Reinforcement Learning with Tic Tac Toe

Christina Yu
PID: A10778557
UC San Diego
Github link: https://github.com/teddybeargun/cogs182project
Youtube link: https://youtu.be/yTCBwCMKkkI

Abstract

Tic Tac Toe is a widely used game in reinforcement learning for learning due to its smaller amount of states compared to many other games like Chess or Go. In this experiment, I utilize this game in order to both practice coding reinforcement learning algorithms, and to test different algorithms against each other in order to see which one is better, if any. The algorithms I'm testing specifically against each other are n-step TD and TD(0).

1. Introduction

Reinforcement learning is a branch of machine learning different from supervised machine learning and unsupervised machine learning. Reinforcement learning is focused on "goal-directed learning from interaction"(Sutton & Barto, 2018). This means that the agent interacts with the environment with a goal in mind, namely the reward that it wants to maximize. Trial and error search and delayed rewards are the most important features of reinforcement learning that distinguishes itself from other forms of machine learning. These distinguishing factors help reinforcement learning to be great algorithms for learning games. Other algorithms like MinMax algorithm assumes a perfect opponent that makes the most optimal choice every time and does not have an answer for flawed opponents. Reinforcement algorithms, on the other hand, learn the strategies in order to maximize rewards, like a human would, but do not choose the most optimal strategy every time. This is because reinforcement algorithms do not have a model of the opponent, and instead treats the opponent as a part of the environment that it interacts with. This lack of perfection helps reinforcement learning algorithms to be much more flexible and able to converge to a better strategy faster.

2. Motivation

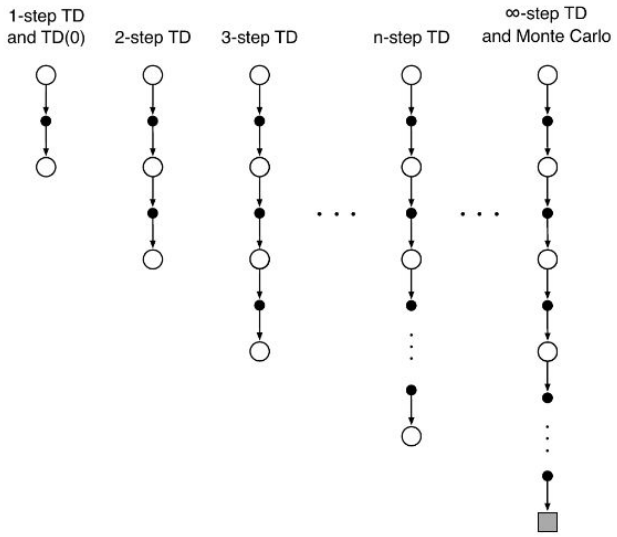
My motivation for choosing Tic Tac Toe is that it is a game I played a lot growing up, and I thought it would be interesting to choose it. I initially chose Connect4 as my game of choice, which is essentially Tic Tac Toe with a larger amount of states and gravity involved. Connect4 proved to be a bit too difficult of an environment for me to build, so I chose to make it simpler and go for 4 in a row Tic Tac Toe. With a game like Tic Tac Toe that has a lot fewer states in comparison to other games like Connect4, Go, and Chess, I thought it would be interesting to compare and see if different reinforcement learning algorithms would have much difference in learning how to play from each other.

3. Methodology

3.1 - Theory

To update value estimation of states, value iteration is applied which is based on this formula: $V(S_t) \leftarrow V(S_t) + \alpha[V(S_{t+1}) - V(S_t)]$. This formula says that the updated value of state t is equal to the current value of state t + learning rate multiplied by the difference between the value of the next state and the value of the current state. TD(0) is very similar to this value iteration function, the difference being that the next reward and decay rate gamma is also factored in, as shown in $V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t))]$. The target for TD update is $R_{t+1} + \gamma V(S_{t+1})$. Given some experience following a policy π, the method updates the estimate V of v_π for the nonterminal states S_t. At time t+1, TD(0) method immediately forms a target and makes the update using the observed reward R_{t+1} and estimate $V(S_{t+1})$. Because TD(0) "bases its updates in part on an existing estimate, we say that it is a bootstrapping method, like DP" (Sutton & Barto, 2018). TD methods combine the sampling of Monte Carlo method with the bootstrapping of DP, which should in theory make TD method better.

N-step TD method is a method that generalizes both Monte Carlo method and TD(0) method. It is a spectrum of Monte Carlo on one side and TD(0) on the other side. Below is a diagram from the class textbook, showing the spectrum of n-step TD method.



Chapter 7: Sutton & Barto, 2018

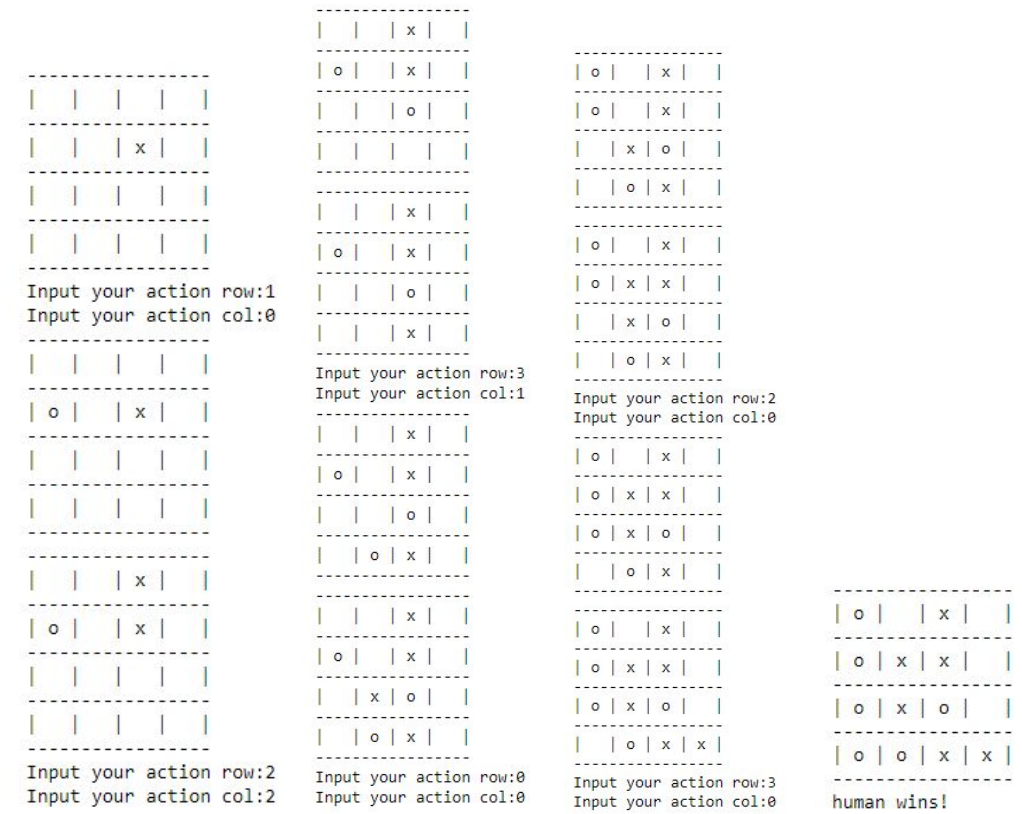
The theory behind n-step, is that the idea for one-step updates makes sense after two steps, as well as one-step. So similarly, the target for an arbitrary n-step update is the formula: $G_{t:t+n} = R_{t+1} + \gamma r_{t+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$. The value learning algorithm for using n-step is similar to TD(0) but factoring in an n amount of steps, rather than an assumed one step, $V_{t+n}(S_t) = V_{t+n-1}(S_t) + \alpha[G_{t:t+n} - V_{t+n-1}(S_t))], \; 0 \leq t < T$. An important property of n-step returns is that "their expectation is guaranteed to be a better estimate of v_π than V_{t+n-1} is, in a worst-state sense" (Sutton & Barto, 1992). This means that the worst error of the expected n-step return is guaranteed to be less than or equal to γ^n multiplied by the worst error under V_{t+n-1}

3.2 - Code

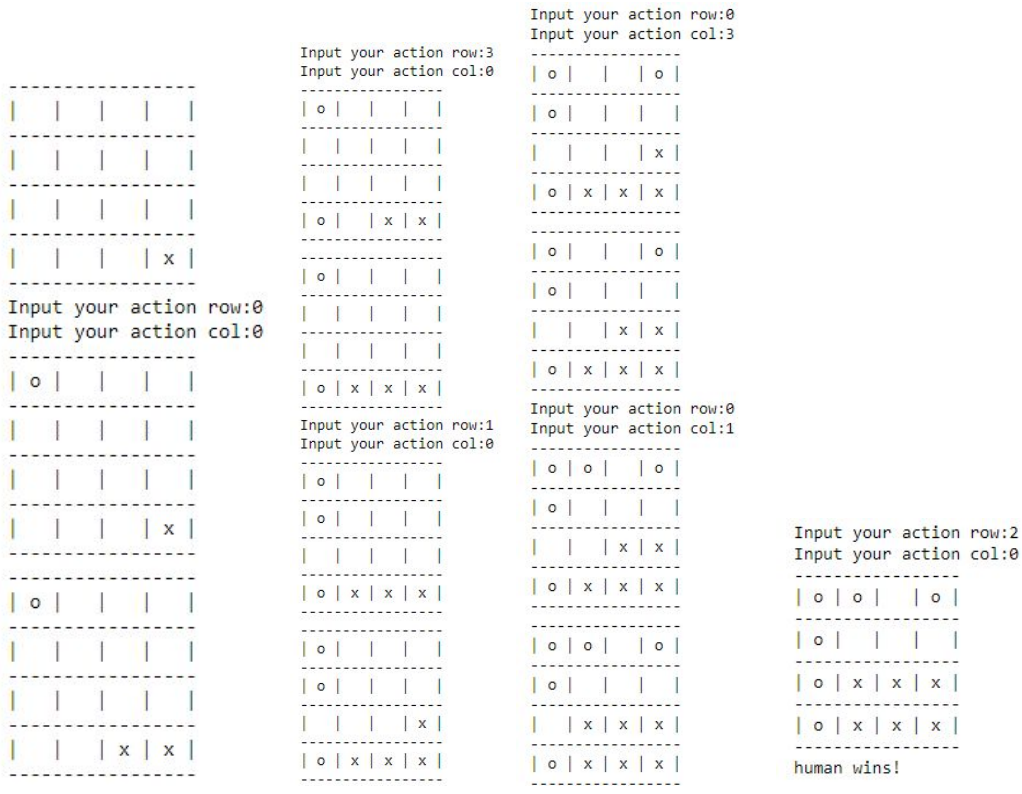
My methodology was to first construct the environment of Tic Tac Toe with a 4x4 board. I assigned a player 1 and a player 2 with functions recording the board state of each player and updating the state when either player takes an action. It then determines at the end of the game which rewards to give out based on which player won. The most important parts of this setting is the state, action, and reward. The state is the board state of the agent and its opponent, the action is what positions the player can choose based on the current board state, and the reward is between 0 and 1 and is given at the end of the game. I then added a player class that is the agent which chooses actions based on the current state, records the state of the game into a list, updates the states-value estimation after the game, and then saves the new policy. After setting up the environment and agent, I train the agent by having the agent play against itself. After, I set up the ability to have you able to play against the agent by loading the policy and then prompting the human player to enter in the index value of the position on the board you would like to make a move on.

4. Results

The results of my algorithms were not what I expected. The algorithm demonstrates an understanding of the game, but does not seem to learn from its training rounds when I had it trained against itself 50,000 times. I also increased the rounds of training to 100,000 and then 500,000 to see if it made a difference, but it did not make a difference in the amount of times won against a human player. I chose these values, because we used similar amounts of runs in the homework problems and it seemed like it would be a good number to use to train. At times it knows what it is doing and blocks my moves when I challenge it, but other times it seems to place it's move at random. Also I noticed sometimes it acts like it is operating on a Tic Tac Toe board that is 3x3, rather than 4x4, which I think might be a flaw in my code, however, I could not figure out what the flaw was. I am not sure if this deficiency in the algorithm is due to a need for more training, a flaw in the code, me being better at the game than the agent, or a combination of the above. The below images depict the agents playing against me after training by playing itself. N-step agent fared a bit better against a human player tha TD(0), but that could also just be natural variance in play. The lack of desire by the agent to win, may also be because of me giving 0.5 reward when the agent ties, so I am not sure if that is a factor in the agent being unable to win against me. I chose 0.5 as a reward for a tie, because I did not want to give the agent the full reward for not winning, because it would most likely go for a tie instead of trying to win, like was mentioned in class before, to only reward what you want to happen rather than subgoals. I also tried 0.1 as a reward, but based on the behavior of the agent, 0.5 at least was better. With some tweaking, perhaps the agent will learn from these rewards more.


Input your action row:1
Input your action col:0
Input your action row:2
Input your action col:2
Input your action row:3
Input your action col:1
Input your action row:0
Input your action col:0
Input your action row:2
Input your action col:0
Input your action row:3
Input your action col:0
human wins!

TD(0)


Input your action row:0
Input your action col:0
Input your action row:3
Input your action col:0
Input your action row:1
Input your action col:0
Input your action row:0
Input your action col:3
Input your action row:0
Input your action col:1
Input your action row:2
Input your action col:0
human wins!

TD(n)

5. Related Work

There is much related work in reinforcement learning for games like Tic Tac Toe. I have seen many different RL methods being used such as Q-learning. Deep Q-learning is an interesting algorithm that I would like to attempt next time for Tic Tac Toe as based on other articles I have read, it seems like it would solve this game quickly and efficiently, while also using significantly less space than a regular Q table in tabular Q learning.

6. Conclusions

At first I attempted to create an algorithm for Connect4, but the environment and agent were not working. Getting the gravity aspect of Connect4 to work seemed like too difficult of an undertaking for me at this current moment with the amount of time I had. After switching to 4 in a row Tic Tac Toe, I was able to set up the environment correctly, and make the agents able to play against a human by inputting in the row and column of the space you want to move in. I was able to implement the human player aspect well, however my code at the moment currently has the agent losing a lot. I would like to improve this code in the future, and figure out how to correctly set up the rewards in order for the agent to learn more efficiently from its training phase.

7. Acknowledgements

I would like to thank Professor Marcelo Mattar and the TA Daniel Acosta-Kane for teaching us this quarter on reinforcement learning, so that I am able to write these algorithms.

8. References

Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. Second Edition MIT Press, Cambridge, MA, 2018.