May 2, 2021

Predicting Chess Outcomes Data Analysis

Denis St. John III
CS5300
University of Missouri-St. Louis

Contents

1 Abstract

Chess is an old game full strategy, complex algorithms, and patterns. Research has been done on every topic and feature chess has to offer, from creating programs to play against humans, to analysis on different openings. Humans have been fascinated with chess and improving their skill in the game ever since it was developed. This paper focuses on a data set of over 20,000 games played on a website called lichess.org. The overall goal of this paper is to be able to predict the winner, either black or white, of a game given information about the game. We have removed the draw from the dataset to create a simple classification problem where either white wins, or black wins. The other features include Rating, number of turns, victory status, time, increment, white's rating, black's rating, which opening was used, and the number of moves in the opening. Since the data had many alpha characters in it, these were converted to numeric values by simply categorizing them with a mapping. After each feature was converted to a number they were normalized by dividing by the max number so that all of the features will be between 0 and 1.

The model is a neural network classifier built with using Keras Tensorflow in Python 3. All the programming is done in a Google Colab notebook linked later in the paper. The model is evaluated using numerous types including Sigmoid and Linear to see which is the best, these were improved with the use of callbacks using Model Checkpoint and Early Stopping. The model was validated by adding the output as an input, and by creating a function from the model using the weights. The last part of the paper goes over the feature importance of each feature by isolating them in the model and comparing them with each other, then slowly removing them from the main model analysis.

2 Introduction

Chess Machines

Since The Mechanical Turk, combining chess with a machine has been always been a dream of mankind. The Turk was touted as an early robot that could play chess at the highest level. Built in Vienna in 1770 by the inventor Wolfgang von Kempelen, the machine consisted of a large pedestal, housing intricate machinery on top of which stood a chessboard.[1] Today people don't play against a mechanical robot but against software developed to predict and choose the most favorable outcomes using Artificial Intelligence. Currently these different AI software engines are battle constantly to try prove they are the best. As of November 2020 the top rated software engine is called Stockfish[2] Stockfish is the strongest chess engine available to the public and has been for a considerable amount of time.[3] It's and open-source chess engine that uses powerful AI techniques to help individuals analyze games and improve.

Google Colab

https://colab.research.google.com/drive/1c0h4KJXlOn3MhNxaZVp7GoZECx3ur-WP?usp=sharing

3 Dataset

The "Chess Game Dataset" was obtained from the Kaggle Data Science [4]. This data was collected from Lichess.org and contains just over 20,000 games. While not each feature is used in this data set the following 9 are, Rated, Number of Turns. Victory Status, Time, Increment, White' Rating, Black's Rating, Opening name, and Number of moves in the opening phase.

3.1 Visualization of the distribution of each input feature

Below is a histogram of each feature for the given dataset with lines showing their mean and standard deviation.

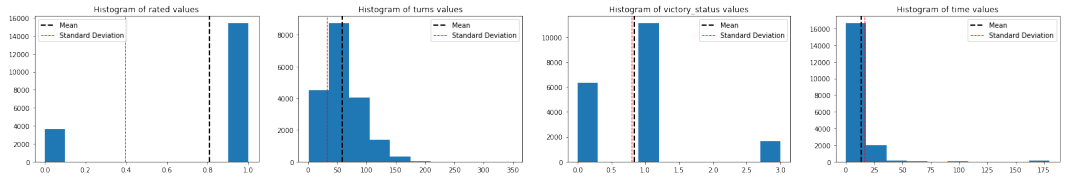

Figure 1: Histograms of input features.

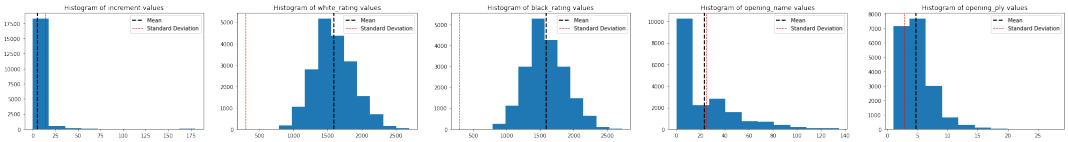

Figure 2: Histograms of input features.

3.2 Distribution of the output labels

The distribution between the games that white won and black won are pretty close to even. This will allow a good analysis without modifying the data at all.

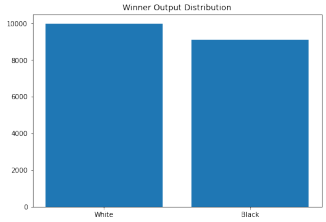

Figure 3: Output Distribution

4 Data Processing

4.1 Data Cleaning

There was a lot of non-numeric data in this data set so a lot of cleaning was necessary. Numerous fields that weren't necessary were removed, player ID, created at, last moves, white id, black id, and opening eco. Rated was converted from TRUE/FALSE to 1/0. The output feature winner was changed from white/black to 0/1 also. Victory status was converted to the following enumeration.

* 0 Mate * 1 Resign * 2 Draw * 3 Out of Time

Increment was split into two columns, Time and increment. Opening name was the hardest to modify, anything after the pipe was ignored and removed, including the variations of the opening. The opening number was also removed. Then the opening was enumerated as follows.

* 0 Slav Defense * 1 Nimzowitsch Defense * 2 99 Game * 3 24 Game * 4 Philidor Defense * 5Sicilian Defense * 6 Blackmar-diemer Gambit * 7 Italian Game * 8 Scandinavian Defense * 9 Van'tKruijs Opening * 10 French Defense * 11 Four Knights Game * 12 Horwitz Defense * 13 EnglishOpening * 14 Scotch Game * 15 97 Refused * 16 97 Accepted * 17 Robatsch (Modern) Defense * 18Indian Game * 19 Dutch Defense * 20 Zukertort Opening * 21 Vienna Game * 22 Modern Defense* 23 Blumenfeld Countergambit * 24 Queen's Pawn * 25 Crab Opening * 26 Queen's IndianDefense * 27 Gruenfeld Defense * 28 97 Declined * 29 Yusupov-Rubinstein 82 * 30 Ruy Lopez * 31Bishop's Opening * 32 Benoni Defense * 33 133Attack * 34 Alekhine Defense * 35 Caro-KannDefense * 36 Goldsmith Defense * 37 Nimzo-Indian Defense * 38 Bogo-Indian Defense * 39 King'sKnight Opening * 40 Amar Opening * 41 Paleface Attack * 42 Hungarian Opening * 43 123'sDefense * 44 Center Game * 45 Englund Gambit Declined * 46 Giuoco Piano * 47 Russian Game *48 Pirc Defense * 49 St. George Defense * 50 Bird Opening * 51 Owen Defense * 52 Semi-0 * 53English Defense * 54 89 Accepted * 55 99 Opening * 56 Polish Opening * 57 Ponziani Opening * 58East Indian Defense * 59 Reti Opening * 60 Nimzo-Larsen Attack * 61 Torre Attack * 62 CreepyCrawly Formation * 63 Elephant Gambit * 64 Latvian Gambit * 65 Trompowsky Attack * 66Englund Gambit * 67 London 82 * 68 89 Declined * 69 125 Accepted * 70 Mieses Opening * 71Carr Defense * 72 133Defense * 73 Three Knights Opening * 74 Ware Opening * 75 BudapestDefense * 76 Richter-Veresov Attack * 77 Franco-32 * 78 Van Geet Opening * 79 Polish Defense *80 Old 32 * 81 Kadas Opening * 82 System * 83 Rat Defense * 84 66 Complex * 85 Gedult'sOpening * 86 Colle 82 * 87 Nimzowitsch-Larsen Attack * 88 Neo-27 * 89 king's Gambit * 90 64Accepted * 91 Grob Opening * 92 Mikenas Defense * 93 Tarrasch Defense * 94 98 Defense * 95Anderssen Opening * 96 Portuguese Opening * 97 Queen's Gambit * 98 Old Indian * 99 King'sPawn * 100 Irish Gambit * 101 24 Opening * 102 Clemenz Opening * 103 6 Declined * 104Mexican Defense * 105 Barnes Defense * 106 Saragossa Opening * 107 Duras Gambit * 108Gunderam Defense * 109 Hippopotamus Defense * 110 Catalan Opening * 111 Lion Defense * 112Guatemala Defense * 113 Borg Defense * 114 Sodium Attack * 115 122 Accepted * 116 AmazonAttack * 117 Wade Defense * 118 Kangaroo Defense * 119 Semi-Bononi * 120 122 Declined * 121Slav Indian * 122 Benko Gambit * 124 84 Declined * 125 Danish Gambit * 126 23 Accepted * 127Lemming Defense * 128 Queen's Indian Accelerated * 129 Ware Defense * 131 Scotch Gambit * 132Czech Defense * 133 King's Indian * 134 125 Declined * 135 Canard Opening

4.2 Data Normalization

When using data for neural networks you will notice your loss does not decrease as your epoch increases. To solve this we normalize the data before we apply the tensorflow model. Tensorflow works best if the input feature data is between 0 and 1, so to achieve this and not disturb the distribution of each feature I divide each element in the feature by its maximum value in the dataset. This will make the maximum value 1 and everything else distributed the same between 0 and 1.

$$X_{new} = \frac{X}{X_{max}} \tag{1}$$

5 Building the Model

There is numerous parameters to account for when trying to building a neural network classifier. The goal is to increase the accuracy and reduce the loss of the model by training it against the given dataset. The dataset will be modified and split to try to achieve higher numbers, and the model will be layered and tested numerous times to try to optimize its performance. There is not specific procedure to figure out the best parameters except with just trial and error, and after numerous tests you will use your best performing model.

5.1 Splitting Data

The data must first be randomized so that there is a bias towards one part of the data verse another. After shuffling the data around I then split it into four parts, the training inputs, training outputs, validation inputs, and validation outputs. The validation set is used as a test to see how accurate our model actually would be, and for this test I only took 30% of the data to create it. Once the model is created from the training set it will be used against the validation set to see how accurate it is.

5.2 Adding layers

The first set in building the model requires adding layers to the Tensorflow model. Since this is a binary classification problem the last layer must always have an activation of sigmoid. The initial layer will be the control when comparing layers; it will consist of a single sigmoid layer with one neuron.

These results were compared to tests ran by adding more and more layers and neurons. Google Colab collapsed after I ran a 8 layer network starting at 512 Neurons and a batch size of about 4. Looking at these results you can see we are not getting better results past 2 layers in the network. The rest of the tests will assume that we will use this configuration of 2 layers.

Table 1: Layer Evaluation

Layer	Accuracy	Loss
1	65.38%	62.63%
2	65.14%	61.06%
3	64.52%	61.05%

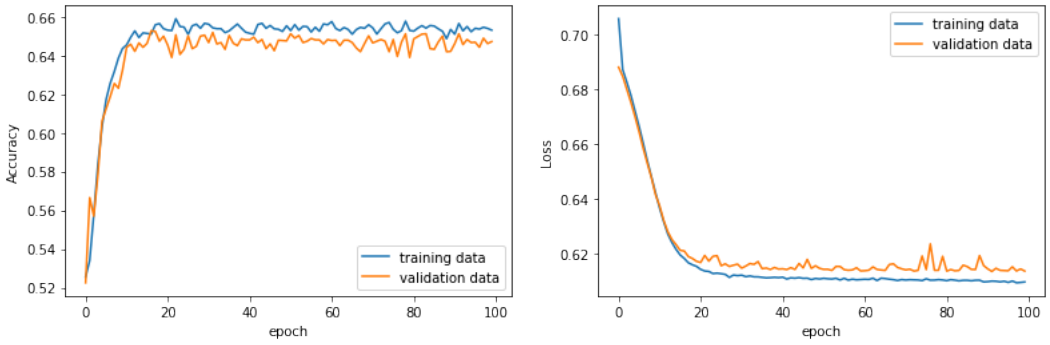

Figure 4: 2 Layer Accuracy & Loss

5.3 Linear Activation Testing

This test will include changing around the activation parameter when adding layers to the neural network. First I will make all the layers use a linear activation instead of the sigmoid. Then just the last one while the rest will be sigmoid. Then all the layers will be sigmoid, and finally just the last one will be sigmoid. As you can see from the below plots and performance table that using the sigmoid activation is better for binary classification problems.

Table 2: Activation Performance

Type	Accuracy	Loss
All Linear	63.95%	64.44%
Last Linear	64.80%	63.82%
All Sigmoid	65.21%	61.46%
Last Sigmoid	64.30%	61.98%

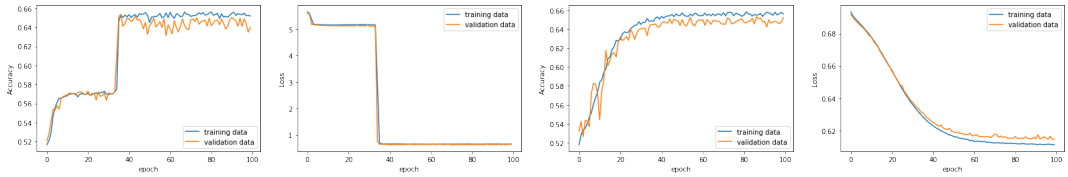

Figure 5: All Linear Accuracy & Loss on the left side and All Sigmoid Accuracy & Loss on the right

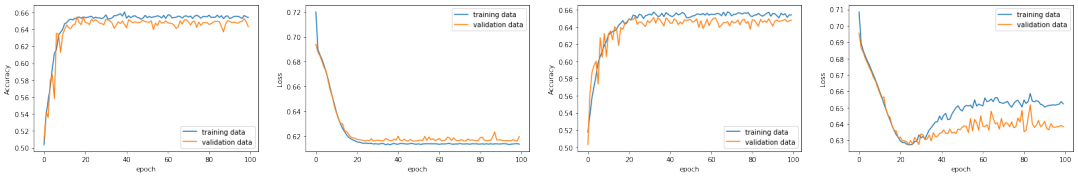

Figure 6: Last Sigmoid Accuracy & Loss on the left and Last Linear Accuracy & Loss on the right

5.4 Overfitting

Overfitting refers to a model that models the training data too well. Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data [5]. I overfitted the model by increasing the epochs to 1000 and increasing the neurons by 10 times.

Table 3: Overfitting Performance

Accuracy	Loss
63.43%	64.06%

6 Model Evaluation

The evaluation of the model consists of three attributes for each model, precision, recall and F1-score. Precision is how accurate my model is out of the predicted positive and how many of them are actually positive[6]. Recall calculates how many of the actual positives the model captures through labeling it as positive[6]. Finally F1 score allows us to see a comparison between precision and recall[6].

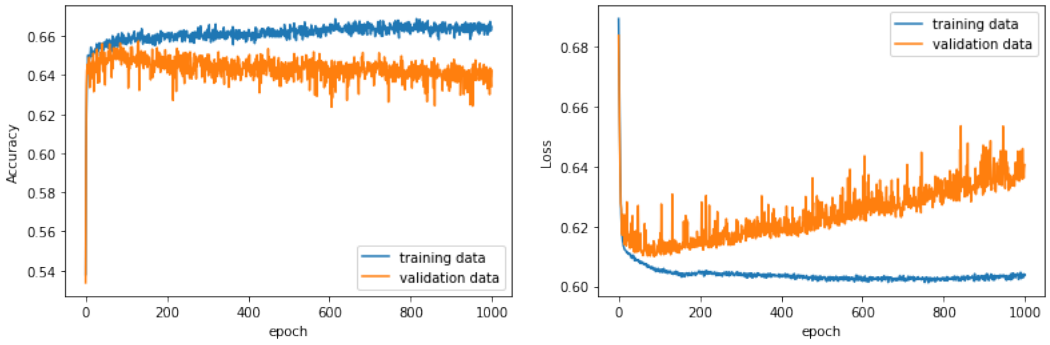

Figure 7: Overfitting Accuracy & Loss

Table 4: Layer Evaluation

Model	Precision	Recall	F1-Score
1 Layer	65.17%	58.82%	0.62
2 Layer	63.44%	63.45%	0.63
Overfit	60.19%	64.62%	0.64
All Linear	65.10%	65.10%	.65
Last Linear	65.37%	65.37%	0.65
All Sigmoid	64.60%	61.01%	0.63
Last Sigmoid	65.69%	57.62%	0.61

7 Callbacks

The goal of implementing callbacks into the code is to create a save point in time of the best accuracy the training has accomplished, and if it doesn't improve after a defined amount of time then we stop the training. This will allow us to save time training the model. In my model I used Model Checkpoint and Early Stopping. The checkpoint allows me to keep track of what was the best accuracy of my model for each epoch. Early Stopping allows my model to stop early after a given number of epochs, in my case I chose 15. The model training was improved from 100 epochs down to 81 epochs, which means the best accuracy in the model was on epoch 66.

8 Output as Input

A good test to see if your pipeline is working and it is your model that is having issues is to put your output in as input. This will show that if your model had the information then it would figure it out. Doing this in my model showed within 3 epochs accuracy was 100

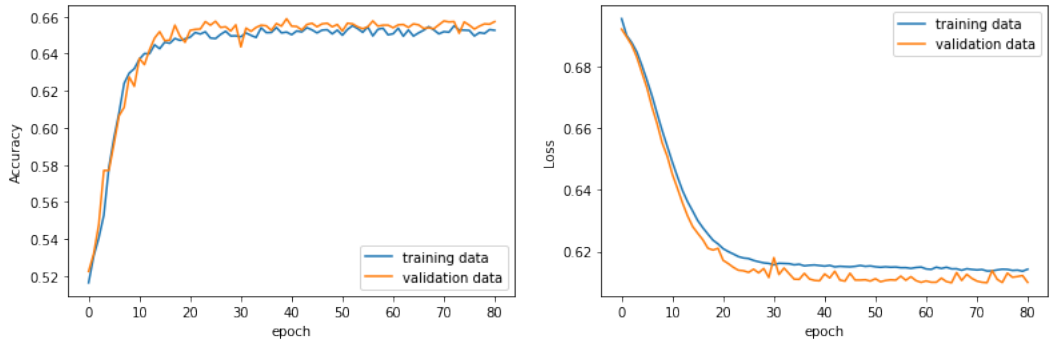

Figure 8: Callback Accuracy & Loss

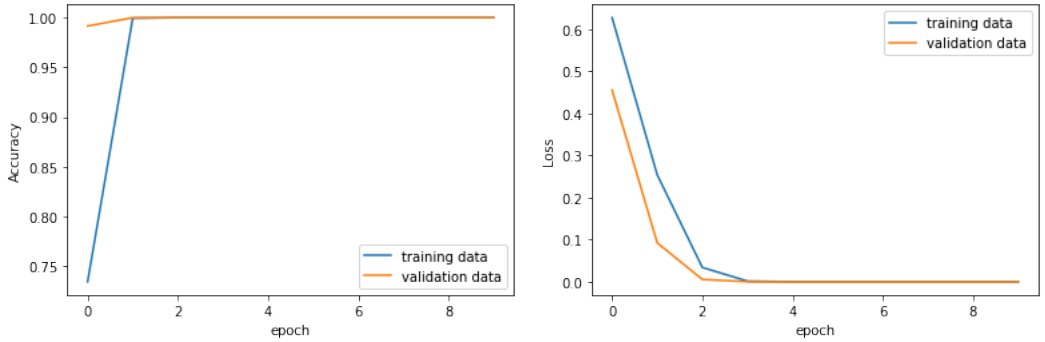

Figure 9: Output as Input Accuracy & Loss

9 Function Representing the Model

After compiling the dataset using a linear model and a single layer I was able to extract the weights from the model. Running the model.layers[0].get_weights()[0] gives me an array of all 9 of my weights for each input. By multiplying these by the input values and adding the bias given by model.layers[0].get_weights()[1], it returns the value given by model.prediction(X).

$$Y = X_1w_1 + X_2w_2 + X_3w_3 + X_4w_4 + X_5w_5 + X_6w_6 + X_7w_7 + X_8w_8 + X_9w_9 + bias \tag{2}$$

10 Feature Importance

The goal was to figure out which features were actually contributing to the model building and which were more or less useless for trying to predict the winner of a chess game. I started by running each input feature in isolation. This allowed me to generate the bar graph below which shows a majority of the importance comes from the black and white rating input features, while the other are more or less insignificant, except for maybe turns which had the third highest accuracy. After I found which features were most important I slowly kept removing the other input features from the one that had the least significance, to the once that had the most significance. This

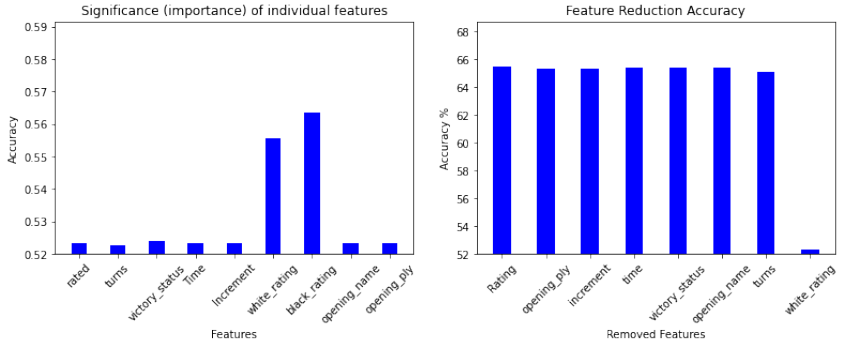

Figure 10: Significance of individual features

generated the following results which shows that removing the features had little to not significance until we removed the white_rating input feature.

This leaves the feature-reduced model with only the white and black ratings as input features. It is almost the same accuracy as the original model with a discrepancy that is withing the error percentage. Below is also a side by side comparison of the feature-reduced model (on the left) and the original model (on the right) for both accuracy and loss.

Table 5: Feature Importance

Model	Accuracy
Original	65.14%
feature-reduced	65.08%
black-rating Only	52.33%

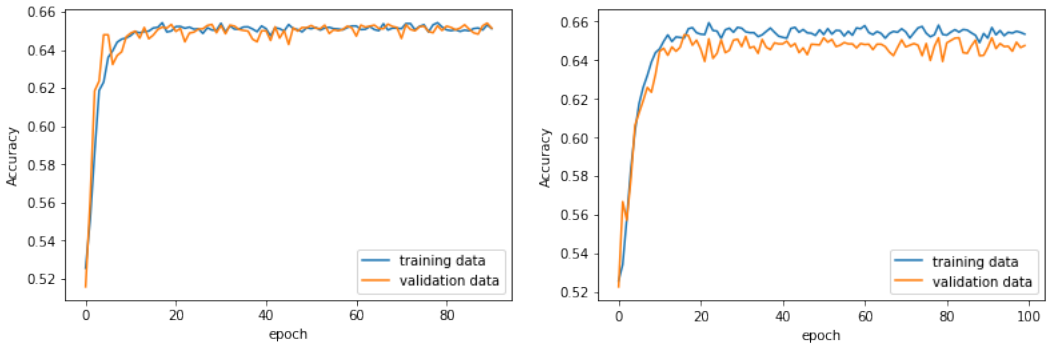

Figure 11: feature-reduced model vs. Original Accuracy

The Learning curves of both of the models are almost the same. The feature-reduced model on the left has a lot less noise compared to the original model. The feature-reduced model also gets to the max accuracy in fewer epochs than the original, which shows in its smoother accuracy graph. This is because the model has a lot less to compute in comparison since there is only 2 input features compared to the original 9.

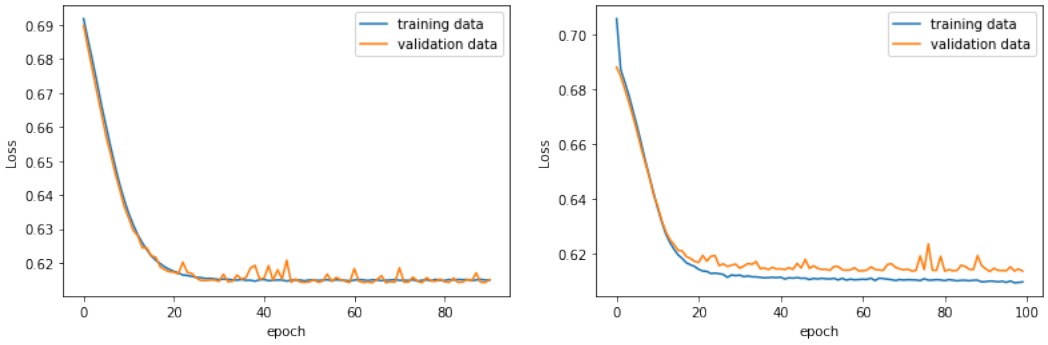

Figure 12: feature-reduced model vs. Original Loss

11 Conclusion

Throughout this paper we have explored yet another avenue of chess that can be improved and analyzed by artificial intelligence. While we found most of the data in the set was not useful for our prediction of the winner of the game the major features like Black and Whites current rating at the time of the game found to be quite important. Going into this the hope was that we could find which opening move would be best under certain conditions, but from the analysis we found that it doesn't matter which opening you do but how skilled a player is in the game, and how well they have done in the past, represented by their rating. Most of our features were not useful and eventually we remove all but just ratings and the accuracy remains the same.

The model building process was achieved easily enough through trial and error, and improved by the callback methods of Model Checkpoint and Early Stopping. The validation set could have be modified to be smaller or larger to see their difference and how it would effect the algorithm. While a wide amount of different configuration of hyper-parameters were tried, we could always improve this by doing more testing with different configurations. We also know the model is working because when we entered the output as an input it showed a 100% accuracy, which builds our confidence in the model itself. While the function representing the model is not identically the model we used, it gives us the path to create a function if needed from the weights of the model. Overall the project was a success and allows us to accurately predict the outcome of a game by 65% when we are just given black and whites ratings.

References

[1] Krešimir Josić. No. 2765: THE TURK, 2011. (accessed: 02.21.2021).

[2] ComputerChessChamps. Chess.com Computer Ratings: Nov. 2020, 2020. (accessed: 02.21.2021).

[3] ComputerChessChamps. Stockfish, 2021. (accessed: 02.21.2021).

[4] Mitchell J. Chess Game Dataset, 2018. (accessed: 02.21.2021).

[5] Jason Browniee. Overfitting and Underfitting With Machine Learning Algorithms, 2019. (accessed: 04.11.2021).

[6] Koo Ping Shung. Accuracy, Precision, Recall or F1?, 2018. (accessed: 04.12.2021).