Decision Trees for Expert Iteration (Reinforcenment Learning)

Dilpa Rao, Registration Number: 1906319

Abstract—Strong intuitive and reasoning skills are required when playing board games because games are like problems which requires logic, reasoning power and intuitive skills. With the technological advancement, board games can now be played online with virtual players and this has been made possible because of Artificial Intelligence. Artficial and Computational Intelligence in games have seen lot of advancement and success in recent years and so it has become a separate research field. The success can be contributed to the fact of using Reinforcement and Supervised Learning to create more complex levels and variations in the game to keep it challenging and engaging for the human player.The most successful approach when it comes to game programming is to use evaluation function by employing tree searches and this has been made possible by combination of Neural Networks (NNs) and Monte Carlo Tree Search (MCTS) in lot of games. In this project we will discuss the application of MCTS and a supervised learning algorithm; Decision tree classifier in a very simple game tic-tac-toe(OXO). The combination of MCTS and Decision Tree Classifier produced very interesting games. However, decision tree was not able to perform as efficiently as a Neural Network model would have even though they have faster performance speed and this area could use some research.

Index Terms—Convergence, Decision Tree, Expert Iteration, Imitation Learning, Monte Carlo Tree System, Neural Networks.

1 Introduction

When playing a game, the steps taken would result in some loss or rewards and these steps would ultimately determine the final results of a game because games are like sequential problems, where each step taken needs to be analysed in order to maximize the rewards or to win a game. The human reasoning comprises of two thinking system according to the dual-process theory. System 1 is more commonly referred to as intuition or heuristic process because it is fast, automatic and unconscious whereas System 2 is more more methodical and rule based because it is slow, explicit and conscious. When humans do any challenging tasks such as planning or playing board games, they use both of these processes. Their intuitions grow stronger with repeated study or practise leading to better analyzing skills and thus creating a learning loop. So human beings employ both reinforcement and supervised learning to master a problem or game by doing it repeatedly [1].
However, in game AI when using Reinforcement Learning algorithms, the neural networks only use the system 1. They make the selections without any lookahead or intuitive knowledge as they do not have a system 2. In this project work we will do Expert Iteration (Ex-It) on decision tree classifier which is analogous to System 2 to assist in improving the performance of the tree search [2]. Ex-It is an extension of Imitation Learning methods where the classifier which here in our case; decision tree is trained to imitate the expert policy which is MCTS and with each iteration an improvement step is done to improve the performance of the decision tree through Expert Iteration [3].

- *D. Rao is a MSc student in Data Science from the Univeristy of Essex.*

  E-mail: dr19597@essex.ac.uk

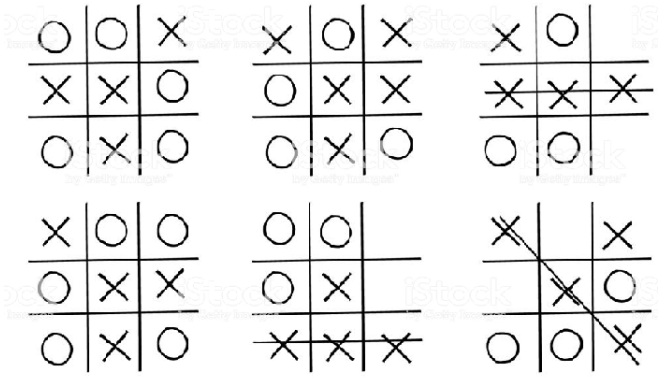
Manuscript received April 20, 2020.

Artificial Intelligence in games has seen lot of advancement in recent years and has been garnering more interest since DeepMind's AlphaGo; Google's Go-playing program defeated the world champion of the Go game which was more impressive than IBM's Deep Blue [4] victory over Chess GrandMaster Kasparov and IBM's Watson victory on Quiz show Jeopardy against champions Brad Rutter and Ken Jennings [5]. AlphaGo program is a combination of machine learning technique and MCTS [6].
Over the years,MCTS has been used in many non-deterministic games such as backgammon, poker and Scrabble. However, due to its stochasticity and branching capacity it can be used for deterministic games such as Go. Most of the present day's MCTS algotrithms for games use uniform sampling or some bias of action selection and this results in the poor performance by the agent [7].
This project work aims to develop a fast learning algorithm using reinforcement learning and supervised learning algorithm like decision tree for an OXO board game. Many Reinforcement learning algorithms does not provide strong policies as they take actions and make selections without a lookahead. Our objective is to train a Decision Tree Classifier using Expert Iteration which will use a tree search method and it will improve its performance with each iteration and perform an expert improvement step. The classifier will guide the MCTS and help to make the accurate move at each stage of the game [8].
The paper consists of a brief literature review of the tic-tac-toe game and MCTS which is in the Background section followed by generating the datasets and the agents in the Methodology section. The Experiment section deals with the evaluation metrics of the decision tree and the competition followed by the Results and finally the Conclusion section for concluding this paper.

2 BACKGROUND

2.1 Tic-Tac-Toe

The tic-tac-toe/OXO is a board game played by two players requiring paper and pens/pencils. The origin of this game could be traced back to first century BC. This game was normally played by children and was very popular. In 1952, a British computer scientist developed OXO at the University of Cambridge, and it became one of the first known video games which could be played perfectly between a human and a machine [9]. In this game there is a three by three grid which is initially empty. Each player makes a move by placing a marker in an open square. The first player generally marks a 'X' and the second player marks his move placing a '0'. The player who is successful in placing his marker in a row, column or diagonal becomes the winner [10].



Tic-tac-toe sample games

The figure given above demonstrates the different moves a player can make to optimize his win or a draw. A player needs to keep in mind the following:

- Win: If the first player already has two markers in a row,column or diagonal combination,he needs to put a third one in the pattern mentioned above to get a win.
- Block: If the second player has two markers in a row ,then the first player should put his marker on the empty grid on the row to block the second player from winning.
- Fork: When there are two ways which would allow a player to win, try to make that particular move.
- Blocking a fork All forks should be blocked by a player to prevent the other player from winning.
- Center: Usually, placing the marker at the center is the best preferred move but if the players are perfect then it wouldn't make much difference.

The player should try for a draw if winning the game is impossible.

2.2 Monte Carlo Tree Search

Most of AI games have been developed using Monte carlo Tree Search and Neural Networks [11].Monte Carlo Tree Search is one of the most popular Reinforcement Learning algorithm for many board games such as Go [12], Scotland Yard [13], Line of action, Hex, Chinese Checkers because it does not rely on domain knowledge instead it uses a stochastic simulations for guiding the search. The UCT (Upper Confidence Bound) uses the bandit based algorithm of exploration and exploitation, recursively to build the search tree. After building the game tree with a root node, it then expands the root node with random rollouts, at the same time keeping a count for the number of visits and wins of each node.The node which has the most wins is selected [5].

```
1: function MonteCarloPlanning(state)
2: repeat
3:     search(state, 0)
4: until Timeout
5: return bestAction(state,0)

6: function search(state, depth)
7: if Terminal(state) then return 0
8: if Leaf(state, d) then return Evaluate(state)
9: action := selectAction(state, depth)
10: (nextstate, reward) := simulateAction(state, action)
11: q := reward + γ search(nextstate, depth + 1)
12: UpdateValue(state, action, q, depth)
13: return q
```

The pseudo code of Monte Carlo algorithm

The algorithm consists of four phases:

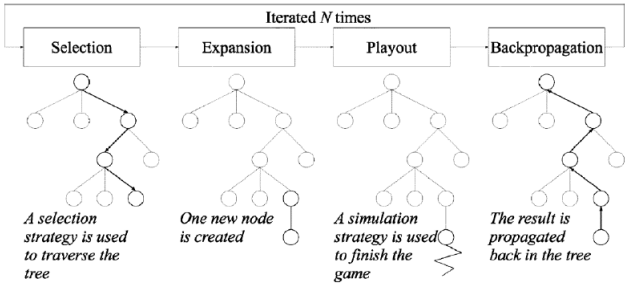
- Selection:
  The MCTS algorithm starts from a node which is generally the root node or state and traverses the tree downwards and selects a child node by using exploration and exploitation. The move with the best result is chosen in exploitation whereas in exploration, the algorithm explores for better results. The optimal child node is selected by the Upper Confidence Bound (UCB) formula. It calculates the UCB value of every child node and the one with the highest value is selected [7]

- Expansion:
  When the MCTS algorithm cannot find the successor node at the Selection phase, it adds a new child node randomly expanding the game tree.

- Simulation:
  The MCTS algorithm selects a child node randomly or by some domain policy, and it simulates or rolls out from the selected node to the end of the game to achieve the result.

- Backpropagation When the algorithm reaches the terminal state of the game, it evaluates and updates the tree by travelling upwards to the root and increments all the visited nodes. It also increments the win score for the node.

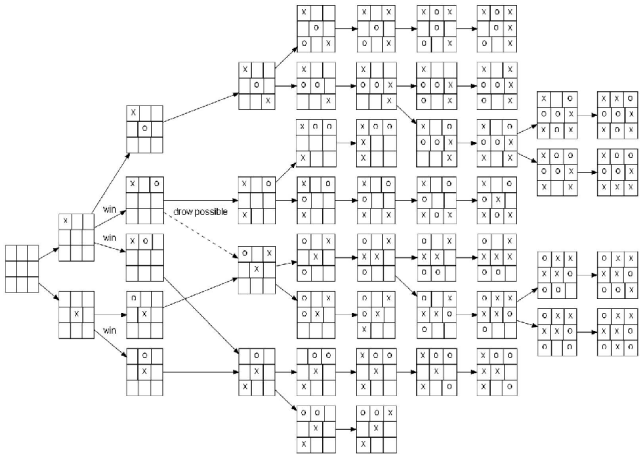
The MCTS algorithm reiterates these four phases for n number of iterations and estimates the results of each node and they improve with the number of iterations [14]



Monte Carlo Tree Search - tic-tac-toe implementation adapted from [13]

2.3 Decision Tree

A decision tree is a fundamental tool for data classification as data can be easily visualised in the flowchart of the tree structure. The node of the tree represents the attributes/features and leaf is the result/outcome. The first node at the top of the tree is the root node and the tree makes a split on the basis of the attribute value providing clear intuition as to which features are the most important, the split is performed in a recursive manner which helps in decision making. Expert Iteration (Ex-It) is a reinforcement learning algorithm in which the problem is decomposed into generalisation tasks and planning which is done by tree search. Imitation learning is akin to repetitive study to improve their intuitive abilities whereas expert improvement is akin to a human being using their improved intuitive abilities for further analysis.
In this project we will develop a Decision Tree Classifier (Ex-It) which is a simpler version of the neural network algorithm. The basic aim is that with Expert Iteration algorithm our classifier is improving, we will be able to solve problems like high variance and slow convergence and get stronger policies because our search is quick and accurate.



Decision tree - implementation of tic-tac-toe game

3 Methodology

3.1 Data Generation

The code for generating the datasets for one game was provided with the assignment. This code was modified and was used to generate additional data for 500 games of OXO. Since the OXO game is for 2 players, 2 MCTS agents played against each other and the best move was selected on a random basis. Both the players were allowed equal number of iterations to explore for the best move in order to avoid bias towards the players. The dataset consisted of 11 attributes, the first attribute being the player name; in our case indicated by 1 or 2, the next 9 columns represented the board of the game from 0 to 8 as tic-tac-toe is a 3X3 grid game, any move made by any of the two players must be between 0 and 8 in order to be a legal move and the final column was for storing the best move.
Whenever a player makes a move the position on the board from 0 to 8 will be filled with 1 for player 1 and 2 for player 2, an empty space is denoted by 0 indicating that the place is available to make a legal move by either of the player.
The data for the game is stored in the oxo-game.csv file consisting of approximately 4399 rows generated from 500 games.

3.2 Creating Agents

The main aim of this project was to build a fast classifier like Decision tree (Ex-It) that can make predictions of the best move, by using supervised learning techniques on the MCTS proposals. Initially the MCTS agents played games against each other to generate the data, this data was used to train a decision tree classifier.
The dataset was split in the ratio 80:20 repersenting the training and the testing set. The classifier was trained on the training set, a grid search was done for tuning the parameters to get the optimal hyperparameters to train the model and then it was used to predict the testing data. The table below shows the pickle file of all the 10 decision tree classifiers along with their accuracy score and kappa statistics.[14]

Pickle File Name	Datasets	Accuracy Score	Kappa Statistics
dt1.pickle	oxo-game.csv	73.86	0.71
dt2.pickle	data_1.csv	85.47	0.84
dt3.pickle	data_2.csv	79.89	0.77
dt4.pickle	data_3.csv	79.77	0.77
dt5.pickle	data_4.csv	79.66	0.77
dt6.pickle	data_5.csv	85.96	0.84
dt7.pickle	data_6.csv	81.36	0.79
dt8.pickle	data_7.csv	76.11	0.73
dt9.pickle	data_8.csv	78.16	0.75
dt10.pickle	data_9.csv	76.30	0.73
dt11.pickle	data_10.csv	78.53	0.76

Performance metrics of the agents within each classifier

A pickle file was generated for every one of the ten decision tree classifier and it was used to modify the rollout function of the MCTS. Subsequently, additional data of 100 games for every iterations of the decision tree classifier was generated by the MCTS which was guided by decision tree. Earlier during data generation, the MCTS did a random search but now with the decision tree classifier the search is more guided and intuitive. As a result the MCTS algorithm will perform much better, but if the decision tree algorithm performs poorly by not being able to learn properly and have insufficient domain knowledge, it might effect the performance of the MCTS and the MCTS will be unable to predict the best move.

4 Experiments And Results

4.1 Evaluation of classifier

As mentioned above the data which was generated by the MCTS was with 1000 itermax in order to allow it to explore and come up with the best move and then it was used for training the decision tree classifier. Additional data was generated for 10 more iterations of the decision tree with itermax parameter which allowed for more exploration resulting in better performance. For each iteration 100 games were played between the players 1 and players 2 even at this stage the itermax parameter was kept 1000 for both the in order to avoid bias towards a particular player and allow room for more exploration, based on game results their winnings and losses were recorded. This was done to show that with each iteration the classifier is learning and is able to predict more accurately. The graph below shows the comparison of all the 10 different classifiers with the draws, wins and losses of the respective players.

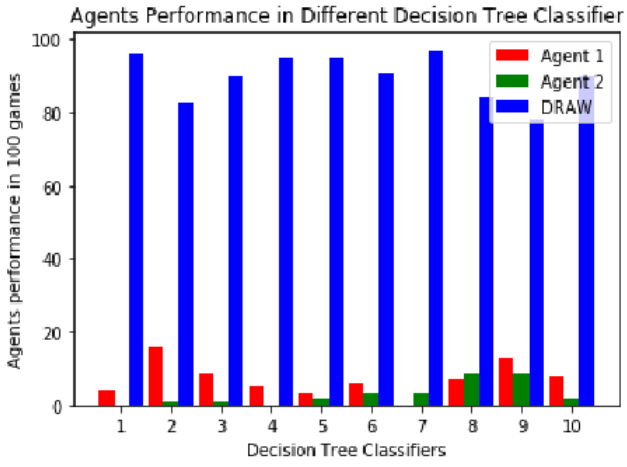


Fig. 1. Performance metrics of the agents within each classifier

4.2 Competition

In order to evaluate whether the decision tree is learning a competition is conducted which compares the current decision tree classifier with past self 10 classifiers and 100 games are played using these classifiers respectively to evaluate the results. For instance the player 1 will be using decision tree classifier 11 and player 2 with classifiers 2 to 10, the results proves that the classifier 11 is more learned and is able to make accurate moves which results in more winnings as comapred to classifier 1. Similarly, then player 1 will play against player 2 who will be using the classifier 2. We will observe the results for all the past 100 games with itself to see if the classifier has improved and has learned. With every 100 games the decision tree classifier is learning more and is improving. The figure below shows the results of comparison of the decision with its successive previous classifiers.

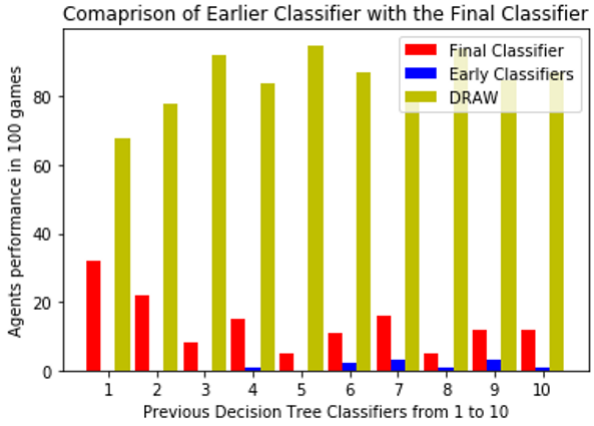


Fig. 2. Performance metrics of Different Decsion Tree classifiers

It is clearly evident from the figure above that the decision tree is learning. When we compared the decision tree classifier1 with that of the decision tree classifier 10, we could easily infer that classifier 1 is performing poorly than classifier 10, as we go from comparing classifier further towards the end we see that from the fifth classifier onwards its improving and that the classifier 10 has fewer wins as compared to the earlier ones.

5 Conclusion

We can conclude that the decision tree classifier (Ex-It) is learning and improving. The MCTS classifier performs well because based on the itermax values it is able to explore and exploit and select the best moves, whereas the decision tree has to make the selection within a few moves. Inspite of this the decision tree provides a very good visualization of the features which is more intuitive than artificial intelligence algorithms and neural networks. Moreover, it can be used in decision making problems and games as it selects the nodes which have a probabaility for winning. It is computationally cheaper and faster as compared to neural networks. However, in order to make more efficient classifiers Neural Networks models should be used for more challenging games and various areas regarding this should be explored.

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*, pages 5360–5370, 2017.

[2] D. Perez-Liebana, J. Liu, A. Khalifa, R. D. Gaina, J. Togelius, and S. M. Lucas. General video game ai: A multitrack framework for evaluating agents, games, and content generation algorithms. *IEEE Transactions on Games*, 11(3):195–214, 2019.

[3] S. Ross and D. Bagnell. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668, 2010.

[4] M. Campbell A. J. Hoane Jr and F. h. Hsu. Deep blue. *Artificial intelligence*, 134(1-2):57–83, 2002.

[5] M. Enzenberger, M. Müller, B. Arneson, and R. Segal. Fuego—an open-source framework for board games and go engine based on monte carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):259–270, 2010.

[6] M. C. Fu. Alphago and monte carlo tree search: The simulation optimization perspective. In *2016 Winter Simulation Conference (WSC)*, pages 659–670, 2016.

[7] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[8] J.Mehat and T. Cazenave. Monte-carlo tree search for general game playing. *Univ. Paris*, 8, 2008.

[9] M. JP Wolf. Encyclopedia of video games: The culture. *Technology, and Art of Gaming*, 2012.

[10] D. B. Fogel. Using evolutionary programing to create neural networks that are capable of playing tic-tac-toe. In *IEEE International Conference on Neural Networks*, pages 875–880 vol.2, 1993.

[11] 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*, 4(1):1–43, 2012.

[12] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[13] P. Nijssen and M. H. M. Winands. Monte carlo tree search for the hide-and-seek game scotland yard. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(4):282–294, 2012.

[14] H. Finnsson and Y. Björnsson. Simulation-based approach to general game playing. In *Aaai*, volume 8, pages 259–264, 2008.

Appendix:

The link to the Github repository https://github.com/Dilparao/Data-Science-and-Decision-Making.