Tic Tac Toe Using Artificial Intelligence

Casey Delaney, Alireza Mahinn parvar, Jay bharadvar [1]

[1]

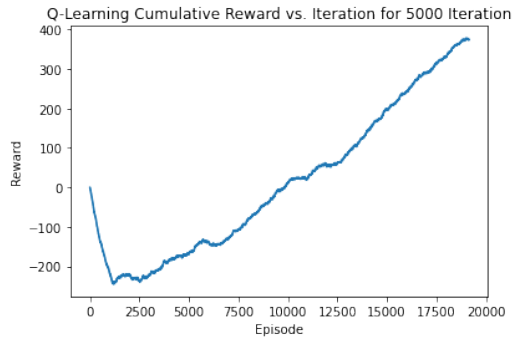

Fig. 1. Q-Learning Reward at 5000th iteration and 20,000 episodes

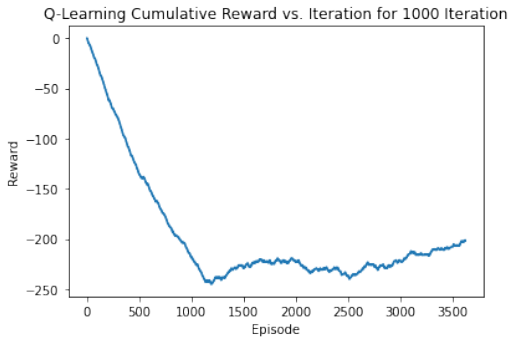

Fig. 2. Q-Learning Reward at 1000th iteration and 4000 episodes

1. Introduction

For this project, our team implemented two different reinforcement learning techniques based on the Markov Decision Process to compare and contrast the different algorithms on a simple game board, Tic Tac Toe. This game consists of three rows and columns which limits the number of possible moves and therefore making it a reasonable game to attempt this reinforcement learning technique on. In this paper, we will compare the performance statistics from each learning agent and determine which technique is better suited for AI in games.

The main part of this project consists of getting the reinforcement learning techniques to work on a game of Tic Tac Toe. This part of the project was successfully completed and will be demonstrated below. After the majority of the project implementation was completed, our team also worked on designing a GUI for the game as an added bonus. The project implementation and GUI were designed separately and the team is currently working on combining these two separate components. This bonus part is not yet completed, so we have decided to include the files separately for you to play with. The GUI game design is within the GUI_design.ipynb file located in the Github repository. Please refer to the README file to run these files. The reinforcement game implementation is held within the main_code.ipynb file.

2. Methods

In this project, our team decided to analyze the SARSA algorithm and Q learning algorithm in a Tic Tac Toe game. After learning about popular AI solutions, we found that both Q learning and SARSA algorithm are very common algorithms that are often used in reinforcement learning algorithm projects around the world. Q learning are SARSA algorithm are methods that do not require model knowledge and the only thing required for them is to have rewards from many experiments runs. As we all know, SARSA and Q learning make updates, after each step they take. By looking at the methods implementation below we can see that the differences in these methods can cause significant changes. But these methods have some similarities also. For example, in both methods, the policy is greedy. For each transition from state S to S' taking action A follows the greedy policy

$$Q(S,A) < -Q(S,A) + a[R + gQ(S',A') - Q(S,A)]$$

The above equation shows the SARSA algorithm implementation and as we can see, the new action a' was chosen and it uses the same policy as the previous action which is a and followed by greedy policy we have and that is why we call this algorithm an on-policy algorithm.

$$Q(S,A) < -Q(S,A) + a[R + gMax(S',A) - Q(S,A)]$$

The above equation shows the Q-learning algorithm implementation and as we see above in this algorithm greedy action A' will be considered and this is the action that maximize the Q-value at state Q(S',A). There is some key information we must remember when we are dealing with these algorithms. The first one is that SARSA algorithm attempts to directly estimate the value of the exploration policy, but Q learning tries to find the value of the optimal policy while following the exploration policy. Another thing we must remember is that the actions we are talking about are not the real actions that will be taken. These are the actions that have been chosen to find targets in updated equations.

In this project, the team started the implementation by defining the parent class which includes all of the common methods for the Q-learning and SARSA algorithms. The common methods like first initialization, getting actions and saving states are the methods and functions we see in this section.

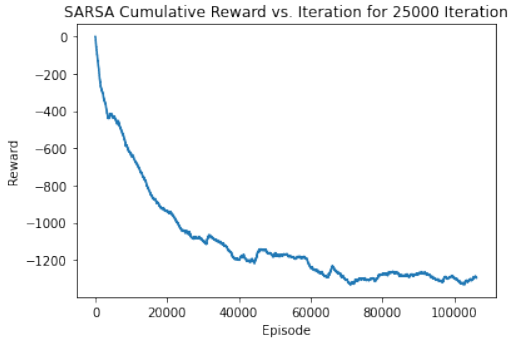

Fig. 3. SARSA at 25000th Iteration and 100000 Episodes

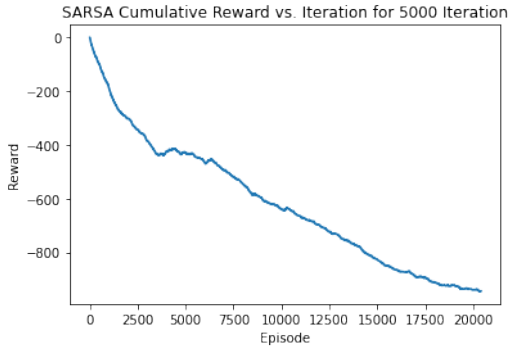

Fig. 4. SARSA at 5000th Iteration and 20000 Episodes

In the second part, we decided to define the Q-learning class that includes the update function and initialization for the Q-learning function and in the same section we have a class for the SARSA algorithm. In other sections of our implementation, we have the TrainerForQs class that includes Random move, Make move, side empty, corner, center, fork, block-fork, win and block win methods. These methods define the possible moves in a game of Tic Tac Toe. Our TrainerForQs class is a class that implements a teacher who knows the optimal playing strategy. It returns the best move at any time given the current state of the game. The last two sections of our implementation include the Game class that involves all of the functionality regarding the board game like check win, check draw and more. We also have a state holder class that includes a Begin Teaching function and it will hold the state of the learning process. This function is used when the user wants to teach the agent themselves by playing against them rather than using automated agents

3. Comparisons

By looking at the figure 1 and figure 2 plots, we can see how the numbers of episodes and iterations made changes in the results. As we mentioned before in the Q-learning and SARSA Algorithms, we do not need knowledge from a model. We are only required to have the number of rewards from many experimental runs. In figure 1 we see that the reward at the 5,000th iteration and 20,000th episode is different from the reward at the 1,000th iteration and the 4,000th episode (Figure 2). In both plots we see the slope is down at the beginning, but it increases after some number of episodes. The slope is sharper toward the upside when we have more iterations and episodes but when the number of iterations and episodes are less, we see that slope is not as sharp, but it still has an upward trend. By having more episodes and iterations, the agent in Q learning method has a better choice to maximize the value of the Q function and it will gives us maximum reward possible.

To compare the SARSA algorithm in different cases, we can see in figure 3 and 4 that by increasing and decreasing the episodes and iterations, the plot will not change by a substantial amount. When we have the 25,000th iteration and the 100,000th episodes, the slope is downward and the plot is decreasing the rewards. One thing that needs to be considered is that in the 5,000th iteration and 100,000th episode (Figure 4) the slope is moving linearly downward and it decreases slowly.

4. Conclusions

In this project, our team decided to implement two different methods of reinforcement learning on a Tic Tac Toe game. One of the methods is the SARSA Algorithm and the other one is the Q learning algorithm. The purpose of this project was to analyze these methods on one of the most famous board games of all time. In this project, our team decided to have the SARSA and Q learning functions as an automated agent and out team also decided to add features to this AI system such as the AI system playing against the user and learning from user input instead of the automated agent. The results from this analysis were interesting and it shows the differences between the SARSA and Q learning algorithm. As we all know, the Q learning function system is going to maximize the reward, and in other words, it attempts to find the value of the optimal policy while following the exploration policy. In Q learning, we see that it attempts to directly estimate the value of the exploration policy as it follows it. By looking at reward results in both methods, we concluded that by increasing the episodes and number of iterations, the reward goes up in the Q learning method. We saw that in both plots (1,2) the slope was downward at first but at one point the direction changed and in the high iteration values the slope became even sharper toward the upside. In the SARSA algorithm, we saw that the number of rewards went down after some number of iterations no matter what. The only thing that made differences was during the time when we had more iterations in the SARSA algorithm, and the slope turned to be linearly downward. In conclusion, this project shows us the differences between two different agents and it showed us how the number of iterations and episodes can change the reward values in each method.