An Analysis of Machine Learning for Chess

Ori Yonay
Department of Computer Science
Texas A&M University
College Station, TX, USA
oyonay12@tamu.edu

***Abstract*—Chess has always been at the heart of artificial intelligence; with more possible positions than atoms in the known universe, it is quite a challenge to solve. Early computer scientists (most notably Alan Turing in 1948) made attempts at writing computer programs that played chess [1], but computers were not advanced enough to run such programs. In 1997, IBM's Deep Blue computer was the first computer program to beat the reigning world chess champion [1]. While such programs are increasingly powerful, they are not 'intelligent'; they are extremely powerful calculators that are still dependent on human knowledge and guidance. In recent years, advances in machine learning have made it possible for computers to master chess without human knowledge. We will investigate and analyze the different machine learning methods for chess (specifically those employed by LeelaChess and DeepMind's AlphaZero) and why they are as effective as they are.**

***Index Terms*—chess, machine learning, artificial intelligence, reinforcement learning**

I. Introduction

Computer scientists have been trying to create chess programs for the better part of the past century. For many decades, chess has been (and is still) considered a benchmark of artificial intelligence and its progress, with more and more advanced algorithms invented and employed over the years. For the past half-century, computer chess primarily took advantage of computers' ability to quickly perform a large number of calculations, essentially allowing chess engines to brute-force their way to superior play over humans. Such methods are 'unintelligent' and do not require much more than hyper-optimized search algorithms and some human knowledge of chess. In recent years, with the rapid advancement of machine learning, Google's DeepMind created AlphaZero, the world's first chess algorithm that learned to master chess (as well as other board games; this generalization of their algorithm is a landmark achievement of its own) completely from self-play, having been given no knowledge about the game other than its rules. Perhaps even more remarkably, this approach allowed AlphaZero to beat Stockfish (one of the strongest chess engines available) in a 100-game match in 2017 [2]. Leela Chess Zero, aimed to be an open-source replication of AlphaZero, has managed to achieve similar results through crowdsourced training of their neural network. At the time of writing, LeelaChess has played over 300 million games against itself and plays at a level comparable to Stockfish [1].

II. AlphaZero vs. LeelaChess

A. Neural Network Structure

While AlphaZero's paper does not explicitly state which neural network the engine actually uses, it does state that the DeepMind team tested (and probably used) either a residual neural network or a convolutional neural network [3]. LeelaChess takes a similar approach using a *residual tower* with *squeeze and excitation* layers [4], as well as an input convolutional layer used to reduce the size and complexity of the input.

AlphaGo (AlphaZero's predecessor that focused on mastering the game of Go)'s neural network constisted of 40 residual layers [5], and AlphaZero's approach is likely similar in design. During the creation of AlphaZero, the development team tested a few similar network architectures but likely stuck with this 40-layer residual network design as it performed best against other versions of itself using different network architectures.

B. Input Data

Both AlphaZero and LeelaChess use the same input format for their networks; they convert the board position into a 112x8x8-bit tensor. For each piece type (6 for each side), an 8x8 bit board is created, with a 1 in every position where a piece of the given type is present and a 0 otherwise [6]. As an implementation detail, bits such as the player's turn are fed as an extra 8x8 layer containing all zeros or all ones [5]. The input also contains the past seven board positions, which gives the neural network context regarding previous moves.

C. Learning Procedure

While both AlphaZero and LeelaChess use reinforcement learning to train their neural networks, they go about doing so in different ways. Both engines use the *Monte Carlo Tree Search* ($MCTS$) algorithm to generate board positions for self-play [5] [6]. There are a few advantages to using MCTS for reinforcement learning tasks (and specifically for problems such as board games). Unlike other tree-search algorithms (such as Minimax), MCTS is not as affected by the high branching factor of games such as chess and go due to the fact that it randomly samples the search space (as a heuristic that, counterintuitively, helps significantly) as well as its focus on the most promising moves. MCTS is also more intelligent about the traversal of the game 'tree', pursuing the most promising moves first and constantly

updating its knowledge according to the results of its random simulations [1].

Unlike AlphaZero, LeelaChess trains its neural networks through crowdsourced self-play rather than self-play on a single machine, mainly due to the lack of resources compared to AlphaZero's team. In order to learn the game of chess from scratch and surpass human knowledge of the game, both programs use reinforcement learning to train and update their knowledge. The advantage of using reinforcement learning in these cases is that no information beyond the rules of the game is necessary for training; the model dynamically learns positions and move favorability on its own, which allows it to transcend human knowledge of the game.

D. Analysis

Both engines use the same basic structure to train and improve themselves. Reinforcement learning is far more intelligent and human-like in style than traditional brute-force methods, and allows for dynamic adjustment of the models rather than the more static, linear approach traditionally employed by classic engines such as Stockfish. This, in combination with MCTS, has the best of both worlds; human-like intuition and intelligence combined with the raw computational power of a machine.

Interestingly, both types of networks (convolutional and residual) perform exceedingly well on image recognition [7]. This is no coincidence; in a way, chess positions and images are quite similar. Both contain local details that impact the structure as a whole; finding features in local parts of an image is very similar to evaluating a chess position in parts, considering small sub-areas of the board in order to find recognizable patterns. Remarkably, not only does this convolutional approach dramatically reduce the dimensionality and complexity of such a function, but it is also very similar to the way humans perceive both images and chess positions. It is exactly this approach that gives AlphaZero and LeelaChess their advanced positional perception.

In order to ensure game diversity, AlphaZero plays a random move (within 1% of the evaluation of its perceived best move) during the first 15 moves of the game. This turned out to improve AlphaZero's win rate and is a feature that is currently implemented in LeelaChess as well [6].

One of DeepMind's original innovations in the field of such game-playing algorithms is the board state. As previously mentioned, both engines provide their network the given position along with the past seven board positions for context. According to the DeepMind team, this acts as an attention mechanism that grants the network a deeper understanding of the current board state. This also simplifies the input, as rules like en-passant are implied via the board history, and remarkably the engines are intelligent enough to deduce this.

Perhaps the biggest dillema in the realm of reinforcement learning is the explore-exploit tradeoff, in which the agent (the chess algorithm, in this case) must choose between exploiting its current knowledge of the game (and making the best move accordingly) or further exploring the game space with the chance of discovering an even better move, learning more about the game, and improving its level of play. This is tackled by both LeelaChess and AlphaZero by generating new networks and testing them against each other (with the hopes that some networks will by chance outperform the current network) as well as through manual fine-tuning of hyperparameters responsible for making such decisions (primarily within the MCTS algorithm).

III. Supervised Learning for Chess

As a culmination of my research regarding machine learning for chess, I have designed my own machine learning model. My approach was to combine classic chess-engine methods (such as minimax with alpha-beta pruning) with a machine learning approach for evaluating static positions (leaf nodes). To train the model, I used a dataset containing 13 million positions and their respective engine evaluations using Stockfish (one of the best chess engines at the time of writing) with a 22-move lookahead. The evaluations are in $centipawns$, a standard system for chess engines to numerically evaluate a position. Chess is traditionally approached by programmers as a $tree$, whereas my project will approach it more as a very complex $function$ that the neural network will be attempting to approximate.

A. Data Preprocessing

The dataset was provided as a list of chess positions as FEN String (a standard, compact way to store a chess position) as well as its respective Stockfish evaluation. Each position was converted into a 768-bit input matrix as follows: for each piece (of which there are 6 types for each side), a 64-bit (8x8) matrix was created, with a 1 in every position where a piece of the given type exists and a 0 otherwise. This gives a three-dimensional (12x8x8) input tensor. This conversion was done largely with the help of the $python-chess$ library.

For the sake of simplicity, I did not provide any information more than the board itself (i.e., castling rights, en passant, or whose turn it is). This is to be contrasted with AlphaZero and Leela's 112x8x8 input tensor, which also provides a move history of up to seven moves previous (in which no en passant bits are necessary; en passant is implied via move history).

As 13-million chess positions would take approximately 20GB of memory to store and a long time to access, positions are imported as FENs and converted to bitboards in batches. This is common practice in machine learning when the training data is too large to store at once.

B. Neural Network Architecture

The network is a convolutional neural network consisting of two convolutional layers (with a kernel size of 3, like both AlphaZero and LeelaChess) and three fully-connected layers with a decreasing number of neurons. Neurons are activated using ReLU after each layer of the network. The output consists of a single neuron, representing the network's evaluation of the given position. This is contrasted by AlphaZero's output of moves along with their respective win probabilities, which

is far more flexible and allows for move randomization (which empirically improves AlphaZero's win rate).

The network was trained for a few days, using a learning rate of 0.0001, on one million positions at a time.

C. Results

The 'engine' was manually matched against five other engines tournament-style (with a depth lookahead of only 3 due to speed issues) for one hundred games. It performed at around the level of a beginner (950-1150 ELO) and rarely picked the best move (as evaluated by Stockfish). When a material count was added to the evaluation, the engine performed significantly better, reaching around 1250-1350 ELO. (While this defeats the purpose of machine learning, it can be thought of as a heuristic to aid the engine as it did not have much time to train. It is also worth mentioning that Stockfish's latest version now implements a similar model, where the evaluation process is shared between a neural network and a more classical manual approach).

D. Limitations and Future Directions

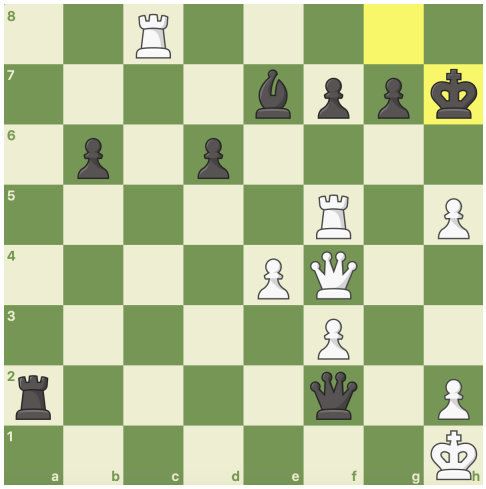
The main limitation of this project was the short timeframe for the implementation and training of this model. Machine learning models, especially for learning a 'function' as complex and intricate as chess, take a long time to train. The network has probably not reached anywhere near its play potential due to the lack of training done on it, but with enough time could probably reach about a 1600-1800 ELO level. The lack of time also did not allow me to explore more complex options like reinforcement training using self-play (which is the main feature of AlphaZero and LeelaChess).

Another limitation to the project is the fact that unlike AlphaZero and LeelaChess, it model was trained using supervised learning. This leads to a few shortcomings in the results: firstly, since evaluations are done by Stockfish, the network is trying to approximate a human's evaluation of the position, which in part defeats the purpose of machine learning. Furthermore, since the network uses positions that are already known rather than dynamically generating its own (like AlphaZero uses MCTS), the range of positions the engine trained over was significantly limited.

Since the engine was not UCI-compatible (due to a lack of time), it was difficult to pair it against other engines automatically, and games were played manually.

The future directions of this project are, in rough order of magnitude, to convert the project from a supervised to an unsupervised approach (as this would ameliorate many of the limitations detailed above), implement UCI-compatibility to allow the engine to play other engines (the benefits of which are twofold, as this would enable the engine to continually learn from other engines as well). Additionally, I would like to change the neural network architecture to one resembling AlphaZero's as it has proven to be extremely effective. From there, moving training to the GPU and creating a runnable application would be the logical next steps, and the ones necessary to create an 'official' chess engine that can play against other engines in tournaments.

My current 'engine' also does not provide the network with the board state in its entirety; important features such as castling rights, en-passant square, and whose turn it is are omitted for simplicity and convenience. Such features could potentially be crucial to a position's evaluation (which would take a neural network a long time to learn - hence another reason such features were omitted, as the project timeline consisted of only a few weeks). To illustrate, consider this final board position from the fourth tiebreaker game from the 2016 world chess championship (Magnus Carlsen [white] vs. Sergey Karjakin [black]):



In this position, it is white's turn to move, and the winning move Carlsen played was $Qh6+$, after which Karkakin resigned. Given the same position, had it been black's turn, though, a checkmate-in-one ($Qg2\#$) would have been possible, completely flipping the evaluation as a result of a single bit-flip in the input.

One of the most effective ways to improve a chess engine's play is to look deeper into the game; it is generally favorable to prune out as many moves as possible and look further ahead at more critical, likely-played positions [8]. This is the reason Stockfish plays much better than many commercial engines - even though they tend to have more accurate evaluations and are very highly optimized, Stockfish manages to look one or two moves further due to intelligent pruning. This is also the reason why human grandmasters are able to play at a similar level as some weaker engines; they only spend time analyzing the most critical, likely moves rather than approaching the game like a gigantic, highly-optimized calculator. AlphaZero goes a step further in the human direction, analyzing only about 80,000 (compared to Stockfish's 70 million) positions per second. The reason AlphaZero still manages to outperform Stockfish is the positions it chooses to analyze.

Similarly, a potential future direction of this project would be to optimize the move generation procedure (the project currently employs a standard function in the $python-chess$ library). This would likely be done by converting the project to c++ and rewriting this part from scratch using advanced

computer-chess concepts such as $magic\ bitboards$ and hashing. Since the $PyTorch$ library (which was used to create and train the neural network) is also available in c++, the same neural network could be used.

Another interesting idea within the realm of move-pruning which I have not yet seen is using a neural network to decide which moves to prune (rather than using it to evaluate static positions, as in the case of this project). Taking inspiration from the $principal\ variation\ search$ algorithm, such an engine would use a neural network to intelligently sort possible moves by order of their relevance. This would not only allow the engine to look further ahead (as it would be evaluating fewer positions; the depth-for-breadth tradeoff is extremely beneficial) but would also help prune out many moves that would have been considered otherwise (this is the main advantage of PVS; considering the likely-best moves first allows for alpha-beta cutoff of more branches). It is also worth noting that such a tactic is very similar to the way humans approach such games; more intelligent, 'human-like' approaches of chess engines have been shown to be more effective (AlphaZero is known for its human-like style of play).

IV. Conclusions

To conclude, even though chess is an extremely complex game, it can be played at superhuman levels using machine learning. Such intelligent, human-like learn and play methods far outweigh the fast, conventional, 'unintelligent' search methods of traditional chess engines, which primarily took advantage of the fact that computers are able to brute-force their way to victory. Chess commentators and grandmasters alike frequently refer to AlphaZero and Leela's moves as 'creative', 'intuitive', and 'human-like', and rightfully so; these engines are built to imitate human intuition and play. By considering only the most relevant, likely moves, AlphaZero and LeelaChess are able to spend their resources effectively. The advantage of using a convolutional neural network (or a network that consists of convolutional layers) is that it helps find positional advantages and highly simplifies the complexity of a position to the computer, allowing for quicker and more accurate training. This also mirrors the way in which humans analyze the board; much like analyzing a complex image, a convolutional neural network is able to reduce the board to its relevant features and achieve a more accurate evaluation of the board.

There is much more to be discovered in the field of machine learning for chess. Counterintuitively, a more hands-free approach to learning chess turns out to outperform hard-coded human knowledge of the game.

Acknowledgment

Special thanks to Dr. Bobak Mortazavi for the support, guidance, and help in the design and implementation of this project.

References

[1] "Deep blue versus garry kasparov," Nov 2020.
[2] Mike Klein, "Google's alphazero destroys stockfish in 100-game match," Dec 2017.
[3] "Network architecture of alphazero," Dec 2017.
[4] Jie Hu, Li Shen, and Gang Sun, "Squeeze-and-excitation networks," *CoRR*, vol. abs/1709.01507, 2017.
[5] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *CoRR*, vol. abs/1712.01815, 2017.
[6] "Leelachess zero," 2018.
[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.
[8] Colin Frayn, "Computer chess programming theory," 2005.