UNIT-III

Finite Markov Decision Processes: The Agent–Environment Interface, Goals and Rewards, Returns, Unified Notation for Episodic and Continuing Tasks, The Markov Property, Markov Decision Processes, Value Functions, Optimal Value Functions, Optimality and Approximation.

1. Finite Markov Decision Processes: The Agent–Environment Interface

- MDPs are meant to be a straightforward framing of the problem of learning from interaction to achieve a goal.
- The learner and decision maker are called the agent.
- The thing it interacts with, comprising everything outside the agent, is called the environment.
- These interact continually, the agent selecting actions and the environment responding to these actions and presenting new situations to the agent.
- The environment also gives rise to rewards, special numerical values that the agent seeks to maximize over time through its choice of actions.

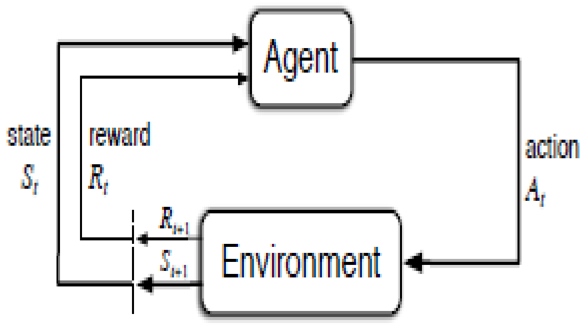


Figure 3.1: The agent–environment interaction in a Markov decision process.

- More specifically, the agent and environment interact at each of a sequence of discrete time steps, t = 0, 1, 2, 3....[2] At each time step t, the agent receives some representation of the environment's state, $S_t \in \mathcal{S}$, and on that basis selects an action, $A_t \in \mathcal{A}(s)$. One time step later, 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 MDP and agent together thereby give rise to a sequence or trajectory that begins like this:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \dots \tag{3.1}$$

- In a finite MDP, the sets of states, actions, and rewards (S, A, and R) all have a finite number of elements.
- In this case, the random variables R_t and St have well defined discrete probability distributions dependent only on the preceding state and action.
- That is, for particular values of these random variables, $s' \in \mathcal{S}$ and $r \in \mathcal{R}$, there is a probability of those values occurring at time t, given particular values of the preceding state and action:

$$p(s', r \mid s, a) \doteq \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\}, \qquad (3.2)$$

for all $s', s \in \mathcal{S}, r \in \mathcal{R}, \text{ and } a \in \mathcal{A}(s)$.

- The function p defines the dynamics of the MDP. The dot over the equals sign in the equation reminds us that it is a definition (in this case of the function p) rather than a fact that follows from previous definitions. The dynamics function $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ is an ordinary deterministic function of four arguments.
- The '|' in the middle of it comes from the notation for conditional probability, but here it just reminds us that p specifies a probability distribution for each choice of s and a, that is, that

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a) = 1, \text{ for all } s \in \mathcal{S}, a \in \mathcal{A}(s). \qquad (3.3)$$

- In a Markov decision process, the probabilities given by p completely characterize the environment's dynamics.
- That is, the probability of each possible value for St and Rt depends only on the immediately preceding state and action, S_t and R_t depends only on the immediately preceding state and action, S_{t-1} and A_{t-1}, and given them, not at all on earlier states and actions.
- This is best viewed a restriction not on the decision process, but on the state.
- The state must include information about all aspects of the past agent–environment interaction that make a difference for the future.
- We consider how a Markov state can be learned and constructed from non-Markov observations from the four-argument dynamics function, p, one can compute anything else one might want to know about the environment, such as the state-transition probabilities (which we denote, with a slight abuse of notation, as a three-argument function $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$),

$$p(s'|s,a) \doteq \Pr\{S_t=s' \mid S_{t-1}=s, A_{t-1}=a\} = \sum_{r\in\mathcal{R}} p(s',r|s,a). \tag{3.4}$$

We can also compute the expected rewards for state–action pairs as a two-argument function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$:

$$r(s,a) \doteq \mathbb{E}[R_t \mid S_{t-1}=s, A_{t-1}=a] = \sum_{r\in\mathcal{R}} r \sum_{s'\in\mathcal{S}} p(s',r|s,a), \tag{3.5}$$

and the expected rewards for state–action–next-state triples as a three-argument function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$,

$$r(s,a,s') \doteq \mathbb{E}[R_t \mid S_{t-1}=s, A_{t-1}=a, S_t=s'] = \sum_{r\in\mathcal{R}} r \frac{p(s',r|s,a)}{p(s'|s,a)}. \tag{3.6}$$

Example 3.1: Bioreactor Suppose reinforcement learning is being applied to determine moment-by-moment temperatures and stirring rates for a bio refactor (a large vat of nutrients and bacteria used to produce useful chemicals). The actions in such an application might be target temperatures and target stirring rates that are passed to lower-level control systems that, in turn, differently activate heating elements and motors to attain the targets. The states are likely to be thermocouple and other sensory readings, perhaps filtered and delayed, plus symbolic inputs representing the ingredients in the vat and the target chemical. The rewards might be moment-by-moment measures of the rate at which the useful chemical is produced by the bioreactor. Notice that here each state is a list, or vector, of sensor readings and symbolic inputs, and each action is a vector consisting of a target temperature and a stirring rate. It is typical of reinforcement learning tasks to have states and actions with such structured representations. Rewards, on the other hand, are always single numbers.

Example 3.2: Pick-and-Place Robot Consider using reinforcement learning to control the motion of a robot arm in a repetitive pick-and-place task. If we want to learn movements that are fast and smooth, the learning agent will have to control the motors directly and have low-latency information about the current positions and velocities of the mechanical linkages. The actions in this case might be the voltages applied to each motor at each joint, and the states might be the latest readings of joint angles and velocities. The reward might be +1 for each object successfully picked up and placed. To encourage smooth movements, on each time step a small, negative reward can be given as a function of the moment-to-moment "jerkiness" of the motion.

Example 3.3: Recycling Robot A mobile robot has the job of collecting empty soda cans in an office environment. It has sensors for detecting cans, and an arm and gripper that can pick them up and place them in an onboard bin; it runs on a rechargeable battery. The robot's control system has components for interpreting sensory information, for navigating, and for controlling the arm and gripper. High-level decisions about how to search for cans are made by a reinforcement learning agent based on the current charge level of the battery. This agent has to decide whether the robot should (1) actively search for a can for a certain period of time, (2) remain stationary and wait for someone to bring it a can, or (3) head back to its home base to recharge its battery. This decision has to be made either periodically or whenever certain events occur, such as finding an empty can. The agent therefore has three actions, and its state is determined by the state of the battery. The rewards might be zero most of the time, but then become positive when the robot secures an empty can, or large and negative if the battery runs all the way down. In this example, the reinforcement learning agent is not the entire robot. The states it monitors describe conditions within the robot itself, not conditions of the robot's external environment. The agent's environment therefore includes the rest of the robot, which might contain other complex decision-making systems, as well as the robot's external environment.

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.
- At each time step, the reward is a simple number, $R_t \in \mathbb{R}$. Informally, the agent's goal is to maximize the total amount of reward it receives.
- This means maximizing not immediate reward, but cumulative reward in the long run. 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.
- Although formulating goals in terms of reward signals might at first appear limiting, in practice it has proved to be flexible and widely applicable.
- The best way to see this is to consider examples of how it has been, or could be, used.
- For example, to make a robot learn to walk, researchers have provided reward on each time step proportional to the robot's forward motion.
- In making a robot learn how to escape from a maze, the reward is often -1 for every time step that passes prior to escape; this encourages the agent to escape as quickly as possible.
- To make a robot learn to find and collect empty soda cans for recycling, one might give it a reward of zero most of the time, and then a reward of +1 for each can collected.

- One might also want to give the robot negative rewards when it bumps into things or when somebody yells at it.
- For an agent to learn to play checkers or chess, the natural rewards are +1 for winning, -1 for losing, and 0 for drawing and for all nonterminal positions.

3. Returns and Episodes

- We have said that the agent's goal is to maximize the cumulative reward it receives in the long run. How might this be defined formally? If the sequence of rewards received after time step t is denoted $R_{t+1}, R_{t+2}, R_{t+3}, \ldots,$ then what precise aspect of this sequence do we wish to maximize? 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.
- In the simplest case the return is the sum of the rewards:

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T, \tag{3.7}$$

where T is a final time step.

- This approach makes sense in applications in which there is a natural notion of final time step, that is, when the agent–environment interaction breaks naturally into subsequences, which we call episodes, 7 such as plays of a game, trips through a maze, or any sort of repeated interaction.
- 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.
- Even if you think of episodes as ending in different ways, such as winning and losing a game, the next episode begins independently of how the previous one ended.
- Thus, the episodes can all be considered to end in the same terminal state, with different rewards for the different outcomes.
- 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 S, from the set of all states plus the terminal state, denoted S+.
- The time of termination, T, is a random variable that normally varies from episode to episode.
- On the other hand, in many cases the agent–environment 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 formulation (3.7) is problematic for continuing tasks because the final time step would be $T = \infty,$ and the return, which is what we are trying to maximize, could itself easily be infinite. (For example, suppose the agent receives a reward of +1

at each time step.) Thus, in this book we usually use a definition of return that is slightly more complex conceptually but much simpler mathematically.

- The additional concept that we need is that of discounting. According to this approach, the agent tries to select actions so that the sum of the discounted rewards it receives over the future is maximized. In particular, it chooses At to maximize the expected discounted return:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \qquad (3.8)$$

where γ is a parameter, $0 \leq \gamma \leq 1$, called the *discount rate*.

- The discount rate determines the present value of future rewards: a reward received k time steps in the future is worth only γ^{k-1} times what it would be worth if it were received immediately.
- If $\gamma < 1$, the infinite sum in (3.8) has a finite value as long as the reward sequence {R_k} is bounded. If $\gamma = 0$, the agent is "myopic" in being concerned only with maximizing immediate rewards: its objective in this case is to learn how to choose at so as to maximize only R_{t+1}.
- If each of the agent's actions happened to influence only the immediate reward, not future rewards as well, then a myopic agent could maximize (3.8) by separately maximizing each immediate reward.
- But in general, acting to maximize immediate reward can reduce access to future rewards so that the return is reduced. As γ approaches 1, the return objective takes future rewards into account more strongly; the agent becomes more farsighted.
- Returns at successive time steps are related to each other in a way that is important for the theory and algorithms of reinforcement learning:

$$\begin{aligned} G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma\left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots\right) \\ &= R_{t+1} + \gamma G_{t+1} \qquad (3.9) \end{aligned}$$

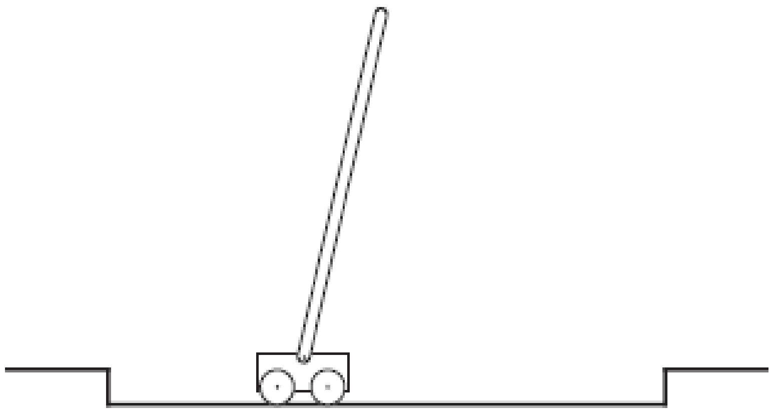
Note that this works for all time steps t < T even if termination occurs at t + 1, if we define GT = 0. This often makes it easy to compute returns from reward sequences.

Note that although the return (3.8) is a sum of an infinite number of terms, it is still finite if the reward is nonzero and constant—if $\gamma < 1$. For example, if the reward is a constant +1, then the return is

$$G_t = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma}. \qquad (3.10)$$

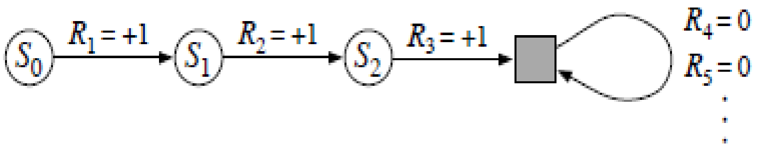
<u>**Example 3.4: Pole-Balancing**</u>

- The objective in this task is to apply forces to a cart moving along a track so as to keep a pole hinged to the cart from falling over: A failure is said to occur if the pole falls past a given angle from vertical or if the cart runs of the track.
- The pole is reset to vertical after each failure.
- This task could be treated as episodic, where the natural episodes are the repeated attempts to balance the pole.
- The reward in this case could be +1 for every time step on which failure did not occur, so that the return at each time would be the number of steps until failure.
- In this case, successful balancing forever would mean a return of infinity.
- Alternatively, we could treat pole-balancing as a continuing task, using discounting.
- In this case the reward would be -1 on each failure and zero at all other times.
- The return at each time would then be related to $-\gamma^K$, where K is the number of time steps before failure. In either case, the return is maximized by keeping the pole balanced for as long as possible.



4. Unified Notation for Episodic and Continuing Tasks

- To be precise about episodic tasks requires some additional notation. Rather than one long sequence of time steps, we need to consider a series of episodes, each of which consists of a finite sequence of time steps.
- We number the time steps of each episode starting anew from zero. Therefore, we have to refer not just to S_t, the state representation at time t, but to $S_{t,i}$, the state representation at time t of episode i (and similarly for $A_{t,i}$, $R_{t,i}$, $\pi_{t,i}$, T_i, etc.).
- However, it turns out that when we discuss episodic tasks we almost never have to distinguish between different episodes.
- We are almost always considering a particular single episode, or stating something that is true for all episodes.
- Accordingly, in practice we almost always abuse notation slightly by dropping the explicit reference to episode number. That is, we write S_t to refer to $S_{t,i}$, and so on.
- We need one other convention to obtain a single notation that covers both episodic and continuing tasks.
- We have defined the return as a sum over a finite number of terms in one case (3.7) and as a sum over an infinite number of terms in the other (3.8). These two can be unified by considering episode termination to be the entering of a special absorbing state those transitions only to itself and that generates only rewards of zero.
- For example, consider the state transition diagram:



- Here the solid square represents the special absorbing state corresponding to the end of an episode.
- Starting from S_0, we get the reward sequence +1, +1, +1, 0, 0, 0.... Summing these, we get the same return whether we sum over the first T rewards (here T = 3) or over the full infinite sequence.
- This remains true even if we introduce discounting.
- Thus, we can define the return, in general, according to (3.8), using the convention of omitting episode numbers when they are not needed, and including the possibility that $\gamma = 1$ if the sum remains defined (e.g., because all episodes terminate).
- Alternatively, we can write

$$G_t \doteq \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k, \qquad (3.11)$$

- Including the possibility that $T = \infty$ or $\gamma = 1$. We use these conventions throughout the rest of the book to simplify notation and to express the close parallels between episodic and continuing tasks.

5. The Markov Property

- In the reinforcement learning framework, the agent makes its decisions as a function of a signal from the environment called the environment's state.
- In particular, we formally define a property of environments and their state signals that is of particular interest, called the Markov property.
- "The state" we mean whatever information is available to the agent. We assume that the state is given by some pre-processing system that is nominally part of the environment. We do not address the issues of constructing, changing, or learning the state signal.
- In other words, our main concern is not with designing the state signal, but with deciding what action to take as a function of whatever state signal is available.
- Certainly, the state signal should include immediate sensations such as sensory measurements, but it can contain much more than that.
- State representations can be highly processed versions of original sensations, or they can be complex structures built up over time from the sequence of sensations.
- For example, we can move our eyes over a scene, with only a tiny spot corresponding to the fovea visible in detail at any one time, yet build up a rich and detailed representation of a scene. Or, more obviously, we can look at an object, then look away, and know that it is still there.
- We can hear the word "yes" and consider ourselves to be in totally different states depending on the question that came before and which is no longer audible.
- At a more mundane level, a control system can measure position at two different times to produce a state representation including information about velocity.
- In all of these cases the state is constructed and maintained on the basis of immediate sensations together with the previous state or some other memory of past sensations.
- On the other hand, the state signal should not be expected to inform the agent of everything about the environment, or even everything that would be useful to it in making decisions.
- If the agent is playing blackjack, we should not expect it to know what the next card in the deck is. If the agent is answering the phone, we should not expect it to know in advance who the caller is.
- If the agent is a paramedic called to a road accident, we should not expect it to know immediately the internal injuries of an unconscious victim.
- In all of these cases there is hidden state information in the environment, and that information would be useful if the agent knew it, but the agent cannot know it because it has never received any relevant sensations.
- In short, we don't fault an agent for not knowing something that matters, but only for having known something and then forgotten it! What we would like, ideally, is a state

signal that summarizes past sensations compactly, yet in such a way that all relevant information is retained.

- This normally requires more than the immediate sensations, but never more than the complete history of all past sensations.
- A state signal that succeeds in retaining all relevant information is said to be Markov, or to have the Markov property (we define this formally below).
- For example, a checkers position—the current configuration of all the pieces on the board—would serve as a Markov state because it summarizes everything important about the complete sequence of positions that led to it.
- Much of the information about the sequence is lost, but all that really matters for the future of the game is retained.
- Similarly, the current position and velocity of a cannonball is all that matters for its future flight.
- It doesn't matter how that position and velocity came about. This is sometimes also referred to as an "independence of path" property because all that matters is in the current state signal; its meaning is independent of the "path," or history, of signals that have led up to it.
- We now formally define the Markov property for the reinforcement learning problem.
- To keep the mathematics simple, we assume here that there are a finite number of states and reward values.
- This enables us to work in terms of sums and probabilities rather than integrals and probability densities, but the argument can easily be extended to include continuous states and rewards.
- Consider how a general environment might respond at time t+ 1 to the action taken at time t. In the most general, causal case this response may depend on everything that has happened earlier.
- In this case the dynamics can be defined only by specifying the complete probability distribution:

$$\Pr\{R_{t+1} = r, S_{t+1} = s' \mid S_0, A_0, R_1, \ldots, S_{t-1}, A_{t-1}, R_t, S_t, A_t\}, \qquad (3.4)$$

for all r, s', and all possible values of the past events: S_0, A_0, R_1, ..., S_{t-1}, A_{t-1}, R_t, S_t, A_t. If the state signal has the *Markov property*, on the other hand, then the environment's response at $t+1$ depends only on the state and action representations at t, in which case the environment's dynamics can be defined by specifying only

$$p(s', r|s, a) = \Pr\{R_{t+1} = r, S_{t+1} = s' \mid S_t, A_t\}, \qquad (3.5)$$

for all r, s', S_t, and A_t. In other words, a state signal has the Markov property, and is a Markov state, if and only if (3.5) is equal to (3.4) for all s', r, and histories, S_0, A_0, R_1, ..., S_{t-1}, A_{t-1}, R_t, S_t, A_t. In this case, the environment and task as a whole are also said to have the Markov property.

- If an environment has the Markov property, then its one-step dynamics (3.5) enable us to predict the next state and expected next reward given the current state and action.
- One can show that, by iterating this equation, one can predict all future states and expected rewards from knowledge only of the current state as well as would be possible given the complete history up to the current time.
- It also follows that Markov states provide the best possible basis for choosing actions. That is, the best policy for choosing actions as a function of a Markov state is just as good as the best policy for choosing actions as a function of complete histories.
- Even when the state signal is non-Markov, it is still appropriate to think of the state in reinforcement learning as an approximation to a Markov state.
- In particular, we always want the state to be a good basis for predicting future rewards and for selecting actions.
- In cases in which a model of the environment is learned, we also want the state to be a good basis for predicting subsequent states.
- Markov states provide an unsurpassed basis for doing all of these things.
- To the extent that the state approaches the ability of Markov states in these ways, one will obtain better performance from reinforcement learning systems.
- For all of these reasons, it is useful to think of the state at each time step as an approximation to a Markov state, although one should remember that it may not fully satisfy the Markov property.
- The Markov property is important in reinforcement learning because decisions and values are assumed to be a function only of the current state.
- In order for these to be effective and informative, the state representation must be informative.
- All of the theory presented in this book assumes Markov state signals. This means that not all the theory strictly applies to cases in which the Markov property does not strictly apply.
- However, the theory developed for the Markov case still helps us to understand the behaviour of the algorithms, and the algorithms can be successfully applied to many tasks with states that are not strictly Markov.
- A full understanding of the theory of the Markov case is an essential foundation for extending it to the more complex and realistic non-Markov case.
- Finally, we note that the assumption of Markov state representations is not unique to reinforcement learning but is also present in most if not all other approaches to artificial intelligence.

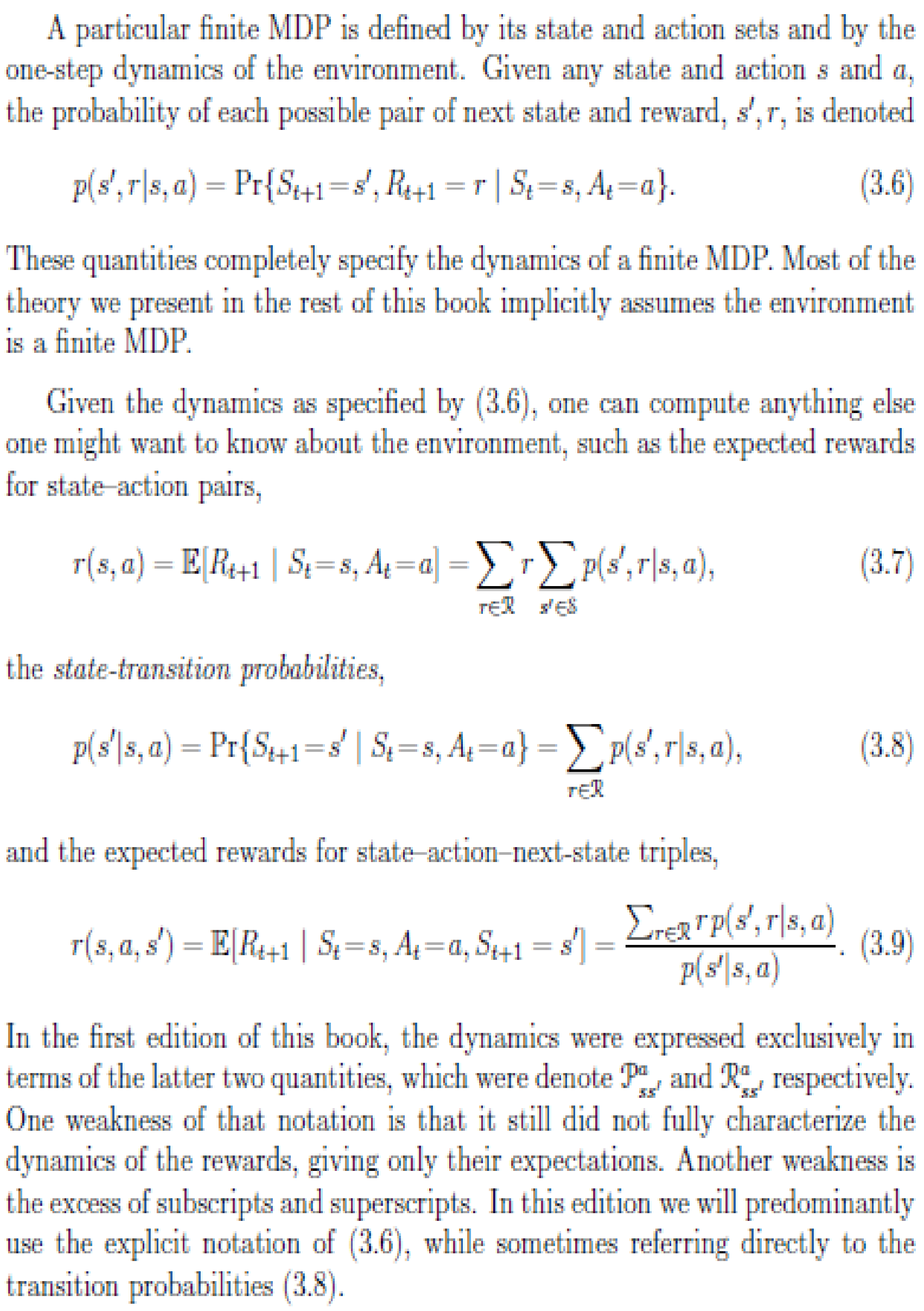
Example 3.5: Pole-Balancing State In the pole-balancing task introduced earlier, a state signal would be Markov if it specified exactly, or made it possible to reconstruct exactly, the position and velocity of the cart along the track, the angle between the cart and the pole, and the rate at which this angle is changing (the angular velocity). In an idealized cart–pole system, this information would be sufficient to exactly predict the future behavior of the cart and pole, given the actions taken by the controller. In practice, however, it is never possible to know this information exactly because any real sensor would introduce some distortion and delay in its measurements. Furthermore, in any real cart–pole system there are always other effects, such as the bending of the pole, the temperatures of the wheel and pole bearings, and various forms of backlash, that slightly affect the behavior of the system. These factors would cause violations of the Markov property if the state signal were only the positions and velocities of the cart and the pole.

Example 3.6: Draw Poker In draw poker, each player is dealt a hand of five cards. There is a round of betting, in which each player exchanges some of his cards for new ones, and then there is a final round of betting. At each round, each player must match or exceed the highest bets of the other players, or else drop out (fold). After the second round of betting, the player with the best hand who has not folded is the winner and collects all the bets. The state signal in draw poker is different for each player. Each player knows the cards in his own hand, but can only guess at those in the other players' hands. A common mistake is to think that a Markov state signal should include the contents of all the players' hands and the cards remaining in the deck. In a fair game, however, we assume that the players are in principle unable to determine these things from their past observations. If a player did know them, then she could predict some future events (such as the cards one could exchange for) better than by remembering all past observations. In addition to knowledge of one's own cards, the state in draw poker should include the bets and the numbers of cards drawn by the other players. For example, if one of the other players drew three new cards, you may suspect he retained a pair and adjust your guess of the strength of his hand accordingly. The players' bets also influence your assessment of their hands. In fact, much of your past history with these particular players is part of the Markov state. Does Ellen like to bluff, or does she play conservatively? Does her face or demeanor provide clues to the strength of her hand? How does Joe's play change when it is late at night, or when he has already won a lot of money? Although everything ever observed about the other players may have an effect on the probabilities that they are holding various kinds of hands, in practice this is far too much to remember and analyse, and most of it will have no clear effect on one's predictions and decisions. Very good poker players are adept at remembering just the key clues, and at sizing up new players quickly, but no one remembers everything that is relevant. As a result, the state representations people use to make their poker decisions are undoubtedly nonMarkov, and the decisions themselves are presumably imperfect. Nevertheless, people still make very good decisions in such tasks. We conclude that the inability to have access to a perfect Markov state representation is probably not a severe problem for a reinforcement learning agent.

6. Markov Decision Processes

- A reinforcement learning task that satisfies the Markov property is called a Markov decision process, or MDP.
- If the state and action spaces are finite, then it is called a finite Markov decision process (finite MDP).
- Finite MDPs are particularly important to the theory of reinforcement learning.

- We treat them extensively throughout this book; they are all you need to understand 90% of modern reinforcement learning.



A particular finite MDP is defined by its state and action sets and by the one-step dynamics of the environment. Given any state and action s and a, the probability of each possible pair of next state and reward, s', r, is denoted

$$p(s', r|s, a) = \Pr\{S_{t+1}=s', R_{t+1} = r \mid S_t=s, A_t=a\}. \tag{3.6}$$

These quantities completely specify the dynamics of a finite MDP. Most of the theory we present in the rest of this book implicitly assumes the environment is a finite MDP.

Given the dynamics as specified by (3.6), one can compute anything else one might want to know about the environment, such as the expected rewards for state–action pairs,

$$r(s, a) = \mathbb{E}[R_{t+1} \mid S_t=s, A_t=a] = \sum_{r\in\mathcal{R}} r \sum_{s'\in\mathcal{S}} p(s', r|s, a), \tag{3.7}$$

the *state-transition probabilities*,

$$p(s'|s, a) = \Pr\{S_{t+1}=s' \mid S_t=s, A_t=a\} = \sum_{r\in\mathcal{R}} p(s', r|s, a), \tag{3.8}$$

and the expected rewards for state–action–next-state triples,

$$r(s, a, s') = \mathbb{E}[R_{t+1} \mid S_t=s, A_t=a, S_{t+1} = s'] = \frac{\sum_{r\in\mathcal{R}} r\, p(s', r|s, a)}{p(s'|s, a)}. \tag{3.9}$$

In the first edition of this book, the dynamics were expressed exclusively in terms of the latter two quantities, which were denote $\mathcal{P}^a_{ss'}$ and $\mathcal{R}^a_{ss'}$ respectively. One weakness of that notation is that it still did not fully characterize the dynamics of the rewards, giving only their expectations. Another weakness is the excess of subscripts and superscripts. In this edition we will predominantly use the explicit notation of (3.6), while sometimes referring directly to the transition probabilities (3.8).

Example 3.7: Recycling Robot MDP The recycling robot (Example 3.3) can be turned into a simple example of an MDP by simplifying it and providing some more details. (Our aim is to produce a simple example, not a particularly realistic one.) Recall that the agent makes a decision at times determined by external events (or by other parts of the robot's control system). At each such time the robot decides whether it should (1) actively search for a can, (2) remain stationary and wait for someone to bring it a can, or (3) go back to home base to recharge its battery. Suppose the environment works as follows. The best way to find cans is to actively search for them, but this runs down the robot's battery, whereas waiting does not. Whenever the robot is searching, the possibility exists that its battery will become depleted. In this case the robot must shut down and wait to be rescued (producing a low reward). The agent makes its decisions solely as a function of the energy level of the battery. It can distinguish two levels, high and low, so that the state set is S = {high, low}. Let us call the possible decisions—the agent's actions— wait, search, and recharge. When the energy level is high, recharging would always be foolish, so we do not include it in the action set for this state. The agent's action sets are

$$\begin{aligned} \mathcal{A}(\texttt{high}) &= \{\texttt{search}, \texttt{wait}\} \\ \mathcal{A}(\texttt{low}) &= \{\texttt{search}, \texttt{wait}, \texttt{recharge}\}. \end{aligned}$$

If the energy level is high, then a period of active search can always be completed without risk of depleting the battery. A period of searching that begins with a high energy level leaves the energy level high with probability α and reduces it to low with probability 1−α. On the other hand, a period of searching undertaken when the energy level is low leaves it low with probability β and depletes the battery with probability 1−β. In the latter case, the robot must be rescued, and the battery is then recharged back to high. Each can be collected by the robot counts as a unit reward, whereas a reward of −3 results whenever the robot has to be rescued. Let r_{search} and r_{wait}, with $r_{search} > r_{wait}$, respectively denote the expected number of cans the robot will collect (and hence the expected reward) while searching and while waiting. Finally, to keep things simple, suppose that no cans can be collected during a run home for recharging, and that no cans can be collected on a step in which the battery is depleted. This system is then a finite MDP, and we can write down the transition probabilities and the expected rewards, as in Table 3.1.

A transition graph is a useful way to summarize the dynamics of a finite MDP. Figure 3.3 shows the transition graph for the recycling robot example. There are two kinds of nodes: state nodes and action nodes. There is a state node for each possible state (a large open circle labelled by the name of the state), and an action node for each state–action pair (a small solid circle labelled by the name of the action and connected by a line to the state node). Starting in state s and taking action a moves you along the line from state node s to action node (s, a). Then the environment responds with a transition to the next state's node via one of the arrows leaving action node (s, a). Each arrow corresponds to a triple (s, s0 , a), where s 0 is the next state, and we label the arrow with the transition probability, p(s 0 |s, a), and the expected reward for that transition, r(s, a, s0). Note that the transition probabilities labeling the arrows leaving an action node always sum to 1.

s	s'	a	$p(s'\|s,a)$	$r(s,a,s')$
high	high	search	α	$r_{\texttt{search}}$
high	low	search	$1-\alpha$	$r_{\texttt{search}}$
low	high	search	$1-\beta$	-3
low	low	search	β	$r_{\texttt{search}}$
high	high	wait	1	$r_{\texttt{wait}}$
high	low	wait	0	$r_{\texttt{wait}}$
low	high	wait	0	$r_{\texttt{wait}}$
low	low	wait	1	$r_{\texttt{wait}}$
low	high	recharge	1	0
low	low	recharge	0	0.

Table 3.1: Transition probabilities and expected rewards for the finite MDP of the recycling robot example. There is a row for each possible combination of current state, s, next state, s', and action possible in the current state, $a \in \mathcal{A}(s)$.

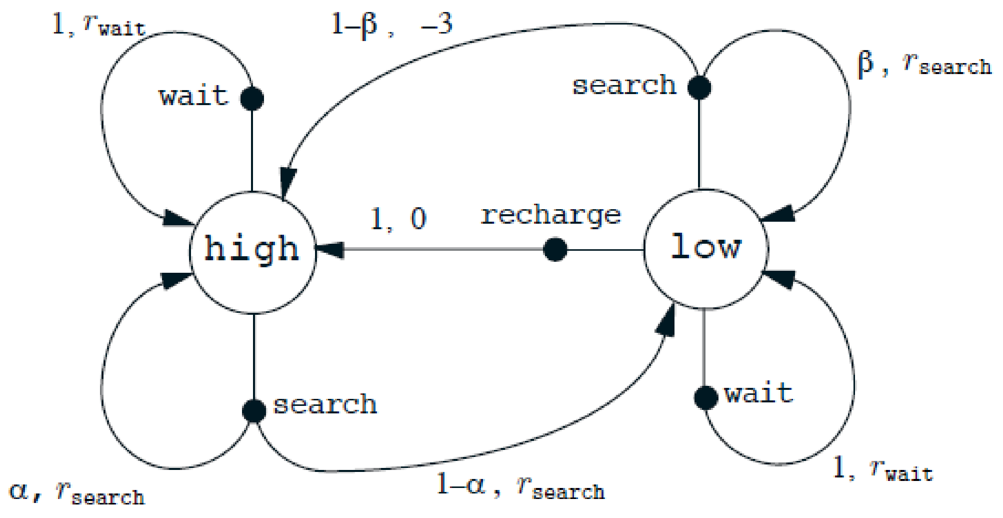


Figure 3.3: Transition graph for the recycling robot example.

7. Value Functions

- Almost all reinforcement learning algorithms involve estimating value functions—functions of states (or of state–action pairs) that estimate 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.
- Of course, the rewards the agent can expect to receive in the future depend on what actions it will take.
- Accordingly, value functions are defined with respect to particular policies.
- Recall that a policy, π, is a mapping from each state, s ∈ S, and action, a ∈ A(s), to the probability π(a|s) of taking action a when in state s.
- Informally, the value of a state s under a policy π, denoted $v_\pi(s)$, is the expected return when starting in s and following π thereafter.
- For MDPs, we can define vπ(s) formally as

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right], \tag{3.10}$$

where Eπ[·] denotes the expected value of a random variable given that the agent follows policy π, and t is any time step. Note that the value of the terminal state, if any, is always zero. We call the function vπ the state-value function for policy π.

- Similarly, we define the value of taking action a in state s under a policy π, denoted qπ(s, a), as the expected return starting from s, taking the action a, and thereafter following policy π:

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right]. \tag{3.11}$$

We call q_π the *action-value function for policy* π.

- The value functions v_π and q_π can be estimated from experience.
- For example, if an agent follows policy π and maintains an average, for each state encountered, of the actual returns that have followed that state, then the average will converge to the state's value, $v_\pi(s)$, as the number of times that state is encountered approaches infinity.
- If separate averages are kept for each action taken in a state, then these averages will similarly converge to the action values, $q_\pi(s, a)$.
- We call estimation methods of this kind Monte Carlo methods because they involve averaging over many random samples of actual returns.
- A fundamental property of value functions used throughout reinforcement learning and dynamic programming is that they satisfy particular recursive relationships.

- 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:

$$\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \,\middle|\, S_t = s\right] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \,\middle|\, S_{t+1} = s'\right]\right] \\
&= \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a) \Big[r + \gamma v_\pi(s')\Big], \qquad (3.12)
\end{aligned}$$

where it is implicit that the actions, a, are taken from the set A(s), the next states, s 0 , are taken from the set S (or from S + in the case of an episodic problem), and the rewards, r, are taken from the set R.

- Note also how in the last equation we have merged the two sums, one over all the values of s 0 and the other over all values of r, into one sum over all possible values of both.
- We will use this kind of merged sum often to simplify formulas. Note how the final expression can be read very easily as an expected value.
- It is really a sum over all values of the three variables, a, s 0 , and r.
- For each triple, we compute its probability, π(a|s)p(s 0 , r|s, a), weight the quantity in brackets by that probability, then sum over all possibilities to get an expected value.
- Equation (3.12) is the Bellman equation for v_π.
- It expresses a relationship between the value of a state and the values of its successor states. Think of looking ahead from one state to its possible successor states, as suggested by Figure 3.4a.
- Each open circle represents a state and each solid circle represents a state–action pair. Starting from state s, the root node at the top, the agent could take any of some set of actions—three are shown in Figure 3.4a.
- From each of these, the environment could respond with one of several next states, s 0 , along with a reward, r.
- The Bellman equation (3.12) averages over all the possibilities, weighting each by its probability of occurring.
- It states that the value of the start state must equal the (discounted) value of the expected next state, plus the reward expected along the way.

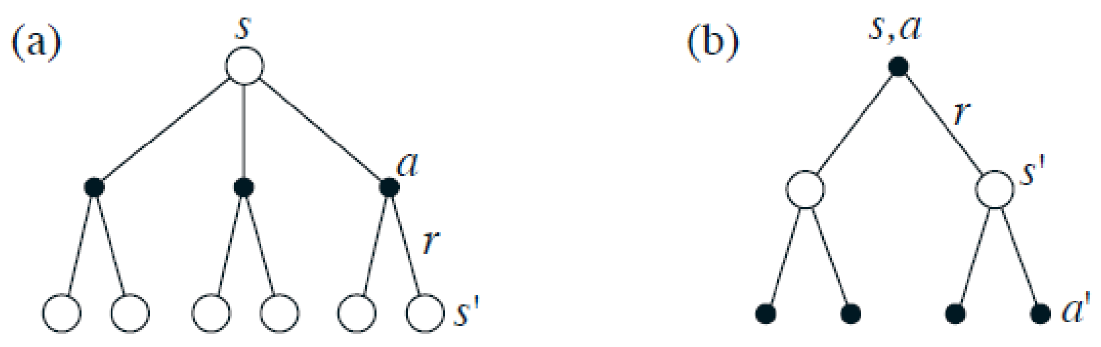
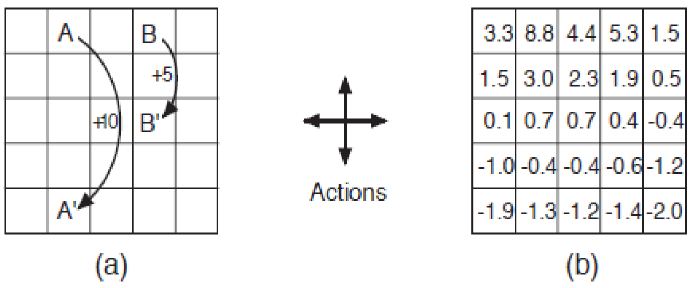


Figure 3.4: Backup diagrams for (a) v_π and (b) q_π.

- The value function v_π is the unique solution to its Bellman equation.
- We call diagrams like those shown in Figure 3.4 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 state–action pair) from its successor states (or state–action pairs).
- We use backup diagrams throughout the book to provide graphical summaries of the algorithms we discuss. (Note that unlike transition graphs, the state nodes of backup diagrams do not necessarily represent distinct states; for example, a state might be its own successor. We also omit explicit arrowheads because time always flows downward in a backup diagram.).

Example 3.8: Gridworld Figure 3.5a uses a rectangular grid to illustrate value functions for a simple finite MDP. The cells of the grid correspond to the states of the environment. At each cell, four actions are possible: north, south, east, and west, which deterministically cause the agent to move one cell in the respective direction on the grid. Actions that would take the agent off the grid leave its location unchanged, but also result in a reward of −1. Other actions result in a reward of 0, except those that move the agent out of the special states A and B. From state A, all four actions yield a reward of +10 and take the agent to A0 . From state B, all actions yield a reward of +5 and take the agent to B0 . Suppose the agent selects all four actions with equal probability in all states. Figure 3.5b shows the value function, vπ, for this policy, for the discounted reward case with $\gamma = 0.9$. This value function was computed by solving the system of equations (3.12). Notice the negative values near the lower edge; these are the result of the high probability of hitting the edge of the grid there under the random policy. State A is the best state to be in under this policy, but its expected return is less than 10, its immediate reward, because from A the agent is taken to A0 , from which it is likely to run into the edge of the grid. State B, on the other hand, is valued more than 5, its immediate reward, because from B the agent is taken to B0 , which has a positive value. From B0 the expected penalty (negative reward) for possibly running into an edge is more than compensated for by the expected gain for possibly stumbling onto A or B.



3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

Figure 3.5: Grid example: (a) exceptional reward dynamics; (b) state-value function for the equiprobable random policy.

Example 3.9: Golf To formulate playing a hole of golf as a reinforcement learning task, we count a penalty (negative reward) of −1 for each stroke until we hit the ball into the hole. The state is the location of the ball. The value of a state is the negative of the number of strokes to the hole from that location. Our actions are how we aim and swing at the ball, of course, and which club we select. Let us take the former as given and consider just the choice of club, which we assume is either a putter or a driver. The upper part of Figure 3.6 shows a possible state-value function, $v_{putt}(s)$, for the policy that always uses the putter. The terminal state in-the-hole has a value of 0. From anywhere on the green we assume we can make a putt; these states have value −1. Off the green we cannot reach the hole by putting, and the value is greater. If we can reach the green from a state by putting, then that state must have value one less than the green's value, that is, −2. For simplicity, let us assume we can putt very precisely and deterministically, but with a limited range. This gives us the sharp contour line labelled −2 in the figure; all locations between that line and the green require exactly two strokes to complete the hole. Similarly, any location within putting range of the −2 contour line must have a value of −3, and so on to get all the contour lines shown in the figure. Putting doesn't get us out of sand traps, so they have a value of −∞. Overall, it takes us six strokes to get from the tee to the hole by putting.

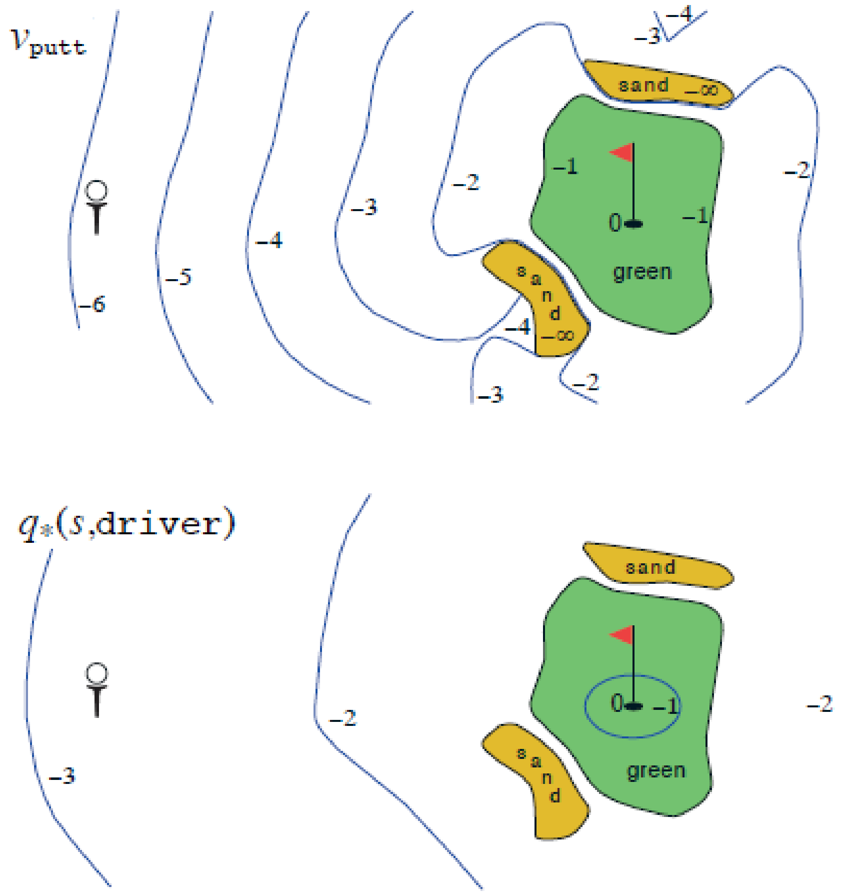


Figure 3.6: A golf example: the state-value function for putting (above) and the optimal action-value function for using the driver (below).

8. Optimal Value Functions

Solving a reinforcement learning task means, roughly, finding a policy that achieves a lot of reward over the long run. For finite MDPs, we can precisely define an optimal policy in the following way. Value functions define a partial ordering over policies. A policy π is defined to be better than or equal to a policy π' if its expected return is greater than or equal to that of π' 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*. Although there may be more than one, we denote all the optimal policies by π_*. They share the same state-value function, called the *optimal state-value function*, denoted v_*, and defined as

$$v_*(s) = \max_\pi v_\pi(s), \tag{3.13}$$

for all $s \in \mathcal{S}$.

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

$$q_*(s,a) = \max_\pi q_\pi(s,a), \tag{3.14}$$

for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$. For the state–action pair (s,a), this function gives the expected return for taking action a in state s and thereafter following an optimal policy. Thus, we can write q_* in terms of v_* as follows:

$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a]. \tag{3.15}$$

Example 3.10: Optimal Value Functions for Golf The lower part of Figure 3.6 shows the contours of a possible optimal action-value function $q_*(s, \text{driver})$. These are the values of each state if we first play a stroke with the driver and afterward select either the driver or the putter, whichever is better. The driver enables us to hit the ball farther, but with less accuracy. We can reach the hole in one shot using the driver only if we are already very close; thus the -1 contour for $q_*(s, \text{driver})$ covers only a small portion of the green. If we have two strokes, however, then we can reach the hole from much farther away, as shown by the -2 contour. In this case we don't have to drive all the way to within the small -1 contour, but only to anywhere on the green; from there we can use the putter. The optimal action-value function gives the values after committing to a particular first action, in this case, to the driver, but afterward using whichever actions are best. The -3 contour is still farther out and includes the starting tee. From the tee, the best sequence of actions is two drives and one putt, sinking the ball in three strokes.

Because v_* is the value function for a policy, it must satisfy the self-consistency condition given by the Bellman equation for state values (3.12). Because it is the optimal value function, however, v_*'s consistency condition can be written in a special form without reference to any specific policy. This is the Bellman equation for v_*, or the Bellman optimality equation. Intuitively, the Bellman optimality equation expresses the fact that the value of a state under an optimal policy must equal the expected return for the best action from that state:

$$
\begin{aligned}
v_*(s) &= \max_{a\in\mathcal{A}(s)} q_{\pi_*}(s,a) \\
&= \max_a \mathbb{E}_{\pi^*}[G_t \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}_{\pi^*}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right] \\
&= \max_a \mathbb{E}_{\pi^*}\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \,\middle|\, S_t = s, A_t = a\right] \\
&= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \qquad (3.16) \\
&= \max_{a\in\mathcal{A}(s)} \sum_{s',r} p(s',r|s,a)\big[r + \gamma v_*(s')\big]. \qquad (3.17)
\end{aligned}
$$

The last two equations are two forms of the Bellman optimality equation for v_*. The Bellman optimality equation for q_* is

$$
\begin{aligned}
q_*(s,a) &= \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \,\Big|\, S_t = s, A_t = a\Big] \\
&= \sum_{s',r} p(s',r|s,a)\Big[r + \gamma \max_{a'} q_*(s',a')\Big].
\end{aligned}
$$

The backup diagrams in Figure 3.7 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 except that arcs have been added at the agent's choice points to represent that the maximum over that choice is taken rather than the expected value given some policy. Figure 3.7a graphically represents the Bellman optimality equation (3.17).

For finite MDPs, the Bellman optimality equation (3.17) has a unique solution independent of the policy. The Bellman optimality equation is actually a system of equations, one for each state, so if there are N states, then there are N equations in N unknowns. If the dynamics of the environment are known ($p(s',r|s,a)$), then in principle one can solve this system of equations

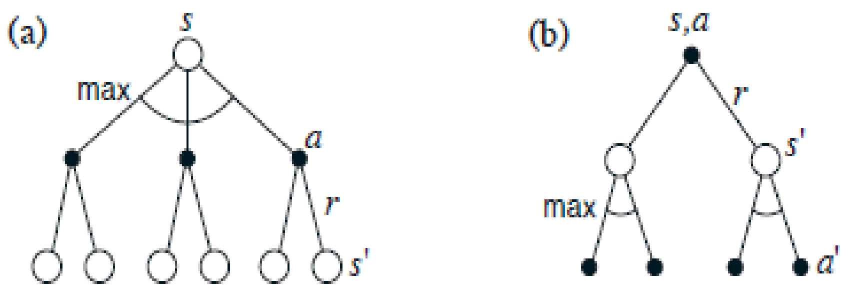


Figure 3.7: Backup diagrams for (a) v_* and (b) q_*

for v_* using any one of a variety of methods for solving systems of nonlinear equations. One can solve a related set of equations for q_*.

Example 3.11: Bellman Optimality Equations for the Recycling Robot Using (3.17), we can explicitly give the Bellman optimality equation for the recycling robot example. To make things more compact, we abbreviate the states `high` and `low`, and the actions `search`, `wait`, and `recharge` respectively by `h`, `l`, `s`, `w`, and `re`. Since there are only two states, the Bellman optimality equation consists of two equations. The equation for $v_*(\mathtt{h})$ can be written as follows:

$$\begin{aligned}
v_*(\mathtt{h}) &= \max\left\{ \begin{array}{l} p(\mathtt{h}|\mathtt{h},\mathtt{s})[r(\mathtt{h},\mathtt{s},\mathtt{h}) + \gamma v_*(\mathtt{h})] + p(\mathtt{l}|\mathtt{h},\mathtt{s})[r(\mathtt{h},\mathtt{s},\mathtt{l}) + \gamma v_*(\mathtt{l})], \\ p(\mathtt{h}|\mathtt{h},\mathtt{w})[r(\mathtt{h},\mathtt{w},\mathtt{h}) + \gamma v_*(\mathtt{h})] + p(\mathtt{l}|\mathtt{h},\mathtt{w})[r(\mathtt{h},\mathtt{w},\mathtt{l}) + \gamma v_*(\mathtt{l})] \end{array} \right\} \\
&= \max\left\{ \begin{array}{l} \alpha[r_\mathtt{s} + \gamma v_*(\mathtt{h})] + (1-\alpha)[r_\mathtt{s} + \gamma v_*(\mathtt{l})], \\ 1[r_\mathtt{w} + \gamma v_*(\mathtt{h})] + 0[r_\mathtt{w} + \gamma v_*(\mathtt{l})] \end{array} \right\} \\
&= \max\left\{ \begin{array}{l} r_\mathtt{s} + \gamma[\alpha v_*(\mathtt{h}) + (1-\alpha)v_*(\mathtt{l})], \\ r_\mathtt{w} + \gamma v_*(\mathtt{h}) \end{array} \right\}.
\end{aligned}$$

Following the same procedure for $v_*(\mathtt{l})$ yields the equation

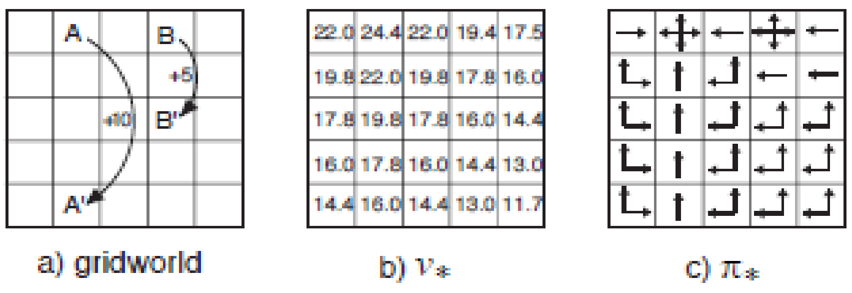
$$v_*(\mathtt{l}) = \max\left\{ \begin{array}{l} \beta r_\mathtt{s} - 3(1-\beta) + \gamma[(1-\beta)v_*(\mathtt{h}) + \beta v_*(\mathtt{l})] \\ r_\mathtt{w} + \gamma v_*(\mathtt{l}), \\ \gamma v_*(\mathtt{h}) \end{array} \right\}.$$

For any choice of $r_\mathtt{s}$, $r_\mathtt{w}$, α, β, and γ, with $0 \le \gamma < 1$, $0 \le \alpha, \beta \le 1$, there is exactly one pair of numbers, $v_*(\mathtt{h})$ and $v_*(\mathtt{l})$, that simultaneously satisfy these two nonlinear equations.

Example 3.12: Solving the Gridworld Suppose we solve the Bellman equation for v_* for the simple grid task introduced in Example 3.8 and shown again in Figure 3.8a. Recall that state A is followed by a reward of +10 and transition to state A′, while state B is followed by a reward of +5 and transition to state B′. Figure 3.8b shows the optimal value function, and Figure 3.8c shows the corresponding optimal policies. Where there are multiple arrows in a cell, any of the corresponding actions is optimal.

Explicitly solving the Bellman optimality equation provides one route to finding an optimal policy, and thus to solving the reinforcement learning problem. However, this solution is rarely directly useful. It is akin to an exhaustive search, looking ahead at all possibilities, computing their probabilities of occurrence and their desirabilities in terms of expected rewards. 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; and (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. For example, although the first and third assumptions present no problems for the game of backgammon, the second is a major impediment. Since the game has about 10^{20} states, it would take thousands of years on today's fastest computers to solve the Bellman equation for v_*, and the same is true for finding q_*. In reinforcement learning one typically has to settle for approximate solutions.



22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

Figure 3.8: Optimal solutions to the gridworld example.

9. Optimality and Approximation

- We have defined optimal value functions and optimal policies.
- Clearly, an agent that learns an optimal policy has done very well, but in practice this rarely happens.
- For the kinds of tasks in which we are interested, optimal policies can be generated only with extreme computational cost.
- A well-defined notion of optimality organizes the approach to learning we describe in this book and provides a way to understand the theoretical properties of various learning algorithms, but it is an ideal that agents can only approximate to varying degrees.
- As we discussed above, even if we have a complete and accurate model of the environment's dynamics, it is usually not possible to simply compute an optimal policy by solving the Bellman optimality equation.
- For example, board games such as chess are a tiny fraction of human experience, yet large, custom-designed computers still cannot compute the optimal moves.
- A critical aspect of the problem facing the agent is always the computational power available to it, in particular, the amount of computation it can perform in a single time step.
- The memory available is also an important constraint.
- A large amount of memory is often required to build up approximations of value functions, policies, and models.
- In tasks with small, finite state sets, it is possible to form these approximations using arrays or tables with one entry for each state (or state–action pair).
- This we call the tabular case, and the corresponding methods we call tabular methods.
- In many cases of practical interest, however, there are far more states than could possibly be entries in a table.
- In these cases, the functions must be approximated, using some sort of more compact parameterized function representation.

- Our framing of the reinforcement learning problem forces us to settle for approximations. However, it also presents us with some unique opportunities for achieving useful approximations.
- For example, in approximating optimal behaviour, there may be many states that the agent faces with such a low probability that selecting suboptimal actions for them has little impact on the amount of reward the agent receives.
- Tesauro's backgammon player, for example, plays with exceptional skill even though it might make very bad decisions on board configurations that never occur in games against experts.
- In fact, it is possible that TD-Gammon makes bad decisions for a large fraction of the game's state set.
- The on-line nature of reinforcement learning makes it possible to approximate optimal policies in ways that put more effort into learning to make good decisions for frequently encountered states, at the expense of less effort for infrequently encountered states.
- This is one key property that distinguishes reinforcement learning from other approaches to approximately solving MDPs.