Expert Iterations using Decision Trees for Tic-Tac-Toe (OXO) Game

Gurkirat Sarna

Abstract—The game learning agent's performance is based upon the knowledge provided to it on which it can train itself and learn to respond with the best move when an unseen state of the game occurs. To test the agent's self learning in an OXO game, we have used Expert Iterations (ExIT) with an expert improvement step in every iteration to train a Decision Tree models using board states of 5000 biased OXO games. The data set consisted of board states with the best move evaluated by UCT with Monte Carlo Tree Search. The expert learnt from random choice of movements, however this was changed to exploit a learnt Decision Tree classifier 90% of the times and explore the next available move randomly 10% of the times. We discuss about the variations introduced in the learning pattern when an unbiased and a biased training data is provided for Expert Iterations. In both the cases, a single player showed an extensive winning initially but gradually the 'No win' situation of the game became prominent, providing insights to learning by the agent. The initially losing player started to learn from the winning player and counteracted to make sure no one wins. The research can be a used in future to include variations in the exploitation and exploration factor of the expert policy to fully train a policy that can always lead to winning an OXO Game.

Index Terms—Expert Iteration (ExIT), Decision Tree, Monte Carlo (MCTS), OXO Game, UCT.

1 Introduction

There are two natural phenomenon of a human thought process applied during a game play; one is reasoning and the other is intuitive. Reasoning is built by learning on the actions and rewards returned on those actions, thereby creating an expert policy. The intuitive thought process is dependent on this expert policy and is improved as and when the expert policy is updated, we call this an apprentice policy. We applied the same concepts to help an agent self learn the OXO game, considering that initially the agent knew nothing, *tabula rasa*, about the best move at a given state of the game. It learnt by creating an expert policy on the basis of 5000 OXO game board states and the player moving from those states to gradually update its apprentice policy to predict the best move by the player.

We implemented the classic algorithm of policy iteration [1] [2] by alternating between *policy evaluation* and *policy improvement* which generated a sequence of improving policies, that is, we used the value function of the current policy to generate a better policy until the agent improved. The value function used on the sample outcomes of a single OXO game for evaluation was UCB1 and the improvement policy was updated to select 90% times the move based on the trained Decision Tree classifier and randomly otherwise.

In the Classification based reinforcement learning [3] that we followed constitutes of many rollout execution during the Monte Carlo search. Bruno Scherrer *et al.* [4] described a modified policy iteration with evaluating the policy by regressing the value function and achieved a then state-of-the-art results for the game of Tetris. David Silver *et al.* [5] used MCTS for both policy improvement and its evaluation. The policy improvement was based on a neural network which executed the MCTS based on the policy's recommendation to project a stronger search policy back into the neural network. Policy evaluation was then applied to the improved search policy which reflected the outcomes of the self-game play (Imitation Learning) back into the neural-network space. Considering the similar concept with positive outcomes, we worked on training the Decision Tree instead of the Neural Netwwork for the OXO game.

Self-play reinforcement learning have been also applied to games like Go. The NeuroGo [6] [7] used a neural network to represent the value function to train the apprentice policy, however a weak level of play was achieved. On the positive side, high levels of performance was achieved in other games like : Chess [8] [9], Othello [10] and many more. These games used a regression value function and have given positive results for self-play learning.

In Section 3 we explain the methodology used in our research with its background in Section 2. Section 4 talks about the results of our research by giving explanation and limitations in Section 5. We conclude our research with suggested improvements in Section 6.

2 Background

Thomas Anthony1, Zheng Tian1, and David Barber [11] 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.

David Silver et al. [5] suggested the self-play reinforcement learning approach using MCTS at every node of the tree which depicted a state of a game. He applied the UCT algorithm to the perfect information game of Go to

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

demonstrate that a pure reinforcement learning approach is possible to obtain a superhuman level trained agent without the human intervention or guidance. This approach has defeated the previous version of AlphaGo which were trained on the human provided data. Motivated by this approach, we have created the states from a self play game of OXO to have the agent learn on it.

The Upper Confidence Bound for Tree (UCT) Algorithm [12] has been applied to many imperfect and perfect information games, with research on it's optimization by Jan Schafer [13] while using the Monte Carlo Policy in XSkat game. We apply the UCT Algorithm with Monte Carlo Tree Search where the moves are sampled at each node according to UCB1 formula as follows :

$$UCT_{(s,a)} = \frac{w_{(s,a)}}{v_{(s,a)}} + \sqrt{\frac{2 * log(v_{(s)})}{v_{(s,a)}}} \quad (1)$$

The $UCT_{(s,a)}$ is the value at the state *s* for taking action *a* on that state. The $w_{(s,a)}$ and $v_{(s,a)}$ is the number of wins and visits respectively of ending up in the state *s* by taking action *a*. $v_{(s)}$ is the total visits an agent makes to this state irrespective of the possible action.

2.1 Monte Carlo Tree Search

Monte Carlo Tree Search is a heuristic search algorithm which models the probability of different outcomes. It creates game simulations from a given state by imitating the actual game for a self play to estimate the value of the states before expanding the tree. OXO game usually has too many possible branches at a given state so we apply the Monte Carlo (MC) simulations to get the best move at that state. It consists of 4 stages :

- Selection : Starting from the root node, which is considered as the current state of the game, it selects the successive child nodes until a leaf node is found.
- Expansion : From the leaf node, it expands the tree with corresponding child nodes depicting 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.
- 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.

Monte Carlo Tree Search has been used to create an automated Hex player by Broderick Arneson *et al* [14]. In our research we updated the rollout default policy from random generating next move to a move predicted by a trained Decision Tree. In a case where Decision tree predicted an unavailable move, the default policy of randomness was set into play.

2.2 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 of exploitation and exploration. 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 in equation 1 handles the exploitation and the second term handles the exploration concepts.

2.3 Expert Iteration and Implicit Learning

Expert Iteration (ExIT) is an iterative process of Imitation Learning with an included expert improvement step. During the self play rollout phase of Monte Carlo search the agent imitates the actual game and improves it's expert policy to return the best move at that state. The apprentice policy learns from the expert however in the next iteration it adds up an improvement step which can be an initial bias in the search algorithm.

In our research, we have created an initial data set using the expert policy of randomly choosing the next move and evaluating it based on its weight. The improvement step is based on the decision tree classifier modelled on the games played in the previous iteration. We will use this data set for expert iterations as mentioned in Section 3.

3 Methodology

3.1 Technical Adjustments and Options for the User

This section explains the available adjustments for the user. The data sets and settings used for our research are explained in the subsequent sections.

3.1.1 Biased and Unbiased Data Creation

Creation of a biased or unbiased data set can be done by editing the *itermax* (hardcoded value) option of the game. The *itermax* option controls the iterations in the rollout phase of Monte Carlo Tree Search for both the players. A player provided with more iterations will have a better expert policy as it would have analysed more available moves at a given state.

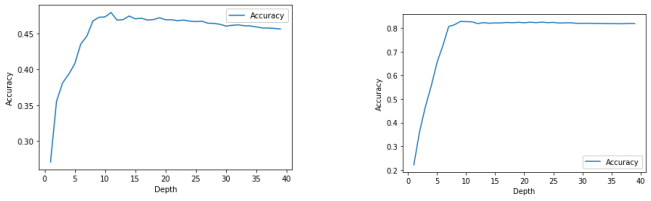
3.1.2 Random Data for Initial Execution

An already existing data set of biased or unbiased games can be used to execute the whole project provided the data is in the format as mentioned in Section 3.2. However, if no data set is available, the project creates the initial unbiased data set in the required format and continues to assess the agent performance. The creation of the initial random data depends on the number of games that a user inputs as mentioned in the Section 3.1.4.

3.1.3 Decision Tree Classifier's Maximum Depth

The Decision Tree classifier modelled on the initial data set is checked for accuracy with respect to tree depth in range of 0 to 39. The depth at which maximum accuracy is reached at the earliest is used to model classifiers on the other data sets. The maximum depth where accuracy was the highest for each data set could vary which may lead to over-fitting and

hence was kept constant for the whole research as per the initial data set. *Figure 1* shows the *Depth* (x-axis) vs *Accuracy* (y-axis) graph for a user provided biased and unbiased initial data set. The graph of accuracy for a classifier trained on biased games has maximum accuracy of 47.92% at tree depth of 11, whereas, the accuracy for a classifier trained on unbiased games has maximum accuracy of 82.81% at a depth of 9. The accuracy is based on the 30% testing states of the data set..



(a) (b)

Fig. 1. (a), (b) Shows the *Depth* vs *Accuracy* (divided by 100) graph of the user provided initial data set on which the maximum depth is decided for all the Decision Tree classifiers generated in this research. (a) corresponds to biased data of 5000 games [15] , whereas, (b) corresponds to unbiased data of 5000 games [16].

3.1.4 Number of Games *N* and Number of Data-sets *Y*

We have created an initial data set [15] of biased 5000 games by changing the *itermax* value as mentioned in Section 3.1.1 to 10 and 1000 for Player 1 and Player 2 respectively. The second data set [16] of unbiased 5000 games had the *itermax* value set at 1000 for both players. The number of games to be played for the recurring data set creations depends on the user input *N*. We should note that in the case of randomly generated unbiased initial data the number of games played is same as the number of games provided by the user.

Furthermore, the user is asked to provide the total number of data sets *Y* (multiple of 10) that have to be created, where each data set contains the states of the *N* number of OXO games along with the player playing. In the case of providing the initial data set, the value *Y* provided by the user for the total number of data sets to be created is reduced by one to maintain the total count.

3.2 OXO Game Data set Generation and it's Structure

The OXO game board can be imagined as a matrix of 9 positions where player 1 is always associated with *X* and player 2 with *O*. Each matrix is written as a one dimensional vector will contain values of 0,1 or 2, where 1 corresponds to the move by player 1 at that position, 2 depicts the move of player 2 at that position and 0 corresponds to an empty cell available for a move by player 1 or player 2. To exemplify, Figure 2 shows a random state of the OXO board which will be represented in the data set as a row of *[0,0,2,1,2,0,0,1,1]*.

Looking at the current OXO board example provided in Figure 2, the next move is of player 2 and the best move it would take is at position 6 *(Note : The data in a one dimensional vector is indexed starting from 0)* and win the game. Hence, the row in the data set is appended with the player playing and the next best move the agent will take from that state. Therefore, the data will look like *[0,0,2,1,2,0,0,1,1,2,6]*. From here the next row in the data set will correspond to the

(a)

Fig. 2. A random OXO Board state

updated board state with player 2 updated at index 6. However, as that will be a winning state for player 2, the player playing and the next best move in this vector will be recorded as 'NaN'. Every time a new game starts, the state board is represented by all zeroes with value 1 representing player 1 and the move the player had taken. All states of *N* games were combined together to create one single data set on which the classifier is trained. Also, the states with 'NaN' were removed as they do not provide any learning of the next best move to the Decision Tree classifier at that state.

We have worked with two data sets described as below:

1) User provided biased states of 5000 games [15].
2) User provided unbiased states of 5000 games [16].

3.3 Best Move Decision

In Section 3.2 we described how the initial data set for the research was structured. The initial data set consists of the states with randomly picked best move by the player on the basis of the expert policy. At each stage the Monte Carlo simulation rollouts a tree like structure describing the next available move for the player from that particular state. These randomly generated moves are recorded as child nodes from that state and are updated with winning weights every time a node is visited. The winning weights are described as below :

1) Weight 1 represents the current player winning
2) Weight 0.5 represents the other player winning
3) Weight 0 represents a No win situation

The node with the highest weight is returned as the next best move.

3.4 Decision Tree Classifier

A Decision Tree classifier is modelled with *Gini* criterion on the data set and is stored for evaluation purposes later as explained in Section 3.6. The Decision tree maximum depth is calculated as described in Section 3.1.3. The classifier is trained on 70% of the data set and tested on the rest 30%. The classifier predicted the best move for a given unseen OXO board state and the player about to move.

3.5 Expert Iterations

The initial data set was created using UCT [12] with Monte Carlo Tree Search Algorithm [17] and a decision tree classifier was modelled on this data set to predict the best move for the player from that state.

To improve the expert policy and thereby the apprentice policy and vice versa, we created the next data set of *N*

games by predicting the move using the classifier trained on the previously generated data set. For a given unseen OXO board state, 90% of the times the decision tree classifier predicted the next best move contributing to the exploitation aspect of the reinforcement learning. However, 10% of the states were subjected to a randomly chosen best move to retain the exploration [18] factor of the learning.

Sometimes, at a given state, the move predicted by the classifier was not valid, that is, the predicted move had already been played by one of the players. The best move was randomly chosen when MCTS encountered such board state. And these states with the player playing and the randomly chosen best move were provided as input to the next decision tree classifier for the training purposes. These states were considered as the exploring factor which helped in improving the expert policy and hence the apprentice policy.

3.6 Game Play and Evaluation

We created 100 (Y) data sets for 100 (N) games, where N was 5000 for user provided initial data set only. Each data set contained the OXO game board state with the player playing from that state and the best move taken by the player from that state. Also, each data set has a corresponding Decision Tree classifier trained on it.

We evaluated the learning of the agent by organising 9 unbiased tournaments of OXO game between every 10^{th} classifier with its previous 9 classifier versions and calculating the number of wins of the 10^{th} classifier against its previous version classifiers. During the Game play, player 1 moves were predicted by every 10^{th} Decision Tree classifier and the moves of player 2 were predicted by the previous 9 versions of this classifier. For example, 9 tournaments were held between 1^{st}, 2^{nd}, 3^{rd}, 4^{th}, 5^{th}, 6^{th}, 7^{th}, 8^{th} and 9^{th} Decision Tree Classifier (trained on their respective data sets) versus the 10^{th} Decision Tree Classifier (trained on the 10^{th} data set which included the most improved expert policy) and we calculated the winnings of 10^{th} Decision tree classifier against its rivals. Such tournaments were held between the 20^{th} Decision Tree Classifier with its previous 9 version (11^{th} until 19^{th}). The total winnings of every 10^{th} classifier were calculated in expectations of having more winnings favored to the 10^{th} classifier everytime as it was always the most learnt classifier.

4 Results

4.1 Initial Biased Data set

The user provided initial data set [15] of 5000 games was used to train the 1^{st} Decision Tree classifier. Execution of 5000 games took near and above 30 minutes and creating 100 data sets of 5000 games was not time efficient. So we created 100 data sets, including the initial data set, where 99 data sets, excluding the initial data set, were a combination of 100 unbiased (*itermax* value as 100 for both players) game states with the best next move evaluated by the improved expert policy. The maximum depth of the classifiers were maintained at 11 depending on the 47.92% maximum accuracy of the 1^{st} classifier (Refer Figure 1).

As explained in 3.6, nine tournaments were held between every 10^{th} classifier and its previous versions. The Figure 3 shows the result of each tournament. On the result board of every tournament *DT10* corresponds to the total winning of every 10^{th} classifier, i.e. in our case it corresponds to winnings of 10^{th}, 20^{th}, 30^{th}, 40^{th}, 50^{th}, 60^{th}, 70^{th}, 80^{th}, 90^{th} and 100^{th} classifier, whereas, *Other DTs* corresponds to the total winnings of the previous versions of the corresponding 10^{th} classifier, i.e during tournament 1 *Other DTs* corresponds to the total winnings of classifier 1^{st} until 9^{th}; in tournament 2 *Other DTs* corresponds to the total winnings of classifier 11^{th} until 19^{th} and so on.

```
Tournament  1  Wins :
 DT10 :  6  Other DTs :  0  No wins :  3
Tournament  2  Wins :
 DT10 :  6  Other DTs :  0  No wins :  3
Tournament  3  Wins :
 DT10 :  5  Other DTs :  1  No wins :  3
Tournament  4  Wins :
 DT10 :  7  Other DTs :  1  No wins :  1
Tournament  5  Wins :
 DT10 :  5  Other DTs :  0  No wins :  4
Tournament  6  Wins :
 DT10 :  6  Other DTs :  0  No wins :  3
Tournament  7  Wins :
 DT10 :  4  Other DTs :  1  No wins :  4
Tournament  8  Wins :
 DT10 :  2  Other DTs :  0  No wins :  7
Tournament  9  Wins :
 DT10 :  5  Other DTs :  0  No wins :  4
Tournament  10  Wins :
 DT10 :  4  Other DTs :  1  No wins :  4
```

(a)

Fig. 3. Outcome of Tournaments with initial biased 5000 games

From the Tournament board in Figure 3 we can conclude that the Expert Iteration with improvement step using decision tree classifier helps the agent learn back and forth. That is, in the initial Tournaments even though the 10^{th} classifier was winning more as compared to its rivals, eventually the rival decision tree classifiers learnt to counter the winning of player 1. Even if player 2 did not win it learnt enough to block the winnings of player 1 and end the game in a 'No win' situation.

4.2 Initial Unbiased Data set

An unbiased data set [16] of 5000 games was used to train the 1^{st} classifier. Similar process to that explained in Section 4.1 was followed and total of 100 data sets, each of 100 unbiased games were created with their corresponding classifiers. As explained in **??** about the Tournament boards, Figure 4 describes the output of the nine tournaments between the classifiers. We have the similar output as compared to the output of Biased data set, that is player 2 did not win but it learnt to block the winnings of player 1 and end the game in a 'No win' situation.

5 Discussion

There are few limitations to this research that have negatively impacted the required expectations. Firstly, the initial data set consisted of 5000 games and the other 99 data sets consisted of only 100 games due to time constraints. This means even though we provided the most available states for the expert to learn initially, it would exhaust all the

```
Tournament  1  Wins :
 DT10 :  7  Other DTs :  0  No wins :  2
Tournament  2  Wins :
 DT10 :  7  Other DTs :  1  No wins :  1
Tournament  3  Wins :
 DT10 :  2  Other DTs :  3  No wins :  4
Tournament  4  Wins :
 DT10 :  6  Other DTs :  2  No wins :  1
Tournament  5  Wins :
 DT10 :  2  Other DTs :  1  No wins :  6
Tournament  6  Wins :
 DT10 :  3  Other DTs :  1  No wins :  5
Tournament  7  Wins :
 DT10 :  5  Other DTs :  0  No wins :  4
Tournament  8  Wins :
 DT10 :  5  Other DTs :  1  No wins :  3
Tournament  9  Wins :
 DT10 :  4  Other DTs :  0  No wins :  5
Tournament  10  Wins :
 DT10 :  4  Other DTs :  1  No wins :  4
```

(a)

Fig. 4. Outcome of Tournaments with initial unbiased 5000 games

possible available moves earlier in the next Expert Iterations and not exploit the learning.

Secondly, the *itermax* value was changed from 1000 (initial dat set) to 100 in the subsequent game plays, which means not many rollout games were played during imitation learning which describes the learning of *Other DTs* over the 10^{th} decision tree learning eventually.

After improving the expert step, there were states for which random decision were taken because Decision Tree predicted an already played position of the board. This also depreciated the improvement step as it led to more random movement decisions and was not contained under 10%. For this, we will need to have more available board states on which Expert Iterations can be performed.

However, on the positive side, we can see that the policies are improving for which means UCT agent is improving its intuition of the best move possible by feeding on the reasoning power of the other player. Which means that if the research is improved by taking into consideration the time efficiency and more available board states in every iteration, we could create a state-of-the-art OXO game player learnt from self play.

6 Conclusion

Even though we could not create a policy for an agent to win the game always but we realised that human intervention is not needed to make an agent learn. Self-play games can efficiently and effectively make the agent learn. More work needs to be done by updating the expert improvement step and by updating the value function of every node in Monte Carlo Tree Search algorithm.

References

[1] R. A. Howard, "Dynamic programming and markov processes." 1960.

[2] R. S. Sutton, A. G. Barto *et al.*, *Introduction to reinforcement learning*. MIT press Cambridge, 1998, vol. 135.

[3] M. G. Lagoudakis and R. Parr, "Reinforcement learning as classification: Leveraging modern classifiers," in *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, 2003, pp. 424–431.

[4] B. Scherrer, M. Ghavamzadeh, V. Gabillon, B. Lesner, and M. Geist, "Approximate modified policy iteration and its application to the game of tetris." *Journal of Machine Learning Research*, vol. 16, no. 49, pp. 1629–1676, 2015.

[5] 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.

[6] M. Enzenberger, "The integration of a priori knowledge into a go playing neural network," *URL: http://www. markus-enzenberger. de/neurogo. html*, 1996.

[7] H. J. van den Herik, H. Iida, and E. A. Heinz, *Advances in Computer Games: Many Games, Many Challenges*. Springer Science & Business Media, 2003, vol. 135.

[8] J. Baxter, A. Tridgell, and L. Weaver, "Learning to play chess using temporal differences," *Machine Learning*, vol. 40, no. 3, pp. 243–263, 2000.

[9] M. Lai, "Giraffe: Using deep reinforcement learning to play chess," 09 2015.

[10] M. Buro, "From simple features to sophisticated evaluation functions," in *International Conference on Computers and Games*. Springer, 1998, pp. 126–145.

[11] 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.

[12] S. Gelly and D. Silver, "Combining online and offline knowledge in uct," in *Proceedings of the 24th international conference on Machine learning*, 2007, pp. 273–280.

[13] 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.

[14] B. Arneson, R. B. Hayward, and P. Henderson, "Monte carlo tree search in hex," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 4, pp. 251–258, 2010.

[15] G. Sarna, "Ce888_datatobeprocessed_1900690_biased_final," 2020, [updated 2020 Apr 22; cited 2020 Apr 22].

[16] ——, "Ce888_datatobeprocessed_1900690_unbiased_final," 2020, [updated 2020 Apr 22; cited 2020 Apr 22].

[17] K.-W. Chang, A. Krishnamurthy, A. Agarwal, J. Langford, and H. Daumé III, "Learning to search better than your teacher," pp. 2058–2066, 2015.

[18] M. Tokic, "Adaptive ε-greedy exploration in reinforcement learning based on value differences," in *Annual Conference on Artificial Intelligence*. Springer, 2010, pp. 203–210.