Using Deep Neural Networks To Predict Chess Player Skill

David Wang

Supervisor: Philipp Krähenbühl

A thesis submitted to
the University of Texas at Austin
for the Turing Scholars Program

University of Texas at Austin
Department of Computer Science
May 2020
1 Introduction

In this paper we attempt to use deep neural networks to predict the skill of the player for White in a game of chess measured by a chess rating system. We use models that rely on two sets of features, one based off of the positions on the board at the start of each turn, and another that views the evaluation of a player’s moves versus the alternatives by a chess engine.

2 Background

Since the 1950s computers have been used to analyze and study the game of chess. For the most part, research has been directed at the problem of how to win the game of chess. In recent years, this has resulted in computer chess programs consistently outperforming even the best human chess players. Chess engines produced for this performance have also allowed players to analyze their games by evaluating their moves with an engine. However, for the most part research has not been conducted into directly predicting the skill of players.

We feel that given the success of using computers to play chess, it should also be possible to also estimate player skill with a computer. Given the success of using deep neural networks to build strong chess engines, we believe that a deep neural network could also potentially be used to predict chess player skill. A common approach to using deep neural networks to build chess engines is to feed in images of positions along with whether the position resulted in a win or a loss in order to get an engine to learn to associate certain features with winning and losing. The network can then be used as a heuristic to evaluate moves within a search algorithm. We feel that it should be possible to generate a similar heuristic function that predicts the skill of a player instead of a win or a loss.

For the purposes of our model, the skill of a player will be measured using the rating system of the source of the games. In the case of the Lichess dataset, this is done with the Glicko-2 rating system [2] and for ChessDB this is primarily the ELO system[1]. Both systems measure player skill relative to that of other players so ratings are only consistent within the same pool of players. The difference is Glicko-2 attempts to factor in uncertainty about player ratings.

Supporting our hypothesis that deep neural networks can evaluate skill is research done by Regan and Haworth [5] suggests that players of different skill levels can be modeled using curves. These curves assign different probabilities of a player making moves of decreasing merit as measured by a chess engine. Their research suggests that ratings are strongly correlated to parameters for these curves that represent sensitivity and consistency when choosing potential moves. Here, sensitivity refers to the ability of players to distinguish between moves of similar merit and consistency referring to the ability to avoid choosing crippling bad moves.
Although this previous research was meant to make more realistic chess opponents or to aid in the detection of cheaters, we hypothesize that this also suggests a Deep Neural Network could estimate the skill level of a player using the evaluations of a player’s moves by a chess engine. We use an existing chess engine to evaluate the potential moves at the start of each turn and the move actually made by the player and provide this information to a deep neural network. We hope that our network can use then use the scores to determine approximately how sensitive and consistent a player is over the course of a game and thereby predict how skilled the player is.

3 Dataset

We used a subset of 20,052 chess games as our dataset [2]. Of these games, 16,155 games had Glicko-2 ratings for both Black and white players which we kept, and the remaining games were discarded. We split the rated games into a training, validation, and test set in a 60-20-20 ratio.

We aim to only predict the rating of the white player and we generally do not provide the Black’s rating to the model because the rating difference between the players tends to be small as most games in our dataset are between players of similar skill.

The White rating for our dataset ranges from 784 to 2700 with an approximately normal distribution. As a baseline to compare our approach against, we measure against the loss generated when we predict the average of our dataset. Predicting the average on the validation set yields an L1 loss of approximately 237 and an L2 loss of approximately 78074.

We also looked into a larger ChessDB dataset [1] that contained 3.5 million games. Unlike our Lichess dataset where all games were played fairly recently by a smaller group of players, the ChessDB dataset contains games from a much larger group of players over a much longer period of time. This could potentially lead to less consistent ratings so we elected to further restrict the dataset by only taking games since 1998 which the creators of the dataset claim is the largest and most consistent portion of the dataset. This left us with 1,950,000 games, which is still significantly more than the Lichess dataset. Whenever we felt that we might be overfitting due to our Lichess dataset containing insufficiently many games, we would create a larger dataset from this ChessDB dataset and see if that would yield better results.
4 Position Based Approach

Initially, we attempted to use an approach analogous to that of chess engines where we would attempt to train our model to learn to associate certain positional features as being associated with high or low level play. Normally for chess engines, only a single position is considered, but we choose to input the entire game in as a single tensor.

4.1 Input

We view every chess game in the dataset as a set of turns. For the position at the start of every turn, we convert into a tensor as follows:

1. The first 12 channels are generated using a one hot encoding of the different types of pieces. There are six distinct piece roles (Pawn, Bishop, Knight, Rook, Queen, King) and two possible colors (Black, White) so in total there are 12 possible combinations (ex. Black King). For each spatial location, at most only one of these channels has a value of one and every other channel must have a value of zero. If there is no piece in the location, then all 12 of these channels should be zero. Additionally, although for a turn at the start of the game there might also be a restriction that exactly two spatial locations with a value for the Black Knight channel for example, as every game starts with two Black Knights, as Black Knights are captured during the game, this may not be the case at the end of the game.

2. The 13th channel is filled with a single value depending on whether it is White or Black’s turn to move.

3. Channels 14-17 are used to indicate whether castling is allowed for White and Black, Kingside and Queenside.

4. Optionally, as an alternative to predicting absolute rating, we attempted to predict the difference in rating and provided black’s rating in channel 18.

This creates a tensor with dimensions 17x8x8 for every turn or 18x8x8 if we are providing Black’s rating. As our model has a sufficient receptive field to view the entire game, it could in theory infer channels 13-17 from channels 1-12. We initially did not include the latter channels but did so later after considering the importance of castling and the turn we opted to include them to ensure the model had access to the information. We considered appending on additional channels that indicate information pertaining to repetitions which are important for the purpose of avoiding draws but as including channels 14-17 did not lead to a large jump in performance, we felt repetitions were even less likely to produce a large benefit and opted not to include them. Partly supporting this judgement is the decision of some other chess neural networks to entirely
disregard games that ended in draws [4].

For every chess game in our dataset, we then considered a maximum of 64 turns to yield a tensor with dimensions 17x8x8x64 or 18x8x8x64 if Black rating is provided. We tested using a number of turns ranging from 16-128. We found that beyond 32 turns, adding more turns did not yield much improvement as long as increased convectional layers were used with more turns to ensure the receptive field of the result encompassed the entire initial board. We elected to use 64 as a step up from 32 to ensure that we were well in the area of having enough turns. If our game exceeded 64 turns in length we only take the first 64 turns into consideration. In the event that a game that has fewer than 64 turns available, we pad the start of the tensor with the tensor representing the position at the start of the game with the assumption that this should not be representative of either bad or good play. Approximately a third of the games in our Lichess dataset are over 64 turns in length and the rest of are shorter than 64 turns.

We tried using the last 64 turns instead of the first 64 turns as well but we found that this resulted in more overfitting. This resulted in a decrease in validation loss of 10-20 points while having no impact on training loss.

4.2 Model

For our model we use several strided factorized 3x3x3 convolutions to reduce the spatial resolution of our input to 1x1x1 while increasing the number of channels. We did not initially factorize our convolutions but we found that doing so decreased our parameters without affecting accuracy. We then run a variable number of 1x1x1 kernels on the result before using up convolutions to restore our original spatial dimensions. We also use skip connections between our down-convolutions to our up-convolutions so any of the kernels can be skipped. Following the convolutional layers, we feed our result into a linear layer which outputs the predicted rating of White.

We also tested an alternate model where we replaced the downstriding and upstriding and instead used several dilated convolutions instead. This alternate model also had sufficient receptive field to cover entire games but we found that this did not work as well as using striding.

The model is evaluated using L1 and L2 loss and for our optimizer, we tested SGD, Adamax, and Adam with a variety of hyperparameters. We found that SGD generally produced the best results.
Figure 1: Position Based Model
5 Single Turn Based Approach

After our position based approach did not yield significant results, we decided to continue using the positions on the chessboard as our input, but instead of viewing the entire game at once, we instead assign an rating for the position at the end of every turn and average to get the rating for the whole game. This is more closely analogous to how a neural network that plays chess functions. Historically, temporal convolutions do not work well, and we hope this approach will help. Additionally, because this approach no longer bounds us to fixed turn length, we hope this will allow us to take full advantage of any additional information longer games may contain.

5.1 Input

Like the model that takes in the entire game as input, we use convert the position at the start of every turn into a tensor as follows:

1. The first 12 channels are generated using a one hot encoding consisting of the six distinct pieces and two possible colors.
2. The 13th channel is filled with a single value depending on whether it is White or Black’s turn to move.
3. Channels 14-17 are used to indicate whether castling is allowed for White and Black, Kingside and Queenside.

This produces a tensor with dimensions of 17x8x8 for every turn in the game.

For training purposes, the label for each turn consists of the rating for the entire game. As a result, the same input tensor may have wildly different labels due to it occurring in the games of both good and bad players. For example, at the start of a game, due to the limited number of choices and the existence of
book openings, we would expect there to be minimal difference in the openings of players of a certain baseline skill. Similarly, during endgame, due to lack of pieces, we expect positions to be more likely to be common to both good and bad players game.

For the purpose of evaluating this model, we predict a rating for each turn and average across the entire game to get a prediction for the game overall.

5.2 Model

For our model we use strided 3x3 convolutions to reduce the spatial resolution of our input to 1x1 while increasing the number of channels. We then use up-convolutions to undo our strided convolutions and to allow skip connections to form between our input and outputs with the same dimensions. Following the convolutional layers, we feed our result into a linear layer which outputs the predicted rating of White.

The model is evaluated using L1 loss and for our optimizer, we tested SGD, Adamax, and Adam with a variety of hyperparameters like the position based model and continued to find SGD gave the best performance.
As an alternative to the position based approaches, we used a Chess Engine to evaluate all possible moves for White at each of White’s turns and compared the scores to that of the move White actually chose. As mentioned in Section 2, work by Regan has suggested that players can be modeled using a chess engine and parameters corresponding to how consistent and sensitive the player is. We hope that our model can recognize those parameters from a game played by the
player and from that infer the player’s skill level.

6.1 Input

For the evaluation of moves, we used the Stockfish 11 Chess Engine which is one of the most successful chess engines. Stockfish is an open source chess engine that uses alpha-beta search and hand tuned heuristics to evaluate moves. Stockfish itself does not use neural networks which we feel is important so that our neural network is not dependent on another neural network.[3].

The chess engine evaluates positions in terms of centipawns, which are meant to roughly correspond to an advantage of $\frac{1}{100}$ of a pawn. The evaluations are zero-sum in the sense that a position with a score of 1 centipawn from White’s perspective is evaluated as -1 centipawn from Black’s perspective. We set the value of a mate score at 100,000 centipawns which indicates that a position of checkmate is worth 100,000 centipawns. This value is large enough that any position that results in checkmate should be worth far in excess of any positions that does not. Additionally, A position that leads to checkmate in $x$ turns is evaluated as $100,000 - x$ centipawns which allows us to compare moves that result in mate to moves that do not.

We tried giving various amounts of processing power to evaluate each move ranging from 0.01 seconds to 0.5 seconds on a single thread of an Intel i7-8700K CPU. Towards the low end of the range, the scores given tended to be inconsistent enough to negatively impact training and at the high end it would take an exceedingly long time to generate our dataset. To determine a balance between these two concerns, we tried training on a small subset of our dataset to estimate what would be a good amount of evaluation time to use. We found that at .04 seconds gains from further increasing evaluation time were negligible without requiring inordinate time to generate the full dataset.

At each of White’s turns we evaluated all possible moves using the above settings and arranged the 16 best moves from greatest to least in the first 16 channels. The score of the move that White chose is then placed in channel 17. If there are fewer than 16 moves, we fill with the score of the move White chose so that relative to the move chosen, the alternatives are neither better nor worse and ideally our model will not penalize or reward White.

For each turn, we then standardize the 17 channels to follow a normal distribution. This is done as a player in a bad position may only have bad moves to choose from, resulting in a low scoring move being chosen even if it was the best move available. Alternatively, a player in a good position might choose a move inferior to the alternatives but still produce a positive score. We also tested normalizing the data to fit in the range [0, 1] for the same reason. We had concerns that in the event that one or more of the possible moves result in checkmate this could result in other moves being pushed to near zero values.
even if they still do not change the final result of the game. However, in training, we found that both standardizing the data and normalizing data yielded better performance.

We then take 32 turns for White to generate a 32x17 tensor which is our input into the model. In the event that we have fewer than 32 turns for White, we zero fill and we only take the first 32 turns for White if there are more.

6.2 Model

Unlike our positional model, for our final model we do not downstride for our convolutional layers. The assumption is that since our scores contain no actual information about position and are standardized, we assume that there is no need for the kernel for each turn to know about the preceding and succeeding turns. We tested downstriding but found that it generally decreased performance. This is in agreement with assumptions made by Regan's research where successive turns are assumed to be independent.

We tested applying a number of successive kernels that only look at a single spatial location to our input before feeding the result into a linear network that outputs our rating prediction. This worked relatively well, but we tested a simpler model consisting of only a single convolutional layer that outputs 64 channels and found that this attained similar performance. We tried removing this convolutional layer entirely and only using a linear classifier as well as changing the number of channels in our convolutional layer but found that a single 64 channel convolutional layer outperformed all alternatives.

Like the position based model, this model is evaluated using L1 and L2 loss and for our optimizer, we tested SGD, Adamax, and Adam with a variety of hyperparameters. We found that Adam and Adamax led to inconsistent convergence and that SGD produced much more consistent results if a lower learning rate and more epochs were used.
7 Results

Our various approaches yielded the following table of losses:

<table>
<thead>
<tr>
<th>Model</th>
<th>Loss for predicting White Rating</th>
<th>Loss for predicting difference in Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position Based Model</td>
<td>195</td>
<td>150</td>
</tr>
<tr>
<td>Single Turn Model</td>
<td>230</td>
<td>160</td>
</tr>
<tr>
<td>Engine Based Model</td>
<td>190</td>
<td>150</td>
</tr>
</tbody>
</table>

Table 1: Comparison of the results of using different models

7.1 Position Based Model Results

Our initial full game model yielded an L1 loss of 195 with the best set of hyperparameters we found. This is superior to just predicting the average which yields an L1 loss 237. Unfortunately, given that the overall range of our dataset is from 784-2700, this still means on average we can only narrow it down to about 1/5th of the overall rating range and encompasses players of significantly different skill. A difference of 400 ELO for example means the stronger player is expected to win 92 percent of the time.
With L2 loss, we obtained a loss on the validation set of approximately 69200 which similar to L1 loss is slightly better than predicting the average of the validation set but not significantly.

If we try predicting the difference in rating instead, at best we attained an L1 loss of 150. Predicting the average rating difference between Black and White would yield an L1 loss of 160 so we are unable to significantly outperform the average here either.

As an experiment, we also tested to see what moves this engine would evaluate as being that of a highly rated player, in essence, using our model as a chess engine. We found that our model was sometimes trained to prefer certain moves that are not typical of good play, such as favoring moving the king forward early on. This suggests our model may be overfitting to patterns that are not generally indicative of skilled play. As a test to see if increasing the dataset size would help with overfitting, we used 20,000 games from the larger ChessDB dataset to train a model and used another 10,000 games for a validation set. As the lichess.org dataset has a training set of about 10,000, this represents approximately a doubling in the size of our training set. As a test we tried to use 20,000 games of this dataset to train models and validated them on 10,000 games from this dataset, but this did not yield significant improvement in loss compared to training with the Lichess model. This leads us to believe the Lichess dataset to be sufficiently large, and the entirety of the ChessDB dataset was not used.

For L1 loss, we generated loss values for games of different length and White rating. Unsurprisingly, we found that the model tended to predict ratings closer to the middle of our range and as a result games with ratings near the middle of our rating range tended to have lower average loss than games at the extremes. Interestingly however, we found that for the most part the correlation between game length and the accuracy of our predictions was limited. Beyond very short games (fewer than ten turns long), the correlation between game length and average loss appeared fairly limited. This does however line up with our observation that generally speaking, increasing the number of turns in our model does not improve accuracy. This could be a result of games on average being too short either as the average chess game is well above 32.

7.2 Single Turn Model Results

After our positional model failed to work very well, we tested out the single turn model. This model did not perform very well, we only obtained a loss of 230 which is worse than 195 which was our best loss for the normal position based model. This loss is essentially the same as just predicting the average rating for every game.
Similarly, when we tried to predict difference in rating instead, at best we were only able to match predicting the average with an L1 loss of 160.

When this model was used to play chess like our position based model, we found that interestingly the behavior was similar to that of the normal position based model where the model would overfit to favor certain unusual moves like the position based model. For example, several models liked to move the king forward early for no real reason. Perhaps there are some highly rated games in the dataset where checks occurred early and players were forced to move their king up as a result. We believe that both of the models may be overfitting to these underlying trends.

7.3 Engine Based Model Results

Our engine based model performed similarly to the position based model, with an L1 loss of approximately 190 with the best hyperparameters. We continue to outperform guessing merely the average, but not significantly. We did find some benefit over the position based model as due to the reduced spatial complexity of this model, it trains and evaluates significantly faster at the cost of the time needed for the chess engine to evaluate moves. With L2 loss, we failed to outperform guessing the average and only generated a loss of approximately 10000.

When we tried to predict difference in rating, we obtained the same loss as the positional model, 150. This is interesting as since we normalize our scores, it should be unlikely that our model can derive any information about which player is better from the information provided to it. This result is most likely the result of overfitting which suggests that our loss obtained from our positional model may also be from overfitting as well.

When we used this model to play chess, we found that this model generally assigned high ratings to moves that were highly rated by the engine indicating that this model did pick up on the correlation between good players and good moves. Unfortunately, periodically the model would expect a good player to make an extremely poor move. We believe this is because, the model is unable to understand why even the highest rated players in the training set still occasionally make blunders and thus is overfitting by predicting certain blunders will be made.

When we generated a table correlating L1 loss to game length and White rating like we did for our positional model, we observed the same trends of White rating being positively correlated with loss but turn length being largely unrelated provided a minimum game length was met.
8 Conclusion

We found that we were unable to significantly outperform guessing the average in predicting player skill using deep neural networks.

For the position based model, this could be due to similar positions occurring in games with both good players and bad players. For models trained to win chess, a given position can be very strongly correlated with winning or losing but for our purposes, that may not be the case. A chess engine generally always seeks positions where they have more material than the opponent and better opportunities to attack but our objectives are more subtle. For us, we need to evaluate more along the lines of how well our player is moving in comparison to the opportunities granted them and the key features might be the differences between the positions at different turns rather than the positions themselves. Unfortunately, learning these differences seems to be more difficult than just learning to recognize more positional features.

The dismal performance of our single turn model reinforces the theory that player skill simply cannot be determined from positional features alone. Because our positional model only takes in single turns at a time, it clearly cannot learn differences between turns which may contribute to its poor performance. Additionally, as mentioned previously, it is likely that during the early and late game, positional features are likely to look very similar regardless of player skill level.

For our engine model, work by Regan et al. examined how to model existing players with curves and we presumed this potentially meant that if a neural network could learn the parameters for these curves from a single game then it could predict a player’s skill. In Regan’s work, curves were constructed with access to a much larger record of players’ past games and a single game may simply not provide sufficient information about an individual player’s ability for our network to infer from.

Future research this could potentially investigate this by training a network on games the same player played over a short period of time. However, even if it is shown that this works, such a network may not necessarily be useful as if the ratings of the opponents are known, a rating could be calculated conventionally from the results of those games.

References


