Assignment 1 - Report

Jose Zavala, *Student, University of Essex*

***Abstract*— This report defines the initial proposal of the creation of a Decision Tree Classifier model that can learn to *play* Tic-Tac-Toe by initially using Supervised Learning with a dataset generated by a Monte Carlo Tree Search with UCT that can successfully play the game, to the use a type of Reinforcement Learning to further tune the model to become a strong adversary in the game. The aim is to create a Decision Tree Classifier capable of recognizing the best possible move for a given game-state, without considering any other factors and without assuming any action on his adversary. This greedy approach has been successful in previous Tic-Tac-Toe playing agents, as well as the reinforcement learning techniques used have been used in the past with more complex models to successfully play more difficult games such as Go. It is expected that the proposed model will be able to recognize the strongest moves in the initial dataset since they should statistically be played more, as well as the reinforcement learning approach taken will accentuate the successful moves over the not-so-successful ones. A timetable of how the project will be taken into action is presented.**

I. Introduction

THIS project aims to demonstrate the use of supervised learning classifiers as tools powerful enough to imitate the best behavior of more complicated reinforcement learning models such as those used by the *Alpha Go* team to defeat the human world champion in a game of Go [1].

The project will explore the use of a Decision Tree Classifier to imitate the behavior of an already existing Monte Carlo Tree Search solution, to then use simple reinforcement learning techniques to strengthen the model's performance [1].

DTs are to be used since they've been demonstrated to imitate MCTSs in the past [2], as well as they offer a more memory and time efficient method to MCTS in practice [3].

If performance results are satisfactory after experiments are conducted, this will show that at least for Full-Information games with small state, simpler models can learn to predict moves given enough data about the current state of the game.

The paper first shows previous research on MCTS and Decision Trees on the topic. Then, it further explains the model that we are used for training, as well as the MCTS that we will use to generate the initial training data. After this, the dataset recollection methodology is explained. Ultimately, the experiments to evaluate and further improve the performance of the DT are described.

II. Background

The famous Tic-Tac-Toe, or OXO, is a "Full Information" game, which refers to the fact that all players have complete knowledge of the state of the match at any given point in time, unlike games like "Poker", in which certain information (i.e. the hand of a given player) is unavailable [4] [5]. Other examples of this type of games are Chess, Go, and Checkers.

A. Tic-Tac-Toe Knowledge

The size of the board for a game of Tic-Tac-Toe is 3x3 and each slot can have any of three symbols ('X', 'O', *empty*) at any given time, so intuition indicates that there are 3^9 or 19,683 possible game states [5]. However, these are limited by both conditions that restrict the feasibility of a game state of occurring (Figure 1) and the fact that by rotating the board some game states are equivalent (Figure 2) [5].

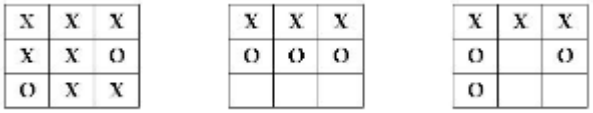

Figure 1 - Infeasible game-states [5]

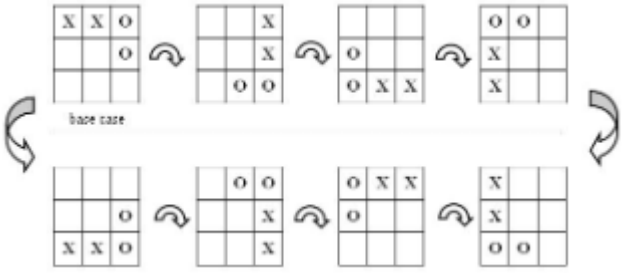

Figure 2 - Equivalent game-states [5]

These conditions reduce the intuitive number to 765 unique game-states [5].

B. Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a commonly used technique for the creation of agents capable of playing these types of games [1] [6].

MCTS uses a variation of the Upper Confidence Bounds (UBC) algorithm, called UCB applied to trees (UTC) [7]. This multi-armed bandit approach takes the action that takes to a node which maximizes Equation 1 for the current state [7]. Where w_i is the number of wins after visiting the ith node, n_i is the number of simulations after the ith node and N_i is the total number of simulations ever considered [7].

$$\frac{w_i}{n_i} + C\sqrt{\frac{ln\, N_i}{n_i}}$$

Equation 1 - UCT Node Selection [7]

This allows the algorithm to explore the possible scenarios of different games just by correctly encoding its mechanics into the algorithm, which demonstrates the effectiveness of MCTS for tracking future game states and predicting the optimal move based on that information [1] [6]. However, its implementation is inherently time and resource expensive since it needs to recreate a full tree in each turn since the game state changes with every move. Also, the random nature of the algorithm allows to overlook specific game paths that may result on a loss, since these may be the consequence of "weird" plays.

The success of MCTSs can be observed in a recent model, referred to as *Alpha Go Lee*, uses two networks: A Policy Network that assigns probabilities to moves trained first by supervised expert play and subsequently refined by policy-gradient reinforcement learning, and a Value Network that outputs a position evaluation trained by reinforcement learning to predict the winner of games of the Policy Network against itself [1]. These two networks were combined with a MCTS to provide a lookahead search, using the Policy Network to narrow down the search of high-probability moves and the Value Network to evaluate positions on the tree [1]. This model went as far as defeating the world champion of Go, Lee Sedol, back in 2016. A new version of this model, called *Alpha Go Zero*, is trained solely by self-play reinforcement starting from random play, using a single deep neural network [1]. This neural network is trained using a MCTS policy to select each move repeatedly in a policy iteration procedure [1]. This new technique of solely Neural Networks with MCTS has seen a huge increase in performance from its predecessor, winning 100-0 against it [1].

This demonstrates the power of MCTS for training a powerful agent for games as complicated as Go.

C. Alternatives to MCTS

The power of MCTS with the disadvantages it carries have motivated the development of models that are able imitate its behavior but are more time and resource efficient [2] [8]. These implementations aim to train models based on remembering the decisions taken by MCTS for quicker decision making [2], or by observing multiple runs and training a model with the examples gathered and the decisions taken by the MCTS for a given state [8].

The objective of these implementations is to create a model that can recognize which actions are taken more frequently over different games, reducing time needed for decision making and removing the random factor of MCTS.

These models show that it is feasible to imitate the behavior of a MCTS with another, more time efficient, Machine Learning model, such as a Deep Neural network with promising results [8].

D. Decision Trees

Decision Trees (DTs) have been proven to work against *Minimax*, another powerful algorithm that provides a No-Loss solution to the game [4]. This solution works since *Minimax* has been proven to not select optimal moves when the opponent makes a sub optimal choice for a given state [4]; however, *Minimax* is able to maintain a No-Loss Strategy in a non-optimal number of moves. DTs have been trained to perform a best optimal action despite the opponent's actions [4]. However, this DT was manually created, which is non-optimal for our purposes.

Following the research mentioned on II.C, it is possible to implement a DT trained on different games performed by a MCTS, to learn to imitate the behavior of the algorithm without the need of creating a full tree each time it needs to perform a move.

This provides enough theoretical ground to explore the performance of training a Decision Tree that imitates the behavior a MCTS to play Tic-Tac-Toe [2] [4] [8].

III. Methodology

The goal of the analysis is to determine if a Decision Tree algorithm may be used to imitate the MCTS behavior for a game of Tic-Tac-Toe with successful results. Unlike [2], the aim of this is not to train a DT with the UCT algorithm (Equation 1). Instead, the approach is to observe the decisions taken by an MCTS and train a DT with that information.

A. Model

To achieve the goal, a DT with information grain (ID3) will be used [3]. The model will receive the state of the board as an input of 9 features, each one symbolizing a position, and target those results to the cell that was selected by the MCTS algorithm.

The DT solution was selected for two major reasons. First, it will be able to identify the most selected plays by the MCTS for a given game state. Since the game has a very limited number of states [5], the DT should be able to determine which choices were most played overall, removing the *random* factor of the MCTS algorithm [7]. The second reason is that, since the DT is going to be trained to model the MCTS agent, and not merely to learn "how to play", the resulting tree will provide more insight about the relevance of certain cells over others at given game states.

To determine the decisions of the tree, the entropy function will be used to determine information gain [3]. The Entropy Function (Equation 2) allows us to know the homogeneity of the dataset for a given class, where p_m^i is the probability of classifying for class i. For a binary classification problem, this equation is graphed in Figure 3, where while probability of one given class increases, the probability of the second decreases; this causes a peak of entropy when both classes are equally represented. The more homogeneous it is, the more relevant it becomes to know which action to take [3]. To calculate the entropy of the dataset when a given feature (a cell in this case) is selected, Equation 3 is used, where n represents the number of attributes for a given features and K the number of classes. This equation let us get a weighted mean for the entropy of the dataset if it was discriminated by a given feature. The information gain takes the Entropy of the

dataset (Equation 2) and the Entropy of the dataset if segmented by a given feature (i.e. or cell in the board) (Equation 3) was to be selected. The feature that gives the most Information Gain is selected to split the node at the given game state.

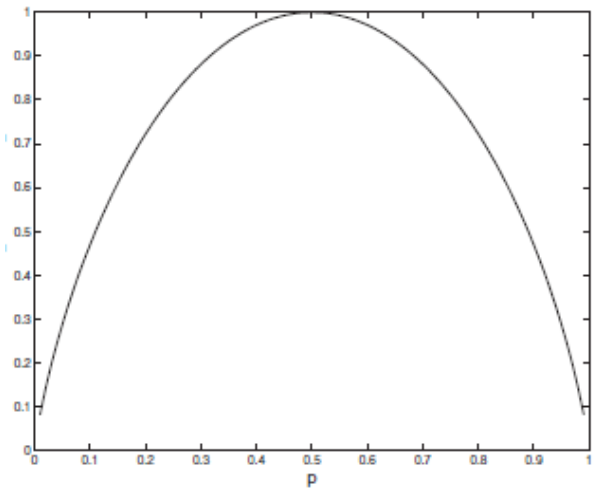

Figure 3 - Entropy Function for a two-class problem [3]

$$I_m = -\sum_{i=1}^{K} p_m^i \, log_2 \, p_m^i$$

Equation 2 - Entropy Function [3]

$$I'_m = -\sum_{j=1}^{n} \frac{N_{mj}}{N_m} \sum_{i=1}^{K} p_{mj}^i \, log_2 \, p_{mj}^i$$

Equation 3 - Entropy Function for Feature [3]

$$Inf_{gain} = I_{class} - I_m$$

Equation 4 - Information Gain [3]

This allows the DT to understand which cells are more important for the MCTS algorithm in a given situation having enough training examples to choose from. To "understand" the importance of a cell is to learn which cell's states possess less entropy as to correctly discriminate the next action taken. At a leaf node, the action taken is the one with the most representation available, which statistically should be the most effective move performed by the target MCTS [7].

To implement the DT, the DecisionTreeClassifier package from scikit-learn (https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html) is going to be used. A game state will be encoded in a vector of data, in which each slot is a cell in the game board, and the target will be one number from 0 to 8, indicating the slot in which the next move will be performed.

B. Dataset Collection

To gather data to train the proposed DT model, the code provided by the Monte Carlo Tree Search Research Hub (http://mcts.ai/code/python.html) was used as a basis. The original code provides an implementation of the UCT algorithm for three games: Nim's Game, OXO (Tic-Tac-Toe), and Othello. The Code was modified to save the game states during a match from the perspective of the player whose current turn is being played.

Previous research mentioned in II.A indicates that there are 765 unique game states in the board at any possible time [5]. However, since neither MCTS nor DT can determine when a state is a rotation of another game state, we undo this knowledge to have a number of 6,120 game states. To be sure to capture all this, an arbitrary number of 10,000 games will be stored as to also account for the validation set. No repeated game states will be removed since they reflect the frequency the MCTS makes those choices.

To allow our implementation to play either as "X" or "O", we encoded the board of the game depending on the current player, and not as "X"'s and "O"'s, , being "X" if it indicates a piece placed by the current player, "Y" if it was placed by the rival, and "_" represents an empty slot (Figure 6).

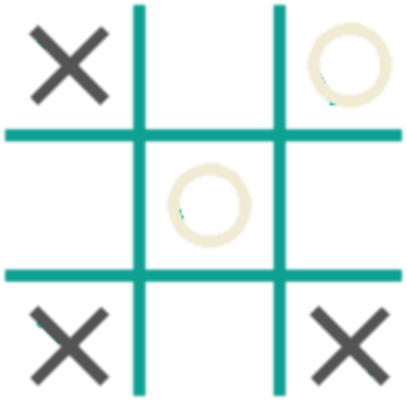

Figure 4 - Game State

1	0	2	0	2	0	1	0	1

Figure 5 - Original Encoding by MCTS Research Hub

Y	_	X	_	X	_	Y	_	Y

Figure 6 - Encoding for Second Player's (O) Perspective

Upon inspection of data, it was decided that every move should be stored, even from the perspective of a losing agent. This was determined since MCTS will always perform the best possible move available for a given scenario [7]. In an optimistic approach, even if the agent loses, it means that no better move was possible for a scenario [1]. Also, it is assumed that in the long run, the MCTS algorithm will chose the best moves more frequently over the non-optimal ones [1] [7], which the DT implementation should capture. A representation of a possible resultant DT is given in Figure 7.

[0:0]	[0:1]	[0:2]	[1:0]	[1:1]	[1:2]	[2:0]	[2:1]	[2:2]	Move
_	_	_	_	_	_	_	_	_	4
_	_	_	_	Y	_	_	_	_	0
Y	_	_	_	X	_	_	_	_	2
X	_	Y	_	Y	_	_	_	_	6
Y	_	X	_	X	_	Y	_	_	3
X	_	Y	Y	Y	_	X	_	_	5
Y	_	X	X	X	Y	Y	_	_	1
X	Y	Y	Y	Y	X	X	_	_	7
Y	X	X	X	X	Y	Y	Y	_	8

Table 1 - Capture of a single MCTS game using the proposed encoding

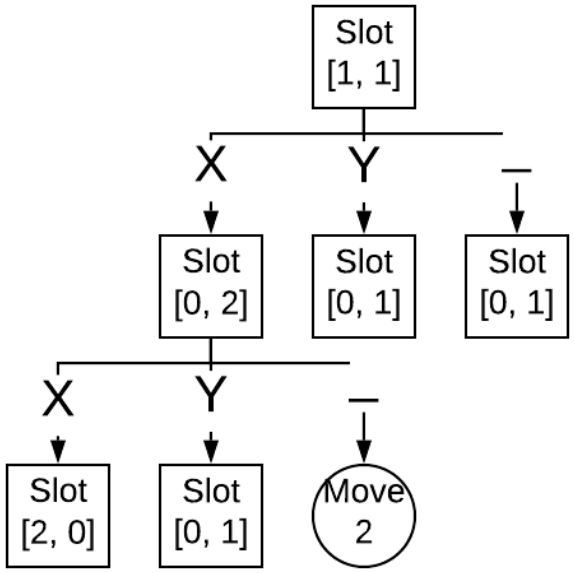

Figure 7 - Decision Tree Representation for State in Figure 4 from Second Player's (O) Perspective with the new encoding

C. Training

Training will ensure that the DT reflects the behavior of the MCTS algorithm, and not the performance of the DT as a playing agent. To evaluate the model's performance over the original dataset, 10-Fold Cross Validation (CV) will be used. The CV is needed for exploring the values for the minimum impurity decrease for pre-pruning, diminishing the possibility of overfitting.

When an optimal parameter is found, the classifier will be trained over 70% of the dataset, using the remaining 30% as a testing set to measure performance with accuracy, precision, recall, and the F1-Score [3].

IV. Experiments

All the processes described in Section III explain how the initial training and classification performance of the model will be realized. However, we now need to test the ability of the model as a candidate to compete against other agents in matches of Tic-Tac-Toe.

An approach similar to the one described to train the previous version of *Alpha Go Zero* in [1] will be used, in which we first train the model with "expert play" (i.e. the MCTS agent), to then implement self-pay to further refine winning moves.

A. Initial Performance Testing

To test the abilities of the DT, we will compare its gameplay against the original MCTS agent used to train it. We will observe its win_rate score, which is only the percentage of the games that resulted in draw or a win, in order to aim for a No-Loss strategy performance for the DT.

100 games will be played against the original MCTS to calculate an initial win_rate score. The games will be stored as explained in section III.B, but instead of storing all games, the model will be fine-tuned with the movements of the winning agent. This was not done before since we assumed an optimal strategy from the MCTS in all scenarios, but with enough examples for basic play, we are now focusing on learning the best strategies. This method of further learning was also performed by [1], in which they first explored Go games between expert human players and then let their own models to further play against previous iterations of themselves for faster learning with successful results.

B. Further Training

After the initial plays with the MCTS agent, a second version will be developed with the new training examples. 10 games will be played against the first version, and the new win_rate score will be saved. Again, new training examples will be saved from the winning agent's moves. These moves will be used to train a new agent each time [1]. To avoid overfitting on the same decisions each time, a random factor will be added, which will consist on selecting a random available move with a probability of 10% each turn.

This process will allow us to understand if the DT is able to learn from successful strategies if enough training examples are provided. The success of the model will be determined with the change of the win_rate score over time.

V. Discussion

As mentioned in Section IV, the performance of the DT as an agent capable of playing Tic-Tac-Toe will be measured with the win_rate score over time.

Even though we are implementing the basic process used to train *Alpha Go* for the far more complex game of Go [1], the team behind that agent was using a Reinforcement-Learning technique, which allowed the final model to find optimal strategies over time in a more dynamic environment.

There are several limitations to our approach, the first and most notorious one is that the model we are training is a supervised learning classifier and not a reinforcement learning agent [3]. This means that it will only go as far as imitating the data it was trained with, and not to actually learn successful strategies (Figure 7). Therefore, we expect small performance improvement after competing against past versions of itself, since the only change is that it will add more weight to successful movements in a given game-state without knowledge on the opponent's strategies nor previous time steps.

Another limitation is the way data will be stored in the Experiments section. Since only the actions already recorded by the classifier are going to be played, this may cause the model to overfit on a set of plays, which then may lose against the original MCTS model.

VI. Conclusion

This project will aim to implement a DT that at first will imitate the behavior of a MTCS playing a game of Tic-Tac-Toe, to then use reinforcement learning techniques [1] to try to obtain more information about game-winning plays.

It is expected that the DT will learn from the MCTS with a high performance, but with a considerable classification error, since MCTS has a random factor which will cause it to perform different moves at the same state for different games, which will impact the classifier performance on initial testing.

For the reinforcement learning part, it is expected that the DT will improve its win_rate score after some iterations against past versions of itself, but since it will only consist of adding new training examples of more efficient plays this increase will be limited.

VII. References

[1] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, pp. 354-359, 2018.

[2] C. Nunes, M. De Craene, H. Langet, O. Camara and A. Jonsson, "A Monte Carlo Tree Search Approach to Learning Decision Trees," in *17th IEEE International Conference on Machine Learning and Applications*, Orlando, 2018.

[3] E. Alpaydin, Introduction to Machine Learning, Massachusetts: Massachusetts Institute of Technology, 2014.

[4] S. Sriram, R. Vijayarangan, S. Raghuraman and X. Yuan, "Implementing a no-loss state in the game of Tic-Tac-Toe using a customized Decision Tree Algorithm," in *International Conference on Information and Automation Information and Automation*, Zhuhai/Macau, 2009.

[5] A. Bhatt, P. Varshney and K. Deb, "In Search of No-loss Strategies for the Game of Tic-Tac-Toe using a Customized Genetic Algorithm," Indian Institute of Technology, Kanpur, 2008.

[6] S. S. Sista, *Adversarial Game Playing Using Monte Carlo Tree Search,* Cincinnati: University of Cincinnati, 2016.

[7] L. Kocsis and C. Szepesv´ari, *Bandit based Monte-Carlo Planning,* Budapest: Computer and Automation Research Institute of theHungarian Academy of Sciences, 1999.

[8] T. Anthony, Z. Tian and D. Barber, "Thinking Fast and Slow with Deep Learning and Tree Search," in *31st Conference on Neural Information Processing Systems*, Long Beach, 2017.

VIII. Plan

A Gantt Chart below show how the development of the project is going to be performed:

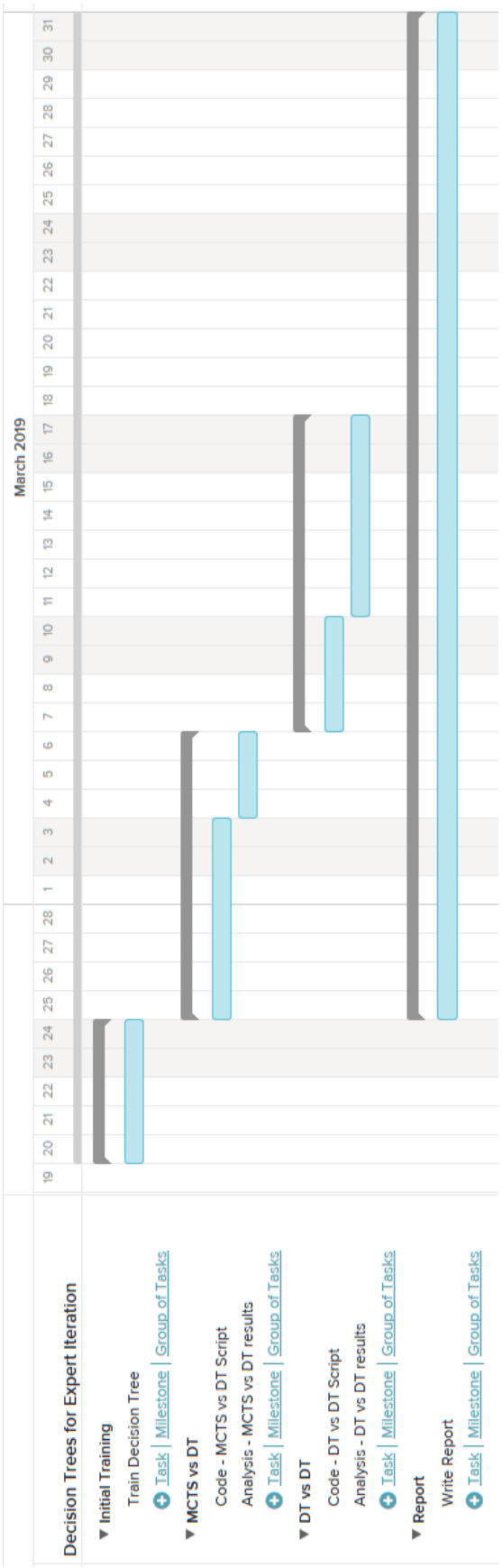

Figure 8 - Gantt Chart of the Project