Computer Science 475 – Fall 2021 – Assignment 2
MNIST Handwritten Digit Recognizer Neural Network in Java
Due: Thursday, October 28, 2020 @ 7:00 am

This assignment will give you experience implementing and training a multi-layer (3 layer) neural network. Specifically, you will construct a Java program that recognizes the MNIST digit set. The data set, in CSV format, is available at https://pjreddie.com/projects/mnist-in-csv/ and also on Moodle. The format of each line of data is: "label, pix-1-1, pix-1-2, pix1-3, … , pix-28-28" where label is a digit 0-9 and pix-X-Y is a greyscale value from 0-255. NOTES: (1) The label will need to be converted to a "one hot vector" – a 10 by 1 array in order to work with our network; (2) For best results, scale your pixel inputs to be a fraction between 0 and 1 by dividing every pixel input by 255, storing the result as a double.

A Python version of this program is presented (and its workings explained in detail) in Chapter One of Michael Nielsen's book "Neural Networks and Deep Learning", freely available online at http://neuralnetworksanddeeplearning.com. You should use Nielsen's program as a guide to constructing your own implementation; in other words, use Nielson's Python program as a guide to understanding the algorithms, but write your own program in **Java** from scratch. Do NOT try to simply transliterate the Python into Java. Based on the experience of past students that path is doomed to failure.

At a minimum, your program should support the following user-selectable operations:

[1] Train the network
In training mode, your program should iterate through the 60,000 item MNIST training data set. I used a learning rate of 3, a mini-batch size of 10, and 30 epochs, randomly initializing my weights and biases to the -1 to 1 range, and scaled my pixel inputs to 0 – 1 range, when performing training.

After each training epoch, your program should print out statistics showing: (1) for each of the ten digits 0-9, the number of correctly classified inputs over the total number of occurrences of that digit; (2) the overall accuracy considering all ten digits. In other words, after each epoch, the output should look something like this:

```
   0 = 4907/4932   1 = 5666/5678   2 = 4921/4968   3 = 5034/5101   4 = 4839/4859   5 = 4472/4506
   6 = 4935/4951   7 = 5140/5175   8 = 4801/4842   9 = 4931/4988   Accuracy = 49646/50000 = 99.292%
```

(Note that I goofed thinking there were only 50,000 items in the training set. There are actually 60,000 in the set I provided you. You should use all 60,000.)

[2] Load a pre-trained network
Your program should be able to load a weight set (previously generated) from a file.

[3] Display network accuracy on TRAINING data {only available after selecting [1] or [2] above}
Iterate over the 60,000 item MNIST training data set exactly once, using the current weight set, and output the statistics shown in [1] above.

[4] Display network accuracy on TESTING data {only available after selecting [1] or [2] above}
Iterate over the 10,000 item MNIST testing data set exactly once, using the current weight set, and output the statistics shown in [1] above.

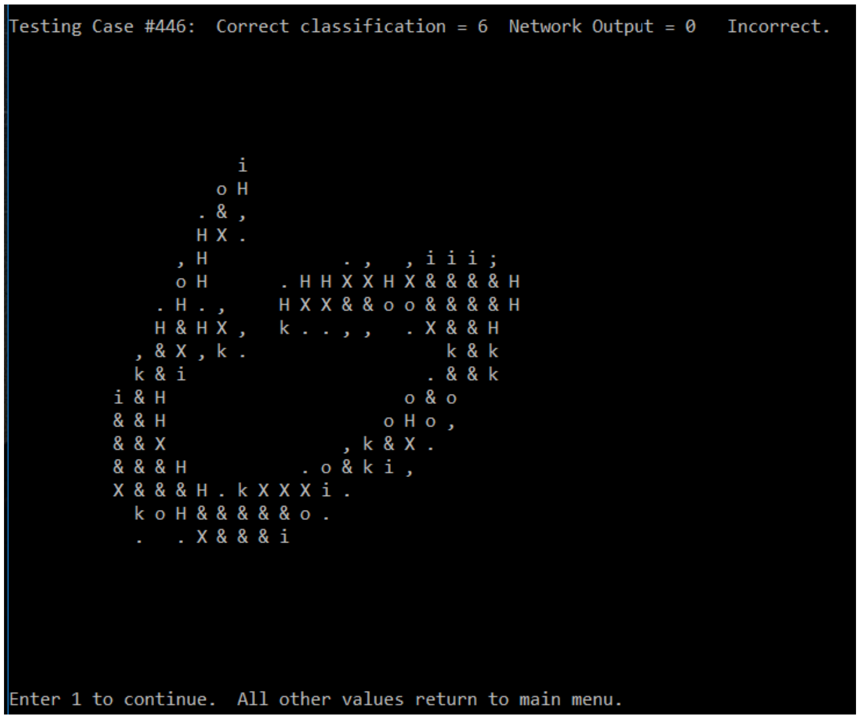
[5] Save the network state to file {only available after selecting [1] or [2] above}
Your program should be able to save the current weight set to a file.

[0] Exit
You should be able to exit the program.

Stretch Goals: Extend your program to include the option to:

(a) Run the network on TESTING data showing, for each input image, a representation of the image itself, its correct classification, the network's classification, and an indication as to whether or not the network's classification was correct. (I used ASCII art for the image in order to stick with a command line program and provided the ability to return to the main menu after each image. See Figure 1.)

(b) Similar to (a) except I displayed ONLY those images that were misclassified by the network.



```
Testing Case #446:  Correct classification = 6  Network Output = 0   Incorrect.

            i
          o H
        . & ,
        H X .
      , H             . ,   , i i i ;
      o H       . H H X X H X & & & & H
    . H . ,     H X X & & o o & & & & H
    H & H X ,   k . . , ,   . X & & H
  , & X , k .                   k & k
  k & i                       . & & k
i & H                       o & o
& & H                     o H o ,
& & X                 , k & X .
& & & H           . o & k i ,
X & & & H . k X X X i .
  k o H & & & & & o .
  .   . X & & & i

Enter 1 to continue.  All other values return to main menu.
```

Figure 1

Additional Requirements:

1. You must work independently and develop your own code. Sharing of code is ABSOLUTELY forbidden. Lifting a solution from the internet will not help you either, as you must be able to explain how the code operates, and if you don't write it yourself you won't be able to do so. It is far better to turn in your own program even if it only "partially" works, rather than some "perfect" program you did not write.

2. You ***must*** thoroughly document your program, including comments at the beginning of your main program with: your name, your student number, the date, the assignment number, and a brief description of what the program does. Comments should also be present throughout the program to explain what each part does. Lack of appropriate comments can cost you up to 20% of your overall program grade.

3. Your program must be written in Java and run in Eclipse or from the command line.

Submitting your assignment for grading:

A copy of your program should be emailed to me (walters@LaTech.edu) **no later than 7am on Thursday, October 28th**. The subject line of the email should be: *CSC 475 : Neural Network Program : <your name>* where *<your name>* is replaced with your name. So, for example, if Susan Calvin were enrolled in the class, her email would use the subject line: *CSC 475 : Neural Network Program : Susan Calvin*

You will be required to meet with me for approximately 15 minutes on Thursday, October 28 or Friday, October 29, to demo your program and explain how it works. Have a laptop with you, with the program ready to go. Be sure you come to the meeting with a weights file that can be loaded, so that you can demo your program's efficiency without us having to wait through a complete training cycle.

You will need to sign-up for an appointment using a sign-up sheet that will be accessed via google sheets (link on Moodle soon) no later than Tuesday, October 26th. These meetings will determine your grade on the program. If you cannot explain to my satisfaction how your code works, or make simple changes to it, you will receive a grade of zero on the program (which being less than 50% will result in failure of the course). Now, don't panic! If you wrote your own program you will find your 15 minute meeting to be exceptionally easy to get through.

Be prepared to demo your program on your own laptop. However, during the meeting I may require you to copy your program onto my laptop and demonstrate that it compiles and runs under my version of Eclipse.

Assignment 2 Grading Rubric

Documentation	20%	Program should include comments at the beginning of your main program with: your name, your student number, the date, the assignment number, and a brief description of what the program does (10%). Comments should also be present throughout the program to explain what each part does. (10%)
Understand the program	10% **or FAIL**	Student should clearly understand what the program does and be able to explain the operation of any part of the program. If so, 10%. **If not 0% as the student did not write the program him or herself.**
Modify a section of the program	10%	Student should clearly understand what the program does and be able to make a small modification to the program (e.g. print a message after each epoch or change then number of nodes in the hidden layer). If so, 10%. **If not 0% as the student did not write the program him or herself.**
I/O Behavior	20%	File I/O supported correctly (10%). End-user menu implemented correctly (10%).
Accuracy	40%	Numbers correspond to **TRAINING** DATA (TESTING DATA should be no more that 5% lower than training or -15%) ::: Accuracy >= 95% (40%); Accuracy >= 90% (30%); Accuracy >= 80% (20%); Accuracy >= 50% (10%); Accuracy < 50% (0%). NOTE: Another way of scoring partial credit (20%) is to show that your code can exactly reproduce the numbers provided in the Excel spreadsheet.

NOTE: Up to an additional 10% bonus can be earned by adding the ability to produce visualizations of the input data. You may use ASCII art (create your OWN ASCII art code to do this, don't use a pre-written library). Gray scales should be indicated by sequentially darker characters (For example . : ; i I T H # -- No, you don't have to use these exact characters). You also have the option of using an actual GUI / graphics library to represent the input data visualizations.