Chapter 2
Reinforcement Learning

Zheng Wen

2.1 Introduction

Reinforcement learning (RL) (Sutton & Barto, 2018) is a subfield of machine learning concerned with how an *agent* (or decision-maker) should learn to take actions to maximize some notion of cumulative reward while interacting with an *environment*, as is illustrated in Fig. 2.1. Specifically, at each time step, the agent first adaptively chooses an action based on its prior knowledge, past observations, and past rewards; then, it will receive a new observation and a new reward from the environment. In general, the agent's observations and rewards are stochastic and statistically dependent on its chosen action and its *state* in the environment. In most RL problems, the environment is only partially known and the agent cannot compute an optimal or near-optimal *policy* based on its prior knowledge. Consequently, it needs to learn to take optimal or near-optimal actions while interacting with the environment.

RL is one of the three basic machine learning (Friedman et al., 2001; Bishop, 2006) paradigms, alongside *supervised learning* and *unsupervised learning*. While supervised learning and unsupervised learning algorithms aim to learn from labeled or unlabeled datasets, in RL problems, the agent aims to learn to take good actions from its interactions with a usually partially known environment. Due to its generality, RL has also been studied in many other fields, such as operations research, control theory, game theory, multi-agent systems, information theory, and statistics. From the perspective of operations research and control theory, RL is closely related to dynamic programming (DP), approximate dynamic programming (ADP), and optimal control (Bertsekas, 2000, 2011; Powell, 2007). Specifically,

Z. Wen (✉)
DeepMind, Mountain View, CA, USA

X. Chen et al. (eds.), *The Elements of Joint Learning and Optimization in Operations Management*, Springer Series in Supply Chain Management 18,
https://doi.org/10.1007/978-3-031-01926-5_2

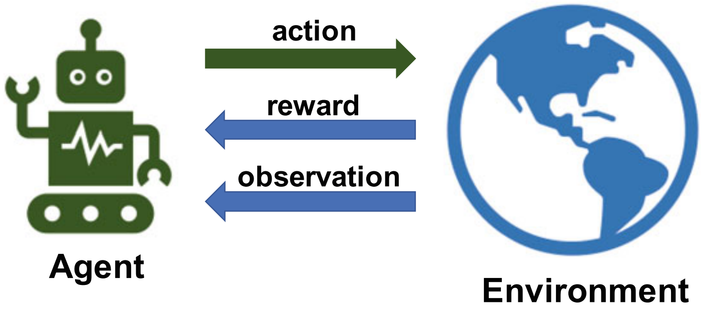

Fig. 2.1 Illustration of reinforcement learning (RL) problems, in which the agent chooses actions and receives the observations and rewards from the environment. In most RL problems, the agent's observation includes the next state it will transit to

similar to classical DP problems that aim to compute an optimal policy in Markov decision processes (MDPs), basic RL problems are usually formulated as problems that aim to learn an optimal or near-optimal policy in MDPs. The main difference is that, in DP problems the agent is assumed to know the model of the MDP and hence can compute an optimal policy based on that model; however, in most RL problems the agent does not fully know the model and has to learn to take optimal or near-optimal actions.

One key challenge that arises in RL, but not in supervised and unsupervised learning, is the *exploration-exploitation* trade-off. Specifically, in RL, to obtain more reward, an agent should prefer actions that it *has found* effective in producing reward (exploitation). However, to discover such actions, the agent needs to try actions that *might be* effective in producing reward, or actions that might provide *useful information* about an optimal or near-optimal policy (exploration). In other words, the agent needs to exploit what it has already learned to obtain reward, but it also has to explore to make better action selections in the future. If an agent exclusively pursues exploration or exploitation, then it can easily fail or lose a lot of reward in some problems. A successful RL agent should carefully balance the exploration-exploitation trade-off by designing an appropriate exploration scheme.

In addition to the exploration-exploitation trade-off, another challenge for RL is that modern RL problems tend to have intractably large state space and/or action space. For example, in an online recommendation system, the state might include the inventory levels of all items, and the action might be an ordered list of items chosen to display. Hence, both the cardinalities of the state space and the action space can be enormous. For such large-scale RL problems, we cannot expect to learn an optimal policy with limited time, data, and computational resources. Instead, our goal is to learn a good approximate solution within limited time and using limited data and computational resources. Many such agents have been built for large-scale RL problems. In particular, deep reinforcement learning (DRL) is a subfield of RL aiming to build agents based on (deep) neural networks that can learn approximate solutions for large-scale RL problems.

RL has extensive applications in many fields, such as online recommendation systems (Chen et al., 2019; Kveton et al., 2015), robotics (Kober et al., 2013), information retrieval (Zhang et al., 2020), energy management systems (Kuznetsova et al., 2013; Wen et al., 2015), revenue management (Gosavii et al., 2002), and financial engineering (Fischer, 2018). In the past decade, several high-performance DRL agents have been built for games like Go, Chess, and Atari games (Silver et al., 2016, 2017b, 2017a; Schrittwieser et al., 2020). Many of them have achieved a performance comparable to or even better than that of a professional human player. In particular, the AlphaGo agent (Silver et al., 2016) beat a world champion in the game of Go. Many researchers are working on extending these agents built for games to other exciting application areas.

The remainder of this chapter is organized as follows: in Sect. 2.2, we briefly review Markov decision processes (MDPs) and dynamic programming (DP) solutions. In Sect. 2.3, we provide a high-level review of some classical RL algorithms. We also discuss two key issues for RL algorithm design: exploration scheme design and approximate solution methods for large-scale RL problems, in that section. Finally, we conclude this chapter and provide pointers for further reading in Sect. 2.4.

2.2 Markov Decision Process and Dynamic Programming

Markov decision processes (MDPs) are stochastic control processes used in a variety of optimization and machine learning problems where the outcomes (e.g., rewards, next states) are partly random and partly controlled by the agent. They provide a framework for modeling decision making in dynamic systems. As we have mentioned in Sect. 2.1, basic RL problems can be formulated as problems in which an agent aims to learn an optimal or near-optimal policy in partially known MDPs. Several classes of MDPs, such as finite-horizon MDPs, infinite-horizon discounted MDPs, and infinite-horizon average-reward MDPs, have been widely studied in the literature. In this section, we will briefly review two classical classes of MDPs: the finite-horizon MDPs in Sect. 2.2.1 and the infinite-horizon discounted MDPs in Sect. 2.2.2. Interested readers might refer to (Bertsekas, 2000, 2011) for further reading.

When the model of an MDP is completely known, its optimal policy can be computed by dynamic programming (DP) algorithms. Though classical DP algorithms are of limited utility in RL due to the assumption that the MDP model is completely known, it does provide a foundation for understanding RL algorithms described later in this chapter. In this section, we will also briefly review the DP algorithms for the finite-horizon MDPs and the infinite-horizon discounted MDPs.

2.2.1 Finite-Horizon Markov Decision Process

A finite-horizon MDP is characterized by a tuple $\mathcal{M}_\mathrm{F} = (\mathcal{S}, \mathcal{A}, P, r, H, \rho)$, where $\mathcal{S}$ is a finite state space, $\mathcal{A}$ is a finite action space, P and r, respectively, encode the transition model and the reward model, H is the finite time horizon, and ρ is a probability distribution over the state space $\mathcal{S}$. At the first period $h = 1$, the initial state s_1 is independently drawn from the distribution ρ. Then, at each period $h = 1, 2, \ldots, H$, if the agent takes action $a_h \in \mathcal{A}$ at state $s_h \in \mathcal{S}$, then it will receive a random reward $r_h \in \Re$ conditionally independently drawn from the reward distribution $r\left(\cdot|s_h, a_h\right)$. Moreover, for period $h < H$, the agent will transit to state $s' \in \mathcal{S}$ in the next period $h + 1$ with probability $P\left(s'\middle|s_h, a_h\right)$. The finite-horizon MDP will terminate after the agent receives reward r_H at period H. To simplify the exposition, we use $\bar{r}(s, a)$ to denote the mean of the reward distribution $r(\cdot|s, a)$ for all state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$. We also define $\mathcal{H} = \{1, 2, \ldots, H\}$ to denote the set of time periods.

In a finite-horizon MDP, the agent's goal is to maximize its expected total reward

$$\mathbb{E}\left[\sum_{h=1}^{H} r_h\right] \tag{2.1}$$

by *adaptively* choosing action a_h for each period $h = 1, \ldots, H$ based on its observations so far, which can be represented as $(s_1, a_1, r_1, s_2, \ldots, s_{h-1}, a_{h-1}, r_{h-1}, s_h)$. Furthermore, since $(s_1, a_1, r_1, s_2, \ldots, s_{h-1}, a_{h-1}, r_{h-1})$ is conditionally independent of future rewards and transitions given the current state s_h and the period h (the Markov property), the agent only needs to choose action a_h based on the state-period pair (s_h, h). This motivates the notion of policy for a finite-horizon MDP. Specifically, a (randomized) policy $\pi : \mathcal{S} \times \mathcal{H} \rightarrow \Delta^{\mathcal{A}}$ is defined as a mapping from the state-period pairs to probability distributions over the action space $\mathcal{A}$. Note that $\Delta^{\mathcal{A}}$ denotes the set of probability distributions (i.e., the probability simplex) over the action space $\mathcal{A}$. Under a policy π, if the agent is at state s_h at period h, then it will choose action $a_h = a$ with probability $\pi(a|s_h, h)$. We say a policy π is deterministic if $\pi(a|s, h) \in \{0, 1\}$ for all action $a \in \mathcal{A}$ and all state-period pair $(s, h) \in \mathcal{S} \times \mathcal{H}$. That is, at all state-period pair (s, h), the agent will choose one action with probability 1 under policy π. With a little bit abuse of notation, for a deterministic policy π, sometimes we use $\pi(s, h)$ to denote the action it chooses with probability 1 at (s, h).

For each policy π, we define its *state value function* $V^\pi : \mathcal{S} \times \mathcal{H} \rightarrow \Re$ as

$$V^\pi(s, h) = \mathbb{E}_\pi\left[\sum_{h'=h}^{H} r_{h'}\middle|s_h = s\right], \quad \forall (s, h) \in \mathcal{S} \times \mathcal{H}, \tag{2.2}$$

where the subscript π in notation $\mathbb{E}_\pi$ indicates the expectation is taken under the stochastic process defined by policy π. Notice that each policy π defines a stochastic process evolving as follows: at each period $h \in \mathcal{H}$ with state s_h, the agent first chooses action $a_h \sim \pi(\cdot|s_h, h)$, then it will receive a reward $r_h \sim r(\cdot|s_h, a_h)$, and if $h < H$, it will transit to a new state $s_{h+1} \sim P(\cdot|s_h, a_h)$ in the next period $h + 1$.

$V^\pi(s, h)$ is the expected total future reward if the agent starts at state s at period h and chooses actions according to policy π.

Similarly, we define the *state-action value function* $Q^\pi : \mathcal{S} \times \mathcal{H} \times \mathcal{A}$ for policy π as

$$Q^\pi(s, h, a) = \mathbb{E}_\pi \left[\sum_{h'=h}^{H} r_{h'} \middle| s_h = s, a_h = a \right], \quad \forall (s, h, a) \in \mathcal{S} \times \mathcal{H} \times \mathcal{A} \tag{2.3}$$

that is, $Q^\pi(s, h, a)$ is the expected total future reward if the agent starts at state s at period h, chooses action a at period h and chooses actions according to policy π for all period $h' \geq h + 1$. By definition of V^π and Q^π, we have the following equation for any $(s, h) \in \mathcal{S} \times \mathcal{H}$ and $(s, h, a) \in \mathcal{S} \times \mathcal{H} \times \mathcal{A}$:

$$V^\pi(s, h) = \sum_{a \in \mathcal{A}} \pi(a|s, h) Q^\pi(s, h, a)$$

$$Q^\pi(s, h, a) = \begin{cases} \bar{r}(s, a) + \sum_{s' \in \mathcal{S}} P\left(s' \middle| s, a\right) V^\pi(s', h + 1) & \text{if } h < H \\ \bar{r}(s, a) & \text{if } h = H \end{cases}. \tag{2.4}$$

Note that Eq. (2.4) is referred to as the *Bellman equation* under policy π. We can rewrite the Bellman equation just in V^π or Q^π, e.g.,

$$V^\pi(s, h) = \begin{cases} \sum_{a \in \mathcal{A}} \pi(a|s, h) \left[\bar{r}(s, a) + \sum_{s' \in \mathcal{S}} P\left(s' \middle| s, a\right) V^\pi(s', h + 1) \right] & \text{if } h < H \\ \sum_{a \in \mathcal{A}} \pi(a|s, h) \bar{r}(s, a) & \text{if } h = H \end{cases}. \tag{2.5}$$

We also define the *optimal state value function* $V^* : \mathcal{S} \times \mathcal{H} \to \Re$ as

$$V^*(s, h) = \max_\pi V^\pi(s, h), \quad \forall (s, h) \in \mathcal{S} \times \mathcal{H}, \tag{2.6}$$

which is the maximum[1] (optimal) expected total future reward if the agent starts at state s at period h. Similarly, we define the *optimal state-action value function* $Q^* : \mathcal{S} \times \mathcal{H} \times \mathcal{A} \to \Re$ as

$$Q^*(s, h, a) = \max_\pi Q^\pi(s, h, a), \quad \forall (s, h, a) \in \mathcal{S} \times \mathcal{H} \times \mathcal{A}, \tag{2.7}$$

which is the maximum (optimal) expected total future reward if the agent starts at state s at period h and chooses action a at period h. Similarly, we have the following Bellman equation for the optimal value function V^* and Q^*:

$$V^*(s, h) = \max_{a \in \mathcal{A}} Q^*(s, h, a)$$

[1] Since we assume the time horizon and the cardinalities of $\mathcal{S}$ and $\mathcal{A}$ are all finite, the maximum is always achieved.

$$Q^*(s,h,a) = \begin{cases} \bar{r}(s,a) + \sum_{s' \in \mathcal{S}} P(s'|s,a) V^*(s', h+1) & \text{if } h < H \\ \bar{r}(s,a) & \text{if } h = H \end{cases}. \tag{2.8}$$

One can prove the above Bellman equation by backward induction, similar to Proposition 1.3.1 in Bertsekas (2000).

We say a policy π' is optimal at a state-period pair (s, h) if

$$V^{\pi'}(s,h) = V^*(s,h).$$

It turns out that there exist policies that are simultaneously optimal for all state-period pairs. Specifically, one such policy is a deterministic policy π^* satisfying[2]

$$\pi^*(s,h) \in \arg\max_{a \in \mathcal{A}} Q^*(s,h,a), \quad \forall (s,h) \in \mathcal{S} \times \mathcal{H},$$

recall that under a deterministic policy π^*, $\pi^*(s,h)$ is the action chosen at state-period pair (s,h). Note that by definition, we have

$$Q^*(s,h,\pi^*(s,h)) = \max_{a \in \mathcal{A}} Q^*(s,h,a) = V^*(s,h), \quad \forall (s,h) \in \mathcal{S} \times \mathcal{H}.$$

To prove that π^* is simultaneously optimal for all state-period pairs, we prove that $V^{\pi^*}(s,h) = V^*(s,h)$ for all $(s,h) \in \mathcal{S} \times \mathcal{H}$ by backward induction in h:

- For $h = H$, we have $Q^*(s,h,a) = \bar{r}(s,a)$. Consequently, we have $\pi^*(s,h) \in \arg\max_{a \in \mathcal{A}} \bar{r}(s,a)$, so we have

$$V^{\pi^*}(s,h) = \bar{r}(s,\pi^*(s,h)) = Q^*(s,h,\pi^*(s,h)) = V^*(s,h).$$

- For any $h < H$, assume that $V^{\pi^*}(s,h+1) = V^*(s,h+1)$ for all $s \in \mathcal{S}$, we now prove that $V^{\pi^*}(s,h) = V^*(s,h)$ for all $s \in \mathcal{S}$. Note that

$$\begin{aligned} V^{\pi^*}(s,h) &= Q^{\pi^*}(s,h,\pi^*(s,h)) \\ &= \bar{r}(s,\pi^*(s,h)) + \sum_{s' \in \mathcal{S}} P(s'|s,\pi^*(s,h)) V^{\pi^*}(s', h+1) \\ &= \bar{r}(s,\pi^*(s,h)) + \sum_{s' \in \mathcal{S}} P(s'|s,\pi^*(s,h)) V^*(s', h+1) \\ &= Q^*(s,h,\pi^*(s,h)) = V^*(s,h), \end{aligned} \tag{2.9}$$

[2] In general, a randomized policy $\tilde{\pi}$ is optimal if $\mathrm{supp}\tilde{\pi}(\cdot|s,h) \subseteq \arg\max_{a \in \mathcal{A}} Q^*(s,h,a)$ for all (s,h), where $\mathrm{supp}\tilde{\pi}(\cdot|s,h)$ is the support of the distribution $\tilde{\pi}(\cdot|s,h)$.

where the first two equalities follow from the Bellman equation under π^*, the third equality follows from the induction hypothesis, the fourth equality follows from the Bellman equation for the optimal value function, and the last equality follows from the definition of π^*, as discussed above.

2.2.1.1 Dynamic Programming Solution

Based on our discussion above, for a finite-horizon MDP $\mathcal{M}_{\mathrm{F}}$, we can compute a deterministic optimal policy π^* based on the dynamic programming (DP) algorithm below:

DP algorithm for finite-horizon MDP

Initialization: set $V^*(s, H+1) = 0$ for all $s \in \mathcal{S}$
Step 1: for each $h = H, H-1, \ldots, 1$:
compute

$$Q^*(s,h,a) = \bar{r}(s,a) + \sum_{s' \in \mathcal{S}} P(s'|s,a) V^*(s', h+1) \quad \forall (s,a) \in \mathcal{S} \times \mathcal{A}$$

and

$$V^*(s,h) = \max_{a \in \mathcal{A}} Q^*(s,h,a) \quad \forall s \in \mathcal{S}$$

Step 2: choose a deterministic policy π^* s.t.

$$\pi^*(s,h) \in \arg\max_{a \in \mathcal{A}} Q^*(s,h,a) \quad \forall (s,h) \in \mathcal{S} \times \mathcal{H}$$

Return π^*

2.2.2 Discounted Markov Decision Process

An infinite-horizon discounted Markov decision process (MDP) is characterized by a tuple $\mathcal{M}_{\mathrm{D}} = (\mathcal{S}, \mathcal{A}, P, r, \gamma, \rho)$, where $\mathcal{S}$ is a finite state space, $\mathcal{A}$ is a finite action space, P and r, respectively, encode the transition model and the reward model, $\gamma \in (0,1)$ is a discrete-time discount factor, and ρ is a probability distribution over the state space $\mathcal{S}$. At the first period $t = 1$, the initial state s_1 is independently drawn from the distribution ρ. Then, at each period $t = 1, 2, \ldots$, if the agent takes action $a_t \in \mathcal{A}$ at state $s_t \in \mathcal{S}$, then it will receive a random reward $r_t \in [0,1]$ conditionally independently drawn from the reward distribution $r(\cdot|s_t, a_t)$ and will transit to state $s' \in \mathcal{S}$ in the next period $t+1$ with probability $P\left(s' \middle| s_t, a_t\right)$. To simplify the exposition, we use $\bar{r}(s,a)$ to denote the mean of the reward distribution $r(\cdot|s,a)$ for all state-action pair $(s,a) \in \mathcal{S} \times \mathcal{A}$. Notice that we assume the random reward $r_t \in [0,1]$ to simplify the exposition.

In an infinite-horizon discounted MDP, the agent's goal is to maximize its expected total discounted reward[3]

$$\mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t\right] \tag{2.10}$$

by adaptively choosing action a_t for period $t = 1, 2, \ldots$ based on its past observations. Similarly as the finite-horizon MDPs, the past observations are conditionally independent of future rewards and transitions given the current state s_t (the Markov property). Moreover, the discounted MDPs are also *time-invariant* in the sense that for any $\tau \geq 1$ and any state $s \in \mathcal{S}$,

$$\max \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \middle| s_1 = s\right] \quad \text{and} \quad \max \mathbb{E}\left[\sum_{t=\tau}^{\infty} \gamma^{t-\tau} r_t \middle| s_\tau = s\right]$$

are two equivalent problems. Thus, the agent only needs to choose action a_t based on the current state s_t. This motivates the notion of policy for a discounted MDP. Specifically, a (randomized) policy $\pi : \mathcal{S} \rightarrow \Delta^{\mathcal{A}}$ is defined as a mapping from the state space to probability distributions over the action space $\mathcal{A}$. Under a policy π, if the agent is at state s_t, then it will choose action $a_t = a$ with probability $\pi(a|s_t)$. Similarly, if π is a deterministic policy, we use $\pi(s)$ to denote the action it chooses with probability 1 at state s.

For each policy π, we define its *state value function* $V^\pi : \mathcal{S} \rightarrow \Re$ as

$$V^\pi(s) = \mathbb{E}_\pi\left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \middle| s_1 = s\right], \quad \forall s \in \mathcal{S}, \tag{2.11}$$

where the subscript π in notation $\mathbb{E}_\pi$ indicates the expectation is taken under the stochastic process defined by policy π. Specifically, note that each policy π defines a stochastic process evolving as follows: at each period $t \in 1, 2, \ldots$ with state s_t, the agent first chooses action $a_t \sim \pi(\cdot|s_t)$, then it will receive a reward $r_t \sim r(\cdot|s_t, a_t)$ and transit to a new state $s_{t+1} \sim P(\cdot|s_t, a_t)$ in the next time $t+1$. $V^\pi(s)$ is the expected total discounted reward if the agent starts at state s and chooses actions according to policy π.

Similarly, we define the *state-action value function* $Q^\pi : \mathcal{S} \times \mathcal{A}$ for policy π as

$$Q^\pi(s,a) = \mathbb{E}_\pi\left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \middle| s_1 = s, a_1 = a\right], \quad \forall (s,a) \in \mathcal{S} \times \mathcal{A} \tag{2.12}$$

[3] Notice that we choose the convention that t starts from 1, thus, the discount at time t is γ^{t-1}. If t starts from 0, then the discount at time t should be γ^t. We choose the convention that t starts from 1 to be consistent with the finite-horizon MDPs.

that is, $Q^{\pi}(s,a)$ is the expected total discounted reward if the agent starts at state s, chooses action a at the first time period, and chooses actions according to policy π at subsequent time periods. By definition of V^{π} and Q^{π}, we have the following equation for any $s \in \mathcal{S}$ and $(s,a) \in \mathcal{S} \times \mathcal{A}$:

$$\begin{aligned} V^{\pi}(s) &= \sum_{a \in \mathcal{A}} \pi(a|s) Q^{\pi}(s,a) \\ Q^{\pi}(s,a) &= \bar{r}(s,a) + \gamma \sum_{s' \in \mathcal{S}} P\left(s' \middle| s,a\right) V^{\pi}(s'). \end{aligned} \tag{2.13}$$

Note that Eq. (2.13) is referred to as the *Bellman equation* under policy π for discounted MDPs. We can rewrite the Bellman equation just in V^{π} or Q^{π}, e.g.,

$$V^{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left[\bar{r}(s,a) + \gamma \sum_{s' \in \mathcal{S}} P\left(s' \middle| s,a\right) V^{\pi}(s') \right]. \tag{2.14}$$

To simplify the exposition, we define the dynamic programming (DP) operator under policy π, $\mathbb{T}_{\pi}$, as

$$(\mathbb{T}_{\pi} V)(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left[\bar{r}(s,a) + \gamma \sum_{s' \in \mathcal{S}} P\left(s' \middle| s,a\right) V(s') \right], \tag{2.15}$$

where $V : \mathcal{S} \to \Re$ is a real-valued function with domain $\mathcal{S}$. Notice that by definition, $\mathbb{T}_{\pi} V : \mathcal{S} \to \Re$ is also a real-valued function with domain $\mathcal{S}$. With the shorthand notation $\mathbb{T}_{\pi}$, we can rewrite the Bellman equation 2.14 as $V^{\pi} = \mathbb{T}_{\pi} V^{\pi}$.

We also define the *optimal state value function* $V^* : \mathcal{S} \to \Re$ as

$$V^*(s) = \max_{\pi} V^{\pi}(s), \quad \forall s \in \mathcal{S}, \tag{2.16}$$

which is the maximum (optimal) expected total discounted reward if the agent starts at state s. Similarly, we define the *optimal state-action value function* $Q^* : \mathcal{S} \times \mathcal{A} \to \Re$ as

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a), \quad \forall (s,a) \in \mathcal{S} \times \mathcal{A}, \tag{2.17}$$

which is the maximum (optimal) expected total discounted reward if the agent starts at state s and chooses action a at the first period. Similarly, we have the following Bellman equation for the optimal value function V^* and Q^*:

$$\begin{aligned} V^*(s) &= \max_{a \in \mathcal{A}} Q^*(s,a) \\ Q^*(s,a) &= \bar{r}(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^*(s'). \end{aligned} \tag{2.18}$$

Similarly, we can rewrite the above Bellman equation just in V^* or Q^*, e.g.,

$$V^*(s) = \max_{a \in \mathcal{A}} \left[\bar{r}(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^*(s') \right]. \tag{2.19}$$

We define the DP operator $\mathbb{T}$ as

$$(\mathbb{T}V)(s) = \max_{a \in \mathcal{A}} \left[\bar{r}(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V(s') \right], \tag{2.20}$$

where V is a real-valued function with domain $\mathcal{S}$. With this shorthand notation, we can rewrite the Bellman equation (2.19) as $V^* = \mathbb{T}V^*$.

We have shown that $V^* = \mathbb{T}V^*$, in other words, V^* is one solution of the equation $V = \mathbb{T}V$. We are also interested in if V^* is the *unique* solution of that equation. It turns out that for the setting considered in this subsection, V^* is the unique bounded function satisfying the equation $V = \mathbb{T}V$. Interested readers might refer to Proposition 1.2.3 in Bertsekas (2011) for a proof.[4] Similarly, we can prove that V^π is the unique bounded function satisfying the equation $V = \mathbb{T}_\pi V$.

We say a policy π' is optimal at a state $s \in \mathcal{S}$ if

$$V^{\pi'}(s) = V^*(s).$$

It turns out that there exist policies that are simultaneously optimal for all states. Specifically, one such policy is a deterministic policy π^* satisfying

$$\pi^*(s) \in \arg\max_{a \in \mathcal{A}} Q^*(s,a), \quad \forall s \in \mathcal{S}$$

recall that under a deterministic policy π^*, $\pi^*(s)$ is the action chosen at state s. Interested readers might refer to Proposition 1.2.5 in Bertsekas (2011) for a proof.

In the remainder of this subsection, we briefly discuss two dynamic programming algorithms for discounted MDPs: *value iteration* and *policy iteration*. Specifically, value iteration can compute a good approximation of V^* in finite steps; and policy iteration can compute an optimal policy π^* in finite steps.

2.2.2.1 Value Iteration

Value iteration is one algorithm that asymptotically computes V^* and can compute a good approximation of V^* in finite steps. It is based on the following two observations: first, V^* is a *fixed point* of the DP operator $\mathbb{T}$, since $V^* = \mathbb{T}V^*$. Also, based on the discussion above, we know that it is the unique bounded fixed point. Second, $\mathbb{T}$ is a *contraction mapping* with respect to the L_∞ norm. Specifically, we have that

[4] Chapter 1 in Bertsekas (2011) considers a cost minimization setting, which is equivalent to the reward maximization setting considered in this chapter if we define the cost as one minus the reward.

$$\|\mathbb{T}V_1 - \mathbb{T}V_2\|_\infty \leq \gamma \|V_1 - V_2\|_\infty , \tag{2.21}$$

for any $V_1, V_2 : \mathcal{S} \rightarrow \Re$. Note that for any $V : \mathcal{S} \rightarrow \Re$, $\|V\|_\infty = \max_{s \in \mathcal{S}} |V(s)|$. To see why Eq. (2.21) holds, notice that for any $s \in \mathcal{S}$, we have

$$\begin{aligned} |(\mathbb{T}V_1)(s) - (\mathbb{T}V_2)(s)| = & \left|\max_{a \in \mathcal{A}} \left[\bar{r}(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V_1(s')\right]\right. \\ & \left. - \max_{a \in \mathcal{A}} \left[\bar{r}(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V_2(s')\right]\right| \\ & \leq \gamma \max_{a \in \mathcal{A}} \left|\sum_{s' \in \mathcal{S}} P(s'|s,a) \left(V_1(s') - V_2(s')\right)\right| \\ & \leq \gamma \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a) \left|V_1(s') - V_2(s')\right| \\ & \leq \gamma \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a) \|V_1 - V_2\|_\infty \\ & = \gamma \|V_1 - V_2\|_\infty . \end{aligned}$$

Consequently, we have

$$\|\mathbb{T}V_1 - \mathbb{T}V_2\|_\infty = \max_{s \in \mathcal{S}} |(\mathbb{T}V_1)(s) - (\mathbb{T}V_2)(s)| \leq \gamma \|V_1 - V_2\|_\infty .$$

Similarly, we can prove that for any policy π, we have

$$\|\mathbb{T}_\pi V_1 - \mathbb{T}_\pi V_2\|_\infty \leq \gamma \|V_1 - V_2\|_\infty$$

for any $V_1, V_2 : \mathcal{S} \rightarrow \Re$.

Moreover, notice that for any $V : \mathcal{S} \rightarrow \Re$, by definition, $\mathbb{T}V$ is also a real-valued function with domain $\mathcal{S}$. Thus, for any integer $k \geq 1$, we can recursively define $\mathbb{T}^{k+1}V = \mathbb{T}\left(\mathbb{T}^k V\right)$. Since $\mathbb{T}$ is a contraction mapping with respect to the L_∞ norm, and V^* is the unique bounded fixed point of $\mathbb{T}$, we have the following result:

Proposition 2.1 *For any bounded function $V : \mathcal{S} \rightarrow \Re$, we have $\lim_{k \rightarrow \infty} \mathbb{T}^k V = V^*$. Moreover, for any integer $k = 1, 2, \ldots$, we have $\left\|\mathbb{T}^k V - V^*\right\|_\infty \leq \gamma^k \|V - V^*\|_\infty$.*

Proof Since $\mathbb{T}$ is a contraction mapping with respect to L_∞ norm, for any integer $k = 1, 2, \ldots$, we have

$$\left\|\mathbb{T}^k V - V^*\right\|_\infty = \left\|\mathbb{T}(\mathbb{T}^{k-1}V) - \mathbb{T}V^*\right\|_\infty \leq \gamma \left\|\mathbb{T}^{k-1}V - V^*\right\|_\infty .$$

Thus, by induction, we have $\left\|\mathbb{T}^k V - V^*\right\|_\infty \leq \gamma^k \|V - V^*\|_\infty$. This implies that $\lim_{k \rightarrow \infty} \left\|\mathbb{T}^k V - V^*\right\|_\infty = 0$ and hence $\lim_{k \rightarrow \infty} \mathbb{T}^k V = V^*$. □

The above proposition implies the following value iteration algorithm:

Value iteration algorithm

```
**Input:** number of iterations $K$
**Initialization:** choose $V_0 : \mathcal{S} \to \Re$ s.t. $V_0(s) = 0$ for all $s \in \mathcal{S}$
**Value Iteration:** for each $k = 1, 2, \ldots, K$, compute $V_k \leftarrow \mathbb{T} V_{k-1}$
**Return** $V_K$
```

As we have discussed above, as $K \to \infty$, V_K returned by the value iteration algorithm converges to V^*. For a finite K, V_K is an approximation of V^*. Based on Proposition 2.1, we have

$$\|V_K - V^*\|_\infty \overset{(a)}{=} \|\mathbb{T}^K V_0 - V^*\|_\infty \overset{(b)}{\leq} \gamma^K \|V_0 - V^*\|_\infty \overset{(c)}{=} \gamma^K \|V^*\|_\infty \overset{(d)}{\leq} \frac{\gamma^K}{1-\gamma},$$

where (a) follows from the definition of V_K, (b) follows from Proposition 2.1, (c) follows from the fact that $V_0(s) = 0$ for all $s \in \mathcal{S}$, and (d) follows from the fact that $r_t \in [0, 1]$ and hence $0 \leq V^*(s) \leq \frac{1}{1-\gamma}$ for all $s \in \mathcal{S}$. Consequently, if we choose a sufficiently large K, the value iteration algorithm will compute a good approximation of V^*.

Finally, we show that when K is sufficiently large, then V_K induces a near-optimal policy. Specifically, consider a policy π_K satisfying[5] $\mathbb{T}_{\pi_K} V_K = \mathbb{T} V_K$, then we have

$$\begin{aligned}
\|V^{\pi_K} - V^*\|_\infty &= \|V^{\pi_K} - \mathbb{T}_{\pi_K} V_K + \mathbb{T} V_K - V^*\|_\infty \\
&\overset{(a)}{\leq} \|V^{\pi_K} - \mathbb{T}_{\pi_K} V_K\|_\infty + \|\mathbb{T} V_K - V^*\|_\infty \\
&= \|\mathbb{T}_{\pi_K} V^{\pi_K} - \mathbb{T}_{\pi_K} V_K\|_\infty + \|\mathbb{T} V_K - \mathbb{T} V^*\|_\infty \\
&\overset{(b)}{\leq} \gamma \|V^{\pi_K} - V_K\|_\infty + \gamma \|V_K - V^*\|_\infty \\
&\overset{(c)}{\leq} \gamma \|V^{\pi_K} - V^*\|_\infty + 2\gamma \|V_K - V^*\|_\infty,
\end{aligned}$$

where (a) and (c) follow from the triangular inequality, and (b) follows from the contraction mapping. Consequently, we have

$$\|V^{\pi_K} - V^*\|_\infty \leq \frac{2\gamma}{1-\gamma} \|V_K - V^*\|_\infty \leq \frac{2\gamma^{K+1}}{1-\gamma} \|V^*\|_\infty \leq \frac{2\gamma^{K+1}}{(1-\gamma)^2}.$$

[5] Note that one choice of such policies is a deterministic policy π' satisfying

$$\pi'(s) \in \arg\max_{a \in \mathcal{A}} \bar{r}(s, a) + \gamma \textstyle\sum_{s' \in \mathcal{S}} P(s'|s, a) V_K(s').$$

Note that $\|V^{\pi_K} - V^*\|_\infty = \max_{s\in\mathcal{S}} \left[V^*(s) - V^{\pi_K}(s)\right]$, thus, for sufficiently large K, π_K is near-optimal.

2.2.2.2 Policy Iteration

Policy iteration is one algorithm that computes an optimal policy π^*, which is described as follows:

Policy iteration algorithm

```
Initialization: choose an arbitrary initial deterministic policy $\pi_0$
Policy Iteration: for each $k = 0, 1, 2, \ldots$
    step 1: (policy evaluation) compute the state value function $V^{\pi_k}$ for policy
    $\pi_k$ by solving the system of linear equations
        $V = \mathbb{T}_{\pi_k} V.$
    step 2: (policy improvement) compute a deterministic policy $\pi_{k+1}$ satisfying
        $\mathbb{T}_{\pi_{k+1}} V^{\pi_k} = \mathbb{T} V^{\pi_k}$
    step 3: if $V^{\pi_k} = \mathbb{T} V^{\pi_k}$, terminate and return $\pi_k$
```

Recall that V^{π_k} is the unique bounded solution for the Bellman equation $V = \mathbb{T}_{\pi_k} V$. We also note that by definition, this Bellman equation is a system of linear equations with $|\mathcal{S}|$ variables and $|\mathcal{S}|$ equations, where $|\mathcal{S}|$ is the cardinality of the state space $\mathcal{S}$. Thus, in the policy evaluation step, V^{π_k} can be computed by solving linear equations. For the policy improvement step, we can choose a deterministic policy π_{k+1} satisfying

$$\pi_{k+1}(s) \in \arg\max_{a\in\mathcal{A}} \left[\bar{r}(s,a) + \gamma \sum_{s'\in\mathcal{S}} P(s'|s,a) V^{\pi_k}(s')\right], \quad \forall s \in \mathcal{S}.$$

Notice that the policy iteration algorithm terminates if and only if $V^{\pi_k} = V^*$, that is, if and only if π_k is optimal.

One can prove that the policy iteration algorithm will find an optimal policy and terminate in a finite number of steps. Interested readers might refer to Proposition 2.3.1 in Bertsekas (2011) for the proof. This is the main advantage of policy iteration over value iteration. On the other hand, the policy evaluation step in policy iteration requires solving a system of linear equations. This step can be computationally expensive if the number of states $|\mathcal{S}|$ is large.

2.3 Reinforcement Learning Algorithm Design

Based on the Markov decision process (MDP) frameworks discussed in Sect. 2.2, in this section, we provide a high-level review of some core algorithm design issues for reinforcement learning (RL), such as the choice of learning target, how to design exploration schemes, and approximate solutions for large-scale RL problems. Specifically, this section proceeds as follows: first, we formulate two standard RL problems in Sect. 2.3.1 based on the finite-horizon MDP and the discounted MDP discussed in the previous section. Then, in Sect. 2.3.2, we discuss the differences between model-based RL and model-free RL, which correspond to different choices of learning targets. We also review some classical RL algorithms, such as Q-learning, Sarsa, and REINFORCE, in Sect. 2.3.2. Third, in Sect. 2.3.3, we review some commonly used exploration schemes and discuss why data efficient RL agents need to be able to accomplish "deep exploration". Finally, in Sect. 2.3.4, we briefly review approximate learning algorithms for large-scale RL problems, such as some state-of-the-art deep reinforcement learning (DRL) algorithms (Silver et al., 2016, 2017b).

It is worth mentioning that RL has been an active research field in the past few decades, and many different problem formulations and algorithms have been developed. Due to the space limit, we can only discuss some core algorithm design issues mentioned above and review a few classical algorithms. Interested readers might refer to Sect. 2.4 for pointers to further reading.

2.3.1 Reinforcement Learning Problem Formulation

In this subsection, we formulate two RL problems based on the MDPs discussed in Sect. 2.2: episodic RL in a finite-horizon MDP, and RL in a discounted MDP.

2.3.1.1 Episodic Reinforcement Learning in Finite-Horizon MDP

The first RL problem we consider is an episodic RL problem in a finite-horizon MDP described in Sect. 2.2.1. Recall that a finite-horizon MDP $\mathcal{M}_\mathrm{F}$ is characterized by a tuple $\mathcal{M}_\mathrm{F} = (\mathcal{S}, \mathcal{A}, P, r, H, \rho)$. In this episodic RL problem, we assume that the agent knows the state space $\mathcal{S}$, the action space $\mathcal{A}$, and the time horizon H; but does not fully know the initial state distribution ρ, the transition model P, or the reward model r. We also assume that the agent will repeatedly interact with $\mathcal{M}_\mathrm{F}$ for T episodes. For any episode $t = 1, \ldots, T$, and any period $h = 1, \ldots, H$, we use s_{th}, a_{th}, and r_{th} to, respectively, denote the state, action, and reward at period h in episode t.

Each episode $t = 1, 2, \ldots, T$ proceeds as follows: at the beginning of this episode, the agent first observes an initial state s_{t1}, which is independently drawn from the initial state distribution ρ. Then, at each period $h = 1, \ldots, H$, the agent adaptively chooses an action $a_{th} \in \mathcal{A}$ based on its prior knowledge and past observations and observes and receives a reward $r_{th} \sim r(\cdot | s_{th}, a_{th})$. If $h < H$, the agent will also observe the next state $s_{t,h+1} \sim P(\cdot | s_{th}, a_{th})$. Episode t terminates once the agent receives the reward r_{tH} at period H. The agent's goal is to maximize its expected cumulative reward in the first T episodes:

$$\max \mathbb{E}\left[\sum_{t=1}^{T} \sum_{h=1}^{H} r_{th}\right].$$

Many canonical or real-world RL problems can be formulated as either special cases or extensions of the episodic RL problems described above. For example, the classical multi-armed bandit problem (Lattimore & Szepesvári, 2020) can be formulated as an episodic RL problem with one state and time horizon $H = 1$. On the other hand, agents aiming to learn good strategies in computer games usually need to interact with the games repeatedly, and each interaction can be viewed as an episode. The computer game setting can be viewed as an extension of the episodic RL problem described above, and the main difference is that the time horizon H in computer games are usually random.[6] Many research works have been dedicated to episodic RL problems in the past decade (Dann et al. 2017; Wen & Van Roy, 2017; Osband et al. 2013, 2019).

2.3.1.2 Reinforcement Learning in Discounted MDP

The second RL problem we consider is a RL problem in a discounted MDP $\mathcal{M}_{\mathrm{D}}$, which has been described in Sect. 2.2.2. Recall that a discounted MDP $\mathcal{M}_{\mathrm{D}}$ is characterized by a tuple $\mathcal{M}_{\mathrm{D}} = (\mathcal{S}, \mathcal{A}, P, r, \gamma, \rho)$. In this RL problem, we assume that the agent knows the state space $\mathcal{S}$, the action space $\mathcal{A}$, and the discrete-time discount factor γ; but does not fully know the initial state distribution ρ, the transition model P, or the reward model r. For each time step $t = 1, 2, \ldots$, we use s_t, a_t, and r_t to, respectively, denote the state, action, and reward at time period t.

This RL problem proceeds as follows: at the first time period $t = 1$, the agent observes an initial state s_1, which is independently drawn from the initial state distribution ρ. Then, at each time step $t = 1, 2, 3, \ldots$, the agent first adaptively chooses an action $a_t \in \mathcal{A}$ based on its prior knowledge and past observations, and then observes the reward $r_t \sim r(\cdot | s_t, a_t)$ and the next state $s_{t+1} \sim P(\cdot | s_t, a_t)$. The agent's goal is to maximize its expected total discounted reward

$$\mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t\right].$$

[6] More precisely, the time horizon H in a computer game is usually a *stopping time*, rather than deterministic.

In other words, the RL problem described in this subsection is the same as the dynamic optimization problem discussed in Sect. 2.2.2, except that the agent does not fully know P, r, and ρ. Consequently, the agent cannot directly compute an optimal or near-optimal policy via the value iteration algorithm or the policy iteration algorithm described in Sect. 2.2.2. Instead, the agent needs to learn to take optimal or near-optimal actions while interacting with $\mathcal{M}_{\mathrm{D}}$. It is worth mentioning that RL in discounted MDPs is one of the most classical RL problems, and many classical RL algorithms, such as Q-learning (Watkins & Dayan, 1992), were first developed under this problem formulation.

2.3.2 Model-Based vs. Model-Free Reinforcement Learning

As we have discussed above, in RL problems, the agent usually does not fully know the environment. For instance, in the RL problems described in Sect. 2.3.1, the agent does not know the reward model r and the transition model P. The agent may observe the reward and possibly other observations (e.g., the next state) after taking an action at each time period. The agent needs to learn an optimal or near-optimal or even high-performance policy $\pi^\dagger$ while interacting with the environment.

Note that the agent does not have to attempt to learn $\pi^\dagger$ *directly*. Instead, it can choose to learn a *learning target* χ (Lu et al., 2021) that contains sufficient information[7] to compute $\pi^\dagger$. We can classify the RL algorithms based on their chosen learning target χ. For the RL problems described in Sect. 2.3.1, some commonly chosen learning targets are:

1. the MDP model;
2. the optimal state-action value function Q^*;
3. an optimal policy π^*, or a near-optimal policy, or just a high-performance policy.

If an algorithm chooses the MDP model as its learning target, then we refer to that algorithm as a *model-based* RL algorithm. On the other hand, if an algorithm chooses Q^*, π^*, or a near-optimal policy as its learning target, then we refer to that algorithm as a *model-free* RL algorithm, since it tries to learn an optimal or near-optimal policy without learning the full MDP model. Specifically, if the learning target of an algorithm is the optimal value function Q^*, then that algorithm is referred to as a *value learning* algorithm. On the other hand, if the learning target is an optimal policy, a near-optimal policy, or just a high-performance policy, then the algorithm is referred to as a *policy learning* algorithm. As we will discuss below, there are pros and cons between model-based RL algorithms and model-free RL algorithms.

[7] Mathematically, it means that $\pi^\dagger = \psi(\chi)$, where ψ is a function known to the agent.

2.3.2.1 Model-Based Reinforcement Learning

A model-based RL algorithm chooses the MDP model as its learning target. To simplify the exposition, let us consider the episodic RL problem described in Sect. 2.3.1.1, and the model-based RL in discounted MDPs is similar. For the episodic RL problem, a model-based RL algorithm maintains a "knowledge state" about the MDP model $\mathcal{M}_\mathrm{F}$, and updates it while interacting with the environment. Depending on the algorithm, this knowledge state could be a point estimate of $\mathcal{M}_\mathrm{F}$, a confidence set of $\mathcal{M}_\mathrm{F}$, or the posterior distribution over $\mathcal{M}_\mathrm{F}$. In each episode, a model-based RL algorithm chooses actions based on its knowledge state about $\mathcal{M}_\mathrm{F}$.

One example of model-based RL algorithms is the posterior sampling for reinforcement learning (PSRL) developed in Osband et al. (2013), which can be viewed as a special case of Thompson sampling (Thompson, 1933; Russo et al., 2017) and is described below.

Posterior sampling for reinforcement learning (PSRL)

```
**Initialization:** a prior distribution $\mathbb{P}_0$ over the environment $\mathcal{M}_\mathrm{F}$
**for** each episode $t = 1, 2, \ldots$
    **Step 1:** sample a finite-horizon MDP model $\tilde{\mathcal{M}}_t \sim \mathbb{P}_{t-1}$
    **Step 2:** compute $\pi_t$, one optimal policy under $\tilde{\mathcal{M}}_t$
    **Step 3:** apply $\pi_t$ in episode $t$, receive reward $r_{t1}, r_{t2}, \ldots, r_{tH}$, and observe the
    state-action-reward trajectory $\mathcal{D}_t = (s_{t1}, a_{t1}, r_{t1}, \ldots, s_{tH}, a_{tH}, r_{tH})$
    **Step 4:** update the posterior $\mathbb{P}_t$ over $\mathcal{M}_\mathrm{F}$ using Bayes' rule, based on $\mathbb{P}_{t-1}$ and
    observation $\mathcal{D}_t$
```

Specifically, the PSRL algorithm maintains and updates a posterior distribution $\mathbb{P}_t$ over the environment $\mathcal{M}_\mathrm{F}$. At each episode t, it first samples an MDP model $\tilde{\mathcal{M}}_t$ from the posterior, then it computes a policy π_t that is optimal under the sampled model $\tilde{\mathcal{M}}_t$. Third, it applies the policy π_t in the true environment $\mathcal{M}_\mathrm{F}$ and observes the state-action-reward trajectory $\mathcal{D}_t$. Finally, it updates the posterior distribution over the environment $\mathcal{M}_\mathrm{F}$ based on $\mathcal{D}_t$, using the Bayes' rule.

Compared with the model-free RL algorithms, one major disadvantage of model-based RL algorithms, including PSRL described above, is that they tend to be computationally expensive for large-scale RL problems. Specifically, a model-based RL algorithm aims to learn the MDP model of the environment and needs to maintain and update a knowledge state about the MDP model. Thus, to decide how to choose actions, a model-based RL algorithm usually needs to compute a policy based on its knowledge state about the MDP model. This step often requires solving a dynamic programming problem. If the MDP model (environment) is large-scale, then this step is usually computationally expensive.

Let us use the PSRL algorithm described above to further illustrate this. In PSRL, the knowledge state about the MDP model is the posterior distribution $\mathbb{P}_{t-1}$ over the MDP model. To choose actions in episode t, PSRL first samples a model $\tilde{\mathcal{M}}_t \sim \mathbb{P}_{t-1}$, and then computes a policy π_t that is optimal under the sampled model

$\tilde{\mathcal{M}}_t$. Note that computing π_t based on $\tilde{\mathcal{M}}_t$ requires solving a dynamic programming problem in the finite-horizon MDP $\tilde{\mathcal{M}}_t$. If the environment $\mathcal{M}_{\mathrm{F}}$ is a large-scale problem and PSRL starts with an appropriately chosen prior, then $\tilde{\mathcal{M}}_t$ is also likely to be a large-scale MDP and hence computing π_t can be computationally expensive.

On the other hand, for many RL problems, especially the large-scale RL problems that require approximate solutions (see Sect. 2.3.4), it is usually easier to develop provably data efficient model-based algorithms than provably data efficient model-free algorithms. In particular, the PSRL algorithm described above is data efficient under appropriate technical conditions (Osband et al., 2013), and we will discuss this more in Sect. 2.3.3.

2.3.2.2 Q-Learning and SARSA

A widely used model-free RL algorithm is the classical Q-learning algorithm (Watkins & Dayan, 1992). As its name indicates, the Q-learning algorithm chooses the optimal state-action value function Q^* as its learning target, and hence it is a value learning algorithm. To simplify the exposition, let us consider a version of Q-learning algorithm for the episodic RL problem described in Sect. 2.3.1.1, which is detailed below.

Q-learning with ϵ-greedy exploration

```
Initialization: learning step size $\alpha \in (0, 1]$, exploration probability $\epsilon \in (0, 1]$,
    and initialize $Q(s, h, a)$ arbitrarily for all $(s, h, a) \in \mathcal{S} \times \mathcal{H} \times \mathcal{A}$
for each episode $t = 1, 2, \ldots$
    observe the initial state $s_{t1} \sim \rho$
    for each period $h = 1, \ldots, H$:
        Step 1 ($\epsilon$-greedy exploration): with probability $\epsilon$, choose action
        $a_{th}$ uniformly randomly from $\mathcal{A}$; with probability $1 - \epsilon$, choose
```

$$a_{th} \sim \text{unif}\left(\arg\max_{a \in \mathcal{A}} Q\left(s_{th}, h, a\right)\right)$$

```
        that is, $a_{th}$ is sampled uniformly randomly from $\arg\max_{a \in \mathcal{A}} Q\left(s_{th}, h, a\right)$
        Step 2: take action $a_{th}$, observe reward $r_{th}$; if $h < H$, also observe the next
        state $s_{t,h+1}$
        Step 3: compute the temporal difference (TD) error
```

$$\delta_{th} = \begin{cases} r_{th} + \max_{a'} Q(s_{t,h+1}, h+1, a') - Q\left(s_{th}, h, a_{th}\right) & \text{if } h < H \\ r_{th} - Q\left(s_{th}, h, a_{th}\right) & \text{if } h = H \end{cases} \tag{2.22}$$

```
        Step 4: update $Q\left(s_{th}, h, a_{th}\right)$ as
```

$$Q\left(s_{th}, h, a_{th}\right) \leftarrow Q\left(s_{th}, h, a_{th}\right) + \alpha \delta_{th}$$

Roughly speaking, the above Q-learning algorithm maintains and updates an estimate Q of the optimal state-action value function Q^* and proceeds as follows: at each period h in episode t, the agent first chooses an action a_{th} based on the ϵ-*greedy* exploration with current estimate Q. That is, with probability ϵ, it chooses the action a_{th} uniformly randomly from $\mathcal{A}$; and with probability $1-\epsilon$, it chooses a_{th} *greedy*[8] to the current estimate Q in the sense that $a_{th} \in \arg\max_{a \in \mathcal{A}} Q(s_{th}, h, a)$. Then, it takes action a_{th}, observes the reward r_{th}, and also observes the next state $s_{t,h+1}$ if $h < H$. Finally, the agent computes the *temporal-difference (TD) error* δ_{th} as specified in Eq. (2.22) and uses the TD error to update the value estimate $Q(s_{th}, h, a_{th})$.

Note that the Q-learning algorithm above is a temporal-difference (TD) learning algorithm, since it updates its estimate Q based on the TD error δ_{th} specified in Eq. (2.22). To see why δ_{th} is referred to as a TD error, let us consider a period $h < H$ in episode t. Recall that $r_{th} \sim r(\cdot|s_{th}, a_{th})$ and $s_{t,h+1} \sim P(\cdot|s_{th}, a_{th})$, thus, conditioning on s_{th} and a_{th}, $r_{th} + \max_{a'} Q(s_{t,h+1}, h+1, a')$ is an unbiased estimate of

$$\bar{r}(s_{th}, a_{th}) + \sum_{s' \in \mathcal{S}} P(s'|s_{th}, a_{th}) \max_{a' \in \mathcal{A}} Q(s', h+1, a'), \tag{2.23}$$

and hence δ_{th} is an unbiased estimate of

$$\bar{r}(s_{th}, a_{th}) + \sum_{s' \in \mathcal{S}} P(s'|s_{th}, a_{th}) \max_{a' \in \mathcal{A}} Q(s', h+1, a') - Q(s_{th}, h, a_{th}). \tag{2.24}$$

If we view Q as an estimate of Q^*, then $Q(s_{th}, h, a_{th})$ is an estimate of $Q^*(s_{th}, h, a_{th})$. On the other hand, based on Eq. (2.23), $r_{th} + \max_{a'} Q(s_{t,h+1}, h+1, a')$ is an estimate of

$$\bar{r}(s_{th}, a_{th}) + \sum_{s' \in \mathcal{S}} P(s'|s_{th}, a_{th}) \max_{a' \in \mathcal{A}} Q^*(s', h+1, a') = Q^*(s_{th}, h, a_{th}),$$

where the equality follows from the Bellman equation. Thus, δ_{th} is the difference between two estimates of $Q^*(s_{th}, h, a_{th})$: $Q(s_{th}, h, a_{th})$ and $r_{th} + \max_{a'} Q(s_{t,h+1}, h+1, a')$. Since $r_{th} + \max_{a'} Q(s_{t,h+1}, h+1, a')$ is based on Q in the next period (period $h+1$), while $Q(s_{th}, h, a_{th})$ is based on Q in the current period (period h), this difference is referred to as a temporal-difference (TD) error.

Let us briefly discuss why the Q-learning algorithm might be able to learn the optimal state-action value function Q^*. Based on the value update equation

$$Q(s_{th}, h, a_{th}) \leftarrow Q(s_{th}, h, a_{th}) + \alpha \delta_{th},$$

[8] The algorithm breaks ties in a uniformly random manner, as specified in the pseudo-code.

with an appropriately chosen learning step size α, the Q-learning algorithm updates Q to minimize the absolute value (or square, which is equivalent) of the TD error δ_{th}. As we have discussed above, the TD error δ_{th} is an unbiased estimate of Eq. (2.24); and the absolute value of Eq. (2.24) is minimized when $Q = Q^*$. Thus, under appropriate conditions, the Q-learning algorithm can learn Q^*. Rigorously speaking, one can prove that if all state-period-action triples are visited infinitely often, with a different choice of the learning step sizes that are episode-varying and satisfy some standard *stochastic approximation* (Kushner & Yin, 2003) conditions, Q will converge to Q^* with probability 1. Please refer to Jaakkola et al. (1994) and Tsitsiklis (1994) for the analysis.

The Q-learning algorithm is an *off-policy* learning algorithm, since it aims to learn a policy different from that used to generate data. The policy used to generate data is also known as the *behavior policy*. Specifically, the Q-learning algorithm aims to learn the optimal state-action value function Q^*, or equivalently, the optimal policy π^*. However, the behavior policy can be any policy that performs sufficient exploration to ensure that all state-period-action triples are visited infinitely often. In the algorithm above, the policy used to generate data is the ϵ-greedy policy with respect to the current estimate Q. It can also be other policies, such as the Boltzmann (softmax) exploration policy with respect to the current estimate Q (see Sect. 2.3.3, and Cesa-Bianchi et al. (2017) and the references therein).

The following learning algorithm, which is referred to as Sarsa (Rummery & Niranjan, 1994; Sutton 1996), is an *on-policy* variant of the Q-learning algorithm. We say Sarsa is on-policy since it attempts to evaluate and improve the policy that is used to make decisions (i.e., the behavior policy). The main difference between Sarsa and Q-learning is the TD error for period $h < H$: in Sarsa, the TD error is defined based on the state-action-reward-state-action quintuple[9] $\left(s_{th}, a_{th}, r_{th}, s_{t,h+1}, a_{t,h+1}\right)$:

$$\delta_{th} = r_{th} + Q(s_{t,h+1}, h+1, a_{t,h+1}) - Q\left(s_{th}, h, a_{th}\right).$$

Assume that the current behavior policy is π, and assume that $a_{t,h+1}$ is chosen under π, i.e., $a_{t,h+1} \sim \pi(\cdot|s_{t,h+1}, h+1)$. Similar to what we have discussed above, for Sarsa, δ_{th} is an unbiased estimate of

$$\bar{r}(s_{th}, a_{th}) + \sum_{s' \in \mathcal{S}} P(s'|s_{th}, a_{th}) \sum_{a' \in \mathcal{A}} \pi(a'|s', h+1) Q(s', h+1, a') - Q(s_{th}, h, a_{th}),$$

whose absolute value is minimized by $Q = Q^\pi$. Consequently, Sarsa continually aims to estimate Q^π for the current behavior policy π. Note that at the same time Sarsa also updates π toward greediness with respect to Q^π, as detailed below. Interested readers might refer to Singh et al. (2000) for the convergence analysis of Sarsa.

[9] This state-action-reward-state-action quintuple gives rise to the name Sarsa for the algorithm.

Sarsa with ϵ-greedy exploration

```
**Initialization:** learning step size $\alpha \in (0, 1]$, exploration probability $\epsilon \in (0, 1]$,
  and initialize $Q(s, h, a)$ arbitrarily for all $(s, h, a) \in \mathcal{S} \times \mathcal{H} \times \mathcal{A}$
**for** each episode $t = 1, 2, \ldots$
  observe the initial state $s_{t1} \sim \rho$
  choose action $a_{t1}$ using $\epsilon$-greedy policy with respect to $Q$
  **for** each period $h = 1, \ldots, H$:
    **Step 1:** take action $a_{th}$, observe reward $r_{th}$; if $h < H$, also observe the next
    state $s_{t,h+1}$, and choose action $a_{t,h+1}$ using $\epsilon$-greedy policy with respect
    to $Q$
    **Step 2:** compute the temporal difference (TD) error
```

$$\delta_{th} = \begin{cases} r_{th} + Q(s_{t,h+1}, h+1, a_{t,h+1}) - Q(s_{th}, h, a_{th}) & \text{if } h < H \\ r_{th} - Q(s_{th}, h, a_{th}) & \text{if } h = H \end{cases} \tag{2.25}$$

```
    **Step 3:** update $Q(s_{th}, h, a_{th})$ as
```

$$Q(s_{th}, h, a_{th}) \leftarrow Q(s_{th}, h, a_{th}) + \alpha \delta_{th}$$

Finally, it is worth mentioning that there are many variants and extensions of the Q-learning algorithm and the Sarsa algorithm described above, such as the expected Sarsa algorithm (Van Seijen et al., 2009), the double Q-learning algorithm (Hasselt, 2010), the n-step TD algorithms (see van Seijen (2016) and Chap. 7 in Sutton and Barto (2018)) and the TD(λ) algorithms (see Sutton (1988), Dayan (1992), Tsitsiklis (1994), and Chap. 12 in Sutton and Barto (2018)). Interested readers might refer to these references for further reading. Also, this subsection has focused on the episodic RL problem; it is straightforward to develop similar Q-learning and Sarsa algorithms for RL in discounted MDPs described in Sect. 2.3.1.2.

2.3.2.3 Policy Gradient

Another class of widely used model-free RL algorithms are the policy gradient methods (see Williams (1992), Marbach and Tsitsiklis (2001), Sutton et al. (2000), and Chap. 13 in Sutton and Barto (2018)). As the name "policy gradient" indicates, these methods choose an optimal policy π^* as their learning target and aim to learn a good approximation of π^* with a parametric model, and hence they are policy learning algorithms. To simplify the exposition, let us motivate and consider a version of policy gradient method for the episodic RL problem described in Sect. 2.3.1.1; a similar policy gradient method can be derived for RL in discounted MDPs described in Sect. 2.3.1.2.

Consider a policy π_θ parameterized by $\theta \in \Re^d$, where d is the dimension of θ. Note that the policy π_θ can be parameterized in any way, as long as $\pi_\theta(a|s,h)$ is differentiable with respect to θ for all (s,h,a). One common kind of parameterization is to parameterize the *preference* $\phi_\theta(s,h,a) \in \Re$ for all state-period-action triple (s,h,a), and define π_θ via the softmax function:

$$\pi_\theta(a|s,h) = \frac{\exp\left(\phi_\theta(s,h,a)\right)}{\sum_{a' \in \mathcal{A}} \exp\left(\phi_\theta(s,h,a')\right)}.$$

For each $\theta \in \Re^d$, we define the expected total reward under policy π_θ as

$$J(\theta) = \mathbb{E}\left[V^{\pi_\theta}(s_1, 1)\right], \tag{2.26}$$

where the expectation is over the initial state[10] s_1, which is drawn from the initial state distribution ρ. Hence, the problem of finding the best policy in the policy class $\Pi = \left\{\pi_\theta : \theta \in \Re^d\right\}$ can be formulated as $\max_{\theta \in \Re^d} J(\theta)$. Of course, one natural method to maximize $J(\theta)$ is the gradient ascent algorithm based on $\nabla_\theta J(\theta)$.

The following theorem is known as the *policy gradient theorem*, which is the mathematical foundation for all policy gradient methods.

Theorem 2.1 (Policy Gradient Theorem) *For $J(\theta)$ defined in Eq. 2.26, we have*

$$\nabla_\theta J(\theta) = \sum_{h=1}^{H} \mathbb{E}_{\pi_\theta}\left[Q^{\pi_\theta}(s_h, h, a_h) \nabla_\theta \log \pi_\theta(a_h|s_h, h)\right],$$

where the subscript π_θ in notation $\mathbb{E}_{\pi_\theta}$ indicates that the expectation is taken under the stochastic process defined by policy π_θ.

Proof Note that $V^{\pi_\theta}(s_h, h) = \sum_{a \in \mathcal{A}} \pi_\theta(a|s_h, h) Q^{\pi_\theta}(s_h, h, a)$, thus

$$\nabla_\theta V^{\pi_\theta}(s_h, h) = \sum_{a \in \mathcal{A}} \left[Q^{\pi_\theta}(s_h, h, a) \nabla_\theta \pi_\theta(a|s_h, h) + \pi_\theta(a|s_h, h) \nabla_\theta Q^{\pi_\theta}(s_h, h, a)\right].$$

From the Bellman equation (2.4), we have $\nabla_\theta Q^{\pi_\theta}(s_h, h, a) = 0$ if $h = H$ and

$$\nabla_\theta Q^{\pi_\theta}(s_h, h, a) = \sum_{s' \in \mathcal{S}} P(s'|s_h, a) \nabla_\theta V^{\pi_\theta}(s', h+1) \quad \text{if } h < H.$$

Since

[10] In Sect. 2.3.2.3, to simplify the notation, we drop the episode subscript t if the discussion/analysis is within one episode.

$$\sum_{a\in\mathcal{A}} \pi_\theta(a|s_h,h) \sum_{s'\in\mathcal{S}} P(s'|s_h,a)\nabla_\theta V^{\pi_\theta}(s',h+1) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta V^{\pi_\theta}(s_{h+1},h+1)\middle|s_h\right]$$

and

$$\begin{aligned}&\sum_{a\in\mathcal{A}} Q^{\pi_\theta}(s_h,h,a)\nabla_\theta \pi_\theta(a|s_h,h)\\ &= \sum_{a\in\mathcal{A}} Q^{\pi_\theta}(s_h,h,a)\pi_\theta(a|s_h,h)\nabla_\theta \log \pi_\theta(a|s_h,h)\\ &= \mathbb{E}_{\pi_\theta}\left[Q^{\pi_\theta}(s_h,h,a_h)\nabla_\theta \log\pi_\theta(a_h|s_h,h)\middle|s_h\right],\end{aligned}$$

we have

$$\begin{aligned}\nabla_\theta V^{\pi_\theta}(s_h,h) =& \mathbb{E}_{\pi_\theta}\left[Q^{\pi_\theta}(s_h,h,a_h)\nabla_\theta \log\pi_\theta(a_h|s_h,h)\middle|s_h\right]\\ &+ \mathbb{E}_{\pi_\theta}\left[\nabla_\theta V^{\pi_\theta}(s_{h+1},h+1)\middle|s_h\right]\mathbf{1}(h<H).\end{aligned}$$

Taking the expectation over s_h, we have

$$\begin{aligned}\mathbb{E}_{\pi_\theta}\left[\nabla_\theta V^{\pi_\theta}(s_h,h)\right] =& \mathbb{E}_{\pi_\theta}\left[Q^{\pi_\theta}(s_h,h,a_h)\nabla_\theta \log\pi_\theta(a_h|s_h,h)\right]\\ &+ \mathbb{E}_{\pi_\theta}\left[\nabla_\theta V^{\pi_\theta}(s_{h+1},h+1)\right]\mathbf{1}(h<H).\end{aligned}$$

Hence we have

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta V^{\pi_\theta}(s_1,1)\right] = \sum_{h=1}^{H} \mathbb{E}_{\pi_\theta}\left[Q^{\pi_\theta}(s_h,h,a_h)\nabla_\theta \log\pi_\theta(a_h|s_h,h)\right].$$

This concludes the proof. □

We now motivate and discuss one policy gradient method, referred to as REINFORCE (Williams, 1992), based on Theorem 2.1. First, note that we can compute a stochastic gradient of $J(\theta)$ based on a state-action-reward trajectory $s_1, a_1, r_1, \ldots, s_H, a_H, r_H$ under policy π_θ. To see it, let us define $G_h = \sum_{h'=h}^{H} r_{h'}$ for any h, which is the total reward from period h to period H. We claim that $\sum_{h=1}^{H} G_h \nabla_\theta \log \pi_\theta(a_h|s_h,h)$ is a stochastic gradient of $J(\theta)$. To see it, notice that

$$\begin{aligned}\mathbb{E}_{\pi_\theta}\left[G_h\nabla_\theta \log\pi_\theta(a_h|s_h,h)\right] &= \mathbb{E}_{\pi_\theta}\left[\mathbb{E}_{\pi_\theta}\left[G_h|s_h,a_h\right]\nabla_\theta \log\pi_\theta(a_h|s_h,h)\right]\\ &= \mathbb{E}_{\pi_\theta}\left[Q^{\pi_\theta}(s_h,h,a_h)\nabla_\theta \log\pi_\theta(a_h|s_h,h)\right],\end{aligned}$$

where the second equality follows from $Q^{\pi_\theta}(s_h,h,a_h) = \mathbb{E}_{\pi_\theta}\left[G_h|s_h,a_h\right]$. The REINFORCE algorithm is described below. As we have discussed above, it is a stochastic gradient ascent algorithm to maximize $J(\theta)$.

REINFORCE

```
Initialization: differentiable policy parameterization $\pi_\theta$, initial $\theta$
    and learning step size $\alpha \in (0, 1]$
for each episode $t = 1, 2, \ldots$
    Step 1: generate trajectory $s_{t1}, a_{t1}, r_{t1}, \ldots s_{tH}, a_{tH}, r_{tH}$ under policy $\pi_\theta$
    Step 2: compute $G_{th} = \sum_{h'=h}^{H} r_{th'}$ for all $h = 1, 2, \ldots, H$
    Step 3: update $\theta \leftarrow \theta + \alpha \sum_{h=1}^{H} G_{th} \nabla_\theta \log \pi_\theta(a_{th}|s_{th}, h)$
```

It is worth mentioning that there are other policy gradient methods in addition to the REINFORCE algorithm presented above. Such methods include REINFORCE with baseline (Williams, 1992; Greensmith et al., 2004) and actor-critic methods (Sutton, 1984; Degris et al., 2012). Interested readers might refer to the references for further reading.

2.3.3 Exploration in Reinforcement Learning

In this subsection, we briefly review exploration in RL. As we have discussed above, the exploration-exploitation trade-off is a key challenge in RL. Specifically, balancing this trade-off is crucial for a RL algorithm to be data efficient, i.e., to learn an optimal or near-optimal policy within few interactions with the environment. Specifically, if an agent does not explore enough (under-exploration), then it might get stuck in sub-optimal policies and never learn an optimal or near-optimal policy; on the other hand, if an agent explores too much (over-exploration), then it might choose sub-optimal actions in too many time steps and hence incur a huge reward loss.

This subsection is organized as follows: we briefly review some commonly used exploration schemes in Sect. 2.3.3.1; in Sect. 2.3.3.2, we motivate and discuss why data efficient RL algorithms need to be able to accomplish "deep exploration".

2.3.3.1 Exploration Schemes

We now briefly review some commonly used exploration schemes, including ϵ-greedy exploration, Boltzmann exploration, exploration based on *optimism in the face of uncertainty (OFU)*, and Thompson sampling. To simplify the exposition, we discuss these exploration schemes under the episodic RL problem discussed in Sect. 2.3.1.1.

ϵ-Greedy Exploration ϵ-greedy exploration is probably the simplest exploration scheme. In Sect. 2.3.2.2, we have presented two algorithms with ϵ-greedy exploration: Q-learning with ϵ-greedy exploration and Sarsa with ϵ-greedy exploration.

Roughly speaking, in value learning algorithms, ϵ-greedy exploration proceeds as follows: assume that Q is a point estimate of the optimal state-action value function Q^*, then at each period h in episode t, with probability $1 - \epsilon$, the agent chooses an action greedy to the current estimate Q, i.e., $a_{th} \in \arg\max_{a \in \mathcal{A}} Q(s_{th}, h, a)$ (exploitation); and with probability ϵ, it chooses a random action (exploration). Similarly, in a model-based RL algorithm that maintains and updates a point estimate of the MDP model, at each time step, the ϵ-greedy exploration chooses an action greedy to the current model estimate with probability $1 - \epsilon$ and chooses a random action with probability ϵ. Note that the choice of ϵ trades off the exploration and exploitation.

Boltzmann (softmax) Exploration Boltzmann (softmax) exploration (Cesa-Bianchi et al., 2017) is similar to ϵ-greedy exploration. In value learning algorithms, Boltzmann exploration proceeds as follows: assume that Q is a point estimate of Q^*, then at each period h in episode t, the agent chooses action $a \in \mathcal{A}$ with probability

$$\pi^{\mathrm{B}}(a|s_{th}, h) = \frac{\exp\left(Q(s_{th}, h, a)/\eta\right)}{\sum_{a' \in \mathcal{A}} \exp\left(Q(s_{th}, h, a')/\eta\right)}, \tag{2.27}$$

where $\eta > 0$ is the *temperature* of Boltzmann exploration and trades off exploration and exploitation. Specifically, as $\eta \to \infty$, $\pi^{\mathrm{B}}(\cdot|s_{th}, h)$ converges to the uniform distribution over $\mathcal{A}$ (exploration only); as $\eta \to 0$, Boltzmann exploration will choose an action greedy to Q (exploitation only).

Optimism in the Face of Uncertainty (OFU) OFU is a class of exploration schemes that are widely used to design provably data efficient RL algorithms. One version of the OFU exploration scheme proceeds as follows: the agent maintains and updates a *confidence set* over a learning target χ (e.g., the MDP model or Q^*); then at the beginning of each episode, it uses this confidence set to assign each state-period-action triple (s, h, a) an *optimistically biased* estimate $\hat{Q}(s, h, a)$ of $Q^*(s, h, a)$; finally, at each period h in the current episode t, it will choose action a_{th} greedy to $\hat{Q}$, i.e., $a_{th} \in \arg\max_{a \in \mathcal{A}} \hat{Q}(s_{th}, h, a)$.

Thompson Sampling (TS) Thompson sampling (Thompson, 1933; Russo et al., 2017) is another exploration scheme widely used to design data efficient RL algorithms. It proceeds as follows: the agent maintains and updates a posterior distribution over a learning target χ (e.g., the MDP model or Q^*); then at the beginning of each episode t, it samples a target $\tilde{\chi}_t$ from the posterior distribution and computes a policy π_t optimal under the sampled target $\tilde{\chi}_t$; finally, it chooses actions in episode t based on π_t. Note that the PSRL algorithm in Sect. 2.3.2.1 is a TS algorithm whose learning target is the MDP model.

In general, the ϵ-greedy exploration and the Boltzmann exploration are computationally more efficient than OFU and TS, since they only require a *point estimate* of the learning target (e.g., Q^*), while OFU requires maintaining and updating a *confidence set* over the learning target and TS requires maintaining and updating a *posterior distribution* over the learning target. On the other hand, ϵ-greedy and Boltzmann exploration can easily lead to data inefficient learning, while OFU and TS are widely used to design mathematically provably data efficient RL algorithms (Kearns & Singh, 2002; Brafman & Tennenholtz, 2002, Jaksch et al. 2010; Osband et al. 2013; Wen et al. 2020). In the next subsection, we will use a simple example to illustrate this.

There are other exploration schemes in addition to those mentioned above. One of them that is particularly interesting is the *information-directed sampling (IDS)* (Russo & Van Roy, 2014; Lu et al. 2021), which samples actions in a manner that minimizes the ratio between the squared expected performance loss (known as *regret*) and a measure of information gain. Interested readers might refer to the references for further reading.

2.3.3.2 Deep Exploration

In this subsection, we motivate and discuss why data efficient RL algorithms need to be able to accomplish "deep exploration" (Osband et al., 2019). As we have discussed above, in RL, exploration means that the agent needs to try actions that might provide some useful information feedback. In the special case of multi-armed bandits (MABs) (Lattimore & Szepesvári, 2020), since there is only one state, if the agent wants to gather some information by taking an action, it can always do it. However, this might not be the case for general RL problems. Specifically, some crucial information might only be obtained by trying an action at a particular state $s^\dagger$; consequently, to obtain this information, the agent needs to *learn to plan* to visit $s^\dagger$ first.

Consequently, a reliably data efficient RL algorithm needs to be able to accomplish "deep exploration". By this we mean that, the algorithm does not only consider *immediate* information gain of taking an action but also the consequences of an action or a sequence of actions on *future* learning. A deep exploration algorithm could, for instance, choose to incur performance losses over a sequence of actions while only expecting informative observations after multiple time steps. In the remainder of this section, we use a simple example to illustrate the notion of deep exploration and compare the data efficiencies of the PSRL algorithm described in Sect. 2.3.2.1 and the Q-learning with ϵ-greedy exploration described in Sect. 2.3.2.2.

Let us consider an episodic RL problem with deterministic transitions and rewards, which is illustrated in Fig. 2.2 and referred to as the "chain example". Specifically, in this problem, $\mathcal{S} = \{1, 2, \ldots, H\}$ where H is the time horizon, $\mathcal{A} = \{1, 2\}$, and the initial state in each episode is always $s_1 = 1$. When the agent takes action $a \in \mathcal{A}$ in state s at period h:

- it will receive a deterministic reward z if $s = H$ and $a = 1$; otherwise, it will receive reward 0.
- it will transition to state $\min\{s + 1, H\}$ if $a = 1$ and $h < H$; it will transition to state $\max\{s - 1, 1\}$ if $a = 2$ and $h < H$.

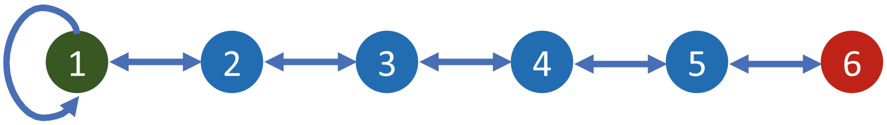

Fig. 2.2 Illustration of the "chain example" with $H = 6$. The nodes denote the states and the arrows denote the possible state transitions. We use the green node to denote the fixed initial state and use the red node to denote the "informative state"

We assume that the agent knows everything about this environment, except the deterministic reward z at state-action pair $(s = H, a = 1)$. We assume that the agent's prior over z is $\mathbb{P}_0(z = 1) = \mathbb{P}_0(z = -1) = 0.5$. Obviously, the optimal policy π^* depends on z. For example, if $z = 1$, the only optimal sequence of actions is to always choose $a = 1$. The agent needs to visit state $s = H$ and take action $a = 1$ there to learn the crucial information z. If the agent *plans* a sequence of actions to do so, we say it accomplishes the deep exploration in this example.

In this example, the data efficiency of an algorithm can be measured by the expected number of episodes it takes for the algorithm to learn z. Let us consider the PSRL algorithm first. Note that for this example, sampling an MDP model $\tilde{\mathcal{M}}_t$ is equivalent to sampling a deterministic reward $\tilde{z}_t \in \{-1, 1\}$ at state-action pair $(s = H, a = 1)$, since other parts of the environment are known. In episode $t = 1$ with prior $\mathbb{P}_0$, the agent will sample $\tilde{z}_1 = \pm 1$ with equal probability 0.5. Note that with $\tilde{z}_1 = 1$, the PSRL algorithm will choose a sequence of actions $a_{t1} = a_{t2} = \ldots = a_{tH} = 1$ in episode $t = 1$ and hence learn z; on the other hand, with $\tilde{z}_1 = -1$, the PSRL algorithm will not learn z in this episode. Thus, in episode 1, the PSRL algorithm will learn z with probability 0.5. Since the PSRL algorithm will not update its posterior before learning z, the expected number of episodes it takes for PSRL to learn z is 2.

On the other hand, for Q-learning with ϵ-greedy exploration, we assume that Q is initialized as $Q(s, h, a) = 0$ for all (s, h, a). Note that under this algorithm, before the agent observes z, $Q(s, h, a) = 0$, $\forall(s, h, a)$ and the algorithm chooses actions uniformly randomly at all state-period pairs. In such episodes, the agent will learn z with probability 2^{-H}. Hence, the expected number of episodes for this Q-learning algorithm to learn z is 2^H.

To sum up, in this example, Q-learning with ϵ-greedy exploration is highly data inefficient compared to PSRL. This is because PSRL accomplishes deep exploration: in each episode, it plans based on a sampled MDP model and hence considers the consequences of a sequence of actions. On the other hand, the Q-learning algorithm just chooses random actions before it observes the crucial information z.

2.3.4 Approximate Solution Methods and Deep Reinforcement Learning

Many modern RL problems tend to have intractably large state space $\mathcal{S}$ and/or action space $\mathcal{A}$. For such large-scale RL problems, an algorithm that aims to learn an optimal policy π^* *asymptotically* will require not only an intractably large memory space but also intractably many time steps for learning. Let us still use the episodic RL problem to illustrate the ideas. Consider the Q-learning algorithm with an exploration scheme that performs sufficient exploration (not necessarily the ϵ-greedy exploration). As we have discussed in Sect. 2.3.2.2, under appropriate conditions this algorithm learns Q^* asymptotically. Notice that this algorithm requires an $O(|\mathcal{S}||\mathcal{A}|H)$ memory space to store the point estimate Q of Q^*. Moreover, since the algorithm only updates its estimate $Q(s, h, a)$ for state-period-action triple (s, h, a) when it visits that triple, thus, to learn a good estimate of Q^*, the algorithm needs to visit each state-period-action triple at least once. This requires $\Omega(|\mathcal{S}||\mathcal{A}|)$ episodes, which is intractably many for large-scale problems.

Thus, for such large-scale RL problems, our goal is to learn a good approximate solution with limited memory space and limited time steps. One such approach, which is commonly used in practice, is to approximate the learning target (e.g., Q^* or π^*) by a low-dimensional parametric model and learn the parameters of that model. Note that if the parametric model can well approximate the learning target, and the number of parameters to learn is much less than the "size" of the learning target (e.g., the "size" of Q^* is $|\mathcal{S}||\mathcal{A}|H$), then learning with this parametric model can significantly improve the data efficiency.

One such learning algorithm is the REINFORCE algorithm described in Sect. 2.3.2.3. Recall that REINFORCE approximates its learning target π^* by a parametric model π_θ and tries to learn a good parameter vector θ via stochastic gradient ascent.

Similarly, many value learning algorithms for large-scale RL problems aim to learn a good approximation of Q^* via a parametric model Q_θ, where θ is the parameter vector to be learned. There are many difference choices of the parametric model Q_θ. One classical choice is to choose Q_θ linear in the parameter vector θ. Specifically, each state-period-action triple (s, h, a) is associated with a known feature vector $\phi(s, h, a) \in \Re^d$, and for any $\theta \in \Re^d$,

$$Q_\theta(s, h, a) = \phi(s, h, a)^T \theta, \tag{2.28}$$

where the superscript T denotes the vector transpose and d is the feature dimension. This parametric model is known as the *linear value function approximation* in the literature (see Chaps. 6 and 7 of Bertsekas (2011) and the references therein).

Another choice of the parametric model, which is widely used in the past decade, is to choose Q_θ as a (deep) neural network with fixed architecture and parameter vector θ. Note that the parameter vector θ typically encodes the weights and the biases in all layers of the neural network. Approximate solution methods based on a (deep) neural network (NN) model are also known as *deep reinforcement learning (DRL)* algorithms (Arulkumaran et al., 2017; Li, 2017). One well-known DRL algorithm is deep Q-learning with *experience replay* (Mnih et al., 2015), which is also known as deep Q-network (DQN) and is described below.

Deep Q-learning with experience replay (DQN)

```
**Initialization:** architecture of NN $Q_\theta$, initial $\theta$, exploration probability $\epsilon$,
  FIFO replay buffer $\mathcal{D}$ with capacity $N$, minibatch size $B$,
  and a gradient-based optimization algorithm optimizer

for each episode $t = 1, 2, \ldots$
  set $\theta^- \leftarrow \theta$
  observe the initial state $s_{t1} \sim \rho$
  for each period $h = 1, \ldots, H$:
    Step 1 ($\epsilon$-greedy exploration): with probability $\epsilon$, choose action
    $a_{th}$ uniformly randomly from $\mathcal{A}$; with probability $1 - \epsilon$, choose

        $a_{th} \sim \mathrm{unif}\left(\arg\max_{a \in \mathcal{A}} Q_\theta(s_{th}, h, a)\right)$

    that is, $a_{th}$ is sampled uniformly randomly from $\arg\max_{a \in \mathcal{A}} Q_\theta(s_{th}, h, a)$
    Step 2: take action $a_{th}$, observe reward $r_{th}$; if $h < H$, also observe the next
    state $s_{t,h+1}$
    Step 3: store transition $(s_{th}, h, a_{th}, r_{th}, s_{t,h+1})$ in the replay buffer $\mathcal{D}$;
    if $h = H$, set $s_{t,h+1} =$ null
    Step 4: sample a random minibatch of transitions $(s_j, h_j, a_j, r_j, s'_j)$ for
    $j = 1, 2, \ldots, B$ from $\mathcal{D}$, and set

        $y_j = r_j + \max_{a' \in \mathcal{A}} Q_{\theta_-}(s'_j, h_j + 1, a') \quad \forall j = 1, 2, \ldots, B$      (2.29)

    we set $Q_{\theta_-}(s'_j, h_j + 1, a') = 0$ if $s'_j =$ null
    Step 5: define the loss function $\ell(\theta)$ and compute the gradient $g$

        $\ell(\theta) = \frac{1}{2} \sum_{j=1}^{B} \left(Q_\theta(s_j, h_j, a_j) - y_j\right)^2, \quad g = \nabla_\theta \ell(\theta),$

    and update $\theta \leftarrow$ optimizer$(\theta, g)$ to minimize $\ell(\theta)$
```

Deep Q-learning with experience replay is similar to the Q-learning algorithm described in Sect. 2.3.2.2. Specifically, its learning target is still the optimal state-action value function Q^*, it still uses ϵ-greedy exploration, and it is still an off-policy learning algorithm. However, there are two main differences: the first difference is that the deep Q-learning algorithm approximates Q^* by a neural network Q_θ and learns the parameter vector θ. The second difference is that it uses a technique known as experience replay (Lin, 1992) to enhance the data efficiency. Specifically, the transitions are stored in a replay buffer $\mathcal{D}$. At each period, a minibatch of transitions are sampled with replacement from $\mathcal{D}$, and the deep Q-learning algorithm updates θ using a stochastic gradient computed based on this minibatch. With experience replay, a transition $(s_{th}, h, a_{th}, r_{th}, s_{t,h+1})$ is potentially used in many parameter update steps, which allows for greater data efficiency.

We also would like to clarify some technical issues in the deep Q-learning algorithm described above. First, how to choose the architecture of Q_θ is highly non-trivial and in general application-dependent. Second, due to the memory space limit, the replay buffer $\mathcal{D}$ has a finite capacity N. Hence, when $\mathcal{D}$ is full and the agent would like to store a new transition, it needs to either delete a transition from $\mathcal{D}$ or discard the new transition. There are many ways to do it, and in the algorithm above, the buffer uses a first in, first out (FIFO) buffer replacement strategy. Third, it is worth mentioning that the optimization algorithm `optimizer` can be any gradient-based algorithm (Ruder, 2016), such as the stochastic gradient descent (SGD) algorithm and the Adam algorithm (Kingma & Ba, 2014). Note that some `optimizer` like Adam also needs to update the optimizer state (e.g., the first and second order moments in Adam), which is abstracted away from the pseudo-code above. Finally, note that in Eq. (2.29), y_j is computed based on θ_- instead of θ. Thus, the gradient g is

$$g = \sum_{j=1}^{B} \left(Q_\theta(s_j, h_j, a_j) - y_j \right) \nabla_\theta Q_\theta(s_j, h_j, a_j).$$

Also notice that though θ is updated in every period, θ_- (and hence Q_{θ_-}, the function used to compute the "target values" y_j's) remains fixed within one episode. Keeping θ_- fixed within one episode might be crucial for the convergence of the deep Q-learning algorithm in some applications.

Deep reinforcement learning (DRL) has been an active research area in the past decade, and the deep Q-learning algorithm described above is one of the first algorithms developed in this area. It is worth mentioning that one agent based on a variant of it has achieved a level comparable to that of a professional human games tester across 49 games of the challenging Atari 2600 games (Mnih et al., 2015). More advanced DRL agents, such as AlphaGo (Silver et al., 2016) and MuZero (Schrittwieser et al., 2020) have also been developed. Interested readers might refer to the references for further reading.

2.4 Conclusion and Further Reading

In this chapter, we have briefly reviewed some fundamental concepts, standard problem formulations, and classical algorithms of reinforcement learning (RL). Specifically, in Sect. 2.2, we have reviewed Markov decision processes (MDPs) and dynamic programming (DP), which provide mathematical foundations for both the problem formulation and algorithm design for RL. In Sect. 2.3, we have classified the RL algorithms based on their learning targets and reviewed some classical algorithms such as PSRL, Q-learning, Sarsa, and REINFORCE. We have also reviewed the standard exploration schemes in RL in Sect. 2.3.3 and reviewed approximate solution methods for large-scale RL problems in Sect. 2.3.4.

Before concluding this chapter, we would like to provide some pointers for further reading. Due to the space limit, we have not covered many exciting topics in RL, such as RL problems based on average-reward MDPs (see Mahadevan (1996) and Chap. 5 of Bertsekas (2011)), hierarchical reinforcement learning (Pateria et al., 2021; Al-Emran, 2015), multi-agent reinforcement learning (Busoniu et al., 2008; Zhang et al., 2021), imitation learning (Hussein et al., 2017), partially observable MDPs (Kaelbling et al., 1998), inverse reinforcement learning (Ng et al. 2000; Arora & Doshi, 2021), and safe reinforcement learning (Garcıa & Fernández, 2015). Interested readers might refer to the references for further reading. There are also several classical textbooks on RL and related topics, such as Sutton and Barto (2018), Bertsekas (2000, 2011, 2019), Szepesvári (2010), and Powell (2007). DRL has been an active research area in the past decade, and there are also some recent and more applied books on DRL (Lapan, 2018; Ravichandiran, 2018). Interested readers might also refer to them for further reading.

References

Al-Emran, M. (2015). Hierarchical reinforcement learning: A survey. *International Journal of Computing and Digital Systems, 4*(02). https://dx.doi.org/10.12785/IJCDS/040207

Arora, S., & Doshi, P. (2021). A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence, 297*, 103500.

Arulkumaran, K., Deisenroth, M. P., Brundage, M., & Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine, 34*(6), 26–38.

Bertsekas, D. (2019). *Reinforcement and optimal control*. Belmont: Athena Scientific

Bertsekas, D. P. (2000). *Dynamic programming and optimal control* (Vol. 1). Belmont: Athena scientific.

Bertsekas, D. P. (2011). *Dynamic programming and optimal control* (Vol. II, 3rd ed.). Belmont: Athena scientific.

Bishop, C. M. (2006). *Pattern recognition and machine learning (Information science and statistics)*. Berlin, Heidelberg: Springer.

Brafman, R. I., & Tennenholtz, M. (2002). R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research, 3*(Oct), 213–231.

Busoniu, L., Babuska, R., & De Schutter, B. (2008). A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 38*(2), 156–172.

Cesa-Bianchi, N., Gentile, C., Lugosi, G., & Neu, G. (2017). *Boltzmann exploration done right*. Preprint. arXiv:170510257.

Chen, X., Li, S., Li, H., Jiang, S., Qi, Y., & Song, L. (2019). Generative adversarial user model for reinforcement learning based recommendation system. In *International Conference on Machine Learning, PMLR* (pp. 1052–1061).

Dann, C., Lattimore, T., & Brunskill, E. (2017). *Unifying PAC and regret: Uniform PAC bounds for episodic reinforcement learning*. Preprint. arXiv:170307710.

Dayan, P. (1992). The convergence of td (λ) for general λ. *Machine Learning, 8*(3–4), 341–362.

Degris, T., White, M., & Sutton, R. S. (2012). *Off-policy actor-critic*. Preprint. arXiv:12054839.

Fischer, T. G. (2018). *Reinforcement Learning in Financial Markets—A Survey*. Tech. rep., FAU Discussion Papers in Economics.

Friedman, J., Hastie, T., Tibshirani, R., et al. (2001). *The elements of statistical learning*. *Springer series in statistics*. New York: Springer.

Garcıa, J., & Fernández, F. (2015). A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research, 16*(1), 1437–1480.

Gosavii, A., Bandla, N., & Das, T. K. (2002). A reinforcement learning approach to a single leg airline revenue management problem with multiple fare classes and overbooking. *IIE Transactions, 34*(9), 729–742.

Greensmith, E., Bartlett, P. L., & Baxter, J. (2004). Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research, 5*(9), 1471–1530.

Hasselt, H. (2010). Double q-learning. *Advances in Neural Information Processing Systems, 23*, 2613–2621.

Hussein, A., Gaber, M. M., Elyan, E., & Jayne, C. (2017). Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR), 50*(2), 1–35.

Jaakkola, T., Jordan, M. I., & Singh, S. P. (1994). On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation, 6*(6), 1185–1201.

Jaksch, T., Ortner, R., & Auer, P. (2010). Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research, 11*(4), 1563–1600.

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence, 101*(1–2), 99–134.

Kearns, M., & Singh, S. (2002). Near-optimal reinforcement learning in polynomial time. *Machine Learning, 49*(2), 209–232.

Kingma, D. P., & Ba, J. (2014). *Adam: A method for stochastic optimization*. Preprint. arXiv:14126980.

Kober, J., Bagnell, J. A., & Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research, 32*(11), 1238–1274.

Kushner, H., & Yin, G. G. (2003). *Stochastic approximation and recursive algorithms and applications* (Vol. 35). New York: Springer Science & Business Media.

Kuznetsova, E., Li, Y. F., Ruiz, C., Zio, E., Ault, G., & Bell, K. (2013). Reinforcement learning for microgrid energy management. *Energy, 59*, 133–146.

Kveton, B., Szepesvari, C., Wen, Z., & Ashkan, A. (2015). Cascading bandits: Learning to rank in the cascade model. In *International Conference on Machine Learning, PMLR* (pp. 767–776)

Lapan, M. (2018). *Deep reinforcement learning hands-on: Apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO, AlphaGo Zero and more*. Birmingham: Packt Publishing Ltd.

Lattimore, T., & Szepesvári, C. (2020). *Bandit algorithms*. Cambridge: Cambridge University Press.

Li, Y. (2017). *Deep reinforcement learning: An overview*. Preprint. arXiv:170107274.

Lin, L. J. (1992). *Reinforcement learning for robots using neural networks*. Pittsburgh: Carnegie Mellon University.

Lu, X., Van Roy, B., Dwaracherla, V., Ibrahimi, M., Osband, I., & Wen, Z. (2021). *Reinforcement learning, bit by bit*. Preprint. arXiv:210304047.

Mahadevan, S. (1996). Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine Learning, 22*(1), 159–195.

Marbach, P., & Tsitsiklis, J. N. (2001). Simulation-based optimization of Markov reward processes. *IEEE Transactions on Automatic Control, 46*(2), 191–209.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015) Human-level control through deep reinforcement learning. *Nature, 518*(7540), 529–533.

Ng, A. Y., Russell, S. J., et al. (2000). Algorithms for inverse reinforcement learning. In *ICML* (Vol. 1, p. 2).

Osband, I., Russo, D., & Van Roy, B. (2013). *(More) Efficient reinforcement learning via posterior sampling*. Preprint. arXiv:13060940.

Osband, I., Van Roy, B., Russo, D. J., Wen, Z., et al. (2019) Deep exploration via randomized value functions. *Journal of Machine Learning Research, 20*(124), 1–62.

Pateria, S., Subagdja, B., Tan, A. H., & Quek, C. (2021). Hierarchical reinforcement learning: A comprehensive survey. *ACM Computing Surveys (CSUR), 54*(5), 1–35.

Powell, W. B. (2007). *Approximate dynamic programming: Solving the curses of dimensionality* (Vol. 703). New York: Wiley.

Ravichandiran, S. (2018). Hands-on reinforcement learning with Python: Master reinforcement and deep reinforcement learning using OpenAI gym and tensorFlow. Birmingham: Packt Publishing Ltd.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. Preprint. arXiv:160904747.

Rummery, G. A., & Niranjan, M. (1994). *On-line Q-learning using connectionist systems* (Vol. 37). Citeseer.

Russo, D., & Van Roy, B. (2014). Learning to optimize via information-directed sampling. *Advances in Neural Information Processing Systems, 27*, 1583–1591.

Russo, D., Van Roy, B., Kazerouni, A., Osband, I., & Wen, Z. (2017). *A tutorial on Thompson sampling*. Preprint. arXiv:170702038.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. (2020). Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature, 588*(7839), 604–609.

van Seijen, H. (2016). *Effective multi-step temporal-difference learning for non-linear function approximation*. Preprint. arXiv:160805151.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature, 529*(7587), 484–489.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2017a). *Mastering chess and shogi by self-play with a general reinforcement learning algorithm*. Preprint. arXiv:171201815.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017b). Mastering the game of go without human knowledge. *Nature, 550*(7676), 354–359.

Singh, S., Jaakkola, T., Littman, M. L., & Szepesvári, C. (2000). Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning, 38*(3), 287–308.

Sutton, R. S. (1984). *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts Amherst.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning, 3*(1), 9–44.

Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in neural information processing systems* (pp. 1038–1044). Cambridge: MIT Press.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. Cambridge: MIT Press.

Sutton, R. S., McAllester, D. A., Singh, S. P., & Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems* (pp. 1057–1063).

Szepesvári, C. (2010). Algorithms for reinforcement learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning, 4*(1), 1–103.

Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika, 25*(3/4), 285–294.

Tsitsiklis, J. N. (1994). Asynchronous stochastic approximation and q-learning. *Machine Learning, 16*(3):185–202.

Van Seijen, H., Van Hasselt, H., Whiteson, S., & Wiering, M. (2009). A theoretical and empirical analysis of expected Sarsa. In *2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning* (pp. 177–184). New York: IEEE.

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine Learning, 8*(3–4), 279–292.

Wen, Z., & Van Roy, B. (2017). Efficient reinforcement learning in deterministic systems with value function generalization. *Mathematics of Operations Research, 42*(3), 762–782.

Wen, Z., O'Neill, D., & Maei, H. (2015). Optimal demand response using device-based reinforcement learning. *IEEE Transactions on Smart Grid, 6*(5), 2312–2324.

Wen, Z., Precup, D., Ibrahimi, M., Barreto, A., Van Roy, B., & Singh, S. (2020). On efficiency in hierarchical reinforcement learning. *Advances in Neural Information Processing Systems* (Vol. 33)

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning, 8*(3), 229–256.

Zhang, K., Yang, Z., & Başar, T. (2021). Multi-agent reinforcement learning: A selective overview of theories and algorithms. In *Handbook of reinforcement learning and control* (pp. 321–384).

Zhang, W., Zhao, X., Zhao, L., Yin, D., Yang, G. H., & Beutel, A. (2020). Deep reinforcement learning for information retrieval: Fundamentals and advances. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 2468–2471)