MAIS deliverable 1 – Data selection proposal

Aurélien Bück-Kaeffer

1: Dataset

The dataset we would like to work with are games of chess in the PGN format. We can either generate it while training the model, download Grandmaster games from the many existing databases, or both. It is already made to be computer processed, so easy to work with and there exists libraries to manipulate them already, allowing us to focus on the model.

2: Methodology

As stated above, if going for a supervised approach, the dataset is easily obtained with information in large quantities. It even has the advantage of containing a label already, that being, the result of the game. While some preprocessing could be interesting to remove positions that appear multiple times, it isn't necessarily a good thing to do so since it means those are positions that require particular focus. Therefore, little to no preprocessing is required. A problem does arise with the fact that even if a Grandmaster lost a game, they most likely still played extremely well, making it not necessarily wise to deem the moves they played to reach such result "bad". We also run into the issue that most grandmaster's games are not played until checkmate, but rather until both players agree to the result (We could parse the data to only keep those that do finish in a checkmate, but this only represents about 1.1% of all games (figure from a dataset of 36 000 games we parsed ourself), and these are not necessarily representative of the dataset as a whole), which means our model would not learn to checkmate, only to get an edge on its opponent.

What is most likely, however, is that we will be going for a reinforcement learning approach, meaning no previous dataset would be needed, and it would instead be generated as the training progresses with the games the AI would play against itself.

The goal of the model would be to predict what is the best move in any chess position, and consequently be able to reliably checkmate its opponents.

While it would maybe be possible to make a simple binary classifier considering every move and deciding whether they are "good" or "bad", it is most likely not a very good approach since you can have positions that have multiple good moves, one being clearly better than the others. We want to be able to quantify how good a move is to rank them and pick the best one. This then becomes a regression problem. This, however, would require that it is possible to model how good a position is for each player given a board with a function, which we have no reason to believe is the case. It would also imply that two similar positions should get a similar score, which is a very wrong assumption, as the position of a single pawn on the board can determine if a player is about to get checkmated in one move or if they are completely winning.

As reinforcement learning is used to learn how to act based on a dynamic environment that will in turn be itself affected by said actions to maximize a cumulative reward over a sequence of actions, it simply feels like the perfect fit for the current problem. The main issue with that being, it will not be covered in the lectures, and requires heavy computing power and/or lots of time to train, probably more than other solutions.

In conclusion, the approach we would like to have would be one similar to that of AlphaZero or Leela: Use a Monte Carlos tree search combined with a reinforcement learning model telling the algorithm on which parts of the tree it should focus its search for good moves.

It is well known that chess can be seen as a tree of possible positions that can be reached, which each node being a board and each edge a move. The main issue is that there are on average 35 legal moves in any given position, meaning a naïve algorithm that would go through all the possibilities n moves deep to find the best one would be about $O(35^n)$. Needless to say, O(no). This is where the Monte Carlos tree search comes into play, using the model to say which part of the trees are interesting to focus on and immediately disregarding a majority of the moves for greatly improved performance. If, say, we only consider the 5 best moves into each position, then we need to consider $35 * 5^n$ moves on average. $O(5^n)$ is already a massive improvement and allows us to consider a lot more moves in advance. Sometimes, however, some moves are "technically" forced, meaning that while other moves are legal, not making an obvious move would instantly give a critical advantage to one of the players, so only that one move needs to be considered in that variation, further reducing the complexity of our tree search. The point of the reinforcement learning model would be to learn to dynamically determine how many moves deserve to be considered at face value for the tree search algorithm to dive deeper into them.

While the ideal thing would be to only award points to the AI when it checkmates its opponent, meaning we would not be introducing bias with human strategies, this would most likely be extremely inefficient as it is particularly rare so stumble upon a checkmate at random, making it so that the AI would simply play randomly without learning anything for the entirety of the training. It is therefore interesting to include new elements in the Q-function, such as those stated in the chess programming wiki "Point value by regression analysis" page : "Calculating the static evaluation score is usually implemented by a linear combination of various position features scaled by some weights - most importantly the number of pieces and pawns of both sides, further the position of these pieces, centralization, and mobility."

The metric we will be using to characterize a model's performance will be the performance rating as a way to estimate Elo on a reduced number of games. A baseline result we would hope to beat would be a performance rating of 1000 on chess.com upon, say, 10 games played against either real people or other rated AIs. Considering an unrated beginner starts at 800 Elo, this would mean the model should be able to reliably beat beginners.

3: Application

The application format comes very naturally: A website presented in a way similar to chess.com or lichess.org, with a chessboard allowing the user to play against the model.