# Reinforcement Learning Agent for Playing Checkers

Hari Surayagari[1, a)] and Tom Peter[1, b)]
*Machine Learning - CS 545, Colorado State University*

(Dated: 15 December 2015)

In this project we will implement reinforcement learning to develop an agent that can learn to play checker after a certain number of games.

Keywords: Reinforcement Learning, Checkers, Temporal difference, Machine Learning

**Checkers is a well-known board game. Using machine learning to develop an algorithm that can learn games like Checkers, Chess, etc has always been a fascination for man. Since the development of modern computers there have been numerous algorithms that are capable of performing well against humans. However most algorithms are artificial intelligence based, in other words the algorithm mimics human intelligence without actually learning the game. In this paper, we will try applying reinforcement learning to develop an agent that will learn to play checkers. We will test its capability by letting the agent play an opponent which makes either random moves or moves selected by an AI.**

## I. INTRODUCTION

The main goal of this paper is to develop a reinforcement learning agent for checkers. It is well suited for the reinforcement learning approach since the checkers board can act as an observable state and the action can be the moves. Checkers seems like an easy enough game in first glance but the total available moves can add up to a gigantic number. A checkers search space can end up with as many as $5 \times 10^{20}$ moves [4]. This number is extremely hard to compute even with the high performance capabilities of modern computers. One of the first applications of reinforcement learning for checkers was done by Arthur Samuel [5]. His approach was different, he used reinforcement learning in tandem with a minimax approach and his method included the reinforcement learning agent playing itself. His approach was a very basic representation of reinforcement learning, it had some potential flaws such as his method was not constrained to finding good evaluation functions, so it was possible for his agent to get worse with experience and his approach also had little to do with winning or losing the game as his reward update was done based on the result from minimax. He also didn't use an explicit reward function, this was similar to what we did.

[a)] Electronic mail: hari7988@colostate.edu
[b)] Electronic mail: tom.peter@colostate.edu

Temporal difference learning is a more specialised approach in reinforcement learning, using temporal difference it is possible to develop an algorithm that can learn the checkers game. In this project, we will try to implement checkers with a reinforcement learning agent in Python and try to demonstrate that the theoretical application of reinforcement learning is practical.

### A. Checkers

Checkers is a popular two player board game. There are various versions of Checkers however in this project we will be using 64 spaces board. Each player will have 12 pieces, placed in 3 rows closest to the player. In this version of checkers the pieces can only move space diagonally forward. The piece cannot move backward unless its a king piece. If there is an option to jump then the piece has to jump. Jump is when there is an opportunity to capture an opponent piece. When a piece reaches the opposition end of the board then that piece becomes a king piece and it can move backwards. Due to time constraints and increased complexity of coding a king piece, we decided to not include a king in this project.

The main goal of this game is to eliminate all the opponent's pieces.

### B. Reinforcement Learning

This is relatively new field of machine learning. Learning from interactions is one of the primary underlying ideas for almost all the concepts of learning and intelligence. Reinforcement learning is computational approach for such a learning method. Reinforcement Learning is a more focused on goal-directed learning interaction than other approaches of machine learning. In this approach an agent is dropped into a situation where it is not informed about the actions to take. The agent takes an action randomly and a reward is given based on how good its action is. Eventually the agent learns which the good actions are and which the bad ones are. *"Reinforcement Learning is defined not by characterizing learning methods, but by characterizing a learning problem."* [3].

In this project, we use a more specific sub topic of reinforcement learning which is called Temporal Difference Learning or TD learning.

### C. Temporal Difference Learning

Temporal Difference Learning is a combinations of Monte Carlo method and Dynamic Programming. We wont be going into the specifics of those methods. But briefly, Monte Carlo method enables an agent to learn directly from raw experience without much knowledge of the environment. Dynamic Programmings estimate updates are in part based on other learning estimates. We use an off-policy TD approach called Q-learning for this project. Q learning can be expressed as:

$$Q(s_t, a_t) < -Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (1)$$

$Q(s_t, a_t)$ = Q value for the given state and the action taken
$\alpha$ = Learning Rate
$\gamma$ = Discount Factor
r = Reward

This approach helps converge quickly and dramatically decreasing the amount of computation.

## II. METHOD USED

The primary task for this project was to define the checkers board and its moves. Then we had to understand how the Q table was update.

### A. Checkers Definitions

First we defined the checkers board. We created a 8x8 board, with the spaces marked as "0" for black pieces, "1" for white and "-1" for empty spaces. Next we had to write a code to check for legal pieces. This is elaborate as the checkers board is complicated. We also had to create an AI (whose importance will be explained later). The AI was developed with using a mini-max approach incorporating alpha-beta pruning to decrease computation. The minimax approach was first introducted by Shannon [6] in 1950. Minimax is a search tree where each depth level is a ply. A ply is a play of one player. If its black's turn to play, one of it's move will be the root. The child nodes of the root will be white's possible moves and its child nodes (depth = 2) will have black's moves and so on. An evaluation function was needed to evaluate the quality of the board at each node of the search tree. This was selected using:

$$V(s) = (7 * X_1) + (5 * X_2) + (3 * X_3) \quad (2)$$

V(s) = Value of state or board
$X_1$ = Number of pieces in the opponent's final third of the board
$X_2$ = Number of pieces in the middle of the board
$X_3$ = Number of pieces in the agent's third of the board

This evaluation function helps give a value for the board state. The reason for taking this equation is that it forces the pieces to move forward. We also had a mandatory jump option. When it is checking for legal moves, it also checks for possible jump actions. If there is such an action then it has to jump. Ones the checker game was coded, we had to build a reinforcement learning agent.

### B. Agent

When we say the agent learns, we mean that a dictionary is created that stores the board state, the action taken and it's corresponding reward. When a board state is reached, the algorithm looks for a action (or move) associated with that board state in the Q table. If such as state is not present then it does a random move. Unfortunately given the number of possible states (as mentioned above) for a reasonably learned agent we need to play a lot of games. For simplicity and faster convergence our Q-learning function was slightly different from the regular one. Our Discount Factor was 1 and Learning Rate was set to 0.2 and the reward was added to the final Q value instead of having a separate "r" variable. So our modified function was:

$$Q(s_t, a_t) < -Q(s_t, a_t) + \alpha[Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (3)$$

Theoretically this should help with faster convergence.

### C. Algorithm

We used the following algorithm:
1. Check for legal moves
2. Check for moves in Q dictionary.
3. If move not found or its value is less than 0, run AI or select random move
4. Store the move and the board state prior to move in Q dictionary.
5. Update the Q dictionary
6. Check if game over, if not - repeat 1-5
7. If game over: reward = 1 if agent won, -1 for defeat and 0 for a draw.
8. Repeat until the max games is reached.

## III. RESULTS

We selected black to be the agent for all the runs.
Random Vs Agent:
- Total games played : 10000

- Number of wins by black :4719
- Number of wins by white 4699
- Number of draws : 582 draws

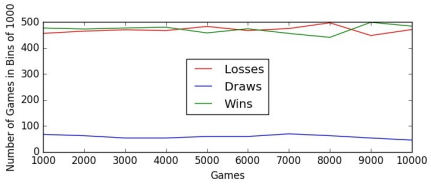


Figure 1. 10,000 Plays Random Vs Agent

When we played 10,000 games with a random agent, we realised that the randomness of the moves will add quite a few useless moves into the Q table. Due to this the learning could take a lot of games and 10,000 was nowhere enough. To counter this we decided to use an AI player as well because this would trim the number of moves to just the quality ones. This way the total number of moves generated would be lesser compared to the random moves.

AI Vs Agent:

- Total games played : 5000
- Number of wins by black :2411
- Number of wins by white : 2287
- Number of draws : 302

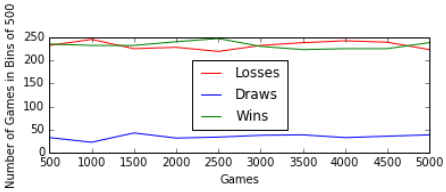


Figure 2. 5,000 Plays AI Vs Agent

The idea of using an AI player was interesting but we ended up facing another issue which was that the run time with AI was lot longer than that without it. To run a 1000 games it took almost twice the time to run 10,000 games without AI. Even with AI and 5000 games, the result doesn't seem to be a great deal better than before. The reason for so many draws is because we hadn't used the king piece concept in checkers. This meant that when a piece ended up on the opposite end of the board, it had nowhere else to go and it would get stuck. If atleast 1 piece from either side ended up on opposite ends, then there was no way from them to fight each other and hence the game ends up in a draw. This was another compromise we had to make to keep up with the time constraints.

Realistically a reasonable learning starts to show after 150,000 - 200,000 games. Due to time constraints this wasn't feasible. Although the approach was fairly decent, the agent doesn't seem to learn at a acceptable rate. We tried modifying the algorithms to help with faster learning but it didn't really help.

## IV. CONCLUSION

As per the results generated above, it is observed that the number of wins by black (learning agent) is slightly more than white which shows this checkers learning agent seems to have learned how to slightly tower over the given opponent. The involvement of AI to help the agent learn faster didn't particularly help as the computation time with AI was more than it was without it. Checkers has a search space of $5\times10^{20}$, this number indicates the requirement of a large number of Q values for the algorithm to function properly. Given the time constraints and computation capabilities, generating enough Q values has not been feasible.

Given more time and perhaps more depth of search might have produced a more prominent learning.

## V. FUTURE WORK

The primary purpose of this project was to explore the numerous possibilities of reinforcement learning and understanding the complexities of checkers. Although the project didn't yield concrete proof of learning, maybe increasing the depth of search and decrease the number of Q values required could provide a better learning. This will hopefully indicate and clear learning by the agent.

Although this specific project didn't give concrete results, it is apparent that reinforcement learning is an approach worth pursuing for such applications.

## VI. REFERENCES

1. Ghori I, "Reinforcement Learning in Board Games", Dept of Computer Science, University of Bristol, report CTR-04-004 2004

2. Marco A.W, Jan Peter P, Henk Mannen, "Learning to Play Board Games using Temporal Diference Methods", Institute of Information and computing sciences, Utrecht University, Report UU-CS-2005-048

3. Richard S Sutton and Andrew G Barto, "Reinforcement Learning: An Introduction" MIT Press 1998

4. Schaeffer, Lake, Lu, Bryant, Treloar. Chinook.University of Alberta, Edmonton, Alberta, Canada http://www.cs.ualberta.ca/ chinook/, last visited 12/10/2015

5. Samuel, A.L.(1959), "Some Studies in Machine Learning Using the Game of Checkers", IBM Journal of Research and Development, 3 (3), pp. 210-229.

6. Shannon, C.E.(1950), "Programming a Computer for Playing Chess", Philosophical Magazine, 41 (4), pp. 256-275.

7. Anderson C, Colorado State University, http://nbviewer.ipython.org/url/www.cs.colostate.edu/ anderson/cs545/notebooks/19last visited 12/15/2015