6.884 Midterm Report: RL Chess - Smoothing the Learning Curve

Raphael Chew
raphael1@mit.edu

Shaun Fendi Gan
shaungan@mit.edu

Peijun Xu
xup@mit.edu

Prelude

Since the project proposal, we explored the problem of how to implement RL in chess with the aim of exploring different chess openings. We started by investigating the different stages of chess complexity, starting from capture chess where an agent was trained to just focus on capturing pieces, before transitioning to real chess, where the goal is to checkmate. Through implementing and modifying raw source code for RL agents and environments, we discovered that there was a larger problem to be tackled - how can we even create a proficient chess engine that can explore these chess openings? Therefore, we have decided to pivot our project to focus on the larger issue of exploring how to reduce the steep learning curve / sparse reward structure to a more lenient learning curve. Exploring chess openings will now act as an accessory to the project.

A comprehensive plan for this can be found below.

1 Problem Motivation

Current implementations of chess RL bots require days to train with high GPU effort. Can we implement tools (from lectures and class) that can create a sufficiently strong chess bot with less time / computation? Can we build a competent chess bot from scratch that wins against a 500 ELO player, within 3 hrs of training time on Colab Pro? We will explore the state space problem by building Q-learning and policy learning engines from the ground up, experimenting with various reward functions and lookahead states, and reporting how each chess bot performs.

2 Describing Method in Detail

The overall goal of building a reinforcement learning chess bot within our computational limits is chronologically broken down into the following steps:

1. Implementing a Q-learning and policy learning chess agent that masters "Capture Chess"
2. Using novel neural network architectures (currently, CNNs) to learn the large state space by creating board embeddings at low dimensionality
3. Benchmarking our agent against currently available agents online
4. Experimenting with the reward shaping and lookahead states to enable the "Capture Chess" bot to play real chess. This could include MCTS.
5. If necessary, reducing the state space by focusing on specific openings
6. Refining the chess bot's playing style through demonstration learning and learning against multiple opponents in random sequence

3 Identify the Exact Experimental Setup and Evaluation Criterion

We will execute experiments in the following manner:

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

1. For Capture Chess, built Agents will play against two types of opponents (as both white and black):
   (a) Random agent
   (b) Existing agent (Arjan's implementation of Q-learning Capture Chess)
   (c) Evaluation criterion will be the endgame material advantage
2. For Real Chess, built Agents will play against a Stockfish bot of 500 ELO (or any existing chess bot of benchmarked ability)
   (a) Evaluation criterion will be the endgame result (checkmate or not) and the material advantage at the end of the game or at the end of the specific opening
3. Each match-up will consist of 200 games, where the mean and standard deviation of the criterion will be used as metrics for the bots' performances
4. Training times will be stored alongside performance to understand the current Q-learning or policy learning framework's computational requirements

4 Related Work

Our work is inspired by Deepmind's paper, *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm* and based on its idea of using reinforcement learning to train agents to play chess. We would re-implement the Monte Carlo Tree Search that DeepMind implemented in the paper. However, due to limited resources available for a course project, we'd like to investigate whether we can train a competent chess bot in a relatively short time with limited resources through experimenting with various neural network representations and reinforcement learning frameworks. We also recognize there are other attempts trying to devise effective chess algorithms but these attempts didn't yield effective results that can produce competent agents. As we referenced in our project proposal, an overview of standard methods is mentioned in the paper, *Computer chess methods. In S. Shapiro, editor, Encyclopedia of Artificial Intelligence*. Although these methods didn't utilize reinforcement learning, we think they provide interesting insights that enable us to develop better model frameworks by empowering these algorithms with reinforcement learning. Other examples of attempts online yielded different interesting methods, but definitely yield room for exploration.

5 Addressing Comments in the Project Proposal

5.1 Comment: Do you have enough computational resources to do this project?

We do not have the computational resources to build an AlphaZero bot, but that is not what we are trying to accomplish. The goal of this project is to implement reinforcement learning structures that shrink the necessary state space of learning, so that we can build a competent chess bot on Google Colab Pro's GPUs. For example, our current Q-learning implementation uses a 18-parameter, 2-layer CNN to learn the 64x64 chess board and return logical moves for Capture Chess within 45 minutes of training. We understand that we will not be able to build a high-standard chess bot by conventional means of assessing chess play, but we hope to investigate reinforcement learning structures that enable simplified learning within less time.

5.2 Comment: Is the goal to adapt a RL agent to play against a player that makes a specific opening move? This seems like a straightforward application fo AlphaZero with some minor changes in gameplay initially. What exactly do you hope to learn from this project? Simply running AlphaZero won't cut the bar for a good course project (unless you implement it yourself and investigate different design choices).

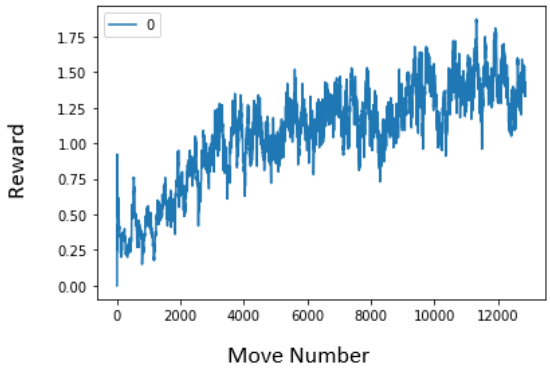
The goal is not to re-implement AlphaZero, but to build Q-learning and policy learning agents from scratch. We understand that AlphaZero is the gold standard, so we will attempt to iterate through the policy state space by using the monte carlo tree search that they implemented in their paper. However, given computational requirements, we may have to reduce the state space by narrowing its application to a specific opening. This is not the main goal of the project though; the main goal is to learn how different neural network representations and reinforcement learning frameworks

affect the training time of a competent chess bot. Also, we want to investigate how reward shaping can train very different playing styles that conventional policy learning bots that are focused on the eventual checkmate reward. Our own goal for this project is learn how to build these agents, engines, and environments on our own (of course with some help from homework and online examples), in addition to getting a deep understanding of reward shaping and how different algorithms converge with different behaviors.

6 Updates on Current Progress

Our current completed work consists of scraping basic open source RL chess implementations, understanding nuances of the chess environment, and initial exploration into methods for creating a reliable checkmate chess agent. Our progress is summarised as:

- Scraped chess environment and agent from available open source code
  - https://github.com/arjangroen/RLC (link this later)
- Implemented Q learning of Arjan's agent against random agent
- Explored training a Q learning agent, against an agent with experience against random agents.
- Modified rewards to perform preliminary exploration into considering reward after black's move, not just white's move
  - DDQN setup
  - Reward used
    - Capturing one state ahead, considering both white and black moves.
- Set up framework to also train as black, not just white.
- Many experimentations of different agents that have produced promising results,



	MoveNumber	Min	Max	Mean	Median	Std
0	5	-3	13	1.41	0	3.280533
1	10	-1	26	5.23	1	7.042521
2	15	-2	34	10.90	10	9.734988
3	20	-2	38	16.58	18	11.066327
4	25	-1	39	20.65	22	10.868648
5	30	-1	39	23.64	25	9.929270
6	35	0	39	25.99	26	9.182042
7	40	11	39	28.26	28	6.686733
8	45	14	39	29.55	30	6.153657
9	50	18	39	30.05	31	5.720795
10	55	18	39	30.21	31	5.655608
11	60	18	39	30.33	31	5.639246

Figure 1: **LHS**: shows the rolling average across 10 moves in a capture chess environment. This shows that the environment is learning how to be a competent agent. **RHS**: shows the statistics for the reward after a certain move number. This displays its competency as the game progresses to both capture pieces and maintain an advantage. By move 45, we can see that our capture chess agent has 30 point mean advantage.

7 Plan for Next Month

Amongst all the possibilities for methods of discovery, our interests of avenues to explore can be broken down into these broad topics

- Rewards Shaping

- Training Environments (e.g. self play)
- Increasing search space
- Imitation / Demonstration learning
- Exploring different NN Architectures (CNN) or different models (PPO, HER)

Week 9 (Apr 12th)	1. Rewards Shaping a. Include checkmate into rewards and explore different weight combinations (Current rewards only include piece values) b. Include rewards for promoting to queens (or perhaps hard coding this [Shaun] 2. Training Environments a. Evaluate ceiling of training against random (Agent name: Arjan) b. Evaluate training against an agent trained against random (vs. Arjan) c. Modify coding framework to allow for self play as it is currently restricted to only white or black. [Raph] 3. Network Architecture a. Explore different network structures (CNN structures or different activations, different linear sizes) [Peijun]
Week 10 (Apr 19th)	1. Training Environments a. Train the agent to play against itself and evaluate performance 2. Search Space a. Modify reward to account for more future predicted rewards (discounting, etc) b. MCTS exploration 3. Imitation / Demonstration Learning a. Train alongside stockfish - allow it to make every alternate move during training (e.g.)
Week 11 (Apr 26th)	1. Different Methods a. Attempt PPO 2. Explore Chess Openings a. Reduce the state search space to moves 5-15
Week 12 (May 3rd)	1. Ensemble of Methods [Extension] a. As we explore and have find methods that work, it would be interesting to combine different networks to specialise on different tasks in chess (mid-game vs. end-game for example) 2. Explore HER models for subrewards [Extension]
Week 13 (May 10th)	1. Report Writing and Presentation Preparation