CS/NEUR125 Brains, Minds, and Machines

Lab 4: Artificial Neural Networks

Due: Wednesday, February 22

This lab explores the design and analysis of artificial neural networks to recognize handwritten digits from images. This application of neural nets was first explored by Yann LeCun and colleagues, who trained a network to recognize handwritten zip code digits from image samples provided by the U.S. Postal Service (LeCun et al., 1989). The first part of this lab uses an interactive tool provided with the MATLAB Neural Network Toolbox to explore basic aspects of network design, and one method for measuring the performance of a network. In the second part of the lab, you will work with a script file to analyze the behavior of a neural network in more detail, for two different network design choices.

To begin, create a copy of this Google document, modify the title of the copy to include your partner names, and share the copy between partners, as you did in previous labs. Questions for you to answer for this lab are shown in blue.

Start MATLAB on the lab Mac that you are sharing, and use **Fetch** to download the folder named **NeuralNetsLab** from the CS file server to the Desktop on your Mac. You will find this folder inside the download folder in your individual account on the CS server. For this lab, **set the Current Folder in MATLAB to the NeuralNetsLab folder**. To do this, you can first set your Current Folder to the Desktop of your Mac as you did in previous labs, and then double-click on the NeuralNetsLab folder that is listed among the contents of the Current Folder on the left side of the MATLAB window.

I. Using the MATLAB nprtool to Design and Test a Neural Network to Recognize Digits

MATLAB provides an interactive software tool for constructing simple neural networks and running simulations to train and test the networks for different datasets. For this lab, we will use a subset of the MNIST dataset consisting of images of handwritten digits (from 0 to 9). Enter the following command in the Command Window to load the data:

```
>> load MNIST.mat
```

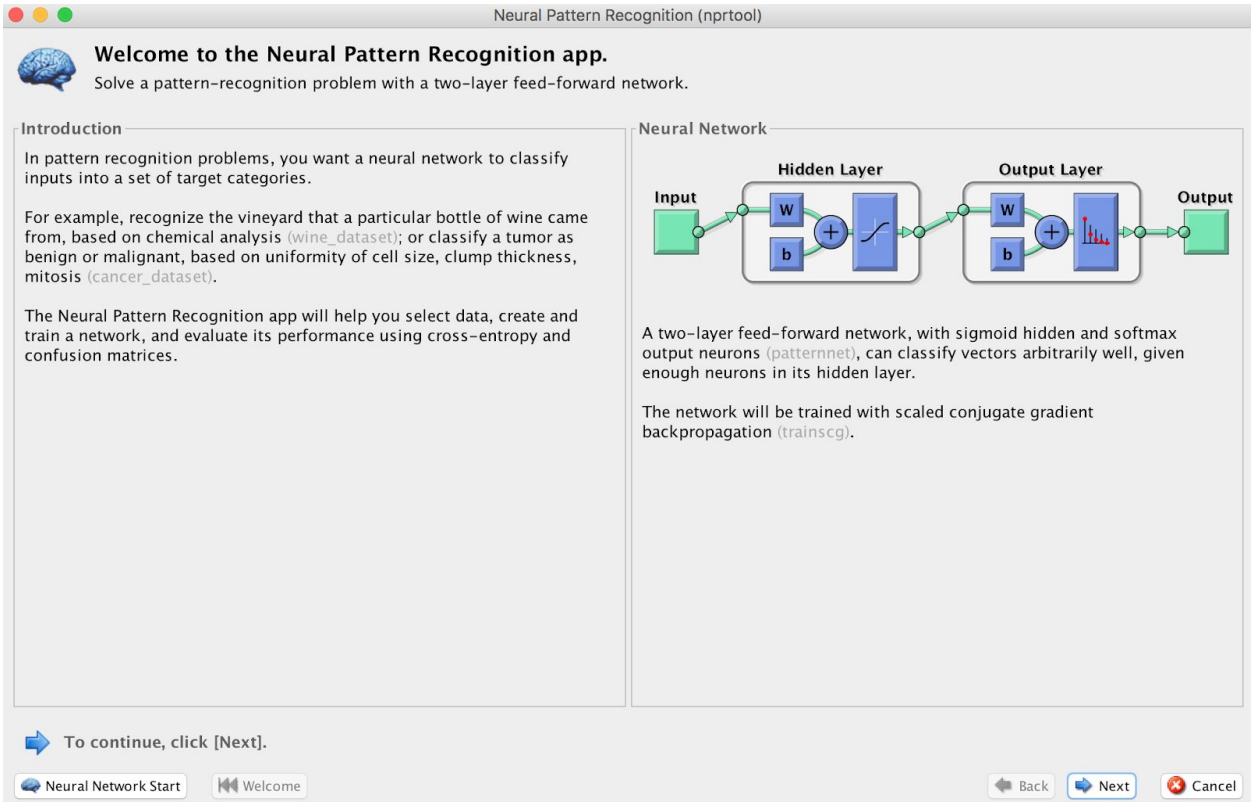
The data consists of 3000 images of handwritten digits, with each image of size 28 x 28 pixels. Enter the following command to view the first 400 image samples:

```
>> displayData(X, 20, 20, 650, '400 MNIST image samples')
```

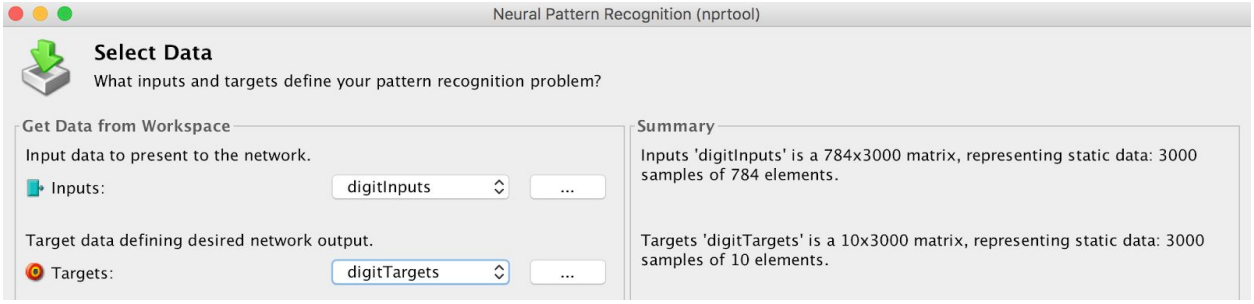
Enter the following command to start the neural network pattern recognition tool:

```
>> nprtool
```

The following window will appear:



Click on the **Next** button in the bottom right corner. The next window (**Select Data**) is used to select the sources of the input and output data for the network. For this application, the inputs are stored in a 784 x 3000 matrix named **digitInputs** and the *correct* outputs are stored in a 10 x 3000 matrix named **digitTargets** (the content of these matrices will be described in lab). Select these two variable names from the pull-down menus labeled **Inputs:** and **Targets:**, as shown below (only the top half of the window is shown):



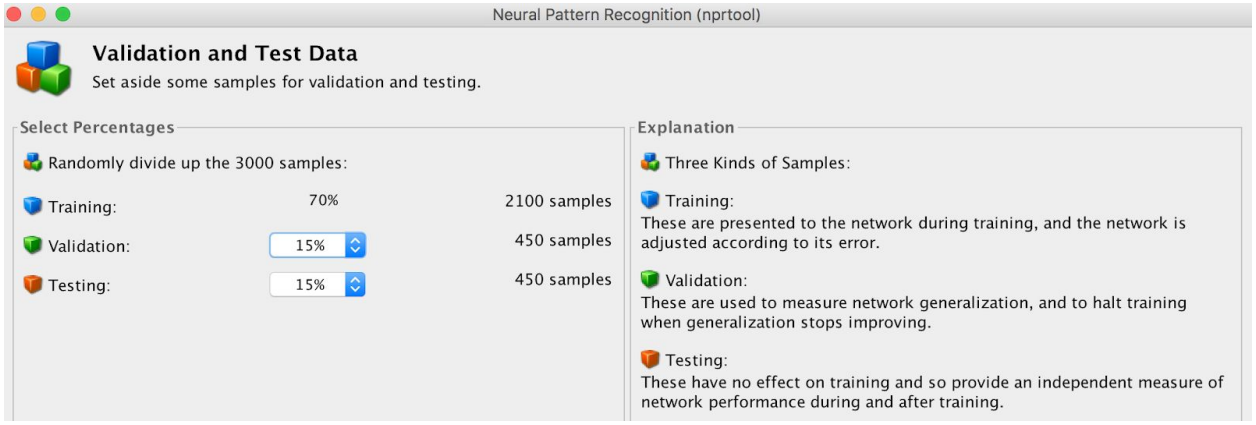
Again click on the **Next** button in the bottom right corner to move to the **Validation and Test Data** window, shown on the next page. The 3000 image samples are divided into three groups:

1) one subset of image samples to be used for **training** the neural network
2) one subset of images that is distinct from the training set, which is used to determine when to terminate the learning process - training ends when there is

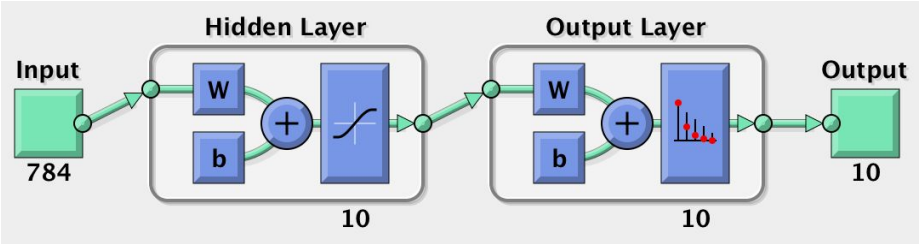
no longer any improvement in performance on this set, which is referred to as the **validation** set

3) a subset of images that is distinct from the training and validation sets, used to **test** the performance of the network at recognizing "new" data samples.

Using the pull-down menus on the left side of the window, you can adjust the portion of the 3000 image samples used for training, validation, and testing. Leave the default values for now.

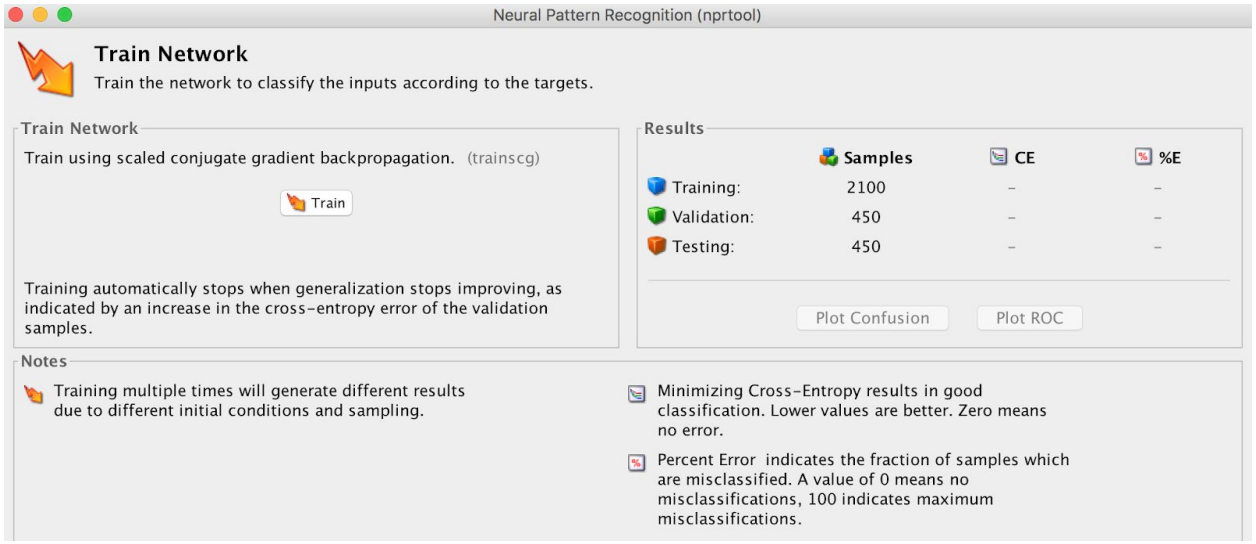


Click on the **Next** button again to see a window that enables you to specify the **Network Architecture**, by indicating the number of **Hidden Neurons** (hidden units) to use. Leave the default value of **10** hidden units for now. You will see a depiction of the structure of the network at the bottom of the window (see picture below). The unit labeled **b** in the Hidden and Output layers is the "bias" unit that we described in class.



Click on the **Next** button once more, to advance to the **Train Network** window (see picture on the next page). Click on the **Train** button to run the backpropagation method to train the neural network that you created. As the network is being trained, a new window appears, labeled **Neural Network Training (nntraintool)** at the top. In the **Progress** box in the middle of this window, you will see the number of iterations (labeled **Epoch:**) increase over time. As indicated earlier, and suggested in comments in the T**rain Network** window, the learning ends when there is no longer any improvement in performance on the validation set. There are two measures used to evaluate the overall performance of the trained network on the test set, **Cross-Entropy** and **Percent Error**. We will focus on the Percent Error, which is the fraction of samples in the test set that were misclassified (e.g. an image of the digit "7" was misclassified as a "9"). This measure is listed in the far right column of the **Results** table in the upper right corner of the **Train Network** window. After the network is trained, the **Train** button changes to a

Retrain button, enabling you to repeat the learning process for the same network. We will explore the **Plot Confusion** and **Plot ROC** options later in this lab.



Q1. In the table below, record the Percent Error (**%E**) listed in the third row of the Results box (labeled **Testing:)** and the total number of iterations shown in the window labeled **Neural Network Training (nntraintool).** You will see that the results vary across the three runs. Why might the results differ for multiple runs of the same training process? In your answer, describe how are the weights of the initial network are set, and how the data samples are divided among the training, validation, and test sets. Enter the average value of the Percent Error and Number of Iterations in the far right column of the table.

(10 Hidden Units)	Run 1	Run 2	Run 3	Mean
Percent Error				
Number of Iterations				

Q2. Click on the **Back** button to return to the **Network Architecture** window and change the number of hidden units to 25. Proceed to the **Train Network** window again, run the training process three times for this new network, and again record the Percent Error and Number of Iterations for each run, as well as the average values, in the table below. Did performance get better or worse relative to the results reported in Q1?

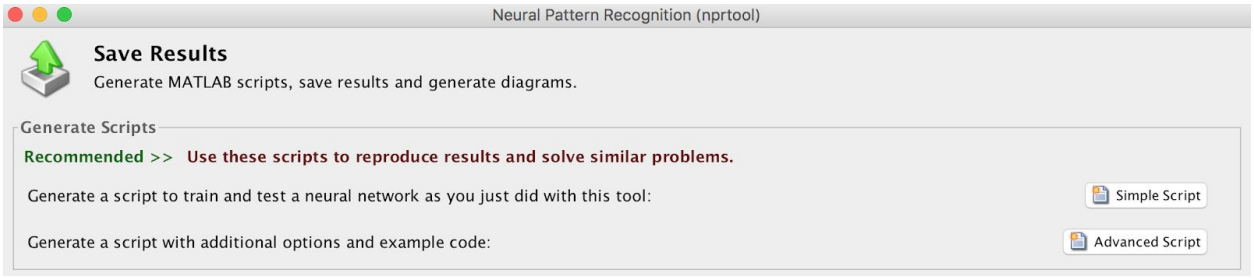
(25 Hidden Units)	Run 1	Run 2	Run 3	Mean
Percent Error				

Number of Iterations				

Q3. Using the **Back** buttons, return to the **Validation and Test Data** window and set the fraction of samples used for both validation and testing to 35%. You will see that the number of samples used for training in this case is reduced to only 30% (900 image samples). Set the number of hidden units to 10 (the default), run the training process three times for these new conditions, and record the Percent Error and Number of Iterations for each Run, and average values, in the table below. Relative to the conditions tested in Q1, did performance get better or worse?

(30% Training Data)	Run 1	Run 2	Run 3	Mean
Percent Error				
Number of Iterations				

Using the **Next** button, advance the tool until you see the **Save Results** window. In the top of the window, click on the **Simple Script** button to see how MATLAB can create a script file that contains comments and code statements to design and train the neural network that you just created inside the **nprtool** program. Read through the comments and code in this file, which will be opened in the MATLAB Editor. You do not need to save this file.



In the **Save Results** window of the **nprtool** program, click on the **Finish** button in the bottom right corner to end the session. The rest of this lab will use a script file like the one you viewed.

Open the **mnistScript.m** code file in the Editor by clicking on its name in the Current Folder. In the Editor, click on the green triangle labeled **Run** in the menu bar at the top of the window, or enter the following command in the Command Window, to run the script file:

```
>> mnistScript
```

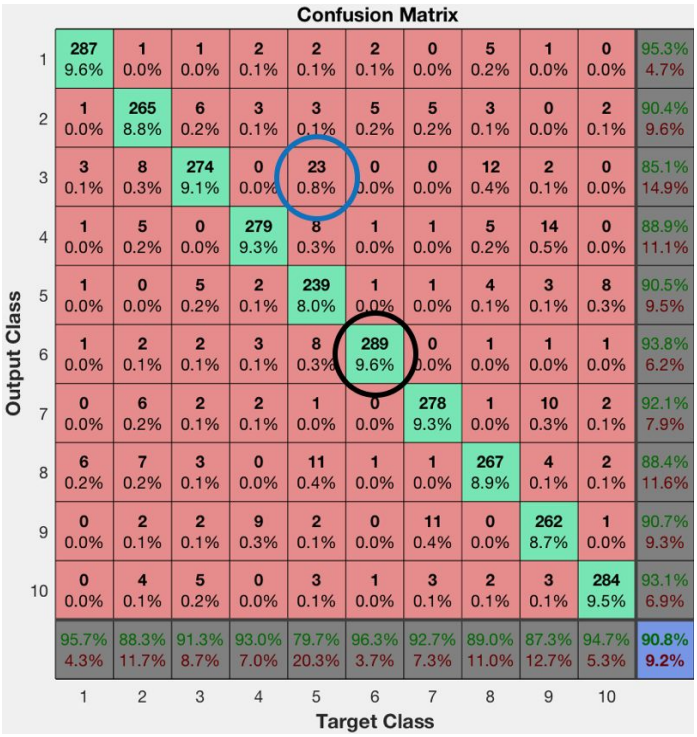
The overall Percent Error and number of iterations will be printed in the Command Window. In lab, we will go through the code and also describe what is displayed in the various figure windows that appear when you run the code (Performance Plot, Error Histogram, Confusion Matrix, and ROC Plot). Following this introduction, answer the following questions about the results obtained for this example (note that if you run the code again, you will obtain slightly different results, similar to your experience with the nprtool program).

Q4. For this example, what was the overall **Percent Correct** obtained, and the **Number of Iterations** performed during the training process?

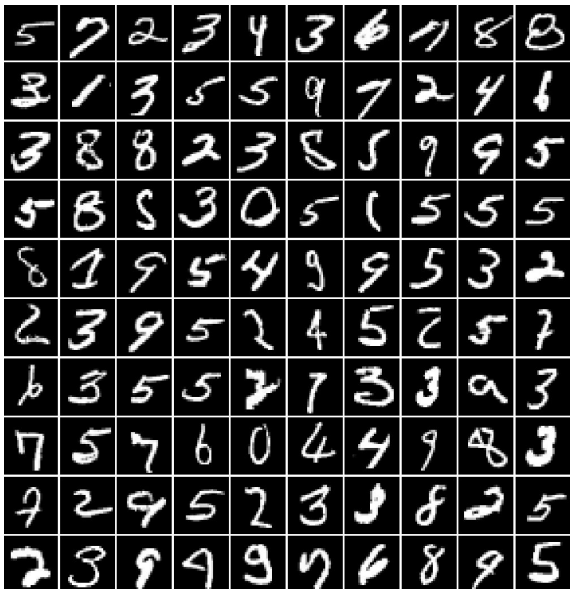
Q5. Examine the plot labeled **Performance** at the top. The vertical axis shows **Mean Squared Error,** and you will see some values labeled on this axis in scientific notation. What are the corresponding values in decimal notation? Does the error obtained for the **Test** set *always decrease monotonically* as each new iteration (Epoch) is performed (i.e. ***always*** decreases from one iteration to the next)? At the point where the "Best" performance on the **Validation** set is reached, is the performance on the **Training** set better or worse?

Q6. Next examine the plot labeled **Error Histogram** at the top. What is the overall range of the error values on the horizontal axis? What are the possible target (correct) values that could be obtained for a particular output unit of this network? How might one get one of the extreme error values (i.e. what are combinations of the "correct output" and output computed by the network, that would yield these extreme error values)? What is the range of error values for the center bin of the histogram that has the most instances?

Now observe the figure labeled **Confusion Matrix**. The **Target Class** (correct digit) of the image samples is shown along the horizontal axis, and the **Output Class** (digit computed by the network) is shown on the vertical axis. Each box shows the number of samples in the network results (out of the full set of 3000 samples) that correspond to each combination of Target and Output class. The green squares show how many samples of each target digit were classified correctly, while the red squares show numbers of samples that were classified incorrectly. Consider the Confusion Matrix shown below. In this example, 289 image samples of the digit "6" were *correctly classified* as a 6 (see matrix location circled in black). On the other hand, 23 image samples of the digit "5" (Target Class) were *incorrectly classified* as a 3 (Output Class), according to the content of the matrix location circled in blue.



The figure below shows some examples of images whose digit class was misclassified by one sample network:



Q7. Save the Confusion Matrix that is displayed for your example and then drag it into this document. What are some sample combinations that yielded large numbers of "confusions"? Report the Target and Output class for four combinations that yielded many confusions. Are you surprised by these confusions (briefly explain why or why not)? Along the bottom row of the matrix, the green numbers show the percentage of samples for each Target Class that were classified correctly, and red numbers show the percentage of each Target Class that were misclassified. For your example, which

digit(s) were the most difficult to recognize correctly?

Q8. Finally, observe the figure labeled **Receiver Operator Characteristic** (or **ROC**) at the top. In lab, we will discuss the meaning of ROC curves. The ROC curve for each Class (digit) is displayed in a different color. Observing the ROC curves for your example, which digit(s) appear to be the most difficult to classify correctly (this many not be apparent if they were all classified well)? Are these digits similar to the ones you observed as difficult to classify in **Q7**?

Q9. The current script specifies 10 hidden units for the network. Modify the script so that 25 hidden units are used instead and run the code with this new network architecture. Describe at least three ways that the results changed from those described for the network with only 10 hidden units.

Q10. Provide two questions of your own, about things related to Neural Networks that you still find confusing.

Reference

LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W. & Jackel, L.D. (1989) Backpropagation Applied to Handwritten Zip Code Recognition, *Neural Computation 1*, 541-551.