Reinforcement Learning: NaVigating a MaZe

Emeka Oguike

November 12, 2021

1 Introduction

If you've ever played Age of Empires you surely know that you start with a blank map. The first thing you do is to send units in every direction to scout the terrain and discover the enemy posts, strategic locations and resources such as mines and forests. What you are actually doing is in fact building a map/model for the world you are in. After creating this map, whether partially or completely, you start planning your moves and prepare for your upcoming battles. Planning here means simulating actions in your head without really executing them in the environment of the game.

Lacking the prior knowledge that humans bring to new situations and environments, reinforcement learning (RL) approaches usually need to explore many more policies than a human would before finding an optimal one. Model-based RL attempts to overcome the issue of a lack of prior knowledge by enabling the agent to construct a functional representation of its environment.

One weakness of the above approach is that it seems to suggest that a fairly accurate model needs to be learned over the entire domain to learn a good policy. In particular, models may be incorrect because the environment is stochastic and only a limited number of samples have been observed, or simply because the environment has changed and its new behavior has not yet been observed. When the model is incorrect, the planning process is likely to compute a sub-optimal policy.

2 Problem Statement

In this homework, we explore some RL techniques for dealing with such challenges in a maze problem whereby an agent has to navigate its way to G in a grid world with obstacles.

A maze presents a unique setting for illustrating this relatively minor kind of modeling error and recovery from it. Initially, there is a short path from start to goal, to the right of the barrier, as shown in Figure 1. After 1000 time steps,

the short path is blocked and a longer path is opened up along the left-hand side of the barrier, as shown in Figure 2.

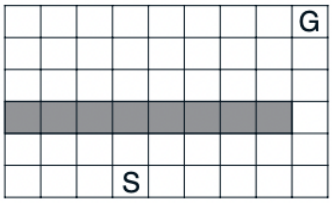

Figure 1

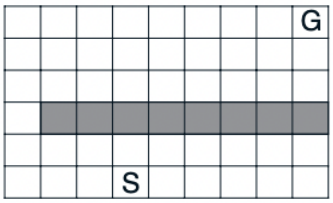

Figure 2

The Dyna-Q+ agent that did solve the shortcut maze uses and interesting heuristic. The agent keeps track for each state–action pair of how many time steps have elapsed since the pair was last tried in a real interaction with the environment. The more time that has elapsed, the more likely that the dynamics of this pair has changed and that the model of it is incorrect. To encourage behavior that tests long-untried actions, a revised reward—bonus reward—is given on simulated experiences involving these actions. For example, if the modeled reward for a transition is r, and the transition has not been tried in τ time steps, then planning updates are done as if that transition produced a reward of $r + k\sqrt{\tau}$, for some small k. This encourages the agent to keep testing all accessible state transitions and even to find long sequences of actions in order to carry out such tests.

The exploration bonus described above actually changes the estimated values of states and actions. Is this necessary? Suppose the bonus $k\sqrt{\tau}$ was used not in updates, but solely in action selection. That is, suppose the action selected was always that for which $Q(S_t, a) + k\sqrt{\tau(S_t, a)}$ was maximal. Carry out a grid world experiment that tests and illustrates the strengths and weaknesses of this alternate approach.

3 Experiment

The problem described above was simulated in a python environment. The maze was implemented as a character array where cells with the character 'B' denotes the boundaries, 'S' the starting position, 'G' the cell value of the goal, and 't' the cells that the learning agent is allowed to traverse. The agent, denoted as 'A' in the maze, has to navigate from the starting cell to the goal cell. After 1000 steps the the location of the boundary within the maze changes. The agent gets no reward until it reach the goal cell upon which it gets a reward of 5 points if the agent attempts to cross any of the boundaries within or bordering the maze, it gets a reward of 0 and is returned to its previous valid position.

The python environment tests and compares two algorithms: the DynaQ+ algorithm where the bonus reward changes the estimated values of states and the 'Alternate Method' whereby actions are selected that maximize the quantity: $Q(S_t, a) + k\sqrt{\tau(S_t, a)}$

The default/unvaried parameters for the learning agents in this experiment

were the ϵ with a value of 0.2; the constant for the bonus reward with a value of 0.1; and the initial values for state-action pairs with a value of 0.

The outputs if the experiment are the plot of the number of steps per episode vs number of episodes, and the plot cumulative reward vs number of steps.

4 hypothesis

In one algorithm, DynaQ+, the bonus reward only applies in the indirect reinforcement learning, or model learning, part of the method. While in the other algorithm, here denoted as the 'Alternate Method', the bonus reward applies to both the direct reinforcement learning and indirect reinforcement learning aspects of the method. As a result I expect the alternate method to react to changes in the environment more quickly, and more efficiently (fewest number of steps to reach the goal).

5 Results

Figure 1 depicts the plot of the number of steps per episode vs the number of episodes for the Alternate Method for three different magnitudes for the planning step. Figure 2, on the other hand, depicts the plot of the number of steps per episode vs the number of episodes for DynaQ+ for three different magnitudes for the planning step. The values for discount and α (learning rate) of 0.4 and 0.2 respectively. From these plots we can see that both methods get better as the number of planning steps increases; i.e the number of steps required to solve the maze in the initial episodes gets smaller as the number of planning steps increases. However, the Alternate Method solves improves more drastically as the number of steps increases. At n values of 10 and 50, we can see that the number of steps in the early episodes approximates the steady state values.

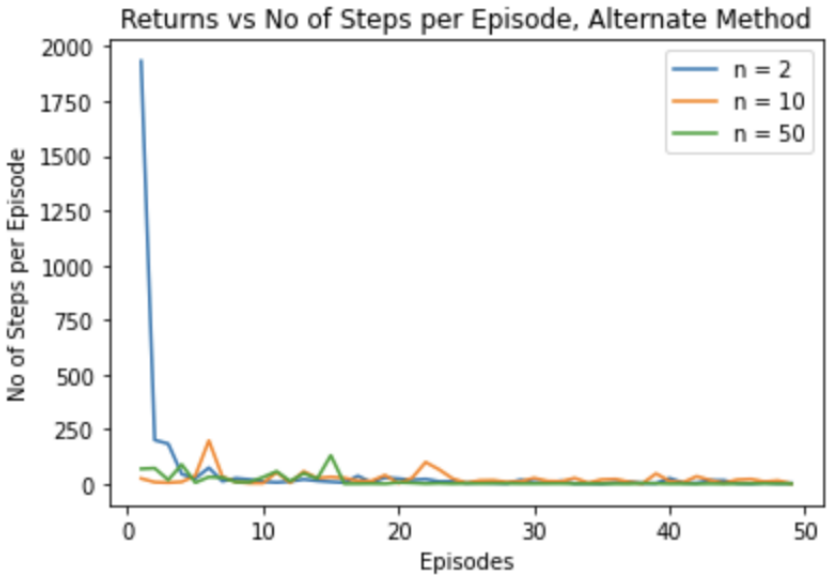

Figure 1

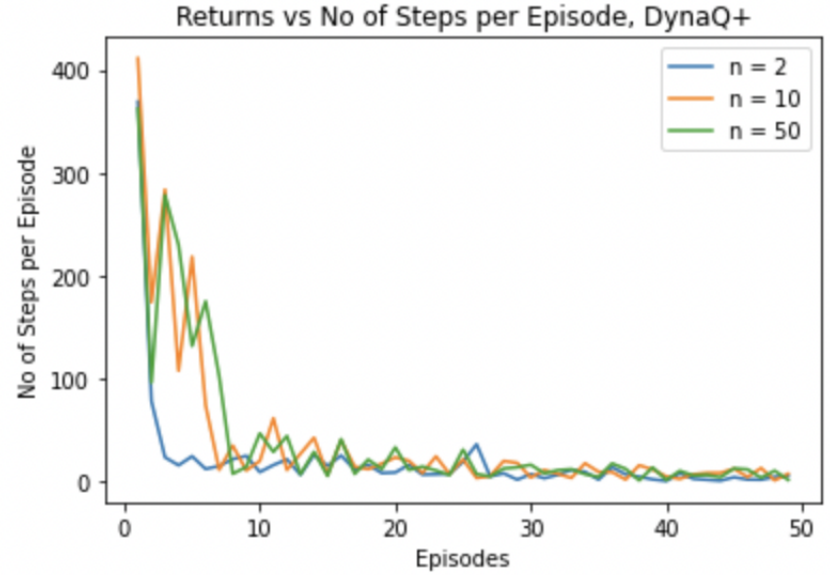

Figure 2

Figure 3 depicts the plot of the cumulative reward vs the number of steps for the Alternate Method for three different magnitudes for the planning step. Figure 4, on the other hand, depicts the plot of the cumulative reward vs the number of steps for DynaQ+ for three different magnitudes for the planning

step. The values for discount and α (learning rate) of 0.4 and 0.2 respectively. Default values were used for other parameters. From these plots we can see that, agents generally get more reward in fewer time steps as the number of planning steps increases. For both algorithms, there wasn't much improvement between planning steps of 10 and 50. For the Alternate method we can see that it reaches higher reward values in fewer steps than the DynaQ+. With lower planning steps (of size 2) the DynaQ+ agent performs better than other models in the long-run. This because the maze doesn't vary ever in the long run. However in the initial steps, when the maze changes, it performs more poorly than other agents other agents

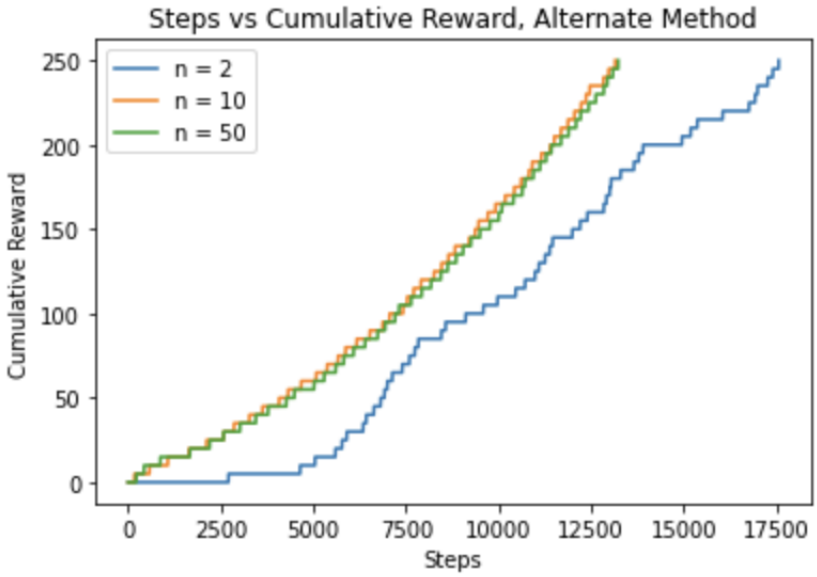

Figure 3

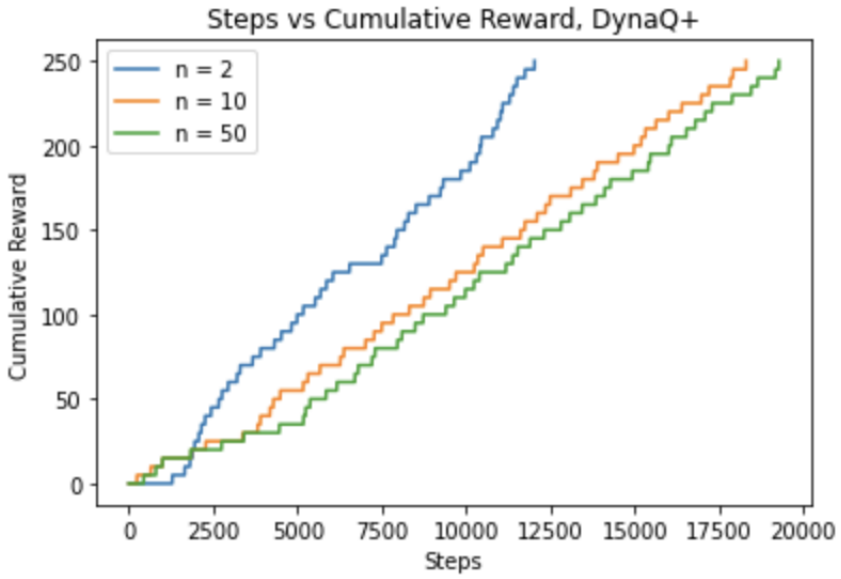

Figure 4

An additional comparison was done for both algorithms with a planning step size of 10 but with lower learning rates and higher discount values, and default values were used for other parameters. The results of this comparison is depicted in Figures 5 and 6. From Figure 5 we can see that both algorithms do not benefit from capitalizing on long-term rewards. More importantly, I noticed that agents (operating under DynaQ+ or the Alternate Method) with higher discount values performed more poorly compared to agents with lower discount values.

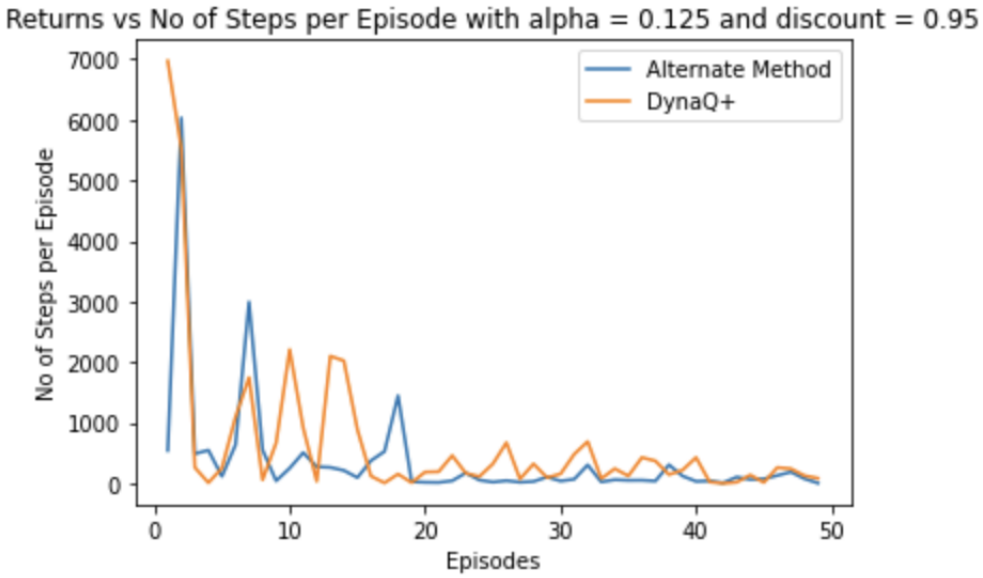

Figure 5

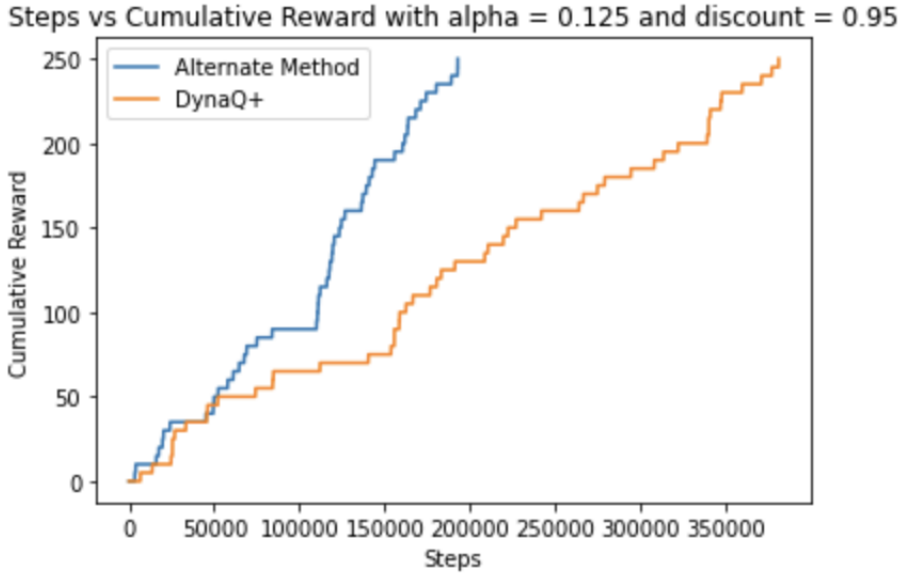

Figure 6

6 Experimental Scenario

Some experimentation was done to determine how both algorithms react to additional uncertainty introduced into the agent's model. With a probability of 0.5, the agents actions had no effect on its current state. Hence the model sometimes registers the agent's proceeding, after taking some valid action, as the agent's original state. The results are depicted in figures 7 and 8. The values for discount, α (learning rate), and number of planning steps were of 0.4, 0.2 and 10 respectively. Default values were used for other parameters. While the performances of both model are comparable in terms of cumulative reward, we can see that the Alternate Method handles this uncertainty much better in the initial episodes of the run.

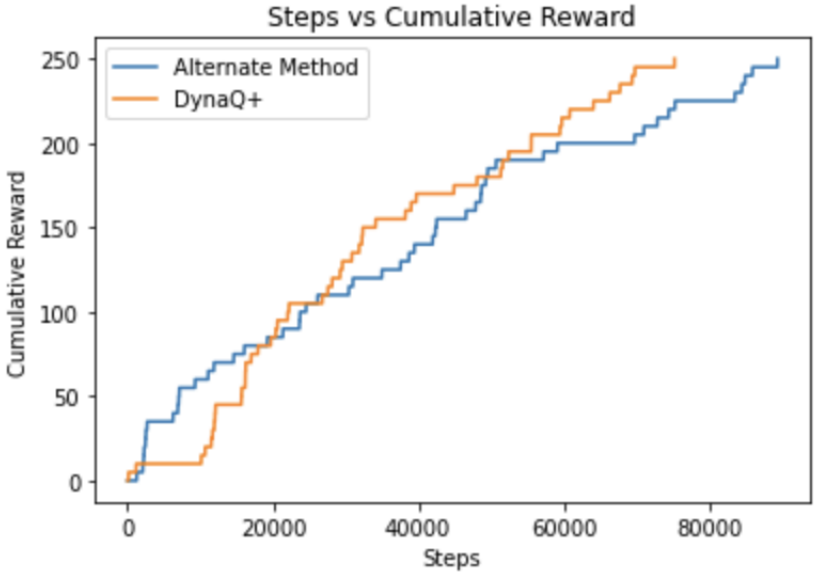

Figure 7

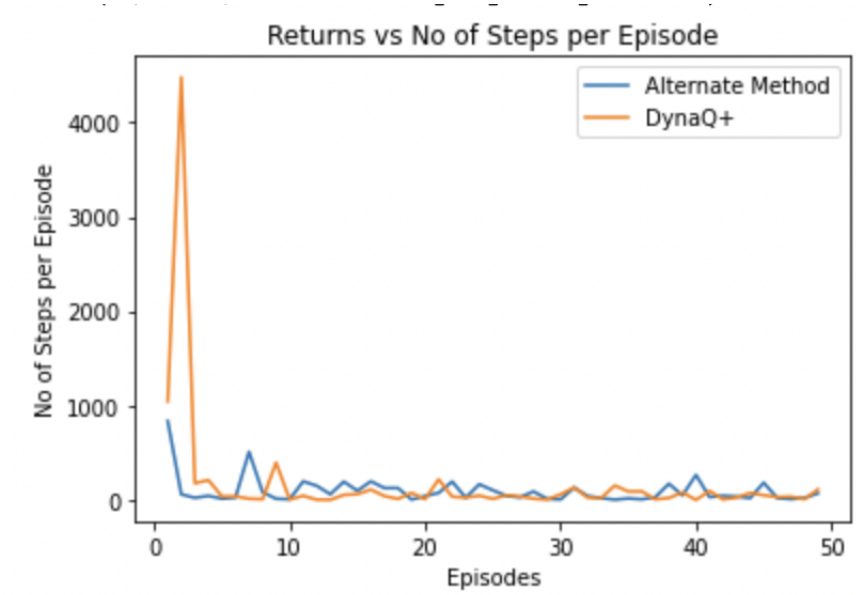

Figure 8

7 Conclusion

From the results of this simulation, we can see that the Alternate method performs better that DynaQ+ especially in the initial episodes. As a result, the Alternate methods an finds optimal strategy in the fewest steps. However, the Alternate method requires more computational resources as I had to update the bonus reward for all possible state-action pairs. Whereas in the DynaQ+, I had to update the bonus reward for state-action pairs involved in simulation by the agent's model. Consequently, I will recommend DynaQ+ whenever the state-action space is really large.

8 Extra Credit

8.1 Exercise 8.1

The Dyna algorithm offers two main benefits. Firstly, it updates sizeable set of values in each training episode, and, secondly, it exploits the both direct reinforcement learning (learning from direct interaction with the learning environment) and indirect reinforcement learning (learning from simulated experience). The multi-step bootstrapping methods from Chapter 7 only offer the first benefit that Dyna offers. However, unlike Dyna, the multi-step bootstrapping methods do not offer the opportunity of both direct and indirect reinforcement learning. Consequently, I would not expect one of the multi-step bootstrapping methods from Chapter 7 could do as well as the Dyna method.

8.2 Exercise 8.2

Dyna-Q+ has a value system that prioritizes both actions that have high values according to previous experience and actions that haven't been chosen for a long time. In the first phase, initially, Dyna simply randomly chooses actions, but Dyna-Q+ tend to try new actions or the effective old actions. In the second phase, Dyna-Q+ is more exploratory. For this reason, Dyna-Q+ outperforms Dyna in both phases.