Using An Agent to Learn the Snakes & Ladders Board Game

Yu Xuan Yong, PID: A16078479[1,✉]

[1]University of California, San Diego, COGS 118A

Abstract - This report presents the use of Q-learning and DynaQ, reinforcement learning algorithms taught in class to teach an agent how to play a simple board game of Snakes & Ladders.

1. Introduction

Throughout the course of the COGS 182 lecture, I have learnt many ways of reinforcement learning, which mostly refers to learning an optimal value function that shows us the optimal state-action pairs to transition to in the environment.

The course has made use of the Gridworld to guide my understanding of reinforcement learning. As such, for my final project, I decided to adopt a similar environment, albeit on a larger scale, to work on. I eventually settled on the idea of a board game known as Snakes and Ladders, as I had experience playing it, and the environment involved with playing the game is similar to the environment of the Gridworld.

The rules of this game are as follows:

- The agent/player will be playing the game on a 10X10 board of 100 squares.
- The agent will start on square 1.
- The agent then proceeds to roll an unfair dice of unknown probability for each number. The dice is assumed to have a higher probability of landing on a smaller number (e.g 1) than a larger number (e.g 6).
- According to the results of the roll, the agent is able to choose how many steps it wants to move forward by.
- If the agent lands on a square with a ladder, the agent instantly moves up to a higher state at the end of the ladder. However, if the agent lands on a square with a snake, it moves down to the tail of the snake instead.
- The agent receives a reward once it reaches square 100, which is the goal state.

The details of the game are as follows:

- The agent will be the player playing the game.
- The environment will be the Snakes & Ladders board, which will have 100 squares in a 10X10 format.
- The states will be the position of the agent/player on the board.
- The action the agent takes will be the number of steps the agent chooses based off of the result of the dice roll.
- The reward function will be -1 for each transition, and 1 when the agent reaches the goal.

The reinforcement learning portion of the game lies in the agent being able to choose how many steps it will be taking. This will allow the agent some agency to learn the fastest way to reach the goal state.

2. Methodology

2.1 Class Methods The environment class and agent class was created for efficiency and convenience in coding. The environment class was named **SnakesAndLadders** while the agent class was named **Agent**.
The **SnakesAndLadders** class contained methods such as **step** to transition through states in the environment. The **Agent** class contained methods like **epgreedy** and **updateQ**, which selected actions based on an epsilon-greedy policy and updated the Q value function respectively.

2.2 Learning Algorithms Attempts to learn the optimal value function for each respective learning algorithm with regards to the chosen board game was made. This section is a summary of the parameters used for each learning algorithm. Algorithms were implemented using numpy and matplotlib.

Q-learning Using the learning algorithm Q-learning, state-action values for each state of the Snakes & Ladders board game was learned. The Q-learning used parameters $\alpha = 0.5, \gamma = 1$ and $\epsilon = 0.05$. The agent played the game for about 5000 episodes, where each game is episodic and represents 1 episode which the agent played till termination. I used 5 experiments as well while running the DynaQ algorithm.

DynaQ Using the planning algorithm DynaQ, a model and the number of steps taken were generated from the function. The DynaQ used parameters $\alpha = 0.5, \gamma = 1$ and $\epsilon = 0.05$. The algorithm was ran for 1,10 and 100 planning steps respectively, with the maximum number of steps being 15. Similarly, the agent played the game for 5000 episodes,

where each game is episodic and represents 1 episode which the agent played till termination.

2.2 Data to Assess Learning

Q-learning For Q-learning, I used the number of actions taken to finish the game vs the number of episodes. I also used a heatmap to determine the maximum value per state using Q-learning[1].

DynaQ For Dyna-Q, using the number of steps per episode, I graphed the number of steps vs the number of episodes, with each line representing how many planning steps were utilized[1].

3. Results

3.1 Main Results

Fig. 1. shows the heatmap produced by the Q value function derived from using the **Q-learning** algorithm.

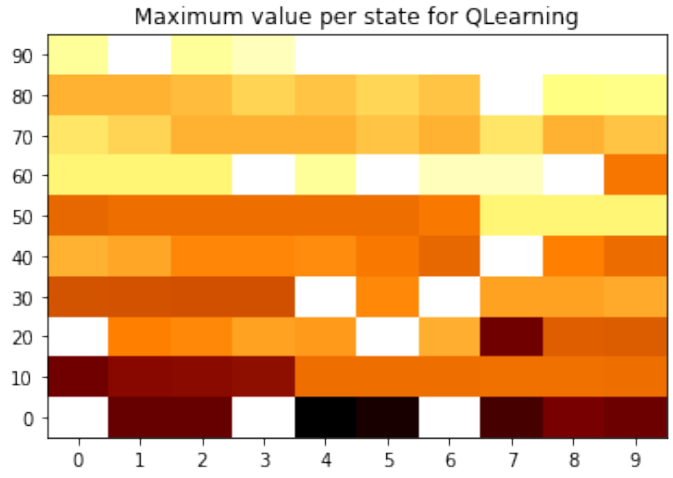

Fig. 1. Heatmap of maximum value of each state.

The value increases near the end of the game, since this would mean that the agent is close to the goal. This can also be seen through noticing that the color of the boxes are lighter towards the end goal at 100. The white squares represent the ladder and snake states, since those will have a value of 0. Choose to step through a certain amount of steps can give the player a faster way to finish the game. This is sufficient to deduce that the heatmap represents how optimal it is to land on a particular state. This is also reflected in how the last 5 states are white due to a higher chance of potentially winning the game and earning the reward.

Fig. 2. shows the number of actions the agent has to take to reach the goal across a total of 5000 episodes.

As shown from the graph in Fig 2. The graph sees a huge number of actions taken at the start and it rapidly decreases as the number of episodes increases. The random spikes

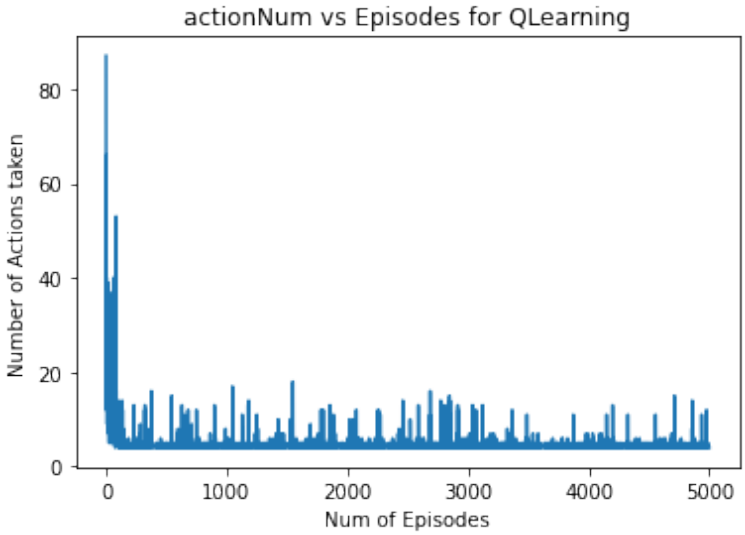

Fig. 2. Graph of number of actions taken by agent against the number of episodes.

through the 5000 episodes probably represent the agent taking exploratory actions to find out if there were any faster ways to get to the goal.

Fig 3. shows a graph plotted with the number of steps taken against the number of episodes. The episodes was set to a range of 150.

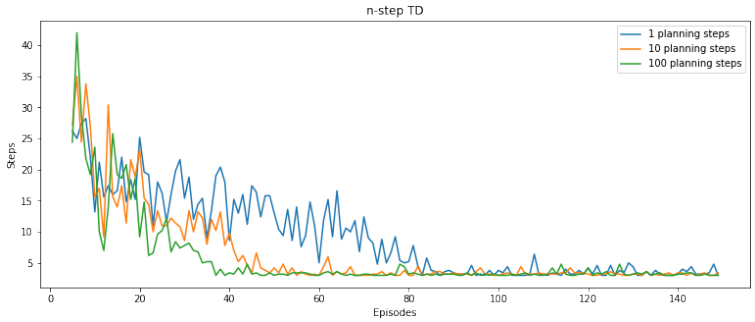

Fig. 3. Graph of n-step TD using DynaQ for the number of steps taken vs the number of episodes.

As seen in the graph in Fig 3., using DynaQ and adjusting the number of planning steps helped me arrive at the optimal and minimum number of steps that the agent could take to reach the goal. With 1 planning step, the agent takes longer to be able to learn the minimum number of steps at around 90 episodes. However, with more planning steps, the agent learns the minimum number of steps to reach the goal in about 60 episodes for 10 planning steps and 40 episodes for 100 planning steps. This is because with more planning steps, a more extensive policy is built to back track to the start state. Even for 100 planning steps, 40 episodes are needed because of the size of the Snakes & Ladders board, which contains 100 states.

4. Merits and Deficiencies

4.1 Merits

The merit of this project comes from the fact that it is built off of an actual game, so this might form a basis on learning other similar board games as well. Furthermore, with Q-learning and DynaQ, the learning is pretty efficient and fast. Furthermore, this Snake & Ladders board game is a more complicated version of the Gridworld that was touched upon in class, so this builds off of the knowledge that I have learnt from the Gridworld. Also, utilizing DynaQ here allows the program to learn from both planning and learning, by learning and experiencing previous encountered states, and planning by taking randomly sampling from those experienced states. Also, the Q-learning algorithm that I have utilized is an off-policy temporal difference control, which is more efficient than SARSA in the sense that SARSA edges the agent towards greediness while Q-learning estimate the optimal value function directly by observing the subsequent state and reward [2]. As seen from the graph shown above, Dyna-Q has also allowed the agent to capitalize off of limited experience and reach the end goal with the minimal number of actions.

4.2 Deficiencies

The deficiency of this project would be that the agent has a very limited amount of actions to learn, and thus could achieve convergence fairly quickly, making it more on the simple side. As shown in the graph of the action vs number of episodes, it took less than 1000 episodes for the agent to learn through Q-learning the minimum number of actions. Furthermore, the heatmap, while showing how good it is to be in a particular state, does not really explain the steps needed from that state to quickly finish the game (i.e Q values). There could have been a better way of representing the snakes and ladders as well.

5. Conclusion

To summarize, the agent was successfully taught to achieve the fastest way to clear the game of Snakes & Ladders. Utilizing Q-learning and DynaQ were good methods to help the agent learn faster, but there could have been clearer ways to represent the data, as well as making the agent learnt a more complicated set of actions than just choosing the number of steps. However, the graphs shown do prove that the board game of Snakes & Ladders could be easily learnt by the agent. Potential ideas for this game could be using the agent taught to challenge human players to see who could reach the end goal faster.

6. Links for Final Project

Youtube link for Final Project: https://youtu.be/GnWgaLKXe6k
Github link for Final Project: https://github.com/yongyx/COGS182-Reinforcement_Learning

References

[1] Dan Acosta-Kane. *HW-8 Skeleton Code*. 2020.
[2] Richard S. Sutton and Andrew G. Barto. "Reinforcement Learning, An Introduction". In: London, England: The MIT Press, 2006.