```
import numpy
# scipy.special for the sigmoid function expit()
import scipy.special
import matplotlib.pyplot
# ensure the plots are inside this jupyter notebook, not an external window
import imageio
# glob helps select multiple files using patterns
import glob

class neuralNetwork:
    # initialise the neural network
    def __init__(self,inputnodes,hiddennodes,outputnodes,learningrate):
        # set number of nodes in each input,hidden,output layer
        self.inodes = inputnodes
        self.hnodes = hiddennodes
        self.onodes = outputnodes
        # learning rate
        self.lr = learningrate

        # link weight matrices ,wih and who
        # weithg inside the arrays are w_i_j, where link is from node i to node j in the next layer
        # w11 w21
        # w12 w22 etc
        self.wih = (numpy.random.normal(0.0, pow(self.hnodes,-0.5), (self.hnodes,self.inodes) )  )
        self.who = (numpy.random.normal(0.0, pow(self.onodes,-0.5), (self.onodes,self.hnodes) )  )

        # activation function is the sigmoid function
        self.activation_function = lambda x: scipy.special.expit(x)

        pass

    # train the neural network
    def train(self,inputs_list,targets_list):
        # convert inputs list to 2d array
        inputs = numpy.array(inputs_list,ndmin=2).T
        targets = numpy.array(targets_list,ndmin=2).T

        # calculate signals into hidden layer
        hidden_inputs = numpy.dot(self.wih,inputs)
        # calculate the signals emerging from hidden layer
        hidden_outputs = self.activation_function(hidden_inputs)

        # calculate signals into final output layer
        final_inputs = numpy.dot(self.who, hidden_outputs)
        # calculate the signals emerging from final output layer
        final_outputs = self.activation_function(final_inputs)

        # output layer error is the (target-actual)
        output_errors = targets - final_outputs
        # hidden layer error is the output_errors,split by weights,recombined at hidden nodes
        hidden_errors = numpy.dot(self.who.T, output_errors)

        # update the weights for the links between the hidden and output layers
        self.who += self.lr * numpy.dot((output_errors * final_outputs * (1.0 - final_outputs)), numpy.transpose(hidden_outputs))

        # update the weights for the links between the input and hidden layers
        self.wih += self.lr * numpy.dot((hidden_errors * hidden_outputs * (1.0 - hidden_outputs)), numpy.transpose(inputs))

        pass

    # query the neural network
    def query(self,inputs_list):
        # convert inputs list to 2d array
        inputs = numpy.array(inputs_list,ndmin=2).T

        # calculate signals into hidden layer
```

```
67            hidden_inputs = numpy.dot(self.wih,inputs)
68            # calculate the signals emerging from hidden layer
69            hidden_outputs = self.activation_function(hidden_inputs)
70
71            # calculate signals into final output layer
72            final_inputs = numpy.dot(self.who, hidden_outputs)
73            # calculate the signals emerging from final output layer
74            final_outputs = self.activation_function(final_inputs)
75
76            return final_outputs
77
78    input_nodes = 784
79    hidden_nodes = 200
80    output_nodes = 10
81
82    # learning rate is 0.3
83    learning_rate = 0.1
84
85    # create instance of neural network
86    n = neuralNetwork(input_nodes,hidden_nodes,output_nodes,learning_rate)
87
88    # train the neural network
89
90    # load the mnist training data csv file into a list
91    training_data_file = open("./mnist_train.csv",'r')
92    training_data_list = training_data_file.readlines()
93    training_data_file.close()
94    print('Training data load Success!')
95    # epochs is the number of times the training data set is used for training
96    epochs = 5
97    for e in range(epochs):
98        # go through all records in the training data set
99        print("epoch=", e)
100       trial = 0;
101       for record in training_data_list:
102           all_values = record.split(',')
103           # scale and shift the inputs
104           inputs = (numpy.asfarray(all_values[1:]) / 255.0 * 0.99) + 0.01
105           # create the target output values (all 0.01, except the desired label which
          is 0.99)
106           targets = numpy.zeros(output_nodes) + 0.01
107           # all_values[0] is the target label for this record
108           targets[int(all_values[0])] = 0.99
109           n.train(inputs,targets)
110           trial += 1
111           if trial % 10000 == 0:
112               print("trial = ", trial)
113           pass
114       pass
115   test_data_file = open("./mnist_test.csv",'r')
116   test_data_list = test_data_file.readlines()
117   test_data_file.close()
118   print('Testing data load Success!')
119   # scorecard for how well the network performs,initially empty
120   scorecard = []
121   # go through all records in the test data set
122   for record in test_data_list:
123       all_values = record.split(',')
124       # correct answer is first value
125       correct_label = int(all_values[0])
126       # scale and shift the inputs
127       inputs = (numpy.asfarray(all_values[1:]) / 255.0 * 0.99) + 0.01
128       # query the network
129       outputs = n.query(inputs)
130       # the index of the highest value corresponds to the label
131       label = numpy.argmax(outputs)
132   #    print("Answer label is:",correct_label," ; ",label," is network's answer")
133       # append correct or incorrect to list
134       if(label == correct_label):
135           # network's answer matches correct answer, add 1 to scorecard
136           scorecard.append(1)
137       else:
```

```
138             scorecard.append(0)
139         pass
140 
141     # calculate the performance score ,the fraction of correct answers
142     scorecard_array = numpy.asarray(scorecard)
143     print("performance = ", scorecard_array.sum() / scorecard_array.size )
```