Implementation of an AI Chess Engine

Raghavesh Viswanath, Dheeraj Bhurewar

ABSTRACT: Chess is not only one of the oldest and most popular board games in the world, but it is also a breeding ground for complex machine-learning algorithms. Recently, researchers at deep mind developed AlphaZero which starting from random play, and given no domain knowledge except the game rules, achieved within 24 hours a superhuman level of play in the games of Chess and Shogi (Japanese chess) as well as Go, and convincingly defeated a world-champion program in each case. Since it was mostly trained on supercomputers, we will look at a way of using the CCRL data set to train the neural network instead of self-play as a way to replicate it on consumer hardware.

KEYWORDS: AlphaZero, Reinforcement Learning, Supervised Learning, MCTS, Chess

1. Introduction

The main goal of the project is to create a chess AI that is at least capable of holding its own against amateur players while being trained on consumer hardware using the AlphaZero framework. Achieving such performance on consumer-level hardware using just self-play as done by Deepmind would require an extensive amount of training for many days. In fact, it took Google 10 hours of training on 4 first-generation TPUs and 44 CPU cores to achieve superhuman levels of performance.

We tackle this by using the CCRL Dataset to train the Neural Network in the AlphaZero which approximates the policy and value function given a state. This allows us to implement MCTS to search for the best moves as done in AlphaZeo.

While AlphaZero and its variants have mostly been used in gaming applications, the project is not limited to this. Theoretically, it can be used to solve any problem that can be defined as an agent acting upon an environment with discrete state-action spaces. For instance, AlphaGo has been used by Google to properly cool their data centers more efficiently. The algorithm helped lower data center cooling costs by 40%. It does this by periodically pulling data from many sensors and feeding that data to a deep neural network to predict how different combinations of potential actions will affect future energy consumption.

Reinforcement learning in general has been used to solve a wide range of problems. Another example in the gaming industry is the use of multiagent reinforcement learning to play as a single team of five agents in the game DotA 2 which is a multiplayer video game where two teams fight against each other, meaning team play is crucial. The goal of the game is to destroy the enemy team's base. This is a very difficult task for an AI to accomplish, as completing this goal often takes around 40 minutes. Simple reward functions are therefore impossible. OpenAI has successfully created an AI that can defeat even the best DotA players in the world with large-scale deep reinforcement learning. In the field of robotics, (deep) reinforcement learning can be used to teach a robot through self-play to perform simple tasks that previously required manual programming. Some examples of this are a robot learning how to flip pancakes and a robot learning how to play table tennis.

Compared to most chess engines which use the Alpha-Beta algorithm and a handwritten evaluation function, AlphaZero evaluates a lot fewer positions every second. Most top programs consider about 45 million chess positions every second, whereas this program considers about 300, even with GPU and multi-core speedups.

2. Background

2.1. *The Game*

Chess is a board game that involves two players and requires strategic thinking and tactical skills. The game is played on a square board with 64 squares arranged in an eight-by-eight grid. Each player has 16 pieces: eight pawns, two rooks, two knights, two bishops, one queen, and one king. The goal of the game is to checkmate the opponent's king, which means putting it in a position where it is under attack and cannot escape capture on the next turn. The game can also end in a draw. Chess has a long history and is considered one of the most popular and challenging strategy games in the world. It is played competitively in international tournaments and championships, and there are various variations and adaptations of the game, such as speed chess, blitz chess, and chess960. The rules of chess are standardized by the World Chess Federation (FIDE).

2.2. Game Setup

In standard chess, the board is set up with the pieces in a specific arrangement. Starting from the bottom left corner, the first row contains a rook, knight, bishop, queen, king, bishop, knight, and rook, while the second row is filled with eight pawns. The opponent sets up their pieces in the same way on the opposite side of the board.

However, some variations of chess, such as Fischer Random Chess (also called Chess960), shuffle the pieces on the first and last rows before the game begins. This randomization introduces an additional level of unpredictability and forces players to rely on their knowledge of chess principles rather than rote memorization of opening moves.

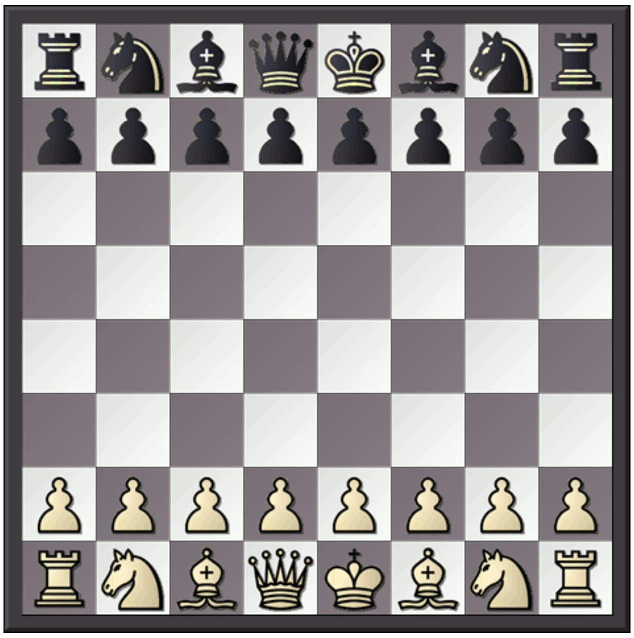


Figure 1: Chessboard setup at the initial position.

2.3. Movement of the pieces

Pawns move one or two squares forward from their initial position, capture diagonally, and can promote to a queen, rook, bishop, or knight. En passant is a special pawn move that allows an opponent's pawn to capture a pawn that has moved two squares forward. The rook moves vertically or horizontally, the bishop moves diagonally, and the queen can move in any direction. The knight moves in an L-shape and can leap over other pieces. The king moves one square at a time and can castle to move two squares towards a rook for safety.

2.4. Working principles of traditional chess engines

2.4.1. The minimax algorithm

Chess engines like Stockfish use a variation of the minimax algorithm that incorporates alpha-beta pruning. The minimax algorithm is a general algorithm that can be applied to a wide range of applications such as artificial intelligence, game theory, and decision theory. In chess, the algorithm seeks to minimize the maximum possible loss, which translates to minimizing the chances of the opponent achieving checkmate. Minimax creates a search tree consisting of chess positions as nodes and legal chess moves as edges. The nodes hold values representing the strength of the position for the current player, and the algorithm explores the tree in a depth-first manner by selecting random legal moves and creating nodes and edges in the process. The leaf node's value is then returned upwards to the parent node, and the maximum or minimum value is selected depending on whether the player is white or black. This process continues until the root node is assigned a value representing the strength of the current position.

2.4.2. Alpha-beta pruning

To optimize the algorithm and reduce the number of nodes explored in minimax, alpha-beta pruning is used. This technique eliminates branches in the search tree that lead to unfavorable outcomes. By assuming that the opponent will always play the best move, minimax can accurately estimate the strength of a position while minimizing the maximum loss. In the example shown, the branch with a value of 8 can be pruned because it leads to a winning position for white.

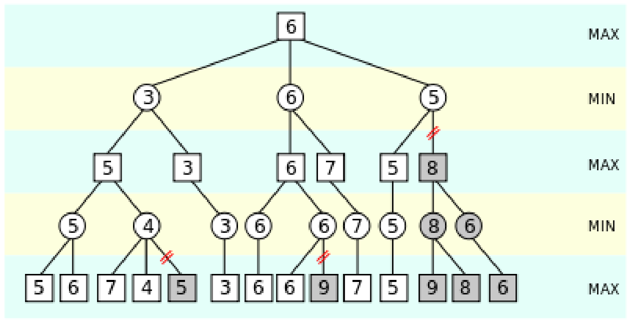


Figure 2: Example of alpha-beta pruning.

2.4.3. *The evaluation function*

The evaluation function used to estimate the value of leaf nodes is game-specific and varies between engines, often developed with input from chess experts.

3. Approach

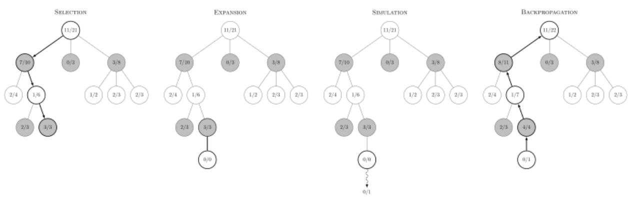


Figure 3: The 4 steps of the MCTS algorithm.

3.1. *MCTS*

One issue with minimax methods is that they heavily rely on the accuracy of the evaluation function, which is sub-optimal and may lead to the suggestion of bad moves. Since 2020, StockFish has been utilizing a sparse and shallow neural network to evaluate positions, though it is still trained through supervised learning and not reinforcement learning. However, using alpha-beta pruning can also pose a challenge as its short-sighted and does not consider many moves into the future. For instance, if a player can sacrifice a piece for a significant advantage later in the game, the algorithm may prematurely prune that branch without exploring the winning line, since it regards the sacrifice as a losing position. To mitigate these challenges, the Monte Carlo Tree Search (MCTS) algorithm can be employed which approximates a position's value by generating a search tree via random exploration of the most promising moves. To construct this search tree for a specific position, MCTS executes the following algorithm several hundred times, each of which constitutes an MCTS simulation.

The Monte Carlo Tree Search (MCTS) algorithm involves four steps. To begin, the root node is selected, and then a child node is chosen based on a formula, often using Upper Confidence Bound (UCB). The selection process continues until an unvisited node, or leaf node, is reached. If the root node is a leaf node, the next step is executed immediately. In the Expansion step, if the leaf node represents the end of the game, the algorithm moves on to the Backpropagation step. If not, a child node is created for every possible move that can be made from that node. Next, in the Simulation/Rollout step, a random child node from the previous step is chosen, and the rest of the game is simulated using only random moves. Finally, in the Backpropagation step, the simulation's outcome is returned up the tree. Each node tracks the number of times it has been visited and the number of times it has resulted in a win. In the case of chess, this algorithm can be inefficient since it requires simulating an entire game in the third step of each simulation. Hundreds of simulations may be necessary to obtain a good estimate of the value of a single position. Therefore, the selection formula should be chosen carefully to balance exploration and exploitation.

3.2. *AlphaZero*

AlphaZero is a generalized version of AlphaGo Zero which was created to master the game of Chess, Shogi, and Go. On evaluation against the StockFish chess engine for 1000 games, it won 155 times, lost 6 times, and drew the remaining games.

3.2.1. *Neural Network*

Unsurprisingly there's a neural network at the core of things. The neural network f_θ is parameterized by θ and takes the state of the board (s) to output the value function $v_\theta(s)$ and the optimal policy $\pi_\theta(s)$

Deepmind tested multiple architectures for AlphaZero and the following model was used:

- A convolutional block which consists of a convolutional layer followed by batch normalization and ReLU.
- A residual block consisting of 2 convolutional blocks and a skip-connection.

The final architecture used consisted of 20 residual blocks with combined policy and value heads.

3.2.2. *Training*

Instead of training the neural network using self-play, we train on theCCRL dataset. This dataset consists of multiple files with 8000 chess games. In each epoch,

- We iterate through a random batch of chess games
- Get the position of the board, the next move, and the outcome of the game.
- Process the data through the model and back-propagate to adjust the weights and biases.

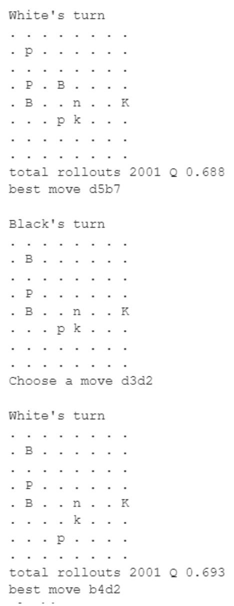
We use the mean square between the predicted value function with the actual outcome of the game as the error metric and Cross entropy as the error metric to train the policy function.

4. Related Work

A very prominent chess engine, Stockfish uses a heuristic evaluation function created by computer scientists and chess grandmasters to estimate the value of a position, but this is not always enough to deal with the complexity of chess. To handle this, Stockfish uses a minimax algorithm to consider potential strategies and the best responses from the opponent. The engine then prunes the search tree using alpha-beta pruning, considering the most promising moves first and ignoring moves that won't improve the current position. To maximize the chances of winning, Stockfish's evaluation function prioritizes aggressive and tactical moves, and the engine also uses null-move searching to gain insight into potential moves. By utilizing these techniques, Stockfish aims to minimize the chance of making an error and increase the probability of playing a winning move.

Our implementation of the algorithm is inspired by the AlphaZero paper, but there is an important difference: instead of using reinforcement learning, it employs supervised learning. This is because reinforcement learning is very computationally expensive, the original paper states that 4 TPUs are used to generate self-play games. In contrast, this program trains on the CCRL Dataset, which consists of 2.5 million high-quality chess games. Since each game has approximately 80 unique positions, this provides about 200 million data points for training.

5. Results and Evaluaton:



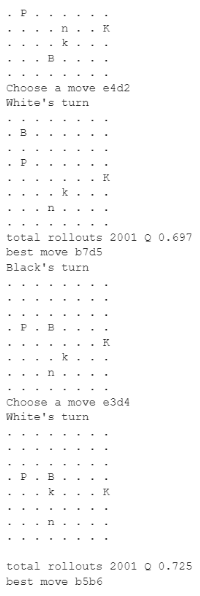
```
White's turn
. . . . . . . .
. p . . . . . .
. . . . . . . .
. P . B . . . .
. B . . n . . K
. . . p k . . .
. . . . . . . .
. . . . . . . .
total rollouts 2001 Q 0.688
best move d5b7

Black's turn
. . . . . . . .
. B . . . . . .
. . . . . . . .
. P . . . . . .
. B . . n . . K
. . . p k . . .
. . . . . . . .
. . . . . . . .
Choose a move d3d2

White's turn
. . . . . . . .
. B . . . . . .
. . . . . . . .
. P . . . . . .
. B . . n . . K
. . . . k . . .
. . . p . . . .
. . . . . . . .
total rollouts 2001 Q 0.693
best move b4d2
```

Figure 4: ASCII image suggesting the best moves by our chess engine.

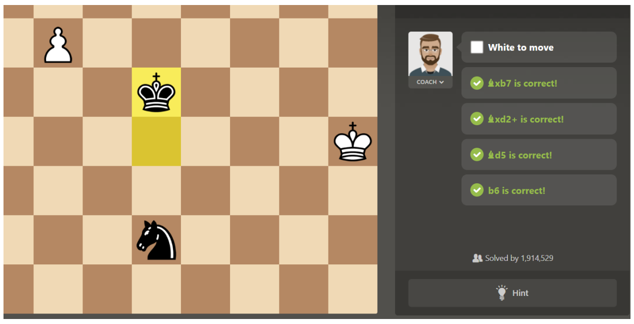
The engines was tested on a few handpicked puzzles from the chess.com website. Fig [4] and Fig[5] are



```
. P . . . . . .
. . . . n . . K
. . . . k . . .
. . . B . . . .
. . . . . . . .
Choose a move e4d2
White's turn
. . . . . . . .
. B . . . . . .
. . . . . . . .
. P . . . . . .
. . . . . . . K
. . . . k . . .
. . . n . . . .
. . . . . . . .
total rollouts 2001 Q 0.697
best move b7d5
Black's turn
. . . . . . . .
. . . . . . . .
. . . . . . . .
. P . B . . . .
. . . . . . . K
. . . . k . . .
. . . n . . . .
. . . . . . . .
Choose a move e3d4
White's turn
. . . . . . . .
. . . . . . . .
. . . . . . . .
. P . B . . . .
. . . k . . . K
. . . . . . . .
. . . n . . . .
. . . . . . . .

total rollouts 2001 Q 0.725
best move b5b6
```

Figure 5: ASCII image suggesting the best moves by our chess engine.



White to move

♗xb7 is correct!

♗xd2+ is correct!

♗d5 is correct!

b6 is correct!

Solved by 1,914,529

Figure 6: Actual sequence of best moves.

the moves suggested by the chess engine after running the MCTS and Fig[6] with the chessboard is the Actual sequence of best moves. Our implementation predicts the best sequence of moves in about two minutes of compute time with about 100 percent accuracy when tested on random chess puzzles and games for an ELO ranging from 600 - 1000.

6. Conclusions

In conclusion, developing a chess engine is a challenging but rewarding task. Our implementation gives good results as is evident by the results shown above. Our results shows that we can get good results for chess models even if we have less compute power by approching the training method in a supervised learning fashion instead of implementing reinforcement learning from scratch. Since there is always room for improvement, and future work could include experimenting with different evaluation functions or search techniques to further enhance the engine's performance. Overall, this project has provided valuable insights into the world of game programming, and has demonstrated the power of algorithms and machine learning in solving complex problems.

References

1. Deepmind, Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm, 2017

2. Deepmind, Mastering the game of Go with deep neural networks and tree search,2016

3. December and 2018, Has Google cracked the data center cooling problem with AI?, 2020.Online

4.OpenAI Five, 2019. Online

5. Learning to Select and Generalize Striking Movements in Robot Table Tennis, 2013