Chapter 2

Markov Decision Processes

How can we *formalize* a reinforcement learning task in a way that is both *sufficiently general* yet also tractable enough for *fruitful analysis*?

In this chapter, we'll turn to **Markov decision processes** as a simple yet general formalism for solving decision problems.

Definition 2.0.1: Markov Decision Process

The key components of a Markov decision process are:

1. The **state** (a.k.a. the **environment**) that the agent interacts with. We use $\mathcal{S}$ to denote the set of possible states, called the **state space**.
2. The **agent** and the **actions** that it can take. We use $\mathcal{A}$ to denote the set of possible actions, called the **action space**.
3. The **reward** signal. In this course we'll take it to be a deterministic function of a state-action pair, i.e. $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. In general, though, the reward function can also be stochastic, and it can also accept the *resulting* state as an argument; that is, $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \Delta(\mathbb{R})$.
4. The **state transitions** (a.k.a. **dynamics**) that describe what state we **transition to** after taking an action. We'll denote this by $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ (as opposed to $\mathbb{P}$ which denotes the underlying probability measure.)
5. A *discount factor* $\gamma \in [0, 1)$. We'll see later that this ensures that the *return*, or total reward, is well-defined in infinite-horizon problems.
6. Some **initial state distribution** $\rho \in \Delta(\mathcal{S})$.

Combined together, we call these a Markov decision process

$$M = (\mathcal{S}, \mathcal{A}, P, r, \gamma, \rho).$$

Add section "What is R Agent takin actions that pact the env ronment. In duce Δ nota tion. We'll overload not tion and use both to repr sent the act state $s_t \in \mathcal{S}$ to represent *event* that v observe stat at time t. M notation (e. uppercase le ters for rand variables) is rowed from ton and Bar

The reason we call it a *Markov* decision process is that the transition function only depends on the "current" state and action. Formally, this implies that the state process satisfies the **Markov property**, that is,

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

Example 2.0.1: Examples of MDPs

Board games and video games are often MDPs. For example, in chess or Go, the state of the game only depends on the pieces on the board and not on the previous history. Several possible reward functions could be possible, e.g. +1 upon winning the game and 0 otherwise, or to receive reward upon taking the opponent's pieces. The state transitions are based on the opponent's moves.

Robotic control can be framed as an MDP task. In this setting, physics provides the state transitions. A possible action might be activating a motor to move forwards. The reward function could be designed based on the task; for example, one could reward the robot for arriving at a desired location.

We'll distinguish between **finite-horizon** MDPs, where the agent eventually enters a **terminal state**, and **infinite-horizon** MDPs, where the agent might keep going on and on.

We call the total reward the *return*. For finite-horizon MDPs, we can just add up the rewards:

$$G_t := R_t + R_{t+1} + \cdots + R_T,$$

where T is the number of time steps and $R_t := r(S_t, A_t)$. However, for infinite-horizon problems (i.e. $T = \infty$), in order for this to be well-defined, we need to *discount* future rewards:

$$G_t := R_t + \gamma R_{t+1} + \cdots + \gamma^{\tau-t} R_\tau + \cdots = \sum_{\tau=t}^{\infty} \gamma^{\tau-t} R_\tau.$$

Can you see why this ensures that G_t is finite?

Note that we recover the finite-horizon definition by letting $\gamma = 1$ and T be finite.

Our key *goal* in a reinforcement learning task is to *maximize expected return*.

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, often making greedy decisions (e.g. procrastinating) will 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**

$$\xi_t = (s_\tau, a_\tau, r_\tau)_{\tau=0}^t$$

2.1 Policies and value functions

A **policy** π describes the agent's strategy: which actions it takes in a given situation.

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 I 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.

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

$$V^\pi(s) := \mathop{\mathbb{E}}_{\pi}[G_0 \mid S_0 = s]$$

We start at time 0 without loss of generality; can you see why we could have chosen to start at any time?

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:

$$Q^\pi(s, a) := \mathop{\mathbb{E}}_{\pi}[G_0 \mid S_0 = s, A_0 = a]$$

2.1.1 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* and that from *future time-steps*. It turns out that this simple observation gives us a way 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:

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

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

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

2.2 Finite MDPs

When the state and action space are finite, we can neatly express quantities as vectors and matrices:

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

2.3 Exercises

Show that without discounting, the reward

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

Suppose we're only able to act for H timesteps.

Now, instead of discounting, all we care about is the (average) total reward that we get over this time.

$$\mathbb{E}[\sum_{t=0}^{H-1} r(s_t, a_t)]$$

To be more precise, we'll consider policies that depend on the time. We'll denote the policy at timestep h as $\pi_h : \mathcal{S} \to \mathcal{A}$. In other words, we're dropping the constraint that policies must be stationary.

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:

  $$Q_{H-1}^\star(s,a) = r(s,a)$$

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

  $$\pi_{H-1}^\star(s) = \arg\max_a Q_{H-1}^\star(s,a)$$

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

  $$V_{H-1}^\star = \max_a Q_{H-1}^\star(s,a)$$

- *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 $O(H|\mathcal{S}|^2|\mathcal{A}|)$