Chess Cheating Detection Using Deep Learning
Executive Summary

Calvin Pozderac, Philip Barron, Nathaniel Tamminga

Motivation

Cheating in chess through the use of advanced computer engines has recently become a widely discussed topic. This increased scrutiny has largely been spurred by Hans Niemann's surprise upset of then world champion Magnus Carlsen during the Sinquefield Cup in September 2022. Many notable players have expressed concern that cheating in chess is more widespread than previously suspected. The challenge of addressing cheating is heightened by the increase in online formats, such as Chess.com and Lichess, where imposing anti-cheating measures is more difficult than standard over the board (OTB) games. Therefore, creating a mechanism to detect cheating in online games is critical to maintaining the credibility of the game and ensuring fair play for all players. Below we discuss one possible avenue to achieving such a cheating detection algorithm using deep learning.

Methodology

In order to train a neural network to identify whether a player is cheating, it is tempting to inspect a player's win rate and see if it is consistent with their Elo or test to see how frequently they play the best moves. While these methods would catch some cheaters, they would also have a high rate of false positives due to various confounding variables. In the case of looking at win rates, effects including specific head to head match-ups, hot streaks, and tilt could all lead to streaks of wins that might seem statistically unlikely just based on their ranking. If we instead consider how frequently a player plays the best move (perhaps one that an engine would recommend), then we run into the issue that certain players could be in positions where the best move is very intuitive for a human player or they are in a well studied line such as a classic opening. Therefore, we need some method that takes into account these subtleties to avoid incorrectly labeling players as cheating (as that is potentially worse than not labeling a cheater as such).

Typically, when a player makes a move in chess, they are performing some calculation in their head based on different metrics (including value of pieces, king safety, positional advantages, etc) that humans have accrued over centuries of playing the game. When an engine is making a move, they are largely doing the same procedure except that their metrics (likely encoded in the weights of a deep neural network) are far more nuanced than the human brain could hope to achieve. Therefore, our goal is to train a neural network to distinguish between these two types of moves. In order to classify a move into one of these two categories, we need to collect a large dataset that includes both cases.

Datasets

Since there is no dataset of moves in which people have admitted to cheating, we need to be creative in order to develop our own. To do this, we make use of two different online databases for human and engine moves:

For engine moves, we leverage Lichess's online database of evaluations (https://database.lichess.org/#evals). This dataset includes over 66 million positions that have been analyzed by engine to a significant depth.

For human moves, we make use of an OTB dataset containing over 4 million games of classical GM level play, where cheating is likely less prevalent than online formats (ajedrezdata.com/databases/otb/over-the-board-database-aj-otb-000/).

In order to use this data for classification, we need to separate it into three categories:

0: moves that only a human would make

½: moves that both a human and engine would make

1: moves that only an engine would make

To perform this sort, we find all the positions that arise in both datasets. Then if a move is played in the OTB dataset and is not a top engine move, it gets a score of 0. If it is in the OTB dataset and is also a top engine move then it gets a score of ½. And if it isn't in the OTB dataset but is a top engine move then it gets a score of 1. This last category is the one that has the potential to identify moves that could be the result of cheating. After doing this procedure, we had approximately 5 million labeled moves to train on.

Deep Neural Network

In order to make use of this data, we needed to convert it into a tensor form that a neural network could use. To do this, we created a (20,8,8) input tensor for each move. The first 12 channels indicated the position of the 12 different pieces on the 8x8 board. Channel 13 indicated whose turn it is to move. Channels 14-17 correspond to the castling rights of the players and channel 18 to any possible en passant moves. Finally, channels 19 and 20 highlight the start and end position of the moved piece. With this set-up we have a way to convert our chess move data into an input tensor for our neural network.

For the neural network architecture, we made use of the insights of Google Deepmind's AlphaZero. This architecture is a convolutional neural network with residual connections that was used to train a neural network to achieve superhuman play in chess. The model uses ReLU activation functions and batch normalization. After the convolutional layers, the results are fed through a fully connected layer to output a final score for the move. Finally we use a binary cross entropy loss with stochastic gradient descent and weight decay to train the model.

Training Results

When we started training the neural network on a Google Colab T4 GPU, we ran into the following complications. For small subsets of our training data, the model was able to quickly lower the training error to nearly zero, but generalization to our test set was never achieved. If we instead increased the amount of training data used, then the training error decreased but much more slowly. Potentially, with more time or more

compute, we would expect the training error to eventually get small followed by generalization to the test data.

Extensions

If the model had successfully learned to classify between human-only and engine-only moves we would have then implemented that classifier to identify when a player is likely cheating. We would have used Lichess's online database (https://database.lichess.org/#standard_games) to test how common cheating is on their online platform. To do so, we would have looked at each game and if a certain number of moves flagged as engine-only (depending on the accuracy of the move identifier) we would have identified that game as cheating.

Conclusions

Overall, we have developed a deep neural network framework that would in theory allow for the identification of cheating in chess. We collected large datasets of both human and engine moves and cleaned it into a usable format for training. We then set up a convolutional neural network with binary cross entropy loss. While the training space of chess moves ended up being too complicated for the amount of training time and compute we had access to, we are confident that given more training, it would effectively generalize to a wide range of moves. Ultimately this could be used to determine whether or not the current chess cheating concern is justified.