Programming Project #3:
Perceptron Learning and Digit Recognition
Due Friday, Dec 1st, 5pm

In this programming project assignment, you will use a *perceptron learning* method to train a simple neural network to recognize handwritten digits. The input to your system will be a set of 1024 binary pixels from a 32x32 image, and the output should indicate which of the digits (0 – 9) is in the image. The data sets (over 3000 instances) are available at the public ics-171 directory: /ics-171/Files/DigitRecog/. These data were retrieved from the UCI Machine Learning repository at http://www.ics.uci.edu/~mlearn/MLRepository.html, and originally created (1995) by Alpaydin & Kayna from Istanbul, Turkey. The digits were handwritten by a group of 43 people, and preprocessed to build the 32x32 pixel images.

As described in class (and in your textbook), you must build a perceptron learning system for this performance task. I would recommend a fully-connected network structure of 1024 input nodes, and 10 output nodes, with one output node for each digit. This means that the learning task is to adjust 10,240 weights! For the activation function, you should be able to simply use the sign function (part (b) of Figure 19.5 in your textbook), and therefore avoid the need for any additional threshold weights. Notice that this activation function means that the correct output is 1 for the appropriate output node and -1 for all other output nodes.

The general learning rule for a perceptron network is $newW_{ij} \leftarrow oldW_{ij} + \alpha \times I_i \times Error_j$. Since you are using the sign function for your activation function, you can simply start by initializing all weights to be zero. I believe that the value for α (the learning rate) can be fairly high – try 0.5. Since the error can only range between 2 and -2, I would recommend setting α no higher than 1.0. If you use a lower value of α, you should see slower learning.

Requirements:

For this assignment, there are significant *testing* requirements, so I would recommend finishing the coding early—**at least** 1 week before the due date. Recall that producing a good neural network usually requires many *epochs* of training over the data. Your requirements are:

- Implement a simple perceptron learning system which outputs one of the 10 digits, given a 32x32 bitmap image of a digit. As with programming assignment #2, the user interface is not a significant part of this project. In fact, the output of the system should go to a file, so that it can be post-processed to build learning curves (see below).
- Demonstrate the system's *learning* ability: produce a learning curve that measures performance against the amount of training you allow your system to do. Performance must be measured by error rate against a separate *test set* data file (provided in the public directory /ics-171/Files/DigitRecog/).
- Do a 10-fold cross validation, producing an average learning curve with 10 different training sets. How does this curve compare to your original learning curve? Why is it different? (Answer these questions in your project report.)

- Create a threshold for "confusing" instances: If more than one of the outputs is activated, then that instance should be set aside for hand-classification. For the remaining, unambiguous instances, what is your system's performance rate? This sort of adaptation is described in your text on page 586.

Advanced stuff (for extra credit):

Unlike the last assignment, completely satisfying all of the above requirements (as well as producing a good project report) is sufficient for an A grade. However, for extra credit, try applying your perceptron network to *character recognition*. (This extra credit will be applied to improve your team's grade on other projects. Note that simply attempting the extra credit won't be worth much—I will judge the extra credit effort based on your results.)

At ftp://ftp.ics.uci.edu/pub/machine-learning-databases/letter-recognition/ you will find a dataset of 20,000 instances of the 26 capital letters in English. Unlike the digit recognition task, the inputs are 16 numeric features that have been abstracted from the images. (The original image data is, I suppose, unavailable.) These are briefly described in the file "letter-recognition.names". Thus, the network should (probably) be a fully connected structure with 16 input nodes and 26 output nodes. This means the perceptron does not need to learn as many weights ("only" 416 weights), but it will need more instances, since there are 26 different classes, rather than just 10. Finally, because the inputs are not just binary, you may need to be more careful about your selection of α, and you might need to use a threshold activation function rather than the simple sign function. Does *normalizing* the input values improve either the learning rate or the final performance? As with project #2, documenting your extra efforts carefully and completely in the project report is especially important.

Project report details

For this project report, you must include your graph of the learning curve, including actual data for the error rates at different points during training. You should describe carefully your process for building this curve, including what time points you used and how you measured error rate.

You must also document how you set the "confusing instance" threshold. You should provide some examples of confusing instances, and indicate which of the output nodes were activated for that instance. (E.g., the system might confuse a "7" with a "1".)

Note that this project doesn't really have much of a user interface: Read data file, and then either go into training mode, (using the file to adjust weights) or go into test mode, and report out a performance measure.

Division of labor:

Once again, a reminder that your team must rotate roles from the previous projects. For this project, perhaps there need be only one principle programmer, who does the early work in coding up the perceptron learner. The other programmer can then be in charge of testing and building the learning curves; this may required post-processing programming to produce an performance score for a given perceptron network and a given test set. Obviously, both programmers must understand the interface from the data files to the system, as well as how to set parameters such as α and the determination of a "confusion" threshold.

Reminder: no late projects! Unlike the other projects, this one is due on Friday, the last day of classes at 5pm (rather than on a Thursday at 2pm).