This coursework requires you to write your own implementation of the backpropagation algorithm for training your own neural network. You are required to do this assignment in the python (python version 3) programming language, using only numpy and/or scipy.

The goal of this assignment is to label images of 10 handwritten digits of “zero”, “one”,..., “nine”. The images are 28 by 28 in size (mnist dataset), which will be represented as a vector x of dimension 784 by listing all the pixel values in raster scan order. The labels t are 0,1,2,...,9 corresponding to 10 classes as written in the image. There are 60000 training cases, containing 6000 examples of each of 10 classes.

The way you choose to design your code for this homework will affect how much time you spend coding. We recommend that you look through all of the problems before attempting the first problem. A good foundation will make the rest of these problems easier. Use object oriented principles to code layers, activation function etc as classes.

Problem 1:

Here you must read an input file. Each line contains 785 numbers (comma delimited): the first number in each row denotes the class label: 0 corresponds to digit 0, 1 corresponds to digit 1, etc. The rest of the values are the 784 pixel values between 0 and 255 correspondig to black and white images. As a warm up question, load the data.

For this problem you must write a function that takes a file path as an argument which contains this data. Your function must return two values (x and y) that contains the data from the file as described. Specifically, the first return value (x) must be a matrix where the rows are individual examples of images, and the columns are individual pixels (n x 784 matrix). The second return value must be a list/array of real numbers representing the labels of the examples (rows) in x.

eg:

1.0,0.0,1.0,0.0,....0.0,0.25,0.0,0.0

...

1.0,0.0,1.0,0.0,...,1.0,0.0,0.0,0.96776

x = [

[1.0,0.0,1.0,0.0,....0.0,0.25,0.0,0.0]

...

[1.0,0.0,1.0,0.0,...,1.0,0.0,0.0,0.96776]

]

y = [5,...,2]

Problem 2:

Implement the backpropagation algorithm in a zero hidden layer neural network (weights between input and output nodes). The output layer should be a softmax output over 10 classes corresponding to 10 classes of handwritten digits (e.g. An architecture: 784 > 10). Your backprop code should minimize the cross-entropy function for multi-class classification problems (categorical cross entropy).

where *j* is the class label

This step should be done with a full step of gradient descent, not stochastic gradient descent or rmsprop. For this problem you must write a function that takes as an input a matrix of x values, a list of y values (as returned from problem 1), a weight matrix, and a learning rate and performs a single step of backpropagation. You will need to do both a forward step with the inputs, and then a backward prop to get the gradients. Return the updated weight matrix and bias in the same format as it was passed.

The list of weight matrices will be a list with 1 entry where the only entry is a matrix in the format where the rows represent all of the outgoing weights for a neuron in the input layer and the columns represent the weights for the incoming neurons. A specific row column index will give you the weight for a neuron to neuron connection.

The list of bias vectors will be in the form where each entry in the list is a vector with the same length as the first set of weights. (e.g. For an architecture of 784 > 10, there will be a single element list with a vector of size 10).

Problem 3:

Extend your code from problem 2 to support a single layer neural network with n hidden units (e.g. An architecture: 784 > 10 > 10). These hidden units should be using sigmoid activations.

For this problem you must write a function that takes as an input a matrix of x values, a list of y values (as returned from problem 1), list of weight matrices, a list of bias vectors, and a learning rate and performs a single step of backpropagation. You will need to do both a forward step with the inputs to get the outputs, and then a backward prop to get the gradients. Return the updated weight matrix and bias in the same format as it was passed.

The list of weight matrices is a list with 2 entries where each entry in the list contains a single weight matrix as previously defined in problem 2. For a network with shape 784 > 10 > 10 the passed list of weight matrices would look like this: [matrix with shape 784x10, matrix with shape 10x10]. Note: though a hidden layer of size 10 is used as an example here, your code must be able to support a hidden layer of dimension n.

The list of bias vectors will be in the form where each entry in the list is a vector with the same length as the first set of weights. (e.g. For an architecture of 784 > 10 > 10, there will be a two element list with an vector of size 10 and a vector of size 10)

Problem 4:

Extend your code from problem 3 (use cross entropy error) and implement a multi-layer neural network, starting with a simple architecture containing any number of hidden units in each layer (e.g. With architecture: 784 > 10 > 10 > 10). These hidden units should be using sigmoid activations.

For this problem you must write a function that takes as an input a matrix of x values, a list of y values (as returned from problem 1), list of weight matrices, a list of bias vectors, and a learning rate and performs a single step of backpropagation. You will need to do both a forward step with the inputs to get the outputs, and then a backward prop to get the gradients. Return the updated weight matrix and bias in the same format as it was passed.

The list of weight matrices is a list with k entries where each entry in the list contains a single weight matrix as previously defined in problem 2. For a network with shape 784 > 10 > 10 > 10 the passed list of weight matrices would look like this: [matrix with shape 784x10, matrix with

shape 10x10, matrix with shape 10x10]. Note: though a hidden layer of size 10 is used as an example here, your code must be able to support a hidden layer of dimension n.

The list of bias vectors will be in the form where each entry in the list is a vector with the same length as the first set of weights. (e.g. For an architecture of 784 > 10 > 10, there will be a two element list with an vector of size 10 and a vector of size 10)

Problem 5:

Extend your code from problem 4 to implement different activations functions which will be passed as a parameter. In this problem all activations (except the final layer which should remain a softmax) must be changed to the passed activation function.

Problem 6:

Extend your code from problem 5 to implement momentum with your gradient descent. The momentum value will be passed as a parameter. Your function should perform “epoch” number of epochs and return the resulting weights.