AI 3603 Artificial Intelligence: Principles and Techniques

By: Kaiwen Zhu 520030910178

HW#: 2

November 14, 2022

1 Reinforcement Learning in Cliff-walking Environment[1]

1.1 Analysis of One Learning Process

1.1.1 Q-Learning

Principle

The learning goal is the Q-table, i.e., the utility $Q(s, a)$ we will ultimately obtain when choosing action a from state s. If the transition probability T and reward function R are available, then we can simply compute $Q(s, a)$ by

$$Q_{k+1}(s, a) = \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma \max_{a'} Q_k(s', a')]$$

through iterations, where $\gamma \in (0, 1]$ is the discounting factor.

But in this environment, T and R are unknown, so we need to estimate. Each time the agent transitions from s to s' taking action a and getting a reward r, we compute the sample value

$$sample = r + \gamma \max_{a'} Q(s', a') \tag{1}$$

and incorporate it into the weighted running average of $Q(s, a)$ by

$$Q_{k+1}(s, a) = (1 - \alpha)Q_k(s, a) + \alpha \cdot sample$$

where $\alpha \in (0, 1)$ is the weight of samples.

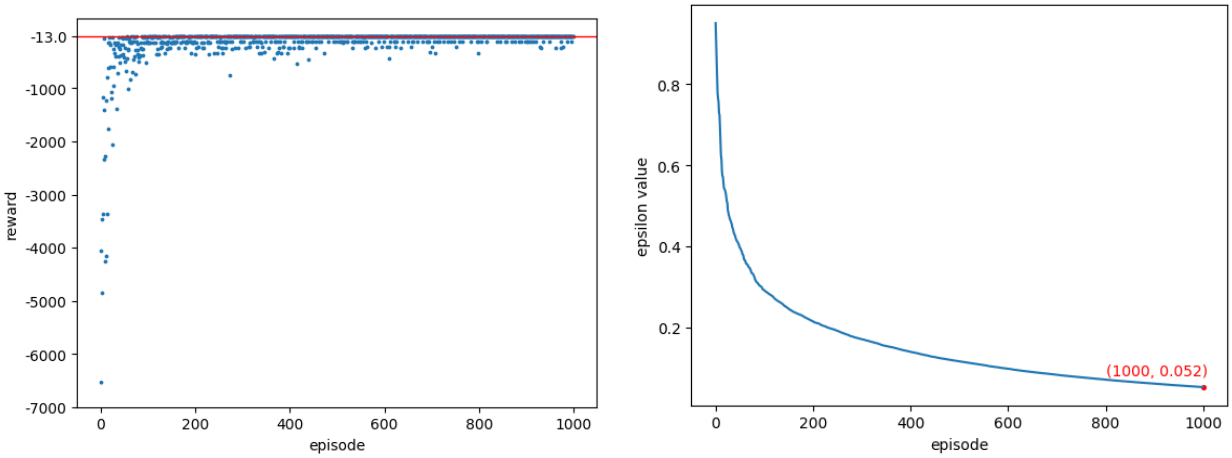
As for making actions, *exploration* and *exploitation* are introduced. Exploration means acting randomly to gather more environmental information, while exploitation means choosing the action with the largest utility according to existing Q-values.

The remaining problem is when to explore and when to exploit. ϵ-greedy policy regulates that when making an action, the agent chooses to explore with probability ϵ and to exploit with the probability $1 - \epsilon$. My implementation lets ϵ decrease by a certain ratio *decay_rate* after each iteration, so that in the early stage of learning is encouraged for more environmental information and later exploitation is encouraged for convergence.

Results

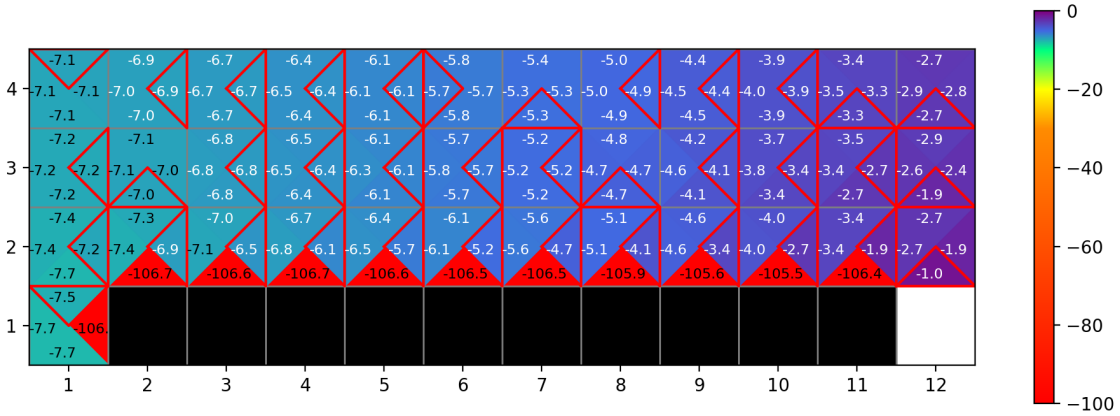
There are three hyper-parameters: the discounting factor γ, the weight of samples α and decay rate of ϵ *decay_rate*. I choose $\gamma = 0.9, \alpha = 0.1, decay\_rate = 0.9999$. The results are shown below. Figure 1 shows the rewards and ϵ in each episode and Figure 2 shows the final Q-table and path after training (in Figure 2a, the triangle with red border in each grid represents the optimal action in this state). Note that ϵ decays at each iteration, and in Figure 1b the ϵ value of one episode is represented by the value when this episode ends.

[1] The programs to plot all figures in this section are provided in file `plot.py`.

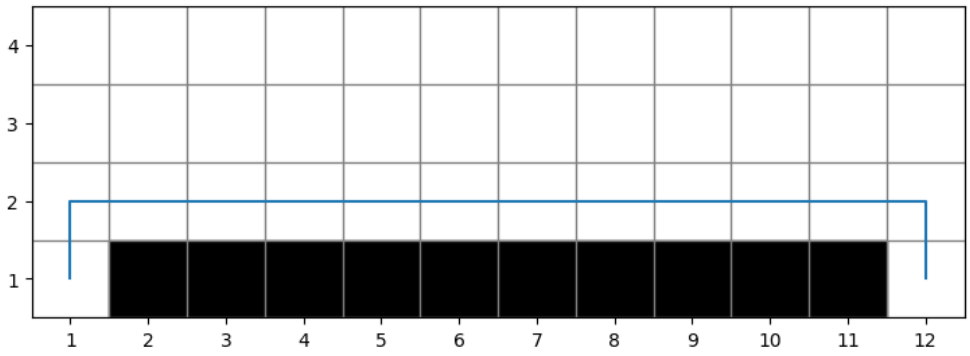


(a) reward (b) ϵ value

Figure 1: the reward and ϵ value in the learning process using Q-Learning



(a) Q-table



(b) path

Figure 2: the final Q-table and path learned using Q-Learning

Analysis on Results

The results convincingly demonstrate that the Q-table does converges to good values. For instance, $Q(24, 1)$ (state 24 corresponds to the grid $(1, 2)$ in Figure 2a and action 1 corresponds to moving rightward) has converged to about -7.2, and we know the true value of $Q(24, 1)$ should be

$$\sum_{i=0}^{11} \gamma^i(-1) \approx -7.2$$

because the transition is actually deterministic (the `info` returned by `env.step` is a dictionary with the key `prob` always being 1) and the reward is -100 for falling into the cliff and -1 otherwise.

With this good Q-table, the agent succeeds to find a shortest path which has the highest score (-13). Now let's take a look on how it converges.

Recall that during the learning process, $Q(s, a)$ are updated by

$$sample = r + \gamma \max_{a'} Q(s', a'), \quad Q_{k+1}(s, a) = (1 - \alpha)Q_k(s, a) + \alpha \cdot sample.$$

And the correct value is computed by iterations using

$$Q_{k+1}(s, a) = \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma \max_{a'} Q_k(s', a')].$$

From these equations, we can see if the value of *sample* is correct, i.e., $\max_{a'} Q(s', a')$ has converges correctly, and enough correct samples are incorporated in $Q(s, a)$, then $Q(s, a)$ will converge correctly. Therefore, when the Q-values in state s converges, then the state-action pairs transitioning to s will converge too, and this propagates from the end to the start. Figure 3 witnesses this trend, which shows the value of

- $Q(24, 1)$ (in grid (1,2), moving rightward),
- $Q(30, 1)$ (in grid (7,2), moving rightward),
- $Q(35, 2)$ (in grid (12,2), moving downward),
- $Q(24, 0)$ (in grid (1,2), moving upward),
- $Q(0, 1)$ (in grid (1,4), moving rightward),
- $Q(6, 1)$ (in grid (7,4), moving rightward),
- $Q(11, 2)$ (in grid (12, 4), moving downward)

during the training process. We can see that the closer the state-action pair to the destination (equivalently, the larger of the converged Q-value), the quickly its Q-value converges.

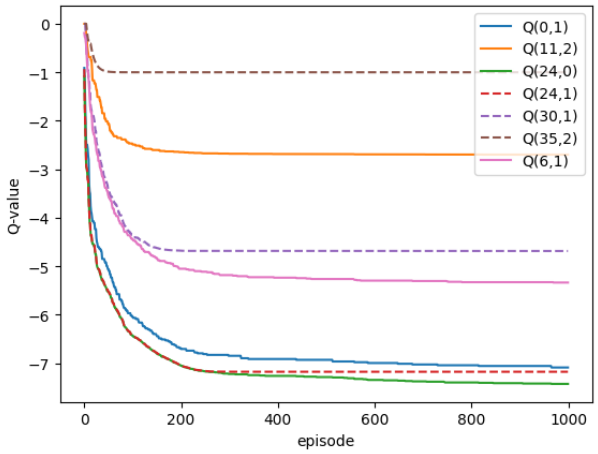


Figure 3: how Q-value's converges in Q-Learning

However, what should be admitted is that not all Q-values have converged correctly. Note that $Q(24, 1)$ has converged to -7.2, but $Q(0, 1)$, which is further from the destination, has converged to $-7.1 > -7.2$. Now focus on $Q(0, 1)$ in Figure 3, and we find that unlike the state-action pairs in the final path (those in dashed lines), $Q(0, 1)$ still decreases occasionally in the later stage, and it holds true for other state-actions pairs not in the final path (those in solid lines). This implies they have yet to converge to the ultimate true value. Indeed, as the ϵ has decayed to a very low value, it is not likely for them to converge even if more episodes are experienced since they rarely reach these states.

Despite this flaw, we can still regard this algorithm as decent, since these Q-values are slightly wrong just because these states are rarely (without exploration, maybe never) reached. As for the final path found, results are quite satisfying if we only consider the score.

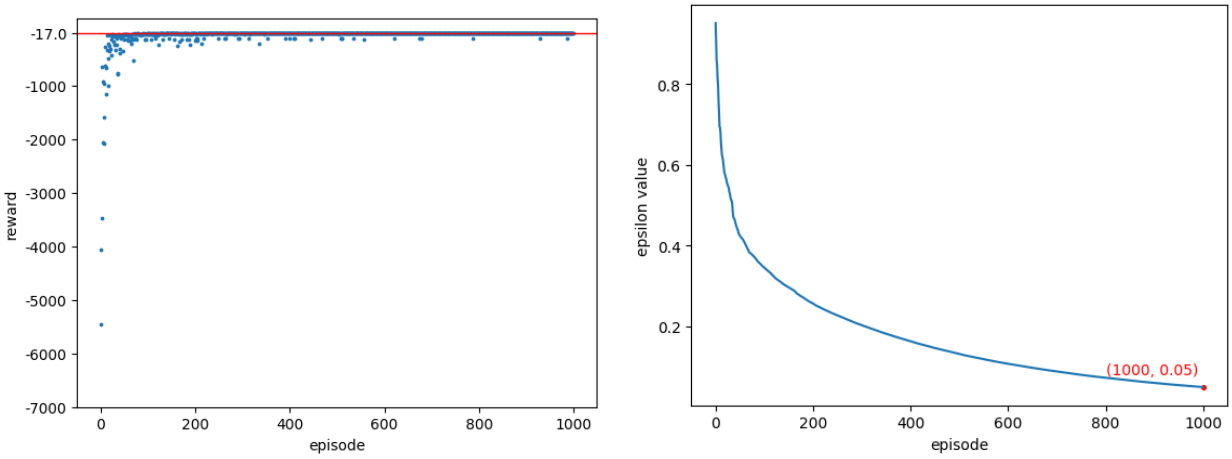
1.1.2 SARSA

Principle

The principle of SARSA is almost the same with that of Q-Learning, the only difference existing in Equation (1). The value of *sample* consists of two parts: the immediate reward r and the long-term reward discounted by γ. For the long-term reward, Q-Learning uses the highest Q-value in the next state, but not necessarily taking this action in the next state because of exploration and stochasticity of transition. In contrast, SARSA first chooses the action in the next state and then uses this Q-value as the long-term reward. That is, Q-Learning is more radical: it does not worries about the probability of not acting optimally next time; while SARSA is more conservative: it takes the risk in the next state into consideration.

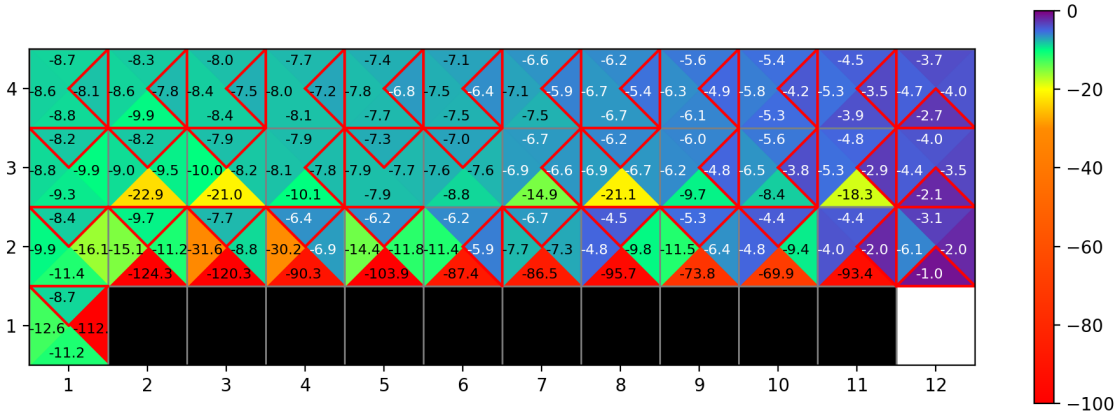
Results

The hyper-parameters I choose in implementation is the same as those in Q-Learning: $\gamma = 0.9, \alpha = 0.1, decay\_rate = 0.9999$, and the results are shown below.

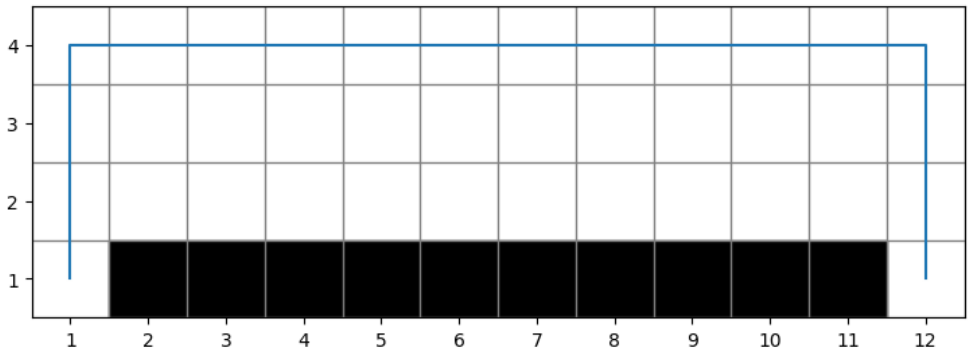


(a) reward (b) ϵ value

Figure 4: the reward and ϵ value in the learning process using SARSA



(a) Q-table



(b) path

Figure 5: the final Q-table and path learned using SARSA

Figure 4a shows the reward in each episode, Figure 4b shows the ϵ value in each episode, Figure 5a shows the final Q-table after 1000 episodes, and Figure 5b shows the path found by the agent after training.

Analysis on Results

If we still assume the correct Q-value is the reward the agent would get taking the shortest path from this state-action pair with future reward discounted (that is, if the shortest path to the destination after this action has length $n(s, a)$, then the reward should be

$$r(s, a, \gamma) = r + \sum_{i=1}^{n(s,a)} \gamma^i(-1),$$

where r is the immediate reward), then it's obvious that SARSA underestimates this reward, or equivalently, overestimates the penalty. Moreover, the extent of overestimation has a trend to be negatively correlated to the distance from the state-action pair to the cliff, except those directly falling into the cliff. With $\gamma = 0.9$, the rate of overestimation is show in Figure 6 (the rate is $[Q(s, a) - r(s, a, \gamma)]/r(s, a, \gamma)$).

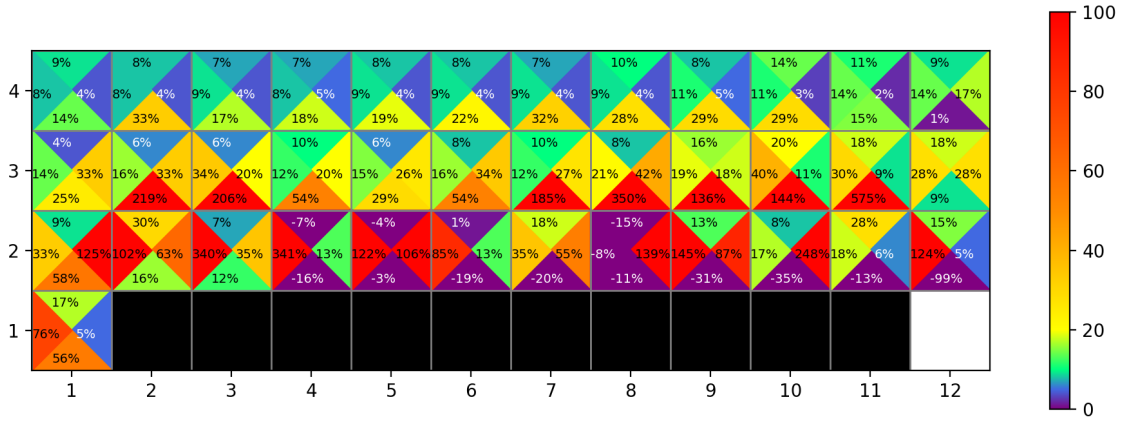


Figure 6: the rate of overestimation of penalty

The reason for such overestimation will be expounded later in the next section, now we first check the process of converging. Like what we did when analysing Q-Learning, the value of $Q(24, 1), Q(30, 1), Q(35, 2), Q(24, 0), Q(0, 1), Q(6, 1)$ and $Q(11, 2)$ are plotted in Figure 7, but we train 2000 episodes for better convergence. Those state-action pairs in the path found by SARSA are in solid lines, and those in the shortest path are in dashed lines.

In Figure 7, what is most noticeable is that unlike the monotonically decreasing values in Q-Learning, there exists oscillation here, especially for $Q(24, 1)$. Specifically, on occasions, an abnormal sample with very low value makes Q-value decrease sharply, and then subsequent normal samples reinstate this Q-value slowly, until the next abnormal low sample comes. And in general, the abnormal samples with low value exert more effects, making the converged Q-value lower. More details are discussed in the next section.

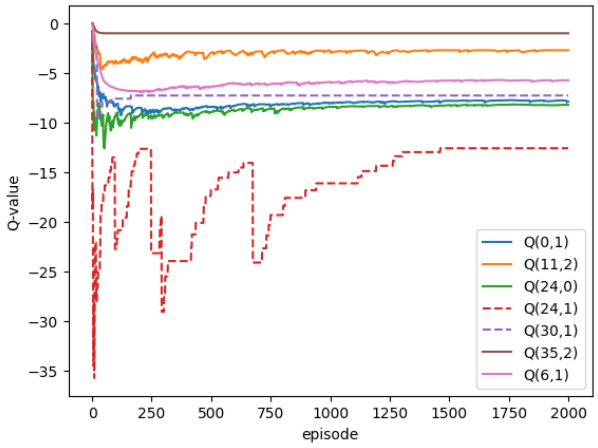


Figure 7: how Q-value's converges in SARSA

1.2 Analysis of Difference between Q-Learning and SARSA

As is clearly shown in Figure 2b and Figure 5b, the paths found by Q-Learning and SARSA are different: the path found by Q-Learning is shortest and thus has the highest score, but it is closely next to the cliff and thus not safe; the path found by SARSA is not the shortest, but it is farthest from the cliff and thus safest.

The differences in paths directly arise from the differences in Q-tables, as is shown in Figure 2a and Figure 5a: in Q-Learning Q-values converge to the rewards the agent would get taking the shortest path; while in SARSA, although it also tries to minimize the penalty, namely, to find a shorter path, it tends to overestimate the penalty a lot if the state-action pair is close to the cliff (dangerous state-action pairs).

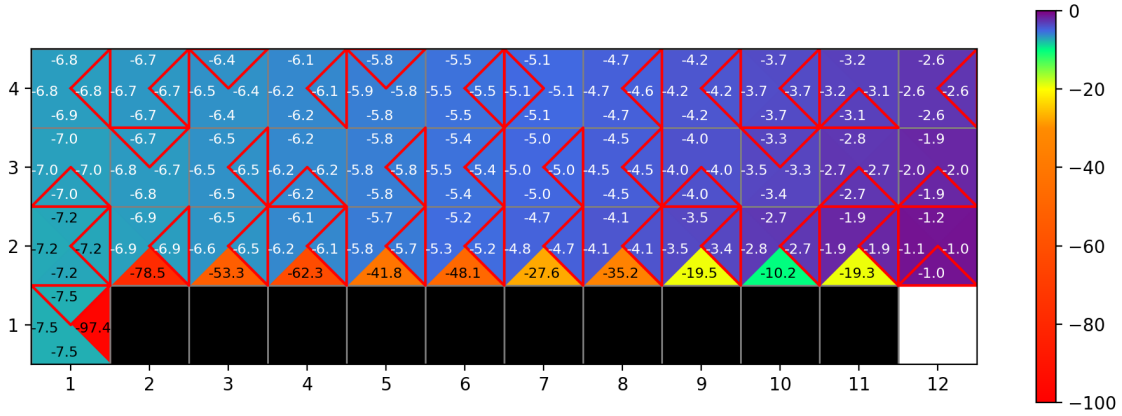
The analysis of Figure 7 tells us that the overestimation in dangerous state-action pairs is due to the abnormal samples with extremely low value. Now let's consider how such abnormal samples arise. The value of a sample consists of the immediate penalty and the long-term penalty. The immediate penalty is always -1 (we do not consider the actions leading immediate fall because they should be always avoided), so the key is the long-term penalty. This is exactly what SARSA varies in compared with Q-Learning.

Recall our explanation of the principle of SARSA: Q-Learning is more radical, as it does not worries about the probability of not acting optimally next time; while SARSA is more conservative, as it takes the risk in the next state into consideration. To be more specific, suppose the agent transitions to state s' from the state-action pair (s, a), then by the principle of SARSA, it first chooses the action a' from s'. If $Q(s', a') = \max_{a''} Q(s', a'')$, then the story goes the same as that in Q-Learning; but if $Q(s', a')$ happens to be very low due to exploration or stochasticity of transition, such as the situation where a' leads the agent to fall into the cliff, then an "abnormal sample" arises and remarkably brings $Q(s, a)$ down. Then the next time the agent reaches state s, it will think a is a bad action. Even if it takes a again and fortunately chooses the optimal action the next state, this normal sample cannot compensate for the loss brought by abnormal samples, as shown in Figure 7. (Note:

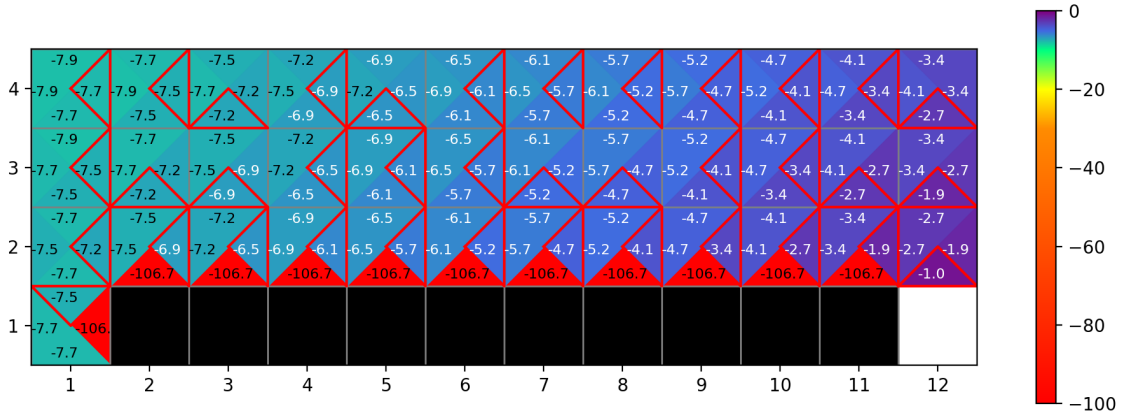
Now we can also explain why $Q(24, 1)$ oscillates more drastically than other pairs next to cliff like $Q(30, 1)$. This is because state 24 is reached every episode, and state 30 is rarely reached due to the low value of $Q(24, 1)$, so $Q(24, 1)$ has much more chances to incorporate abnormal samples.)

The above is the underlying causes of overestimation of penalty in dangerous state-action pairs. Such overestimation will propagate from dangerous pairs to less dangerous pairs, and the extent decreases in propagation because α and γ are less than 1. The final result is that the Q-value of state-action pairs far from the cliff is lower, thereby leading the agent to prefer these pairs.

From the above analysis, we can see the difference between Q-Learning and SARSA exists in that Q-Learning always maximizes the expectation of reward, while SARSA would rather obtain less reward safely than obtain the most reward with high risk. In essence, in bias-variance tradeoff, Q-Learning values bias more, while SARSA values variance more.



(a) decay rate = 0.999



(b) decay rate = 0.99999

Figure 8: the Q-tables learned with different decay rates, $\gamma = 0.9, \alpha = 0.1$ using Q-Learning

1.3 Sensitivity Analysis

Now we analyse the sensitivity of the two algorithms to the three hyper-parameters: the discounting factor γ, the weight of samples α and decay rate of ϵ *decay_rate*.

1.3.1 Q-Learning

Let $(\gamma, \alpha, decay\_rate)$ takes the value of each one of $\{0.7, 0.8, 0.9, 1\} \times \{0.3, 0.4, 0.5\} \times \{0.999, 0.9999, 0.99999\}$. In all cases, the Q-table converges to good values and the agent succeeds to find the shortest path, which means Q-Learning is not sensitive to these hyper-parameters. The difference among these Q-tables is about the minor problem discussed in the end of section 1.1.1: the Q-value far away from the cliff has not converged enough. When the decay rate of ϵ is 0.999, the problem becomes more obvious (shown in Figure 8a), and when decay rate is 0.9999, the problem disappears with all values having converge to exactly the correct value (shown in Figure 8b). This is easy to understand: this problem is due to lack of exploration, and the higher value of decay rate encourages the agent to explore more.

1.3.2 SARSA

When it comes to SARSA, things become more complicated. Let $(\gamma, \alpha, decay\_rate)$ takes the value of each one of $\{0.7, 0.8, 0.9, 1\} \times \{0.1, 0.2, 0.3\} \times \{0.999, 0.9999, 0.99999\}$, and the rewards are shown in table 1.

Table 1: rewards with different hyper-parameters

(a) decay rate = 0.999

		γ			
		0.7	0.8	0.9	1
α	0.1	-15	-15	-13	-13
	0.2	-15	-17	-15	-15
	0.3	-15	-17	-17	-17

(b) decay rate = 0.9999

		γ			
		0.7	0.8	0.9	1
α	0.1	$-\infty$	-17	-17	-17
	0.2	$-\infty$	$-\infty$	-17	-17
	0.3	$-\infty$	$-\infty$	$-\infty$	-17

(c) decay rate = 0.99999

		γ			
		0.7	0.8	0.9	1
α	0.1	$-\infty$	$-\infty$	$-\infty$	-17
	0.2	$-\infty$	$-\infty$	$-\infty$	-17
	0.3	$-\infty$	$-\infty$	$-\infty$	-17

Here the reward begin $-\infty$ means that the agent is trapped in a place and will never reach the destination. Now let's see what happened. Take the example of the case where $\gamma = 0.9, \alpha = 0.1$ and the decay rate is 0.99999. Figure 9 shows the Q-table.

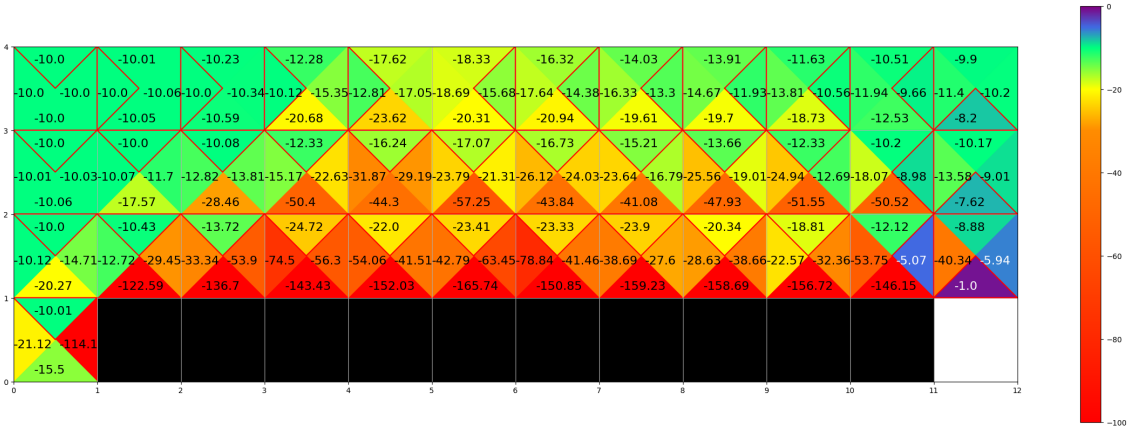


Figure 9: the Q-table with decay rate = 0.99999, $\gamma = 0.9, \alpha = 0.1$

In this case, $Q(0,0)$ (in the grid in the left-top corner and move upward) converges to -10.0. So when the agent take action 0, it stays in place and update $Q(0,0)$ as $-10.0 + \alpha \times (-1 + \gamma \times -10.0 - (-10.0)) = -10.0$, that is, no change. Thus the agent repeat this action forever. In fact, -10 is indeed the reward of an infinite path because

$$\sum_{i=0}^{+\infty} \gamma^i (-1) \tag{2}$$

equals -10 when $\gamma = 0.9$. The strange thing is how the other actions in state 0 are considered worse than staying in place to take this infinite path. Specifically, why moving rightward is considered worse? The only answer is that $Q(0,1)$ has incorporated the risk propagated from the cliff, as is explained in the previous section. Here emerges the fallacy of SARSA: it gives too much weight to risks, and γ makes taking a infinite path acceptable, so the agent would take the totally wrong action. Only if $\gamma = 1$ can this fallacy be avoided, as this makes the formula (2) diverges so walking infinitely is worse than all risks.

Consider what is the relationship between the hyper-parameters and the probability of the agent encountering this fallacy. From Table 1b, we can see the higher α is or the lower γ is, the more likely the agent is trapped. Comparing Table 1b with 1c, we can see the the lower ϵ decays, the more likely the agent is trapped. My explanation is as follows.

- We already know that "abnormal samples" (with low values due to the risk of falling into cliff) have more influence on Q-values than "normal samples". Thus the increase of the weight of samples α enhances such inequality between "abnormal samples" and "normal samples". It follows that the agent becomes more reluctant to take risks and willing to cowardly stay in the corner.

- The rewards of the infinite path and the shortest path from state 0 are

$$\sum_{i=0}^{+\infty} \gamma^i (-1), \quad \sum_{i=0}^{13} \gamma^i (-1)$$

respectively. So the gap between them is

$$\sum_{i=14}^{+\infty} \gamma^i(-1)$$

The fallacy arises because "abnormal samples" fills this gap. The lower γ is, the smaller this gap is, and the easier for those "abnormal samples" to fill this gap, and finally, the easier for the agent to be trapped.

- As is explained in the previous section, an important reason for the arising of "abnormal samples" is exploration: the agent explores the map but falls into the cliff or get closer to the cliff. The low rate of the decay of ϵ encourages the agent to explore more, so "abnormal samples" appears more frequently.

Now look at Table 1a to consider the effects of ϵ decreasing more rapidly. We can see the agent becomes not so conservative and willing to take shorter and more dangerous paths, like Q-Learning. This is because if ϵ decays rapidly, then exploration is reduced and thus "abnormal samples" appears less. Moreover, generally the decrease of γ and the increase of α play the same role in keeping the agent far away from the cliff. The reasons for these phenomena is the same as what is explained just now.

2 Deep Reinforcement Learning

2.1 Principle

In the cliff-walking environment, the state space and action space is small, so it is easy to maintain Q-values for all state-actions pairs in a look-up table. However, when the spaces become larger, the "curse of dimensionality" emerges. Since it is no longer practical to maintain all Q-values, we need to estimate them with function approximation, and here comes deep neural network. We build a network so that when this network is fed with a state, it should return the estimate of the Q-values of actions in this state.

There are two major problems when training the network, calling for additional tricks.

- The first problem is about the data. We use mini-batch gradient descent to train, but successive samples are sequential and thus correlated. Also, one sample is used only once so collecting data is too time-consuming. The solution is to use a replay buffer to store experiences. When training, samples are randomly picked from it so that they can be seen as i.i.d. and samples are utilized more efficiently.

- The second problem is about the robustness of the network. It turns out that policy changes rapidly with slight changes to Q-values. To enhance robustness, we build two networks. One (network1) is for obtaining new experiences, and the other serves as the target when updating parameters of network1. This solves the problem of oscillation of the policy because the target is fixed in a period.

2.2 Train and Tune the Agent

2.2.1 Tuning Hyper-Parameters

The given default agent fails to learn to land, for which hyper-parameters may be responsible.

- The discounting factor `gamma` is set as 0.6, which I think is too small. Recall Equation (1), `gamma` measures the weight of the long-term rewards with respect to the immediate reward. In this lunar-landing environment, the ultimate and unique goal is landing correctly, so we should pay much attention to long-term rewards. Hence I set `gamma` as 0.99.

- The exploration schema is not reasonable enough. In the beginning, as the agent knows nothing about the environment, ϵ should be high. When enough experiences are accumulated, exploitation should be encouraged for convergence. Therefore, I increase `start_e` from 0.3 to 0.8 and decrease `end_e` from 0.05 to 0.01. As the range of change of ϵ becomes wider, I also extend `exploration_fraction` from 0.1 to 0.3 in order to slow down the decay.

- Clearly, to solve the problem of correlated data, replay buffer should be large enough. So I enlarge `buffer_size` from 10000 to 30000.

2.2.2 Training Process and Result

Having tuned these hyper-parameters, the training process and result is as below. Figure 10 shows the episodic return and Figure 11 shows the loss.

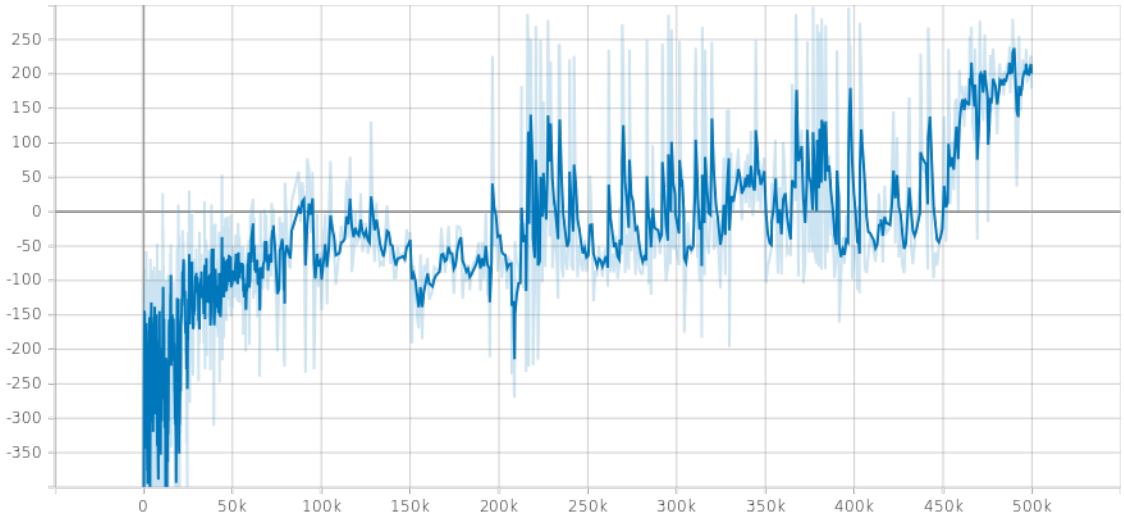


Figure 10: the episodic return during the training process

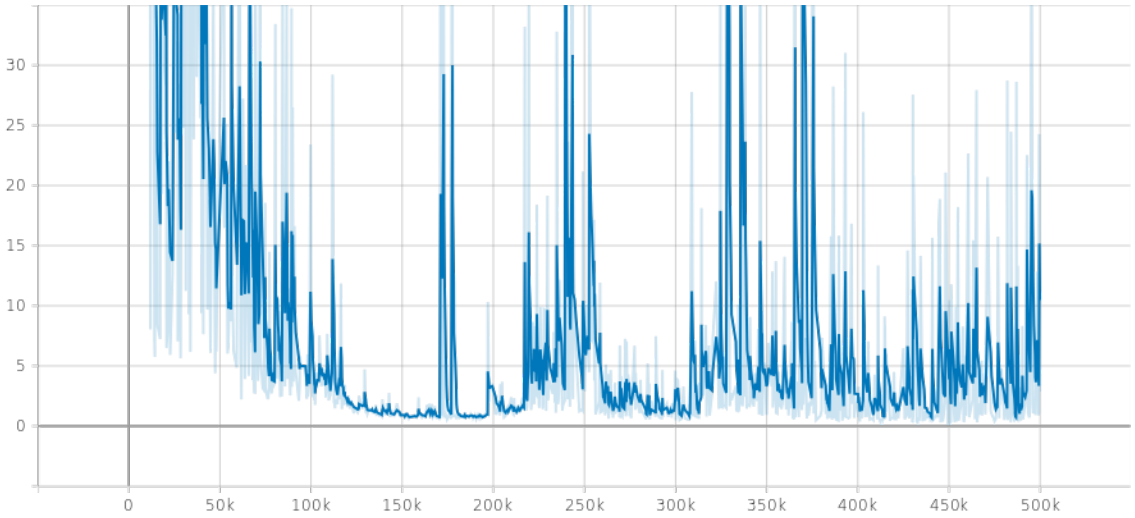


Figure 11: the loss during the training process

From these two figures we can see the loss decreases generally and the episodic rewards converge to around 200, which is the reward for solving the problem according to the official document[2]. In the beginning 220k steps, the agent crasheds nearly each episode. By trial and error, it gradually learns correct actions, and thus seldom crashes in the latter stage of learning. In fact, the last 10 episodes has an average reward 205, with the highest one being 236. The video of the landing process after training is provided, which demonstrates the agent has really learned to land successfully.

2.3 Improve DQN Agent

2.3.1 Drawback and Analysis

One remarkable drawback of DQN is that it converges too slowly, as is clearly exposed in Figure 10 and Figure 11. This aries from that learning is not efficient enough. In the learning, we minimize the distance from the current Q-network to the target Q-network in order to update parameters, and the distance is measured by sampling from the replay buffer randomly. Sampling randomly means we assume all experiences are equally significant for the agent to learn, which is obviously unreasonable. Motivating by this problem, an improvement of DQN called "Prioritized Experience Replay" is proposed, greatly accelerating learning and thus convergence[3].

2.3.2 Solution

From the analysis above, we know prioritising experiences would make the agent learn more efficiently, so the problem is how to measure the significance of one experience for the agent

[2] https://www.gymlibrary.dev/environments/box2d/lunar_lander/#rewards
[3] https://arxiv.org/pdf/1511.05952.pdf

to learn. That an experience is significant means the agent can learn a lot from it, and learning a lot means the current Q-value is far from the target. Therefore, the priority of an experience should be positively correlated to its temporal-difference error.

Under the guidance of such thought, the priority p_t of a transition $(S_{t-1}, A_{t-1}, R_t, \gamma, S_t)$ (the agent transitions from state S_{t-1} to state S_t after taking action A_t, getting a reward of R_t) can be set as the absolute value of its TD-error, i.e.,

$$p_t = \left| R_t + \gamma Q_{\text{target}}(S_t, \arg\max_a Q(S_t, a)) - Q(S_{t-1}, A_{t-1}) \right|$$

where Q_{target} is the target Q-network and Q is the current Q-network.

After introducing priorities, the simplest way to replay experiences is greedily sampling a mini-batch with highest priorities. But this results in that only a small subset of experiences are considered and those with low priority would be never replayed. To address this, stochastic sampling method is applied which interpolates between pure greedy prioritization and uniform random sampling. Specifically, we still sample experiences stochastically but ensure that the probability of one experience $(S_{t-1}, A_{t-1}, R_t, \gamma, S_t)$ being sampled is positively correlated to its priority p_t. We can set this probability as

$$P(t) = \frac{p_t^\alpha}{\sum_k p_k^\alpha}.$$

Here, the exponent $\alpha \geq 0$ represents how much we value the priorities, with $\alpha = 0$ corresponding to uniform random sampling and $\alpha \to +\infty$ corresponding to pure greedy prioritization. A data structure called "sum-tree" can efficiently support such sampling.

All these work enable the agent to make the most effective use of the experiences and thereby learns with high expected progress. It turns out that the resulting algorithm can achieve faster learning and "state-of-the-art" performance.