Deep Reinforcement Learning Othello

Thesis subtitle

Master Thesis

Thomas Steinmann

University of Bern

Month and Year

Abstract

Abstract (max. 1 page)
Name of the Supervisor, Group, Institute, University, Supervisor
Name of the Assistant, Group, Institute, University, Assistant

Contents

1 Introduction

With recent successes such as AlphaGo defeating the reigning world champion and fast progress towards fully autonomous cars by Tesla as well as Weymo it is clear that Reinforcement Learning is the solution to many problems that could not even be tackled before. Many of these powerful implementations rely on equally powerful machines in order to train them, often requiring over hundred CPUs and GPUs for weeks on end. Inspired by these grand achievements and the technology behind them but lacking comparable resources I settled on a more achievable goal: Othello. This simple board game has accompanied me since the first semester when we were tasked to implement a search based Othello player. During my masters studies I suspected that a superior player could be created using machine learning techniques. This thesis documents the way to such a player and its key components.

2
Related Works

2.1 Othello

According to [4] Othello is a game played on a square board, usually made up of eight by eight tiles. The opening position is shown in Figure 2.1 Othello stones are black on one side and white on the other.

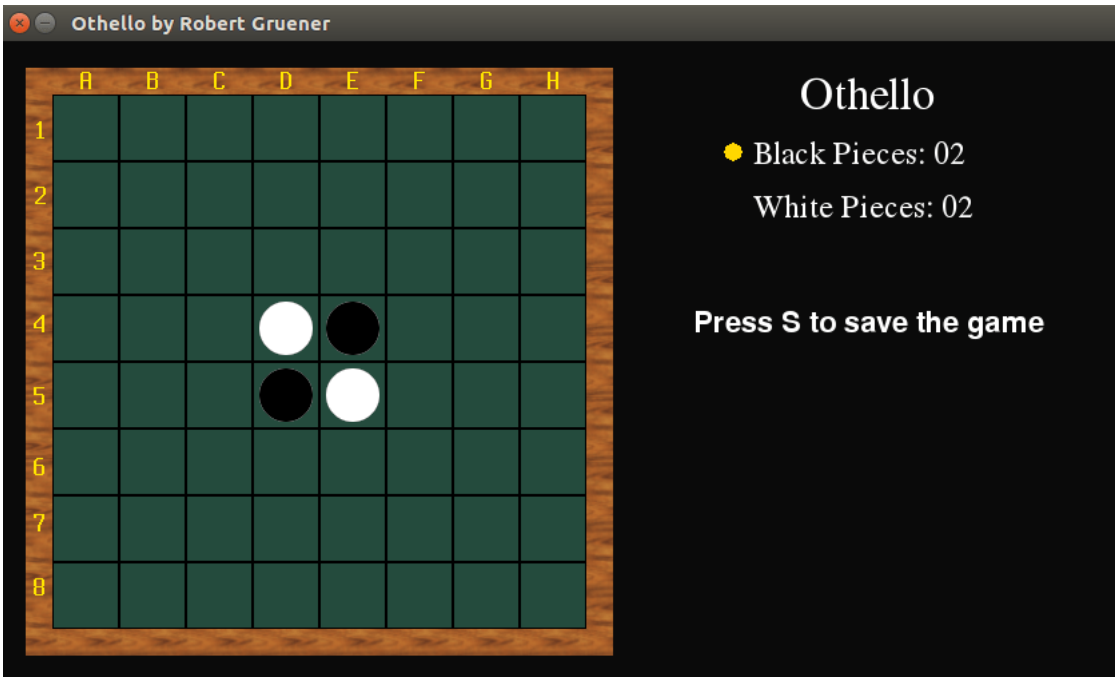

Figure 2.1: The opening position

Players take turns in placing one of these stones in their respective color on the board. Stones can only be placed in such a way that the new stone *traps* one or more of the opponents stones inbetween itself and any other stone of the same color. All opposing stones that were trapped by placing the new stone are flipped and now belong to the player who trapped them. If a player cannot perform a legal move he simply passed and his opponent places the next stone. The game ends if both players pass successively or there are no free tiles left on the board. The player who controls more stones at this time wins the game.

2.2 Heuristic Player

The player we call the *Heuristic* Player in this thesis is often used for benchmarking because of its decent performance, low computational cost and deterministic nature. It utilises a heuristic table which assigns a value to each tile on the board. These values are symmetrical for eight axes through the board. The table gives high values to tiles such as the edges that are of high value because they lead to stones that cannot be captured later in the game. Tiles around the edges however receive a low value because occupying one of them may allow the opponent to occupy the adjacent corner.

2.3 Search based Algorithms

Othello is widely used to teach search based game theory algorithms, most notably the min-max algorithm and its optimization the alpha-beta pruning. [5] The general idea of these methods is to build a search tree of all relevant moves for both players and the resulting game states. Ideally a complete graph for such a game could be computed and a computer player would just choose a move that results in a win for every single turn. However the search space for most games is so big that such a graph is not feasible with the computational resources available. The ancient chinese game of Go for example is said to have more possible game states than there are atoms in the universe. The challenge lies therefore in approximating a complete graph as close as possible and with maximal efficiency. For this the search tree is usually truncated at a certain depth or after a given time has passed and the rest of the tree is approximated with a so called value function that evaluates the value of the state where the tree stopped. This value function traditionally consists of a set of given features such as possible moves for each player, save stones that cannot be captured again or just greedily the number of stones a player controls. Search based algorithms can also be used to improve the *Heuristic Player* described in section 2.2. In most search based algorithms this would make the player nondeterministic however.

2.4 Machine Learning based Algorithms

Machine learning techniques for Othello are related to search based algorithms but often do not perform a search at play time. A straight forward approach is to inprove a search based algorithm by learning its value function. [1] used machine learning to learn his own heuristics function, similar to the one used by the *Heuristic Player*, in his alpha-beta search algorithm. A more advanced technique basically moves the search from play time to training time by simulating thousands or millions of games and uses them to train an agent. This agent learns a value function that captures the knowledge gained during the search and predicts the value of a possible move without having to perform another search at play time. Such an approach was used by [2] to master Go. They went one step further still and used training time search in order to train an agent and then improved that agents decision again with another search at play time.

2.5 Reinforcement Learning

Reinforcement Learning levarages special algorithms to train an agent on a task while performing it. In regular intervals the agent receives feedback, reinforcing behaviour that leads to good results while discouraging less performant ones. This is handled by awarding a positive or negative reward respectively, whenever the algorithm can determine something good or bad happened based on the agent's actions.

2.5.1 Monte Carlo Learning

Monte Carlo Learning is the most basic variant of a group of reinforcement learning algorithms called Monte Carlo Methods. In contrast to many other reinforcement learning algorithms they do not require full knowledge of the environment that they are applied o. Instead they work on experience alone. This experience can be gathered in the real environment and learning can therefore be done on the fly, without any prior knowledge. Another very powerful approach is to simulate the environment which allows for simpler aggregation of training data that would be difficult to obtain in the real world, such as driving cars, steering rockets or playing millions of board game iterations on a physical board.

In Monte Carlo Methods a task can be split in two major parts. First the reinforcement algorithm which is generating training data from real or simulated experience and an according label. Second a supervised learning task where an agent is trained on the data generated from the first part. Those two parts are mostly independent of each other, as long as the structure of in and outputs match. This means that the same reinforcement algorithm can be used with a variety of different learning agents such as, in the case of this thesis, multiple different neural networks.

The basic Monte Carlo Algorithm simply simulates an entire episode, evaluates the final state and labels each training sample with the resulting label. In a board game this would mean playing a single game, computing the winner and labeling the game state for each time step with the final result.

2.5.2 Temporal Difference Learning

The Temporal Difference algorithm is a variant of the basic Monte Carlo algorithm decribed above. In contrast to its predecessor it does not assign the same label to each game state but rather computes each label based on the prediction of the subsequent state's value. This can be better understood when analysing the algorithm from the end of an episode. The final label is still the result of the game just like in the basic Monte Carlo algorithm. The labels of all other states n however are computed based on the value predictions of states n and n+1. This allows for a so called online version of the algorithm which does not have to complete an entire episode before it can learn from the gathered experience, but can apply changes at every time step of the episode. However the online version has weaker convergence guarantees and since standard Othello episodes have a maximum length of 30 game states per player there is a guaranteed reward in relatively few time steps which makes offline Temporal Difference Learning more suitable for this thesis.

2.5.3 Monte Carlo Tree Search

3 Thesis Objectives

The objectives of this thesis are threefold: *Build a framework for a reinforcement learning othello player*, *Build and train a RL othello player* and *experiment with different optimizations with the goal of beating a set of predefined search based players*.

3.1 Framework

The framework is responsible for simulating one or several games of othello automatically given two players as well as documentig the results. It therefore contains and enforces the game rules and provides an interface in order to interact with two players regardles of their implementation and strategy. It also computes and plots statistics of results over time in order to give insight into how an agent behaves during training.

3.2 Othello Player

The othello player is at the core of the othello framework. It can represent any kind of strategy from one step heuristic evaluations to intricately learned agents using neural networks, search algorithms etc. In the case of a machine learning player this is where training data is aquired and the agent is trained.

3.3 Optimization

Once the first iteration of the framework and a reinforcement player are working, the goal shifts towards optimizing all parts of their implementation. This includes performance in regards to computation time for the framework as well as in regards to winning rate for the players.

1. Framwork for othello agents
2. Playground for RL algorithms, network optimizations, regularizations, etc.
3. High performance Othello Agent

4 Implementation

4.1 Framework

The framework is an integral part of the application and makes up the largest part of the code base. As a base we forked [3], a python othello framework that allowed both human and computer players to compete against each other. The computer player was implemented using an alpha-beta search and evaluating positions on the basis of several hand crafted heuristics. It included a graphical user interface for interacting with the games in progres.
Since the framework was intended for a human to play or at least spectate the games being played, some major changes had to be made for automated play, such as the introduction of a headless mode and support for running thousands of games in rapid succession. For this purpose the core framework was wrapped in an additional layer of scripts that control the different use cases such as evaluation and several different training algorithms. In order to analize training and evaluation games, a Plotter was introduced, accumulating data and representing them in a human readable form using several plots.

— UML? —

4.2 Othello Player

The Player is the origin of all the different strategies. It is an interface between an agent and the othello framework. For machine learning agents this is where most of the actual learning algorithm is implemented. First however, we introduced several non learning players as a baseline. A *RandomPlayer* as well as a search based *ComputerPlayer* were also available from the framework. We introduced three more: a *HeuristicPlayer*, a *MobilityPlayer* and a *SaveStonesPlayer*

4.2.1 HeuristicPlayer

The HeuristicPlayer uses a Heuristic Table of tile values for its value function. It then alots points to the player holding these tiles according to the table. This rather simple player is often used as a baseline because it is fast to execute, deterministic and easy to understand and debug. The Heuristic Table gives this

100	-25	10	5	5	10	-25	100
-25	-25	2	2	2	2	-25	-25
10	2	5	1	1	5	2	10
5	2	1	2	2	1	2	5
5	2	1	2	2	1	2	5
10	2	5	1	1	5	2	10
-25	-25	2	2	2	2	-25	-25
100	-25	10	5	5	10	-25	100

Table 4.1: The Heuristic Table of *HeuristicPlayer*

player its strategy. We used table 4.1 as found in [6] as our strategy for the *HeuristicPlayer*. By looking at the values alotted to each tile one can see that the corners are regarded as the most important tiles on the board, therefore whenever the *HeuristicPlayer* has the chance to take a corner, he will. Additionally the tiles around a corner have negative values. This prevents the player from taking those tiles because doing so would allow its opponent to take the corner. Further more the table is symmetrical by eight axes which guarantees that rotations and reflections of a game state do not change its value.

4.2.2 ComputerPlayers

The *ComputerPlayer* consists of two main parts: the alpha-beta cut algorithm that simulates multiple moves ahead of the current game state [5], and the value function that scores each game state in order to determine its value. The original player available in the framework used following features and weighted them depending on how far the game has progressed:

- **Stone count**
  The number of stones each player controls. This is most important in the endgame since it determines the winner of the game.
- **Corner count**
  The number of edges each player controls. This is most important in the midgame since corner stones cannot be taken by the opponent.
- **Mobility**
  The number of valid moves each player has. Restricting the opponents possibilities may interfere with the opponents strategy and allow the player to take many stones at a time in the endgame.
- **Edge count**
  The number of edges each player controls. Edge stones are valuable because they are harder to overturn and may lead to winning a corner. They are most valuable early and midgame, depending on the game state.
- **Corner edge count**
  The number of stones directly surrounding an empty corner each player controls. These stones are valued negatively since they may allow the opponent to occupy a corner.
- **Stability count**
  The number of stones connected to a corner through stones of the same color. These stones are protected from four sides, three from the edge of the board and one from the corner stone or a number of other stones that are connected to the edge stone, and therefore cannot be taken by the opponent.

We created two additional players for more variety and to assess if our agents can learn the concepts behind them:

- **MobilityPlayer**
  A player implementing a strategy based purely on maximizing the mobility in the early and midgame. As soon as the end of the game can be simulated the stones count is considered instead.
- **SaveStonesPlayer**
  A player maximizes its own save stones and minimizes its opponent's. A save stone is any stone that cannot be taken by the opponent for the rest of the game. The computation of save stones starts at a corner stone which is the first save stone. The number of save stones can then be expanded along one of the two edges adjacent to the corner. Each stone along the edge and directly adjacent to a save stone is also save as it is protected from four sides. The same can then be applied for the next row where every stone that is protected by four save stones becomes a save stone as well. Just with any other player in the endgame only the stone count is considered for scoring.

4.2.3 ReinforcementLearningPlayers

4.3 Optimization

One of the main difficulties in applying reinforcement learning to Othello is the very delayed and sparse reward. The only evaluation the learning algorithm can make is the result of the game, every other measure would introduce prior knowledge and therefore influence the strategy that is learned by the player. This constrains the learning algorithm to a single reward per game, rather than one for every action taken like for example in a jump and run game.

5
Validation

6
Conclusion

7
Future Work

Bibliography

[1] Kevin Anthony Charry. An intelligent othello combining machine learning and game specific heuristics. Master's thesis, Louisiana State University, 2008.

[2] Demis Hassabis et al. Mastering the game of go without human knowledge. *Nature*, 2017.

[3] Othello by rgruener. https://github.com/rgruener/othello.

[4] British othello federation. `http://www.britishothello.org.uk/rules.html`.

[5] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, third edition, 2009.

[6] Michiel van der Ree and Marco Wiering. Reinforcement learning in the game of othello: Learning against a fixed opponent and learning from self-play.