>epoch = 4 , lrate = 0.500 , error = 4.519
>epoch = 5 , lrate = 0.500 , error = 4.173
>epoch = 6 , lrate = 0.500 , error = 3.835
>epoch = 7 , lrate = 0.500 , error = 3.506
>epoch = 8 , lrate = 0.500 , error = 3.192
>epoch = 9 , lrate = 0.500 , error = 2.892
>epoch = 10 , lrate = 0.500 , error = 2.626
>epoch = 11 , lrate = 0.500 , error = 2.377
>epoch = 12 , lrate = 0.500 , error = 2.153
>epoch = 13 , lrate = 0.500 , error = 1.953
>epoch = 14 , lrate = 0.500 , error = 1.774
>epoch = 15 , lrate = 0.500 , error = 1.614
>epoch = 16 , lrate = 0.500 , error = 1.470
>epoch = 17 , lrate = 0.500 , error = 1.346
>epoch = 18 , lrate = 0.500 , error = 1.233
>epoch = 19 , lrate = 0.500 , error = 1.132
```

```
[{'weights': [-1.4688375095432327, 1.8508873054395514,
  1.8085817862955029 7], 'output': 0.029980305604426185
 'delta': -0.005954660416232355},

 {'weights': [0.3771109814246215 7, -0.06259098945529897
  0.27651237026422716], 'outputs': 0.945602000211323
  'delta': 0.002627965285086387}]

 [{'weights': [2.5153946493978 49, -0.3391927502445
  985, -0.9671565426390275], 'output':
  0.23648794202357587, 'delta': -0.042700592783
  648587}, {'weights': [-2.558414984848 4263,
  1.0036402210620902 02, 0.42383086467582715],
 'output': 0.7790535202438367,
  'delta': 0.038031325964373354}]
```