Problem 1. Implementing a Neural Network Classifier

In this homework, you will implement a neural network and the backpropagation algorithm and stochastic gradient descent with mini-batches. The goal is to implement this yourself with only basic linear algebra packages such as *numpy*, rather than a deep learning library. We will use MNIST, a common classification data set consisting of 28x28 pixel grayscale images of handwritten digits (0-9), with 60,000 training examples and 10,000 test examples. The task is to predict the digit from the image, using a *softmax* output layer.

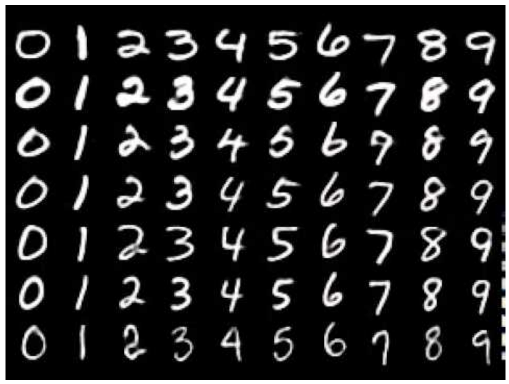

Figure 1: MNIST example images

Requirements:

(a) Use tanh or relu hidden units. Logistic (also called sigmoid) units don't work as well in the hidden layers.

(b) Use mini-batch gradient descent (e.g. mini-batches of size 100). You can use momentum or any other tricks you wish to speed up training.

(c) Train a network with at least two hidden layers, and try to tune the hyperparameters (e.g. learning rate) get good performance on the test data. Plot both the training and test *loss* (the NLL) throughout training (at every epoch), and write a paragraph describing the network, training algorithm, and any hyperparameter choices. Make sure to report the *accuracy* (also called the *01-loss* — to get a class prediction from a softmax layer, we simply take the *argmax* of the probabilities.

(d) Next, try to construct a network that *overfits* to the training data, so that the test loss increases while the training loss continues to decline. (Hint: this should be easier with a network of a single hidden layer.)

Tips:

(a) For a minimalist implementation of a neural network, see:
https://iamtrask.github.io/2015/07/12/basic-python-network/

(b) To quickly download the MNIST data set, see:
https://github.com/hsjeong5/MNIST-for-Numpy

(c) A single pass through the training data is called an *epoch*. You may have to do 50-100 epochs to see overfitting. If your computer is slow, go ahead and use a fraction of the data set, e.g. 6,000 examples.

(d) We typically compute the validation loss at the end of every training epoch to see if we are overfitting. For the training data set, we usually keep a running average of the loss while performing the mini-batch training updates to save on computation.