Chapter 2

Markov Decision Processes

Contents

2.1 Introduction

RL studies how an agent can learn to make sequential decisions in an environment. This is a very general problem! How can we *formalize* this task in a way that is both *sufficiently general* yet also tractable enough for *fruitful analysis*?

Let's consider some examples of sequential decision problems to identify the key common properties we'd like to capture:

- **Board games** like chess or Go, where the player takes turns with the opponent to make moves.
- **Video games** like Super Mario Bros, where the player can move around and interact with the environment.
- **Robotic control**, where the robot can move and interact with the environment.

All of these fit into the RL framework (consider what the agent, state, and possible reward signals are in each example). In particular, the *rules* of the environment stay the same over time. We can formalize such environments using **Markov decision processes** (MDPs).

2.1.1 Definition

MDPs are environments where the state transitions only depend on the most recent state and action. Formally, we say that the state transitions satisfy the **Markov property**, that is,

$$\mathbb{P}(s_{t+1} \mid (s_\tau, a_\tau)_{\tau=0}^t) = P(s_{t+1} \mid s_t, a_t).$$

We'll see that this simple assumption leads to a rich set of problems and algorithms.

MDPs are usually classified as **finite-horizon**, where the interactions end after some finite number of time steps, or **infinite-horizon**, where the interactions can continue indefinitely. We'll begin with the finite-horizon case for ease of analysis, but the ideas naturally extend to the infinite-horizon case.

Definition 2.1.1: (Finite-horizon) Markov Decision Process

The key components of a (finite-horizon) Markov decision process are:

1. The **state** that the agent interacts with. We use $\mathcal{S}$ to denote the set of possible states, called the **state space**.
2. The **actions** that the agent can take. We use $\mathcal{A}$ to denote the set of possible actions, called the **action space**.
3. Some **initial state distribution** $\mu \in \Delta(\mathcal{S})$.
4. The **state transitions** (a.k.a. **dynamics**) that describe what state we transition to after taking an action. We'll denote this by $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$.
5. The **reward** signal. In this course we'll take it to be a deterministic function on state-action pairs, i.e. $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$.
6. A time horizon $H \in \mathbb{N}$.

Combined together, these objects specify a finite-horizon Markov decision process:

$$M = (\mathcal{S}, \mathcal{A}, \mu, P, r, H).$$

We call the total reward the **return**:

$$G := R_0 + \cdots + R_{H-1},$$

where $R_t := r(S_t, A_t)$. The key *goal* in a reinforcement learning task is to *maximize expected return* $\mathbb{E}[G]$.

We'll also consider the **return-to-go**, the total reward from a given time step onwards:

$$G_t := R_t + \cdots + R_{H-1}.$$

Why can't we just maximize the current reward at each timestep, i.e. use a greedy strategy? Well, in RL as in real life, making greedy decisions (e.g. procrastinating) will often leave you worse off than if you make some short-term sacrifices for long-term gains.

We call the "video recording" of states, actions, and rewards a **trajectory**:

$$\tau = (s_t, a_t, r_t)_{t=0}^{H-1}$$

2.2 Policies and value functions

A **policy** describes the agent's strategy: which actions it takes in a given situation. One key goal of RL is to find the **optimal policy** that maximizes the expected return.

Policies can either be **deterministic** (in the same situation, the agent will always take the same action) or **stochastic** (in the same situation, the agent will sample an action from a distribution).

What do we mean by "situation"? In the most general setting, this could include all of the states, actions, and rewards in the trajectory so far. However, due to the Markov assumption, the state transitions only depend on the current state. Thus a **stationary** policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ — one that only depends on the current state — can do just as well.

In the finite-horizon case, we'll consider **time-dependent** policies that depend on the time step t as well, i.e. $\pi = \{\pi_0, \ldots, \pi_{H-1}\}$.

Once we've chosen a policy, we can sample trajectories by choosing actions according to the policy and observing the state transitions and rewards. We call this the **trajectory distribution**:

$$\begin{aligned} \mathbb{P}(\tau; \pi) &= \mu(s_0)\pi_0(a_0 \mid s_0)P(s_1 \mid s_0, a_0) \cdots \pi_{H-1}(a_{H-1} \mid s_{H-1}) \\ \text{where} \quad \tau &= (s_t, a_t, r_t)_{t=0}^{H-1}. \end{aligned}$$

We'll abuse notation and use $\tau \sim \pi$ to denote that τ is sampled from this distribution.

We'd like a concise way to refer to the expected return when *starting in a given state at a given time* and acting according to π. We call this the **value function** of π and denote it by

$$V_t^\pi(s) := \mathop{\mathbb{E}}_{\tau \sim \pi}[G_t \mid S_t = s]$$

Similarly, we can define the **action-value function** of π (aka the **Q-function**) as the expected return when starting in a given state and taking a given action:

$$Q_t^\pi(s, a) := \mathop{\mathbb{E}}_{\tau \sim \pi}[G_t \mid S_t = s, A_t = a]$$

2.2.1 Bellman self-consistency equations

Note that we can break down the return as

$$G_t = R_t + \gamma G_{t+1},$$

the reward from the *current time-step* plus the total reward from from *future time-steps*. It turns out that this simple observation, along with linearity of expectation, gives us a set of equations to solve for the value function analytically!

Let's expand out the definition of the value function to see what I mean. Let's first consider the simple case where $\pi : \mathcal{S} \to \mathcal{A}$ is deterministic:

$$\begin{aligned} V^\pi(s) &:= \mathop{\mathbb{E}}_{\pi}[G_0 \mid S_0 = s] \\ &= r(s, \pi(s)) + \gamma \mathop{\mathbb{E}}_{s' \sim P(\cdot \mid s, \pi(s))} \mathop{\mathbb{E}}_{\pi}[G_1 \mid S_1 = s'] \\ &= r(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, \pi(s)) V^\pi(s'). \end{aligned}$$

This is a set of $|\mathcal{S}|$ equations (one per state) in $|\mathcal{S}|$ unknowns (the value of each state), which we can solve for V^π.

For stochastic policies, we simply average out over the relevant quantities:

$$\begin{aligned} V^\pi(s) &:= \mathop{\mathbb{E}}_{\pi}[G_0 \mid S_0 = s] \\ &= \mathop{\mathbb{E}}_{\pi}[R_0 + \gamma G_1 \mid S_0 = s] \\ &= \mathop{\mathbb{E}}_{a \sim \pi(\cdot \mid s)} \left[r(s, a) + \gamma \mathop{\mathbb{E}}_{s' \sim P(\cdot \mid s, a)} \mathop{\mathbb{E}}_{\pi}[G_1 \mid S_1 = s'] \right] \\ &= \sum_a \pi(a \mid s) \left[r(s, a) + \gamma \sum_{s'} P(s' \mid s, a) V^\pi(s') \right]. \end{aligned}$$

These are called the **Bellman self-consistency equations**. They encapsulate that the value function's prediction of the current state must be consistent with its prediction of other states.

Can you write the Bellman self-consistency equations for the action-value function?

2.3 Tabular MDPs

When the state and action space are finite and small, we can think of the value function and Q-function as *lookup tables* with each cell corresponding to the value of a state (or state-action pair). We can neatly express quantities as vectors and matrices:

$$r \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|}, \quad P \in [0,1]^{(|\mathcal{S}|\times|\mathcal{A}|)\times|\mathcal{S}|}, \quad \rho \in [0,1]^{|\mathcal{S}|}, \quad \pi \in [0,1]^{|\mathcal{A}|\times|\mathcal{S}|}, \quad V^\pi \in \mathbb{R}^{|\mathcal{S}|}, \quad Q^\pi \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|}.$$

Make sure that these dimensions make sense!

Note that when the policy is deterministic, by definition, the actions can be determined from the state, and so we can chop off the action dimension in most cases:

$$r^\pi \in \mathbb{R}^{|\mathcal{S}|}, \quad P^\pi \in [0,1]^{|\mathcal{S}|\times|\mathcal{S}|}, \quad \rho \in [0,1]^{|\mathcal{S}|}, \quad \pi \in \mathcal{A}^{|\mathcal{S}|}, \quad V^\pi \in \mathbb{R}^{|\mathcal{S}|}.$$

Then, rewriting the system of Bellman equations in this notation gives

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

Note that we've assumed that $I - \gamma P^\pi$ is invertible. Can you see why this is the case?

Recall that a linear operator, i.e. a square matrix, is invertible if and only if its null space is trivial; that is, it doesn't map any nonzero vector to zero. In this case, we can see that $I - \gamma P^\pi$ is invertible because it maps any nonzero vector to a vector with at least one nonzero element.

2.4 Optimality

Theorem 2.4.1: Value Iteration

Initialize:

$$V^0 \sim \|V^0\|_\infty \in [0, 1/1-\gamma]$$

Iterate until convergence:

$$V^{t+1} \leftarrow \mathcal{J}(V^t)$$

Analysis

This algorithm runs in $O(|\mathcal{S}|^3)$ time since we need to perform a matrix inversion.

Theorem 2.4.2: Exact Policy Evaluation

Represent the reward from each state-action pair as a vector

$$R^\pi \in \mathbb{R}^{|\mathcal{S}|} \qquad R^\pi_s = r(s, \pi(s))$$

Also represent the state transitions

$$P^\pi \in \mathbb{R}^{|\mathcal{S}\times\mathcal{S}} \qquad P^\pi_{s,s'} = P(s'|s, \pi(s))$$

That is, row i of P^π is a distribution over the *next state* given that the current state is s_i and we choose an action using policy π.

Using this notation, we can express the Bellman consistency equation as

$$\begin{pmatrix} \vdots \\ V^\pi(s) \\ \vdots \end{pmatrix} = \begin{pmatrix} \vdots \\ r(s, \pi(s)) \\ \vdots \end{pmatrix} + \gamma \begin{pmatrix} \vdots \\ P(s' \mid s, \pi(s)) \\ \vdots \end{pmatrix} \begin{pmatrix} \vdots \\ V^\pi(s') \\ \vdots \end{pmatrix}$$

$$\begin{aligned} V^\pi &= R^\pi + \gamma P^\pi V^\pi \\ (I - \gamma P^\pi)V^\pi &= R^\pi \\ V^\pi &= (I - \gamma P^\pi)R^\pi \end{aligned}$$

if $I - \gamma P^\pi$ is invertible, which we can prove is the case.

Theorem 2.4.3: Iterative Policy Evaluation

How can we calculate the value function V^π of a policy π?

Above, we saw an exact function that runs in $O(|\mathcal{S}|^2)$. But say we really need a fast algorithm, and we're okay with having an approximate answer. Can we do better? Yes!

Using the same notation as above, let's initialize V^0 such that the elements are drawn uniformly from $[0, 1/(1-\gamma)]$.

Then we can iterate the fixed-point equation we found above:

$$V^{t+1} \leftarrow R + \gamma P V^t$$

How can we use this fast approximate algorithm?

Theorem 2.4.4: Policy Iteration

Remember, for now we're only considering policies that are *stationary and deterministic*. There's $|\mathcal{S}|^{|\mathcal{A}}$ of these, so let's start off by choosing one at random. Let's call this initial

policy π^0, using the superscript to indicate the time step.

Now for $t = 0, 1, \ldots,$ we perform the following:

1. *Policy Evaluation*: First use the algorithm from earlier to calculate $V^{\pi^t}(s)$ for all states s. Then use this to calculate the state-action values:

$$Q^{\pi^t}(s,a) = r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) V^{\pi^t}(s')$$

2. *Policy Improvement*: Update the policy so that, at each state, it chooses the action with the highest action-value:

$$\pi^{t+1}(s) = \arg\max_a Q^{\pi^t}(s,a)$$

In other words, we're setting it to act greedily with respect to the new Q-function.

What's the computational complexity of this?

2.5 Finite Horizon MDPs

This is also called an *episodic model*.

Note that since our policy is nonstationary, we also need to adjust our value function (and Q-function) to account for this. Instead of considering the total infinite-horizon discounted reward like we did earlier, we'll instead consider the *remaining* reward from a given timestep onwards:

$$V_h^\pi(s) = \mathbb{E}\left[\sum_\tau^{H-1} r(s_\tau, a_\tau) \mid s_h = s, a_\tau = \pi_h(s_h)\right]$$
$$Q_h^\pi(s,a) = \mathbb{E}\left[\sum_\tau^{H-1} r(s_\tau, a_\tau) \mid (s_h, a_h) = (s,a)\right]$$

We can also define our Bellman consistency equations, by splitting up the total reward into the immediate reward (at this time step) and the future reward, represented by our state value function from that next time step:

$$Q_h^\pi(s,a) = r(s,a) + \mathop{\mathbb{E}}_{s' \sim P(s,a)}[V_{h+1}^\pi(s')]$$

Theorem 2.5.1: Computing the optimal policy

We can solve for the optimal policy using dynamic programming.

- *Base case.* At the end of the episode (time step $H-1$), we can't take any more actions, so the Q-function is simply the reward that we obtain:

  $$Q^\star_{H-1}(s,a) = r(s,a)$$

  so the best thing to do is just act greedily and get as much reward as we can!

  $$\pi^\star_{H-1}(s) = \arg\max_a Q^\star_{H-1}(s,a)$$

  Then $V^\star_{H-1}(s)$, the optimal value of state s at the end of the trajectory, is simply whatever action gives the most reward.

  $$V^\star_{H-1} = \max_a Q^\star_{H-1}(s,a)$$

- *Recursion.* Then, we can work backwards in time, starting from the end, using our consistency equations!

Note that this is exactly just value iteration and policy iteration combined, since our policy is nonstationary, so we can exactly specify its decisions at each time step!

Analysis

Total computation time $O(H|\mathcal{S}|^2|\mathcal{A}|)$