DRAFT ... Us Versus Stockfish:

Using Nerual Networks to Play Chess

Philmore Morrison

philmoremorrison2023@u.northwestern.edu

Nov 18, 2021

Abstract

We use a deep neural network to learn how to play chess against the famous Stockfish program.

Keywords: Chess, Stockfish, Policy Estimation, Deep Learning

Introduction

We survey multiple learning methods for learning policy approximation in reinforcement learning. Our objective is to train a neural network model to play chess.

... add more information to the introduction section.

https://github.com/arjangroen/RLC

is a great example of the types of agents we can explore.

Literature Review

A long-standing ambition of artificial intelligence has been to create programs that can instead learn for themselves from first principles (Silver et al. 2018). Researchers have used countless resources in the pursuit of a proficient chess-playing algorithm. Through experience, one might learn to predict whether chess positions will lead to a win (Sutton 1988). The game of chess is the longest-studied domain in the history of artificial intelligence (Silver et al. 2018). The seminal works of Claude Shannon, Alan turning, and countless researchers derived the theory and built hardware to play chess efficiently. Chess programs evaluated positions using handcrafted features and carefully tuned weights, constructed by strong human players and programmers, combined with a high-performance alpha-beta search that expands a vast search tree using a large number of clever heuristics and domain-specific adaptations (Silver et al. 2018). Handcrafted heuristics guarantee that subsequent chess algorithms could not be useful outside of the world of chess. There are also too many permutations and moves for humans to hand-code an algorithm effectively. The rules in chess are position and piece dependent. For example, pawns can move two steps forward from the second row (player‘s perspective). The queen can move across the board freely, and under the right circumstances, a player can castle. We can describe a move in chess in two parts. The agent first selects a piece to move, then selects aan action from the possible moves for the chosen piece.

Deep Blue was developed at IBM in the 1990s after multiple years of research and development, focusing on building a world-class chess machine (Campbell, Hoane, and Hsu 2002). Deep Blue I lost to Garry Kasparov in 1996, four to two. The Deep Blue team modified the Deep Blue architecture and rechallenged Kasparov again in 1997. Deep Blue defeated Garry Kasparov in the 1997 match by a score of 3.5–2.5, and the Deep Blue team was awarded the Fredkin prize for defeating the human world champion in a regulation match (Campbell, Hoane, and Hsu 2002).

Chess subsequently became a grand challenge task for a generation of artificial intelligence researchers, culminating in high-performance computer chess programs that play at a super-human level (Silver et al. 2018). Monte Carlo Tree Search (MCTS) is one such method. The MCTS algorithm consists of the following phases; selection, expansion, simulation, and backpropagation. During the selection phase, the algorithm searches the portion of the tree that has already been represented in memory (Świechowski et al. 2021). Next, the expansion phase is triggered if the selection phase does not reach a terminal state. The expansion adds new nodes corresponding to state s-prime (new-state) based on the previous action. Next, simulation implements the Monte Carlo part of the algorithm to estimate possible reward and terminal state. Finally, backpropagation distributes the simulated scores up the tree to the root node while updating the statistics for visits n_i and state estimate S_i.

MCTS has an advantage over traditional depth-limited minimax search with alpha-beta pruning in games with high branching factors such as Go, while minimax search with alpha-beta pruning surpasses MCTS in domains like Chess (Lin 2017). MCTS does not detect shallow traps as well as minimax search. Missing a shallow trap has the consequence of an opponent winning within a few moves. During minimax search, we build a tree from position x by examining all possible moves for the computer in that position, then all possible moves for the opponent, followed by all possible moves for the computer to a predetermined depth d (Baxter, Tridgell, and Weaver 2000).

Temporal Difference (TD) learning is another algorithm for learning policies. Arthur Samuel introduced TD learning in 1959. Samuel (1959) created a checkers playing system based on TD learning. Sutton (1988) contributed to the literature by using TD learning as an incremental learning process for making predictions. TD approximates the expected long-term future cost (or cost-to-go) of a stochastic dynamical system as a function of the current state (Baxter, Tridgell, and Weaver 2000). The main objective is to minimize the error between the prediction for the current and the next state. TD learning became popular because it combined the advantages of dynamic programming and the Monte Carlo method without the total computational overhead. TD learning also leverages the bootstrap method similar to Dynamic Programming. One significant benefit of TD learning is that we do not need to wait until the end of an episode to compute the Q value. Said another way, “TD learning bootstraps like DP, so it does not have to wait till the end of the episode, and like the MC method, it does not require the model dynamics of the environment to compute the value function or the Q function” (Science 2021). We provide a visual comparison between MCTS and TD learning in figure 1.

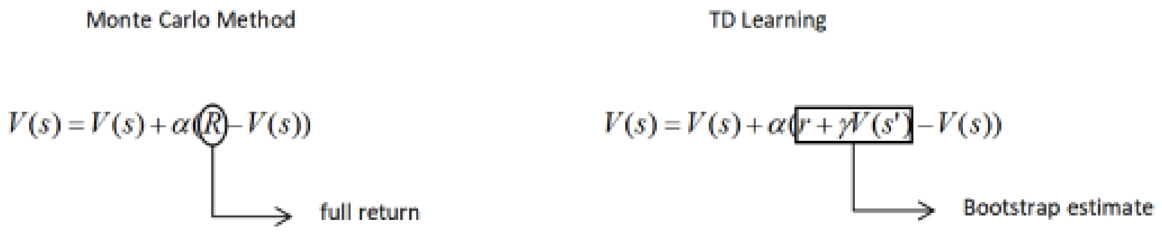
Monte Carlo Method
$V(s) = V(s) + \alpha(\underbrace{R}_{\text{full return}} - V(s))$
full return
TD Learning
$V(s) = V(s) + \alpha(\boxed{r + \gamma V(s')} - V(s))$
Bootstrap estimate

Figure 1. *Value function based on Monte Carlo tree search versus Temporal Difference learning*

AlphaZero replaces the handcrafted knowledge and domain-specific augmentations used in traditional game-playing programs with deep neural networks, a general-purpose reinforcement learning algorithm, and a general-purpose tree search algorithm (Silver et al. 2018). AlphaZero uses a deep neural network to play chess. The network uses board position as inputs and outputs for move probabilities based on the non-linear evaluations of the board position. AlphaZero learned to approximate the policies through self-play and experience replay based on the Monte Carlo tree search algorithm. AlphaZero can search 60,000 positions per second in chess. Silver et al. (2018) states that AlphaZero defeated Stockfish when given 1/10 as much thinking time as Stockfish. Stockfish is an open-source chess program that is ranked as the strongest CPU chess engine in the world.

We leverage the work of the practitioners that came before us to build neural network agents to play chess against the Stockfish chess program. The remainder of this document is as follows. We discuss the training dataset next, followed by our design of experiments, methodologies, and results. Finally, we conclude the paper with our overall findings and discussion for future enhancements.

Data

Discuss the data that we use in the experiments ... experience replay (self-play) or against an opponent (Stockfish). Elo range for 'training'.

Methods & Results

We conduct a series of experiments using different network topologies. Our agent competes against Stockfish. We need to frame the problem and produce some plots to include in the final presentation.

Experiment One

Exp one is ...

Experiment One

Exp two goes burrrrrr ...

Conclusion & Future Works

In closing, our results show ... blah blah blah

Future considerations include parallel or multi-processing during training or leveraging REINFORCE or Actor-Critic methods.

References

Baxter, Jonathan, Andrew Tridgell, and Lex Weaver. 2000. "Learning to Play Chess Using Temporal Differences." *Machine Learning* 40 (September): 243–263. https://doi.org/10.1023/A:1007634325138.

Campbell, Murray, A.Joseph Hoane, and Feng-hsiung Hsu. 2002. "Deep Blue." *Artificial Intelligence* 134, no. 1 (January): 57–83. Accessed November 8, 2021. https://doi.org/10.1016/S0004-3702(01)00129-1. https://linkinghub.elsevier.com/retrieve/pii/S0004370201001291.

Lin, Jonathan Fun. 2017. "MONTE CARLO TREE SEARCH AND MINIMAX COMBINATION – APPLICATION OF SOLVING PROBLEMS IN THE GAME OF GO." Publisher: Digital Repository at the University of Maryland, accessed November 22, 2021. https://doi.org/10.13016/M2VD6P64Q. http://drum.lib.umd.edu/handle/1903/20449.

Samuel, A. L. 1959. "Some Studies in Machine Learning Using the Game of Checkers." *IBM Journal of Research and Development* 3 (3): 210–229. https://doi.org/10.1147/rd.33.0210.

Science, ODSC-Open Data. 2021. "Understanding the Temporal Difference Learning and its Predication." Medium, January 11, 2021. Accessed November 8, 2021. https://medium.com/@ODSC/understanding-the-temporal-difference-learning-and-its-predication-efc91016144a.

Silver, David, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, et al. 2018. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play." *Science* 362, no. 6419 (December 7, 2018): 1140–1144. Accessed November 8, 2021. https://doi.org/10.1126/science.aar6404. https://www.science.org/doi/10.1126/science.aar6404.

Sutton, Richard S. 1988. "Learning to predict by the methods of temporal differences." *Machine learning* 3 (1): 9–44.

Świechowski, Maciej, Konrad Godlewski, Bartosz Sawicki, and Jacek Mańdziuk. 2021. *Monte Carlo Tree Search: A Review of Recent Modifications and Applications.* arXiv: 2103.04931 `[cs.AI]`.