```
#backpropagation algo  with multilayer perceptron

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

# intialize a network

def initialize_network(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

#   Calculate neuron activation for an input

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

#   Transfer neuron activation

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



# Forward propagate input to a network output

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




#   Calculate the derivative of an neuron output

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




# Backpropagate error and store in neurons

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

```
#   Update network weights with error

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']



# Train a network for a fixed number of epochs

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_propagate(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))])
      backward_propagate_error(network, expected)
      update_weights(network, row, l_rate)
    print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l_rate, sum_error))

# Test training backprop algorithm

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],
[7.627531214,2.759262235,1],
[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_network(n_inputs, 3, n_outputs)
train_network(network, dataset, 0.5, 20, n_outputs)
for layer in network:
  print(layer)
```

```
>epoch=0, lrate=0.500, error=6.880
>epoch=1, lrate=0.500, error=5.740
>epoch=2, lrate=0.500, error=5.326
>epoch=3, lrate=0.500, error=5.338
>epoch=4, lrate=0.500, error=5.299
>epoch=5, lrate=0.500, error=5.136
>epoch=6, lrate=0.500, error=4.924
>epoch=7, lrate=0.500, error=4.692
>epoch=8, lrate=0.500, error=4.419
>epoch=9, lrate=0.500, error=4.112
>epoch=10, lrate=0.500, error=3.785
>epoch=11, lrate=0.500, error=3.456
>epoch=12, lrate=0.500, error=3.138
>epoch=13, lrate=0.500, error=2.838
>epoch=14, lrate=0.500, error=2.563
>epoch=15, lrate=0.500, error=2.313
>epoch=16, lrate=0.500, error=2.089
>epoch=17, lrate=0.500, error=1.889
>epoch=18, lrate=0.500, error=1.710
>epoch=19, lrate=0.500, error=1.552
[{'weights': [0.2663982735658251, 0.7860076057871438, 0.7603367533444867], 'output': 0.9961594902966306, 'delta': -0.000112669012226
[{'weights': [-0.7317965482405842, 2.3716621057165623, -0.5690560485083125, -0.03474552839897625], 'output': 0.23494719808744585, 'd
```