Using Deep Learning to Predict a Player's Rating on Online Chess Websites

CS 230 Project Milestone Report; Robert Bennett (Solo group)

Code Available: `https://github.com/RKABennett/CS_230_project.git`

1 Introduction

Chess websites use a rating systems to quantify a player's performance. Recently, many top-level players have released popular YouTube videos where they inspect previously played games and attempt to guess the players' ratings. The idea is simple: top-level players have sufficient knowledge of chess to determine whether a player is playing well or not, allowing them to guess (often with reliably high accuracy) the players' ratings based only on a single game, i.e., without knowing anything about either player beforehand. The goal of this project is to use deep learning to accomplish the same task. Specifically, the program will aim to output both players' chess ratings based on a single game that they play without any prior knowledge of the players' skill levels. Practically, such a program would be useful within the chess community for three main reasons: cheating detection (to see if a player is playing well above their actual ranking), rating initialization for new accounts, and gaining understanding of chess players of different ratings.

2 Dataset

This project makes use of Lichess.org's open database, which is a free database released under a Creative Commons CC0 license containing detailed information about the vast majority of games that have been played on Lichess.org since 2013. As of November 2022, this database is 989 GB in size and contains data from 3.8 billion games. An example of a game from this database is shown below. Note that some irrelevant features have been removed (e.g., players' usernames), the format has been slightly modified to improve readability, and the game is truncated for brevity.

```
1     [Site "https://lichess.org/e5ZEY0QD"]
2     [BlackElo "1920"]
3     [ECO "C40"]
4     [Termination "Normal"]
5     [TimeControl "600+0"]
6     [WhiteElo "1446"]
7
8     1. e4 { [%eval 0.33] [%clk 0:10:00] } 1... e5 { [%eval 0.12] [%clk 0:10:00] }
9     2. Nf3 { [%eval 0.19] [%clk 0:09:58] } 2... d5 $2 { [%eval 1.23] [%clk 0:09:59] }
10    3. d4 $6 { [%eval 0.39] [%clk 0:09:47] } 3... dxe4 { [%eval 0.87] [%clk 0:09:58] }
```

The first six lines feature useful information about the game: a permanent URL to the game is shown in line 1 (which is very useful for bug-checking the pre-processed game) and the ratings of both players are shown in lines 2 and 6. Lines 3, 4, and 6 list the opening system featured in the game (i.e., a name given to the first several moves), how the game ended (e.g., checkmate, resignation, draw by agreement), and the time control (i.e., the time allotted to each player, along with additional time the player receives which each move that they play).

The final three lines in the file shown above document the actual moves played in the game. The move played by White is given in algebraic chess notation (e.g., "Re1" means "rook moved to square e1"), followed by the *engine evaluation*[1] of the position after the move was played and the time left on the clock after the move was played.

3 Baseline

A baseline for the task of guessing a player's rating based on a single game could be rooted in the accuracy of the average move that they played throughout that game. The accuracy of a single moved can be judged by its

[1]The engine evaluation is a quantitative score, evaluated by a chess computer, that assesses the advantage held by either player. The sign of the score indicates which player is winning, and the magnitude indicates the degree to which they are winning. The engine evaluation is given in the unit of *centipawns*, where one centipawn is equivalent to one one-hundreth of the value of one pawn. For example, if the engine evaluation is +1, then White holds an advantage equivalent to having one more pawn than Black.

centipawn loss, which is defined as the change in the engine's evaluation after a single move is played. If the best possible move is played then the engine evaluation should not change, since the best possible move was taken into account while computing the engine evaluation. Hence, "accurate" moves should have very low centipawn losses, while inaccurate moves will have high centipawn losses. For example, a player's Queen (the strongest chess piece and second in importance to only the King) is worth roughly 800 centipawns, so a move that gives away a player's Queen will carry a centipawn loss of approximately 800. Finally, the average accuracy of the game can be calculated by taking the average centipawn loss of every move.

Therefore, the baseline I am using for this project is a simple linear regression on average centipawn loss vs. rating across games. The intuition behind this baseline is simple: stronger players should, on average, play better moves than weaker players. Therefore, games played by stonger players should have lower average centipawn losses than weaker players, suggesting that the strength of a player could be crudely estimated based on their average centipawn loss in a single game. From computing this linear regression, the average centipawn loss in a game does indeed appear to be weakly correlated to the strength of the players. However, this correlation is weak; applying this "model" results in a root mean square error of 344.2, which is 93.5% of the dataset's standard deviation of 367.8.

4 Method

Conceptually, the problem of assigning a rating to a game of chess is conceptually similar to sentiment analysis. In sentiment analysis, we are given a sequence of words (where the length of the sequence is not fixed). Based on these words, attempt to assign a rating corresponding to positivity of negativity of the phrase being expressed. Likewise, in this particular problem, we are given a sequence of moves representing a game of chess and we are attempting to assign a rating to the strength of the players. Therefore, neural network architectures that are used in sentiment analysis appear especially promising for the problem at hand.

4.1 Representing Moves as Manually Embedded Vectors

Taking inspirating from natural language processing (NLP) techniques, I am representing individual chess moves as vectors, just as we represent individual words with embedding vectors in NLP. In this representation, the i^{th} move, $M^{[i]}$, is represented by the following vector:

$$M^{[i]} = \left[E^{[i]}, t^{[i]}, Chk^{[i]}, Cap^{[i]}, Cask^{[i]}, Casq^{[i]}, Pr^{[i]}, K^{[i]}, Q^{[i]}, R^{[i]}, B^{[i]}, N^{[i]}, P^{[i]}, c^{[i]}, r^{[i]}, Term, Res\right]$$

where $E^{[i]}$ is the engine evaluation after the move is played. $t^{[i]}$ is the length of time spent on the move. $Chk^{[i]}, Cap^{[i]}, Cask^{[i]}, Casq^{[i]}$, and $Pr^{[i]}$ are binary variables that describe the move itself, where the variable is given a variable of 1 if the feature is present and 0 otherwise. In order, these variables describe whether or not the move is a check, capture, kingside castle, queenside castle, and pawn promotion. $K^{[i]}, Q^{[i]}, R^{[i]}, B^{[i]}, N^{[i]}$, and $P^{[i]}$ are binary variables that are 1 if the piece moved was a king, queen, rook, bishop, knight, or pawn, respectively. $c^{[i]}$ and $r^{[i]}$ denote the file (column of the chess board, A-H, represented as 1-8) and file (row, 1-8) of the chess board. Finally, $Term$ and Res are quantites associated with the overall game outcome. $Term$ is 1 if the game ended by time forfeit (i.e., if one player lost because they ran out of time) and 0 otherwise, and $Res = 1$, -1, or 0 if the game was won by white, won by black, or tied, respectively. $Term$ and Res remain constant throughout any given game but of course will vary between games.

For example: consider a game where Black won as a result of White running out of time. In the tenth move of the game, Black spent 22 seconds thinking before moving their knight to A3, capturing one of White's pieces and putting White's king in check. The engine evaluation of the board after playing the move is -0.53. Then, the vector corresponding to the tenth move is:

$$M^{[10]} = \left[\underbrace{-0.53}_{E^{[10]}}, \overbrace{22}^{t^{[10]}}, \underbrace{1}_{Chk^{[10]}}, \overbrace{1}^{Cap^{[10]}}, \underbrace{0,0,}_{Cask^{[10]},Casq^{[10]}} \overbrace{0}^{Pr^{[10]}}, \underbrace{0,0,0,0,}_{K^{[10]},Q^{[10]},R^{[10]},B^{[10]}} \overbrace{1}^{N^{[10]}}, \underbrace{0}_{P^{[10]}}, \overbrace{1,3,}^{c^{[10]},r^{[10]}} \underbrace{1,-1}_{Term,Res}\right]$$

Then, the game matrix G is compiled by horizontally stacking all of the moves. That is, for a game with N moves, G is given by $G = \left[M_0^T, M_1^T ... M_N^T\right]$.

4.2 Neural Network Architecture

Because we have sequential data, and because of the similarities of the present problem and sentiment analysis, I am using a long short-term memory (LSTM) as the NN architecture for the present problem. The network architecture uses n_{lstm} LSTM layers with m_{lstm} hidden units, which are eventually fed into a series of n_d dense layers, each of which have m_d hidden units. Finally, the prediction for the average players' rating is computed using a dense layer with 1 unit using a tanh activation function (where data is normalized between -1 and 1 to accomodate the tanh function; I have also tried linear output layers and found the tanh layer to be slightly more effective).

4.3 Current Results

Initially, the lowest root mean square error (RMSE)I achieved with any architure was 258 (70% of the standard deviation of my dataset) using $n_{lstm} = 2$, $m_{lstm} = 64$, $n_d = 1$, and $m_d = 8$. . From a preliminary analysis, I noticed the NN was doing especially poor on games where the players' rating was less than 1300 or greater than 1900, prompting me to add additional games with these ranges of ratings to my dataset (leaving the test dataset unchanged). Doing so allowed the RMSE error to further decrease to is 211 (57% of the standard deviation of my dataset). Compared to the baseline error of 344.2, this already represents a significant improvement over my baseline. At the moment variation is quite low, with the training set having an RMSE of 187 (51% of the standard deviation of the dataset),

5 Future Work

I have not yet completed a proper hyperparameter sweep – hence, my RMSE may significantly decrease upon optimizing my NN's hyperparameters. Therefore, the next stage of my work will be to optimize my NN's hyperparameters. I will accomplish this by calculating the training and test error of a variety of NN architectures by training many NN's while randomly varying n_{lstm} (between 1 and 4), m_{lstm} (between 16 and 256), n_d (between 1 and 4), and m_d (between 4 and 64). I will also randomly vary the learning rate $\alpha = 10^{-}r$, where r is a randomly chosen number between 2 and 5.

After finding the hyperparameters that minimize the *training* error, I will aim to minimize the test error by enlarging my training set, focusing on game types where the model tends to fail (e.g., if the model underperforms in games where players' ratings are above 1900, I will add more of these games to the training set). If this does not help I will try reducing the minibatch size and adding dropout. Finally, I may attempt to add additional features to my vectorized representation of games in order to manually increase the features my model can be trained to recognize.

6 Novelty

The elements of novelety in the current work are as follows:

1. Problem choice: The problem of judging a players' rating at chess based on a single game they have played has not been studied in academic literature. Based on the rubric provided for the CS 230 project`https://edstem.org/us/courses/29493/discussion/1857921`, this alone should already be sufficient to receive +5 in the rubric for the "novelety" category.

2. Encoding chess moves as vectors "NLP style": To my knowledge, the concept of encoding a chess move as a vector, just as words as encoded as vectors in NLP problems, is itself a novel approach to any chess-related problem in NN's and beyond. Therefore, the present work provides a conceptually novel method of representing chess games such that they can be fed into NN's, which may be used in chess-related applications beyond the present problem.

7 Contributions

I am working on the project alone and hence have done all components of this project by myself. The only assistance I have received has been from CS 230 TA's during their office hours.