Question 1: A Perceptron Network for Digit Recognition

There is an excellent online database of handwritten digits that is well-suited for our programming projects:

http://yann.lecun.com/exdb/mnist/

It contains a training set of 60,000 digitized 28×28 pixel bitmap images of handwritten digits, and a test set consisting of another 10,000 images. To simplify access and save memory, the images are encoded in a very simple format. All details are provided on the website. As you can see, many different neural networks have been tested on this database, and the resulting misclassification rates are listed.

(a) Now it is your turn to program a single-layer perceptron network to do this task and see what error rate you get. Since there are ten different digits, i.e., classes, you should use ten perceptrons, each with 28×28 = 784 inputs, and implement the network the way we discussed it in class. Luckily for you, there is a simple Java framework programmed by Tyler Garaas that will make your task easier (nn_framework.zip on the course homepage). First of all, you can use the following functions to read the training and test data (those data are also included in the zip file):

```java
trainingSet = new MNISTImages("trainingLabels", "trainingImages");
 testSet = new MNISTImages("testLabels", "testImages");
```

The images are the 28×28 pixel bitmaps, and the labels are the classes, that is, the desired outputs 0 to 9. The framework also contains abstract “Neuron” and “Synapse” classes, which are rather self-explanatory (like the rest of the code) and will help you write clear and reusable code. You have to use these classes in your program. You do not have to define a termination criterion for your training algorithm, but simply let it terminate after a certain number of epochs (one epoch is finished if each training sample was presented to the network exactly once).

Please note one important thing: the GetPixel values are interpreted as signed values, i.e., ranging from -128 to 127. This makes the job for the network unnecessarily
difficult, because in that case black to white doesn't form a scale. So in your code, you should access pixel values like this:

\[
\text{double temp} = \frac{(\text{image.GetPixel(p) & 0xff})}{255.0};
\]

This resolves the unsigned issue and scales inputs from 0 to 1. Regarding learning rates, please test different ones; for the current task, small rates of 0.01 to 0.1 should work best. Also, you could try out different ways of normalizing the input and scaling the initial, random weights.

(b) To evaluate the performance of your perceptron network, run 50 epochs of training on the training set. After each epoch, compute the network error (proportion of misclassifications) on the test set, without adjusting any weights. Also, compute the error for the training samples in each epoch. Then draw a chart with a horizontal axis titled “Epoch” ranging from 1 to 50, and a vertical axis titled “Error (%).” Then add a line graph showing the network error for the training set as a function of the epoch. Finally, add another line graph showing the error for the test set, also across epochs 1 to 50.

**Question 2: … And Now the Same with an Adaline Network**

(a) Modify your program from Question 1 so that it now simulates an Adaline network. The structure of this network is basically the same as for the perceptron network, but the learning algorithm is different.

(b) Evaluate the network in exactly the same way as in Question 1b.

(c) Compare the error charts of the perceptron and Adaline networks. Are there any differences? Discuss the reason for these differences. If there is no visible difference, discuss what differences you would have expected based on your knowledge about these network types.

**Question 3: The Advantage of Multilayer Networks**

Multilayer networks are supposed to perform better than single-layer networks on most classification tasks. This makes sense, of course, since having more layers means having more weights, and so there are more degrees of freedom, i.e., more values to adjust to achieve a better match with the training data. However, is the number of weights the most important advantage of multilayer networks? What do you think would happen if you just increased the number of neurons in your single-layer Adaline network, for example, by assigning three neurons to each class? Would the network reach the performance of a 30-neuron (excluding input neurons), three-layer, backpropagation network? Why or why not? Please discuss these questions in two or three paragraphs.