**Eric Lipe & Kevin Ritter**

**CS 395**

**Project Part 2**

**April 14, 2019**

This is the second of three parts of your course project. This second part asks you to demonstrate that you have decided on your approach, have considered my feedback from part 1, developed your model, began to evaluate your hyperparameters, and have obtained some preliminary results. At this point, your outputs might be a bit rough and you may be still experimenting with some hyperparameters, but you are close to the final results.

The third (and final) part will be due in just over a week. In class, you will be asked to provide/present your results as part of a presentation (more about that in the next assignment). Remember the final project deliverable will incorporate what is being asked in this assignment (subject to refinements after receiving feedback) so when you prepare this second part, keep this in mind.

Requirements for this second project assignment are given below.

***(5 points) Choose and describe the problem you are trying to solve. Clarity is important in your answer.***

We are working on a solution to the Kaggle PUBG data set:

<https://www.kaggle.com/c/pubg-finish-placement-prediction>

The focus of our project is to be able to take any players in game statistics and predict any player’s probability of winning the game based off of those statistics. The problem is a regression task since we are trying to predict a real valued number in the range [0, 1].

***(10 points) Describe your approach in detail, taking into account all the feedback you have received (and any additional research you have performed).***

Our approach to the problem is to get both the Mean-Square-Error and the Mean-Absolute-Error as close to zero as possible. We are trying to accomplish this goal by using several techniques. Firstly, by testing different network topologies we have discovered that having five hidden layers seems to give us optimal solutions versus having ten or even twenty layers. Along with that, setting each layer to have 50 hidden nodes seems also to be optimal. Secondly, we have implemented a class to handle batch training for our model. Implementing batch training has given a significant boost to not only the speed of the networks training, but also to the networks ability to minimize the loss. We have yet to find the optimal batch size, however, we are still having great results. Lastly, we intend to add a learning rate schedule and possibly some dropout to the network. For the learning rate schedule we will be measuring the difference in the each epochs loss compared to the previous. Based off of how small this number is the learning rate will also decrease to a smaller value so that we don’t risk overshooting the minimum. Conversely, if the difference in the two epochs is high, the learning rate will be set to something larger so that we can approach the minimum faster. As for the dropout, Kevin and I have not decided if a network of only five layers would even need dropout. Dropout is something we plan to test more heavily in the next few days.

***(5 points) Describe the state-of-the-art solutions by others on the same dataset.***

What are the differences with your approach with theirs?

What are the differences according to your metrics?

The current state of the art, i.e. the winner of the Kaggle competition, used a method know as Extreme Gradient Boosted Trees along with a very impressive topological sort based off of feature correlation to the target variable. To perform the topological sort this kaggler performed a fairly heavy amount of feature engineering so as to find as many highly correlated features as possible. This kaggler did use the same metrics that we are using (MSE, MAE) to assess his models’ success. The difference between this kagglers model and ours is mainly the algorithm he implemented. The Extreme Gradient Boosted Trees algorithm is an ensemble method that creates many singular trees to find singular features and adds their predictions together to produce one inclusive prediction. Our model has a similar structure, however, we are not building many models and accumulating their predictions to make an inclusive prediction. Our model is one large structure that will find all of the defining features of what makes a player more probable to win.

***(5 points) What is the metric(s) you are using? Justify their use. If you are using data from a competition, it may be that others are using that metric, so you want to compare your answer against theirs.***

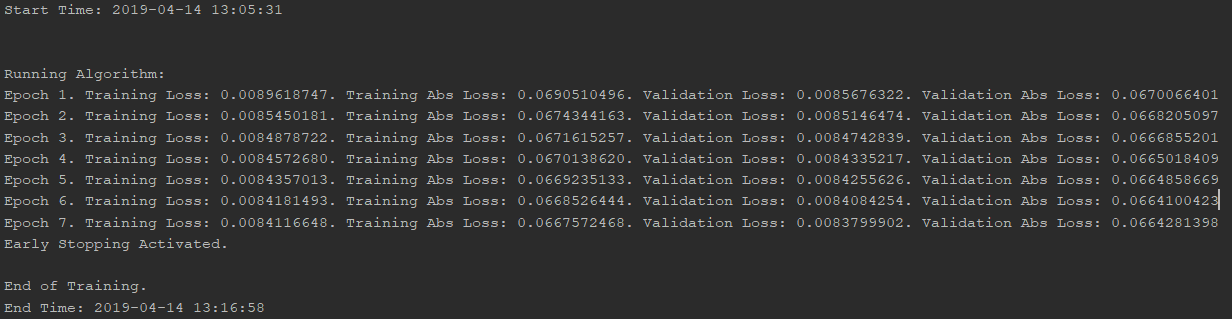
We are using square loss and absolute mean loss metrics. We are using these metrics because the player’s probabilities of winning are given by a value in a continuous range. If we used accuracy in our model instead of mean loss then we would be measuring our ability to pinpoint a single value from an effectively infinite set. This type of precision is so improbable as to be impossible. Instead we are seeking to see how close we can get and are comparing our results to those of the state-of-the-art solutions on Kaggle.

***(5 points) What is/are the activation functions used in your model? Where are they used? Show a few variations and how this affects your results.***

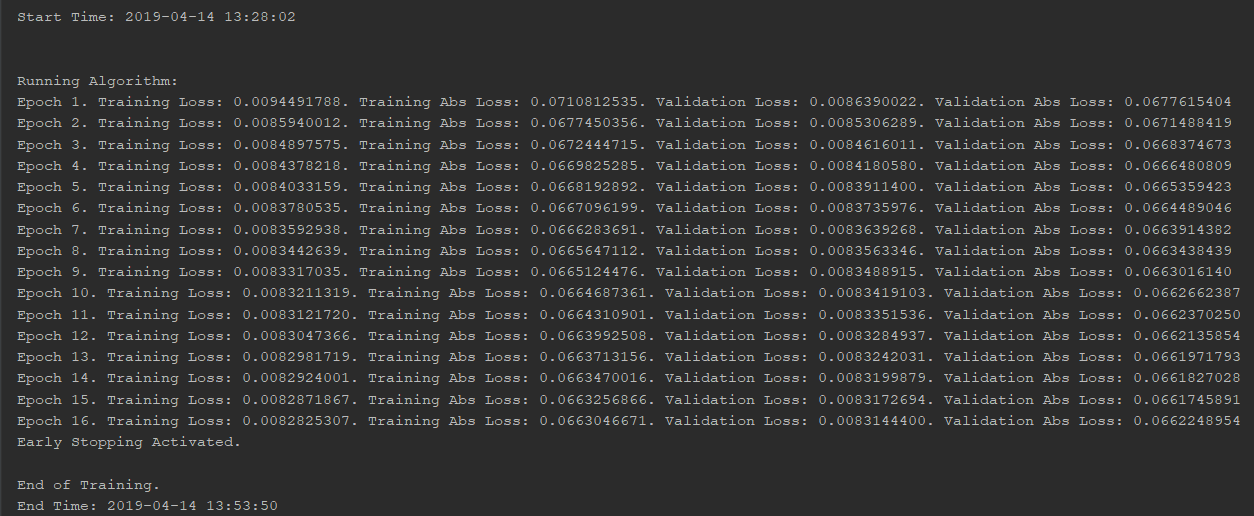
We’ve chosen to use ReLu as our activation function for the time being and have experimented with the Tanh and the Sigmoid. As of our most recent run, Relu might be pushed aside by the Sigmoid. If you look below, you can see that Sigmoid performed better than Relu despite having to go through several more epochs. From the test, we can see that the Tanh seems to be a very inefficient activation for our problem. Results below:

**Note**: Training Loss = MSE, Training Abs Loss = MAE and similarly Validation Loss = MSE, Validation Abs Loss = MAE

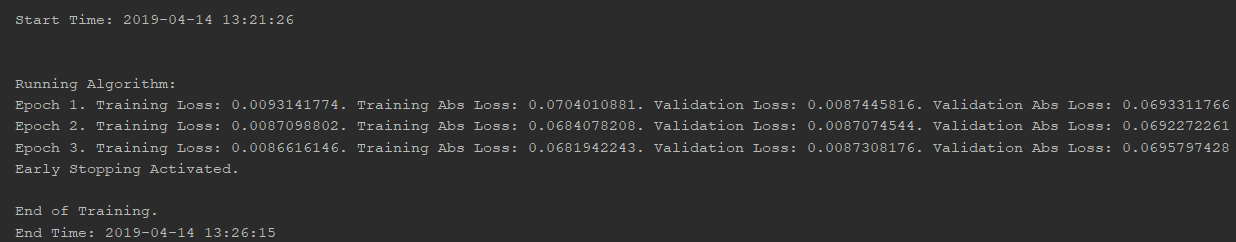
Relu:



Sigmoid:



Tanh:



***(5 points) Next, after choosing the best activation functions, what is/are the loss functions used in your model? Where are they used? Show a few variations and how this affects your results.***

We are currently optimizing based off of the Mean-Squared-Error and not the Absolute-Mean-Error. We are doing this because we think the non-differentiability of the absolute mean error at zero maybe causing network failure. We chose these loss functions because we are solving a regression problem and these loss functions, along with several others, are typically reserved for regression problems. Both outputs for the both the of the loss functions can be seen in the above pictures.