Project Report

You Ren Chen

CSCE 420 500

Report

Problem Statement and Significance:

- The purpose of this project is to implement our own Neural Network using back propagation. As stated in the project statement, the Neural Network will be given a set of inputs that represents 5x7 ASCII capital letters. Training will be done using a set of given test data. Then network must identify what the letter is as well as be able to identify letters with distorted pixels. The program must also be able to take in user input and guess what letter it represents.
- This project is our first practice of implementing deep learning. Having never done this before, this is a great insight into applications that have real world implications. Not only did this project push us to learn and understand Artificial Neural Networks, it also showed us how such a useful program could be used to solve practical problems.

Restrictions and Limitations:

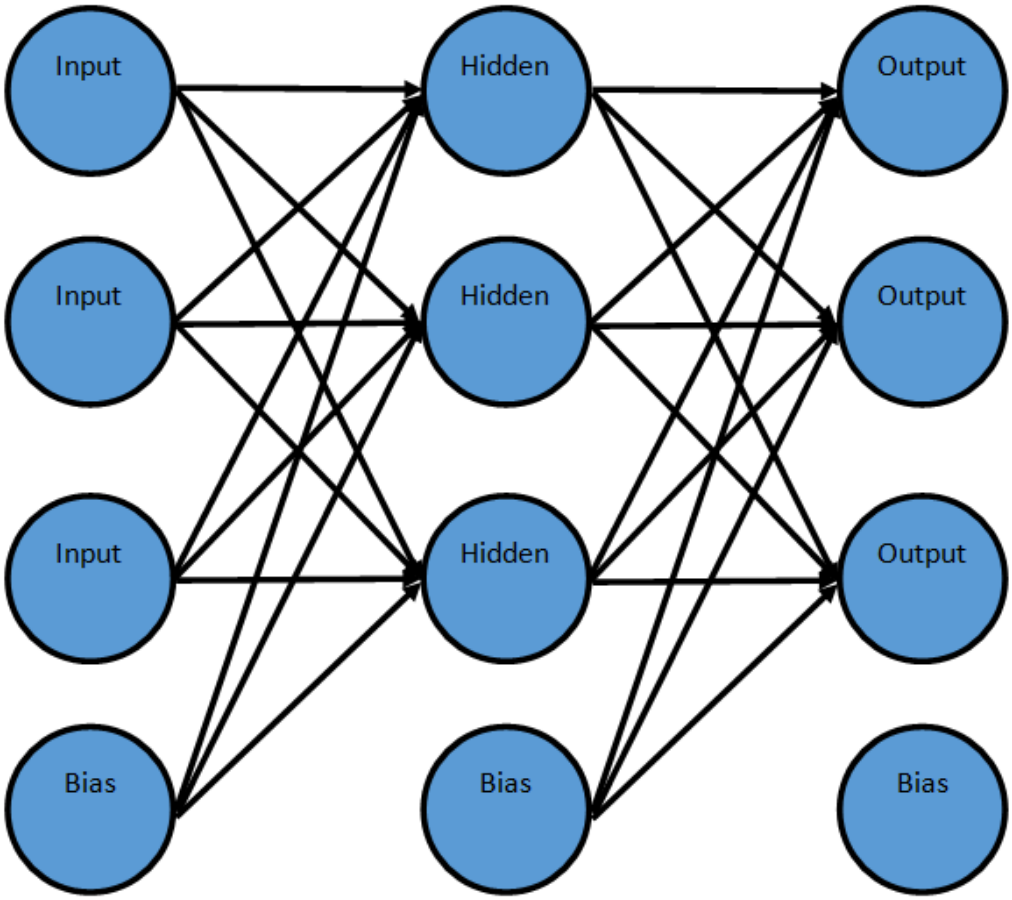
- For this project, we are not allowed to use pre-built Neural Networks, as we are to implement our own. The program has to be able to run on compute.tamu.edu and follow the given parameters specified by the project document.
- The Neural Network I have written does not save the weight values or the delta weight values, meaning that the program forgets its training once it ends. However, due to the size of the network, training does not take long. There is limited input checking whenever the user is prompted for input parameters, please note that all user input has to follow the format of examples given or risk unintended behavior.

Explanation of Approach:

- I approached the problem in a similar manner described by Dr. Daughterly in class. I also had to do some external research in order to understand the details of back propagation. The reason for this is due to the ease of finding relevant information.
- The Neural Network consists of two components, the Neural_Network class and the Neuron class. Within the Neural_Network contains the error_value, which is the RMS error between the target and the output, as private variables. The Neural_Network is constructed by inputting a 2D vector that specifies the number of layers and the number of neurons within each layer. An extra neuron is added to each layer of neurons as the bias neuron.
- First a vector of inputs is passed as the arguments into the feed_forward function. The function sets the outputs of the input layer as the passed in arguments. Then the function calls the feed_forward functions of each individual neuron, passing in the previous layer. This allows the neurons to access the previous layer's output values. The previous output * weight values are summed, then passed into the hyperbolic tangent transfer function to replace the current neuron's output.
- For the network to be trained, it is necessary to have a back propagation function. The function takes the "correct" output and compares it with the network's output. Note, the RMS error is calculated for documentation's sake. The gradient for each of the output

nodes is first calculated by multiplying the derivative transfer function with the difference between the output and "correct" output. Then the hidden layers' gradients are calculated before the weights of each node is updated. Please view the commented source code for the Neural Network and Neurons for details. To summarize, the program uses gradient descent to train its neurons.

- The update function, which is how the network actually learns, employs a momentum function to alter its weight. The new delta weight is calculated by multiplying the learning rate with the output and gradient plus the momentum times the old delta weight. The actual weight of the connection is summed with the new delta weight to calculate the new weight.
- The get_result function simply returns the output of the network as a vector of doubles. Below, is an image representing the simplified structure of the network.



Sample Run:

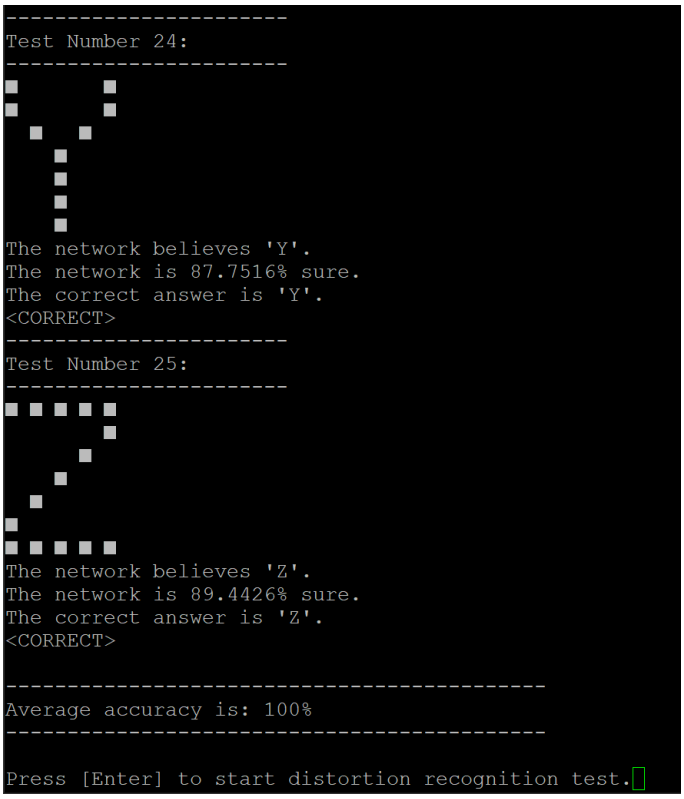
- The program starts by creating the Neural Network and stating the required average training accuracy.

```
[yourenchen]@compute ~/Senior/CSCE_420/project> (13:19:09 04/23/18)
:: make run
./a.out
-----------------------------------------
Welcome to my project Neural Network!
-----------------------------------------

Creating Neural Network...
Done!

>Default training accuracy is set to 96%.
Press [Enter] to start training.
```

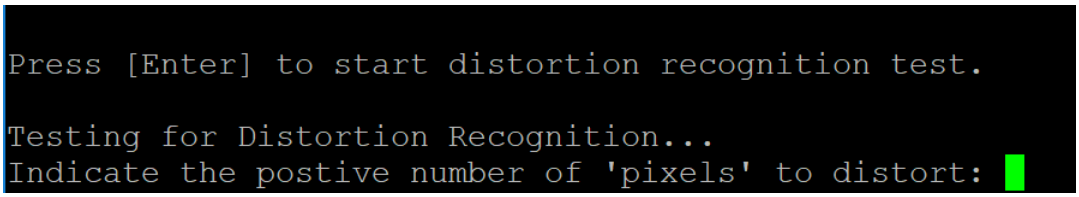
- The network will train itself given a set of predefined input and output data. Each average is calculated from the sum of 26 training sessions. The training stops once the desired accuracy is reached.

```
[194] Average Training Accuracy: 95.3425%
[195] Average Training Accuracy: 95.3912%
[196] Average Training Accuracy: 95.4394%
[197] Average Training Accuracy: 95.4871%
[198] Average Training Accuracy: 95.5343%
[199] Average Training Accuracy: 95.581%
[200] Average Training Accuracy: 95.6272%
[201] Average Training Accuracy: 95.6729%
[202] Average Training Accuracy: 95.7181%
[203] Average Training Accuracy: 95.7627%
[204] Average Training Accuracy: 95.8068%
[205] Average Training Accuracy: 95.8503%
[206] Average Training Accuracy: 95.8934%
[207] Average Training Accuracy: 95.9358%
[208] Average Training Accuracy: 95.9778%
[209] Average Training Accuracy: 96.0191%

Done Training!

Press [Enter] to start standard testing.
```

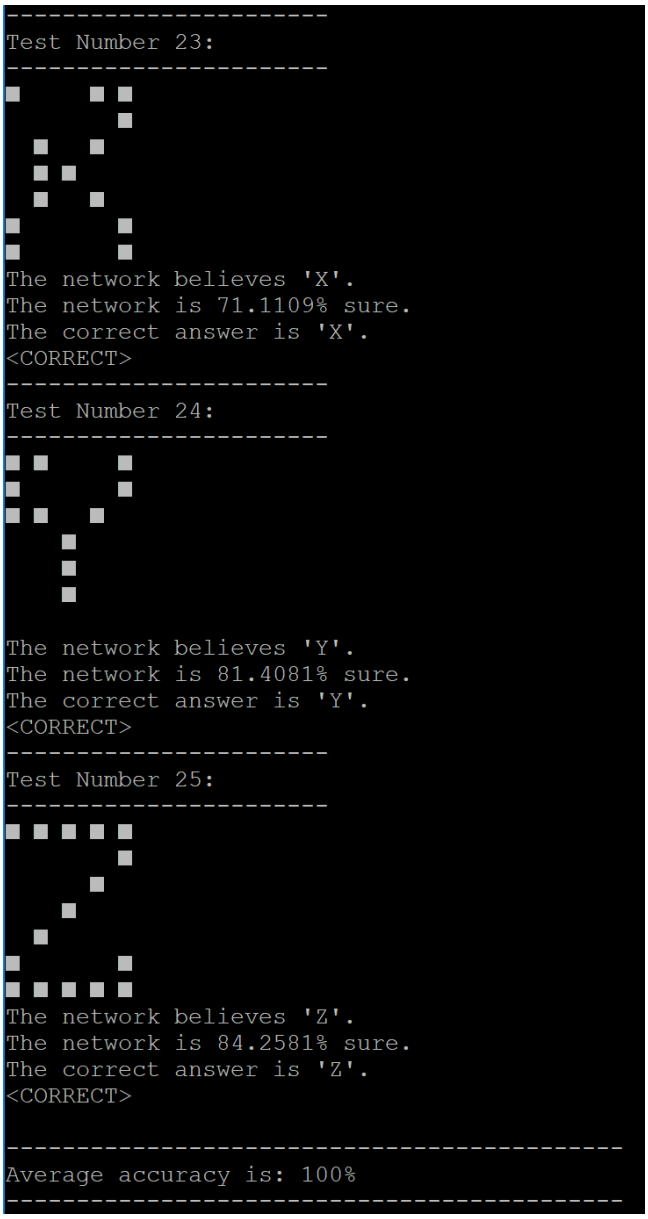
- Next, a set of standard testing will be done, this is simply feeding the network an unaltered version of the training data to check whether it has learned.



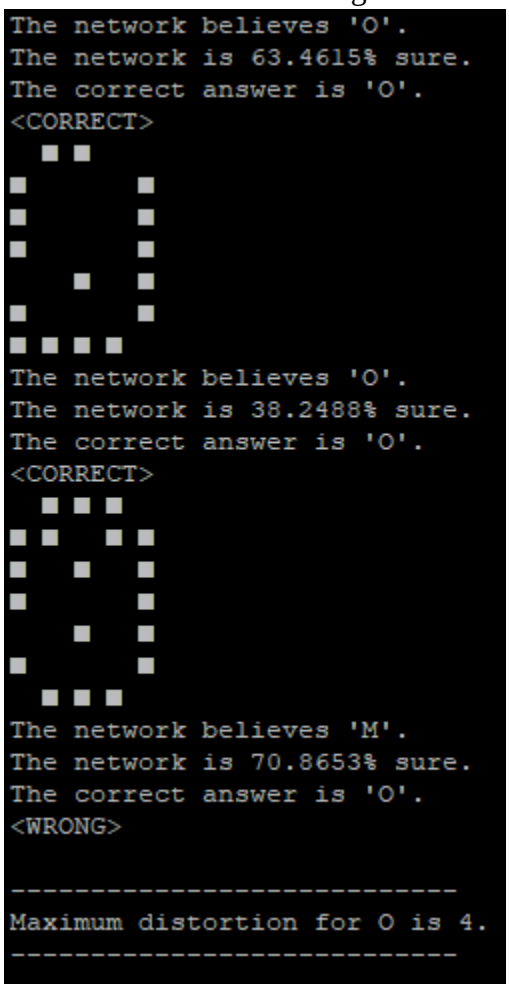
- The user is not prompted for a positive integer less than 35 as the number of pixels distorted.



- The distorted input will be fed to the network and the network will attempt to identify the given letters.



- The maximum distortion test the tolerance for noise of each letter. The letter is continuously distorted until the network fails to recognize it.



- The user will be prompted to input 5 separate integers. These integers will be converted to a usable input to the neural net. The data is then fed to the neural net and the output is displayed.

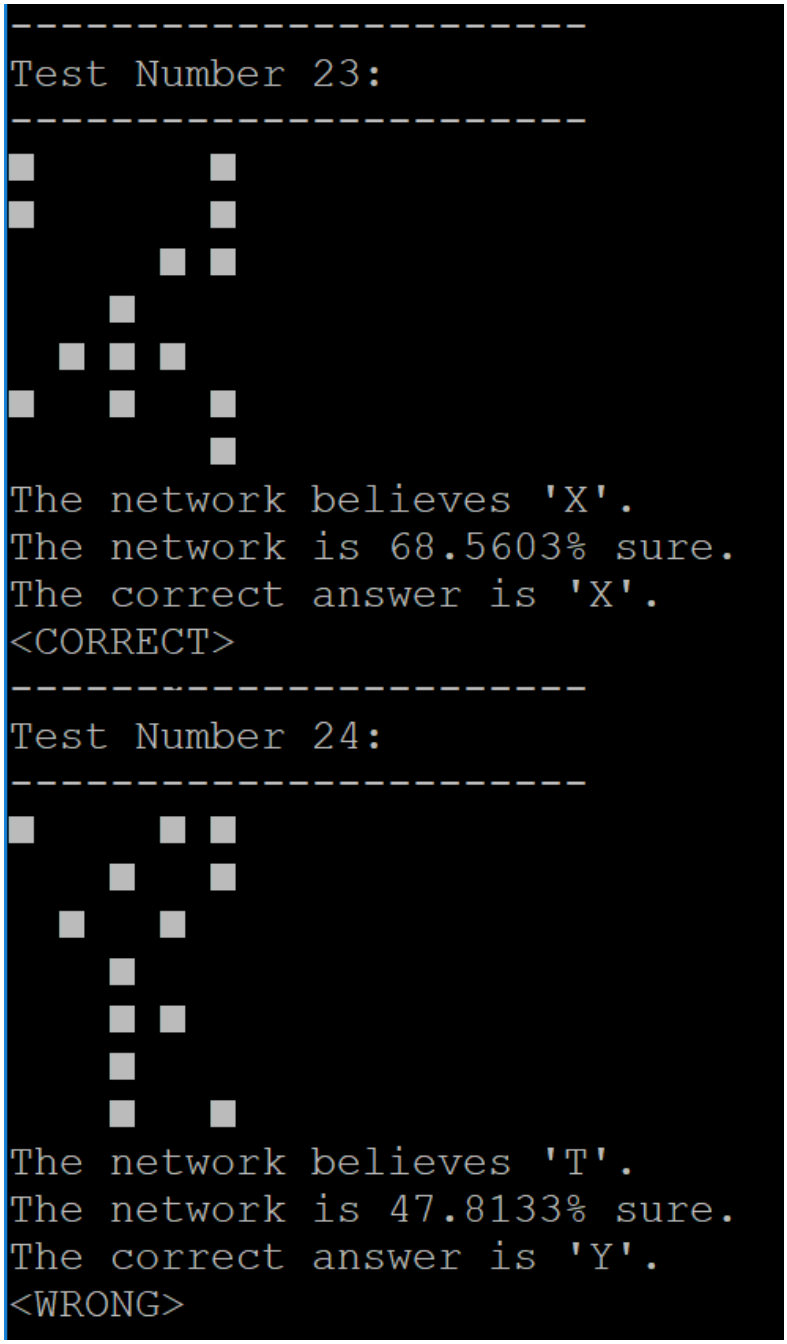
```
Press [Enter] to start user input tests.


Example: 126 17 17 17 126 represents 'A'.
Be sure to press [enter] between each integer.
Input a negative number to quit.
Please input 5 decimal numbers that represent the letter:
[1] 126
[2] 126
[3] 17
[4] 17
[5] 126

--------------------------------
User Input
--------------------------------
    ■ ■
  ■ ■   ■
  ■ ■   ■
  ■ ■   ■
  ■ ■ ■ ■
  ■ ■   ■
  ■ ■   ■
The network believes 'A'.
The network is 77.831% sure.

--------------------------------
```

- The program will continuously loop for user input until the user gives the program a negative number.

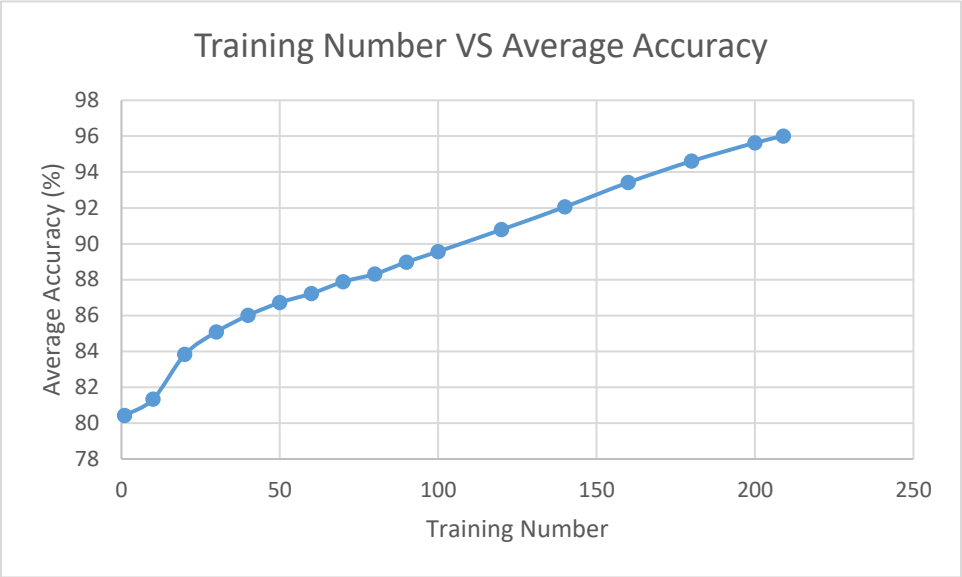
```
Input a negative number to quit.
Please input 5 decimal numbers that represent the letter:
[1] -4

Quitting...

[yourenchen]@compute ~/Senior/CSCE_420/project> (13:34:39 04/23/18)
::
```

Results and Analysis:

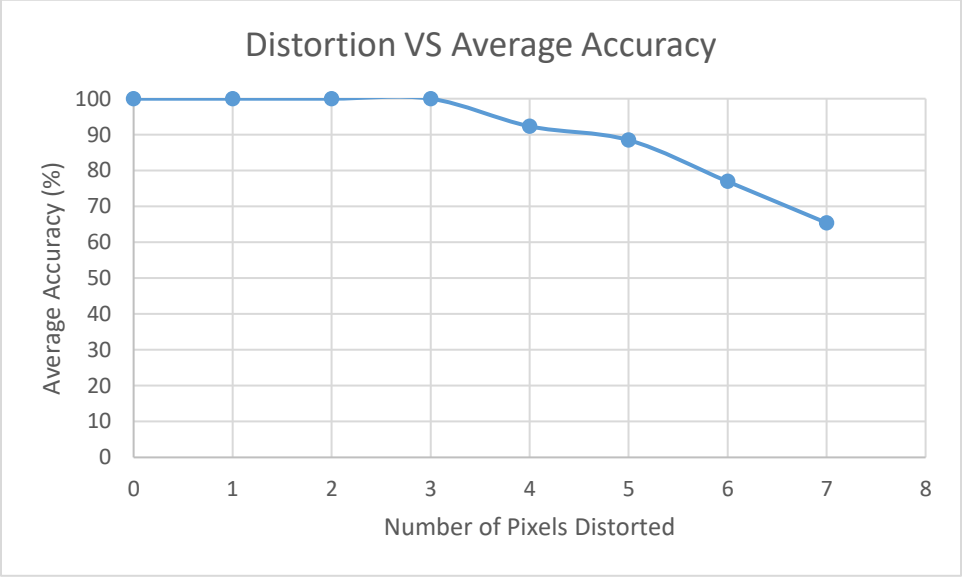
- Overall, the neural network is fairly accurate. What I've found while experimenting is that even when the input is extremely distorted, the neural network is still able to make good deductions.
- In the example to the right, about 5 pixels are distorted from the letter and yet, the network is still able to identify 'X'. It did fail to identify the distorted letter 'Y', but that is to be expected due to the relatively high value of distorted pixels.
- I'm not exactly sure what relationship the network formulates, as it can identify some of the letter I fail to identify, at higher distortions. Still, it's very interesting how the program made such connection between the individual pixels.



- Below is a plot of the training data against the average accuracy of the neural network.



- Below is a plot of the average accuracy resulting of a varying number of pixel distortions. The letter is essentially unrecognizable beyond 8 pixels, and the results seem to be basically guessing. Therefore, the data is neglected from the graph.



Conclusions:

- This project was extremely interesting as it gave me my first experience into deep learning. During the process of creating the Neural Network, I've experienced many strange errors. The segfaults were unpleasant but relatively easier to track down than logical error, which required the majority of my programming time to find and fix. The issue is with the input to the transfer function. I did not scale the input to the hyperbolic tangent function initially, which caused the output to be indistinguishable from 1, which resulted in the program never learning. Once that was fixed, the program works perfectly fine. With the insight I have received form this project, I hope that I will have an easier time when I eventually explore other implementations of deep learning while expanding my knowledge of artificial intelligence.

Future Research:

- My implementation of the Neural Network could be optimized. Since I've only tested with a small number of neurons, runtime is not an issue. But with the increase in complexity of the application, I may find it necessary to multithread the forward and back propagation methods. How this could be done will require some research and creativity.
- The network current does not remember what it has learned from session and must be trained each time before usage. Although the basic implementation is simple, I need to research on efficient methods of giving the network memory.
- There are also other implementations of Neural Networks that exist, each with their own positives and negatives. In the future, I wish to look into the other possible methods to improve my Neural Network implementation.

Instructions:

- To run the program, first ensure you and in the proper file directory.
  1. Compile the program by typing the “make” command
  2. Then to run the program, type “make run”
  3. The program will display the default desired training accuracy, this can be changed in program. To progress, simply press [enter].
  4. The program will run its training sessions until the desired accuracy is reached. The user will be alerted with the “Done Training!” message. Press [enter] to continue.
  5. A set of standard tests will be displayed. The input data for the tests are taken directly from the training data to check whether the program has actually learned. Press [enter] to proceed to the distortion test.
  6. The user will be prompted to submit a positive integer less than 35. This specifies the number of pixels out of the 35 total pixels the user wants to distort. Note that the pixels are chosen at random thus the same pixels may be flipped multiple times. Input the desired pixel distortion number and press [enter].
  7. The results of the distortion test will be displayed in a manner similar to the standard testing. Press [enter] to proceed to the maximum distortion tests.
  8. The user does not need to input anything as the test will stop once the neural net is wrong. Press [enter] to proceed to the user input test.
  9. Now, the user must input five integers, each equating to the values of 7 binary bits. The format is much like the data given in the 5x7font array, except it uses decimal number instead hexadecimal. Once the five numbers are inputted, a letter will be created from those values and fed to the Neural Network with the results.
  10. If at any time during the user input section, the user wishes to quit, inputting a negative number will cease the looping and terminate the program.

Bibliography:

- Nielsen, and Michael A. “Neural Networks and Deep Learning.” Neural Networks and Deep Learning, Determination Press, 1 Jan. 1970, neuralnetworksanddeeplearning.com/chap1.html.
- Nielsen, and Michael A. “Neural Networks and Deep Learning.” Neural Networks and Deep Learning, Determination Press, 1 Jan. 1970, neuralnetworksanddeeplearning.com/chap2.html.
- Nguyen, Vinh. “Neural Net in C++ Tutorial on Vimeo.” YouTube, YouTube, 9 Mar. 2013, www.youtube.com/watch?v=KkwX7FkLfug.