MPL

from random import random

from random import seed

seed(1)

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

## Testing

network = initialize\_network(2,1,2)

for layer in network:

print(layer)

#FORWARD PROP

def activate(weights, input):

activation = weights[-1]

for i in range(len(weights)-1):

activation += weights[i] \* input[i]

return activation

from math import exp

def transfer(activation):

return 1.0 / (1.0 + exp(-activation))

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

#TEST FOR FORWARD PROP

row = [1,0, None]

output = forward\_propagate(network, row)

print(output)

#BACK PROP

def transfer\_derivative(output):

return output \* (1.0 - output)

#ERROR BACKPRO

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'])

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\_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))

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]))

print(f"n\_inputs: {n\_inputs}, n\_outputs: {n\_outputs}")

network = initialize\_network(n\_inputs, 2, n\_outputs)

print(f"Network: {network}")

train\_network(network, dataset, 0.01, 200000, n\_outputs)

for layer in network:

 print(layer)