Reinforcement Learning Homework 03

Qiu Yihang

April 2023

1 Policy Gradient

Proof. We know $\sum_{a\in\mathcal{A}} \pi(a|s) = 1$, i.e. $\dfrac{\partial \sum_{a\in\mathcal{A}}}{\partial\theta} = 0$. Thus,

$$\begin{aligned}
\text{LHS} &= \sum_{s\in\mathcal{S}} \rho^{\pi_\theta}(s) \sum_{a\in\mathcal{A}} \pi_\theta(a|s) \frac{\partial \log \pi_\theta(a|s)}{\partial\theta} f(s) \\
&= \sum_{s\in\mathcal{S}} \rho^{\pi_\theta}(s) \sum_{a\in\mathcal{A}} \pi_\theta(a|s) \frac{1}{\pi_\theta(a|s)} \frac{\partial \pi_\theta(a|s)}{\partial\theta} f(s) \\
&= \sum_{s\in\mathcal{S}} \rho^{\pi_\theta}(s) \sum_{a\in\mathcal{A}} \frac{\partial \pi_\theta(a|s)}{\partial\theta} f(s) = \sum_{s\in\mathcal{S}} \rho^{\pi_\theta}(s) \frac{\partial \sum_{a\in\mathcal{A}} \pi_\theta(a|s)}{\partial\theta} f(s) = 0.
\end{aligned}$$ ■

2 Implementation of the Dyna-Q and Dyna-Q+ Algorithms

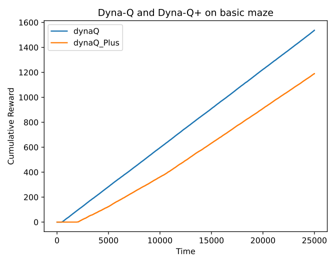
2.0 Tricks in the Implementation of Dyna-Q+

Before we move on to discuss the impacts of the number of planning steps and the differences between Dyna-Q and Dyna-Q+, I would like to talk about the implementation of Dyna-Q+ first.

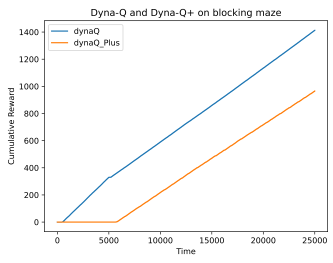
We figure out that compared with Dyna-Q, Dyna-Q+ algorithm needs more time to explore and learn before taking a good action. The performances without any initialization is shown in Fig.1.

It is in fact explainable. Dyna-Q and Dyna-Q+ only explore visited states along with the past actions at these states. Meanwhile, with the additional reward, Dyna-Q+ is more likely to explore the states that was once visited but have not been revisited for a long time while it is less likely to explore unvisited states and unperformed actions. Therefore, it takes longer time for Dyna-Q+ to explore and find the optimal policy in static environments, which is supported by Fig.1(g), 1(h), and 1(i).

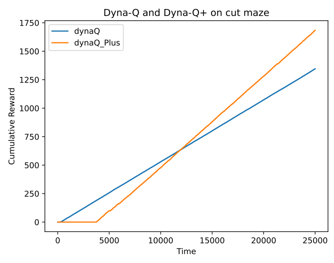
Note that in Fig.1(h) and 1(i), in order to show that Dyna-Q+ is able to find the optimal policy eventually, we set the time when blocking and shortcut maze changes to be $T = 20000$.



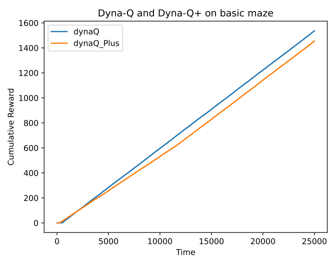
(a) w.o. init



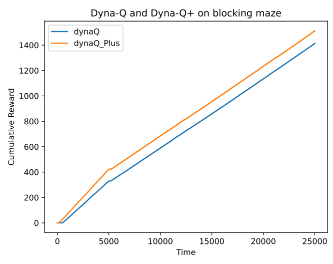
(b) w.o. init



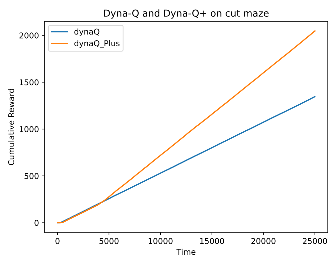
(c) w.o. init



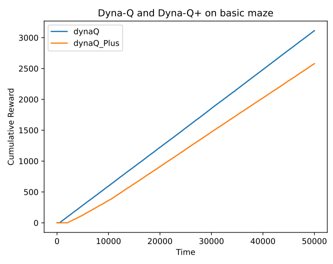
(d) w. init, seed=666



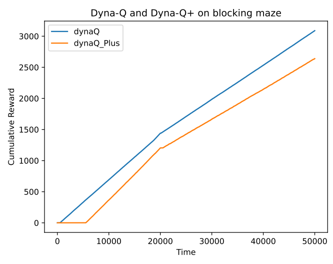
(e) w. init, seed=666



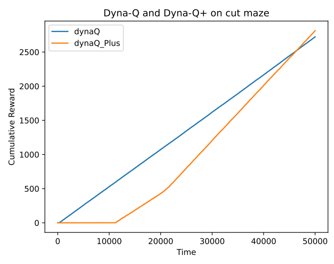
(f) w. init, seed=666



(g) w.o. init, time=50000



(h) w.o. init, changes at $T = 20000$



(i) w.o. init, changes at $T = 20000$

Figure 1: Performances of Dyna-Q+ without initialization

We notice that when environment changes, it seems that Dyna-Q+ spends shorter time on finding the optimal policy in the new environment than it spends on the attempts to find the optimal policy in the original environment. We suppose it is the exploration on the original environment that helps Dyna-Q+ to find the optimal policy on the new environment.

Based on the above observations, we use a small trick to help Dyna-Q+ to find the optimal policy faster in our implementation. We initialize the model in Dyna-Q+ in the following way. **The main idea is to initialize the last visit time of all possible states and actions.**

$$\texttt{model}(s, a) = \{\texttt{reward} = 0, \texttt{next\_state} = s, \texttt{last\_visit\_time} = T_0\} \qquad (T_0 \leq 0)$$

With the initialization, during the early period, with the help of additional reward $\kappa\sqrt{\tau}$, Q-planning would assign greater Q-value for these unvisited states and unperformed actions, leading to more exploration in Q-learning. In this way, Dyna-Q+ would find the optimal policy faster.

Note that we actually initialize with a faulty observation of the next state and reward. We could interpret the initialization in the following way. The environment used to be the faulty environment, the one we use in initialization, in the past (before time 0) and Dyna-Q+ has already observed the environment correctly at time $T_0 \leq 0$. At time 0, the environment suddenly changes to the real environment and Dyna-Q+ starts to collect correct observations.

Based on the above analyses, it is recommended to assign T_0 with some negative integers, -100 for example, since the initialization should be regarded as observations long before.

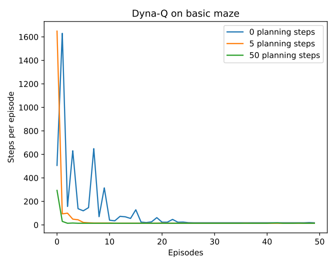
Moreover, T_0 should be a hyperparameter. Nevertheless, altering code without marker *YOUR CODE HERE* is not allowed. Thus, we set $T_0 = 0$ in our implementation.

To be fair in the comparison, we also implement this initialization trick in Dyna-Q algorithms when comparing the performances between Dyna-Q and Dyna-Q+.

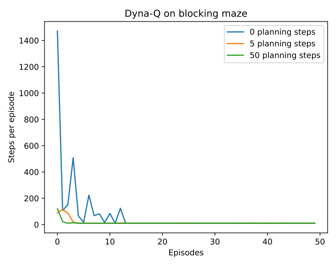
Fig.1(d), 1(e), and 1(f) have shown the effectiveness of this initialization trick. Setting hyperparameter T_0 to a negative integer would result in better performances, though.

2.1 The Impacts of the Number of Planning Steps

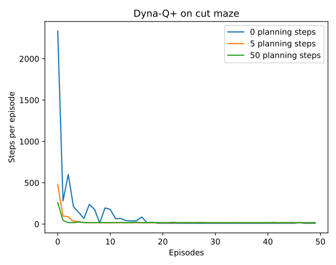
In this section, we do not apply the initialization trick mentioned in **2.0**. The performances of Dyna-Q and Dyna-Q+ with different planning steps in three different environments are shown in Fig.2.



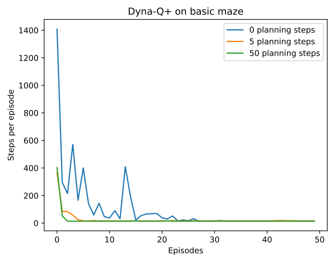
(a)



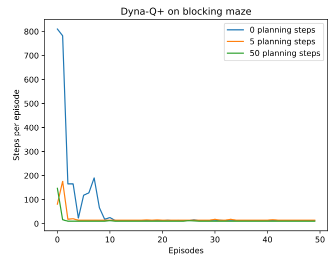
(b)



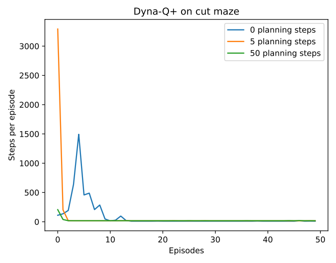
(c)



(d)



(e)



(f)

Figure 2: Dyna-Q and Dyna-Q+ under different planning steps in three different environments.

Note that for dynamic environments, i.e. blocking and shortcut maze, this section of experiments only consider the maze at time 0, which is in fact a static environment.

It is plain to see that **the larger** the number of planning steps, **the less fluctuated** the performance, both for Dyna-Q and Dyna-Q+.

Meanwhile, at first more planning steps might lead to more steps per episode. However, after a few iterations, which is only one iteration in most cases, steps per episode with more planning steps will decrease greatly. Also, Dyna-Q and Dyna-Q+ with more planning steps **converges to the optimal policy faster** than those with less planning steps.

The reason is that more planning steps helps the agent to simulate the environment more times based on past observations, leading to a more accurate model of the environment. Thus, the agent can learn faster and converge to the optimum faster.

2.2 Differences between Dyna-Q and Dyna-Q+

In this section, to be fair, we implement the initialization trick in both Dyna-Q and Dyna-Q+.

The performances of Dyna-Q and Dyna-Q+ in three different environments under different random seeds and with or without initialization tricks are shown in Fig.3.

2.2.1 Basic Environment (Simple Environment)

On the simple maze, which is a static environment, the cumulative reward of Dyna-Q+ **is smaller than** Dyna-Q in most cases.

The application of initialization helps to decrease the difference, but Dyna-Q+ with initialization **is still slightly smaller** in cumulative reward than Dyna-Q with initialization.

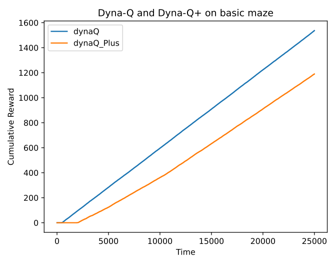
The reason might be that Dyna-Q+ tends to visit states that have not been revisited for a long time, which might waste some time compared with Dyna-Q, who sticks to the optimal path found before.

2.2.2 Blocking Environment (Blocking Maze)

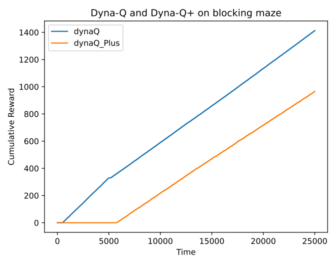
On the blocking maze, since the original path is blocked for both Dyna-Q and Dyna-Q+, both algorithms tends to find the new path.

With initialization, Dyna-Q+ can find the new path **no slower than** Dyna-Q. In the case when random seed is 5454122 (shown in Fig.3(j), (k), (l), (m), (n), and (o)), Dyna-Q+ finds the new path **even faster** than Dyna-Q, with or without initialization.

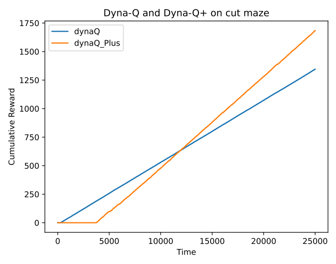
However, without initialization, in most cases Dyna-Q+ **is slightly slower than Dyna-Q** in finding the new path in most cases. This is because Dyna-Q+ tends to explore more states that have not been revisited for a long time and requires more time to find the optimal policy.



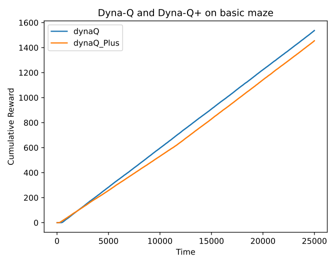
(a) w.o. init, seed=666



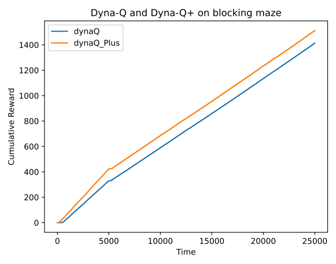
(b) w.o. init, seed=666



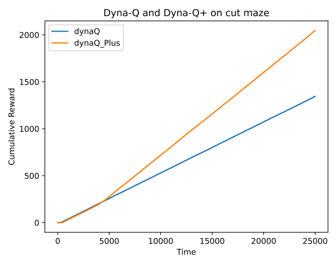
(c) w.o. init, seed=666



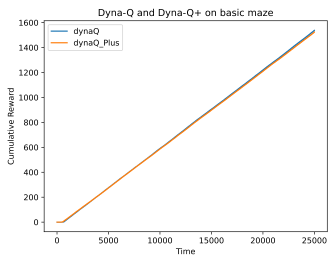
(d) w. init, seed=666



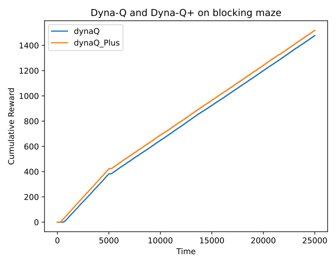
(e) w. init, seed=666



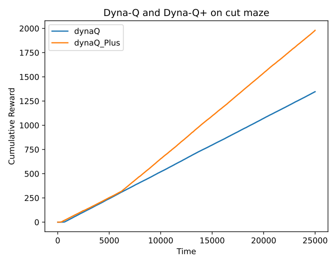
(f) w. init, seed=666



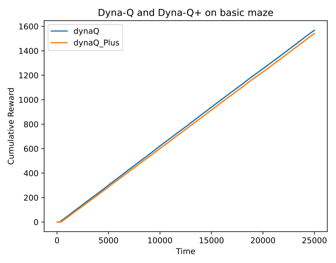
(g) w. init, seed=20230404



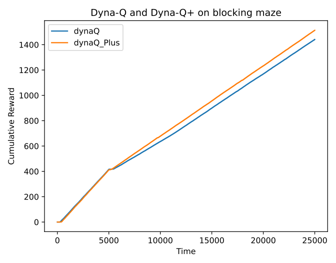
(h) w. init, seed=20230404



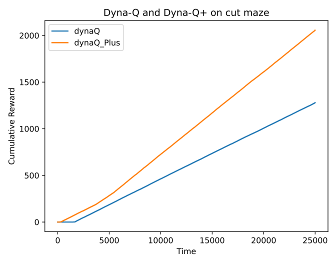
(i) w. init, seed=20230404



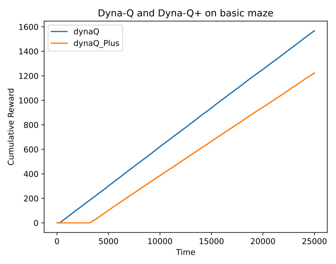
(j) w. init, seed=5454122



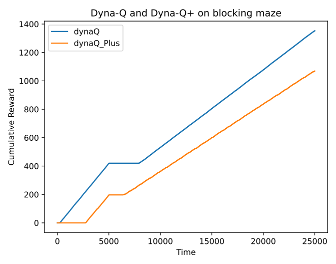
(k) w. init, seed=5454122



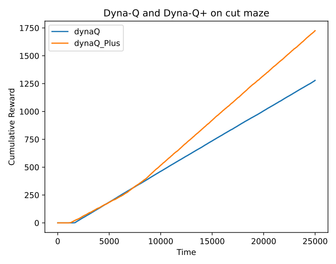
(l) w. init, seed=5454122



(m) w.o. init, seed=5454122



(n) w.o. init, seed=5454122



(o) w.o. init, seed=5454122

Figure 3: Performances of Dyna-Q and Dyna-Q+ on different random seeds and w./w.o. initialization.

2.2.3 Shortcut Environment (Shortcut Maze)

On the shortcut maze, it is plain to see that Dyna-Q+, with or without initialization, **can find the shortcut** while Dyna-Q could not.

The explanation for this is that Dyna-Q+ tends to explore states that have not been revisited for a long time, which helps it to find the shortcut. On the other hand, Dyna-Q tends to stick to the optimal path found before, which makes it unable to find the shortcut.