Ozyegin University

CS 545 Deep Reinforcement Learning

Assignment1

Solving Checkers with Value Iteration

Author:
Jareth Moyo

November 2, 2017

1 Introduction

The goal of the assignment was to develop an RL agent that learns to play checkers by using value iteration. I decided to develop everything from scratch, from the game rules to the RL agent itself, and my results are therefore based on these designs. I also wrote a visualization GUI to help visualize how the agent plays, i.e., whether the moves it makes actually make sense or not. Please run visualisation.py in order to use this tool, it has no dependencies outside your standard python installation (see requirements.txt for more information on this).

2 State Space and MDP

Initially I had used the actual board positions to model the state, but this approach was making it too difficult to store all the states, even after eliminating the "unlikely" states. I then used a very simple state space definition to define the notion of a "state". I represented the state space by the following :

- number of own uncrowned pieces
- number of opponent's uncrowned pieces
- number of own crowned pieces
- number of own uncrowned pieces

My reward function was defined as in the assignment description, i.e., +100 for a win, -100 for a loss and 0 for a draw. However, I also introduced a time step reward of-0.5 from moving from any one state to another (I didn't want to actually use the state information to define the reward, because it feels like then the agent would be cheating, as it is supposed to learn on its own what states are good and what states aren't.). This was at an attempt of making the agent not only learn how to win, but also how to win fast. I was making my RL agent go against a random opponent. Since I made the opponent a random agent, that means each of his moves were equally probable. Therefore, for each state I defined the p(s', r — s, a) as 1/(number of actions available to the random opponent). This is because all possible s' states are dependent on how the opponent moves, and since

the opponent chooses all actions randomly (and uniformly), this distribution made the most sense to draw from.

3 Results

The agent does show intelligent behavior (as you will see when you run the visualization tool). However, since my state definition does not completely represent the game of checkers (I made it simple to make value iteration plausable for this problem), it does make suboptimal moves at times. I saved the values learnt by the agent in the file valueouput.txt for reference.