Final Report

Brendan Muldowney

ECE 4424/CS 4824 Machine Learning

Chris Wyatt

12/11/2019

Abstract:

For this project, I decided to create three different neural networks. One being a one-layer network, the second being a two-layer network, and the last being a three-layer network. My motivation for doing three different neural networks was actually due to the fact that my three-layer network was not working so I went to create a one-layer network. After this, I ended up getting my three-layer network working which allows me to compare all three. These networks were compared on three different criteria. The first being error, the second being miss rate, and lastly time per iteration. The error was calculated using a multi-dimensional variation of square error. This was done by finding the difference of the target vector (one hot encoded) and output vector calculated by the neural network for each data entry in the set. The magnitude of these new vector would then be calculated and summed to create the error. Miss rate was a very simple calculation. Because the output was one hot encoded, in order to determine if an output was correct, the argmax of both the target and output vectors would be compared. If they were not the same, it would be interpreted as a miss. The last parameter, time per iteration, was chosen because time was not a good measurement by itself. The two-layer neural network ran for 1480 iterations while both the one-layer and three-layer ran for less than a thousand iterations. One important design feature that does affect the speed of my algorithm is the ability to choose to calculate error every twenty iterations. While this will be explained in more detail later on, the choice was made to allow my algorithms to run a lot faster. There are many other parameters that could be used to compare the results, but these were the three chosen in my design.

The test error produced from the one-layer network was 1773.811 with a miss rate of 10.0 percent (missed 1,000/10,000). This ran at a speed of 88 milliseconds per iteration. The two-layer had the best result of all my implementations, having a test error of 529.805, a miss rate of 3.4 percent (missed 340/10,000), and a speed of 143 milliseconds per iteration. The last neural network, the three-layer, had a test error of 690.209 and a miss rate of 4.45 percent (missed 445/10,000). This network ran at a speed of 118 milliseconds per iteration.

Theoretical Description:

"Neural networks are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns." [1]. Neural network connects strings of "neurons" or nodes together to create a kind of web. These networks are split into layers which contain a number of these nodes. The neurons are intertwined from layer to layer and each neuron. Each of the nodes have an activation function and a series of weights. These weights are used to create some value or values that then can be put through the activation function which will be passed onto the next layer. When we say one-layer, two-layer, or three-layer neural networks, we are talking about the number of layers after the input layer. In Figure 1, we can see there are two layers after the inputs. The in-between layers between the input layer and output layer (the maroon nodes in Figure 1), are referred to as hidden layers. When an input is applied to the network, forward propagation occurs. This happens by taking the input vector and multiplying it by the first layer matrix. This will allow the node to get a value that then can be used in an activation function. Activation functions are ways to turn the value at the node into a probability that can be viewed as determining whether the node is active or not [3]. The output of the first layer can be applied to the second layer and so on. Eventually we will reach the output layer and receive an output result.

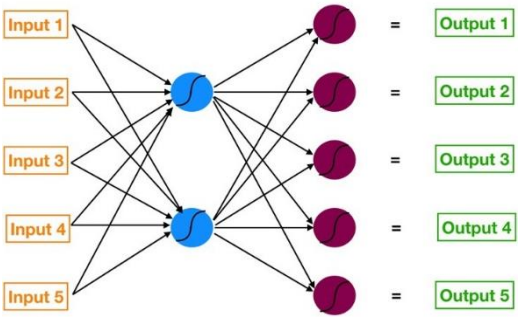

Figure 1: Two Layer Neural Network [2]

The output result subtracted by the target vector, the desired result, creating a delta value. This delta value, or vector, has two major uses. One use is to help calculate error in order to help determine convergence as well as help determine back propagation to update our weights and nodes. Error has a couple different methods of calculating it. However, for this project, the error was calculated by finding the magnitude of the delta vector. As mentioned, the delta vector's other use is for back propagation. Back propagation is how our neural networks learn. Essentially, they take the delta vector which represents how far off out calculations was, the loss. This loss is carried back using a combination of multiply the weights by the loss and derivative of the layer's activation function which can be imagined as helping the loss travel, or propagate, backwards to help adjust all the nodes. This mean to propagate back one layer, it would be require multiplying that layer's weights by the delta vector and derivative of the output activation. If we want to then propagate back more layers, we then take this new loss we calculated at the output layer and multiply it by the weights of the last hidden layer and the derivative of that layer's activation function. This process can be continued across all hidden layers, eventually creating a gradient for each layer [4].

These gradients are then used to preform stochastic gradient descent. Stochastic infers some random probability which we will use to create batches [5]. These batches are random subsets of the training data. These values are then used to calculate a gradient which will help us find the correct step (adjusting weights) to take.

This then brings us to the Adam algorithm. This is an algorithm that helps speed up the process of updating weights. Sometimes, the difference in steps can be very small which makes training take a very long time. Algorithms like Adam implement *'momentum'* which in a way helps us get though these slow-training areas. The gradients are taken and put through Adam which will update our weights for the next iteration. The Adam algorithm was derived from a paper by Kingma and Ba, *Adam: A Method For Stochastic Optimization* [6].

The algorithm keeps iterating and updating weights until it converges. This convergence is calculated by looking at the last twenty validation errors calculated and if there is no decrease, then the weights corresponding to the lowest error are taken and the iteration stage ends. This ends the training and these optimal weights are our neural network.

For all three neural networks created, this process is the same, the only alteration is the number of layers which affects the number of forward propagations and backward propagations.

Implementation Details:

The code starts with the finalProject function. This function allows the user to control which algorithms they want to run and at what speed. This is further explained in the following section, but there is a way the network can be run while calculating multiple iterations before it calculates the error in the validation set.

Based on the selection made by the user, some combination of the three network functions will run. These functions are called layer1(), layer2(), and layer3() which represent a 1-layer network, 2-layer network, and 3-layer network, respectively. These can be found on lines 632, 645, and682. While leveraging my modifyData function, all three functions will train using the training set, find all the optimal layer weights (this varies in amount depending on the network type), and then use these weights to find the error for the test dataset.

Each network consists of two main functions, test#Layer and train#Layer (# is replaced with how many layers are implemented). The train#Layer is in charge of creating a training and validation subset as well as training the weights using forward propagation, determining the delta, calculating gradients using stochastic gradient descent and back propagation, and updating these weights using an Adam algorithm. The test#Layer function is used to calculate error and

miss rate for the validation subset and testing set. The following table is where these functions can be found:

	test#Layer	**train#Layer**
1	59	93
2	212	249
3	402	445

As mentioned, the user can decide if they want the validation error to be calculated every step or only after a set amount of iterations (this set amount happens to be 20). This decision is made at the bottom of every train#Layer. If speed is set to 1, then the test#Layer is only called every time the iteration modulated by 20 equals 0. This calculation has no effect on the training besides finding convergence.

For all hidden layers across all networks, the sigmoid activation function was used. This function as well as its derivative can be found on lines 24 and 26. For the output layer of all the networks, a softmax activation function was used. The softmax function and its derivative can be found on lines 28 and 30.

Reproducibility:

The results found with these networks is very easy to reproduce. The packages being used are Random, HDF5, Distributions, and LinearAlgebra. Once all these packages are installed, run the script. This will produce a function called finalProject. finalProject is a function that takes four parameters. The first three parameters represent which neural network the user wants to run. The reason for using three separate variables is so the user can choose whether they want to just run one network, and combination of them, or all three. To say you want to run a network, just set its variable to one. The first parameter corresponds to the one-layer neural network, the second corresponds to the two-layer network, and the third corresponds to the three-layer network. The fourth parameter corresponds to the speed that the network runs at. This means that the user has a choice to calculate the error for every iteration, or to wait to calculate the error every twenty iteration. Since the error function is only used to determine convergence and does not affect backpropagation or the Adam algorithm, allowing the code to only run this time expensive function every twenty iterations greatly speeds up the code. Below are a few examples of how to run the code and what will happen.

julia> include(“final_project.jl”) //this will create an instance of the finalProject function.

julia> finalProject(1,0,0,0) //this will run only the one-layer network at regular speed

julia> finalProject(1,0,0,1) //this will run only the one-layer network at a faster speed

julia> finalProject(1,1,1,1) //this will run all the networks at a faster speed

Results:

Some of the results from the three neural networks were expected while others were surprising. The one-layer network had the most predictable result when compared to the other networks. It had the largest error at 1773.811 with a miss rate of 10 percent. The algorithm took 47.66 seconds to run an did 540 iterations which meant that each iteration took 88 milliseconds, the fastest iteration speed. This makes a lot of sense because it has the least number of layers and nodes which would lead to faster computation. There were also 9.49 million memory allocations (~17,600 per iteration) as well as 38.454 GB used (~71 MB per iteration). These were also the smallest values amongst all the networks.

The two-layer network preformed the best in many ways. It had the smallest error and miss rate at 529.805 and 3.4 percent respectively. These were the best all-around test results between all the networks. This came at a cost however, and quite an expensive one at that. This network ran for 10353.82 seconds which is just south of three hours (while only calculating error every twenty iterations). This means that it took 143 milliseconds per iteration which is the longest iteration speed of all three algorithms. It is also the most space expensive, making 57.5 million allocations totaling 8.9 TB. These translate to 38,000 allocations per iteration and 60 GB per iteration.

The three-layer network was the middle of the three algorithms. It had a test error of 690.209 and a miss rate of 4.45 percent. It took an average of 118 milliseconds per iteration totaling to 8308.41 seconds for the 980 iterations it ran. The reason that this is smaller than the two-layer algorithm is due to there being less nodes per layer. Overall, there were an equal number of nodes in the two and three-layer networks, however the organization led to their being less weights to consistently calculate and update while training. Although this led to a faster design, it probably also contributed to slightly less accuracy. It did happen to have more memory allocations at 63.76 million but totaled less space at 6.359 TB. Per integration, this translate to approximately 65,000 allocations per iteration and 64 GB per iteration.

The test error produced from the one-layer network was 1773.811 with a miss rate of 10.0 percent (missed 1,000/10,000). This ran at a speed of 88 milliseconds per iteration. The two-layer had the best result of all my implementations, having a test error of 529.805, a miss rate of 3.4 percent (missed 340/10,000), and a speed of 143 milliseconds per iteration. The last neural network, the three-layer, had a test error of 690.209 and a miss rate of 4.45 percent (missed 445/10,000). This network ran at a speed of 118 milliseconds per iteration.

In conclusion, all three neural networks had their pros and cons. If you cared less about accuracy and more about training cost (speed and memory), then the one-layer network is the best option. If you care about only accuracy, the two-layer network is a good choice. The three-layer network is the best, average algorithm. It has better accuracy than the one-layer, is less time expensive than the two-layer option, and although it requires more memory per iteration than the two-layer, it converges much faster which ends up using less memory.

References:

[1] C. Nicholson, "A Beginner's Guide to Neural Networks and Deep Learning," pathmind, [Online]. Available: https://pathmind.com/wiki/neural-network. [Accessed 9 12 2019].

[2] D. Gupta, "Fundamentals of Deep Learning – Activation Functions and When to Use Them?," Analytics Vidhya, 23 10 2017. [Online]. Available: https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions-when-to-use-them/. [Accessed 9 12 2019].

[3] A. Al-Masri, "How Does Back-Propagation in Artificial Neural Networks Work?," Torwards Data Science, [Online]. Available: https://towardsdatascience.com/how-does-back-propagation-in-artificial-neural-networks-work-c7cad873ea7. [Accessed 9 12 2019].

[4] R. Roy, "ML | Stochastic Gradient Descent (SGD)," GeeksforGeeks, [Online]. Available: https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/. [Accessed 9 12 2019].

[5] D. P. Kingma and J. L. Ba, "ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION," ICLR , 2015.

[6] T. Yiu, "Understanding Neural Networks," Towards Data Science, [Online]. Available: https://towardsdatascience.com/understanding-neural-networks-19020b758230. [Accessed 9 12 2019].