Predicting Moves in Chess using Convolutional Neural Networks

Barak Oshri
Stanford University
boshri@stanford.edu

Nishith Khandwala
Stanford University
nishith@stanford.edu

1. Introduction

Convolutional neural networks have been shown to be successful in various longstanding AI challenges that can be reduced to classification problems. Clark and Storkey have reported a 44.4% accuracy in predicting professional moves in Go, a game known for its abstract logical reasoning that experts often describe as being motivated by faithful intuition. This is an exciting result, indicating that CNNs trained with appropriate architectures and a valid dataset can catch up with much of the experience-based human reasoning in complex logical tasks.

The success of CNN-Go can be attributed to smooth arrangements of positions that are approximately continuous through and between games. Additionally, since each move in Go adds a single piece to the board, essentially flipping the value of one pixel, the difference in board representations before and after a move is smooth, constant, and almost always linked to the important patterns observed by the network, which contributes to the consistency of Go classification algorithms.

1.1. Challenges of CNN Approaches to Chess

Unlike Go, chess is more motivated by heuristics of many kinds of pieces in diverse and short-term tactics that build into longer-term strategies. This is essentially because the advantage of a position is always rooted in the relationships between the rules of the pieces. This makes pattern identification of chess more reliant on understanding how the nuanced and specific positioning of pieces leads to their advantages.

For these reasons, it is less clear that the logical patterns in chess can be described in activation layers of a neural network. Important concepts such as defending or pawn chains are often times best expressed by heuristic methods and logic information systems, such as "if pawn diagonally behind piece" or "if bishop on central diagonal" conditionals. Critics of CNNs argue that neural networks cannot adequately explain such tactical advantages because the forms of these conditions are too global across the board and effected by extraneous variables.

1.2. Incorporating Human Reasoning into Pattern Recognition

These arguments, however, are only correct in the domain of learning algorithms labelled on loss, draw, and win positions, where the network acts as an explicit evaluation function of the network.

Modern approaches to chess intelligences are comprised of two parts: an evaluation function and a search function. The evaluation function scores a board in a relative assessment of how likely it is to lead to a win, and the search function is a lookahead implementing minimax using the evaluation function. Since chess is a finite state hence solvable game, this approach is first limited by computational needs and second by the success of the evaluation function. Leaps in chess AIs therefore improve on either of these limitations, by cleverly navigating the search space or incorporating chess principles into the board evaluation.

It is thus not surprising that machine learning approaches to chess capitalized on the challenge of producing a successful evaluation function by attempting pattern recognition on data points labeled with a 1 if the move was made by the winning side and a 0 if the move was made by the losing side. Although such approaches are principally correct, they do not reflect the nature of chess games played by above-average players: rarely is a move characteristically winning or losing. A move is rather made in the spirit of achieving some local objective.

When deep neural networks were not trained against achieving a local objective but rather the final binary state of the game, the algorithm attempts to labor at developing an oracle intuition for whether a representation corresponds to a winning or losing state. It is thus very clear why developing an intuition for short-term tactics and conditionals did not succeed.

For this reason, we eschewed the one-sided labeling of chess boards and modelled incremental tactical choices by

labeling each board state with the move made from it. This philosophy better captures the nature of chess moves in an experienced chess game: almost every move played by a high-ELO chess game is a reasonable move, especially when averaged over the entire training set. In this approach, the patterns that matter in the raw image are those that encourage human actors to make certain moves: the burden of evaluation is thus relayed to the stronger labels and consequently a larger class space.

Also interesting to note is that classifying for the next best move acts as a precomputation of the lookahead involved in the search function, as the needs for the search are now met with an understanding of which move was played for a given board representation.

1.3. Approach

The greatest challenge to this approach to training is that the space of possible moves is unwieldy large. The class space for the next move in Go is always some subset of $19 \times 19 = 361$ possible positions; but in chess, although there are generally an average of fifty possible moves given any position, there are $(8 \times 8)^2 = 4096$ possible classes which a CNN would have to score for.

For this reason, we divided the classification challenge into two parts using a novel approach (a very novel approach...). The first part is training a CNN to predict which coordinate a piece needs to be moved *out* of. This captures the notion of escape when a piece is under attack or the king needs to move. The network takes as input a board representation and then outputs a probability distribution over the grid for how desirable it is for a piece to leave a square, with all squares without pieces or with opponent pieces clipped to 0. The second part is training *six* other CNNs to encode which coordinates would be advantageous to put each of the six possible piece on. For example, this includes a bishop neural network that takes as input a chess board and outputs a probability distribution on the whole grid for how desirable it is to have a bishop in each square, with all squares that a bishop cannot move to clipped to 0.

We then compose the out-CNN (oCNN) with all in-CNNs (iCNNs) by multiplying the values in the oCNN by the highest value in the corresponding iCNN and taking the argmax over the entire composition to obtain two coordinates for the leave position and the put position.

Note that each CNN now only has a class size of 64 (a square root of the original absurdity!) for a computational cost of doubling the training time. Interestingly, though, this approach captures much of the human intuition behind chess thinking: the logic and decision behind when to move a piece out of a square and into a square is often evaluated and made independently.

2. Problem Statement

This project will learn move predictions from ten thousand Free Internet Chess Server (FICS) chess games recorded online from players with an ELO rating of greater than 2000. At this time, it would be early to predict what accuracy of results to expect from this dataset as it is not known whether a CNN can fully capture the basic reasoning used in chess games. Since each turn of chess usually has about fifty possible moves, we expect an accuracy at minimum ten times better than random, which is a useful baseline for this project. As an upper bound, any chess algorithm that makes correct move predictions 50% of the time is deemed to understand basic chess logic. We may also pit the CNN on GNU-chess and record the percentage of games won.

3. Technical Approach

3.1. Data structures

A chess board is represented in our model as an $8 \times 8 \times 6$ image, with two dimensions covering the chess board and six channels corresponding to each possible piece; a different representation is used for each piece since its value is of discrete worth - the difference between two pieces is not continuous and so cannot be measured on the same channel.

Although it does not matter which color the algorithm is training from, we must ensure that the side of the board the algorithm is training from is the same. For this reason, when the algorithm trains from black we reflect the board vertically and then horizontally (including the label associating with this board) to preserve the orientation that white plays from even when encoding black's move. We use +1 to denote friendly pieces, and -1 to denote opponent pieces.

3.2. CNN architecture

All seven networks take as input the chess representation described above and output an 8×8 probability distribution. We use a five layer convolutional neural network of the form [conv-relu]x2-pool-conv-relu-affine-relu-affine-softmax. We emphasize the need for two affine layers so that the low level features can be accompanied by stronger global logic evaluations.

3.3. Preprocessing

The FICS dataset is comprised of chess games in PGN format, which enlists every move played by both players in Algebraic Chess Notation (ACN). In order for this dataset to be relevant to our project, the ACN notation had to be converted to two coordinates representing the position a piece was moved from and the position a piece was moved to. These moves were then played out on a chess board to obtain the board representation at each position, encoded in the data structure described above. The two sets of labels used are then the coordinate a piece was moved out from and the coordinate the piece was moved into.

3.4. Training

Using chessboard representations of size $8 \times 8 \times 6$, we train the oCNN and iCNNs on each move in the game. At each move, we only train one of the iCNNs. The iCNN to be trained is determined by the piece moved in the previous step.

From an implementation perspective, we initialized the weights randomly using the fan-in and fan-out approach. This would ensure that the symmetry is broken and the weights actually get updated to reflect the training of the model. We do not use regularization as the concept of "smoothing" the activation of the inputs is not applicable for the chess image and could break much of the chess logic nuances.

We flipped the chess board vertically and trained on these images too. We measure the accuracy independently for both the oCNN and iCNN.

3.5. Testing

In testing, we run the input on the oCNN and iCNNs to produce the probability distributions for each neural network. For each piece of the oCNN we then reference on the associated probability distribution from the iCNN and clip to 0 all the squares that are invalid moves using a chess logic algorithm. We multiply the maximum value of this by the value from the oCNN and record it and the associated in and out coordinates. We then choose the in and out coordinates with the maximum probability and output that as the move.

4. Preliminary Results

As of today, we have been successful at preprocessing chess game datsets in PGN format into a representation that suits our model requirements. In the process of doing so, we have made use of python modules like pgn-parser and python-chess, which along with inspiration from Erik Bern's work, have been tremendously useful in getting us set up for the project. We now plan to use Theano, a Deep Learning library, to run our CNN model. We expect to achieve statistically significant results that justify the use of CNNs and further the application of deep learning to solve similar strategy games. Competitive results will also serve to validate our proposed algorithm which would in turn encourage such approaches to game AI.

It is important to note that we do not incorporate information about wins or losses in our current preprocessing step. We reason that the result of a chess game is not an outcome of a single move by a player, but a constant series of contemplated plays. Also, the dataset we obtained was restricted to world-class chess players, ensuring that each recorded move was, in most cases, a thoughtful ideal move. Hence, it would be more coherent with our philosophy if the final outcome of the game was ignored and every move was given equal weight.

However, in the scenario that our model fails to achieve satisfactory results, we would then experiment with labeling moves with a "potential win" and a "potential loss" tag. This would give our model some sense of direction, thereby increasing the accuracy rate. But, for the reasons illustrated in the previous paragraph, we would prefer to adhere to our initial philosophy.

References

1. Christopher Clark and Amos J. Storkey, "Teaching Deep Convolutional Neural Networks to Play Go", 2014
2. Erik Bernhardsson, "Deep Learning for... Chess", 2014