Decision Trees for Expert Iteration (Reinforcement learning)

Gurkirat Sarna

Abstract—Reinforcement Learning requires an algorithm which can feed on the prior knowledge consisting of the best information and minimum errors. We describe the combination of Imitation Learning and Reinforcement Learning using the Expert Iterations (ExIT) by playing a game of OXO. Here we generate the dataset using the UCT policy with Monte Carlo simulations at each state of the game. This dataset is created using the imitation learning and will be used to train a Decision Tree classifier. We will change the expert policy of learning from UCT to 90% predictions from the Decision Tree classifier and 10% randomness of the game and perform expert iterations to retain the knowledge to update the apprentice policy.

Index Terms—Expert Iteration (ExIT), Exploration, Exploitation, Monte Carlo, OXO.

1 Introduction

THERE are two natural phenomenon of a human thought process; one is intuitive and the other is reasoning. Reasoning is built by learning on the actions and rewards returned on those actions. While playing a game a human brain applies both a heuristic process and the rule-based learning. We will apply the same concept while making an agent learn the OXO game, that is considering the agent initially knows nothing, *tabula rasa*, it gradually learns from the expert all the possible moves of a game.

The games like Chess, Checkers and Othello which fall under the category of 'games with perfect information', that is all the players have the same information about the game status at any given point, we apply the min-max algorithm to train an agent on the rewards returned. However, the other games with imperfect information like Poker, Bridge, OXO and Skat have some hidden information about the game status and we cannot apply the min-max algorithm to explore the next best move. These game usually have too many possible branches at a given state so we apply the Monte Carlo (MC) simulations during the implicit learning to get the best move at that game state. For now this paper describes the generating of the dataset using implicit learning, we will discuss about the expert iterations in the Discussion section as the future prospects, thereby combining the implicit learning with reinforcement learning.

2 Background

Thomas Anthony1, Zheng Tian1, and David Barber [1] have used the Expert Iterations (ExIT) motivated by the dual human thought process and applied it to the game of Hex. They executed the Monte carlo Tree Search (MCTS) algorithm to get the most promising next move at any given root node. Root node here is the state of the actual game after which a new move is yet to be decided.

• *G. Sarna is with the Department of Mathematics, University of Essex, Colcheseter, UK, CO43ZE.*
E-mail: gs19979@essex.ac.uk

David Silver et al. [2] suggests the self-play reinforcement learning approach using MCTS at every node of the tree which depicts a state of a game. He applied the UCT algorithm to the imperfect information game of Go.

Jan Schafer [3] talks about optimization of UCT algorithm using the Monte Carlo Policy by playing XSkat game.

To extend the research we will be using the UCT algorithm with MCTS simulations for OXO game to define the expert policy and use the expert iterations to train the apprentice policy.

3 Methodology

3.1 Monte Carlo Tree Search

Monte Carlo Tree Search is a heuristic search algorithm which models the probability of different outcomes. It repeats the game simulations at a given state to estimate the value of the states before expanding the tree. It consists of 4 stages :

- Selection : Starting from the root node R, the current state of the game, select the successive child nodes until a leaf node is found using a *tree policy*.
- Expansion : From the leaf node, expand the tree with child nodes consisting of the possible valid moves until a game ends in a win/lose/draw for any player.
- Rollout : Complete one random playout from any of the child node using some *default policy*.
- Backpropogation : Use the result of the playout to update information (wins/values) in the nodes on the path back from the child node to the root node.

Each node of the tree depicts a state, lets say s1 and s2. The edge joining the two state represents the action taken from s1 to s2 and denoted by (s1,a). At each node the number of iterations is stored as n(s) and the edge stores the number of times it has been traversed n(s,a) and the sum of all rewards r(s,a) obtained during the simulations.

We use the Upper Confidence Bound for Tree (UCT) policy to decide the best move at that current state, it uses the formula :

$$UCT_{(s,a)} = \frac{r(s,a)}{n(s,a)} + c_b \sqrt{\frac{log(n(s))}{n(s,a)}}$$

3.2 Exploitation and Exploration

Knowing the best move and choosing it always as the next move is called exploitation and picking moves which were not explored taking into consideration that the best move has not been discovered yet, the UCT policy maintains a balance between these two concepts. For example, we have a multi-armed bandit problem, with fixed number of gambling machines (K) with different unknown rewards. Each machine is played n times giving a reward of X. Choosing the machine with the highest known reward may not result in the highest reward overall because there is a possibility that the highest rewarding machine has not yet been discovered. The first term of the UCT policy handles the exploitation and the second term handles the exploration concepts.

3.3 Expert Iteration

Expert Iteration (ExIT) is used to make the agent learn from its own moves and store the learning. At each iteration i, the algorithm creates a set of states Si playing with a policy pidash (apprentice policy). At each state s of Si a policy pistar (expert policy) imitates the game in itself and learns a new target at s. We train the pidash using the outcomes of the expert policy and then we use this newly trained policy to update our expert policy in the next iteration.

In our example of the OXO game play, we have currently created a dataset using the expert policy with the outcomes at each stage s. The expert policy has been calculated using the Monte Carlo tree search algorithm. We will use this dataset for expert iterations as mentioned in the 'Discussion' section.

The algorithm for the Expert Iteration is provided here [1].

3.4 OXO Game Dataset Generation

The OXO game board can be imagined as a matrix of 9 positions. Each matrix if written as a one dimension vector will contain values of 0,1, or 2, where 1 depicts the movement of player 1 at that position, 2 depicts the movement of player 2 at that position and 0 depicts an empty cell. The game is played using the UCT algorithm with 5000 iterations. Each iteration returns set Si of 1-D vectors describing each stage s of the board and the next best move that the agent took at that stage. At each stage s monte carlo simulations are done to calculate the wins and the visits. The maximum visits at a state s is picked up as the next best move.

If the game is won by any player the set Si will not have all possible states of the game until it was won and will have a 'NaN' as the next best move at the winning state. The Set Si always starts at a state of all 0's depicting the initial state of the board. When the Si is returned it will return maximum of 10 states where no player won the game.

We combine the Si of the 5000 iterations and remove the winning state as it gives us no information of the next best move. This final dataset will be used to train the Decision tree and perform the expert iterations.

4 Results

We performed the Monte Carlo iterations by keeping a bias of giving one player more iterations than the other. This was done to make one of the player win more so that the other player could learn form the moves as a part of implicit learning. We performed 1000:2000, 1500:2000 iterations but the no player won the game. We increased the bias of letting one player iterate 10 times and the other 1000 times and played 2000 games of this setting. Nearly 2.5% games were won. However when the bias was kept as 10:1000 and 5000 games played there was an increase to 12.5% games being won. This means there were more learning done implicitly to win a game using the best possible moves. In other words UCT agent is learning on the intuition of the best moves possible by another player.

5 Discussion

The final file generated above will be used for expert iterations to train an apprentice policy. A Decision Tree classifier will be trained on the above dataset using the position of the board as the features and the next best move as the output variable. The 0's act as NULL values and With the combinations of 1's and 2's we will predict the next best move.

We will change the expert learning policy from depending on UCT itself to learning from the outcomes of Decision Tree classifier. The classifier will help in 90% imitation learning and the rest will be on random outcomes. N games will be played to learn through the new expert policy. Each nth decision tree classifier will be played against the (n-1) decision trees outcomes to realise the number of winnings.

We will repeat the aforementioned task to see how better does our algorithm learn by retaining the knowledge of the previous imitations. These expert iterations should help the agent win more games by the end of the experiment.

6 Conclusion

The 'CE888_DataToBeProcessed_1900690_6261Wins.csv' file is intended to serve as a "starter file" for conducting the experiments suggested in the discussion section of this thesis.

References

[1] 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.

[2] J. Schäfer, M. Buro, and K. Hartmann, "The uct algorithm applied to games with imperfect information," *Diploma, Otto-Von-Guericke Univ. Magdeburg, Magdeburg, Germany*, vol. 11, 2008.

[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.

7 Plan

Following is the Gantt Chart to show our current and future progress of the experiment.

Decision Trees for Expert Iteration

University of Essex

Project Start Date: 2/5/2020 (Wednesday)
Project Lead: Gurkirat Sarna
Display Week: 8

WBS	TASK	LEAD	PREDECESSOR	START	END	DAYS	% DONE	WORK DAYS
1	**Decision Trees for Expert Iteration**				-			-
1	Generating the data and submission	Gurkirat S	NA	Wed 2/05/20	Fri 2/21/20	17	100%	13
2	Train the Decision Tree classifier	Gurkirat S	NA	Mon 3/23/20	Wed 3/25/20	3	0%	3
3	Modify code and generate the new data based on results from Decision Tree classifier (90%) and randomization (10%). Repetition of this process to generate data.	Gurkirat S	NA	Thu 3/26/20	Wed 4/01/20	7	0%	5
4	Comparing the various Decision Tree learning against each other	Gurkirat S	NA	Thu 4/02/20	Wed 4/08/20	7	0%	5
5	Generate the report for the above	Gurkirat S	NA	Thu 4/09/20	Sat 4/18/20	10	0%	7