CHAPTER 11

Markov Decision Processes

So far, most of the learning problems we have looked at have been *supervised*, that is, for each training input $x^{(i)}$, we are told which value $y^{(i)}$ should be the output. From a traditional machine-learning viewpoint, there're two other major groups of learning problems: one is the *unsupervised* learning problems, in which we are given data and no expected outputs, and we will look at later in Chapter 7 and Chapter 10.

The other major type is the so-called *Reinforcement learning* (RL) problems. Reinforcement learning differs significantly from supervised learning problems, and we will delve into the details later in in Chapter 12. However, it's worth pointing out one major difference at a very high level: in supervised learning, our goal is to learn a one-time static mapping to make predictions, whereas in RL, the setup requires us to sequentially take actions to maximize cumulative rewards.

This setup change necessitates additional mathematical and algorithmic tools for us to understand RL. *Markov decision process* (MDP) is precisely such a classical and fundamental tool.

11.1 Definition and value functions

Formally, a Markov decision process is $\langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, and:

- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is a *transition model*, where

$$T(s, a, s') = \Pr(S_t = s' | S_{t-1} = s, A_{t-1} = a) \ ,$$

  specifying a conditional probability distribution;

- $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a *reward function*, where $R(s, a)$ specifies an immediate reward for taking action a when in state s; and
- $\gamma \in [0, 1]$ is a *discount factor*, which we'll discuss in Section 11.1.2.

In this class, we assume the rewards are deterministic functions. Further, in this MDP chapter, we assume the state space and action space are finite (in fact, typically small).

The notation $S_t = s'$ uses a capital letter S to stand for a random variable, and small letter s to stand for a concrete value. So S_t here is a random variable that can take on elements of $\mathcal{S}$ as values.

The following description of a simple machine as Markov decision process provides a concrete example of an MDP. The machine has three possible operations (*actions*): "wash", "paint", and "eject" (each with a corresponding button). Objects are put into the machine. Each time you push a button, something is done to the object. However, it's an old machine, so it's not very reliable. The machine has a camera inside that can clearly detect what is going on with the object and will output the state of the object: "dirty", "clean", "painted", or "ejected". For each action, this is what is done to the object:

Wash:

- If you perform the "wash" operation on any object, whether it's dirty, clean, or painted, it will end up "clean" with probability 0.9 and "dirty" otherwise.

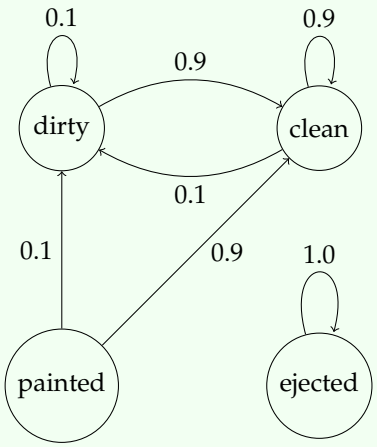
Paint:

- If you perform the "paint" operation on a clean object, it will become nicely "painted" with probability 0.8. With probability 0.1, the paint misses but the object stays clean, and also with probability 0.1, the machine dumps rusty dust all over the object and it becomes "dirty".
- If you perform the "paint" operation on a "painted" object, it stays "painted" with probability 1.0.
- If you perform the "paint" operation on a "dirty" part, it stays "dirty" with probability 1.0.

Eject:

- If you perform an "eject" operation on any part, the part comes out of the machine and this fun game is over. The part remains "ejected" regardless of any further action.

These descriptions specify the transition model T, and the transition function for each action can be depicted as a state machine diagram. For example, here is the diagram for "wash":



You get reward +10 for ejecting a painted object, reward 0 for ejecting a non-painted object, reward 0 for any action on an "ejected" object, and reward -3 otherwise. The MDP description would be completed by also specifying a discount factor.

A *policy* is a function π that specifies what action to take in each state. The policy is what we will want to learn; it is akin to the strategy that a player employs to win a given game. Below, we take just the initial steps towards this eventual goal. We describe how to evaluate how good a policy is, first in the *finite horizon* case (Section 11.1.1) when the total number of transition steps is finite. In the finite horizon case, we typically denote the policy as $\pi_h(s)$, where $h \in \mathbb{N}$ is a non-negative integer denoting the number of steps remaining and $s \in \mathcal{S}$ is the current state. Then we consider the *infinite horizon* case (Section 11.1.2), when you don't know when the game will be over.

11.1.1 Finite-horizon value functions

The goal of a policy is to maximize the expected total reward, averaged over the stochastic transitions that the domain makes. Let's first consider the case where there is a finite *horizon* H, indicating the total number of steps of interaction that the agent will have with the MDP.

In the finite-horizon case, we usually set the discount factor γ to 1.

We seek to measure the goodness of a policy. We do so by defining for a given horizon h and MDP policy π_h, the "horizon h *value*" of a state, $V_\pi^h(s)$. We do this by induction on the horizon, which is the *number of steps left to go*.

The base case is when there are no steps remaining, in which case, no matter what state we're in, the value is 0, so

$$V_\pi^0(s) = 0 \ . \tag{11.1}$$

Then, the value of a policy in state s at horizon $h+1$ is equal to the reward it will get in state s plus the next state's expected horizon h value, discounted by a factor γ. So, starting with horizons 1 and 2, and then moving to the general case, we have:

$$V_\pi^1(s) = R(s, \pi_1(s)) + 0 \tag{11.2}$$

$$V_\pi^2(s) = R(s, \pi_2(s)) + \gamma \sum_{s'} T(s, \pi_2(s), s') V_\pi^1(s') \tag{11.3}$$

$$\vdots$$

$$V_\pi^h(s) = R(s, \pi_h(s)) + \gamma \sum_{s'} T(s, \pi_h(s), s') V_\pi^{h-1}(s') \tag{11.4}$$

The sum over s' is an *expectation*: it considers all possible next states s', and computes an average of their $(h-1)$-horizon values, weighted by the probability that the transition function from state s with the action chosen by the policy $\pi_h(s)$ assigns to arriving in state s', and discounted by γ.

Study Question: What is $\sum_{s'} T(s, a, s')$ for any particular s and a?

Study Question: Convince yourself that Eqs. 11.1 and 11.3 are special cases of Eq. 11.4.

Then we can say that a policy π is better than policy $\bar{\pi}$ for horizon h if and only if for all $s \in \mathcal{S}$, $V_\pi^h(s) \geqslant V_{\bar{\pi}}^h(s)$ and there exists at least one $s \in \mathcal{S}$ such that $V_\pi^h(s) > V_{\bar{\pi}}^h(s)$.

11.1.2 Infinite-horizon value functions

More typically, the actual finite horizon is not known, i.e., when you don't know when the game will be over! This is called the *infinite horizon* version of the problem. How does one evaluate the goodness of a policy in the infinite horizon case?

If we tried to simply take our definitions above and use them for an infinite horizon, we could get in trouble. Imagine we get a reward of 1 at each step under one policy and a reward of 2 at each step under a different policy. Then the reward as the number of steps

grows in each case keeps growing to become infinite in the limit of more and more steps. Even though it seems intuitive that the second policy should be better, we can't justify that by saying $\infty < \infty$.

One standard approach to deal with this problem is to consider the *discounted* infinite horizon. We will generalize from the finite-horizon case by adding a discount factor.

In the finite-horizon case, we valued a policy based on an expected finite-horizon value:

$$\mathbb{E}\left[\sum_{t=0}^{h-1} \gamma^t R_t \mid \pi, s_0\right] \quad , \tag{11.5}$$

where R_t is the reward received at time t.

What is $\mathbb{E}[\cdot]$? This mathematical notation indicates an *expectation*, i.e., an average taken over all the random possibilities which may occur for the argument. Here, the expectation is taken over the *conditional probability* $\Pr(R_t = r \mid \pi, s_0)$, where R_t is the random variable for the reward, subject to the policy being π and the state being s_0. Since π is a function, this notation is shorthand for conditioning on all of the random variables implied by policy π and the stochastic transitions of the MDP.

A very important point is that $R(s, a)$ is always deterministic (in this class) for any given s and a. Here R_t represents the set of all possible $R(s_t, a)$ at time step t; this R_t is a random variable because the state we're in at step t is itself a random variable, due to prior stochastic state transitions up to but not including at step t and prior (deterministic) actions dictated by policy π.

Now, for the infinite-horizon case, we select a discount factor $0 \leqslant \gamma \leqslant 1$, and evaluate a policy based on its expected *infinite horizon discounted value*:

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t \mid \pi, s_0\right] = \mathbb{E}\left[R_0 + \gamma R_1 + \gamma^2 R_2 + \ldots \mid \pi, s_0\right] \quad . \tag{11.6}$$

Note that the t indices here are not the number of steps to go, but actually the number of steps forward from the starting state (there is no sensible notion of "steps to go" in the infinite horizon case).

Eqs. 11.5 and 11.6 are a conceptual stepping stone. Our main objective is to get to Eq. 11.8, which can also be viewed as including γ in Eq. 11.4, with the appropriate definition of the infinite-horizon value.

There are two good intuitive motivations for discounting. One is related to economic theory and the present value of money: you'd generally rather have some money today than that same amount of money next week (because you could use it now or invest it). The other is to think of the whole process terminating, with probability $1-\gamma$ on each step of the interaction. This value is the expected amount of reward the agent would gain under this terminating model.

At every step, your expected future lifetime, given that you have survived until now, is $1/(1-\gamma)$.

Study Question: Verify this fact: if, on every day you wake up, there is a probability of $1-\gamma$ that today will be your last day, then your expected lifetime is $1/(1-\gamma)$ days.

Let us now evaluate a policy in terms of the expected discounted infinite-horizon value that the agent will get in the MDP if it executes that policy. We define the infinite-horizon value of a state s under policy π as

$$V_\pi(s) = \mathbb{E}[R_0 + \gamma R_1 + \gamma^2 R_2 + \cdots \mid \pi, S_0 = s] = \mathbb{E}[R_0 + \gamma(R_1 + \gamma(R_2 + \gamma \ldots))) \mid \pi, S_0 = s] \quad . \tag{11.7}$$

Because the expectation of a linear combination of random variables is the linear combination of the expectations, we have

$$\begin{aligned} V_\pi(s) &= \mathbb{E}[R_0 \mid \pi, S_0 = s] + \gamma \mathbb{E}[R_1 + \gamma(R_2 + \gamma \ldots))) \mid \pi, S_0 = s] \\ &= R(s, \pi(s)) + \gamma \sum_{s'} T(s, \pi(s), s') V_\pi(s') \; . \end{aligned} \tag{11.8}$$

The equation defined in Eq. 11.8 is known as the Bellman Equation, which breaks down the value function into the immediate reward and the (discounted) future value function. You could write down one of these equations for each of the $n = |\mathcal{S}|$ states. There are n unknowns $V_\pi(s)$. These are linear equations, and standard software (e.g., using Gaussian elimination or other linear algebraic methods) will, in most cases, enable us to find the value of each state under this policy.

This is *so* cool! In a discounted model, if you find that you survived this round and landed in some state s', then you have the same expected future lifetime as you did before. So the value function that is relevant in that state is exactly the same one as in state s.

11.2 Finding policies for MDPs

Given an MDP, our goal is typically to find a policy that is optimal in the sense that it gets as much total reward as possible, in expectation over the stochastic transitions that the domain makes. We build on what we have learned about evaluating the goodness of a policy (Sections 11.1.1 and 11.1.2), and find optimal policies for the finite horizon case (Section 11.2.1), then the infinite horizon case (Section 11.2.2).

11.2.1 Finding optimal finite-horizon policies

How can we go about finding an optimal policy for an MDP? We could imagine enumerating all possible policies and calculating their value functions as in the previous section and picking the best one – but that's too much work!

The first observation to make is that, in a finite-horizon problem, the best action to take depends on the current state, but also on the horizon: imagine that you are in a situation where you could reach a state with reward 5 in one step or a state with reward 100 in two steps. If you have at least two steps to go, then you'd move toward the reward 100 state, but if you only have one step left to go, you should go in the direction that will allow you to gain 5!

One way to find an optimal policy is to compute an optimal *action-value function*, Q. For the finite-horizon case, we define $Q^h(s, a)$ to be the expected value of

- starting in state s,
- executing action a, and
- continuing for $h-1$ more steps executing an optimal policy for the appropriate horizon on each step.

Similar to our definition of V^h for evaluating a policy, we define the Q^h function recursively according to the horizon. The only difference is that, on each step with horizon h, rather than selecting an action specified by a given policy, we select the value of a that will

maximize the expected Q^h value of the next state.

$$Q^0(s,a) = 0 \tag{11.9}$$

$$Q^1(s,a) = R(s,a) + 0 \tag{11.10}$$

$$Q^2(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q^1(s',a') \tag{11.11}$$

$$\vdots$$

$$Q^h(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q^{h-1}(s',a') \tag{11.12}$$

where (s',a') denotes the next time-step state/action pair. We can solve for the values of Q^h with a simple recursive algorithm called *finite-horizon value iteration* that just computes Q^h starting from horizon 0 and working backward to the desired horizon H. Given Q^h, an optimal π^*_h can be found as follows:

$$\pi^*_h(s) = \arg\max_a Q^h(s,a) \quad . \tag{11.13}$$

which gives the *immediate* best action(s) to take when there are h steps left; then $\pi^*_{h-1}(s)$ gives the best action(s) when there are $(h-1)$ steps left, and so on. In the case where there are multiple best actions, we typically can break ties randomly.

Additionally, it is worth noting that in order for such an optimal policy to be computed, we assume that the reward function $R(s,a)$ is bounded on the set of all possible (state, action) pairs. Furthermore, we will assume that the set of all possible actions is finite.

Study Question: The optimal value function is unique, but the optimal policy is not. Think of a situation in which there is more than one optimal policy.

Dynamic programming (somewhat counter-intuitively, dynamic programming is neither really "dynamic" nor a type of "programming" as we typically understand it) is a technique for designing efficient algorithms. Most methods for solving MDPs or computing value functions rely on dynamic programming to be efficient. The *principle of dynamic programming* is to compute and store the solutions to simple sub-problems that can be re-used later in the computation. It is a very important tool in our algorithmic toolbox.
Let's consider what would happen if we tried to compute $Q^4(s, a)$ for all (s, a) by directly using the definition:

- To compute $Q^4(s_i, a_j)$ for any one (s_i, a_j), we would need to compute $Q^3(s, a)$ for all (s, a) pairs.
- To compute $Q^3(s_i, a_j)$ for any one (s_i, a_j), we'd need to compute $Q^2(s, a)$ for all (s, a) pairs.
- To compute $Q^2(s_i, a_j)$ for any one (s_i, a_j), we'd need to compute $Q^1(s, a)$ for all (s, a) pairs.
- Luckily, those are just our $R(s, a)$ values.

So, if we have n states and m actions, this is $O((mn)^3)$ work — that seems like way too much, especially as the horizon increases! But observe that we really only have mnh values that need to be computed: $Q^h(s, a)$ for all h, s, a. If we start with $h = 1$, compute and store those values, then using and reusing the $Q^{h-1}(s, a)$ values to compute the $Q^h(s, a)$ values, we can do all this computation in time $O(mnh)$, which is much better!

11.2.2 Finding optimal infinite-horizon policies

In contrast to the finite-horizon case, the best way of behaving in an infinite-horizon discounted MDP is not time-dependent. That is, the decisions you make at time $t = 0$ looking forward to infinity, will be the same decisions that you make at time $t = T$ for any positive T, also looking forward to infinity.

An important theorem about MDPs is: in the infinite-horizon case, there exists a stationary optimal policy π^* (there may be more than one) such that for all $s \in S$ and all other policies π, we have

$$V_{\pi^*}(s) \geqslant V_\pi(s) \quad . \tag{11.14}$$

Stationary means that it doesn't change over time; in contrast, the optimal policy in a finite-horizon MDP is *non-stationary.*

There are many methods for finding an optimal policy for an MDP. We have already seen the finite-horizon value iteration case. Here we will study a very popular and useful method for the infinite-horizon case, *infinite-horizon value iteration*. It is also important to us, because it is the basis of many *reinforcement-learning* methods.

We will again assume that the reward function $R(s, a)$ is bounded on the set of all possible (state, action) pairs and additionally that the number of actions in the action space is finite. Define $Q(s, a)$ to be the expected infinite-horizon discounted value of being in state s, executing action a, and executing an optimal policy π^* thereafter. Using similar reasoning to the recursive definition of V_π, we can express this value recursively as

$$Q(s, a) = R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q(s', a') \quad . \tag{11.15}$$

This is also a set of equations, one for each (s, a) pair. This time, though, they are not linear (due to the max operation), and so they are not easy to solve. But there is a theorem

that says they have a unique solution!

Once we know the optimal action-value function, then we can extract an optimal policy π^* as

$$\pi^*(s) = \arg\max_a Q(s,a) \quad . \tag{11.16}$$

As in the finite-horizon case, there may be more than one optimal policy in the infinite-horizon case.

We can iteratively solve for the Q^* values with the infinite-horizon value iteration algorithm, shown below:

INFINITE-HORIZON-VALUE-ITERATION$(\mathcal{S}, \mathcal{A}, T, R, \gamma, \epsilon)$

```
for s ∈ S, a ∈ A :
    Q_old(s, a) = 0
while not converged:
    for s ∈ S, a ∈ A :
        Q_new(s, a) = R(s, a) + γ Σ_s' T(s, a, s') max_a' Q_old(s', a')
    if max_{s,a} |Q_old(s, a) − Q_new(s, a)| < ε :
        return Q_new
    Q_old = Q_new
```

Theory There are a lot of nice theoretical results about infinite-horizon value iteration. For some given (not necessarily optimal) Q function, define $\pi_Q(s) = \arg\max_a Q(s,a)$.

- After executing infinite-horizon value iteration with convergence hyper-parameter ϵ,

$$\|V_{\pi_{Q_{new}}} - V_{\pi^*}\|_{max} < \epsilon \quad . \tag{11.17}$$

- There is a value of ϵ such that

$$\|Q_{old} - Q_{new}\|_{max} < \epsilon \implies \pi_{Q_{new}} = \pi^* \tag{11.18}$$

Note the new notation! Given two functions f and f', we write $\|f - f'\|_{max}$ to mean $\max_x |f(x) - f'(x)|$. It measures the maximum absolute disagreement between the two functions at any input x.

- As the algorithm executes, $\|V_{\pi_{Q_{new}}} - V_{\pi^*}\|_{max}$ decreases monotonically on each iteration.
- The algorithm can be executed asynchronously, in parallel: as long as all (s,a) pairs are updated infinitely often in an infinite run, it still converges to the optimal value.

This is very important for reinforcement learning.