

4. Build an artificial neural network by implementing the back-propagation algorithm and test the same using appropriate dataset.

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

```
def initialize_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(output_layer)
    return network
```

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

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

```
def forward_propagate(network, row):
    inputs = row
    for layer in network:
        new_inputs = [
```

```
for network neuron in layer:  
    activation = activate(neuron['weights'], inputs)  
    neuron['output'] = transfer(activation)  
    new_inputs.append(neuron['output'])  
input = new_inputs  
return inputs
```

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

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

```

def update_weights(network, row, l_rate):
    for i in range(len(network)):
        inputs = row[i-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_err = 0
        for row in train:
            outputs = forward_propagate(network, row)
            expected[row[-1]] = 1
            sum_error += sum([(expected[i] - output[i]) ** 2 for i in range(len(expected))])
            backward_propagate_errors(network, expected)
            update_weights(network, row, l_rate)
        print(' > epoch = %.1d, l_rate = %.3f, error = %.3f' %
              (epoch, l_rate, sum_error))

```

```

sum(1)

```

```

dataset = [
    [2.7810836, 2.550537003, 0],
    [1.465489372, 2.362125076, 0],
    [3.396561688, 4.400293529, 0],

```


[1.38807019 , 1.850220317, 0],
[3.06407232 , 3.005305913, 0],
[7.627531214 , 2.759262235, 1],
[6.332441248 , 2.088626775, 1],
[6.922596716 , 2.77106367, 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)

Output

> epoch = 0, lr = 0.500, error = 6.350
 > epoch = 1, lr = 0.500, error = 5.531
 > epoch = 2, lr = 0.500, error = 5.221
 > epoch = 3, lr = 0.500, error = 4.951
 > epoch = 4, lr = 0.500, error = 4.519
 > epoch = 5, lr = 0.500, error = 4.173
 > epoch = 6, lr = 0.500, error = 3.835
 > epoch = 7, lr = 0.500, error = 3.506
 > epoch = 8, lr = 0.500, error = 3.192
 > epoch = 9, lr = 0.500, error = 2.898
 > epoch = 10, lr = 0.500, error = 2.626
 > epoch = 11, lr = 0.500, error = 2.377
 > epoch = 12, lr = 0.500, error = 2.153
 > epoch = 13, lr = 0.500, error = 1.953
 > epoch = 14, lr = 0.500, error = 1.714
 > epoch = 15, lr = 0.500, error = 1.614
 > epoch = 16, lr = 0.500, error = 1.472
 > epoch = 17, lr = 0.500, error = 1.346
 > epoch = 18, lr = 0.500, error = 1.233
 > epoch = 19, lr = 0.500, error = 1.132

['weights': [-1.4688375095432327, 1.850887325439514,
 1.0858178629550297], 'output': 0.0299803056
 -0.4089, 'delta': -0.005956609162323625},
 {'weights': [-0.37711098142462157, -0.0625909894552987,
 0.2765123702642716], 'output': 0.94562290002113,
 'delta': 0.0026279652850863837}]

Jovita, 4MT1705044

h' weights : [2.515394649397849, -0.3391929502445985,
-0.9671565426390275, 'output' : 0.2364879420235758,
'delta' : -0.042700592183645873,

h' weights : [-2.558414984848, 1.0036422106209202,
0.42383086467582715, 'output' : 0.7790535204,
3836, 'delta' : 0.038031325964373543]