Neural Network Fundamentals

This assignment will give you experience implementing and training a multi-layer fully connected, feed forward network consisting of sigmoidal neurons, trained with stochastic gradient descent and back propagation.

This assignment is divided into two parts – the training of a small network and the training of a larger network that solves the real-world problem of recognizing handwritten digits (0-9).

Part 1: The Small Network

Part 1 requires that you manually implement and verify the functionality of the core neural network and training algorithms in Java. To do this, you will write a Java program that implements and trains a very tiny network consisting of 3 layers as follows:

- An input layer with 4 nodes
- A hidden layer with 3 nodes
- An output layer with 2 nodes

The values computed by your Java program must be identical to the values calculated by the sample spreadsheet accompanying this assignment titled *Part 1 – Small Network.xlsx*. Essentially, you will design your neural network engine to include diagnostic print statements that show the value of every variable that is displayed in the spreadsheet.

Normally, the initial weights and biases of a network are randomized before training begins, however, in order to ensure your program's outputs match the spreadsheet, you will need to use the same initial weights and biases provided in the spreadsheet. They are provided below for your convenience and highlighted in green in the spreadsheet.

$$W^1 = \begin{bmatrix} -0.21 & 0.72 & -0.25 & 1 \\ -0.94 & -0.41 & -0.47 & 0.63 \\ 0.15 & 0.55 & -0.49 & -0.75 \end{bmatrix} \qquad B^1 = \begin{bmatrix} 0.1 \\ -0.36 \\ -0.31 \end{bmatrix}$$

$$W^2 = \begin{bmatrix} 0.76 & 0.48 & -0.73 \\ 0.34 & 0.89 & -0.23 \end{bmatrix} \qquad B^2 = \begin{bmatrix} 0.16 \\ -0.46 \end{bmatrix}$$

The learning rate is $\eta = 10$.

The training data for this network consists of four input/output pairs divided into two mini-batches. Typically with stochastic gradient descent, we would randomize the minibatches between epochs, however, for this part of the assignment we will not do any randomization just to ensure your outputs match those in the spreadsheet.

The training data is as follows, where X_i is an input and Y_i is the corresponding output:

$$X_1 = \begin{bmatrix}0\\1\\0\\1\end{bmatrix} \quad Y_1 = \begin{bmatrix}0\\1\end{bmatrix} \qquad X_2 = \begin{bmatrix}1\\0\\1\\0\end{bmatrix} \quad Y_2 = \begin{bmatrix}1\\0\end{bmatrix}$$

$$X_3 = \begin{bmatrix}0\\0\\1\\1\end{bmatrix} \quad Y_3 = \begin{bmatrix}0\\1\end{bmatrix} \qquad X_4 = \begin{bmatrix}1\\1\\0\\0\end{bmatrix} \quad Y_4 = \begin{bmatrix}1\\0\end{bmatrix}$$

The data should be divided into minibatches where X_1 and X_2 are found in minibatch 1 and X_3 and X_4 are found in minibatch 2.

Running through 6 epochs (with stochastic gradient descent and back propagation) will result in the following outputs.

$$Y_1 = \begin{bmatrix}0.284737\\0.794477\end{bmatrix} \qquad Y_2 = \begin{bmatrix}0.686228\\0.227963\end{bmatrix}$$

$$Y_3 = \begin{bmatrix}0.230277\\0.832609\end{bmatrix} \qquad Y_4 = \begin{bmatrix}0.692288\\0.20266\end{bmatrix}$$

Do not worry about what this small network has learned, it is merely meant to serve as an example so you can get your neural network engine working with on a small set of data where you can check the outputs every step of the way.

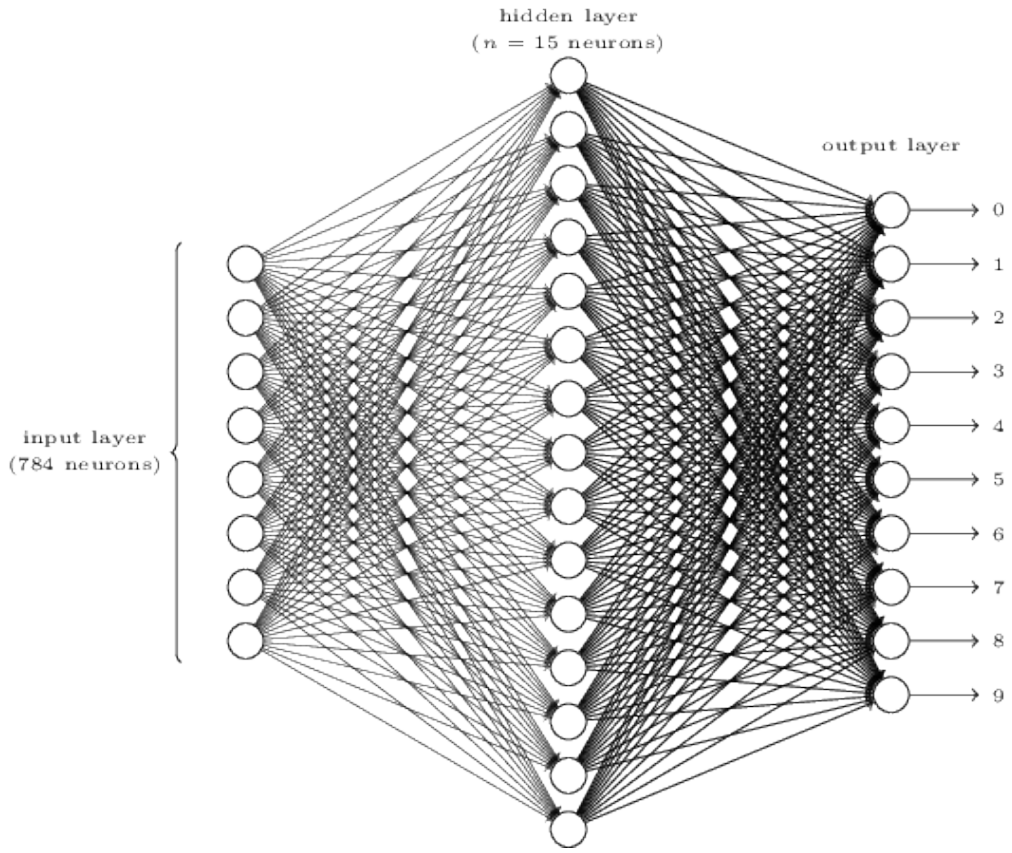
Part 2: MNIST Handwritten Digit Recognizer

Now that you have a working fully connected feed forward network and verified back propagation algorithms using SGD, we can expand the core Java program to recognize the MNIST digit set.

The architecture of this network will be as follows:

- An input layer with 784 nodes
- At least one hidden layer with 15 nodes
- An output layer with 10 nodes

This can be represented pictorially as follows:



The training and test data is provided at https://pjreddie.com/projects/mnist-in-csv/ and on canvas. The data was created by capturing images of handwritten digits. Each image was 28 pixels by 28 pixels. Each pixel was then converted to a grayscale value from 0 to 255.

Each line of data in the accompanying files has 785 values. The format of each line of data is

$$x = \left(label, pixel_{1,1}, pixel_{1,2}, \dots, pixel_{1,28}, pixel_{2,1}, pixel_{2,2}, \dots, pixel_{28,28}\right)$$

where $label$ is the digit 0 – 9 and $pixel_{ij}$ is a value from 0 – 255.

The label should be converted to a *one hot vector* with 10 elements so that it can be easily compared to the activations in the output layer. A one hot vector with 10 elements will have one value of 1 and nine values of 0 in it. Examples of the labels 7, 0, and 9 as one hot vectors follow:

$$7 \text{ is } \begin{bmatrix}0\\0\\0\\0\\0\\0\\1\\0\\0\end{bmatrix} \qquad 0 \text{ is } \begin{bmatrix}1\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} \qquad 9 \text{ is } \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\\1\end{bmatrix}$$

The 784 values for nodes in the input layer represent the 784 pixels in each image in the training data. For best results, you should scale these input values from 0 – 255 values to be a value between 0 and 1 by dividing each by 255 and storing the result as a double.

Each node in the output layer will represent a digit from 0 – 9.

Your program should come with a user-interface (a CLI is sufficient) that allows the user to select between the following options

1. **Train the network**

   In training mode, your program should iterate through the 60,000 item MNIST training data set. Some suggested parameters would be a learning rate of 3, minibatch size of 10, and 30 epochs. Alternatively, you can stop at a specified accuracy, such as 99% (instead of specifying the number of epochs). Have the weights and biases be generated randomly as values from -1 to 1.

   After each epoch completes, your program should print out information about that epoch including the following:

   a. For each of the 10 digits, the number that were correctly classified out of the total number of times that digit appeared. That may look something like this:

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

   b. Statistics concerning the overall accuracy of identification. That may look something like this:

   ```
   Accuracy: 49646/50000 = 99.292%
   ```

   As a note, the data above only used 50,000 items in the training set. You should

use all 60,000.

2. **Load a pre-trained network**

   Your program should be able to load a previously generated set of weights and biases from a file.

3. **Display network accuracy on training data**

   This option should only be available after selecting items (1) or (2) above (i.e. there is information about a pre-trained network).

   This option should iterate over the 60,000 item MNIST training data set exactly once, using the current set of weights and biases, and output the statistics shown in item 1 above.

4. **Display network accuracy on testing data**

   This option should only be available after selecting items (1) or (2) above (i.e. there is information about a pre-trained network).

   This option should iterate over the 10,000 item MNIST testing data set exactly once, using the current set of weights and biases, and output the statistics shown in item (1) above.

5. **Run network on testing data showing images and labels**

   This option should only be available after selecting items (1) or (2) above (i.e. there is information about a pre-trained network).

   While running the network on the testing data, this option should show a representation of each image itself, its correct classification, the network's classification, and an indication as to whether or not the network classified it correctly.

   Here is an example using ASCII art:

In the above example, note that it gives an option to hit 1 to continue to the next piece of data, and any other value will return to the main menu.

6. **Display the misclassified testing images**

   This option should only be available after selecting items (1) or (2) above (i.e. there is information about a pre-trained network).

   This option is similar to option (5) above, except it only shows the images that are misclassified by the network (instead of showing every image).

7. **Save the network state to file**

   This option should only be available after selecting items (1) or (2) above (i.e. there is information about a pre-trained network).

   Your program should be able to save the current set of weights and biases to a file.

0. **Exit**

   You should be able to exit the program.

Additional Requirements

1. Your code should be able to run via the command line (i.e. compiled with the javac command and run with the java command).

2. External libraries are not allowed. That is, if you need to do matrix multiplication within your program, you need to implement the source code for how that multiplication works yourself. You are allowed to use java.util.

3. You must thoroughly document your program. That is, you must include comments and documentation all through out the program. Use the comments to explain the purposes of you classes and functions.

4. You must include your name, date, and a description of the assignment at the top of the file as well. Lack of comments can cost you up to 20% of your overall grade for this program.

5. Grading will be done through code interviews. You will meet with me for approximately 15 minutes to demo your program and explain how it works. Be sure you have your laptop with you to run the demo. Also, there is no reason to panic about the meeting, if you wrote the program it will go very smoothly. If not every part works, it is ok to explain to me what doesn't. I value your honesty.

6. This is an individual assignment. All the work should be yours, not your friend's, not an LLM's, not someone from stack exchange. If you need to look-up information on the internet to refresh your memory of how to accomplish certain functionality in java, do cite it.

   For example, consider keeping a work-log/citation file where you can indicate what you had to search for, where you found similar example, what those examples were and what may have been useful (i.e. what you found). You can cite the work log as comments in your code.

   It is far better to turn in partially working code that you can explain what it does than to turn in code that is perfect, that you didn't write, that you cannot explain.