A Metric of Chess Complexity

FM David Peng

In this paper, we propose that popular chess websites like chess.com and lichess should use neural networks to develop a numerical measure of chess position complexity. We first discuss the benefits of such a measurement for chess and then show that an accurate numerical metric of complexity can be developed even with a relatively small dataset. We then discuss the technical details behind developing such a metric, and why a two-network approach may be preferable to previous one-network regressions.

Acknowledgments

Special thanks to the previous work of Goldammer[1] showing that neural networks can predict position complexity to a high degree of accuracy, which inspired this project.

1. Introduction

1.1 The First Chess Revolution: A Metric of Position Value

Modern chess engines like Stockfish have revolutionized the way we play chess by providing a fast, automated way to quantify how good or bad a position is. This metric lets us quickly find mistakes in games, generate free online tactics, track the status of top grandmaster games in real time, and make significant progress in opening and endgame theory. The ability to measure the value of positions allowed us to gain deeper insights in those positions, which has changed our understanding of every part of chess.

1.2 The Second Chess Revolution: A Metric of Position Complexity

A fast, objective measure of position complexity would similarly revolutionize chess. Because modern engines evaluate positions in a vacuum without humans, they are often criticized for lacking human reasoning. For instance, two positions with similar Stockfish evaluations may feel completely different to chess players:

[1] https://github.com/cgoldammer/chess-analysis/blob/master/position_sharpness.ipynb

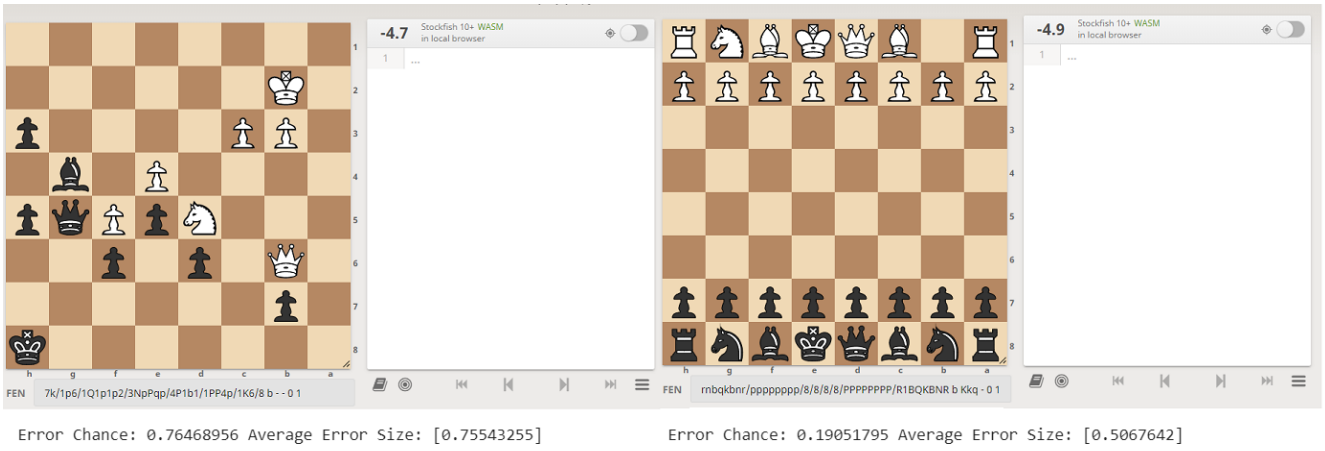

Figure 1: Two side-by-side positions with the Stockfish evaluation (depth 19) on the top right of each board and our complexity measurement on the bottom. On the left is a tactic with a rating of 2478 from ChessTempo (ID #166037), a free online puzzle service. On the right is the starting position with the White knight on b1 removed. In both positions, it is Black to move.

In this case, our measure of complexity provides more information about the position than the Stockfish evaluation. In the left position, we predict a high chance (76%) of making a large error (75 centipawns). In the right position we predict a low chance of making an error (19%), but if an error is made, it will be moderate (50 centipawns). This is reflective of how a chess player would understand these two positions.

Put simply, if the Stockfish evaluation allows us to understand how good a position is, our complexity metric allows us to understand how difficult a position is. We propose several applications of such a metric:

1.2.1 Online Puzzle Generation

Free online tactics are currently generated by engines like Stockfish, which search through games to find positions where only one move is evaluated to be winning[2]. These puzzles are then rated in difficulty by chess players who attempt to solve it (successfully solving a puzzle lowers its rating, failing to solve a puzzle raises its rating). A measure of position complexity provides many improvements to this model:

1. **Non-Tactical Puzzles**
   Modern engines can only generate tactics because they cannot understand position difficulty. With a metric of position complexity, we could identify complicated puzzles from any game. These puzzles would represent every type of challenging position that occurs in games, not just tactics.

[2] There are more human-coded criteria to ensure these puzzles are not easy to solve. A more comprehensive list can be found at https://blog.playmagnus.com/generating-tactical-puzzles-with-stockfish-and-chess-jl-part-i/.

2. **Puzzle Ratings**
   With a metric of position complexity, we would know the difficulty of any generated puzzle and give it an accurate rating before it is released to users. This would save a lot of time compared to the current model, where a puzzle overrated or underrated has to be attempted by many users before being adjusted to its correct rating.

3. **Holistic Opening Preparation**
   With objective position evaluations by modern engines, chess players frequently memorize openings instead of understanding them. A measure of position complexity would remedy this problem by generating puzzles that occur from popular openings. Specifically, we would first identify a subset of games played in an opening of interest and then scan those games for complicated positions. Giving chess players a set of difficult positions that occur from openings would give them a much more in-depth understanding of those openings.

1.2.2 Selective Search

Authors of chess books manually search through games to find interesting and complicated positions. We can automate this process by evaluating the complexity of positions from chess games to create a database of interesting and complicated chess positions that chess authors can then search through. This would cut the time-consuming process of manually searching through chess games to find complicated positions.

1.2.3 Difficulty by Rating

With a large database of analyzed chess games, we could factor in rating to our model of position complexity (explained in more detail in Section 2), meaning we could predict how difficult a position is to a player of any rating. This would have many significant uses:

1. **Chess Improvement Theory**
   A model of complexity that factors rating would allow us to find positions that are difficult for lower-rated players but are easy for higher-rated players. If a human grandmaster were to analyze the similarities in those positions, we could identify the concepts that are vital to improving one's chess. This would allow for much more specialized and effective chess training.

2. **Rating Estimation**
   A model that factors rating could analyze a player's games and estimate their elo performance. If we learned what positions tend to be hard for one rating category but easier for another, we could also generate a diagnostic chess exam that not only predicts a user's rating but identifies key concepts for improvement.

3. **Extensions to Other Categories**
   With a large database of chess games, our model of chess complexity can factor in any variable, like the amount of time a player has on the clock. This data is tracked by all major chess websites and would give significant insight into what positions require a lot

of thinking and which positions can be played intuitively with little time. Other categories of interest include rating difference to see if there are unique strategies that should be employed when facing a much higher or lower rated player.

1.2.4 Personality Analysis

Naturally, a metric on the difficulty of positions to chess players should give us insight into how we play chess. With a metric of chess complexity, we could analyze the games of chess players and determine their chess personality: whether they tend to dive into complex positions or avoid them. We could determine the level of complexity at which different players tend to make errors, and levels of complexity where they begin to seek simplifications. By understanding how people play with respect to complexity, we could then create chess engines that play with human personalities.

1.2.5 Opening Preparation

A metric of chess complexity would encourage chess players to take more risks and pull their opponents into positions that are complicated, but perhaps not optimal. Players could also tailor different strategies depending on their opponent's tolerance of complexity, or personality. Through the ability to evaluate the tradeoffs between optimal play and complexity, we would expect new, more complex chess games.

1.2.6 Spectating

With Stockfish evaluations and best moves next to top chess games, modern-day spectators severely underestimate the complexity of the positions they spectate. A metric of complexity would remedy some of the issues created by Stockfish and give spectators an extra dimension of information to understand and enjoy the games they watch.

1.3 Chess.com and Lichess: Drivers of the Revolution

We propose the usage of neural networks to estimate complexity as both the probability of making an error as well as the average size of an error if it were to occur. As complex positions can involve deciding between the best move and an alternative, it is insightful to know both pieces of information.

We trained our model on 25,000 games on Kaggle compiled in 2015[3] (more details in Section 2) analyzed by Stockfish at 1 second per move. Players of the games varied in rating from 1000 to 2800. We display our results below:

[3] https://www.kaggle.com/c/finding-elo/data

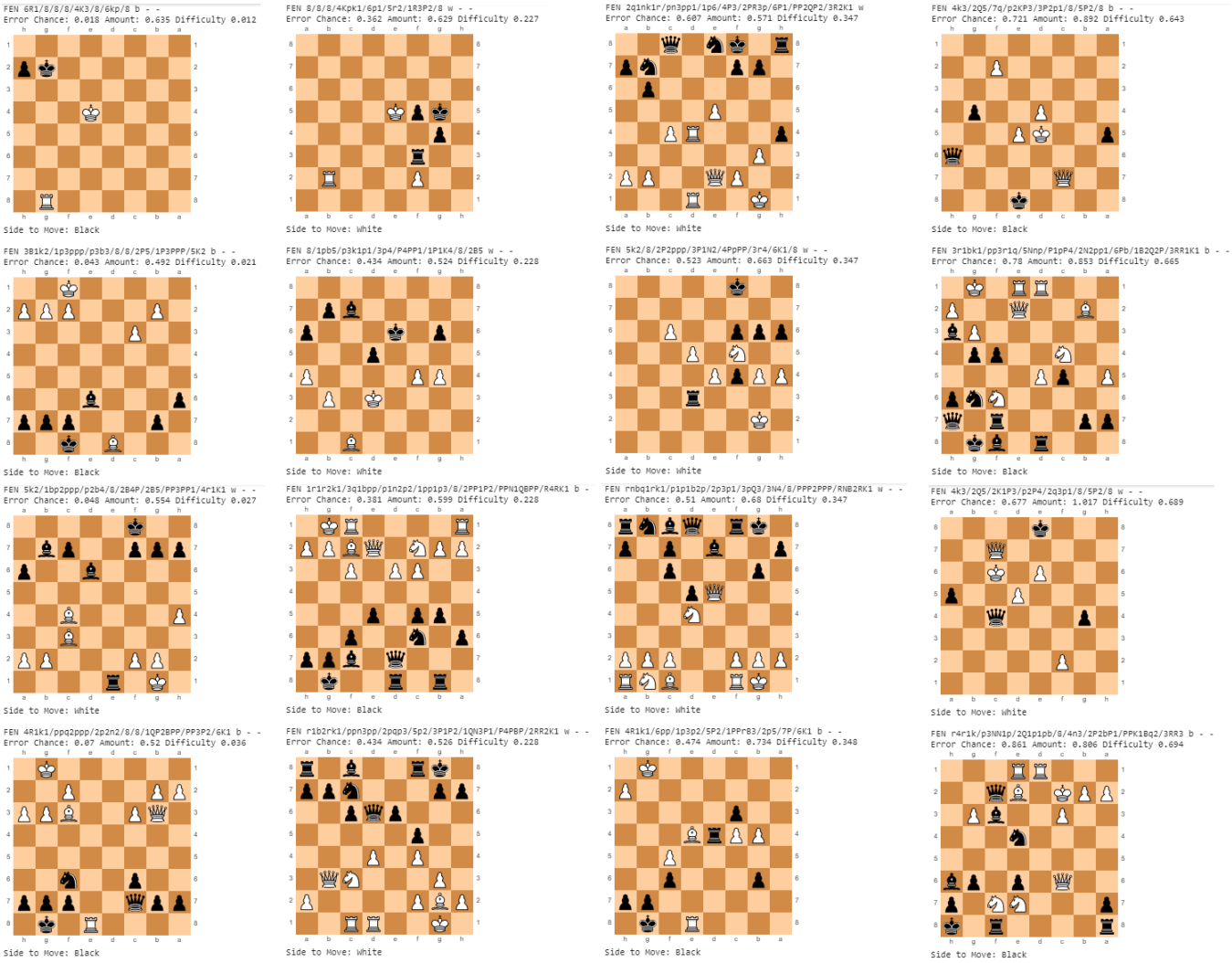

Figure 2: 16 positions, varying in predicted complexity, selected from 1000 random positions from the 2019 World Cup. On the far-left column are 0th percentile difficulty positions, to the right are the 33rd percentile difficulty positions, then the 67th, then the 100th on the far right.

Our networks are quite successful in classifying positions by complexity, as measured by the probability of making a mistake multiplied the average amount of a mistake. The primary factor in the accuracy of neural networks is the amount and quality of the data used. While we used 25,000 games evaluated at 1 second per position, websites like chess.com and lichess have **over a billion games** in their database[4], a large portion of which are already analyzed in depth by users. We therefore call upon these chess websites to use their vast amounts of data to develop an accurate measure of complexity which will revolutionize chess.

[4] https://database.lichess.org/

2. Methods

2.1 Dataset

We take 25,000 games from Kaggle pre-labeled with Stockfish evaluations and create a database of positions. Similar to past work, we omit winning positions (evaluation exceeding 350 centipawns) and opening theory (positions before move 12). For each position, we calculate error as the difference in evaluation from the current position to the next. We then scale errors so they do not exceed 100 centipawns (errors can reach up to thousands of centipawns but any move losing more than 100 centipawns can be considered a mistake), and round errors less than thirty centipawns to zero. We round down errors because Stockfish evaluations fluctuate at 1 second per position, meaning perfect moves are sometimes flagged with errors up to 30 centipawns.

2.2 Board Representation

We represent each chess position by a 1x769 binary sequence of bits representing the location of the pieces and the side to move. We split the chess board into 12 separate 8x8 chessboards, one for each color piece (6 types of pieces, 2 colors). Each square of the 8x8 boards has a 1 if that type of piece is present on that square or a 0 if it isn't. The turn is represented by a 0 if it is Black to move or a 1 if it is White to move. This results in 769 binary inputs (12 x 8 x 8 + 1).

2.3 Data Skewness

```
(array([996201.,      0.,   7245.,  53868.,  40409.,  31300.,  18111.,
         15385.,  11603.,  57912.]),
 array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
 <a list of 10 Patch objects>)
```

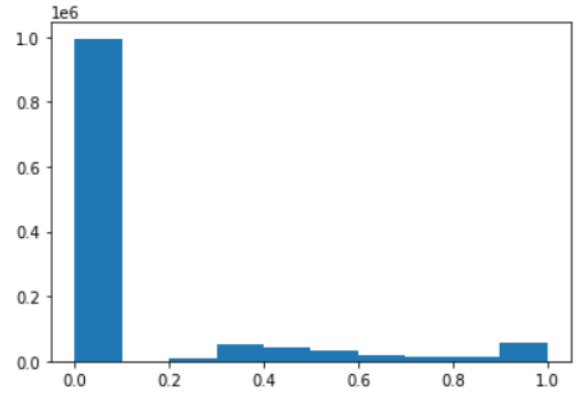

Figure 3: A histogram of number of training examples and the associated error amounts.

After processing the data, we observe that a vast majority of the positions in our dataset have 0 error. This skewness in the dataset creates a problem for the neural network, which will not learn to distinguish complexity between positions but instead predict low amounts of error for every position. To solve this problem, we split our data into two sets: no error (the bar at 0) and error.

We also use two neural networks: a classification network to predict whether or not an error was made (whether it falls within the 0 bar or outside of it), and a regression network to predict the magnitude of an error if it were made. Thus, the regression network trains only on positions where an error was made, and ignores all positions where no error was made. To further reduce skew, we train the classification network on a dataset of 50% positions with no error and 50% positions with error by undersampling from positions with no error. This ensures the classification network does not predict no error for all positions to reduce the chances of making a classification error.

2.4 Model Architecture

For the classification and regression networks, we borrow the architecture from Sabatelli et al. (2018)[5]. We use a three hidden layer dense network with 1048, 500, and 50 units in each layer respectively. We apply a 20% dropout regularization to each layer to prevent overfitting. Each hidden layer is connected with the ReLU activation function, and the output layer is Softmax for the classification network and Linear for the regression network. The Adam optimizer was used with a learning rate of 0.001.

2.5 Results

2.5.1 Overall Results

The classification model achieved an accuracy of 60.6% on the test data (classification loss of 0.65) and the regression model achieved a mean-squared error loss of 0.058 on the test data (mean absolute error of about 0.22). As demonstrated before (section 1.3), the models are quite successful in separating complex positions from simple positions, where complexity is measured as the probability of an error given by the classification network multiplied by the average error amount given by the regression network. While this metric is a relative measurement of complexity and is correlated with actual expected centipawn loss, it is not equivalent to average centipawn loss (due to removing skewness in the data and scaling errors).

2.5.2 Results by Elo

We also split the 25,000 games into 6 elo categories (<1600, 1600-1900, 1900-2200, 2200-2500, >2500) and trained six sets of classification and regression networks on each category of data. We found that there was substantial overlap between the hardest positions suggested by the neural networks from each rating category. This indicates that while there are positions that are easier for higher rated players but difficult for lower rated players, some positions are difficult for both lower and higher rated players. We also found that network performance decreased when training on rating categories due to the reduction in data, so we stuck to the model that trained on all 25,000 games. However, models trained on much more

[5] https://www.ai.rug.nl/~mwiering/GROUP/ARTICLES/ICPRAM_CHESS_DNN_2018.pdf showing that neural networks can approximate the evaluations of Stockfish

data are likely to discern differences between rating categories. Rating can be factored into a model either by first splitting the game data by elo category and training multiple sets of neural networks, or by using rating as a direct input into the neural network. Other factors, like time spent, can be factored into a model similarly.

2.6 Comparison to Previous Results

2.6.1 Neural Network Methods

We implemented some changes from the previous work of Goldammer that should improve performance in theory, such as more stable evaluations (running Stockfish for 1 second/move instead of 0.1 seconds per move) and using two neural networks to remove data skewness. While there are some positions where our model outperforms Goldammer's model (one shown below), for the most part our model ranks complex positions as well as Goldammer's model[6]. However, our two-network approach has significantly increased the accuracy of our prediction. As suspected, Goldammer's model trained on a dataset with a large majority of low error positions, which resulted in both a low mean error and blunder value.

As an example, try to find the best move for Black in the following position.

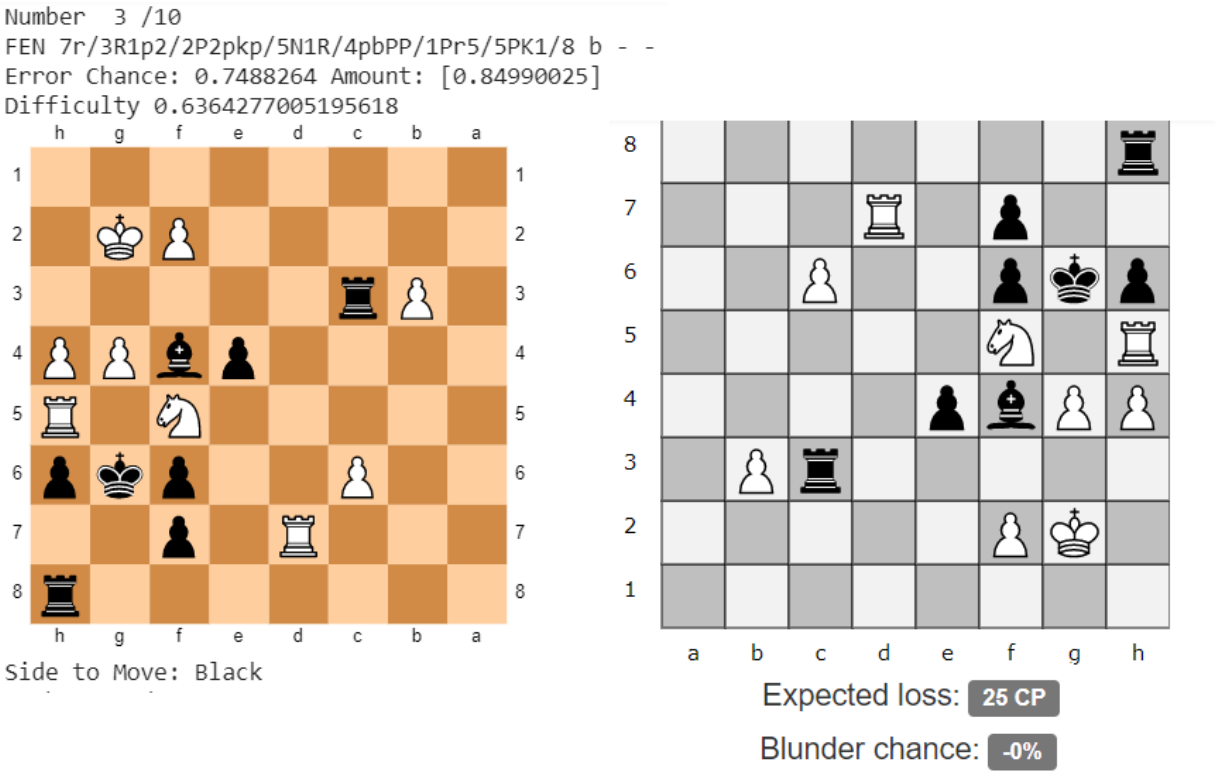

Figure 4: A position evaluated by Goldammer's model on the right for a 1000 elo player and our model on the left.

[6] https://chessinsights.org/analysis/

Goldammer's model predicts that for most players, there is a 0% chance of a blunder and an expected loss of 25 centipawns. However, the only move that gives Black a fighting chance is ...e3, which is a difficult move to spot. The next best move (...Rc2) loses 80 centipawns, and the next best move loses 160 centipawns. Evidently, Goldammer significantly underestimates this position's complexity, and a chess player interpreting his results would conclude that this a very simple position. In contrast, our model is much more accurate, predicting a 75% chance of making an error with the average error being around 85 centipawns.

2.6.2 Stockfish-Based Methods of Complexity

Existing research by Guid and Bratko (2006)[7] measures complexity as the change in position evaluation as Stockfish depth increases. Goldammer established that there is only a weak correlation between neural-network methods of evaluating complexity and Stockfish-based methods. We also believe that as modern engines doesn't think like humans, it is unlikely for Stockfish to understand which positions chess players find difficult.

2.7 Improvements

Beyond increasing the amount and quality of training data, several improvements can be made to our model. One minor error noticeable in the results of section 1.3 is that messy positions with only one legal move are identified as complex. We suspect that this may be fixed by adding more handcrafted features to the model such as number of legal moves or whether the current player is in check. We also suggest testing different models and learning rates.

3. Conclusion

This paper provides strong evidence that neural networks can measure position complexity to a high degree of accuracy, and that such a metric would provide massive benefits to chess, similar to the evaluations provided by modern engines.

However, chess players will also be critical to driving this revolution. Because engines do not think like humans, evaluations without human reasoning have no meaning. We know the futility of understanding the logic behind computers, or attempting to memorize computer suggested best moves in long opening variations. I see modern engines as rulers and scales, and we must use them to determine the laws of chess. Chess players must use these tools to find positions commonly misevaluated and understand why we misevaluate them. We need more human logic and less memorization. My hope is that a metric of complexity will supplement modern-day engines and allow us to better understand how we play chess.

You can find models, samples of my code, and a demo software of how a measure of complexity can generate puzzles on my GitHub page: https://github.com/Amethyst-Cat/ChessComplexity/.

[7] https://ailab.si/matej/doc/Computer_Analysis_of_World_Chess_Champions.pdf