Preston Stepic

625004438

CSCE 420-500

Project

Due: April 25, 2019

Problem & Significance

In this project, I will be discovering different solutions for a program representing a neural network. This neural network will be trained to recognize letters on a 9x14 bitmap.

Functionally, the significance of this project is to accurately predict what a letter is depending on the bits on the 9x14 bitmap inputted, even when the input does not represent a perfect letter. Additionally, however, this project will give me a better understanding of neural networks. It will teach me how to implement different layers of neurons with weights, as well as, functions to forward calculate output values and backward propagate output values.

After implementing a neural network in this project, I will use the knowledge gained to implement machine learning in other problems with similar mechanics of hidden layers and back propagation. It is important to learn about machine learning because it is such a powerful computing tool and can be used to solve an infinite amount of problems.

Restrictions and limitations

Implementation restrictions:

For this neural network, only one bitmap will be learned: a 9x14 map. Any other sized maps or fonts will fail in the network. Additionally, the network will only be trained on perfect letters, meaning that every bit will be in the exact same location every time. This means if the letter is still perfect but shifted around the map, the network will have no way of recognizing the character and will give incorrect results most of the time. Next, only capital letters will be trained in the network. The program will not be able to recognize lower-case characters nor symbols. In this implementation, the network can only solve for 26 characters in the font (upper-case letters) which leaves a lot of characters unaccounted for and possibly used as an input.

Machine restrictions:

This neural network will be programed, trained, and tested on my Dell Precision 5510 laptop. Although this is a powerful machine, it still has its limitations. Adding, layers of neurons in a neural network adds dimensionality to the program and can provide better results. However, whenever a new layer is added it creates more neurons with weights that must be looped through every time a forward calculation or back propagation is called. So, adding layers of neurons may increase the robustness of the network to give correct answers a higher percentage of the time but it also comes with a higher cost of time efficiency. I predict my laptop will be able to handle a reasonable number of hidden layers of neurons and will provide accurate enough results with the layers and number of neurons it can handle in a reasonable time.

Approach

Initial setup:

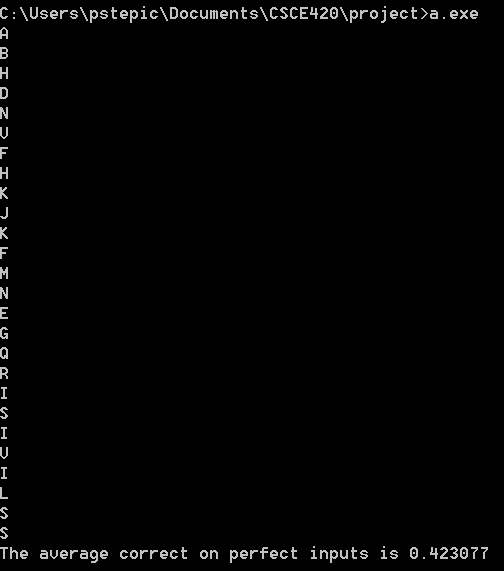
To import the bdf bit maps, I used a separate C++ program to act as a parser through the .bdf file. This program exported the hex values of the map to a .csv file in a list. I then copied these lists into an unordered map with key of the letter they represent. Then within my project program, I converted each of these hex values into a binary string of 8 bits and added an extra 0 to the end because the last column of the 9x14 map is always empty. Next, I was able to take those strings of binary strings, representing a row, and split them into 126 individual bits which was then used as input into the neural network.

In the beginning of my testing, I used the values provided in the project document as my values for my weights and learning rate. I initially set my weights to a random number between -0.1 and 0.1 and used 0.1 as my learning rate. I also implemented one hidden layer in my neural network with 60 neurons just as a benchmark starting amount. After I was finished programming functions for forward calculation, back propagation, and letter identification, I was able train my network and test it on a test input. The test inputs consisted of perfect letter bitmaps. After each round of training and testing different values were manipulated to observe different and hopefully better results. There are essentially four variables I could change in this neural network: the number of back passes, learning rate, number of neurons in each hidden layer, and the number of hidden layers. The input layer is always going to be 126 neurons and the output layer is always 26 neurons, so I could only manipulate the hidden layers. In my testing, I would first change the number of back passes. Back passes are when the network forward calculates then back propagates through the network and changes the weights (also known as an epoch). Next, I changed the learning rate. This is the measure of how quickly a neural network learns. In theory, a high learning rate means that the network needs little training to recognize a character but often over generalizes and recognizes every input as the same letter. A smaller learning rate means that the network needs more examples to accurately give a guess to what an inputted character is. After the learning rate, I changed the number of neurons in each hidden layer. With one layer, this was easy because my program’s functions were set up to handle any number of neurons in one hidden layer. Lastly, I changed the number of hidden layers. Adding hidden layers helps add more levels of computation and robustness to a neural network giving it a higher level of dimension. However, when I changed the number of hidden input layers, I had to edit my program’s forward calculation and backwards propagation functions because they were hardwired for only one hidden layer. Out of time efficiency, changing the number of hidden layers was my last resort in testing.

Training and Finetuning:

The neural network was trained with perfect letters only and compared to a list of bits representing a perfect letter afterwards. The expected output of the neural network would be a list of 1’s and 0’s: 1 (100% sure) for the inputted letter at the correct index (A at 0, B at 1, etc.) and 0 for all other numbers in the list. The actual output of the neural network is something between 0 and 1. The test data contains 2,000 random letter inputs separate from the training data. After a few rounds of training and testing with one hidden layer of neurons I was observing that my network was giving almost random characters as answers no matter what I changed the learning rate and number of neurons in the layer to (see Sample Runs in this report). I assume if I increased the number of back propagation iterations it would help improve the accuracy, but I soon learned my computer could not handle the number of iterations needed. So, my next course of action was to increase the number of hidden layers to two. In the end, two hidden layers gave me accurate results, as can be seen in the Results and Analysis of this report. After I determined two hidden layers was the best implementation for me accuracy and time wise, I was able to change and experiment with different numbers of neurons, learning rate/schedule, and back propagation passes to give me the combination yielding the best results. Finally, I tested the network with flipping bits progressively and observing at how many bitflips the network would fail to recognize each letter. I set the threshold of a fail as less than 50% correct on a test set which was tested after each bit flip.

Sample Runs (screenshots)

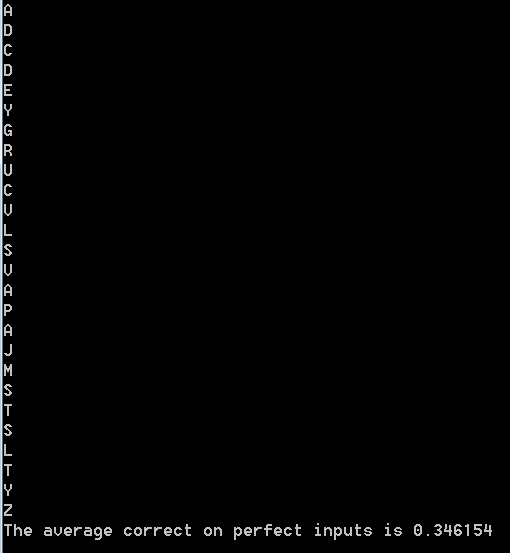




1 hidden layer, 60 neurons, learning rate of .1, 15,000 back passes

1 hidden layer, 60 neurons, learning rate of .2, 15,000 back passes





1 hidden layer, 85 neurons, learning rate of .2, 15,000 back passes

1 hidden layer, 85 neurons, learning rate of .1, 15,000 back passes



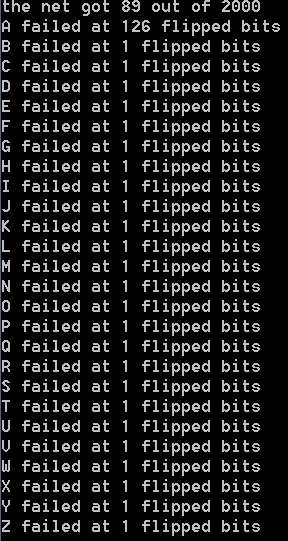
2 hidden layers, 5 neurons each, learning rate of .1, 100 back passes

2 hidden layers, 5 neurons each, learning rate of .1, 500 back passes



2 hidden layers, 5 neurons each, learning rate of .1, 1,000 back passes

2 hidden layers, 5 neurons each, learning rate of .1, 5,000 back passes



2 hidden layers, 25 neurons each, learning rate of .1, 100 back passes

2 hidden layers, 5 neurons each, learning rate of .5, 15,000 back passes

2 hidden layers, 5 neurons each, learning rate of .05, 15,000 back passes

2 hidden layers, 5 neurons each, learning rate of .1, 15,000 back passes



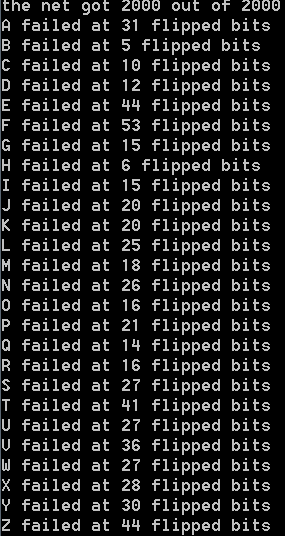


2 hidden layers, 25 neurons each, learning rate of .1, 20,000 back passes

2 hidden layers, 25 neurons each, learning rate of .1, 10,000 back passes

2 hidden layers, 25 neurons each, learning rate of .1, 5,000 back passes

2 hidden layers, 25 neurons each, learning rate of .1, 500 back passes



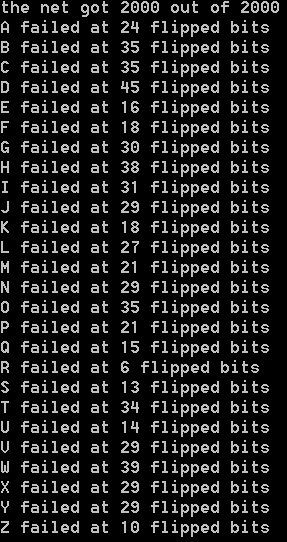
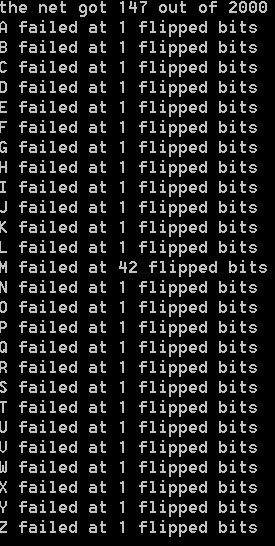




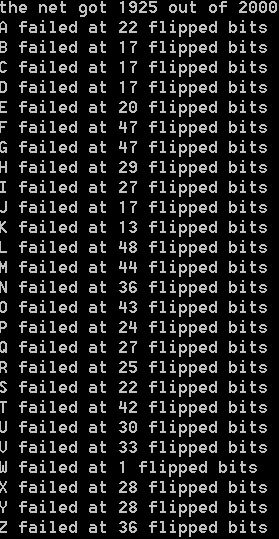
2 hidden layers, 60 neurons each, learning rate of .1, 100 back passes

2 hidden layers, 25 neurons each, learning rate of (.1/pass #), 5,000 back passes

2 hidden layers, 25 neurons each, learning rate of .01, 5,000 back passes



2 hidden layers, 25 neurons each, learning rate of .5, 5,000 back passes



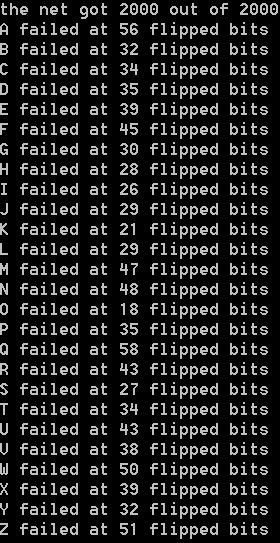
2 hidden layers, 60 neurons each, learning rate of .1, 5,000 back passes

2 hidden layers, 60 neurons each, learning rate of .1, 1,000 back passes



2 hidden layers, 60 neurons each, learning rate of (.1/pass #), 5,000 back passes

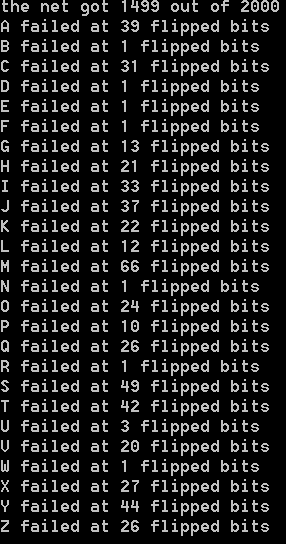
2 hidden layers, 60 neurons each, learning rate of .01, 5,000 back passes



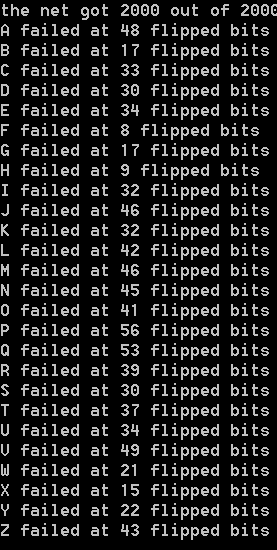
2 hidden layers, 85 neurons each, learning rate of .1, 1,000 back passes

2 hidden layers, 85 neurons each, learning rate of .1, 100 back passes

2 hidden layers, 60 neurons each, learning rate of .5, 5,000 back passes

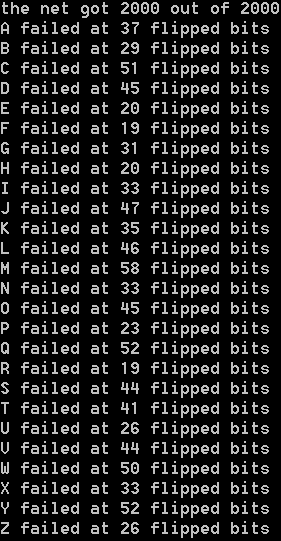


2 hidden layers, 85 neurons each, learning rate of .1, 3,000 back passes



2 hidden layers, 85 neurons each, learning rate of (.1/pass #), 4,000 back passes

2 hidden layers, 85 neurons each, learning rate of .1, 4,000 back passes



2 hidden layers, 85 neurons each, learning rate of .5, 4,000 back passes

2 hidden layers, 85 neurons each, learning rate of .01, 4,000 back passes

Results and Analysis

Discussion of Sample Runs:

From the screenshots in the previous section of this report, it can easily be seen that the learning rate, number of hidden layers, number of neurons in each layer, and the number of back passes each provided a significant role in the results of the neural network. Firstly, keeping the learning rate constant at .5 showed the best results for all the test runs. Decreasing the learning rate and having a decreasing schedule significantly decreased the accuracy of the network. Additionally, increasing the learning rate to a constant ex.to .8 also negatively affected accuracy, as can be seen in the test with two hidden layers with 85 neurons. Next, increasing the network’s hidden layers from one to two greatly increased accuracy. In the tests with one hidden layer, the accuracy maxed out anywhere from 30% to ~70% accurate no matter the learning rate and pass number. The results of this one hidden layer network were also sporadic, providing accuracies anywhere in the 30-70% accurate range on each run. When the extra hidden layer was added, accuracies drastically increased. With two hidden layers and 5 neurons in each layer, it took many back propagations for the network to show any kind of accuracy with 4.45% accuracy at 15,000 back passes with a learning rate of .1. As the number of neurons increased, the accuracy also increased. At 25, 60, and 85 neurons, the network was able to give 100% accuracy on 2000 test inputs with a .5 learning rate for each. Further in this analysis, it will be observed how quick these configurations learn to determine which is fastest. Lastly, as the number of back passes increased, the accuracy of the tests also increased for every network setup. When the number of passes were low, the network seemed to only learn one letter and generalized every other letter as that one letter. This can be seen in the low accuracies and one letter always never failing with any amount of bitflips (126 flips mean the entire bitmap was flipped).

Bar graph showing each layer/neuron amount pair network configuration’s maximum accuracy in testing

From the graph, the top three configurations all got 100% accuracy on the test data. However, it will be determined later in the report which configuration reaches this threshold the quickest.

Scatter Plot showing the accuracy of a 2 hidden layer 5 neuron per each layer vs the back-pass number

From this plot, the accuracies never really pass .15 except in some cases. There is also no real trend of the neural network which most likely means it is essentially guessing at each input.

Scatter Plot showing the accuracy of a 2 hidden layer 25 neuron per each layer vs the back-pass number

From this plot, the accuracies of the configuration with the .5 learning rate show an increase over increasing pass number. It reaches consistent 90%+ accuracy at around 1750 to 2000 back passes. This configuration increases at about .04% per back pass.

Scatter Plot showing the accuracy of a 2 hidden layer 60 neuron per each layer vs the back-pass number

From this plot, the accuracies of the configuration with the .5 learning rate show an increase over increasing pass number. It reaches consistent 90%+ accuracy at around 1100 to 1200 back passes. This configuration increases at about .07% per back pass.

Scatter Plot showing the accuracy of a 2 hidden layer 85 neuron per each layer vs the back-pass number

From this plot, the accuracies of the configuration with the .5 learning rate show an increase over increasing pass number. It reaches consistent 90%+ accuracy at around 1300 to 1500 back passes. This configuration increases at about .07% per back pass.

After observing all these scatter plots, the 2 hidden layers with 60 neurons per layer configuration performed the best. The 5-neuron configuration was almost completely random, so it isn’t a useful setup to use for this problem. The 25, 60, and 85 neuron configurations all reached 90%+ accuracy. However, the 60-neuron configuration reached 90% accuracy the quickest. Additionally, after running linear regression on all the configurations (except 5 neurons) the 60-neuron setup had the greatest slope .07%/back pass which means it is learning faster than the other configurations. The configurations with a learning rate of .01 were ignored because the accuracies were all very low and hard to compare. Also, the configurations with one layer were also ignored due to the same reason.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | E | F | G | H | I | J | K | L | M |
| Two hidden layers, 5 neurons, .5 learning rate | 1 | 1 | 1 | 126 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Two hidden layers, 25 neurons, .5 learning rate | 29 | 21 | 17 | 39 | 13 | 38 | 53 | 26 | 23 | 27 | 23 | 28 | 9 |
| Two hidden layers, 60 neurons, .5 learning rate  Configuration (with 5,000 back passes) | 54 | 43 | 38 | 34 | 38 | 32 | 45 | 37 | 28 | 28 | 49 | 34 | 47 |
| Two hidden layers, 85 neurons, .5 learning rate | 49 | 34 | 26 | 27 | 15 | 37 | 28 | 20 | 15 | 40 | 46 | 36 | 52 |

Number of bit-flips until failure

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| Two hidden layers, 5 neurons, .5 learning rate | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Two hidden layers, 25 neurons, .5 learning rate | 53 | 10 | 37 | 20 | 1 | 31 | 44 | 31 | 37 | 38 | 26 | 29 | 21 |
| Two hidden layers, 60 neurons, .5 learning rate | 49 | 34 | 32 | 43 | 28 | 35 | 51 | 28 | 29 | 38 | 38 | 37 | 50 |
| Two hidden layers, 85 neurons, .5 learning rate | 43 | 30 | 29 | 36 | 14 | 35 | 48 | 34 | 44 | 37 | 39 | 39 | 49 |

After analyzing the graphs and table above, it can be determined from testing that the configuration with 2 hidden layers and 60 neurons in each layer with a .5 learning rate was the best overall performer. It reached 90%+ accuracy the quickest and had the steepest rate of learning out of any of the other configurations. It also performed the best in the bit flip test producing the highest number of bits in 12 of the letters out of all the configurations.

Conclusion

In this project, I showed that a neural network can efficiently read capital letters from the IBM EGA 9x14 font matrix and determine what the correct letter even with some amount of noise. I experimented with different neural network configurations to determine which one was most effective. In this specific testing environment, I determined that a 2 hidden layer with 60 neurons in each layer and a .5 learning rate was the best performer. I learned that adding layers to a neural network adds another dimension to the calculations and usually gives better results than a network with less layers. Additionally, I learned that increasing the number of neurons in each layer does not necessarily lead to better results. Also, I observed how the learning rate can change the results of the network, in this case, decreasing the learning rate negatively impacted the accuracy. The most important thing I learned, however, was setting up, training, and debugging a neural network. I discovered very quickly that these structures are very hard to debug because there is so much going on within the different layers. To debug, I learned that adding output statements at each stage of forward and backward passes helped me understand what my program was doing and what was going on. In the future, I can use these skills to help me with other neural networks and machine learning problems to further my knowledge of A.I. and computer science.

Future Research

To further develop my program to produce better results, testing more configurations of the neural network in the future may lead to better results. Additionally, more inputs could be used representing different characters in the font, so the program isn’t only one dimensional in recognizing only capital letters. This program could also be running on a faster computer to determine if adding more hidden layers produces better results than the configurations in this report. Lastly, different activation functions may be explored to possibly improve the program’s accuracy.

Instructions for Running the Program

To compile my program, use the command: g++-8.2.0 -std=c++2a csce420Project.cpp

on a terminal system

Then run the executable file produced

The program will give you the accuracy of a configuration, I’ve set (this can be edited)

Also, it will run a bit flip test on one letter (reduced code down to one letter to save printing space)

\*note the testing program uses the same structure for every letter for a bit flip test

Bibliography

“IBM Fonts.” *IBM Fonts by Farsil*, farsil.github.io/ibmfonts/.

3Blue1Brown, director. *But What \*Is\* a Neural Network? | Deep Learning, Chapter 1*. *YouTube*, YouTube, 5 Oct. 2017, [www.youtube.com/watch?v=aircAruvnKk&t=511s](http://www.youtube.com/watch?v=aircAruvnKk&t=511s).

Russell, Stuart J., and Peter Norvig. *Artificial Intelligence: a Modern Approach*. Prentice Hall, 2010.