Programming Problem - MNIST Neural Network

In this assignment, you will be implementing a 1-Layer feed forward neural network for classifying MNIST handwritten digits, a common dataset for learning how to build deep neural networks. You will be responsible for reading in the training and testing data (which can all be found in `/course/cs1460/asgn/mnist/`), building the logic for both the forward pass and backwards pass of your single-layer network, writing the training code to actually learn the network parameters from the data, and finally, writing the testing code to evaluate your network's performance.

Because you will need to be able to manipulate matrices and vectors to successfully complete this assignment, we strongly suggest that you complete this assignment using **Python Version 2.7, with the Numpy Library, for scientific computing**.

It's ok if you're not familiar with Python/Numpy in particular - there are many tutorials online (we specifically recommend this one: http://cs231n.github.io/python-numpy-tutorial/), and nothing about this assignment will require any complex features of the language. As always, feel free to come to hours if you have any questions, or are stuck at any point.

The MNIST dataset is broken up into two parts - training, and test, where each part is made up of a series of images (28 x 28 pixel images of handwritten digits), and the respective labels (values from 0 - 9, representing which digit the image corresponds to). The four files, each of which contains the relevant portion of the dataset can be found in `/course/cs1460/asgn/mnist/*.gz`.

Note: Each of these files come in a very specific format, detailed at the end of the page here: http://yann.lecun.com/exdb/mnist/. To make life easy for you, we've written a simple python script (/course/cs1460/asgn/mnist/mnist_helper.py) that takes care of reading and processing the data, loading it into easy-to-use Numpy arrays. We suggest you familiarize yourself with this code, and copy-paste it into your final implementation. The code is commented, and the variables that contain the Numpy arrays are explicitly pointed out at the end of the program.

As for implementing the actual neural network, we strongly suggest that you take the following approach (this information can be found in Chapter 1 of the Deep Learning book linked from the course webpage):

1. Create Numpy arrays for your network parameters. In this case, these will just consist of two parameters - the weight and bias variables for your neural network.
    - Your network will take input vectors of dimension 784, and output probability distributions over the 10 classes (output will be vectors

of dimension 10). Ensure that your weight and bias parameters have the appropriate shape.

- Initialize your parameters as all 0s (numpy.zeros) to begin with.

2. Code the forward pass of your model. This will consist of the matrix multiplication of your input with the weight parameter, plus the bias parameter, followed by the softmax activation. Note that you will have to write the softmax function on your own - Numpy does not have an out-of-the-box softmax.

3. Code the calculation of the cross-entropy loss between your network's predicted output vectors, and the true labels.

4. Code the backwards pass of your model. This will require you to compute the gradients of the loss with respect to each of the parameters (the weight and the bias). This is the backpropagation algorithm, also detailed in Chapter 1 of the Deep Learning book.

5. Run your training updates for 10,000 batches, where each batch consists of 32 randomly chosen training examples. In other words, the batch size you will use for training is 32. Your learning rate will be 0.5.

6. Evaluate your network's accuracy on the test data.

   - This means you should run all 10,000 test examples through your network after you've run the previous training step, and take the argmax of each of the 10,000 prediction vectors (find which of the 10 labels is assigned the highest probability).
   - If this is equal to the true label, then your network correctly classifies the given example, otherwise, your network makes a mistake.
   - Return your accuracy value as a decimal (you should get a value around .90).

The template script is `/course/cs1460/asgn/mnist/mnist`. *Copy this file and fill in the specified line with the command that runs your code and include this with your handin.* **Make sure to adhere to the output guidelines detailed in the template script - any submissions that fail to do so will have points deducted from the total score**. To hand in, run `/course/cs1460/bin/cs146_handin mnist` from the directory that contains your code.