Project 3: Decision Trees for Expert Iteration

Francisco Parrilla Andrade 1900939

Abstract—Reinforcement learning has been a promising field for the gaming industry. Games such as Go, OXO, Chess rely on intelligent agents that are able to generate the best action when playing against a player. Monte Carlo Tree Search algorithm has been a base line to do an initial exploration of the different possibilities and then take an action. In this work, using MCTS algorithm as initial data generator, a decision tree classifier will be trained,tested and evaluate the performance of it, when taking an action for a player in the game Tic-Tac-Toe.

1 Introduction

Building intelligent games that can approximately reach a human-mind decision making criteria is something that in the past was seen as impossible. Over the past years, researchers have dedicated time on finding how to optimally create agents that are able to mimic human behaviour when playing a game. For example, on board games such as OXO, GO, and Othello, Monte Carlo Tree Search algorithm has been a solution to explore every possible combination from a current game state, and take an action based on that. However, depending on the complexity and type of game, the tree search algorithm can get extensive. A solution has been to use a neural network, by collecting each action the player took at each stage of the game and train a neural network with that data, to guide the tree search. However, a simpler version of this approach can be done using decision trees as a classifier, where it will read the state of the game, and it will predict an action, presenting a possibility for intelligent agents to not rely on neural networks that might cause a long training time. The following sections of this report are: Background Literature Survey, Methodology, Results, Discussion and Conclusion.

2 Background

2.1 Monte Carlo Tree Search Algorithms

Monte-Carlo Tree Search algorithm, also known as MCTS, is an algorithm that builds a search tree iteratively, the process stops until a computational constraint is reached, for example, number of iterations, time, memory, and performs an action based on root performance [1]. Each iteration has four steps: selection, expansion, simulation and backpropagation. MTCS are considered an effective approach to find an optimal action based on a game state, up to a point that MCTS have been proved to increase performance in games such as Pac-Man, Go [2].

2.2 Related Work

MCTS have given opportunity to researchers to use it as an early approach for systems to play by themselves and then use another method to learn [3]. AlphaGo Zero is an example that used MCTS for an initial training phase, where it was trained by self-play reinforcement learning using the MCTS, where it took random actions in the game Go [3]. A key comparison is made between AlphaGo Zero and its predecessors, which is that previous systems rely on human data for their initial phase, where as AlphaGo Zero uses MCTS instead [3]. [3] approach was to use a deep neural network, that took the board representation and its history as state values, and MCTS was used a policy operator to decide the move [3]. AlphaGo Zero was tested against different predecessor, for example AlphaGo Master, which used human data, and the score finished 60-0 in favour to AlphaGo Zero, proving that a reinforcement learning approach is viable and not dependent on human data or knowledge [3].. For a more detailed explanation of the methodology refer to [3]. At the same time, [4] presented ExIT (Expert Iteration), based on the premise that a decision-making problem requires a planning policy and a generalization of it combined. A tree search was used for the planning of new policies, more specifically MCTS, each move was simulated 10,000 with MCTS [4]. After, a convolutional neural network was used to predict an action based on the data collect with MCTS algorithm. For a more detailed explanation refer to [4].

More recently, [5] applied MCTS algorithm in rehabilitation game that focus on alleviating the physical impairment of their limbs caused by stroke or brain injuries by presenting objects closed to character. The game makes use of kinetic devices to measure the movements by the user (muscle, orientation and joint activity data) , and depending of the measurement of the movement, the MCTS is used to determine the difficulty of the game, which will determine the distance and amount of objects presented to the user [5]. It was found that this was a rehabilitation game that adapted to the physical state of the user, making it useful and playable by different type of users (with different percent of physical impairment) [5].

On the other hand, [2] presented an approach that uses heuristics, MCTS approach and deep reinforcement learning for training an agent to play Chinese Checkers. Considering that there is no definite amount of checkers that can be in the game, and the tree search performed by MCTS does not decrease overtime, compared to games like Go, and since a piece in the board can be moved more than once, the upper bound on how deep the search tree is not define [2]. At the end, experiments showed that the agent reached a human player experience, for further details refer to [2].

3 Methodology

This work will analyse whether using a supervised learning algorithm can improve the performance of a game agent instead of only using MCTS for finding the best action.The code provided by [6] uses MCTS for performing the action at each stage, however, for the OXO game, all moves where performed even if a player won, therefore, a slight variation of the code is used. The code to generated and collect data for task 1, and the corresponding data can be found here[1].

This work will focus on the OXO game, so the final goal is to see if by collecting the actions taken by the MCTS over multiple games of OXO, and training a series of classifier with that data, the agents (player 1 and player 2) improve eventually after repeating this process multiple times.

Several decision trees will be trained, and the goal is to eventually form a decision tree that outperforms his predecessors,i.e. decision tree classifier 10 will outperform the decision tree classifier 1, but would not necessarily outperform decision tree 9. Therefore, the methods used for this project would be to gather data of 300 games, where each game can have 5 to 9 instances, and record the board state values, the player who took the action, and the action it took, and repeat as many times as possible to see if a trained decision tree classifier actually helps an agent to win, as it collects data generated from previous classifiers.

The code provided represents the OXO board as an array of length 9, where index 0 is the top left box, 1 is the box space next to it, and so on. At the beginning, the OXO board with the following configuration:

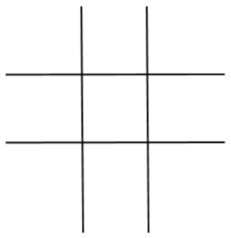


Fig. 1: Initial state of the board at start of the game

Is represented as:

$$0, 0, 0, 0, 0, 0, 0, 0, 0$$

and at each stage where a player takes an action, represented from 0-8, the array will be filled with either 1 (player 1) or 2 (player 2), so if player 1 puts an X in the middle box of the board, such as the configuration of the board is the following: the board state values will change to:

$$0, 0, 0, 0, 1, 0, 0, 0, 0$$

At the end of each game, the values inside the array will be a representation of where each player put their X or O.

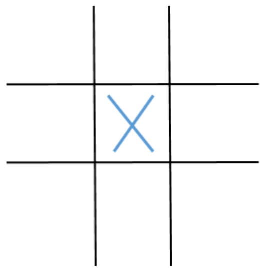


Fig. 2: State of the board after first action

For the first iteration of collecting data (task 1), the steps are the following:

1) Play the 300 games of OXO game, using only the action taken by MCTS.
2) Record the state of the board, which player took the action, and the action the player took at each stage of the game. And then merge them in a single line.
3) Save the data from 300 games into a separate file.

The state of the board will be represented as explained before, for example, at the start of the game where player 1 has the first move, and by using MCTS, the action is to put an X in the middle box, the collected data format for that instance will be the following:

$$0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 4$$

Where the last two numbers represent which player took the action and where did they put it respectively, in this case player 1 put a X in the middle box. And the following instance, assuming that player 2 decides to put a O in the top left box, will be the following:

$$0, 0, 0, 0, 1, 0, 0, 0, 0, 2, 0$$

The pipeline (Figure. 6) for this project would be the following:

1) Collect initial data using only the MCTS as action taker, i.e. the moves taken by the player will be random and will not take into context the board state.
2) Train a decision tree classifier with the new data.
3) Instead of purely using MCTS as action taker, 90% of the time it will used the classifier prediction and 10% of the time it will used MCTS. Collect data using the same format and save collected data.
4) Repeat steps 2 and 3, for 10 times.
5) The last classifier will be allocated to take actions for player 1, and player 2 will use classifier 1-9 to take actions (individually).
6) Change player 2 classifier and repeat previous step.
7) Record how many times player 1 wins the games.

1. https://github.com/fjpa121197/DataScienceDM/tree/master/Project

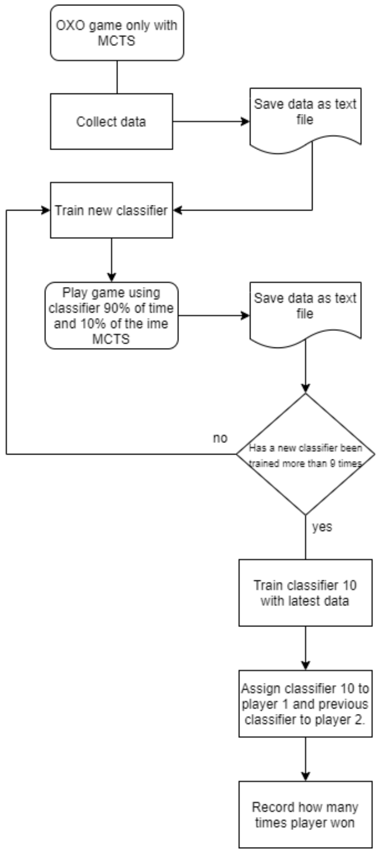
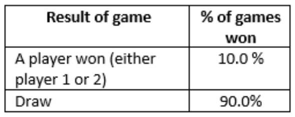

OXO game only with MCTS
Collect data
Save data as text file
Train new classifier
Play game using classifier 90% of time and 10% of the ime MCTS
Save data as text file
no
Has a new classifier been trained more than 9 times
yes
Train classifier 10 with latest data
Assign classifier 10 to player 1 and previous classifier to player 2.
Record how many times player won

Fig. 3: Pipeline of the experiment

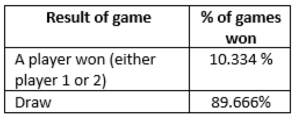
4 Results

While generating the first data set, beside capturing the board state values, the player and its action, it was recorded how many times a player won over the 300 games. These were the results of three different runs:



Result of game	% of games won
A player won (either player 1 or 2)	10.0 %
Draw	90.0%

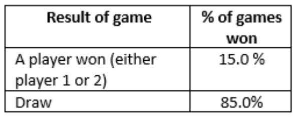
Fig. 4: Run 1 of using only MCTS for decision taking



Result of game	% of games won
A player won (either player 1 or 2)	10.334 %
Draw	89.666%

Fig. 5: Run 2 of using only MCTS for decision taking

When running the code provided, most of the times, the final result of a game ends up in a draw, looking at the previous three runs (300 games each), the vast majority of the time the game is a draw. With those results presented, the following hypothesis are presented:



Result of game	% of games won
A player won (either player 1 or 2)	15.0 %
Draw	85.0%

Fig. 6: Run 3 of using only MCTS for decision taking

- H0: Using a trained classifier to take an action, instead of using MCTS, makes no difference in the amount of times a player wins.
- H1: Using a trained classifier to take an action, in this case a decision tree, and iterating multiple times, it improves the decision making of the player,therefore, the player 1 would win have a better performance than player 2.

Looking at these results, it is clear that taking an action using MCTS, does not guarantee a win. Which gives the opportunity to replace the MCTS as decision maker, and try a different approach, in the case of this project, using a decision tree classifier.

5 Discussion

In order to prove or refute the hypothesis previously defined, a permutation test, with significance level of 5%, will be done in order to see if the new schema for taking decisions is better than the current option (using MCTS).

At the end, 20,000 games will be played. Half of those games will be played using only the MCTS as action taker, and every time player 1 wins, a value of 1 will be collected, and if not, 0 will be collected. The remaining games will be played using the last trained classifier, and it would take the actions of player 1, while player 2 will use the MCTS. If player 1 wins, a value of 1 will be collected, and if not a value of 0 will be collected.

6 Conclusion

In retrospect, only the first task was done. The following tasks might be challenging, specially keeping track of the different classifiers and make a comparison between them. However, if done properly, it might be possible to implement a intelligent agent that wins based on previous games data, without making use of any neural network, but instead, making use of a decision tree as a classifier to determine the best move by a player.

References

[1] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, “A survey of monte carlo tree search methods,” *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[2] Z. Liu, M. Zhou, W. Cao, Q. Qu, H. W. F. Yeung, and V. Y. Y. Chung, “Towards understanding chinese checkers with heuristics, monte carlo tree search, and deep reinforcement learning,” *arXiv preprint arXiv:1903.01747*, 2019.

[3] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, “Mastering the game of go without human knowledge,” *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[4] T. Anthony, Z. Tian, and D. Barber, “Thinking fast and slow with deep learning and tree search,” in *Advances in Neural Information Processing Systems*, 2017, pp. 5360–5370.

[5] S. S. Esfahlani and G. Wilson, “Development of rehabilitation system (rehabgame) through monte-carlo tree search algorithm,” *arXiv preprint arXiv:1804.10381*, 2018.

[6] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, “Monte-carlo tree search: A new framework for game ai.” in *AIIDE*, 2008.