Lecture 3: Finite Markov Decision Process (MDP)

Tianpei Xie

Aug 1st., 2022

Contents

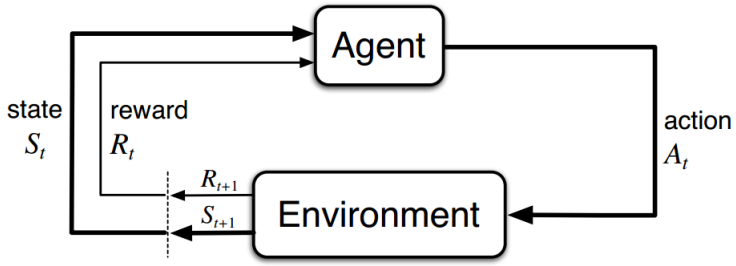


Figure 3.1: The agent–environment interaction in a Markov decision process.

Figure 1: Markov Decision process formulation. [Sutton and Barto, 2018]

1 Markov Decision Process

Markov Decision Processes (MDPs) are a classical formalization of sequential decision making, where actions influence not just immediate rewards, but also subsequent situations, or states, and through those future rewards. Thus MDPs involve delayed reward and the need to tradeoff immediate and delayed reward. MDPs are meant to be a straightforward framing of the problem of learning from interaction to achieve a goal. There are several aspects in the formulation:

- **agents**: The learner and decision maker;
- **environment**: The thing agent interacts with, comprising everything outside the agent;
- the agent and environment interact at each of a sequence of **discrete time steps** $t = 1, 2, \ldots,$;
- At each time step t, the agent receives some representation of the environments **state**, $S_t \in \mathcal{S}$; state is the *basis* on which the action is taken
- and on that basis selects an **action**, $A_t \in \mathcal{A}(s)$. Action signals represent *choices* made by the agent.
- One time step later $t+1$, in part as a consequence of its action, the agent receives a numerical **reward**, $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$, and finds itself in a **new state**, S_{t+1}; The reward defines the ***objective*** of learning and the ***goal*** of agent.
- For finite MDP, i.e. $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ are finite sets, the **dynamic function** is defined as $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathcal{R} \to [0, 1]$, $p(s', r|s, a) := P(S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a)$ for all $s, s' \in \mathcal{S}, a \in \mathcal{A}(s), r \in \mathcal{R}$. The function p defines the ***dynamics*** of the MDP. (| in p is borrowed from conditional prob but the function determined by all of inputs)

This formulation is called *Markov Decision Process* since the future states and rewards are only dependent on the immediately preceding state and action, not past histories. Figure 1 illustrate the relations between these aspects. We can fully represent a MDP as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p(\cdot|\cdot))$.

The formulation of MDP has high requirement on definition of *state*. The state must include information about *all aspects of the past agent-environment interaction* that make a difference for the future. The *state transition probablity* can be obtained via marginalization

$$p(s'|s, a) = \sum_{r \in \mathcal{R}} p(s', r|s, a).$$

The *expected rewards* for state-action pair $r : \mathcal{S} \times \mathcal{A} \to \mathcal{R}$ can be obtained

$$r(s,a) = \mathbb{E}_p \left[R_{t+1} | S_t = s, A_t = a \right] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r | s, a). \tag{1}$$

The *expected rewards* for state-action-next-state pair $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathcal{R}$ can be obtained

$$r(s,a,s') = \mathbb{E}_p \left[R_{t+1} | S_{t+1} = s', S_t = s, A_t = a \right] = \sum_{r \in \mathcal{R}} r \frac{p(s', r | s, a)}{p(s' | s, a)}$$

The MDP framework is abstract and flexible and can be applied to many different problems in many different ways. In general, actions can be any decisions we want to learn how to make, and the states can be anything we can know that might be useful in making them. In particular, the boundary between agent and environment is typically not the same as the physical boundary of a robots or animals body. Usually, the boundary is drawn closer to the agent than that. Rewards, too, presumably are computed inside the physical bodies of natural and artificial learning systems, but are considered external to the agent.

The general rule we follow is that *anything that* ***cannot be changed arbitrarily*** *by the agent is considered to be* ***outside*** *of it* and thus part of its environment. We do not assume that everything in the environment is unknown to the agent. For example, the agent often knows quite a bit about how its rewards are computed as a function of its actions and the states in which they are taken. But we always consider the *reward computation to be* ***external*** to the agent because it defines the task facing the agent and thus must be beyond its ability to change arbitrarily. In fact, in some cases the agent may know everything about how its environment works and still face a diffcult reinforcement learning task, just as we may know exactly how a puzzle like Rubiks cube works, but still be unable to solve it. **The agent-environment boundary represents the limit of the agents absolute control, not of its knowledge**. In practice, the agentenvironment boundary is determined once one has selected particular states, actions, and rewards, and thus has identified a specific decision making task of interest.

2 Goals and Rewards

In reinforcement learning, the **purpose or goal** of the agent is formalized in terms of a special signal, called the ***reward***, passing from the environment to the agent. Informally, the agents goal is to maximize the total amount of reward it receives. We can clearly state this informal idea as the ***reward hypothesis***:

"That all of what we mean by goals and purposes can be well thought of as the **maximization** of the ***expected*** value of the ***cumulative sum*** of a received **scalar** signal (called reward)."

The use of a *reward signal* to formalize the *idea of a goal* is one of the most **distinctive features of reinforcement learning**. Note that the reward signal is **not** the place to impart to the agent prior knowledge about how to achieve what we want it to do. If the winning is the goal, then achieving subgoals are not considered as rewards, since the agents may stop when achieving the subgoal.

In many real-world examples, however, the reward hypothesis may not hold. For instance, if the goal is to achieve the best outcome under worst case situations, then simply maximing the rewards may not yield the good solution. Similar case such as risk-sensitive/avert examples also requires

us to find solution that may not be the best in terms of cumulative rewards but avoids the worst situations. Also, sometimes, if the goal is very far away, and solving sub-problems may still make sense to achive the end results,

3 Returns and Episodes

The **return** is defined as the *cumulative rewards in the long run/ in future*. In general, we seek to maximize the ***expected return***, where the **return**, denoted G_t, is defined as some specific function of the reward sequence.

$$G_t = \sum_{\tau > t} R_\tau.$$

In applications, where a final state at T can be observed and after that, the enviroment and agents will be reset to inital state, the agent-environment interaction breaks naturally into *subsequences*, which we call **episodes**. Examples include games such as chess, maze et al. Each episode ends in a special state called **the terminal state**, followed by a reset to a standard starting state or to a sample from a standard distribution of starting states. Tasks with episodes of this kind are called **episodic tasks**. In episodic tasks we sometimes need to distinguish the set of all nonterminal states, denoted $\mathcal{S}$, from the set of all states plus the terminal state, denoted $\mathcal{S}+$. The time of termination, T, is a random variable that normally varies from episode to episode. For episodic tasks, the return is finite $G_t = \sum_t^T R_\tau$.

On the other hand, in many cases the agentenvironment interaction does not break naturally into identifiable episodes, but goes on continually *without limit*. For example, this would be the natural way to formulate an on-going process-control task, or an application to a robot with a long life span. We call these **continuing tasks**. The return for continuing task can be infinite. So, in order to compute finite expectation of returns, we usually apply **discounts** γ for future rewards based on the time passing. The ***discounted return*** is formulated as below

$$\begin{aligned} G_t &= \sum_{\tau > t} \gamma^\tau R_\tau < \infty \\ &= R_{t+1} + \gamma\, G_{t+1}, \end{aligned} \tag{2}$$

where $\gamma \in [0, 1]$. The **discount rate** γ determines the *present* value of *future rewards*. Then $\gamma = 0$, $G_t = R_{t+1}$, i.e. the short-sighted returns. If $\gamma \to 1$, we have the far-sighted returns, where the return objective takes future rewards into account more strongly.

We can combine the notion of returns for episodic and continuing task as follows:

$$\begin{aligned} G_t &= \sum_{\tau=t+1}^{T} \gamma^{\tau-t-1} R_\tau. \\ &= R_{t+1} + \gamma\, G_{t+1} \end{aligned} \tag{3}$$

Note that we can allow $T = \infty$ or $\gamma = 1$ but not both. To convert an episodic task into continuing task, we can define an absorbing state that transitions only to to itself and generates zero rewards.

4 Policies and Value Functions

The policy and value functions are defined as follows:

- Formally, a ***policy*** is a mapping from *states* to probabilities of selecting each possible *action*. Define $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ as the policy function. $\pi(a|s) = P(A_t = a|S_t = s)$ is the probability that $A_t = a$ if $S_t = s$. Like p, π is an ordinary function; the $|$ in the middle of $\pi(a|s)$ merely reminds that it defines a probability distribution over $a \in \mathcal{A}(s)$ for each $s \in \mathcal{S}$. Reinforcement learning methods specify how the agents policy is *changed* as a result of its experience.
  - **deterministic policy**: when $\pi : \mathcal{S} \to \mathcal{A}$ is a deterministic function, i.e. given a state, the policy $\pi(a|s) = \mathbb{1}\{a\}$ only assign a fixed action a from the state s.
  - **stochastic policy**: when $\pi(a|s) = P(A_t = a|S_t = s)$, i.e. the policy randomly select an action according to a probability distribution $\pi(a|s)$
- The ***value function (state-value function)*** $v_\pi : \mathcal{S} \to \mathbb{R}$ of **a *state*** s **under a *policy*** π, denoted $v_\pi(s)$, is the ***expected future return*** when *starting* in s and *following* π thereafter.

$$\begin{aligned} v_\pi(s) &:= \mathbb{E}_\pi\left[G_t|S_t = s\right] \\ &= \mathbb{E}_\pi\left[\sum_{\tau=0}^{\infty} \gamma^\tau R_{\tau+t+1}\Big|S_t = s\right], \quad \gamma \in [0,1), \forall\, s \in \mathcal{S}, \end{aligned} \tag{4}$$

  where $\mathbb{E}_\pi[\cdot]$ denotes the expected value of a random variable given that the agent follows policy π, and t is any time step.
- The ***action-value function for policy*** π. denoted $q_\pi(s, a)$, is the ***expected return*** starting from s, taking the action a, and thereafter following policy π:

$$\begin{aligned} q_\pi(s,a) &:= \mathbb{E}_\pi\left[G_t|S_t = s,\ A_t = a\right] \\ &= \mathbb{E}_\pi\left[\sum_{\tau=0}^{\infty} \gamma^\tau R_{\tau+t+1}\Big|S_t = s,\ A_t = a\right], \end{aligned} \tag{5}$$

Note that the *policy* π is the function of **current state value only**. It does not depends on previous actions, states or time. The value function reflects an estimate of *how good* it is for the agent to be in a given state (or how good it is to perform a given action in a given state). The notion of how good here is defined in terms of future rewards that can be expected, or, to be precise, in terms of *expected return*. Both $v_\pi(s)$ and $q_\pi(s, a)$ can be estimated via averaging from samples with Monte Carlo methods.

See following exercises.

Exercise 4.1 *Give an equation for v_π in terms of q_π and π.*

Solution: Note that v_π can be obtained directly via marginalization of q_π

$$\begin{aligned} v_\pi(s) &:= \mathbb{E}_\pi\left[G_t|S_t = s\right] \\ &= \mathbb{E}_\pi\left[\mathbb{E}_\pi\left[G_t|S_t = s,\ A_t = a\right]\right] \\ &= \mathbb{E}_\pi\left[q_\pi(s, A_t)|S_t = s\right] \qquad (6) \\ &= \sum_a \pi(a|s) q_\pi(s,a). \ \blacksquare \qquad (7) \end{aligned}$$

Exercise 4.2 *Give an equation for q_π in terms of v_π and the four-argument p*

Solution:

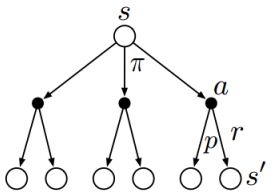
$$
\begin{aligned}
q_\pi(s,a) &:= \mathbb{E}_\pi\left[G_t | S_t = s,\ A_t = a\right] \\
&\text{(since } G_t = R_{t+1} + \gamma\, G_{t+1}\text{)} \\
&= \mathbb{E}_\pi\left[R_{t+1} | S_t = s,\ A_t = a\right] + \gamma\, \mathbb{E}_\pi\left[G_{t+1} | S_t = s,\ A_t = a\right] \\
&= \sum_{s'}\sum_{r} p(s', r|s, a) r + \gamma \sum_{s'}\sum_{r} p(s', r|s, a) \mathbb{E}_\pi\left[G_{t+1} | S_{t+1} = s', S_t = s,\ A_t = a\right] \\
&= \sum_{s'}\sum_{r} p(s', r|s, a)\left[r + \gamma\, \mathbb{E}_\pi\left[G_{t+1} | S_{t+1} = s', S_t = s,\ A_t = a\right]\right] \\
&\text{by Markov property} \\
&= \sum_{s'}\sum_{r} p(s', r|s, a)\left[r + \gamma\, \mathbb{E}_\pi\left[G_{t+1} | S_{t+1} = s'\right]\right] \\
&\text{(since } v_\pi(s') := \mathbb{E}_\pi\left[G_{t+1} | S_{t+1} = s'\right] \\
&= \underline{\sum_{s'}\sum_{r} p(s', r|s, a)\left[r + \gamma\, v_\pi(s')\right]} \qquad (8) \\
&= \underline{\mathbb{E}\left[R_{t+1} + \gamma\, v_\pi(S_{t+1}) | S_t = s, A_t = a\right]}.\ \blacksquare \qquad (9)
\end{aligned}
$$

4.1 Bellman equation

A fundamental property of value functions used throughout reinforcement learning and dynamic programming is that they satisfy ***recursive* relationships** similar to that which we have already established for the return.

From above equation (8) and substitue equation (7) we can find action-value function

$$
\begin{aligned}
q_\pi(s,a) &= \sum_{s'}\sum_{r} p(s', r|s, a)\left[r + \gamma\, v_\pi(s')\right] \\
&= \underline{\sum_{s',r} p(s', r|s, a)\left[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a')\right]}, \quad \forall\ s \in \mathcal{S},\ a \in \mathcal{A}(s) \qquad (10) \\
&= \underline{\mathbb{E}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a\right]} \qquad (11)
\end{aligned}
$$



Backup diagram for v_π

Figure 2: backup diagram. [Sutton and Barto, 2018]

Similarly, the derivation of equation (7) can be seen as

$$
\begin{aligned}
v_\pi(s) &:= \mathbb{E}_\pi\left[G_t | S_t = s\right] \\
&\text{(since } G_t = R_{t+1} + \gamma\, G_{t+1}\text{)} \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma\, G_{t+1} | S_t = s\right] \\
&= \mathbb{E}_{A_t, R_{t+1}, S_{t+1}}\left[\mathbb{E}_\pi\left[R_{t+1} + \gamma\, G_{t+1} | S_t = s, A_t = a, R_{t+1} = r, S_{t+1} = s'\right]\right] \\
&= \mathbb{E}_{A_t, R_{t+1}, S_{t+1}}\left[r + \gamma\, \mathbb{E}_\pi\left[G_{t+1} | S_t = s, A_t = a, R_{t+1} = r, S_{t+1} = s'\right]\right] \\
&\text{by Markov property} \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r | s, a)\left[r + \gamma \mathbb{E}_\pi\left[G_{t+1} | S_{t+1} = s'\right]\right] \\
&= \underline{\sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r | s, a)\left[r + \gamma v_\pi(s')\right]}, \quad \forall\ s \in \mathcal{S} \qquad (12) \\
&= \underline{\mathbb{E}\left[R_{t+1} + \gamma\, v_\pi(S_{t+1}) | S_t = s\right]} \qquad (13)
\end{aligned}
$$

Both (10) and (12) are referred as the ***Bellman equation*** for action-value function $q_\pi(s, a)$ and value function $v_\pi(s)$, respectively. Bellman equation states that for any policy π and any state s, the following consistency condition holds between the *value* of s and the *value* of its possible *successor states*; similarly for the action-value function:

$$
v_\pi(s) = r(s) + \gamma \sum_{s'} \left[\sum_a \pi(a|s) p(s'|s, a)\right] v_\pi(s'), \quad \forall\ s \in \mathcal{S} \qquad (14)
$$

$$
q_\pi(s, a) = r(s, a) + \gamma \sum_{s'} \sum_{a'} \left[\pi(a'|s') p(s'|s, a)\right] q_\pi(s', a'), \quad \forall\ s \in \mathcal{S},\ a \in \mathcal{A}(s) \qquad (15)
$$

where $r(s, a) := \mathbb{E}_\pi\left[R_{t+1} | S_t = s,\ A_t = a\right]$ from (1) and $r(s) = \mathbb{E}_\pi\left[r(s, a)|s\right] = \sum_a \pi(a|s) r(s, a)$. The Bellman equation states that the **value** of the *start state* must equal the (discounted) **value** of the **expected next state** (the second term), *plus* the **expected reward** *along the way* (first term).

Theorem 4.3 *For any finite state MDP, the Bellman equation in* (12) *has an unique solution.*

Since (13) is a system of $|\mathcal{S}|$ **linear equations** with $|\mathcal{S}|$ unknown values, there exists an unique solution.

The Bellman equation can be visualized via a diagram in Figure 2. We call this diagrams ***backup diagrams*** because they diagram relationships that form the basis of the *update* or *backup* operations that are at the heart of reinforcement learning methods. These operations transfer value information *back to a state* (or a stateaction pair) from *its successor states* (or stateaction pairs).

From Bellman equation we can see that the aggregate statistics on Markov network can be computed recursively via random walk.

4.2 Optimal Policy and Optimal Value Function

Value functions define a partial ordering over policies. A policy π is defined to be *better* than or *equal to* a policy $\pi^{'}$ if its expected return is greater than or equal to that of $\pi^{'}$ for all states. In other words, $\pi \geq \pi^{'}$ if and only if $v_{\pi}(s) \geq v_{\pi^{'}}(s)$ for all $s \in \mathcal{S}$. **There is always at least one policy that is better than or equal to all other policies**. This is an ***optimal policy***.

Note that if $v_{\pi_1}(s_1) \geq v_{\pi_2}(s_1)$ and $v_{\pi_1}(s_2) \leq v_{\pi_2}(s_2)$, then we can define a new policy $\pi_3 = \pi_1$ if $s = s_1$ and $\pi_3 = \pi_2$ if $s = s_2$ so that $\pi_3 \geq \pi_1$ and $\pi_3 \geq \pi_2$. For finite state MDPs, there always exists an optimal policy. Also even for finite state MDPs, **the optimal policy is not guaranteed to be unique**.

Denote the optimal policy as π_*. They share the *same state-value function*, called the **optimal state-value function**, denoted v_*, and defined as

$$\begin{aligned} v_*(s) &:= v_{\pi_*}(s) \\ &= \max_{\pi} v_{\pi}(s), \quad s \in \mathcal{S} \end{aligned} \tag{16}$$

Optimal policies also share the same **optimal action-value function**, denoted q_*, and defined as

$$\begin{aligned} q_*(s,a) &:= q_{\pi_*}(s,a) \\ &= \max_{\pi} q_{\pi}(s,a), \quad s \in \mathcal{S}, \;\; a \in \mathcal{A}(s) && (17) \\ &= \mathbb{E}\left[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, \, A_t = a\right] && (18) \end{aligned}$$

Because v_* is the value function for a policy, it must satisfy the self-consistency condition given by the Bellman equation for state values (12). Since v_* is the optimal value function, the Bellman equation can be reformulated as below

$$\begin{aligned} v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) \\ &= \max_{a \in \mathcal{A}(s)} \mathbb{E}_{\pi_*}\left[G_t | S_t = s, \, A_t = a\right] \\ &= \max_{a \in \mathcal{A}(s)} \mathbb{E}_{\pi_*}\left[R_{t+1} + \gamma G_{t+1} | S_t = s, \, A_t = a\right] \\ &= \max_{a \in \mathcal{A}(s)} \mathbb{E}\left[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, \, A_t = a\right] && (19) \\ &= \max_{a \in \mathcal{A}(s)} \sum_{s'} \sum_{r} p(s', r | s, a) \left[r + \gamma v_*(s')\right] \quad s \in \mathcal{S} && (20) \\ &= \max_{a \in \mathcal{A}(s)} \left[r(s,a) + \gamma \sum_{s'} p(s' | s, a) v_*(s') \right] && (21) \end{aligned}$$

(19), (21) and (20) are three forms of **Bellman optimality equations** for v_*. Note that in the equation above, the **optimal value function does not depend on the the policy** π. In other words, multiple optimal policies share the same optimal value function. For finite MDPs, although

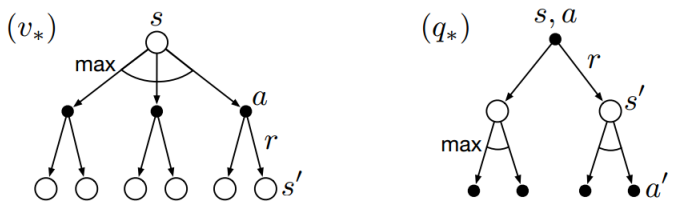


Figure 3.4: Backup diagrams for v_* and q_*

Figure 3: backup diagram for v_* and q_*. [Sutton and Barto, 2018]

the optimal policy π_* is not unique, the *optimal value function* v_* is ***unique*** and is determined by the Bellman equation (21) or (20).

Theorem 4.4 *For finite MDP, the Bellman optimality equation* (20) *for v_* has a unique solution.*

Same as the before, the Bellman optimality equation in (20) is a system of $|\mathcal{S}|$ equations with $|\mathcal{S}|$ unknowns.

The *Bellman optimality equation* for the *optimal action-value function* is

$$\begin{aligned} q_*(s,a) &= \mathbb{E}\left[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s,\, A_t = a\right] \\ &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a' \in \mathcal{A}(s)} q_*(S_{t+1}, a') \middle| S_t = s,\, A_t = a\right] \qquad (22) \\ &= \sum_{s'} \sum_{r} p(s', r|s, a) \left[r + \gamma \max_{a' \in \mathcal{A}(s)} q_*(s', a')\right] \qquad (23) \\ &= r(s,a) + \gamma \sum_{s'} p(s'|s,a) \max_{a' \in \mathcal{A}(s)} q_*(s', a') \qquad (24) \end{aligned}$$

The backup diagrams in the Figure 3 show graphically the spans of future states and actions considered in the Bellman optimality equations for v_* and q_*. These are the same as the backup diagrams for v_π and q_π presented earlier except that arcs have been added at the agents choice points to represent that the maximum over that choice is taken rather than the expected value given some policy. The backup diagram on the left graphically represents the Bellman optimality equation (20) and the backup diagram on the right graphically represents (23).

Given the optimal value v_* for each state, an **optimal policy** can be obtained by constructing a *greedy* policy: $\pi(a|s) = 0$ for all $a \notin \mathcal{A}_*(s)$, where

$$\mathcal{A}_*(s) := \text{argmax}_{a \in \mathcal{A}(s)} \sum_{s'} \sum_{r} p(s', r|s, a)[r + \gamma v_*(s')]$$

is a set of actions that reach the maximum in the Bellman equation (20). This is a *greedy policy* since it selects actions based only on their short-term consequences. This is an **one-step search**. If you have the optimal value function, v_*, then the actions that appear best after a one-step search will be optimal actions. Although the (optimal) value function v_* is used to evaluate the short-term consequences, a greedy policy is actually optimal in the long-term sense. The *optimal value function* encodes the *reward* consequences of all possible future behavior. v_* stores the optimal expected long-term return locally and is immediately available for each state.

Based on the greedy deterministic optimal policy π_* constructed as above, we can also derive the Bellman optimality equation based on Bellman equation in (12):

$$\begin{aligned} v_*(s) = v_{\pi_*}(s) &= \sum_a \pi_*(a|s) \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma v_{\pi_*}(s') \right], \quad \forall\ s \in \mathcal{S} \\ &\quad \text{(consider a deterministic optimal policy } \pi_* = \mathbb{1}\{A_t = a \in \mathcal{A}_*(s)\}) \\ &= \max_{a \in \mathcal{A}} \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma v_*(s') \right] \end{aligned}$$

Given the optimal action-value q_* choosing optimal actions is even easier. With q_*, the agent does not even have to do a one-step-ahead search: for any state s, it can simply find any action that maximizes $q_*(s, a)$. The action-value function effectively *caches* the *results of all one-step-ahead searches*. It provides the optimal expected long-term return as a value that is locally and immediately available for each stateaction pair.

Explicitly solving the Bellman optimality equation provides one route to finding an optimal policy, and thus to solving the reinforcement learning problem. In practice, this brute-force search rarely works since the number of states $|\mathcal{S}|$ is usually extremely large for real-world problems, thus making the solution finding very inefficient. Moreover, this solution relies on at least three assumptions that are rarely true in practice:

1. we accurately know the *dynamics* of the environment;
2. we have enough *computational resources* to complete the computation of the solution;
3. the Markov property.

For the kinds of tasks in which we are interested, one is generally not able to implement this solution exactly because various combinations of these assumptions are violated.

Many different decision-making methods can be viewed as ways of **approximately** solving the Bellman optimality equation. For example, heuristic search methods such as A_* are almost always based on the episodic case. It can be viewed as expanding the right-hand side of (20) several times, up to some depth, forming a tree of possibilities, and then using a heuristic evaluation function to approximate v_* at the leaf nodes. We can also solve (20) via **dynamic programming algorithm**. Many reinforcement learning methods can be clearly understood as ***approximately*** solving the Bellman optimality equation, using actual *experienced transitions* in place of knowledge of the *expected transitions*.

References

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.