Predicting Chess Elo Using Neural Nets

Camille Chow and Chris Kim

Abstract

We implemented a neural network to predict the chess Elo scores of two players using the moves from a single game based on various experiments to determine the best data preprocessing method, as well as the best network architecture. Our model is trained on 25,000 games from a Kaggle dataset. The model predicts the Elos from the Kaggle test set with a mean absolute error of 193, and correctly identifies the stronger player from the 2018 World Chess Championship. While our model's performance does not improve upon the results of the original Kaggle competition, they provide a proof of concept that neural networks can be applied in this area with relative success. Additionally, we have outlined ways in which the model can be improved.

1. Introduction

The Elo rating system is a method for calculating the relative skill levels of players in zero-sum games such as chess [2]. A player's Elo rating increases or decreases depending on the outcome of games played; the winner takes Elo points from the loser. The number of points transferred depends on the difference in the Elos of the players. If the winner has a much higher Elo than the loser, than few points are taken, but if the winner has a much lower Elo, then many points are transferred. Typical Elo scores are around 800 for a beginner, 1600

for a mid-level player, and 2400 for a professional. The highest Elo score ever achieved was 2882 by Magnus Carlsen in 2014.

The field of machine learning has long held ties to the game of chess, particularly with the development of chess engines and that learn to play the game [4][6]. However, there has been significantly less focus on utilizing machine learning to predict chess Elo. Some work has been done to predict Elo in games such as Go [1] and soccer [3] in order to improve predictions of match outcomes.

To the best of the authors' knowledge, the only work that uses machine learning to predict chess Elo scores has been done on Kaggle [7]. This competition held in 2014 challenged participants to predict the Elo ratings of two chess players based solely on the moves of one game. Kaggle provided a dataset of 50,000 games with move sequences and Stockfish scores, as well as the Elos of each player. The submissions were evaluated based on the mean absolute error of the predicted Elos, the lowest of which was 155.8. For the most part, all of the submissions to this Kaggle competition relied on traditional machine learning techniques to perform the task. Our project

differs in that it uses neural networks to predict Elo scores from a single game.

The remainder of the paper is organized as follows: Section 2 details the experiments we conducted to determine the best data preprocessing techniques and network architectures. Section 3 shows the results from applying our model to the Kaggle test set, as well as data from the recent World Chess Championship. Section 4 draws conclusions about our results, and section 5 details possibilities for future work.

2. Experiments

2.1 Kaggle Dataset

As mentioned before, we utilized a Kaggle dataset consisting of 25,000 games for training and 25,000 games for testing. Each game in the training set is labeled with the white and black Elo. All the games contain the sequence of moves played as well as each move's Stockfish evaluation.

Stockfish is considered one of the world's strongest chess engines. It uses alpha beta search to determine the strength of a move given the board state, and assigns the move a score representing the advantage of the white player in centipawns, or cp (1 pawn = 100 cp). Thus, a positive Stockfish score indicates that white has the advantage, while a negative score indicates that black has the advantage. The larger the value of the score, the larger the advantage.

Because our neural network requires moves to be in a machine interpretable format, the Stockfish scores provided a convenient quantitative value to pass as input. Though there are other ways we could have formatted moves (see Future Work), we only pursued this method due to time constraint.

2.2 Preprocessing

Various experiments were conducted to see how data preprocessing affected our model's performance. The first issue we addressed was how to pad games. Since the neural network requires a fixed game length, we decided to pad all games to the maximum game length of 330. Our first experiment tested two different padding methods: padding with the last score in the Stockfish sequence, and padding with a constant value. It was determined that padding with a constant value provided significant gains over padding with the last value. Further experiments determined that the best value to pad with was 850, however the value of the constant had a less significant impact on results than the decision to use the constant. We padded all games in which white won with +850, all games in which black won with -850, and all ties with 0.

The next experiment tested the way in which we labeled the training data. We passed the labels as a tuple of two Elo scores, one for each player. We tested two methods of ordering this tuple: white Elo first, and winning Elo first. It was determined that always listing the white Elo first resulted in better performance. This is likely due to the fact that tie games were not properly handled (in the case of a tie, a random

order of Elos was chosen). Additionally, while the order of the Elos was switched, the order of the moves was not, which may have confused the model as well.
The next question we wanted to address was whether the first few moves of the game were necessary in predicting Elo, as opening moves are generally very similar from game to game. We experimented by removing the first 2, 4, and 6 moves (only even numbers so as not to offset the starting player) of each game, however, none of these experiments provided gains over the original model.

Our final preprocessing experiment attempted to aid the model in learning the differences in Elo scores by normalizing the Elo data (subtracting the mean and dividing by standard deviation). However, our architecture was unable to learn the small differences in the labels and only output the average (zero). Therefore, this idea was abandoned.

2.3 Architectures

We first experimented with neural network architectures that only included dense layers. Many experiments were performed to find the optimal layer number and size. Dropout was included and tuned for performance boost as well. The most notable improvement came from switching an activation function in a hidden layer from a relu to and elu, reducing the mean squared error by half and resulting in a mean absolute error of 215. However, upon closer inspection, it was found that this resulted in a much smaller variance in output Elos, as the model learned to simply output Elos that were close to the average of the training set (see Figure 1).

Beyond this, most of our experimentation with this architecture (tuning the optimizer, batch size, epochs, etc…) produced nonexistent or negligible gains.

In an attempt to improve our results, we decided to additionally utilize convolutional layers. We conducted an initial test to see whether or not the convolutional layers had potential by implementing a simple convolutional neural network architecture based on a model used to classify the CIFAR10 dataset. We saw the original model as a good starting point, since it is relatively small and has a lot of room for improvement. The mean absolute error was 240, a large improvement over our starting point with dense layers (298).
With this promising result, we added more convolutional layers and adjusted the filter size and kernel size of our model. With these experiments, we improved our mean absolute error to 210. The convolutional network demonstrated much more promise than only using dense layers, so at this point we fully committed to the convolutional approach.

Upon further experimentation, which included tuning of parameters and implementing different types of layers like MaxPooling1D, we brought the mean absolute error down to 193. This error was achieved while still maintaining the variance of the training Elo distribution (see Figure 1). We believe that there is much room for improvement, and with the right convolutional architecture, the mean absolute error can be further reduced.

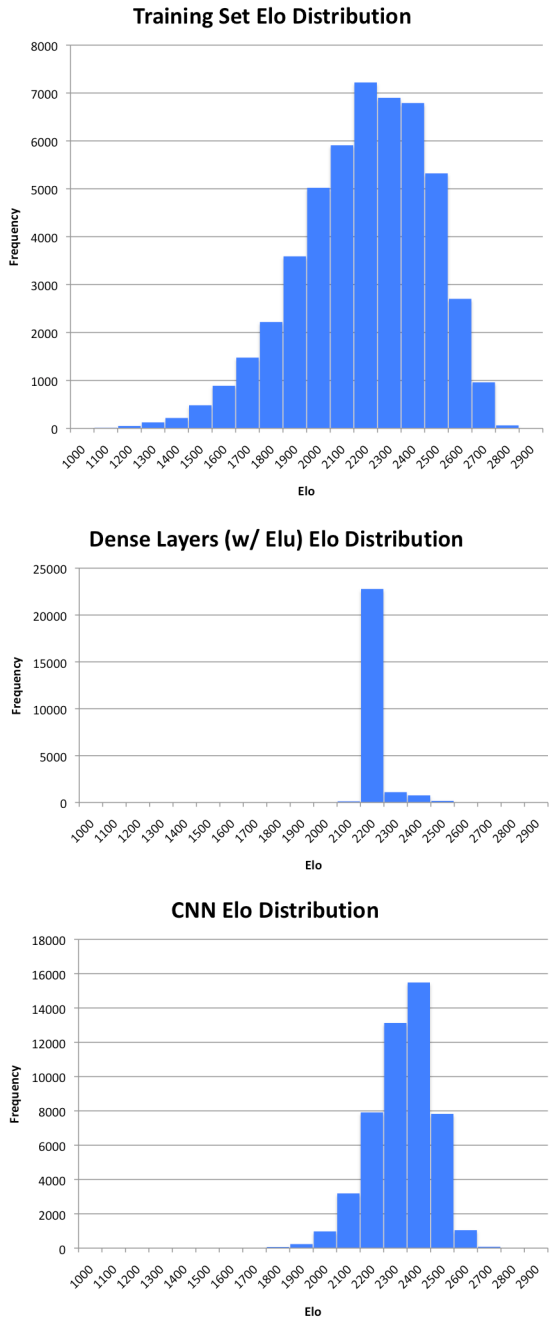

Figure 1: Elo distribution histograms.
Top: Elo distribution of Kaggle training set.
Middle: Distribution of Elo predictions from dense model with elu activation (mean abs. error = 215).
Bottom: Distribution of Elo predictions from convolutional model (mean abs. error = 193).

3. Results

In order to test our model, we used the Kaggle test set and submitted our predictions to the online scorer which provided the mean absolute error. Additionally, we compiled data from the 2018 World Chess Championship and used that to predict the Elos of two of the top chess players.

3.1 Kaggle Test Set

The test set provided by Kaggle was the dataset we used to determine whether or not the changes to our models resulted in improvement. We compared our generated mean absolute error to the top score (156) of the contest as well as the range of scores for the top 50 (156-193). Our goal was to achieve a score that would break the top 50 barrier for this test set.

The convolutional neural network model provided the best result (193) we achieved for this competition, which just broke the top 50 mark for the kaggle competition. The model seemed to start converging at around 35-40 epochs, so we chose to run the model with 50 epochs. While we do believe that the convolutional approach has a lot of room for improvement, we think that solely utilizing stockfish engine evaluations does not give us enough data to contend with the top mean absolute error of the contest.

Many of the top performers of the competition utilized more than the provided data, such as features related to the position of the board and generating better stockfish evaluations. For

context, the provided stockfish dataset was generated with 1 second of engine time per move, and the third place finisher generated a dataset with 2.5 seconds of engine time per move, which leads to better evaluations. We feel that if we also implemented more data than just the provided stockfish evaluations, we can achieve better results. This is detailed in section 5, where we discuss how we can potentially achieve lower mean absolute error in the future.

3.2 World Championship Data

In addition to running our model on the Kaggle test dataset, we wanted to see how our model would perform in other scenarios. Two weeks prior to the conclusion of this project, the 2018 chess world championship had just reached its end. We saw this as an opportunity to see how our model would interpret the gameplay of the two current best chess players in the world.

Fifteen games were played in the world championship. Twelve were classical games, in which each player has a large amount of time to make moves and complete the chess game. The remaining three games were rapid games, in which each player has only twenty five minutes, in addition to a small time increment per move, to complete the chess game. Since the players have much more time to think in the classical time control than in the rapid time control, the players demonstrate a drop in gameplay when entering the rapid session. While we were interested in seeing if our model would be able to accurately generate Elos close to the Elos of the contenders, we were more interested in seeing if our model would be able to capture the drop in gameplay from the classical games to the rapid games through their Elo predictions.

Players	Actual Elo	Predicted Classical Elo	Predicted Rapid Elo
Magnus Carlsen	2835	2250	2141
Fabiano Caruana	2832	2243	2039

Table 1: Model Elo predictions of the gameplay compared to the actual Elos from the 2018 Chess World Championship. The average of the generated Elo predictions for the twelve classical games played are noted in the third column. The average of the generated Elo predictions for the three rapid games played are noted in the fourth column.

Table 1 shows the average Elo results of our model for the classical games and the rapid games in comparison to the contenders' actual Elos. While we were displeased with the large mean absolute error (over 500 points off), we did expect the error to be considerably more than the error for the Kaggle test dataset. Since the training data for our model contained a miniscule amount of games for Elos over 2800 (only 28 players out of 50,000), it is reasonable that the model was not able to capture the high level of gameplay of the world championship.

However, the model was successfully able to capture the small difference between the Elos of the contenders, the outcome of the games and the decrease of gameplay level from the

classical to rapid session. The two competitors are only three Elo points away from each other; in comparison, our model predicted a difference of seven points between the two. Since the twelve classical games resulted in draws, the outcome of these games may have had a big influence in why our model was able to capture the close level of the two players. In the rapid portion, our model correctly predicted a higher Elo for Magnus Carlsen, who outperformed Fabiano Caruana in all three of these games. Most importantly, our model was able to capture the decrease in level between the two time controls. While the previous observations can be attributed to the model's basic ability to attribute draws to potential closeness of Elos and wins to a comparatively higher Elo, this observation clearly shows that our model is able to interpret the content of the games themselves to a certain extent.

4. Conclusions

In this paper we present a deep learning approach to predicting the chess Elo scores of two players only using the moves played in a single game. Using the training dataset provided by a Kaggle contest and stockfish evaluations of each game in the dataset, we utilized deep learning to achieve a mean absolute error of 193. We also tested our model with the recent games from the 2018 chess world championship, in which the convolutional neural network was able to discern a drop in performance when comparing two different time controls. In terms of the neural network architecture, we first experimented with architectures that only consisted of dense layers, which gave us mediocre results. We then switched to a convolutional layer approach and were able to break the top 50 barrier of the Kaggle contest.

The model has room for improvement in regards to its architecture. Additionally, more experiments with data preprocessing techniques and usage of different data can potentially improve the results of our deep learning approach. However, we believe that it is not feasible to accurately predict Elo scores with just the chess moves played in a game. One reason may be the inconsistent nature of human performance. For instance, a player might play a much better game of chess if the game begins with a sequence of moves that he or she has spent a great amount of time studying, which may result in a performance that does not reflect the player's Elo. The opposite may also occur, in which a player fumbles into a chess opening that he or she has spent little to no time preparing for. Nevertheless, more complex problems have been solved with deep learning; with continued research in the field and/or the rise of new technology, the problem of "predicting chess Elo with gameplay" may be solved in the near future.

5. Future Work

Due to the time constraints on this project, we were unable to experiment with certain methods that could potentially improve our results.

Instead of using the scores produced by the Stockfish Engine, the model may be able to produce better results if we use the Stockfish ranking of moves. This ranking would be produced by the Stockfish engine first ranking the Stockfish scores of all possible moves, and then matching the move played with its ranking. Since higher Elo players are more likely to play the best or close-to-best possible moves, this data format may be more clear for the neural network to learn and associate with the actual Elo ratings. Unfortunately, this dataset of rankings would take an extremely long time to generate with a single Stockfish Engine (quick approximation was over 50 days), so we were unable to experiment with this approach.

Additionally, rather than using Stockfish scores at all, it may be more appropriate to input moves as one-hot encodings (eg: an 8x8x32 matrix representing the state of each piece). With the data in this format, it could additionally be useful to append coordinate information to the move data using the "CoordConv" method [5] in order to improve translational variance. This method may lend itself better to our convolutional model and prove to be a better representation of the data than move scores.

Acknowledgements

We would like to thank Chris Curro for pointing us to the Kaggle dataset, providing his insight that prompted many of our experiments, contributing to the code for the CNN, and teaching us the theory that allowed us to perform this research. Additional thanks to George Ho and Gordon Macshane for providing their input as well.

References

[1] Coulom, Rémi. Computing "Elo Ratings" of Move Patterns in the Game of Go. *ICGA Journal,* vol. 30, no. 4, pp. 198-208, 2007.

[2] en.wikipedia.org/wiki/Elo_rating_system.

[3] Herbinet, Corentin. Predicting Football Results Using Machine Learning Techniques. 2018.

[4] Lai, Matthew. Giraffe: Using Deep Reinforcement Learning to Play Chess. *arXiv preprint arXiv:1509.01549.* 2015.

[5] Liu, Rosanne, Lehman, Joel, Molino, Piero, Petroski Such, Felipe, Frank, Eric, Sergeev, Alex, and Yosinski, Jason. An Intriguing Failing of Convolutional Neural Networks and the CoordConv Solution. *arXiv preprint arXiv:1807.03247*, 2018.

[6] Thrun, Sebastian. Learning to Play the Game of Chess. *NIPS'94 Proceedings of the 7th International Conference on Neural Information Processing Systems, Pages 1069-1076.* 1994.

[7] www.kaggle.com/c/finding-elo