Chapter 5

Monte Carlo Learning

This chapter introduces the reinforcement learning (RL) algorithms based on Monte Carlo (MC) estimation. They are the first class of model-free RL algorithms ever introduced in this book. When system models are not available, MC methods learn optimal policies from the interaction experience between the agent and the environment.

The basic idea of MC learning is very simple. In particular, the simplest MC learning algorithm can be easily obtained by replacing the model-based policy evaluation step in the policy iteration algorithm introduced in the last chapter with a model-free one. This simplest algorithm can be further extended in various ways to obtain more complex MC learning algorithms that can use samples more efficiently.

5.1 Motivating example: Monte Carlo estimation

Monte Carlo estimation refers to a broad class of techniques that use stochastic samples to solve approximation problems. The basic idea of MC estimation is demonstrated by the following *mean estimation* problem.

Consider a random variable X that can take values in a finite set of real numbers denoted as $\mathcal{X}$. Suppose our task is to calculate the mean or expectation of X: $\mathbb{E}[X]$. There are two approaches to calculating $\mathbb{E}[X]$.

– The first approach is *model-based*. In particular, the model refers to the probability $p(X = x)$ for X taking any $x \in \mathcal{X}$. If the model is known, then the mean can be calculated based on the definition of expectation:

$$\mathbb{E}[X] = \sum_{x \in \mathcal{X}} p(x)x.$$

For example, suppose we flip a coin repeatedly. The random variable X has two possible values: $X = 1$ (head of the coin) and $X = -1$ (tail of the coin). If we know the probability distribution of X exactly, for example, $p(X = 1) = 0.5$ and

$p(X = -1) = 0.5$, then the mean can be calculated directly as

$$\mathbb{E}[X] = 0.5 \cdot 1 + 0.5 \cdot (-1) = 0.$$

– The second approach is *model-free*. In practice, the probability distribution of X may not be known precisely. In this case, how to estimate the mean? The basic idea is to sample X many times and get a sampling sequence $\{x_1, x_2, \ldots, x_n\}$. Then, the mean can be approximated as

$$\mathbb{E}[X] \approx \bar{x} = \frac{1}{n} \sum_{j=1}^{n} x_j.$$

When n is small, the approximation may not be accurate. However, as n increases, the approximation becomes more and more accurate. When $n \to \infty$, we have $\bar{x} \to \mathbb{E}[X]$ (Figure 5.1). This is guaranteed by the *Law of Large Numbers*: The average of a large number of samples would be close to the expected value and will be closer to the expected value when more samples are used (the proof is given in the following shaded box).

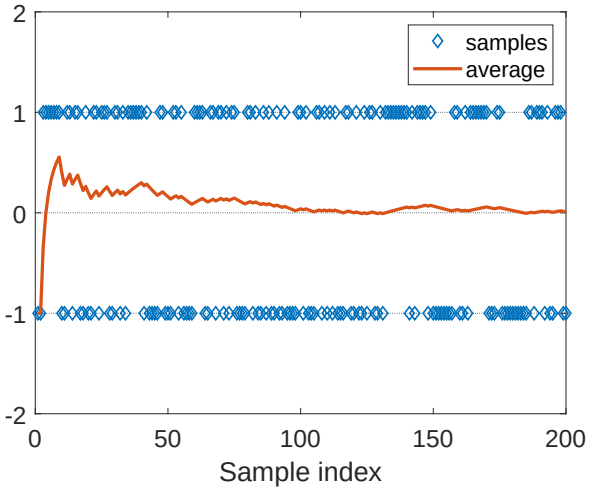


Figure 5.1: An example to demonstrate the Law of Large Numbers. Here, the samples are drawn from $\{+1, -1\}$ following a uniform distribution. The average gradually converges to zero, the true expectation value, as the sample number increases.

The above two approaches clearly demonstrate the fundamental philosophy of model-based and model-free RL: When the system model is available, the expectation can be calculated based on the model. When the model is unavailable, the expectation can be estimated approximately using stochastic samples. Readers may wonder why we suddenly start talking about the mean estimation problem in this chapter. That is simply because state value and action value are both defined as expectations (or means).

It is worth mentioning that the samples used for mean estimation must be *independent and identically distributed* (iid). The reason is obvious. If the sampling values are correlated to each other, then it is impossible to approximate the expected value. An

extreme case is that all the sampling values are the same as the first one. In this case, the average of the samples is always equal to the first sample no matter how many samples we use.

Law of Large Numbers

For a random variable X. Suppose $\{x_i\}_{i=1}^n$ are some iid samples. Let $\bar{x} = \frac{1}{n}\sum_{i=1}^n x_i$ be the average of the samples. Then,

$$\mathbb{E}[\bar{x}] = \mathbb{E}[X],$$
$$\text{var}[\bar{x}] = \frac{1}{n}\text{var}[X].$$

The above two equations indicate that $\bar{x}$ is an unbiased estimate of $\mathbb{E}[X]$ and its variance decreases to zero as n increases to infinity.

The proof is given below.

First, $\mathbb{E}[\bar{x}] = \mathbb{E}\big[\sum_{i=1}^n x_i/n\big] = \sum_{i=1}^n \mathbb{E}[x_i]/n = \mathbb{E}[X]$, where the last equability is because the samples are *identically distributed* (that is, $\mathbb{E}[x_i] = \mathbb{E}[X]$).

Second, $\text{var}(\bar{x}) = \text{var}\big[\sum_{i=1}^n x_i/n\big] = \sum_{i=1}^n \text{var}[x_i]/n^2 = (n \cdot \text{var}[X])/n^2 = \text{var}[X]/n$, where the second equality is because the samples are *independent* and the third equability is because the samples are *identically distributed* (that is, $\text{var}[x_i] = \text{var}[X]$).

5.2 The simplest MC learning algorithm

We now introduce the simplest MC learning algorithm. This algorithm is obtained by replacing the *model-based policy evaluation step* in the policy iteration algorithm as introduced in the last chapter with a *model-free MC estimation step*. This algorithm is too simple to use in practice. However, it is important for us to understand the core idea of model-free RL. This algorithm will be extended to obtain more complex and practical MC learning algorithms later in this chapter.

5.2.1 Converting policy iteration to be model-free

Policy iteration is a model-based algorithm introduced in the last chapter. We next show that the model-based part of this algorithm can be replaced by MC estimation so that the algorithm can become model-free.

The policy iteration algorithm contains two steps in each iteration. The first is *policy evaluation*, which is to compute v_{π_k} by solving $v_{\pi_k} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}$. The second step is *policy improvement*, which is to compute the greedy policy $\pi_{k+1} = \arg\max_\pi (r_\pi + \gamma P_\pi v_{\pi_k})$.

The elementwise form of the policy improvement step is

$$\begin{aligned}\pi_{k+1}(s) &= \arg\max_{\pi} \sum_a \pi(a|s) \left[\sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a) v_{\pi_k}(s') \right] \\ &= \arg\max_{\pi} \sum_a \pi(a|s) q_{\pi_k}(s,a), \quad s \in \mathcal{S}.\end{aligned}$$

It can be seen from the above equation that $q_{\pi_k}(s,a)$ is the core. Once $q_{\pi_k}(s,a)$ is obtained, $\pi_{k+1}(s)$ can be easily obtained.

How to obtain action values? There are two approaches as demonstrated in the coin-flipping example in the last section.

– The first approach is *model-based*. This is the approach adopted by the policy iteration algorithm. In particular, we can first calculate the state value v_{π_k} by solving the Bellman equation. Then, we can calculate the action values by using

$$q_{\pi_k}(s,a) = \sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a) v_{\pi_k}(s'). \tag{5.1}$$

This approach require to know the system model $\{p(r|s,a), p(s'|s,a)\}$.

– The second approach is *model-free*. Recall that the definition of action value is

$$\begin{aligned}q_{\pi_k}(s,a) &= \mathbb{E}[G_t|S_t = s, A_t = a] \\ &= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | S_t = s, A_t = a],\end{aligned}$$

which is the expectation of the return obtained starting from (s,a). Since $q_{\pi_k}(s,a)$ is an expectation, it can be estimated by MC estimation as demonstrated in the last section. Specifically, starting from (s,a), the agent can interact with the environment following policy π_k and then obtain a number of episodes. The return of each episode is a random sample of $R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots$. Suppose there are n episodes and denote the return of the ith episode as $g^{(i)}(s,a)$. Then, $q_{\pi_k}(s,a)$ can be approximated as

$$q_{\pi_k}(s,a) = \mathbb{E}[G_t|S_t = s, A_t = a] \approx \frac{1}{n} \sum_{i=1}^{n} g^{(i)}(s,a).$$

This approximation is the MC estimation. We know that, if the number of episodes n is sufficiently large, the approximation will be sufficiently accurate.

5.2.2 MC Basic algorithm

We are now ready to present the first MC learning algorithm.

In particular, given an initial policy π_0, there are two steps in each iteration. Suppose the current iteration is the kth.

Pseudocode: MC Basic algorithm (a model-free variant of policy iteration)

Initialization: Initial guess π_0.
Aim: Search for an optimal policy.

While the value estimate has not converged, for the kth iteration, do
  For every state $s \in \mathcal{S}$, do
    For every action $a \in \mathcal{A}(s)$, do
      Collect sufficiently many episodes starting from (s, a) following π_k
      MC-based policy evaluation step:
      $q_{\pi_k}(s, a) =$ average return of all the episodes starting from (s, a)
    Policy improvement step:
    $a_k^*(s) = \arg\max_a q_{\pi_k}(s, a)$
    $\pi_{k+1}(a|s) = 1$ if $a = a_k^*$, and $\pi_{k+1}(a|s) = 0$ otherwise

– *Step 1: policy evaluation.* This step is to obtain $q_{\pi_k}(s, a)$ for all (s, a). Specifically, for each action-state pair (s, a), collect sufficiently many episodes. The average of their returns is used to approximate $q_{\pi_k}(s, a)$.

– *Step 2: policy improvement.* This step is to solve $\pi_{k+1}(s) = \arg\max_\pi \sum_a \pi(a|s) q_{\pi_k}(s, a)$ for all $s \in \mathcal{S}$. The greedy optimal policy is $\pi_{k+1}(a_k^*|s) = 1$ where $a_k^* = \arg\max_a q_{\pi_k}(s, a)$.

This algorithm is the simplest MC learning algorithm. Though simple, it is important for understanding more complicated algorithms. In this book, this algorithm is called *MC Basic*. The pseudocode is given in the following box.

MC Basic is a variant of the policy iteration algorithm. The difference is that MC Basic calculates action values directly from experience samples, whereas policy iteration calculates state values first and then action values based on the system model.

Why does MC Basic estimate action values instead of state values? That is because state values cannot be used to improve policies directly. Even if we are given state values, we still need to calculate action values from these state values using (5.1). However, such a calculation requires the system model. Therefore, when models are not available, we should directly estimate action values.

Since policy iteration is convergent, MC Basic is also convergent given as long as the experience samples are sufficient. Finally, though simple, MC Basic is not practical in practice due to its low sample efficiency. It is, however, important for us the grasp the core idea of model-free RL. Later in this chapter, we will extend MC Basic to more complex and practical MC learning algorithms.

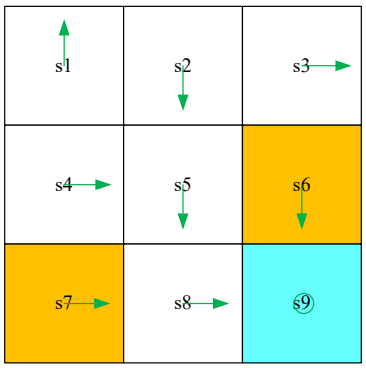


Figure 5.2: An example to illustrate the MC Basic algorithm.

5.2.3 Illustrative examples

A simple example: A step-by-step implementation

We next use an example to illustrate the implementation details of the MC Basic algorithm. The reward setting is $r_{\text{boundary}} = r_{\text{forbidden}} = -1$, and $r_{\text{target}} = 1$. The discount rate is $\gamma = 0.9$. The initial policy π_0 as shown in Figure 5.2 is not optimal. Specifically, the policy for s_1 or s_3 is not optimal.

Although we need to calculate the action values of all states, we merely demonstrate those of s_1 due to space limitation. Starting from s_1, there are five possible actions. For each action, we need to run sufficiently many episodes to well approximate the action value. However, since this example is deterministic in terms of both policy and model, the estimation of each action value merely requires a single episode, because running multiple times would generate exactly the same trajectory.

For iteration $k = 0$, we can obtain the following five episodes starting from s_1:

– Starting from (s_1, a_1), the episode is $s_1 \xrightarrow{a_1} s_1 \xrightarrow{a_1} s_1 \xrightarrow{a_1} \dots$. Hence, the action value is

$$q_{\pi_0}(s_1, a_1) = -1 + \gamma(-1) + \gamma^2(-1) + \cdots = \frac{-1}{1-\gamma}.$$

– Starting from (s_1, a_2), the episode is $s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_3} \dots$. Hence, the action value is

$$q_{\pi_0}(s_1, a_2) = 0 + \gamma 0 + \gamma^2 0 + \gamma^3(1) + \gamma^4(1) + \cdots = \frac{\gamma^3}{1-\gamma}.$$

– Starting from (s_1, a_3), the episode is $s_1 \xrightarrow{a_3} s_4 \xrightarrow{a_2} s_5 \xrightarrow{a_3} \dots$. Hence, the action value is

$$q_{\pi_0}(s_1, a_3) = 0 + \gamma 0 + \gamma^2 0 + \gamma^3(1) + \gamma^4(1) + \cdots = \frac{\gamma^3}{1-\gamma}.$$

– Starting from (s_1, a_4), the episode is $s_1 \xrightarrow{a_4} s_1 \xrightarrow{a_1} s_1 \xrightarrow{a_1} \ldots$. Hence, the action value is

$$q_{\pi_0}(s_1, a_4) = -1 + \gamma(-1) + \gamma^2(-1) + \cdots = \frac{-1}{1-\gamma}.$$

– Starting from (s_1, a_5), the episode is $s_1 \xrightarrow{a_5} s_1 \xrightarrow{a_1} s_1 \xrightarrow{a_1} \ldots$. Hence, the action value is

$$q_{\pi_0}(s_1, a_5) = 0 + \gamma(-1) + \gamma^2(-1) + \cdots = \frac{-\gamma}{1-\gamma}.$$

By observing the action values, we see that

$$q_{\pi_0}(s_1, a_2) = q_{\pi_0}(s_1, a_3) = \frac{\gamma^3}{1-\gamma} > 0$$

are the maximum. As a result, the policy can be improved as

$$\pi_1(a_2|s_1) = 1 \quad \text{or} \quad \pi_1(a_3|s_1) = 1.$$

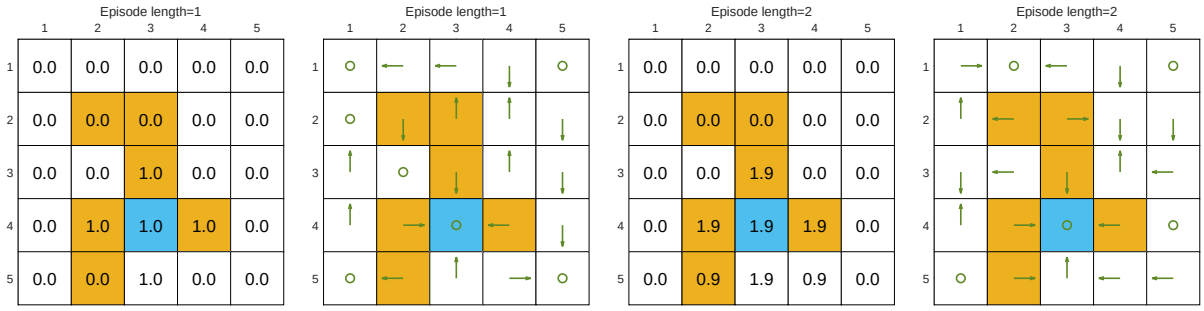
It can be seen that the improved policy, which takes either a_2 or a_3 at s_1, is optimal. Therefore, we can successfully obtain the optimal policy by using merely one iteration for this simple example. In this simple example, the initial policy for all the states except s_1 and s_3 is already optimal. Therefore, the policy can become optimal after merely a single iteration. When the policy is non-optimal for other states, more iterations are required.

A comprehensive example: Episode length and sparse reward

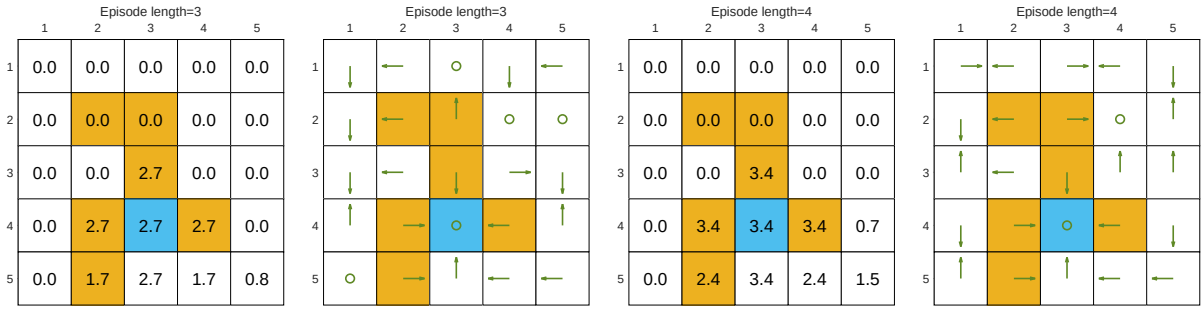
We next present a more comprehensive example to discuss some interesting properties of MC learning. The example is a 5-by-5 grid world. The reward setting is $r_{\text{boundary}} = -1$, $r_{\text{forbidden}} = -10$, and $r_{\text{target}} = 1$. The discount rate is $\gamma = 0.9$.

First of all, we show by example that the length of each episode plays an important role. All the plots in Figure 5.3 show the final results given by the MC Basic algorithm. The difference is that different results are based on different episode lengths. It is notable that the episode length has a great impact on the final policies. When the length of each episode is too short, neither the policy nor the value estimate is optimal. See, for example, Figure 5.3(a)-(d). When the episode length increases, the policy and value estimate gradually approach the optimal ones. See, for example, Figure 5.3(h). In the extreme case where the episode length is 1, only the states that are adjacent to the target have nonzero values. All the others have zero values because each episode is too short to reach the target or get positive rewards.

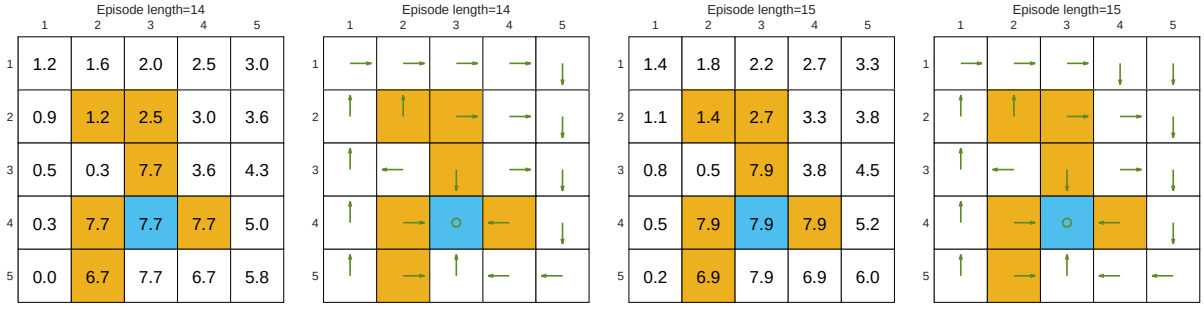
As the episode length increases, an interesting spatial pattern emerges. That is, the states that are closer to the target possess nonzero earlier than those farther away. The reason is as follows. Starting from a state, the agent has to travel at least a certain number of steps to reach the target state and then receive positive rewards. If the length



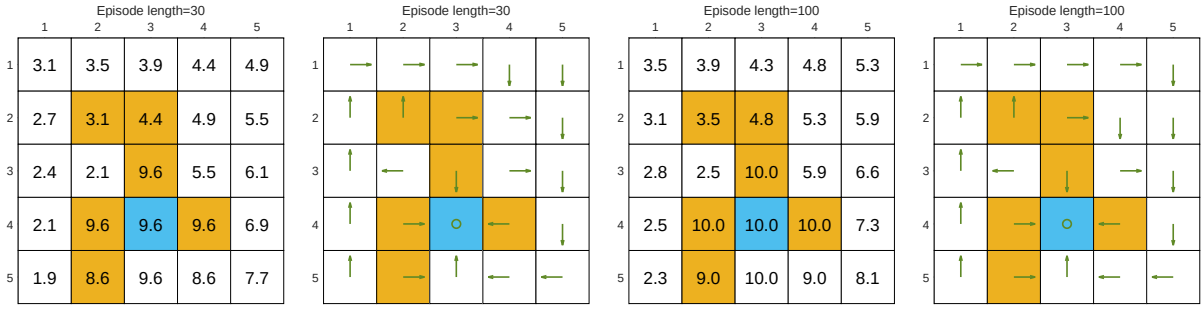
(a) Final value and policy with episode length=1 (b) Final value and policy with episode length=2



(c) Final value and policy with episode length=3 (d) Final value and policy with episode length=4



(e) Final value and policy with episode length=14 (f) Final value and policy with episode length=15



(g) Final value and policy with episode length=30 (h) Final value and policy with episode length=100

Figure 5.3: The policies and state values obtained by the MC Basic algorithm given different episode lengths. As can be seen, only if the length of each episode is sufficiently long, the state values can be accurately estimated.

of an episode is less than this desired number of steps, it is certain that the agent cannot reach the target state. Hence, the return would be zero and so is the estimated state value. In this example, the episode length must be no less than 15, which is the minimum number of steps required to reach the target starting from the bottom-left state.

While the above analysis suggests that each episode must be sufficiently long, the episodes are not necessarily infinitely long. As shown in this example, when the length is 30, the algorithm can find an optimal policy although the value estimate is not optimal yet.

The above analysis is related to an important reward design problem: *sparse reward*, which refers to the scenario that, only if the agent reaches the target state, can a positive reward be obtained. In other words, no positive rewards can be obtained unless the target is reached. Sparse reward requires long episodes that must reach the target. This requirement is challenging to meet when the state space is large. As a result, sparse reward downgrades learning efficiency. One simple technique to solve this problem is to design *non-sparse rewards*. For instance, in this grid world example, we can redesign the reward setting so that the agent can obtain a small amount of positive rewards when reaching those states located near the target. In this way, an "attractive field" can be formed around the target so that the agent can find the target easier.

5.3 MC Exploring Starts

We next extend the simple MC Basic algorithm to obtain another MC learning algorithm that is slightly more complicated but more sample-efficient.

5.3.1 Using samples more efficiently

Following a policy π, suppose we have an episode of samples as follows:

$$s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_4} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \dots$$

where the subscripts refer to the state or action indexes rather than time steps. Every time a state-action pair appears in the episode, it is called a *visit* of that state-action pair. There are different strategies to utilize the visits.

The first strategy is to use the *initial visit*. That is the episode is only used to estimate the action value of the starting state-action pair $q_\pi(s_1, a_2)$. The initial-visit strategy is adopted by the MC Basic algorithm but *not sample-efficient*. That is because the episode also visits many other state-action pairs such as (s_2, a_4), (s_2, a_3), and (s_5, a_1). These visits can also be used to estimate the corresponding action values. In particular,

we can decompose this episode into multiple sub-episodes:

$$
\begin{aligned}
s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_4} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \dots \quad & \text{[original episode]} \\
s_2 \xrightarrow{a_4} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \dots \quad & \text{[episode starting from } (s_2, a_4)\text{]} \\
s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \dots \quad & \text{[episode starting from } (s_1, a_2)\text{]} \\
s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \dots \quad & \text{[episode starting from } (s_2, a_3)\text{]} \\
s_5 \xrightarrow{a_1} \dots \quad & \text{[episode starting from } (s_5, a_1)\text{]}
\end{aligned}
$$

That is, the trajectory after the visit of a state-action pair can be viewed as a new episode. The new episodes can be used to estimate more action values such as $q_\pi(s_2, a_4)$, $q_\pi(s_2, a_3)$, and $q_\pi(s_5, a_1)$. In this way, the samples in the episode are utilized more efficiently.

A state-action pair may be visited multiple times in an episode. For example, (s_1, a_2) is visited twice in the above episode. If we only count the first-time visit, such kind of strategy is called *first-visit*. If every time a state-action pair is visited and the rest of the episode is used to estimate its action value, such a strategy is called *every-visit*.

In terms of the efficiency of using samples, the every-visit strategy is the best. If an episode is sufficiently long so that it can visit all the state-action pairs many times, then this single episode is sufficient to estimate all the action values by using the every-visit strategy. However, the samples obtained by the every-visit strategy are relevant, because the trajectory starting from the second visit is merely a subset of the trajectory starting from the first. Nevertheless, if the two visits are far away from each other, which means there is a significant non-overlap portion, the relevance would not be strong. Moreover, the relevance can be further suppressed due to the discount rate. Therefore, when there are few episodes and each episode is very long, the every-visit strategy is a good option.

5.3.2 Updating estimate more efficiently

Another aspect of MC learning is when to update the policy. There are two strategies.

The first strategy is, in the policy evaluation step, to collect all the episodes starting from the same state-action pair and then approximate the action value using the average return of these episodes. This strategy is adopted in the MC Basic algorithm. The drawback of this strategy is that the agent has not to wait until all episodes have been collected. The second strategy, which can overcome this drawback, is to use the return of a single episode to approximate the corresponding action value. In this way, we can improve the policy in an episode-by-episode fashion.

While the return of a single episode cannot accurately approximate the corresponding action value, one may wonder whether the second strategy is good or not. In fact, such kind of *inaccurate approximation* falls into the idea of generalized policy iteration introduced in the last chapter.

Pseudocode: MC Exploring Starts (a sample-efficient variant of MC Basic)

Initialization: Initial guess π_0.
Aim: Search for an optimal policy.

For each episode, do
Episode generation: Randomly select a starting state-action pair (s_0, a_0) and ensure that all pairs can be possibly selected. Following the current policy, generate an episode of length T: $s_0, a_0, r_1, \ldots, s_{T-1}, a_{T-1}, r_T$.
Policy evaluation and policy improvement:
Initialization: $g \leftarrow 0$
For each step of the episode, $t = T-1, T-2, \ldots, 0$, do
$g \leftarrow \gamma g + r_{t+1}$
Use the first-visit strategy:
If (s_t, a_t) does not appear in $(s_0, a_0, s_1, a_1, \ldots, s_{t-1}, a_{t-1})$, then
$Returns(s_t, a_t) \leftarrow Returns(s_t, a_t) + g$
$q(s_t, a_t) = \text{average}(Returns(s_t, a_t))$
$\pi(a|s_t) = 1$ if $a = \arg\max_a q(s_t, a)$

5.3.3 Algorithm description

If the MC Basic algorithm is modified so that the sample usage and estimate update are more efficient, the resulting algorithm is called *MC Exploring Starts* in this book.

The pseudocode of MC Exploring Starts is given in the box below. *Exploring starts* is an important assumption in MC learning. It requires generating sufficiently many episodes starting from *every* state-action pair. Both MC Basic and MC Exploring Starts need this assumption. However, exploring starts may be difficult to achieve in practice. We will study how to remove this requirement in the next section.

In addition, when calculating the discounted return starting from each state-action pair, the procedure starts from the ending states and travels back to the starting state. In this way, the calculation is more efficient. The algorithm uses the first-visit strategy. We can also change to the every-visit strategy without examining whether the state appears for the first time.

5.4 MC learning without exploring starts

We next further extend the MC Exploring Starts algorithm introduced in the last chapter by removing the assumption of exploring starts.

Why is exploring starts important? In theory, exploring starts is necessary to find optimal policies. Only if every action of every state is well explored, can we select the optimal actions correctly. Otherwise, if an action is not explored, this action may happen to be the optimal one and hence be missed. In practice, exploring starts is

difficult to achieve. For many applications, especially those involving physical interactions with environments, it is difficult to collect episodes starting from every state-action pair. Therefore, there is a gap between theory and practice. Can we remove the requirement of exploring starts? We next show that we can do it by using soft policies.

5.4.1 Soft policies

A policy is *soft* if the policy has a positive probability to take any action at any state. With a soft policy, a single episode that is sufficiently long can visit *every* state-action pair many times. Thus, the single episode can provide sufficient samples and we do not need to have a large number of episodes starting from different state-action pairs. Then, the requirement of exploring starts can be removed.

The most common soft policy is ϵ-*greedy*. An ϵ-greedy policy is a stochastic policy that has a positive probability to take any action but a higher chance to take the *greedy action*. Here, the greedy action refers to the action with the greatest action value. In particular, suppose $\epsilon \in [0, 1]$. The ϵ-greedy policy has the form of

$$\pi(a|s) = \begin{cases} 1 - \dfrac{\epsilon}{|\mathcal{A}(s)|}(|\mathcal{A}(s)| - 1), & \text{for the greedy action,} \\ \dfrac{\epsilon}{|\mathcal{A}(s)|}, & \text{for the other } |\mathcal{A}(s)| - 1 \text{ actions.} \end{cases}$$

Here, $|\mathcal{A}(s)|$ denotes the number of actions associated with s. It is worth noting that the probability to take the greedy action is always greater than that of any other action, because

$$1 - \frac{\epsilon}{|\mathcal{A}(s)|}(|\mathcal{A}(s)| - 1) = 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(s)|} \geq \frac{\epsilon}{|\mathcal{A}(s)|}$$

for any $\epsilon \in [0, 1]$. When $\epsilon = 0$, ϵ-greedy becomes greedy.

5.4.2 Algorithm description

How to embed ϵ-greedy policies into MC learning? The answer is to change the policy improvement step from greedy to ϵ-greedy. In particular, the policy improvement step in MC Basic or MC Exploring Starts is to solve

$$\pi_{k+1}(s) = \arg\max_{\pi \in \Pi} \sum_a \pi(a|s) q_{\pi_k}(s, a). \tag{5.2}$$

where Π denotes *the set of all possible policies*. The greedy policy is

$$\pi_{k+1}(a|s) = \begin{cases} 1, & a = a_k^*, \\ 0, & a \neq a_k^*, \end{cases}$$

Pseudocode: MC ϵ-Greedy (a variant of MC Exploring Starts)

Initialization: Initial guess π_0 and the value of $\epsilon \in [0, 1]$
Aim: Search for an optimal policy.

For each episode, do

Episode generation: Randomly select a starting state-action pair (s_0, a_0). Following the current policy, generate an episode of length T: $s_0, a_0, r_1, \ldots, s_{T-1}, a_{T-1}, r_T$.
Policy evaluation and policy improvement:
Initialization: $g \leftarrow 0$
For each step of the episode, $t = T-1, T-2, \ldots, 0$, do
$g \leftarrow \gamma g + r_{t+1}$
Use the first-visit strategy:
If (s_t, a_t) does not appear in $(s_0, a_0, s_1, a_1, \ldots, s_{t-1}, a_{t-1})$, then
$Returns(s_t, a_t) \leftarrow Returns(s_t, a_t) + g$
$q(s_t, a_t) = \text{average}(Returns(s_t, a_t))$
Let $a^* = \arg\max_a q(s_t, a)$ and

$$\pi(a|s_t) = \begin{cases} 1 - \frac{|\mathcal{A}(s_t)|-1}{|\mathcal{A}(s_t)|}\epsilon, & a = a^* \\ \frac{1}{|\mathcal{A}(s_t)|}\epsilon, & a \neq a^* \end{cases}$$

where $a_k^* = \arg\max_a q_{\pi_k}(s, a)$.

Now, the policy improvement step is changed to solve

$$\pi_{k+1}(s) = \arg\max_{\pi \in \Pi_\epsilon} \sum_a \pi(a|s) q_{\pi_k}(s, a), \tag{5.3}$$

where Π_ϵ denotes *the set of all ϵ-greedy policies* with a fixed value of ϵ. The key difference of (5.3) from (5.2) is to search the policy in Π_ϵ, which is a subset of Π. Suppose $a_k^* \doteq \arg\max_a q_{\pi_k}(s, a)$. Then, the ϵ-greedy policy is updated by

$$\pi_{k+1}(a|s) = \begin{cases} 1 - \frac{|\mathcal{A}(s)|-1}{|\mathcal{A}(s)|}\epsilon, & a = a_k^*, \\ \frac{1}{|\mathcal{A}(s)|}\epsilon, & a \neq a_k^*. \end{cases}$$

The pseudocode of the algorithm is given in the following box. This algorithm is referred to as *MC ϵ-Greedy* in this book. The only difference between MC ϵ-Greedy and MC Exploring Starts is that the former uses ϵ-greedy policies and the every-visit strategy. Here, the every-visit strategy must be used. That is because, while an episode can visit every state-action pair many times, if the first visit strategy is used instead, it is a waste of samples and cannot approximate the action values accurately.

5.4.3 Algorithm convergence

If we replace greedy policies with ϵ-greedy policies in the policy improvement step, can we still guarantee to find optimal policies? The answer is both yes and no. By yes, it means that, if given sufficient samples, the algorithm can converge to an ϵ-greedy policy that is optimal among the set Π_ϵ. By no, it means that the finally obtained policy is merely optimal in Π_ϵ but not optimal among all possible policies.

The following result shows that the policy improvement step can still generate better and better ϵ-greedy policies.

Lemma 5.1 (Policy Improvement)**.** *If $\pi_k \in \Pi_\epsilon$ and $\pi_{k+1} = \arg\max_{\pi \in \Pi_\epsilon}(r_\pi + \gamma P_\pi v_{\pi_k})$, then $v_{\pi_{k+1}} \geq v_{\pi_k}$ for any k.*

Proof of Lemma 5.1

Since π_k, π_{k+1} are both in Π_ϵ and $\pi_{k+1} = \arg\max_{\pi \in \Pi_\epsilon}(r_\pi + \gamma P_\pi v_{\pi_k})$, we have

$$r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_{\pi_k} \geq r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}.$$

Since

$$\begin{aligned} v_{\pi_{k+1}} &= r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_{\pi_{k+1}}, \\ v_{\pi_k} &= r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}, \end{aligned}$$

it follows that

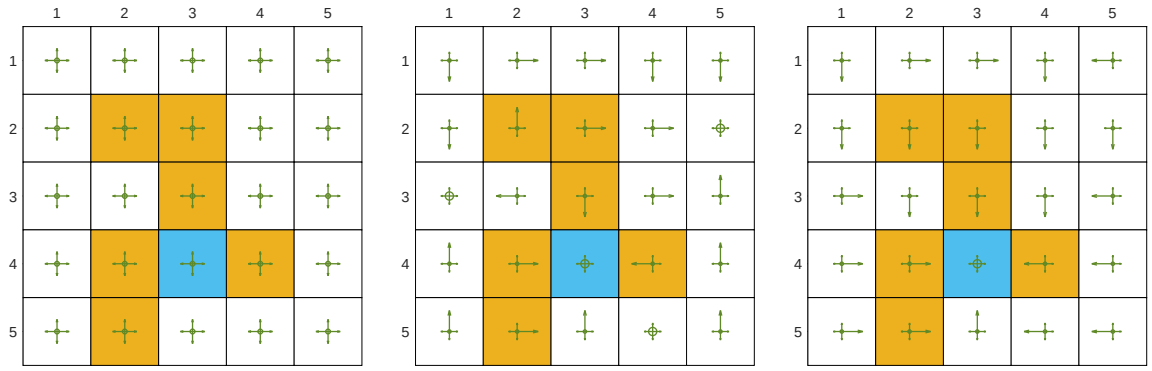
$$\begin{aligned} v_{\pi_{k+1}} - v_{\pi_k} &= (r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_{\pi_{k+1}}) - (r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}) \\ &\geq (r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_{\pi_{k+1}}) - (r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_{\pi_k}) \\ &= \gamma P_{\pi_{k+1}} (v_{\pi_{k+1}} - v_{\pi_k}). \end{aligned}$$

As a result,

$$v_{\pi_{k+1}} - v_{\pi_k} \geq \gamma^2 P_{\pi_{k+1}}^2 (v_{\pi_{k+1}} - v_{\pi_k}) \geq \cdots \geq \lim_{n \to \infty} \gamma^n P_{\pi_{k+1}}^n (v_{\pi_{k+1}} - v_{\pi_k}) = 0,$$

where the last equality is due to $\lim_{n \to \infty} \gamma^n = 0$ and $P_{\pi_{k+1}}^n$ is a stochastic matrix.

Although Lemma 5.1 shows that the ϵ-policy is continuously improved, it still requires a proof showing that it can reach an optimal ϵ-policy. We need to first show that $f(v) = \max_{\pi \in \Pi_\epsilon}(r_\pi + \gamma P_\pi v_{\pi_k})$ is a contraction mapping and then the proof is analogous to Theorem 3.2. Finally, the above analysis is merely for the replacement of greedy policies with ϵ-greedy ones. In addition to that, the algorithm of MC ϵ-Greedy also adopts the every visit strategy that makes the convergence analysis nontrivial.



(a) Initial policy (b) After the first iteration (c) After the second iteration

Figure 5.4: The evolution process of the MC ϵ-Greedy algorithm based on single episodes.

5.4.4 Illustrative example

Consider a grid-world example. The aim is to find the optimal policy for every state. In the episode generation step of the MC ϵ-greedy algorithm, a single episode of one million steps is generated. Here, $r_{\text{boundary}} = r_{\text{forbidden}} = -1$, $r_{\text{target}} = 1$, and $\gamma = 0.9$.

The initial policy is a uniform policy that has the same probability of 0.2 to take any action as shown in Figure 5.4. The optimal ϵ-policy with $\epsilon = 0.5$ can be obtained after two iterations. The convergence is fast because the episode is sufficiently long so that all the actions can be visited sufficiently times.

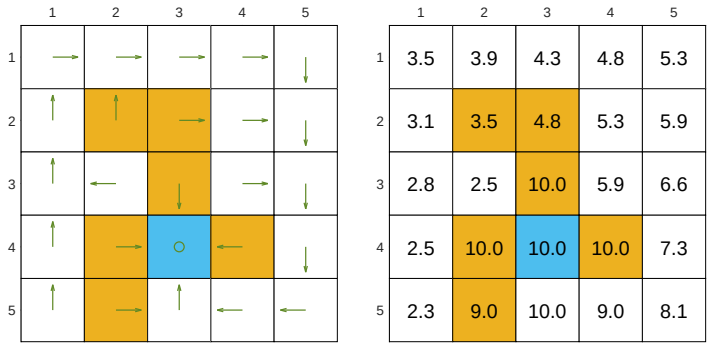
5.5 Exploitation vs exploration

An ϵ-greedy policy aims to well balance *exploitation and exploration*. On the one hand, an ϵ-greedy policy has a higher probability to take the greedy action so that it can exploit the existing value estimates. On the other hand, the ϵ-greedy policy also has a chance to take any other actions so that it can keep exploring.

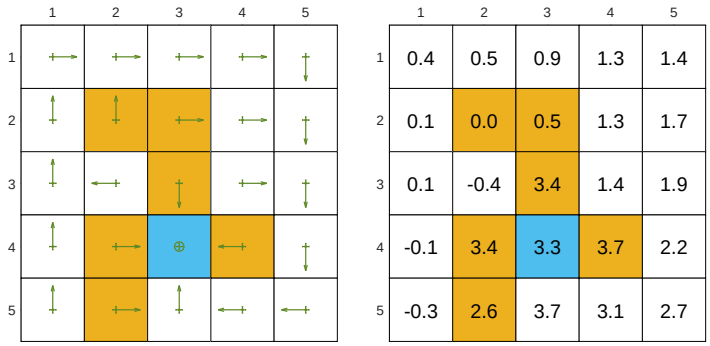
Exploitation is related to *optimality* because we know the optimal policy should be greedy. The final ϵ-greedy policy given by the MC ϵ-Greedy algorithm is *not optimal* because it is merely optimal in the set Π_ϵ. Therefore, the fundamental idea of ϵ-greedy policies is to enhance exploration by sacrificing optimality.

Exploitation and exploration form a fundamental trade-off in RL. That is, if we would like to enhance exploitation and hence optimality, we need to reduce the value of ϵ. However, if we would like to enhance exploration, we need to increase the value of ϵ.

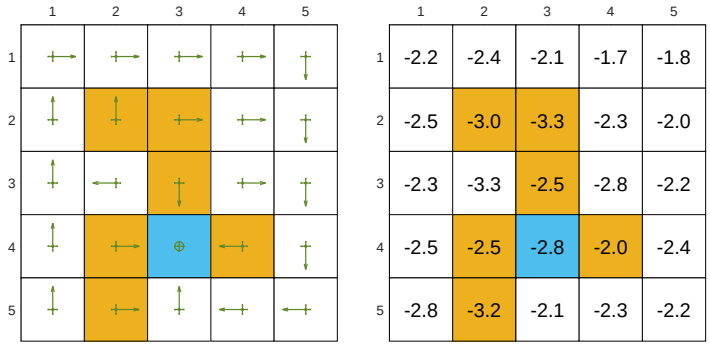
We next discuss this trade-off based on some interesting examples. The RL task here is a 5-by-5 grid world. The reward setting is $r_{\text{boundary}} = -1$, $r_{\text{forbidden}} = -10$, and $r_{\text{target}} = 1$. The discount rate is $\gamma = 0.9$.



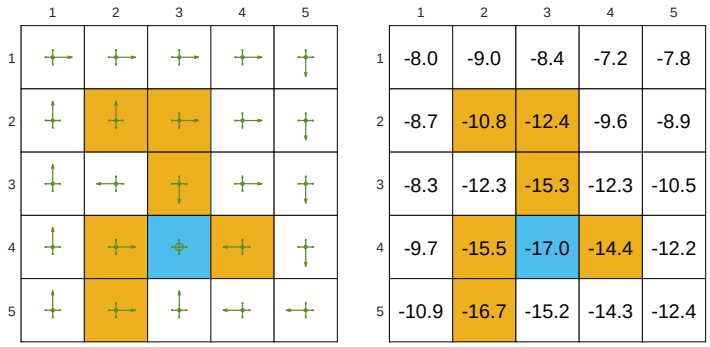
(a) Given an ϵ-greedy policy with $\epsilon = 0$, the corresponding state values.



(b) Given an ϵ-greedy policy with $\epsilon = 0.1$, the corresponding state values.

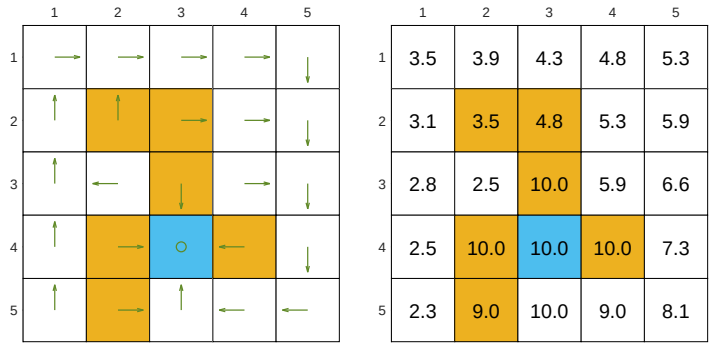


(c) Given an ϵ-greedy policy with $\epsilon = 0.2$, the corresponding state values.

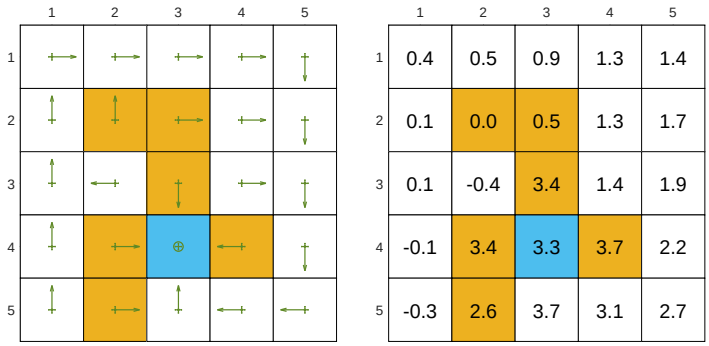


(d) Given an ϵ-greedy policy with $\epsilon = 0.5$, the corresponding state values.

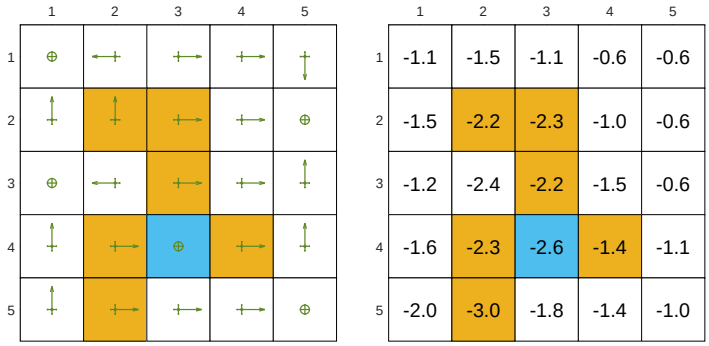
Figure 5.5: The state values of some given ϵ-greedy policies. These ϵ-greedy policies are consistent with each other in the sense that the actions with the greatest probabilities are the same. As can be seen, when the value of ϵ increases, the state values of the ϵ-greedy policies diverge from the optimal ones.



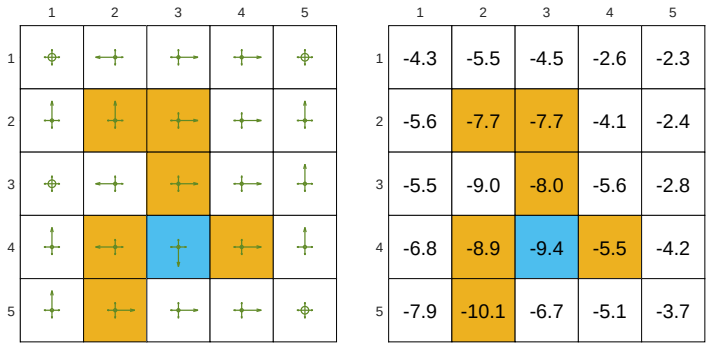
(a) The optimal ϵ-greedy policy and its state values where $\epsilon = 0$.



(b) The optimal ϵ-greedy policy and its state values where $\epsilon = 0.1$.



(c) The optimal ϵ-greedy policy and its state values where $\epsilon = 0.2$.



(d) The optimal ϵ-greedy policy and its state values where $\epsilon = 0.5$.

Figure 5.6: The optimal ϵ-greedy policies and their corresponding state values given different values of ϵ. Here, these ϵ-greedy policies are optimal among all ϵ-greedy ones (with the same value of ϵ). They are searched by the policy iteration algorithm. As can be seen, when the value of ϵ increases, the optimal ϵ-greedy policies are no longer consistent with the optimal grade one.

Optimality

We now show how the optimality of ϵ-greedy policies becomes worse when ϵ increases.

- First, a greedy optimal policy and the corresponding optimal state values are shown in Figure 5.5(a). They are obtained by the model-based value iteration algorithm.
- Second, the state values of some given ϵ-greedy policies are shown in Figure 5.5(b)-(d). These given ϵ-greedy policies are *consistent* with the greedy policy in Figure 5.5(a) but has different values of ϵ. Here, an ϵ-greedy is said to be consistent with another greedy policy if the actions with the greatest probabilities in the ϵ-greedy policy are the same as those in the greedy one.

  It is clear that, as ϵ increases, the state values of the ϵ-greedy policies decrease, indicating that the optimality of these ϵ-greedy policies becomes worse. It is notable that the value of the target state becomes the lowest instead of the highest when ϵ is large. That is simply because the target area is surrounded by some forbidden areas. When ϵ is large, the agent at the target area may enter these forbidden areas with a higher chance and get more negative rewards.
- Third, while the ϵ-greedy policies in Figure 5.5 are given, we next search optimal ϵ-greedy policies using the policy iteration algorithm with the constraint that all policies must be ϵ-greedy. Figure 5.6 shows the obtained optimal ϵ-greedy policies.

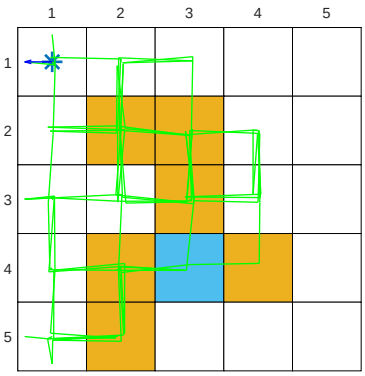
  When $\epsilon = 0$, the policy is greedy and optimal among all policies. As ϵ increases, the obtained ϵ-greedy policies are *not consistent* with the optimal greedy one anymore. Specifically, when ϵ is small such as 0.1, the resulting ϵ-greedy policy is consistent with the optimal greedy one. However, when ϵ increases to, for example, 0.2, the obtained ϵ-greedy policies are not consistent with the optimal greedy one. Therefore, if we would like to obtain ϵ-greedy policies that are consistent with the optimal greedy ones, the value of ϵ should be sufficiently small.

  Why are the ϵ-greedy policies not consistent with the optimal greedy one when ϵ is large? We can answer this question by considering the target state. In the greedy case, the optimal policy at the target state is to stay unchanged to gain positive rewards. However, when ϵ is large, the value of staying at the target state may be negative because there is a high chance to enter the forbidden areas and get negative rewards. Therefore, the optimal policy at the target state in this case is to escape instead of staying unchanged.

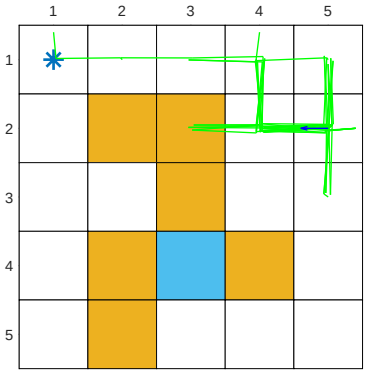
Exploration ability

We next illustrate how the exploration ability of an ϵ-greedy policy is enhanced as ϵ increases by using examples.

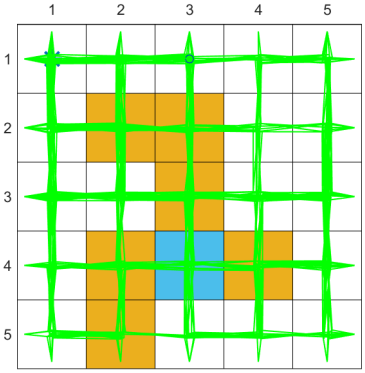
First, consider the figures in the left column of Figure 5.7. Here, $\epsilon = 1$. As a result, the ϵ-policy has the same probability of 0.2 to take any action at any state. Stating from



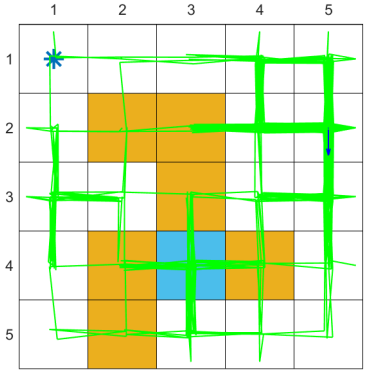
(a) $\epsilon = 1$, trajectory of 100 steps



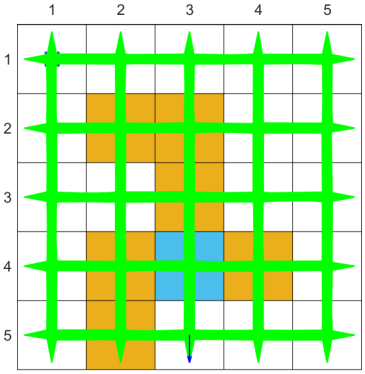
(e) $\epsilon = 0.5$, trajectory of 100 steps



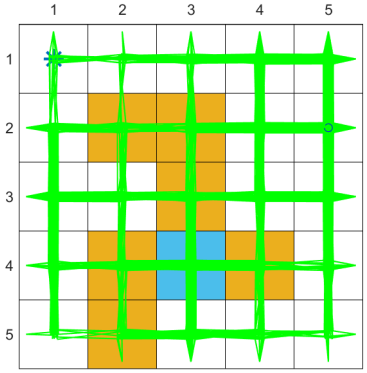
(b) $\epsilon = 1$, trajectory of 1000 steps



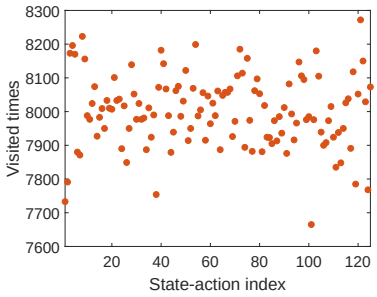
(f) $\epsilon = 0.5$, trajectory of 1000 steps



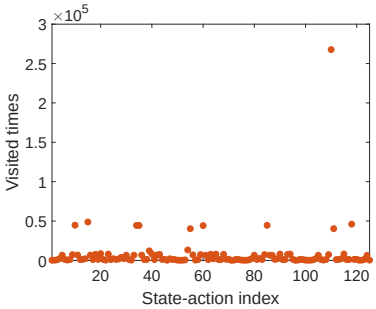
(c) $\epsilon = 1$, trajectory of 10000 steps



(g) $\epsilon = 0.5$, trajectory of 10000 steps



(d) $\epsilon = 1$, visited times of each action of 1 million steps



(h) $\epsilon = 0.5$, visited times of each action of 1 million steps

Figure 5.7: Exploration ability of ϵ-greedy policies with different values of ϵ.

the state-action pair (s_1, a_1), the trajectories generated following the policy are shown in Figure 5.7(a)-(c). As can be seen, all the state-action pairs can be visited many times when the episode is sufficiently long thanks to the strong exploration ability of the policy. Specifically, the times that the state-action pairs are visited are almost even as shown in Figure 5.7(d).

Second, consider the figures in the right column of Figure 5.7. Here, $\epsilon = 0.5$. As a result, the ϵ-greedy policy, which is plotted in Figure 5.5, has less exploration ability than the case of $\epsilon = 1$. Stating from the state-action pair (s_1, a_1), the trajectories generated following the policy are shown in Figure 5.7(e)-(g). Although every action can still be visited when the episode is sufficiently long, the distribution of the visit times may be extremely uneven. Those actions with greater selecting probabilities are visited much more times. For example, given an episode of one million steps, some actions are visited more than 250,000 times, while most actions are visited merely hundreds or even tens of times as shown in Figure 5.7(h).

According to the above analysis, although ϵ-greedy policies have a certain exploration ability, the exploration ability is limited when ϵ is small. Therefore, sufficient samples are required to ensure all the state-action pairs can be visited. Note that our task here is to find out the optimal policy for *every* state, which demands many visits for every state-action pair. In practice, if we are only interested in finding a good path from a given state to the target state, fewer samples are required. Moreover, one common technique used in practice is to set ϵ to be large initially to enhance exploration and gradually reduce it to ensure optimality of the final policy.

5.6 Summary

This chapter introduced the first class of model-free RL algorithms: MC learning methods. We first introduced the idea of MC estimation and how to estimate the expectation of a random variable using samples. When models are unavailable, the model-based component in the policy iteration algorithm can be replaced by a model-free MC estimation component. This leads to the simplest MC learning algorithm, which is called MC Basic in this book. The complication of MC learning appears when we would like to use samples more efficiently or avoid unrealistic assumptions on exploring starts. In particular, three MC-based algorithms were introduced in this chapter.

– MC Basic: This is the simplest MC learning algorithm. Although this algorithm is too simple to be used in practice, it clearly demonstrates the core idea of MC learning. This algorithm is obtained by simply replacing the model-based policy evaluation step in the policy iteration algorithm with a model-free MC-based estimation component. It is guaranteed that, given sufficient samples, this algorithm can converge to optimal policies and optimal state values.

– MC Exploring Starts: This algorithm is a variant of MC Basic. It can be obtained from the MC Basic algorithm by using first-visit or every-visit strategies to use experience samples more efficiently.

– MC ϵ-Greedy: This algorithm is a variant of the MC Exploring Starts algorithm. Specifically, in the policy improvement step, it searches for the best ϵ-greedy policies instead of greedy policies. In this way, the exploration ability of the policy is enhanced and hence the unrealistic assumption on exploring starts can be removed.

One important tradeoff of ϵ-greedy policies is exploration and exploitation. Specifically, when we increase the value of ϵ, the exploration ability of the ϵ-greedy would increase. However, the exploitation of the greedy action would decrease. On the other hand, if we reduce the value of ϵ so that we can fully exploit the greedy actions, the exploration ability of the policy would be compromised and hence the assumption on exploring starts becomes necessary.

5.7 Q&A

– Q: What is Monte Carlo estimation?

A: Monte Carlo estimation refers to a broad class of techniques that use stochastic samples to solve approximation problems.

– Q: What is the mean estimation problem?

A: The mean estimation problem refers to calculating the expectation of a random variable based on stochastic samples.

– Q: What are the two ways to solve the mean estimation problem?

A: The two ways are model-based and model-free. In particular, if the probability distribution function is known, the expectation can be calculated based on its definition. If the probability distribution is unknown, we can use Monte Carlo estimation to approximate the expectation. Such kind of approximation is accurate when the number of samples is large.

– Q: How is mean estimation related to reinforcement learning?

A: A core problem in reinforcement learning is calculating state values and action values. Both state value and action value are defined as expected values of returns. Hence, estimating state or action values is essentially a mean estimation problem.

– Q: What is the basic idea to obtain model-free reinforcement learning algorithms?

A: The basic idea is to convert the model-based policy iteration algorithm to a model-free one. In particular, the policy iteration algorithm aims to calculate values based on the system model. We can also use Monte Carlo estimation to directly approximate action values based on episodes.

– Q: What are initial-visit, first-visit, and every-visit strategies, respectively?

  A: They are different strategies to use the samples in an episode. An episode may visit many state-action pairs. The initial-visit strategy uses the entire episode to estimate the action value of the initial state-action pair. In addition, one state action pair may be visited multiple times in an episode. If we only count the first-time visit, such kind of strategy is called first-visit. If every time a state-action pair is visited and the rest of the episode is used to estimate its action value, such a strategy is called every-visit.

– Q: What is the difference between MC Exploring Starts and MC Basic?

  A: MC Exploring Starts is a sample-efficient version of MC Basic. It introduces the first-visit or every-visit strategies to use episodes of samples more efficiently and, in the meantime, convert the policy update step to episode-by-episode.

– Q: What is exploring starts? Why is it important?

  A: Exploring starts is a requirement that an infinite number of (or sufficiently many) episodes must be generated starting from every state-action pair. In theory, exploring starts is necessary to find optimal policies. Only if every action value for every state is well explored, can we guarantee to select the optimal actions correctly. On the contrary, if an action is not explored, this action may happen to be the optimal one and hence be missed.

– Q: What is the idea of removing the requirement of exploring starts?

  A: The fundamental idea to remove the requirement of exploring starts is to make the policies soft. On the one hand, soft policies are stochastic so that an episode can visit state-action pairs many times. On the other hand, soft policies still exploit greedy actions to search for optimal policies.

– Q: Can an ϵ-greedy policy be optimal?

  A: The answer is both yes and no. By yes, it means that, if given sufficient samples, the MC ϵ-greedy algorithm can converge to an ϵ-greedy policy that is optimal among the set of all ϵ-greedy policies. By no, it means that the finally obtained policy is not optimal among all possible policies.

– Q: What does it mean when we say one ϵ-greedy policy is consistent with another greedy policy?

  A: An ϵ-greedy policy is called consistent with another greedy policy if the actions with the greatest probabilities in the ϵ-greedy policy are the same as those in the greedy one.

– Q: Is it possible to use one episode to visit all state-action pairs?

  A: Yes, if the policy is soft such as ϵ-greedy.

– Q: What is the relationship between MC Basic, MC Exploring Starts, and MC ϵ-greedy?

A: MC Basic is the simplest MC-based reinforcement learning algorithm. It is important because it reflects the core idea of model-free MC-based reinforcement learning. MC Exploring Starts is a variant of MC Basic by adjusting the sample usage strategy. Furthermore, MC ϵ-greedy is a variant of MC Exploring Starts by removing the requirement of exploring starts. Therefore, the basic idea is simple, but the complication appears when we would like to achieve better performance. Nevertheless, it is important to split the core idea from the complication that may be distracting for understanding a problem for beginners.