Machine Learning Engineer Nanodegree
Capstone Proposal
Maxfield Thompson, April 28, 2017

Domain Background

I want to do research on deep learning total node shape/number and hidden layer depth, and how these factors effect training time and accuracy on the MNIST dataset. Deep learning is a complicated subject, with many different parameters which can be adjusted to form different models. I would like to run formal experiments testing different variants of total network node shape/size and the number of hidden layers to better understand how these factors effect learning accuracy and training time. This research is exploratory, not solving a specific question. Though, the end result of this research will be better clarity on the total node shapes/sizes and hidden layer depths which best fit the MNIST dataset. Assisting research that should prove helpful in my investigations is Jurgen Schmidhiber's "Deep Learning in Neural Networks: An Overview".[1]

As previously stated, there are many parameters that can be changed in a deep learning algorithm, this research is important because it investigates three important parameters a machine learning engineer has control over: total node shape, total number of nodes, and the number of hidden layers. I personally find this research interesting because I want to better understand the workings of deep learning, and see this research as a great experiment based way of deeply exploring deep learning.

Problem Statement

The problem I will solve through exploratory research is "what deep learning node shape/size and hidden layer depth will produce the best accuracy on the MNIST dataset as it relates to training time". In deep learning here are many parameters that can be altered by an engineer. Three important parameters a machine learning engineer has control over are total node shape, total number of nodes, and the number of hidden layers in a network. All of these parameters are quantifiable and easily altered when creating a deep learning function. The effect these parameters have is also easily observable, just run the algorithm with the change! Plus altering these parameters is consistent and reliable (assuming someone sets a random seed the same as me), making my research repeatable.

Datasets and Inputs

The Dataset being used for this project is the MNIST dataset. This is the Modified National Institute of Standards and Technology's large database of handwritten digits.[2] There are different collections of this data, but I'm using the MNIST dataset hosted on Yann LeCun's website.[3] The original dataset used 20x20 pixel boxes to represent the hand written digits, but Yann's dataset transformed the digits into 28x28 pixel boxes to remove some of the gray pixels. That way an algorithm can be trained in a binary way only using 1's and 0's (black and white pixels). Some other pros to Yann's dataset is that he mixed the writers into the training and test sets. The original MNIST dataset's training images were competed by Census Bureau employees, while the testing images were completed by high-school students. This has created problems in previous models. Yann's dataset spreads the high-school students digits across the training and testing set, and does the same with the Census Bureau employees, this way the training and testing sets are as fair as possible.

Between the training and test sets, the MNIST dataset comprises approximately 500 separate writers. This means many styles of writing are represented throughout the datasets, making an algorithm's

1 http://www.sciencedirect.com/science/article/pii/S0893608014002135
2 https://en.wikipedia.org/wiki/MNIST_database
3 http://yann.lecun.com/exdb/mnist/

ability to generalize learning of these digits impressive. To upload this dataset into my jupyter notebook and start running various python functions on it, some simple upload statements are provided by a tutorial written on one of tensor-flow's development pages.[4]

All in all, this dataset will be loaded into a python IDLE (preferably jupyter notebook), and all the deep learning will be completed using tensor-flow and various other python packages like numpy, pandas, random, time, seaborn, and matplotlib. This dataset is a common "hello world" of machine learning and deep learning research. So there will be plenty of tutorials and online references to code I can use to help me get started. Of course the unique approach my research is taking is novel, but the basic architecture of how to code a deep learning algorithm in tensorflow is fairly generic. This will be helpful being that I'm new to tensorflow and deep learning, thus making this research challenging, but achievable.

Solution Statement

As mentioned earlier, in deep learning here are many parameters that can be altered by an engineer. Three important parameters a machine learning engineer has control over are total node shape, total number of nodes, and the number of hidden layers in a network. Now for this solution statement, the measurable outcome of changing these parameters will be accuracy and total training time (or potentially average training time per epoch). Both of these outcome metrics are measurable, quantifiable, and reproducible. Once the research has been finished, all steps will be clearly documented and the results will be visualized and discussed. Since all the code will be included in this research, the results will be replicable.

(**Note**: The only results that will differ in these tests from one person to the next would be traing time because someone may have a faster/slower computer than I'm running the code on. Now, the time for various models, even on different computers, should, in a *relative sense,* still stay the same. For example, say I run two different models with different node shapes (say a cone and a box). The cone takes 30 seconds to train, and the box takes 10 seconds to train. The exact times will differ based on the computer used, but the ratio of times should always stay the same. That is, the box will always train in 1/3 the time of the cone.)

Benchmark Model

A "benchmark model" is not extremely applicable to my research because I'm conducting exploratory research to better understand how deep learning networks training times and accuracies change with respect to total node shape, total number of nodes, and hidden layer depth. I'm not trying to find the "fastest" or "best" model, that research has been done before and I'm not qualified enough to compete in that realm. It is as much success for me to discover what *doesn't work* as much as I discover what *does work*. For example, I may find a certain network shape that has terrible accuracy and slow training time. My goal is to better understand *why* that model is not very successful. But with all that said, I will end up with a best model by the end of my research, and if I wanted to compare that model's accuracy (based on number of hidden layers AND total nodes) I can use Yann's MNIST website.[5] Towards the bottom he has roughly 60 different machine learning models, what preprocessing they completed, the test error rate, and the reference to that algorithm. I wouldn't want to compare all my models to those listed on Yann's website, but comparing the top model would yield interesting results.

Evaluation Metric

The evaluation metric for this data is rather simple, it's error rate. Error rate (or one minus the accuracy) is *one minus* the ration of right classifications divided by the total number of classifications predicted all multiplied by 100. For example, if there were 1000 total test images, and I correctly identified

4 https://www.tensorflow.org/get_started/mnist/beginners

5 http://yann.lecun.com/exdb/mnist/

956 of them, my error rate would be (1 - (956 / 1000)) * (100) which equals 4.6%. It is the percent of images I miss-classified. This metric is consistent regardless of who runs a particular model (assuming the model is the same, and the random seed is set). Where as my other metric, training speed, would differ depending on the computer used. (See the note section on the solution statement)

Project Design

The actual design of this project will include 3 separate experiments. Each experiment's control will be the *size* and *shape* of the nodes. Then on a higher level, each experiment will differ based on the number of hidden layers. This way there is solid comparison within an individual experiment, as well as solid comparison between experiments. For example, does the cone shape produce the highest accuracy (or lowest error rate) compared to other models consistently at hidden layer depths of 3, 7, and 20? This research would flush that out.

In deep learning there is an input layer, one or more hidden layers, and an output layer. My research will change the number of nodes at each hidden layer to form different network shapes. For example a network might have 100 nodes in the first hidden layer, then 15 nodes in the second hidden layer, and lastly 100 nodes in the final hidden layer. This would form an hour glass shape in the hidden layers. Different shapes could include cone shapes of different sizes, rectangular networks, square networks, sine-wave looking total node shapes, ect. *I will separate my experiments by the number of hidden layers.* For example, only the shapes formed with three hidden layers will be compared to one another in experiment one. I will run *three experiments.* The first experiment will compare seven different node shapes/sizes formed with three hidden layers. The second experiment will compare eight different node shapes/sizes formed with seven hidden layers. And the last experiment will compare seven different node shapes/sizes formed with twenty hidden layers. By “compare” I mean I will graph the accuracies as they relate to epoch number, and also visualize the total, and node/per_second training times each network takes.

The end result of an experiment is running x number of deep learning algorithms, each with different node shapes and node numbers, gathering the accuracies of theses models, and graphing them all on a singular line graph (where each line, each a different color, represents a different model's accuracy as in compares to epoch number). This way it becomes easy to visually compare the results of different deep learning algorithm parameters. I will create three of these graphs, each one representing each experiment. A big factor in each experiment will be *shape size* as well as just shape. That is, in experiment one, with hidden layer depth three, I will have a cone shape as well as a narrow cone shape as two separate models. How will this look? Well in the larger cone the hidden layer node numbers would be something like [layer_1 = 500, layer_2 = 300, layer_3 = 100] (notice the cone being formed), and the narrow cone would be something like [layer_1 = 50, layer_2 = 30, layer_3 = 10]. In this example the narrow cone is one tenth the size of the larger cone, but interestingly enough it could have similar testing accuracies, or possibly much lower accuracies. This research would flush out those types of questions.

A final part of each experiment will be graphing the training time divided by total node size for each model as a bar chart. This way I can compare the weighted training times for each model. Yes a certain model may take longer to train, but this doesn't tell the whole picture because a model may be 10 times the size (as in the previous example of the cone and the narrow cone). Thus looking at these weighted times will help me see how quickly a deep learning model is learning relative to its total size. That way I have a standard time based metric I can compare my models to.