Reinforcement learning in Chess

Ganesh Arvindh Ramanan
SC21B150
Engineering physics

Harshit Dhanwalkar
SC21B164
Engineering physics

Abstract

The paper we have chosen for our project is Block et al. [2]. The paper tries to improve the performance of the KnightCap engine implemented by Baxter et al. [1], Tridgell [4]. Chess has historically been seen as a hallmark for intelligence. The fact that the process of playing chess has been automated using machine learning piqued our interest. In the paper a chess engine was developed which reached an elo of 1800 ± 50 after just 50 games. In this paper we made a chess engine in python using the TD(λ) algorithm. We found that the engine reached an approximate elo of 1000 in a few games.

1 Introduction

The goal of this project was to create a chess engine which will play at the level of a human. We implemented the chess engine using a reinforcement learning algorithm called TD(λ). This algorithm was first tested in the program SAL by Gherrity [3]. Although not very successful, this lead to improvements in the algorithm over time. Temporal difference was first successfully implemented in the program KnightCap by Baxter et al. [1], Tridgell [4]. In the first trial run of KnightCap, the program reached an elo of 2150 from 1650 in only 3 days after 308 games.

2 Model and problem statement

2.1 The environment

The environment used in the project is the opponent itself. This environment is quite different from the traditional environments used in algortihms like ϵ-greedy algorithm as it doesn't give a reward for every action. This will become more clear when we discuss how the engine works.

The state space S is the set of all possible states or chess positions where $x_t \in S$ is the state at time t. Note that t was obtained after t numbers of actions or moves by the engine. Let us assume that the game was played for N moves. Thus, $T = N$ is the final time-step. The action space A_t consists of all the legal moves that can be played by the engine at time t and consequently, $a \in A_t$ is the move played by the engine at t.

After each game, the engine receives a reward for the final move $r(x_N)$ which is given by

$$r(x_N) = \begin{cases} 1 & \text{for victory} \\ -1 & \text{for loss} \\ 0 & \text{for draw} \end{cases}$$

We also define an evaluation function $J'(x)$ which takes in a board and returns a number between -1 and 1. This number denotes how good the position is (positive for winning and negative for loosing). We can model this as

$$J'(x) = E_{x_N|x} r(x_N) \tag{1}$$

Thus $J'(x)$ becomes the expected value of terminal reward given the current state.

2.2 Use of the evaluation function

The evaluation function is used along with a parameter called 'depth' to assign a reward for every possible move. The quantity 'depth' denotes the number of moves that the engine will think ahead to determine the quality of the position that will be reached. This quality is determined using the evaluation function $J'(x)$.

When it is the engine's turn to play, it iterates through all possible moves. For each move, it thinks ahead by a number of moves and the evaluation of the resultant position is used as a reward for the move under consideration. After this process is complete, the engine plays the move which gives the maximum reward.

2.3 Performance evaluation

To get the approximate elo of the engine, we pair it with an engine of fixed elo in chess.com. chess.com has a feature which determines the elo of the player after the game is over. Multiple games can be played with the same opponent and the average of all the estimated ratings will be the elo of our engine.

Optimization criteria : Since it is very difficult to precisely determine the quantity $J'(x)$ as given in Equation 1, we use an approximate linear function $J(x)$ given by,

$$J(x) = \sum_{i=1}^{k} \omega_i J_i(x) \tag{2}$$

Each of these functions ($J_i(x)$) represents a quantity which will determines a parameter which serves as an indication of the evaluation of the position. A good example will be a function which determines the total count of material on the board as having more material increases the probability of winning. The goal then, it to determine the optimal value of the parameters $\{\omega_i\}$ to maximize performance.A good set of parameters will lead to a accurate evaluation function which will in theory improve performance. For a given set of parameters, the elo of the engine can be found using the process described above.

3 The reinforcement learning agent

In the paper by Block et al. [2] the TD(λ) algorithm was improved to TD-leaf(λ) algorithm to improve performance. However, in this paper, we will use a rudimentary version of the TD(λ) algorithm implemented by Baxter et al. [1], Tridgell [4]. The algorithm implemented in this paper is shown below.

After the end of a game played using a fixed set of parameters $\{\omega_i\}$, we obtain a state sequence $x_1, x_2, ..., x_N$. We now compute the *temporal difference* at every time-step:

$$d_t = J(x_{t+1}, \omega) - J(x_t, \omega)$$

For the last time-step, the function $J(x_N, \omega)$ can be written as $r(x_N)$. Thus,

$$d_{N-1} = r(x_N) - J(x_{N-1}, \omega)$$

We now define a quantity called the *strength of correction* as

$$\Delta t = \sum_{j=t}^{N-1} \lambda^{j-t} d_j$$

The update of ω is performed as

$$\omega \leftarrow \omega + \alpha \sum_{t=1}^{N-1} \nabla J(x_t, \omega) \Delta t \tag{3}$$

After this, another game is played using the updated coefficients and the corresponding elo is measured. The above process is repeated until the optimal values of the coefficients are reached.

References

[1] Jonathan Baxter, Andrew Tridgell, and Lex Weaver. Knightcap: A chess program that learns by combining td () with minimax search. In *Proceeding 15th International Conference on Machine Learning*, pages 28–36, 1997.

[2] Marco Block, Maro Bader, Ernesto Tapia, Marte Ramírez, Ketill Gunnarsson, Erik Cuevas, Daniel Zaldivar, and Ral Rojas. Using reinforcement learning in chess engines. *Research in Computing Science*, 35:31–40, 2008.

[3] Michael Gherrity. *A game-learning machine*. University of California, San Diego, 1993.

[4] Andrew Tridgell. Knightcap-a parallel chess program on the ap1000. In *Proceedings of the Seventh Fujitsu Parallel Computing Workshop*, 1997.