ReLu-king at Neural Nets

Benjamin Murphy
Undergrad, Dept. Of Computer Science
UC Berkeley
Berkeley, CA USA
ben_murphy@berkeley.edu

Eddie Gao
Undergrad, Dept. Of Computer Science
UC Berkeley
Berkeley, CA USA
eddiegao98@gmail.com

***Abstract*—This document is an exploration into potential neural net improvements based on new discoveries in the world of neuroscience. Using ReLu functions in different places in the network, we tried to design a neural net that was more accurate to the neurons that were found in nature. We also attempted to replicate nature's neurons by using AdaBoost with neural networks instead of with decision trees.**

***Index Terms*—Neural Networks, AdaBoost, ReLu, Neuroscience, Activation Function**

I. Introduction

For our project, we were testing a certain number of ways to make neural nets better at predicting. We were hoping that if we better modeled neurons in the neural networks with what was typically seen in nature, we might have better results. We tried a combination of things from AdaBoost with Neural Nets instead of decision trees to inserting an extra ReLu function at the end of each neuron. Overall, the hope was that we could discover ways to improve Neural Networks based partially on nature.

A. Doing What Comes Naturally

Firstly, a little background on neurons. The brain is made up of many cells called neurons that release electrical signals when stimulated. These cells are called neurons. Each neuron is made up of 3 primary parts: dendrites, nucleus, an the axon.

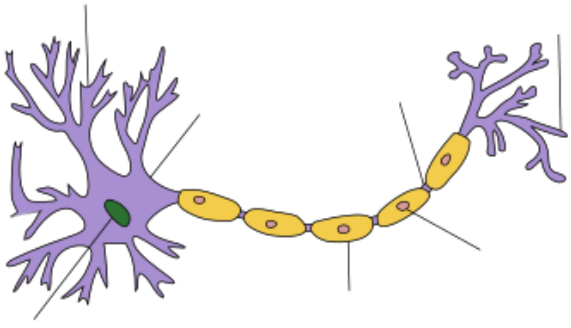

Fig. 1. A typical neuron found in a brain

A neuron receives input in the form of electrical signals from its dendrites. These inputs then all congregate in the nucleus, which then decides, based on some function of the inputs, whether or not to fire a signal down the axon. This is where things get interesting. It was previously believed that the dendrites inputs would all be treated equally when they arrived at the neuron. Once the neuron had summed up all of the inputs, it would then see if the sum had cleared some threshold and would fire a signal with a strength based on the sum of the inputs. However, this is where the paper comes in. This paper has made some interesting discoveries on what exactly is going on in the nucleus of a neuron and given us more insight as to how this activation function might work. According to Sardi et. al., it seems like each individual signal might pass through an activation function of its own before being congregated into the sum with the nucleus. Furthermore, it seems as though information on the distance and proximity of neurons are also kept for the activation function in the nucleus.

The first model (Fig. 2) is basically what were were describing in the first part of the introduction, the nucleus that takes a linear sum of the inputs and then outputs based on whether or not it was past a certain threshold. The second diagram shows an alternative take on what could have been happening with the neurons. In this diagram, the neuron is receiving input from each of the three dendrites represented by weights on a spring. In this diagram, it would appear that they each have a threshold that they must clear to fire, with that input being passed on to the main activation function. And finally, in the third description, the dendrites operate independently of each other, and they all output their own signals that combine into the overall output of the neuron. We attempted to model the second neuronal model with our Neural Networks.

Doing this proved to be a little difficult. Neurons as are not represented this way in neural networks. Theyre represented mostly as the first model would suggest, with no regard to the proximity in the calculation of the summation. This meant that we couldnt really use default implementations of a neural network from libraries like Tensorflow and others. However, we happened upon an implementation of a neural network using numpy.

We decided to use the classic MNIST Dataset which

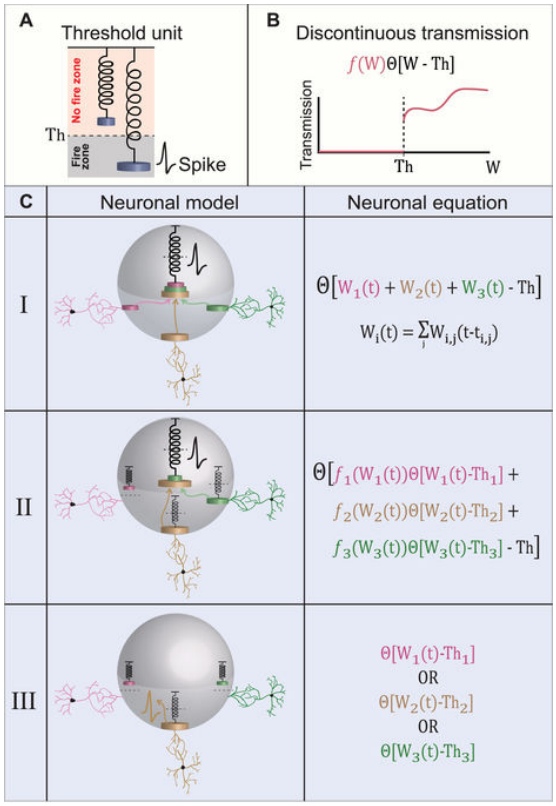

Fig. 2. Models of a Neuron

is a collection of handwritten numbers for out network to learn on. We recognize that a regular neural network, as opposed to a convolutional neural network, is not the industry standard when classifying images. However, since all we were trying to show was a proof of concept, we eventually decided that working with strictly a neural network would be both easier and probably better in terms of showing results, seeing as there was more to improve on. Current MNIST classifiers are getting errors of less than 0.27%, so showing any improvement on that would no doubt be very difficult.

II. AdaBoost to Simulate Distance

A. Why?

Initially, we were thinking about ways to implement some sort distance or proximity in each neuron. We kicked around the idea of weighting maybe the first half of the features with one weight and the second half of the features with another weight. However, we realized that this was essentially the same thing as training 2 separate neural nets on half of the data.

The multiple neural net and the majority vote scheme did, however, remind us of another thing that weve done in class, AdaBoost. In a nutshell, AdaBoost is the idea that you can take decision trees and fit them to a subset of the data. After doing this, you classify which points the tree is still getting wrong and make those points more likely to be included in the subset, as well as adjusting the weight (representing a measure of faith) on that tree. In the end, you run all of the trees on the test set and then take a weighted majority vote to decide which class the point should be in.

B. Implementing AdaBoost

We thought that perhaps, if we shook it up a little bit, AdaBoost could be used with neural nets instead of decision trees. We adapted the Homework 7 solution code to implement AdaBoost with neural networks. Hpwever, this meant that we had to make several design choices when doing so. For instance, how many training iterations should each neural network go through before we move on to training the next one? How many neural nets should we have when running the algorithm?

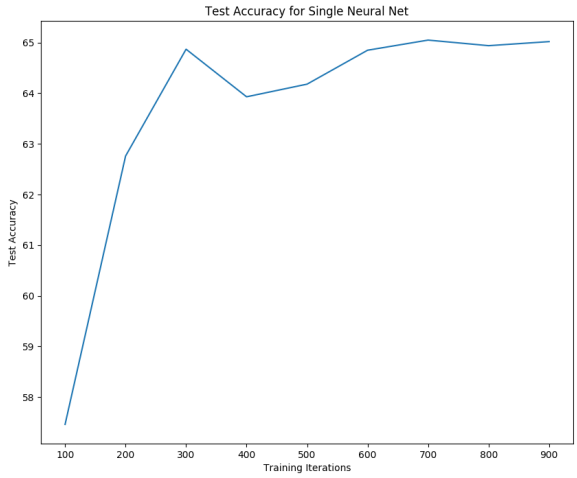

Fig. 3. Number of Training Iterations vs. Accuracy for a Single Neural Net

In our hyperparameter search of a single neural net, it performed best when left to train for about 1000 iterations. However, this was too much to do for multiple neural nets and would have been too computationally expensive. So, we came to a compromise of about 100 training iterations, because that was the best in terms of both feasibility and accuracy, with an accuracy of roughly 57.46%.

There were other hyperparameters that we also had to tune, including the regularization parameter, the learning rate. We ran a hyperparameter search over the regularization parameter and the learning rate. We were thinking that if we did this first, then we would be able to find the best possible numbers to use for our training iterations, which would be the most computationally expensive.

We found that a learning rate of 0.000001 and a regularization parameter of 0.001 were the best for a single

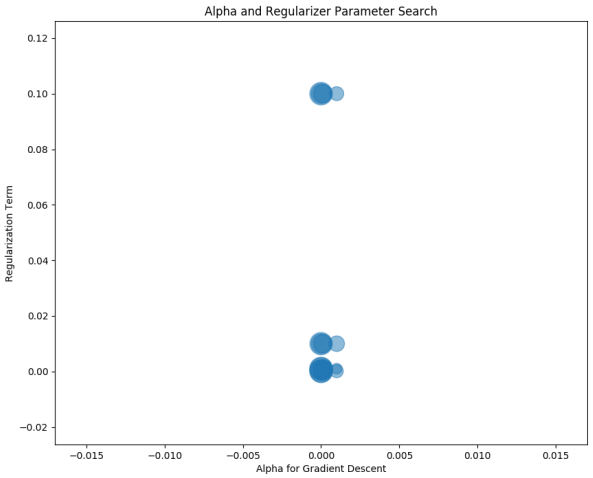

Fig. 4. Number of Training Iterations vs. Accuracy for a Single Neural Net

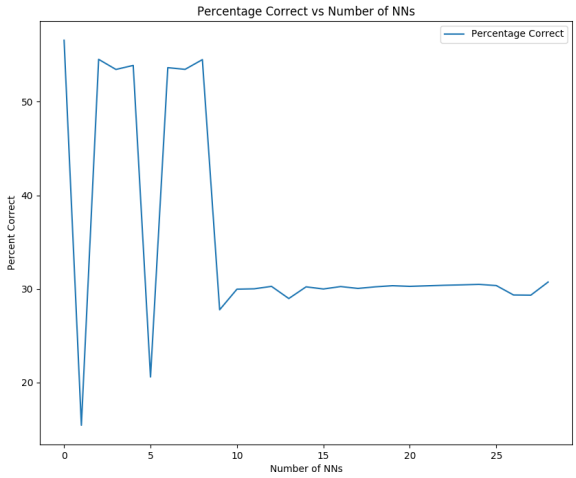

Fig. 5. Number of Neural Nets In AdaBoost vs. Accuracy

neural net (Fig. 4). Essentially, we were hedging our bets that with the right learning rate and regularization parameter for 100 training iterations, that this trend would continue for more training iterations. This makes sense because of how gradient descent works. If youre regularizing properly and have a reasonable learning rate, you shouldnt be following gradients that will lead you to overfitting.

We reasoned that, if the basic neural net could get 57.46%, then the AdaBoost should be able to do a lot better than that because its a collection of neural nets that are theoretically that accurate. This would mean that, in order to select the incorrect choice, not only would the correct choice have to be outweighed, it would have to be the same incorrect choice that the other networks were making. Or at the very least, it would have to be highly weighted neural nets that were making the wrong choice, which theoretically would be difficult, as we trained and tested the base neural net on the same sets of data, so they should all have the same base correctness level. Furthermore, as a result of AdaBoost, the data that causes mistakes will be trained on until it can be correctly classified, so with this in mind, we made our assumptions about AdaBoost and moved forward.

C. Results

To be more thorough in our assessment of the performance of neural net AdaBoost, we decided to do 2 hyperparameter searches, one for the optimal number of training iterations for each neural net, and another for the optimal number of neural nets to make decisions on. It turns out to be around 7 (Fig. 5).

However, the idea that AdaBoost would do significantly better did not pan out. At it's best, we were getting around 55% of the point correct, which is actually slightly worse than the single neural net. In order to verify this conclusion, we trained each of the neural nets in the 7 net AdaBoost for more iterations (Fig. 6) We discovered that training for more iterations didn't really have an effect.

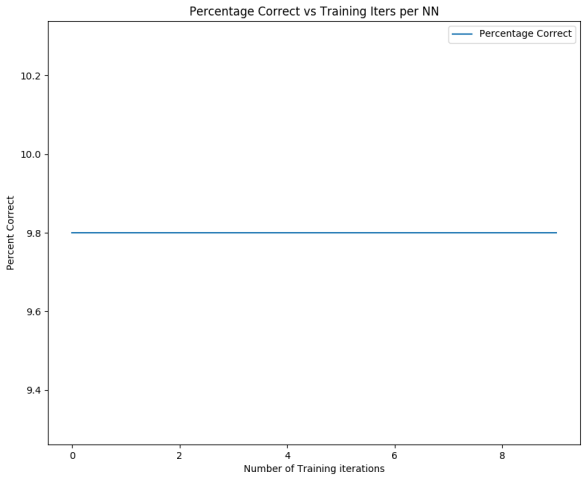

Fig. 6. Number of Training Iterations vs. Accuracy for 7 Networks in AdaBoost

We postulated that this could be because of overfitting, so we took a look at the training error for AdaBoost with a small subsection before and after the optimal value of Neural Nets. (Fig. 7) As it turns out, the train error seems to increase ever so slightly as you progress with the number of layers. This is very interesting, but could in fact be because the more networks you have, the more differing opinions you have. As a result, there is more noise introduced into your guesses and therefore more of them are made wrong. Because the error was less than a percent, this seems like a reasonable guess.

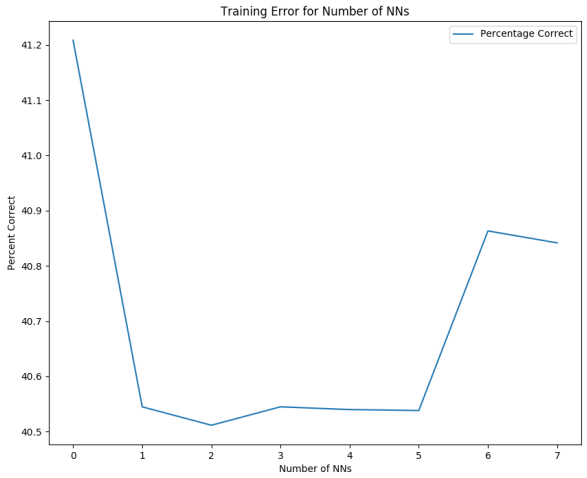

Fig. 7. Number of Neural Nets in AdaBoost vs. Training Error

D. *Conclusion*

Training AdaBoost with neural nets is both computationally intensive and not really that effective. It ends up not really being that much better than a single neural net. In some cases, it actually ends up being worse, potentially because of more noise being introduced.

III. ReLu-ke Use the Force (of nature)

A. *Eddie, put your stuff here*

Also, if you could cite any sources that you used in your write up in the bibliography, that would be great. Feel free to edit anything and everything in case I missed anything. Thanks!

References

[1] S. Sardi, R. Vardi, A. Sheinin, A. Goldental, and I. Kanter, New Types of Experiments Reveal that a Neuron Functions as Multiple Independent Threshold Units, Nature News, 21-Dec-2017. [Online]. Available: https://www.nature.com/articles/s41598-017-18363-1. [Accessed: 03-May-2018].

[2] Dennybritz, dennybritz/nn-from-scratch, GitHub, 19-Oct-2017. [Online]. Available: https://github.com/dennybritz/nn-from-scratch. [Accessed: 03-May-2018].

[3] Akesling, mnist.py, Gist. [Online]. Available: https://gist.github.com/akesling/5358964. [Accessed: 03-May-2018].

[4] THE MNIST DATABASE, MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges. [Online]. Available: http://yann.lecun.com/exdb/mnist/. [Accessed: 03-May-2018].

[5] MNIST database, Wikipedia, 24-Apr-2018. [Online]. Available: https://en.wikipedia.org/wiki/MNIST_database. [Accessed: 03-May-2018].

[6] Neuron, Wikipedia, 30-Apr-2018. [Online]. Available: https://en.wikipedia.org/wiki/Neuron. [Accessed: 03-May-2018].