# Decision Trees for Expert Iteration (Reinforcenment Learning)

Dilpa Rao, Registration Number: 1906319

***Abstract*—Abstract—Playing board games requires strong reasoning and intuitive skills. A game is like a sequential problem which needs a combination of good deal of reasoning and intuition in order to solve it. In this project we will generate a sample data set for an OXO game which is basically tic-tac-toe using the Monte Carlo Tree Search(MCTS) , a supervised learning algorithm like Decision Tree using Expert Iteration(Ex It) algorithm will be trained on this data-set generated by the MCTS and it will predict the best action to take.**

***Index Terms*—Convergence, Decision Tree, Expert Iteration, Monte Carlo Tree System**

## I. Introduction

Games are like sequential decision-making problems where each step results in some rewards or loss and the result is a win or loss based on the steps taken. According to the dual process theory human reasoning comprises of two processes namely System1 and System 2. System 1 is fast, unconscious and emotional whereas System 2 is more logical, effortful and slow. Human beings tend to use both of these processes which is nothing but supervised and reinforcement learning while solving a complex and uncertain problem or while playing games and with a skilful combination of these two systems they are able to learn or master a game over a period of time or by playing it multiple times. [1] [2]

The project is to develop a fast learning algorithm using reinforcement learning and supervised learning for an OXO board game, so it is basically a semi-supervised learning algorithm. In supervised learning data consists of input features and output labels. The data is pre-labelled in the form of certain features which gives information about the data whereas on the other hand, reinforcement learning is where the agent learns through trial and error and tries to maximize his rewards at each step. The agent learns to achieve a goal in an uncertain, potentially complex environment. A semi-supervised learning algorithm uses a supervised algorithm with ground truths as well as makes random moves without any guidance or domain knowledge beyond the game rules.

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 then be trained on supervised and reinforcement learning algorithms and it will be compared whether it has learned better and if it can predict better after n iterations.

## II. Background

The tic-tac-toe/OXO is a two player game generally played by children and has been around since 14th century. Each player take turns marking the spaces by X's or 0's in a 3 X 3 grid. The first player who places three of their marks in a row, column or diagonal is the winner. A player can choose a move from the following choices :

- Win: If a player has two marks in a row,column or diagonal combination,he needs to put a third mark to get a win.
- Block: If the other player has two marks in a row ,the first player should place his mark on the grid to block the other player from winning.
- Fork: When the player has two ways to win.
- Blocking a fork A player should try to block all forks to prevent from the other player winning.
- Center: Marking the center is usually the best first move but it doesn't make difference it there are perfect players.

If winning is not possible a player should try strategies for a draw.

The code given generated a sample dataset for the OXO game using a MCTS algorithm. It included combinations of state values and the ac-tion that the agent took. The state here, is the board [ 0, 0, 0, 0, 0, 0, 0, 0, 0], the action is the action the MCTS took. Multiple games have been developed using neural networks and Monte carlo Tree Search and the action that are taken by the player at each state are kept, training a neural network which would mimic a human being. [3]

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.

The UCT/MCTS algorithm builds a game tree with a root node, and then expands the root node with random rollouts, while doing so it keep a count of visits and the number of wins for each node. In the end, the node with most wins is selected. [4] The algorithm consists of four phases:

- Selection:
  The MCTS algorithm starts with a root node and selects a child node with maximum win rate. The child node is selected by the UCT(Upper Confidence Bound applied to trees) formula

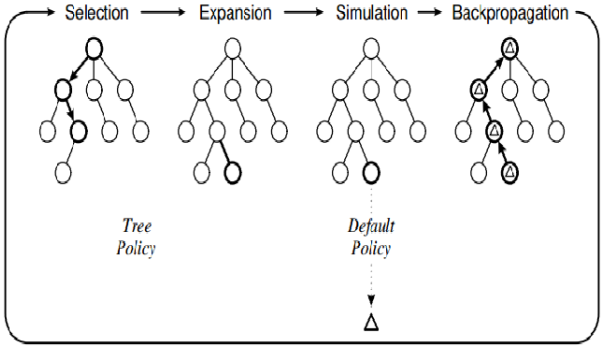
- Expansion:

When the algorithm can no longer apply UCT to find the successor node, it expands the game tree by appending all possible states from the leaf node.

- Simulation:
  After Expansion, the algorithm selects a child node arbitrarily, and it simulates a randomized game from selected node until it reaches the resulting state of the game.

- Backpropagation
  Once the algorithm reaches the end of the game, it evaluates the state to find out which player has won. It then traverses upwards to the root and increments visit score for all visited nodes. It also updates win score for each node if the player for that position has won the playout.

The MCTS algorithm repeats these four phases for some fixed n number of iterations. After that based on the random moves we estimate the winning score for each nodes. So more the number of iteration, more is our estimate reliable.At the beginning the algorithm is less accurate but it will keep on improving after reasonable amount of time.




Example of Monte Carlo Tree Search - implementation for tic-tac-toe.

## III. Methodology

The UCT/MCTS algorithm has initialised a 3X3 board with zero [ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ] indicating the available positions and update positions with 1 if player 1 takes a move and 2 if the computer takes a move and this is the state of the board. The action is what position a player will choose based on the current board state. The best move is also stored. The game results is in range 0 and 1. If it is a 0 it is a loss and 1 in case of a win else it is a draw.. [5] Lets demonstrate this with an example :

Suppose player1 decides to mark 5. So the first row will be stored in the dataset as follows:

1, [ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ] , 5.

and if the player2 decides to mark 6 so now second row will be stored as follows:

2, [ 0 , 0 , 0 , 0 , 0 , 1 , 0 , 0 , 0 ] , 6.

player1 decides to mark 3. So the third row will be as follows:

1, [ 0 , 0 , 0 , 0 , 0 , 1 , 2 , 0 , 0 ] , 3.

and if the player2 selects 4, so the fourth row will be stored as follows:

2, [ 0 , 0 , 0 , 1 , 0 , 1 , 2 , 0 , 0 ] , 4.

player1 decides to select 2, So the fifth row will be as follows:

1, [ 0 , 0 , 0 , 1 , 2 , 1 , 2 , 0 , 0 ] , 2.

and if the player2 selects 1 so the sixth row will be stored as follows:

2, [ 0 , 0 , 1 , 1 , 2 , 1 , 2 , 0 , 0 ] , 1.

player1 decides to select 7, So the seventh row will be as follows:

1, [ 0 , 2 , 1 , 1 , 2 , 1 , 2 , 0 , 0 ] , 2.

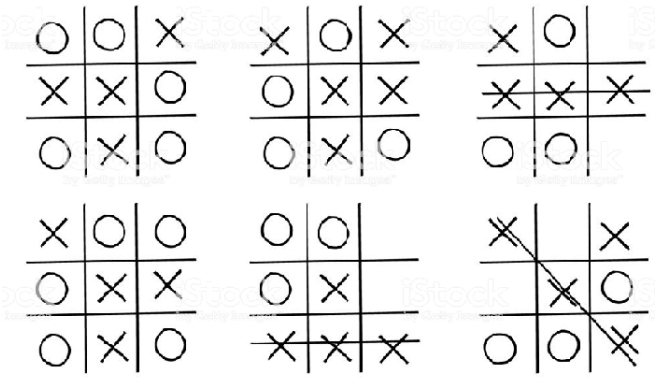
and if the player2 selects 8 so the eighth row will be stored as follows:

2, [ 0 , 2 , 1 , 1 , 2 , 1 , 2 , 1 , 0 ] , 1.

player1 decides to select 0, So the ninth row will be as follows:

1, [ 0 , 2 , 1 , 1 , 2 , 1 , 2 , 1 , 2 ] , 2.

In this way the UCT/MCTS generates a single game dataset comprising of 9 rows and it decides whether its a win, loss or draw. We will do this n times and the algorithm will generate the game dataset.



Sample tic-tac-toe games.
We will be using a Decision tree classifier and not neural network for our project.

Decision Tree

A decision tree is a flowchart-like tree structure where each node is a feature, the branch is a decision rule, and each leaf node is the outcome. The topmost node in a decision tree is the root node. It learns to partition on the basis of the feature

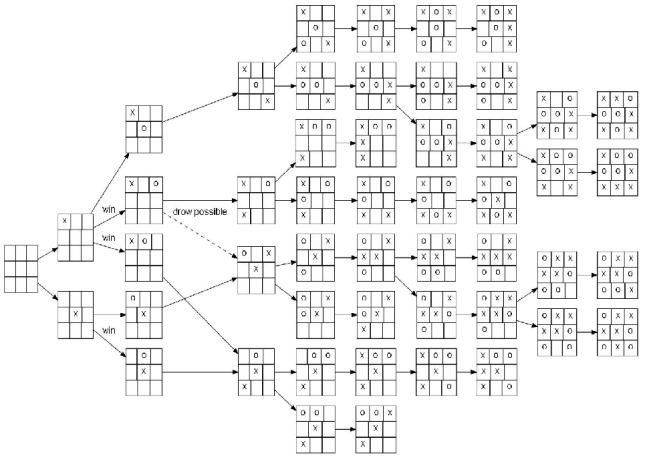
value. The tree is partitioned in recursive manner and it is known as recursive partitioning. This flowchart-like structure helps in decision making.

Decision tree classifiers perform more successfully, specifically for complex classification problems, due to their high adaptability and computationally effective features.

Based on the data generated by the UCT/MCTS algorithm which we generated n times we will train a decision tree classifier and will use to predict the next move in the game using the UCT code.

We will use this decision tree classifier in the UCT code and modify our code to help us choose the next move. We will use this classifier 90% of the times and will make random moves 10% of the time and generate another data-set Data2.

We will again train our decision tree classifier in the UCT/MCTS code with Data2 and generate a new decision tree and data-set Data3. We will keep on doing this n times and this is Expert Iteration , so we have generated n Data's and n Decision Tree and we assume that our present data is more learned and improved than our previous data. For eg Data3 has more learning than Data2.




Decision tree - implementation for tic-tac-toe

## IV. Experiments

Based on the number of decision trees generated , for every 10 iterations we will compare a decision tree with the previous 9 decision trees. For eg: Suppose in case of Decision Tree 20, we will compare this with the previous 9 ones.

Decision tree 20 Vs Decision Tree 11

Decision tree 20 Vs Decision Tree 12

Decision tree 20 Vs Decision Tree 13

Decision tree 20 Vs Decision Tree 14

Decision tree 20 Vs Decision Tree 15

Decision tree 20 Vs Decision Tree 16

Decision tree 20 Vs Decision Tree 17

Decision tree 20 Vs Decision Tree 18

Decision tree 20 Vs Decision Tree 19

and after comparing it with every iteration we need to check our accuracy score or measure the performance whether our learning is improving. We can make the assumption that the difference between Decision Tree 20 Vs Decision Tree 11 will be more than Decision Tree 20 Vs Decision Tree 19. So basically by comparing with every 10 iterations we can know that if our classifier is learning more.

## V. Conclusion

It is difficult to trust an algorithm that relies 10% of the time on random choices . However, through the implementation of Expert Iteration of Decision Tree Classifier we can conclude that the classifier is indeed improving and learning more with each iteration.It can indeed provide us a solution which can be used in many games as well as in decision-making problems as it focuses on nodes with higher chances of winning and is very easy to implement as compared to other Neural Network or Artificial Intelligence algorithm.

## VI. Data Loading

We have used the code which is provided and it uses the UCT Monte Carlo Tree Search algorithm which generates a dataset for 1 game of OXO. We have used this code and generated the dataset for 100 games and stored it in a oxo-game.csv file. For every move in a game it stores the player, the state of the current board and the best move.

## VII. Plan

The table below shows the breakdown of the different task and the time required accordingly.

| Breakdown of the work for completing the project | | |
|---|---|---|
| Task | Date | Time |
| Task 1 | 12 February - 19 February | 1 week |
| Task 2 | 24 February - 7 March | 2 weeks |
| Task 4 | 9 March - 23 March | 2 weeks |
| Task 5 | 23 March - 6 April | 2 weeks |
| Task 6 | 6 April - 20 April | 2 weeks |

## REFERENCES


[1] 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," in IEEE Transactions on Games, vol. 11, no. 3, pp. 195-214, Sept. 2019.

[2] Anthony, Thomas, Zheng Tian, and David Barber. "Thinking fast and slow with deep learning and tree search." In Advances in Neural Information Processing Systems, pp. 5360-5370. 2017.

[3] C. B. Browne et al. "A survey of Monte Carlo tree search methods" IEEE Trans. Comput. Intell. AI Games vol. 4 no. 1 pp. 1-43 2012.

[4] M. Enzenberger, M. Müller, B. Arneson, R. Segal, "Fuego—An open-source framework for board games and go engine based on Monte Carlo tree search", IEEE Trans. Comput. Intell. AI Games, vol. 2, no. 4, pp. 259-270, Dec. 2010.

[5] D.Silver,J. Schrittwieser, K. Simonyan et al. Mastering the game of Go without human knowledge. Nature 550, 354–359 (2017). https://doi.org/10.1038/nature24270