2
Sequential Decisions DRAFT

Many decision making settings of interest have a sequential aspect, from solving puzzles to personalizing user news feeds. For such applications, the decision context is not an independent sample from some distribution, but rather an observation which depends on the sequence of actions. In this chapter we generalize our understanding of reinforcement learning to this sequential setting. We begin by just considering auto-regressive policies, which are a compact way to handle combinatorial action spaces. Then, we introduce a general notion of "state" and see how it opens up the scope of settings in which reinforcement learning can be applied. We consider interactive decision problems on indefinite time horizons, and carefully build the foundations needed for this setting.

2.1 From Auto-regressive Policies to State

In applications like recommendations and language models, the action space can grow quite large. Selecting a single recent movie (as in Example 1.1) requires choosing one from a hundred options, but selecting a ranked list of ten suggestions has *quintillions* of different possibilities. Similarly, while the number of characters is bounded, the possible combinations into words and sentences grows exponentially large. For such combinatorial action spaces, it would be unreasonable to directly consider such a large number of possibilities at once.

$100 \times 99 \times ... \times 91 \approx 8 \times 10^{18}$

A common strategy is to use an auto-regressive policy architecture: make one choice at a time, with each one considering the past choices. For personalized recommendation, this means choosing the top movie, then the next best movie of those remaining, then the next best, etc. For language models, this means generating one word (or token), then appending it to the context, then generating the next, and so on. At each step, it is only necessary to make a choice from a relatively small set of options. The only new requirement is that the

policy must take as input the choice history.

To formalize these ideas, we introduce some notation. We will write $\Pi_\theta(x)$ as the composite policy which produces the combinatorial outputs based on the observed context x, and use π_θ to denote the auto-regressive policy. This notation departs from that of the prior chapter. Denote by a_t the choice made at sequence position t, and letting H be the maximum sequence length, we denote by $a_{0:H-1}$ the full sequence, e.g. the list of movies or the generated sentence. We have that $a_{0:H-1} \sim \Pi_\theta(x)$ according to

$$\Pi(a_{0:H-1}|x) = \prod_{t=0}^{H-1} \pi(a_t|a_{0:t-1}, x)\,.$$

We have changed the notation for policies and actions: π vs. Π; a_i vs. $a_{0:H-1,i}$

Aside from notation, nothing has yet changed from the prior chapter. In order to evaluate or optimize this policy, we need only translate the composite probabilities to the auto-regressive policy. For the importance weight, this boils down to

$$\frac{\Pi_\theta(a_{0:H-1}|x)}{\Pi_{\theta'}(a_{0:H-1}|x)} = \prod_{t=0}^{H-1} \frac{\pi_\theta(a_t|a_{0:t-1}, x)}{\pi_{\theta'}(a_t|a_{0:t-1}, x)}\,.$$

For the score,

$$\nabla \log \Pi_\theta(a_{0:H-1}|x) = \sum_{t=0}^{H-1} \nabla \log \pi_\theta(a_t|a_{0:t-1}, x)\,.$$

With these expressions, we can directly use the policy optimization algorithms from the previous chapter.

There may be something unsatisfying about conditioning π on the entire history $a_{0:t-1}$, whose size grows with t. It seems inefficient, especially in settings where much of the history may be irrelevant to the decision at hand.

Example 2.1 (Maze navigation)**.** Consider using an auto-regressive policy to solve a maze: at each step, the policy observes the full sequence of past moves (up, down, left, right) and the initial map, and decides the next move. Alternatively, we could track the agent's current position on the two-dimensional grid, and choose the next move based on that position alone.

In principle, the auto-regressive policy can do just as well because the position can always be reconstructed from the initial map and the action sequence. However, it is the agent's position, not the path, that matters in a maze. The position is a far more natural and lower-dimensional summary.

Many problems of interest admit a summary like position in the maze example – quantities which capture everything relevant to future decisions, potentially without retaining the full history. We call such a summary the *state*, and denote it s_t. A state always exists, in the trivial sense that we may take $(a_{0:t-1}, x)$ as the state. Whether the state is a true compression depends on the setting.

Once an action is taken, the state may change. For example, in a maze, the position will update left, right, up, or down depending on the selected action. In the auto-regressive case, the new action is appended $(a_{0:t-1}, x) \to (a_{0:t}, x)$. Notice that in both examples, the next state depends only on the current state and action. This dependence will be a key assumption we make about how states *transition*. We may want to consider nondeterministic transitions.

Example 2.2 (Maze with stochastic transitions)**.** Suppose moves in a maze are unreliable: heading left succeeds only some of the time, and otherwise the agent stays put. Some positions may be "slippery", causing the agent to sometimes move down regardless of the action. A combinatorial policy $\Pi(a_{0:T}|x)$ commits to a full move sequence in advance, without observing outcomes. If position can be observed, a position-dependent policy will be able to react to where the agent actually ends up.

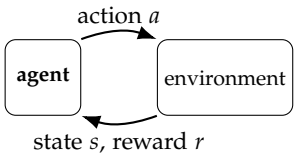

Figure 2.1: The interaction loop.

The uncertain maze example shows that the concept of state can lead to something new. Observing the state after each action and reacting to it is a substantively different setting than committing to an action sequence based on the initial context alone. This *feedback* reduces the variance of the outcome, since the policy can correct for randomness in the transitions. This is a truly *interactive* setting, in that the agent's actions change the state of the environment, and the agent in turn observes and responds to the state it has produced.

Exercise 1. *For an auto-regressive policy, the off-policy gradient estimate is*

$$\frac{1}{n}\sum_{i=1}^{n}\prod_{t=0}^{H-1} w_{t,i} \sum_{t=0}^{H-1} \nabla \log \pi_\theta(a_{t,i}|a_{0:t-1,i}, x_i)\, r_i.$$

for $w_{t,i} = \frac{\pi_\theta(a_{t,i}|a_{0:t-1,i},x_i)}{\pi_{\theta'}(a_{t,i}|a_{0:t-1,i},x_i)}$*. A common approximation is to use the estimate*

$$\frac{1}{n}\sum_{i=1}^{n}\sum_{t=0}^{H-1} w_{t,i} \nabla \log \pi_\theta(a_{t,i}|a_{0:t-1,i}, x_i)\, r_i.$$

This approximation is commonly made in the PPO "clip" algorithm (and variants like GRPO) because it allows the importance weight of each point in the sequence to be decoupled.

Suppose that $|w_{t,i} - 1| \le \epsilon$*. Find an upper bound on the error of this approximation.* Hint: you may first want to use $(1-\epsilon)^m \ge 1 - m\epsilon$.

2.2 *Sequential Interaction*

Motivated by the previous section, we formalize interactive decision settings with states and transitions. States now subsume the notion of context x considered in the previous chapter.

The state of the environment is denoted s and the *state space* $\mathcal{S}$ is the set of all possible states. There is an *initial state distribution* denoted $\mu \in \Delta(\mathcal{S})$. Actions are still denoted by $a \in \mathcal{A}$, and rewards depend on states and actions according to $r \sim R(s,a)$ for $R : \mathcal{S} \times$

$\mathcal{A} \to \Delta(\mathbb{R})$. The environment satisfies the *Markov property* if the probability of the next state only depends on the current state and action. Mathematically,

$$\mathbb{P}(s_{t+1} \mid s_0, a_0, \ldots, s_t, a_t) = P(s_{t+1} \mid s_t, a_t)$$

where $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ describes the *state transitions*.

Named for the Russian mathematician Andrey Markov, who introduced this property in the early 1900s while developing theory for sequences of dependent random variables, which he later used to analyze letter patterns in the classic Russian novel *Eugene Onegin* by Alexander Pushkin.

We consider sequential settings, indexed by time steps $t = 0, 1, 2, \ldots$ and continuing indefinitely. The quantity of interest is now *cumulative* reward over these time steps. We introduce a *discount factor* $\gamma \in [0, 1]$ which multiplies future rewards as they accumulate:

$$r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k}.$$

Indefinite horizons need not be literally infinite. Termination can be modeled with an *absorbing state*, i.e. a state that transitions only to itself. For example, the end state of a maze or the solve state of a puzzle. Mathematically, we write infinite sums, but in practice, the episode terminates at some finite time.

We can think of γ as measuring how much we care about the future: if it is close to 0, we only care about near-term rewards; if it is close to 1, we put more weight on rewards far in the future. Whenever $\gamma < 1$, we can guarantee that the cumulative reward is bounded as long as $R(s, a)$ is bounded. Reviewing geometric series, you can see that when the maximum reward at any time step is one, then the maximum cumulative (discounted) reward is bounded by $\frac{1}{1-\gamma}$.

Another way to view γ is as the probability of *continuing* time step (and otherwise terminating early). If γ is close to 0, trajectories will typically be short, while if γ is close to 1, they will typically be long. As long as $\gamma < 1$, we can expect a finite time termination.

Putting together all of the components, this formal setting is called a *Markov decision process* (MDP), specified by the tuple

$$M = (\mathcal{S}, \mathcal{A}, \mu, P, R, \gamma).$$

These components of this tuple correspond to the properties of the environment in which our reinforcement learning agent will act.

The agent chooses actions, which in general may depend on everything up until time t. Given a policy, we repeatedly chose actions, transition states, and collect reward until termination. We call the resulting sequence of states, actions, and rewards a *trajectory*:

$$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots)$$

We call the interactive process which generates a trajectory *a roll-out* or *an episode*. The trajectory is a sample from a distribution induced by μ, P, π, and R. Due to the Markov property, we mainly consider state-dependent policies $\pi : \mathcal{S} \to \Delta(\mathcal{A})$. Then, using the Markov property, the distribution ρ^π is given by

$$\rho^\pi(\tau) = \mu(s_0)\pi(a_0 \mid s_0)R(r_0|s_0, a_0)P(s_1 \mid s_0, a_0)\pi(a_1 \mid s_1)R(r_1|s_1, a_1)\cdots$$

Since μ, P, r are fixed by the environment, our notation usually suppresses the dependence on them.

We can now formalize the main goal in reinforcement learning which is to find a policy that maximizes expected cumulative reward:

$$\max_\pi \ \mathbb{E}_{\tau\sim\rho^\pi}[r_0 + \gamma r_1 + \gamma^2 r_2 \cdots].$$

It would be natural to introduce parametrized policies, as in the previous chapter, and revisit estimation, gradients, and policy optimization in this sequential setting. In fact, we will do just that in a couple of chapters. In the present chapter, we instead develop further fundamental understanding of what sequential decisions entail.

2.3 *Exact Policy Evaluation*

Suppose we are handed the exact reward function and transition model. How can we exactly evaluate the performance of some given policy? In the previous chapter, we focused on Monte Carlo estimation, which only ever provides approximate quantities.

Towards exact evaluation, we need a way to make the infinite sum (over the indefinite horizon) tractable. Our first step is to define a state-dependent version of the cumulative reward. We define the *value function* $V^\pi : \mathcal{S} \to \mathbb{R}$ to be the expected total reward starting in a given state and following policy π.

$$V^\pi(s) = \mathbb{E}_{\tau\sim\rho^\pi}[r_0 + \gamma r_1 + \gamma^2 r_2 \cdots \mid s_0 = s]$$

This way of defining a value function is useful because it leads to a recursive description: the cumulative reward from state s onwards is just the *current* reward plus the discounted total reward from the *next* state onwards. Mathematically,

Definition 2.1. *Bellman consistency equation for the value function*

$$V^\pi(s) = \mathbb{E}_{\substack{a\sim\pi(s)\\ s'\sim P(s,a)}} [\mathbb{E}_{r\sim R(s,a)}[r] + \gamma V^\pi(s')] \quad \forall\, s \in \mathcal{S} \tag{2.1}$$

Named after Richard Bellman who worked on decision and control theory in the context of U.S. Air Force logistics and control problems at the RAND corporation in the 1950s.

This is called the Bellman consistency equation. Recalling that expectation is a linear operation, this equation is actually a system of linear equations which characterize the value function. To see this concretely, let us restrict attention to the case of finitely many states, so that the expectation over s' can be written as an explicit sum rather than an integral. Under this assumption, the Bellman consistency equation becomes the set of $|\mathcal{S}|$ equations

$$V^\pi(s) = \mathbb{E}_{\substack{a\sim\pi(s)\\ r\sim R(s,a)}} [r] + \gamma \sum_{s'\in\mathcal{S}} \mathbb{E}_{a\sim\pi(s)}[P(s'|s,a)]\, V^\pi(s') \quad \forall\, s \in \mathcal{S}\,,$$

where we have also simplified the expression using the linearity of expectation. Let's represent the value function as a vector V^π with dimension $|\mathcal{S}|$, and define

$$r^\pi \in \mathbb{R}^{|\mathcal{S}|}, \qquad r^\pi[s] = \mathbb{E}_{a\sim\pi(s)}\Big[\mathbb{E}_{r\sim R(s,a)}[r]\Big]\,,$$
$$P^\pi \in [0,1]^{|\mathcal{S}|\times|\mathcal{S}|}, \qquad P^\pi[s,s'] = \mathbb{E}_{a\sim\pi(s)}\big[P(s'|s,a)\big]\,.$$

Notice that by averaging the transition function $P(s'|s,a)$ over the actions $a \sim \pi(s)$, we have P^π as a transition matrix over states alone and r^π as a reward depending only on state. This state-only setting is a Markov chain: once a policy is fixed, the evolution of the state is described by this combination of policy and environment. Different policies result in different Markov chains. Designing a good policy can be viewed as picking a good Markov chain, i.e., one that is likely to spend a lot of time in high reward states.

The resulting Markov chain is called the *closed-loop* system.

Returning to the linear Bellman consistency equation, we use the matrix-vector notation and see that it can be solved with a matrix inversion:

For now, we assume that $I - \gamma P^\pi$ is invertible.

$$V^\pi = r^\pi + \gamma P^\pi V^\pi \implies V^\pi = (I - \gamma P^\pi)^{-1} r^\pi.$$

We see that exact policy evaluation reduces to solving a single linear system with a matrix inversion. The computational complexity of this operation depends on the size of the state space $|\mathcal{S}|$. We must also compute r^π and P^π by computing expectations over actions. When the action space $\mathcal{A}$ is also finite, this expectation is a simple weighted sum. We call the setting where both state and action spaces are finite the tabular MDP setting.

This matrix inversion takes $O(|\mathcal{S}|^3)$ time.

Exercise 2. *Assume that $\gamma < 1$. Show that $I - \gamma P^\pi$ is always invertible by showing that its nullspace is trivial.*

2.4 Iterative Policy Evaluation

When the state space grows large, a matrix inversion becomes a bottleneck. Can we trade off the requirement of finding the *exact* value function for a faster *approximate* algorithm? A simple idea for *iterative policy evaluation* is to start with a guess $V_0 \in \mathbb{R}^{|\mathcal{S}|}$ and then repeatedly apply the following equation

$$V_{t+1} = r^\pi + \gamma P^\pi V_t \,.$$

Each iteration only takes $O(|\mathcal{S}|^2)$ time for the matrix-vector multiplication. One benefit of this idea is that if we happened to guess the correct V^π, the Bellman consistency equation guarantees that the update will return the same V^π unchanged. The key question, however, is whether $V_t \to V^\pi$ for an incorrect initial guess V_0.

To answer this question, define the *Bellman consistency operator* of a policy π as

$$\mathcal{B}^\pi : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}, \quad \mathcal{B}^\pi(V) := r^\pi + \gamma P^\pi V$$

This is a mapping which takes a $|\mathcal{S}|$ dimensional vector and returns another $|\mathcal{S}|$ dimensional vector. Our algorithm idea is simply to

repeatedly apply $\mathcal{B}^\pi$. The Bellman consistency equation means that

$$V^\pi = \mathcal{B}^\pi(V^\pi),$$

or in other words, V^π is a *fixed point* of $\mathcal{B}^\pi$.

Does the Bellman consistency operator bring guesses V_t closer to the true V^π? It turns out that for any $\gamma < 1$ the answer is yes, and in fact that $\mathcal{B}^\pi$ is always a *contraction mapping*. This is a crucial property which ensures that outputs of $\mathcal{B}^\pi$ are always closer than the inputs were, for any pair of inputs. Let's measure closeness using the max norm

$$\|V\|_\infty := \sup_{s\in\mathcal{S}} |V(s)| .$$

Then we can show the contraction property:

$$\|\mathcal{B}^\pi(V) - \mathcal{B}^\pi(U)\|_\infty \le \gamma \|V - U\|_\infty .$$

We can prove this bound directly:

$$\begin{aligned}
|[\mathcal{B}^\pi(V)](s) - [\mathcal{B}^\pi(U)](s)| &= \Big| \mathop{\mathbb{E}}_{a\sim\pi(s)} \left[\mathop{\mathbb{E}}_{r\sim R(s,a)} [r] + \gamma \mathop{\mathbb{E}}_{s'\sim P(s,a)} [V(s')] \right] \\
&\qquad - \mathop{\mathbb{E}}_{a\sim\pi(s)} \left[\mathop{\mathbb{E}}_{r\sim R(s,a)} [r] + \gamma \mathop{\mathbb{E}}_{s'\sim P(s,a)} [U(s')] \right] \Big| \\
&= \gamma \left| \mathop{\mathbb{E}}_{a\sim\pi(s)} \mathop{\mathbb{E}}_{s'\sim P(s,a)} \left[V(s') - U(s') \right] \right| \\
&\le \gamma \mathop{\mathbb{E}}_{a\sim\pi(s)} \mathop{\mathbb{E}}_{s'\sim P(s,a)} \left| V(s') - U(s') \right| \qquad \text{(Jensen's inequality)} \\
&\le \gamma \max_{s'} |V(s') - U(s')| \qquad \text{(average less than maximum)} \\
&= \gamma \|V - U\|_\infty .
\end{aligned}$$

The first equality just expands the definition of $\mathcal{B}^\pi$ and in the second like terms cancel. The next line applies Jensen's inequality, moving the absolute value inside the expectation over a and s'. We then bound this expectation by the maximum value of $|V(s') - U(s')|$ over all s'. This is exactly $\|V - U\|_\infty$ by definition. Since this inequality holds for any s on the left hand side, we are done.

The contraction is true for any V, U, so we may apply it to the iterate V_t and the true value function V^π:

$$\underbrace{\|\mathcal{B}^\pi(V_t) - \mathcal{B}^\pi(V^\pi)\|_\infty}_{=\|V_{t+1} - V^\pi\|_\infty} \le \gamma \|V_t - V^\pi\|_\infty .$$

Applying this one-step bound repeatedly, starting from the initial guess V_0, results in geometric decay of the error after t iterations:

$$\|V_t - V^\pi\|_\infty \le \gamma^t \|V_0 - V^\pi\|_\infty .$$

How many iterations do we need for an ϵ-accurate estimate? We can work backwards to solve for t:

$$\begin{aligned}\gamma^t \|V_0 - V^\pi\|_\infty &\leq \epsilon \\ t &\geq \frac{\log(\epsilon / \|V_0 - V^\pi\|_\infty)}{\log \gamma} \\ &= \frac{\log(\|V_0 - V^\pi\|_\infty / \epsilon)}{\log(1/\gamma)}.\end{aligned}$$

Thus, the number of iterations is only logarithmic in $1/\epsilon$.

Though we have been considering a finite state space, both for concreteness and because it lets us treat V_t as an ordinary vector in $\mathbb{R}^{|\mathcal{S}|}$, the same argument can be made more generally. We can define the Bellman contraction operator $\mathcal{B}^\pi : (\mathcal{S} \to \mathbb{R}) \to (\mathcal{S} \to \mathbb{R})$ as

$$[\mathcal{B}^\pi(V)](s) := \mathbb{E}_{\substack{a \sim \pi(s) \\ s' \sim P(s,a)}} \left[\mathbb{E}_{r \sim R(s,a)}[r] + \gamma V(s')\right].$$

This is still a contraction mapping and the Bellman consistency equation still ensures that V^π is a fixed point.

The convergence argument above is in fact a special case of a much more general template that we will reuse later on. It is important enough to have a special name: the *Banach fixed-point theorem*, which states the following. Suppose X is a complete vector space equipped with a norm and $T : X \to X$ is an operator with a fixed point $x^\star$ (i.e. $T(x^\star) = x^\star$) which is a contraction, i.e. $\|T(x) - T(x')\| \leq \gamma \|x - x'\|$ for $\gamma < 1$ and all x, x'. Then, repeatedly applying T from *any* starting point converges to $x^\star$ at a geometric rate governed by γ.

Named for Polish mathematician Stefan Banach, who was a founder of functional analysis.

Exercise 3. *Prove the Banach fixed point theorem.*

2.5 Fixed Horizon Policy Evaluation

So far, we rely on $\gamma < 1$ to compute the value function. Without discounting, the infinite sum may not converge, the matrix may not be invertible, and the Bellman operator is not guaranteed to be a contraction. Another way to guarantee that value is well-defined and computable is to guarantee that episodes always terminate at a fixed finite time. In particular, suppose the environment terminates after exactly H time steps, for some known *horizon* $H \in \mathbb{N}$.

This is actually a special case of our more general setting, but it is worth working out how it simplifies computation of the value function. It is a special case because we may augment the state to include the time step, $\tilde{s}_t = (s_t, t)$, which deterministically increments. Since the time step is now part of this augmented state, the value

Termination at $t = H$ may simply be defined as an absorbing state.

function and policy may depend on t. Rather than actually use the augmented state notation, we will write V_t^π and π_t to make this dependence explicit. For simplicity we also take $\gamma = 1$ in the fixed horizon setting.

In the fixed horizon case, the *value function* simplifies to

$$V_t^\pi(s) := \mathbb{E}_{\tau\sim\rho^\pi}[r_t + \cdots + r_{H-1} \mid s_t = s]\,.$$

The value function still satisfies a recursive equation; the Bellman consistency equation now relates the value at time step t to the value at time step $t+1$ for all $s \in \mathcal{S},\ t = 0, \ldots, H-1$

$$V_t^\pi(s) = \mathbb{E}_{\substack{a\sim\pi_t(s)\\ s'\sim P(s,a)}}[\mathbb{E}_{r\sim R(s,a)}[r] + V_{t+1}^\pi(s')]$$

In the indefinite-horizon case, the Bellman consistency equation held for a single, time-independent value function V^π, appearing on both sides of the equation. In contrast, the fixed horizon consistency equation expresses V_t^π in terms of V_{t+1}^π. At the final time step, no reward remains to be collected, so $V_H^\pi(s) = 0$ for all s. This means we can compute the value function *exactly*: we simply start at the end of the horizon $t = H$, where the value is known, and work backwards in time, applying the Bellman consistency equation once at each time step to obtain $V_{H-1}^\pi, V_{H-2}^\pi, \ldots, V_0^\pi$ in turn. This technique is known as *dynamic programming*: solving a problem by working backwards from a known base case through a sequence of subproblems.

The term was coined by Bellman who wrote in his autobiography that he chose 'dynamic' because of the time-varying, multistage nature, but also because "it's impossible to use the word dynamic in a pejorative sense [...] something not even a Congressman could object to." The 1950s were a time when the federal government was wary of funding basic mathematical research.

Dynamic programming for policy evaluation is the following algorithm:

- $V_H(s) \leftarrow 0$ for all $s \in \mathcal{S}$
- for $t = H-1, \ldots, 0$:
  - for $s \in \mathcal{S}$:
    - $V_t(s) \leftarrow \sum_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} \pi_t(a \mid s)\, P(s' \mid s, a)\, [\mathbb{E}_{r\sim R(s,a)}[r] + V_{h+1}(s')]$

Unlike the general case, there is no approximation error. This procedure computes $V_0^\pi, \ldots, V_{H-1}^\pi$ exactly, using a finite number of arithmetic operations. Updating $V_t(s)$ for a single pair (h, s) requires summing over all $a \in \mathcal{A}$ and $s' \in \mathcal{S}$. Then, counting the loops, the total running time is $O(H \cdot |\mathcal{S}|^2 \cdot |\mathcal{A}|)$: linear in the horizon, and polynomial in the sizes of the state and action spaces.

2.6 *Action-value Function*

The value function $V^\pi(s)$ is defined in terms of the reward function R and transition function P. For now, we assume these are known,

and thus we can compute the value function exactly, iteratively, or using dynamic programming. Later on, we will encounter settings where R and P are *not* known in advance, and instead we must learn from data collected by interacting with the environment. In that setting, it will be useful to have a single function that packages together everything on the right-hand side of the Bellman equation. This motivates the *action-value function*, the expected total reward when starting in a given state, taking a given action, and following π thereafter:

The action-value function is also known as the Q-function

$$Q^\pi(s,a) := \mathbb{E}_{\tau\sim\rho^\pi}\left[r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \mid s_0 = s,\ a_0 = a\right]$$

Comparing this to the definition of V^π, the only difference is that the first action a is fixed, rather than drawn from $\pi(s)$. This immediately gives us a way to recover V^π from Q^π: averaging the action-value over actions drawn from the policy returns exactly the value function,

$$V^\pi(s) = \mathbb{E}_{a\sim\pi(s)}[Q^\pi(s,a)]\,.$$

Going back to the definition of Q^π, unrolling it one time step, and then using the Markov property lets us express the action-value function in terms of the value function at the next state:

$$Q^\pi(s,a) = \mathbb{E}_{r\sim R(s,a)}[r] + \gamma\,\mathbb{E}_{s'\sim P(s,a)}[V^\pi(s')]\,.$$

The intuition is the same as before: the total reward from taking action a in state s is the immediate reward $r(s,a)$, plus the discounted value of whatever comes next. Combining the two equations, we arrive at the Bellman consistency equation for action-values.

$$Q^\pi(s,a) = \mathbb{E}_{r\sim R(s,a)}[r] + \gamma\,\mathbb{E}_{\substack{s'\sim P(s,a)\\ a'\sim\pi(s')}}[Q^\pi(s',a')]$$

Just as we did with V^π, it is possible to define an action-value Bellman consistency operator and show that it is a contraction. Similarly, we may consider the special case of fixed horizons and $\gamma = 1$. We define the time-dependent action-value function analogously, as the expected reward-to-go from taking action a in state s at time step t:

$$\begin{aligned} Q_t^\pi(s,a) :&= \mathbb{E}_{\tau\sim\rho^\pi}[r_t + \cdots + r_{H-1} \mid s_t = s,\ a_t = a] \\ &= \mathbb{E}_{r\sim R(s,a)}[r] + \mathbb{E}_{s'\sim P(s,a)}[V_{t+1}^\pi(s')]\,. \end{aligned}$$

The value function is recovered by averaging over actions drawn from the policy at step t:

$$V_t^\pi(s) = \mathbb{E}_{a\sim\pi_t(s)}[Q_t^\pi(s,a)]\,.$$

The action-value Bellman consistency equation for fixed horizons is:

$$Q_t^\pi(s,a) = \mathbb{E}_{r\sim R(s,a)}[r] + \mathbb{E}_{\substack{s'\sim P(s,a)\\ a'\sim\pi_{t+1}(s')}}[Q_{t+1}^\pi(s',a')]$$

As with the value function, $Q_H^\pi(s,a) = 0$ for all s, a, since no reward remains to be collected after the horizon. Thus, we may apply the same backward dynamic programming procedure as for V_t^π.

The action-value function will play an important role in what follows: it is the natural object for comparing actions within a fixed state, for deriving improved policies, and for learning directly from data once R and P are no longer assumed known.