Playing Chinese Checkers with Reinforced Learning

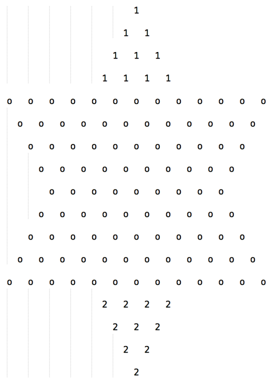
CS 229 Spring 2016 Project Milestone

Sijun He, Wenjie Hu, Hao Yin.
[sijunhe, huwenjie, yinh]@stanford.edu

1 Introduction

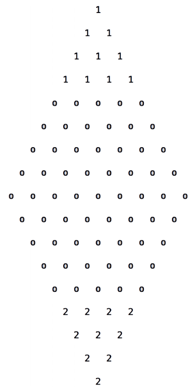
Chinese checkers is a game played on a hexagram-shaped board that can be played by two to six players individually or as a team. The objective is to be the first to move all ten pieces across the board into the opposite starting corners. The allowed moves include rolling and hopping. Rolling means simply moving one step in any direction to an adjacent empty space. Hopping stands for jumping over an adjacent piece into a vacant space. Multiple continuous hops are allowed in one move. A more detailed introduction of the Chinese checkers can be seen in Wikipedia.

To simplify the problem, in this project, we will focus on solving the 1 vs 1 mode of the Chinese checkers. The left-minipage of Figure 1 is the starting board of this case, where each **o** stands for a vacant spot, **1** stands for a spot occupied by player 1's piece, and **2** stands for a spot occupied by player 2's piece. In order to reduce the computation cost during the development, we modify the hexagram-shaped board into a heuristic diamond-shaped board, as in shown in the middle-minipage of Figure 1. Moreover, we use a small board of 3 pieces for each player for testing and development purpose at the beginning.



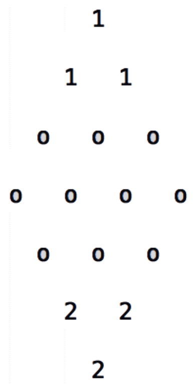
```
            1
           1 1
          1 1 1
         1 1 1 1
o o o o o o o o o o o o o
 o o o o o o o o o o o o
  o o o o o o o o o o o
   o o o o o o o o o o
    o o o o o o o o o
   o o o o o o o o o o
  o o o o o o o o o o o
 o o o o o o o o o o o o
o o o o o o o o o o o o o
         2 2 2 2
          2 2 2
           2 2
            2
```

FullBoard illustration



```
        1
       1 1
      1 1 1
     1 1 1 1
    o o o o o
   o o o o o o
  o o o o o o o
 o o o o o o o o
o o o o o o o o o
 o o o o o o o o
  o o o o o o o
   o o o o o o
    o o o o o
     2 2 2 2
      2 2 2
       2 2
        2
```

Heuristic FullBoard illustration



```
   1
  1 1
 o o o
o o o o
 o o o
  2 2
   2
```

Heuristic Smallboard illustration

Figure 1: Starting Boards.

2 Method

The methodology of the AI agent is a game search tree that best mimics the behavior of a human player while demonstrates super-human performance by taking advantage of the computing power. With each node representing a board state of the game, and each edge representing one possible move from one board state to a subsequent board state, the game search tree can emulate the thinking process of a human player. Chinese checker is a two-player zero-sum game, thus an objective "score" is needed to judge the situation on the board. Player 1's goal is to maximax the "score", while Player 2 minimizes it. Such setting leads to the use the Minimax tree, which is a decision tree that minimizes the possible loss for a worst case scenario resulted from the opponent's optimal move.

The state space complexity of Chinese checker is 10^{23} (Bell 2009), which is unrealistic to build a top down game search tree. Instead, a shallow k-depth MiniMax game tree that searches only the tip of the tree is utilized. At each node, the "score" is taken as the "MiniMax score" which is computed by the MiniMax algorithm of depth k. When the search has reached the bottom of the k-depth search tree, a.k.a. the leaves, the score there is approximated by a raw score, which is a linear evaluation function based on the features of the board state. We exploits 8 features that are based on the positions of pieces on the board, which are described as the following:

- A_i: the squared sum of the distances to the destination corner of each pieces of player i;
- B_i: the squared sum of distance to the vertical central line of each pieces of player i;
- C_i: the variance(how scatted) of pieces of player i;
- D_i: the maximum vertical advance for all pieces of player i;

with $i = 1, 2$. Note that we use the square of the distances to penalize the outliers, which would motivate each player to make her pieces cohesive and thus enhance hopping. The evaluation function is the form of

$$E = a(A_2 - A_1) + b(B_2 - B_1) + c(C_2 - C_1) + d(D_1 - D_2)$$

where the weights $w = (a, b, c, d)^T$ would be trained via function approximation and Q-learning.

3 Implementation and results

We implement our algorithm using Python. Currently most of our codes are tested and implemented only on the heuristic small board. The following is a detailed description of how we implement methods in the previous section.

3.1 Minimax search tree and alpha-beta pruning

A detailed description of alpha-beta pruning can be seen in Liang (2015b). In order to expedite the alpha-beta pruning, at each level, we put all the legal moves of the player in a priority queue where the priority is in the order of the raw score of next board if the corresponding move is taken. In Table 1, we list the number of nodes visited as well as time used for the starting board (the heuristic FullBoard) in 1 to compute the Minimax value. Here as we can see, after applying alpha-beta tuning, the time used as well

	Without alpha-beta pruning		With alpha-beta pruning	
depths	nodes visited	time (s)	nodes visited	time (s)
2	196	4.07	27	0.85
3	4032	84.39	301	7.31
4	82944	1763	848	30.68

Table 1: Number of nodes visited and time used in computing the Minimax value

as the total number of nodes visited is significantly reduced.

3.2 Weights tuning and function approximation

An introduction of function approximation and Q-learning can be seen in Liang (2015a). Our weights-tuning algorithm is in Algorithm 1.

Algorithm 1 Tuning weights

```
1: Initialize a weights $w$;
2: **repeat**
3:     Play one game with both player following Minimax-rule using weights $w$, record the features as well as the Minimax score at each board state;
4:     $w \leftarrow LeastSquare(F, u)$ where $F$ is the matrix with each row contains the features at a board state, and $u$ is the vector containing the computed Minimax score at each board state;
5: **until** converged
```

We plot the R^2 (coefficient of determinant) in Figure 2. As we can see, even though the R^2 is not monotonically increasing, and it seems not stable, it usually remains in a high level, which means our method works generally. By looking at the weights at each gram, we find that such weights are not stable either. We will apply some methods such that our weights can stabilize and the R^2 can almost monotonically increase and plateau.

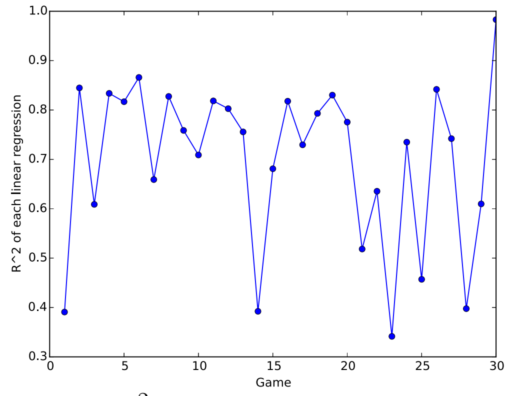


Figure 2: R^2 of the regression at each game.

4 Progress and Future Works

Up till now, we have finished the basic framework of our coding project, accomplished fully functional game rule. Given a weight, computer can play against each other following the Minimax rule. We have found a way to tune the weights, though it is not stable.

We may work on the following problems in the upcoming two weeks. (1) Try some other tuning methods to stabilize our weight update. (2) Apply our current algorithm and codes on full board to train the weights. (3) Build a UI so that human can play against computer.

References

Bell, George I. 2009. The shortest game of chinese checkers and related problems. *Integers* **9**(1) 17–39.

Liang, Percy. 2015a. Lecture 8: MDPs II. URL `http://web.stanford.edu/class/cs221/lectures/mdp2.pdf`.

Liang, Percy. 2015b. Lecture 9: Games I. URL `http://web.stanford.edu/class/cs221/lectures/games1.pdf`.