Matthew Stevens
UIN: 924000693
Dr. Daugherity
CSCE 420 – 500
25 April 2018

Project Report

1) **Problem Statement/significance:** We were tasked with creating a neural network that could determine binary or hex input of a 5x7 font grid. This neural network needed to have a 90% accuracy when identifying a letter with 1 bit of noise (when a bit is incorrect). This project demonstrates the basics of deep learning. Deep learning neural networks are modeled after the human brain and understanding how a neural network works allows us to understand how the human brain works. This unlocks many additional opportunities or choices for humans.

2) **Restrictions and Limitations:** The restrictions of my neural network involves speed and memory. My program has a large quantity of vectors, and a different solution may be more optimal in terms of memory storage. Additionally, I store a large quantity of values as it is being used. Using a program that does not store any unnecessary values sill increase the speed. Another approach may use less storage and would be faster in the end. My program is restricted and limited to capital alphabet letters.

3) **Explanation of my Approach:** My approach involved two layers of weights and two activation vectors. The first step was to take the 35 length binary string (could be hex but will ultimately be converted to this string) and "fan" out the input to the hidden layer. The hidden layer then propagates the input through by using the initial weights (between -.1 and .1 but not 0) and calculating the dot product. Optionally, we can add bias to the dot product. Next, tanh() the results to obtain 35 weights that are between -1.0 and 1.0. These weights are used for the output layer's input. Next, I calculate another dot product and apply the sigmoid function to obtain a second activation input. I calculate the derivatives and apply them (as well as the learning rate) to all the weights. I adjusted my learning rate to take half steps (.5 scalar) because I wanted the weight steps to be small to obtain a more accurate result. I have my code set to run 10,000 epochs for training before evaluating the input. Afterwards, it tests for bitflips on each letter and plots the accuracy vs epoch graph using gnuplot. Overall, my neural network consisted of two layers of weights and two activation vectors, along with many unimportant vectors!

4) **Sample Run:** Below is an example run using hex values:
here is the compile command:

```
ttizrawr]@compute ~/420/project> (22:40:39 04/23/18)
g++ -std=c++11 neural.cpp neuralNetwork.cpp
```

here is the example input:

```
[mattizrawr]@compute ~/420/project> (16:44:56 04/23/18)
:: ./a.out 0x7E 0x11 0x11 0x11 0x7E
Beginning Hex
converted
done populating
training the neural net:
completed training
beginning input analysis
the input is: A with a 0.999996 certainty
```

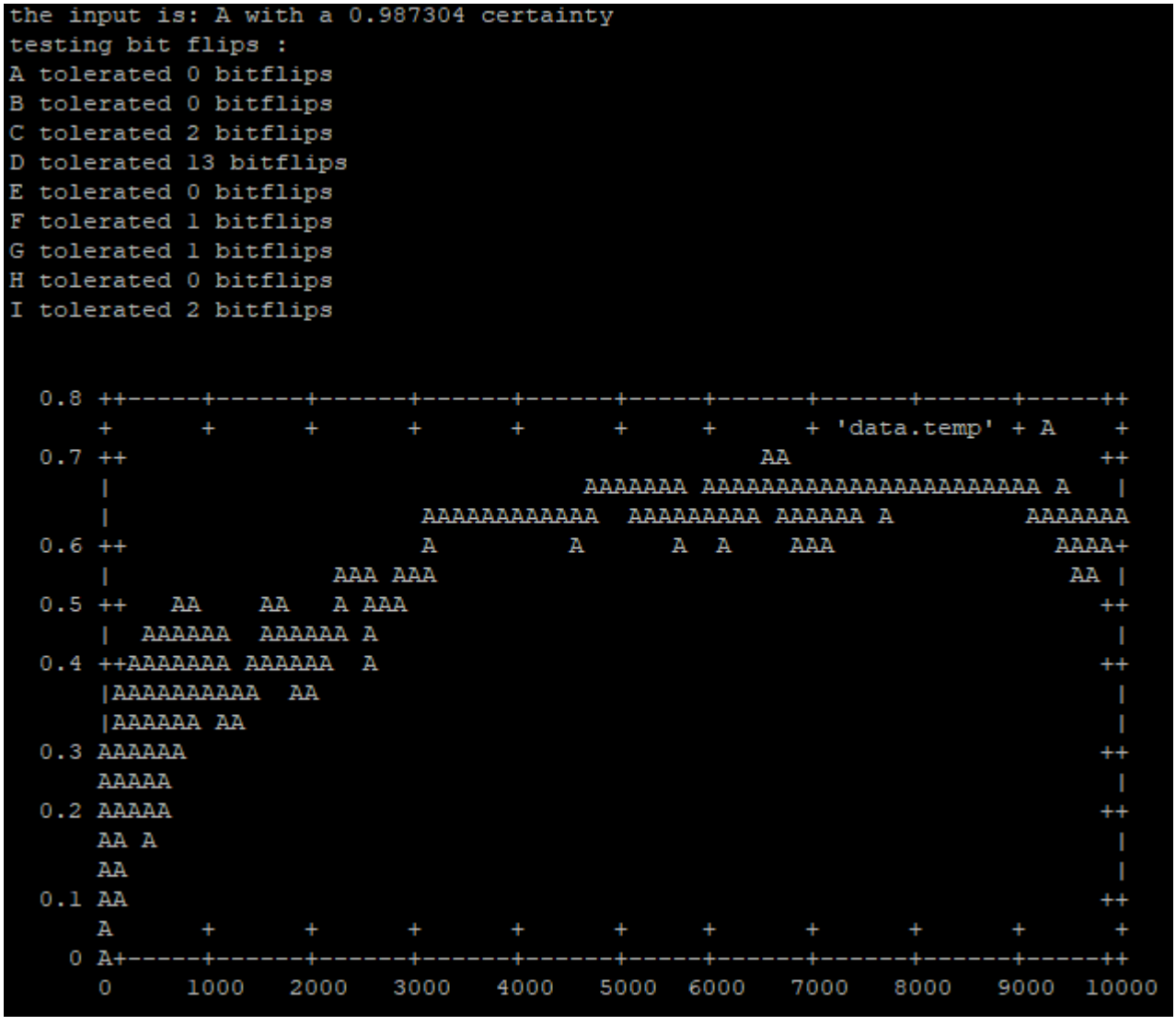
here is an example input with binary:

```
[mattizrawr]@compute ~/420/project> (22:43:35 04/23/18)
:: ./a.out 11111100001011000101100010111111110
done populating
```

Examples of the output may be seen in section 5.

5) **Results and Analysis:** After training this my neural network, I was able to achieve a 98.7% accuracy when attempting to identify the correct input for A. Sometimes, the graphical gnuplot does not allow for the rest of the bitflips to be seen (printed over them). Surprisingly, some letters cannot maintain their identities when they have noisy input of just 1 bit.

```
the input is: A with a 0.987304 certainty
testing bit flips :
A tolerated 0 bitflips
B tolerated 0 bitflips
C tolerated 2 bitflips
D tolerated 13 bitflips
E tolerated 0 bitflips
F tolerated 1 bitflips
G tolerated 1 bitflips
H tolerated 0 bitflips
I tolerated 2 bitflips
```

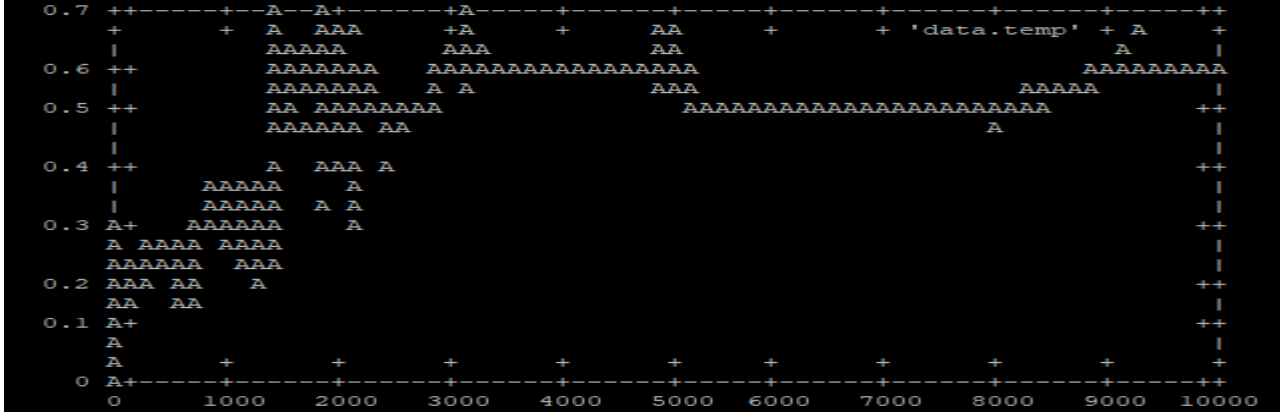


After obtaining this output, I attempted to check for consistency. After investigation, my initial weights were determining whether my letter succeeded or not. I tried investigating the steps too see if they were too large, but even when scaling the learning rate down to .01 it still had problems. From observing the data, the neural network has a long chain of consistent accuracy, so I am led to believe that the accuracy is not properly being reset. This is seen below:

```
4388 0.423077
4389 0.423077
4390 0.423077
4391 0.423077
4392 0.384615
4393 0.384615
4394 0.384615
4395 0.384615
4396 0.346154
4397 0.346154
4398 0.346154
4399 0.346154
4400 0.346154
4401 0.346154
4402 0.346154
4403 0.346154
4404 0.346154
4405 0.346154
4406 0.346154
4407 0.346154
4408 0.346154
4409 0.346154
4410 0.346154
4411 0.346154
4412 0.346154
4413 0.346154
4414 0.346154
4415 0.346154
```

This most likely occurs because the weight shifts are very small. Another potential reason is that the accuracy is not properly being reset. Even with the redundant calculations, my neural network is able to maintain 1 bit of noise on A with a .996% certainty. This is seen below:

```
sig activation 0.044841
sig activation 0.037553
sig activation 0.026603
sig activation 0.014668
sig activation 0.001838
sig activation 0.011894
sig activation 0.010874
sig activation 0.026132
sig activation 0.002114
sig activation 0.000154
sig activation 0.000549
sig activation 0.037172
sig activation 0.027769
sig activation 0.011741
sig activation 0.000911
sig activation 0.044483
the input is: A with a 0.996538 certainty
testing bit flips :
A tolerated 1 bitflips
B tolerated 0 bitflips
C tolerated 0 bitflips
D tolerated 2 bitflips
E tolerated 0 bitflips
F tolerated 0 bitflips
G tolerated 0 bitflips
H tolerated 0 bitflips
I tolerated 0 bitflips
```



My neural network is not very accurate during the training, but when it obtains results, the results are decent (90% or higher accuracy) after 10000 epochs.

6) **Conclusions:** I implemented the correct structure for a neural network, however, the initial weights or some weights while propagating may be incorrect. This can be observed by the looking at the accuracy data. Neural networks are very useful in for many different tasks, such as web-crawling, data mining, and deep learning. Without this project, I probably would have very little knowledge of how a neural network is executed and structured.

7) **Future Research:** Deep learning neural networks are very useful for modern-day analysis. Instead of font recognition, we can improve our neural network by receiving input from a camera or an image and use the data for emotion recognition based off body language, features, shadows, and other aspects. This will allow for a computer to recognize an emotion and perhaps modify its course of action based off the emotion that the neural network sensed. This would be useful for targeted advertisement. Additionally, neural networks can be used to solve difficult problems that humans cannot possibly calculate in the amount of time given. An example of this would be an AI for chess, where the AI learns chess first then plays the grandmaster and wins.

8) **Instructions on execution of the program:** To execute the program, compile with "g++ -std=c++11 neural.cpp neuralNetwork.cpp" on compute.cse.tamu.edu and run with "./a.out" followed by either a 35 bit long binary string or 5 separate hex values as seen in the sample run.

9) **Source Code Listing with //comments:** Will turn in neural.cpp, neuralNetwork.cpp, neuralNetwork.h on paper.

10) **Bibliography:**

Russell, Stuart J., et al. *Artificial Intelligence: a Modern Approach*. Prentice Hall, 2010.