Chapter 8

Exercise 8.1:

In any N-Step TD method, values of N states will be updated at the end of the first episode. These N states will be the last N states that occur in the trajectory of 1st episode. So a N-Step method without learning would certainly be better than the one step method without learning. After the second episode at best, a total of 2*N last states will be updated and so on. One thing to note here is that in N-Step TD methods, updates from one episode are only propagated in the next episode. Thus the updates are 'delayed'. With Dyna-Q however, with sufficient number of planning steps, updates from an episode are propagated quickly to nearby states and further on. In this sense, Dyna-Q is better than the N-Step TD method even though the N-Step TD solves the problem faced by one-Step TD method. This is a major advantage because this allows the planning portion of Dyna-Q to run in parallel with the sampling steps thereby making Dyna-Q more computationally efficient.

Exercise 8.2:

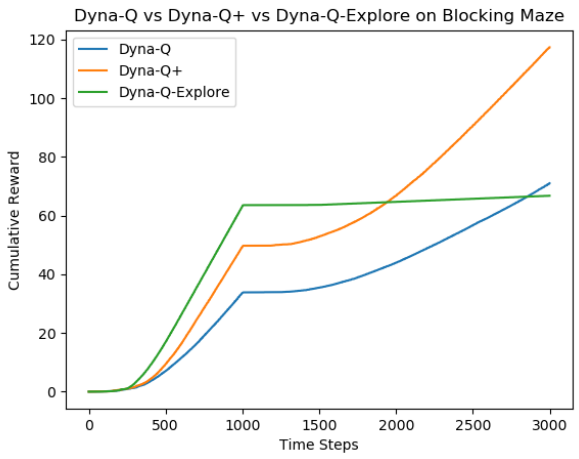
The Dyna-Q+ algorithm promotes exploration by giving additional reward for exploring. Dyna-Q+ thus explores more aggressively than a normal $\varepsilon - greedy$ method. This is why Dyna-Q+ is able to find the optimal policy faster than Dyna-Q and thus performs better in both phases of the given examples.

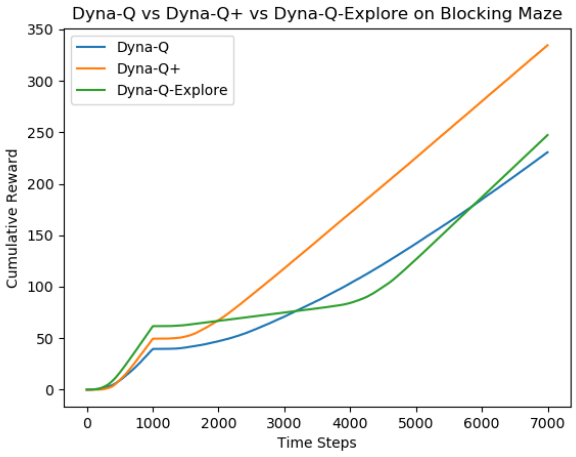
Exercise 8.3:

Dyna-Q+ promotes exploration more aggressively than Dyna-Q. This leads to faster discovery of optimal policy which brings a gap between the curves of the two methods. However, once the optimal policy is discovered, Dyna-Q+ continues to explore aggressively which causes it to pick non optimal trajectories leading to a lower reward. While Dyna-Q follows the greedy trajectory more often and thus the gap between the two curves narrows.

Exercise 8.4 (Programming):

Code: Check Github
Result:





Let's call the algorithm described in the exercise as Dyna-Q-Explore(for no specific reason). The same Blocking maze example has been used here. In the left figure, it can be seen that the Dyna-Q-Explore method is doing even better than the Dyna-Q+ method. This is expected because

the new method uses the normal rewards while still maintaining exploration.

Interesting things happen after the shift is made at 1000 time steps. First, it seems like the Dyna-Q-Explore method is struggling a lot and is not able to find the new path at all. But the right figure shows that after a large number of time steps, it is actually able to find the new path.

The reason for this is that Dyna-Q-Explore promotes exploration based on time and not on rewards. This is the fundamental difference between Dyna-Q+ and Dyna-Q-Explore. In Dyna-Q+ the actual reward is changed, which leads to change in the state-action value. This change then promotes further exploration from that state. Thus, Dyna-Q+ promotes exploration over a long path. Since Dyna-Q-Explore promotes exploration by time, it is not ensured that the exploration will happen over a long path leading to the discovery of a new path. Thus Dyna-Q-Explore's response to change in the maze is much slower.

Exercise 8.5:

To handle stochastic environments, some changes need to be made in the planning part of the algorithm. For each State, Action pair encountered, instead of simply storing the last reward, next_state observed, we should store a list of all reward, next_state pairs and how many times they are observed and make expectation based updates during planning. This is because in a stochastic environment, taking an action at a state might lead to different states and different rewards all with different probabilities. Effectively, we change the predicted model to an estimate of the actual distributed model (that is the environment).

However, this will not work in a changing environment because all the dynamics seen before the change will be irrelevant to the new environment. To fix this, we can form an estimation of the environment based on only the most recent actual experiences with the environment while discarding all old experiences (older than a set threshold). Such a planning structure would successfully detect a change in the environment while handling the stochastic property.

Exercise 8.6:

If some states are much more likely to occur than most, it will add to the advantages of sampling updates. In this case, expectation updates would still perform 'b' number of updates which will be computationally expensive. Though expectation based updates will produce the exact value of the state it will take 'b' steps to do so. This is not required here because since only a few states have high probability of occurrence, the contribution of most of the states would be very less. Sampling based updates, which interact with the environment, would see more and more of the states which contribute more to the value and form a close estimate of the value function much faster than the expectation based update.

Exercise 8.7:

It is scalloped for $b = 1$ since the number of states is 1000 and b is only 1 so in a uniform distribution the order in which the states are updated will matter. Since b is 1, each state depends completely on one other state. For higher value of b, each state value depends on multiple states. So for b=1 it will take several sweeps for the uniform case to reach the

same value as on-policy. For 10,000 states, it will be even more scalloped.

Exercise 8.8 (Programming):

Code: Check Github
Results:

As it can be seen in the figure, The Uniform method for b=3 is able to reach the same value as on-policy quicker than that for b=1. This supports the answer to exercise 8.7.