Cheng Chen

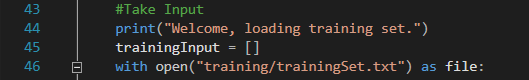
UIN: 421008518

CSCE 420-500 Dr. Daugherity

**Final Project Report**

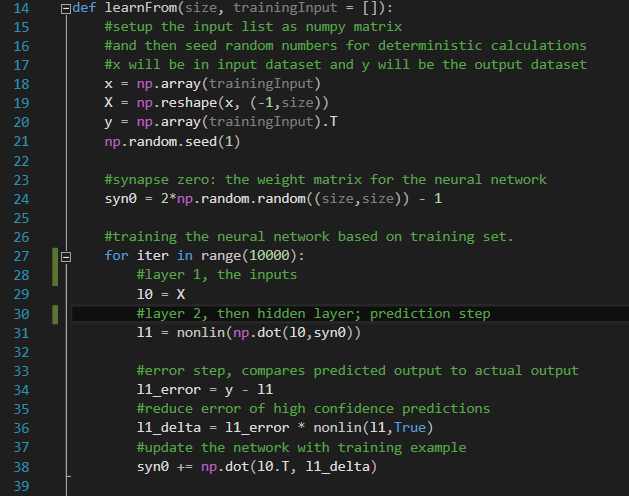
For the final project, I was to design and write a computer game program using a neural network. The object of the game is to keep the computer from guessing the player’s next choice of two possible values. A neural network is a computational data model that is able to represent a problem in an input/output model. A neural network resembles a human brain in that it acquires knowledge through learning, and its knowledge is stored within synaptic weights which connect then neurons. A neural network’s strength lies in its ability to represent both linear and non-linear relationships and learn directly from the modeling of the data. A neural network learns through back propagation, where input data is repeatedly presented to the network, in each iteration the output is compared to the desired output and an error is computed. The error is fed back to the network and weighs are adjusted so the error decreases with each iteration and the network will ‘learn’ and approach the desired output.

The game first takes 32 inputs in the form of a preset “TrainingSet.txt” text file for purposes of ease of access and demonstration. This can easily be modified to a dynamic user specified file. Afterwards, the neural network will attempt to predict the player’s next 32 inputs. Some limitations I have found when testing is that it is possible to “fool” the neural network by using a biased training set such as 31 0’s and one 1. In such a situation, the network will commonly predict the player’s input as 0, even if the player is putting ones as their first few inputs. On a less biased training set, the accuracy of the predictions seem to be more leveled out in a range of roughly 40% to 60% , which is on average, roughly 50% correctness.

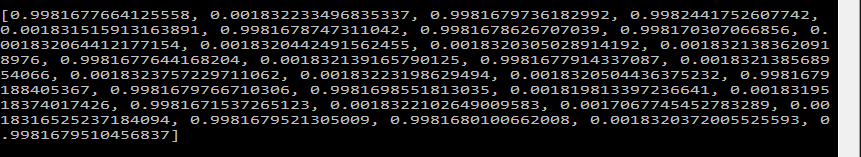
If the user has their own training set they would like to use, they can create a 32 line text file with each line being a 0 or a 1, and save the file under the training directory. 

Next, the user is required to modify like 46, changing trainingSet.txt to filename.txt, where filename is whatever the user named their custom training set.

In this project, I used the post “A Neural Network in 11 lines of Python(Part 1)” on iamtrask’s blog, <http://iamtrask.github.io/>, to help me develop a neural network for the game. I wrote a two layered neural network. Listed below is the code.



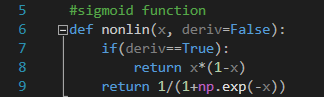
I will refer to the two layers, as l0, the input layer, and l1, the hidden layer. The neural network initially has a setup of 32—32, this is because each training set consists of 32 inputs and the hidden layer gives 32 potential outputs. So on the initial pass, the neural network will take the training set of 32 numbers and return the next 32 possible values, as shown below in a modified sample run where I added a print statement to show the output of the neural network.



The amount of neurons in each layer will grow as the program runs as it takes each of the player’s inputs into the training list, and as the neural network learns from the new inputs, the output values will also change. The reason why I ended up with a 32-32 implementation is because my original attempt at using a 32-1 implementation resulted highly inaccurate predictions where the computer only predicted 1’s, as a result I found a 32-32 implementation to be more accurate.

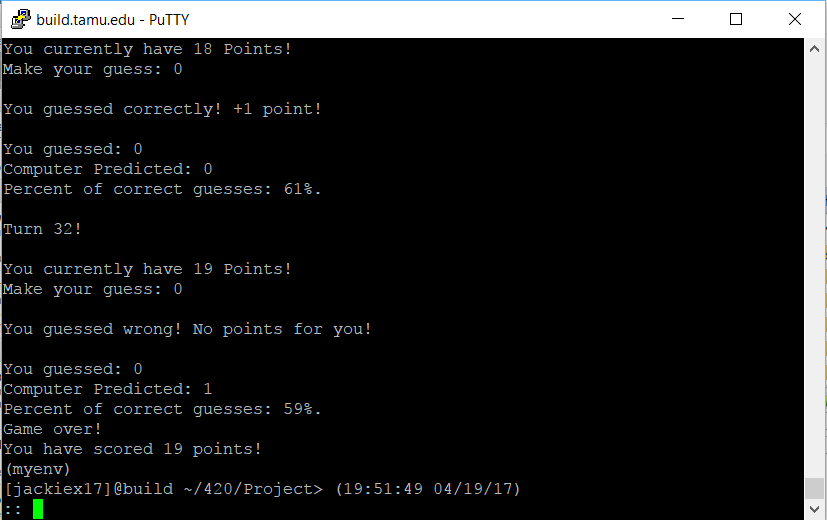
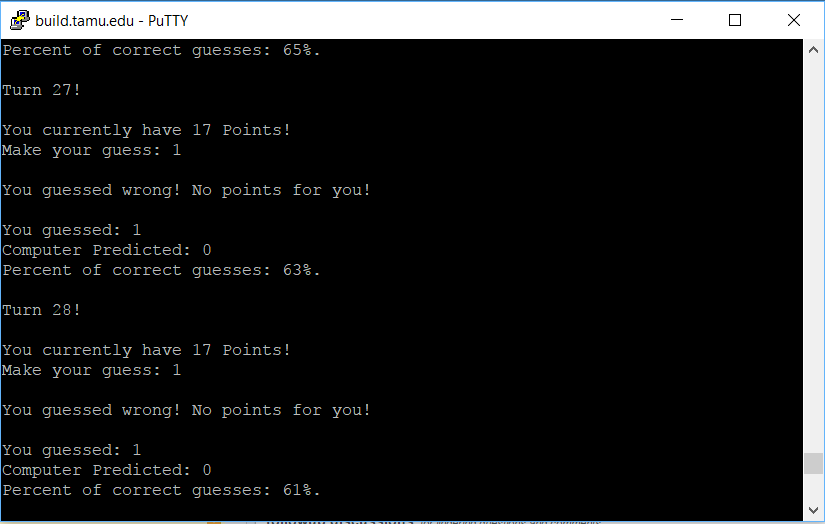
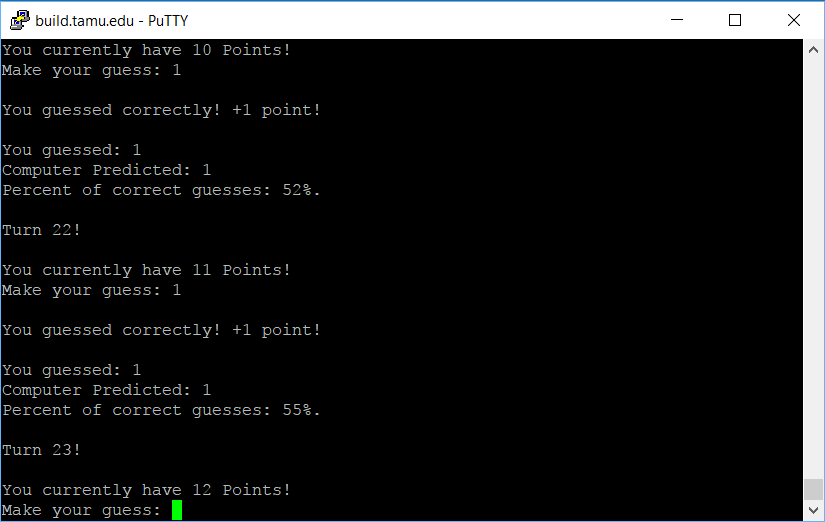
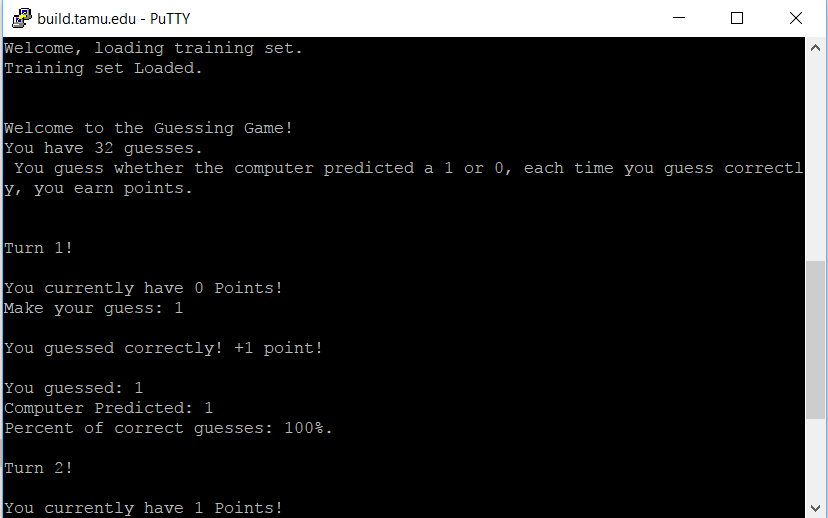
Walking through the algorithm above, it takes in a training list, and the size of the training list. Reshape is used to reshape the list x into ‘size’ inputs, and ‘y’ will be a transpose of the x list, representing a desired output pattern. Syn0 is the weight matrix that connects the two layers, input and output, of this neural network. The dimension of the weight matrix is (size, size).

Afterwards, starting from line 27, we start to actually train our neural network through back propagation with a total of 10000 iterations. In the backward propagation loop, we first try to predict the output on line 31, it does so by multiplying the l0 by syn0, and passing our output to the sigmoid function(listed below).



Next we then compare the predicted output to the desired output, y, and try to adjust the weights of syn0 accordingly. This is done by first calculating the error, which is then multiplied elementwise with the ‘slopes’ of the prediction given by the sigmoid function. These steps can be found on the lines 34 and 36 respectively. Lastly, on line 38, we updates the weights of synapse0 accordingly to decrease the margin of error and make more accurate predictions in future iterations of this training loop.

A few screenshots of a sample run of this game with the default training set, “trainingSet.txt”.



The results of the neural network can be broken down into three distinct cases with different results: an unbiased training set with a random player, a biased training set with a random player, and an unbiased training set with a biased player. Given an unbiased training set with a random player, the accuracy by the end of the testing phase can be as low as 41% sometimes, or as high as 60% in other runs, this however averages out to roughly 50% given enough runs. Given the case of a biased training set, such as trainingSet2 or trainingSet7 in my test runs the prediction accuracy plummets if suddenly given a random player, usually averaging at around 35% accuracy by the end. However, given a random player with an unbiased training set, I’ve found the neural network would suffer a few inaccurate predictions at first sometimes, dipping to an accuracy of roughly 33-35%, before it starts to climb and reach an accuracy of roughly 50% near the end. Lastly, given an unbiased training set with a biased player, such as someone who only selects 1’s, on the other hand, also seems to have an interesting effect on the accuracy of the predictions. It seems as if the neural network will catch on, and initially have a high accuracy in the 60-70% area, but then it starts to drop as it expects the player to select 0, before settling with an accuracy of ~45-47% by the time it reaches the last prediction. As an addendum to this analysis, an interesting case, when if the training set were in a pattern like say trainingSet4, which is “01010101…” and the player continues this pattern, the network will have a 100% prediction accuracy, this can also be seen if one were to use an unbiased, random training set, then make their choices exactly the same as the unbiased set.

In conclusion, I have developed a simple 2-layer neural network that, given an unbiased, random input, and a random player, can accurately guess what the player selects. The neural network also is able to learn and recognize patterns such as “11001100…” and predict what follows in the pattern. I’ve learned what back propagation is and how to implement it when I decided to write my own neural network and back propagation training for this project. To run this code on the build.tamu.edu server, first create a python virtual environment, then call:

source ./myenv/bin/activate

Followed by:

python3 FinalProject.py

More detailed instructions to run this project can be found in the ReadMe.txt file.

**Bibliography**

**A Neural Network in 11 lines of Python (Part 1). (n.d.). Retrieved April 19, 2017, from http://iamtrask.github.io/2015/07/12/basic-python-network/**