Reading of theoretical aspect of reinforcement learning – Chapter 3

Zhe Rao

- Finite MDPs problem involves evaluative feedback, as in bandits, but also an associative aspect – choosing different actions in different situations
  - MDPs are a classical formalization of sequential decision making, where actions influence not just immediate rewards, but also subsequent situations, or states.
  - Whereas in bandit problems we estimated the value $q_*(a)$ of each action a, in MDPs we estimate the value $q_*(s,a)$ of each action a in each state s, or we estimate the value $v_*(s)$ of each state given optimal action selections
  - As in all of artificial intelligence, there is a tension between breadth of applicability and mathematical tractability.

1. 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 is called the *agent*.
  - The thing it interacts with, comprising everything outside the agent, is called the *environment*.
  - 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

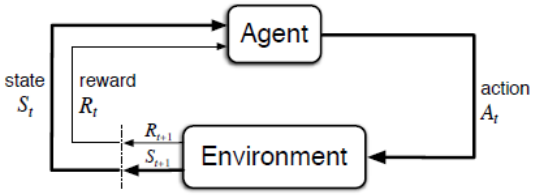


Figure 3.1: The agent–environment interaction in a Markov decision process.

- More specifically, the agent and environment interact at each of a sequence of discrete time steps, $t = 0,1,2,\ldots$
  - At each time step t, the agent receives some representation of the environment's $state$, $S_t \in S$, and on that basis selects an *action*, $A_t \in A(s)$.
  - One time step later, in part as a consequence of its action, the agent receives a numerical *reward*, $R_{t+1} \in R$, and finds itself in a new state, S_{t+1}.
  - The MDP and agent together thereby give rise to a sequence or *trajectory* that like this

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots \quad (3.1)$$

- In a *finite* MDP, the sets of states, actions, and rewards all have a finite number of elements.
  - In this case, the random variables R_t and S_t have well defined discrete probability distributions dependent only on the preceding state and action.
  - That is, for particular values of these random variables, $s' \in S$ and $r \in R$

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

  - The function p defines the *dynamics* of the MDP. This function is an ordinary deterministic function of four arguments
    - Note that p specifies a probability distribution for each choice of s, a

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

- The state must include information about all aspects of the past agent environment interaction that make a difference for the future.
  - If it does, then the state is said to have the *Markov property*.
  - From p, one might want to know about the environment, such as the ***state-transition probabilities***, denote

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

  - We can also compute the ***expected rewards*** for state-action pairs as a two-argument function

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

  - And the expected rewards for state-action-next-state triples as a three-argument function

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

- The states can take a wide variety of forms.
  - They can be completely determined by low-level sensations, such as direct sensor readings, or they can be more high-level and abstract, such as symbolic descriptions of objects in a room.
  - Some of what makes up a state could be based on memory of past sensations or even be entirely mental or subjective
    - E.g. an agent could be in the state of not being sure where an object is, or of having just been surprised in some clearly defined sense.
  - Similarly, some actions might be totally mental or computational
    - E.g. some actions might control what an agent chooses to think about, or where it focuses its attention.
  - In general, actions can be any decisions we want to learn how to make, and states can be anything we can know that might be useful in making them

- In particular, the boundary between agent and environment is typically not the same as the physical boundary of a robot's or an animal's body
  - Usually, the boundary is drawn closer to the agent than that.
    - For example, the motors and mechanical linkages of a robot and its sensing hardware should usually be considered parts of the environment rather than parts of the agent.

    - ✧ Similarly, if we apply the MDP framework to a person or animal, the muscles, skeleton, and sensory organs should be considered part of the environment
  - ➢ Rewards, too, presumably are computed inside the physical bodies of natural and artificial learning systems, but are considered external to the agent.

- The **general rule** we follow is that anything that cannot be changed arbitrarily by the agent is considered to be outside of it and thus part of its environment.
  - ➢ The agent-environment boundary represents the limit of the agent's *absolute control*, not of its knowledge.

- The MDP framework proposes that whatever the details of the sensory, memory, and control apparatus, and whatever objective one is trying to achieve, any problem of learning goal-directed behavior can be reduced to three signals passing back and forth between an agent and its environment:
  - ➢ One signal to represent the choices made by the agent (the actions)
  - ➢ One signal to represent the basis on which the choices are made (the states)
  - ➢ One signal to define the agent's goal (the rewards)

- Of course, the particular states and actions vary greatly from task to task, and how they are represented can strongly affect performance.
  - ➢ In reinforcement learning, as in other kinds of learning, such representational choices are at present more art than science.

Example

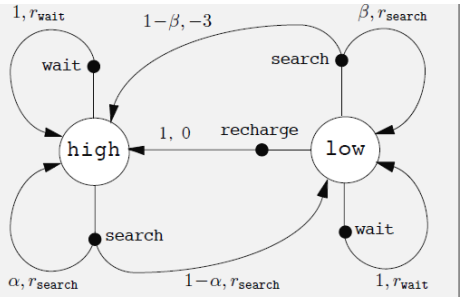
- Consider a problem involve high-level decisions about how to search for cans, they are made by a reinforcement learning agent based on the current charge level of the battery
  - ➢ We assume that two charge levels *high, low* are considered.
  - ➢ The agent can decide whether to
    - ✧ 1) actively search for a can for a certain period of time
    - ✧ 2) remain stationary and wait for someone to bring it a can
    - ✧ 3) head back to home base to recharge
  - ➢ The action sets are
    - ✧ $A(high) = \{search, wait\}$
    - ✧ $A(low) = \{search, wait, recharge\}$

- The rewards are zero most of the time, but become positive when the robot secures an empty can, or large and negative if the battery runs all the way down.
  - ➢ The best way to find cans is to actively search for them, but this runs down the robot's battery, whereas waiting does not.
    - ✧ 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 with probability α and reduces it to low with probability $1 - \alpha$.

- ✧ A period of searching undertaken with the energy is low leaves it low with probability β and depletes the battery with probability $1-\beta$
- ➢ Each can 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 $r_{research} > r_{wait}$, denote the expected number of cans the robot will collect
- ➢ Finally, 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, with dynamics as indicated below

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



- ➢ Note that there is a row in the table for each possible combination of current state, *s*, action, *a*, and next state, *s'*.
- Shown on the right is another useful way of summarizing the dynamics of a finite MDP, as a *transition graph*.
  - ➢ There are two kins of nodes: state nodes and action notes. There is a state node for each possible state, and an action node for each state-action pair
  - ➢ Each arrow corresponds to a triple (*s, s', a*), where *s'* is the next state, and we label the arrow with the transition probability, $p(s'|s,a)$, and the expected reward for that transition, $r(s,a,s')$.
  - ➢ Note that the transition probabilities labeling the arrows leaving an action node always sum to 1

2. Goals and Rewards

- We can clearly state 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)

- 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.

- Note that, if we want the robot to do something for us, we must provide rewards to it in such a way that in maximizing them the agent will also achieve our goals.
  - ➢ It is thus critical that the rewards we set up truly indicate what we want accomplished.
  - ➢ In particular, the reward signal is not the place to import to the agent prior knowledge about *how* to achieve what we want it to do.
    - ✧ E.g. a chess-playing agent should be rewarded only for actually winning, not for achieving subgoals such as taking its opponent's pieces or gaining control of the center of the board
  - ➢ The reward signal is your way of communicating to the agent *what* you want achieved, not *how* you want it achieved

3. Returns and Episodes

- If the sequence of rewards received after time t is denoted R_{t+1}, ..., 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 rewards

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

  - ➢ 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***.
  - ➢ Each episode ends in a special state called the ***terminal state***, followed by a rest to a standard starting state or to a sample from a standard distribution of starting states.
  - ➢ 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 task***.
    - ✧ In episodic tasks we sometimes need to distinguish the set of all nonterminal state, S, from the set of all states plus the terminal state 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.
  - ➢ E.g. 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***.

- 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.

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

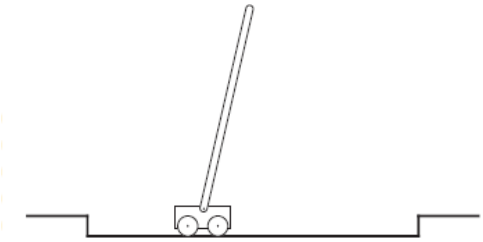
- Where γ is a parameter between 0 and 1, called the ***discount rate***.
- As γ approaches 1, the return objective takes future rewards into account more strongly; the agent becomes more farsighted

- Returns at successive time steps are related to each other in a way that is important for the theory and algorithms of reinforcement learning

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

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

- Example: Pole-Balancing



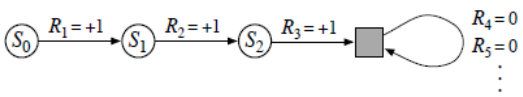
- A failure is said to occur if the pole falls past a given angle from vertical or if the cart runs off the track. The pole is rest to vertical after each failure
- This task could be treated as episodic, where the natural episodes are the repeated attempts to balance the pole
  - The reward in this 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 $-\gamma^{K-1}$, where K is the number of time steps before failure (as well as to the times of later failures)
- 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
- 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 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 other convention to obtain a single notation that covers both episodic and continuing tasks.
  - Return can be defined as a sum over a finite number of terms or as a sum over an infinite number of terms
  - These two can be unified by considering episode termination to be the entering of a special ***absorbing state*** that transitions only to itself and that generates only rewards of zero.



  - 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, …
  - Thus, we can define the return in general, using the convention of omitting episode numbers when they are not needed, and including the possibility that $\gamma = 1$ if the sum remains defined.
  - Alternatively, we can write

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

    - Including the possibility that $T = \infty$ or $\gamma = 1$ (but not both)

5. Policies and 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.
  - 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 ways of acting, called ***policies***.

- Formally, a *policy* is a mapping from states to probabilities of selecting each possible action.
  - If the agent is following policy π at time t, then $\pi(a|s)$ is the probability that $A_t = a$ if $S_t = s$.
  - Reinforcement learning methods specify how the agent's policy is changed as a result of its experience.

- The ***value function*** of a state s under a policy π, denoted $v_\pi(s)$, is the expected return when starting in s and following π thereafter.

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right], \text{ for all } s \in \mathcal{S}, \qquad (3.12)$$

  - Note that the value of the terminal state, if any, is always zero.
  - We call the function v_π the ***state-value function for policy π***.

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

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right]. \qquad (3.13)$$

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

- The value function 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 each 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

- Of course, if there are very many states, then it *may not be practical to keep separate averages* for each state individually.
  - Instead, the agent would have to maintain v_π and q_π as ***parameterized functions*** (with fewer parameters than states) and adjust the parameters to better match the observed returns
  - This can also produce accurate estimates, although much depends on the nature of the parameterized function approximator.

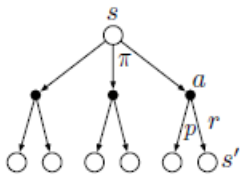
- A fundamental property of value functions used throughout reinforcement learning and dynamic programming is that they satisfy recursive relationships.
  - For any policy π and any state s, the following consistency condition holds between the value of s and the value of its possible successor states

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] && \text{(by (3.9))} \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)\left[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']\right] \\ &= \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a)\left[r + \gamma v_\pi(s')\right], \quad \text{for all } s \in \mathcal{S}, && (3.14) \end{aligned}$$

  - Note also how we have merged the two sums, one over all the values of s' and the other over all the values of r, into one sum over all the possible values of both
  - Note how the final expression can be read easily as an expected value. It is really a sum over all values of the three variables, a, s', and r.

- ✧ For each triple, we compute its probability, $\pi(a|s)p(s',r|s,a)$, weight the quantity in brackets by that probability, then sum over all possibilities to get an expected value.

- The equation above is call the ***Bellman equation for*** v_π.
  - ➢ It expresses a relationship between the value of a state and the values of its successor states



Backup diagram for v_π

  - ➢ 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 the diagram – based on its policy π
    - ✧ From each of these, the environment could respond with one of several next state, s' (two are shown in the figure), along with a reward, r, depending on its dynamics given by the function p.
  - ➢ The equation averages over all the possibilities, weighting each by its probability of occurring.
    - ✧ It states that the value of the start state must equal to discounted value of the expected next state, plus the reward expected along the way

- The value function v_π is the unique solution to the Bellman equation.
  - ➢ We show in subsequent chapters how this equation forms the basis of a number of ways to compute, approximate, and learn v_π
  - ➢ We call diagrams like that above ***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).

6. Optimal Policies and 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
    - ✧ 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.
    - ✧ There is always at least one policy that is better than or equal to all other policies

- We denote all the optimal policies by π_*.
  - ➢ They share the same state-value function, called the ***optimal state-value function***, defined as

$$v_*(s) \doteq \max_\pi v_\pi(s), \tag{3.15}$$

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

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

$$q_*(s,a) \doteq \max_\pi q_\pi(s,a), \tag{3.16}$$

  - 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

$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a]. \tag{3.17}$$

- Because v_* is the value function for a policy, it must satisfy the self-consistency condition given by the Bellman equation for state values
  - 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 the state:

$$\begin{aligned} v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) \\ &= \max_a \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] \\ &= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] && \text{(by (3.9))} \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] && (3.18) \\ &= \max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma v_*(s')\big]. && (3.19) \end{aligned}$$

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

$$\begin{aligned} q_*(s,a) &= \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \;\Big|\; S_t = s, A_t = a\Big] \\ &= \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma \max_{a'} q_*(s', a')\Big]. \end{aligned} \tag{3.20}$$

- The backup diagrams in the figure below show graphically the spans of future states and actions considered in the Bellman optimality equation for v_* and q_*.
  - There are the same as the backup diagrams represented earlier 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.

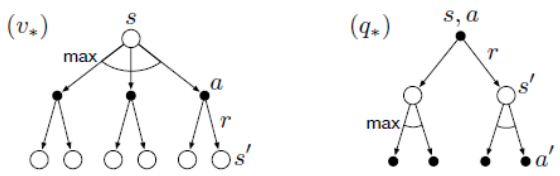


Figure 3.4: Backup diagrams for v_* and q_*

- For finite MDPs, the Bellman optimality equation for v_* has as unique solution
  - 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 p of the environment are known, then in principle one can solve this system of equations 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_*.

- Once one has v_*, it is relatively easy to determine an optimal policy.
  - For each state s, there will be one or more actions at which the maximum is obtained in the Bellman optimality equation.
  - Having q_* makes choosing optimal actions even easier. It can simply find any action that maximizes $q_*(s, a)$.
  - Hence, at the cost of representing a function of state-action pairs, instead of just of states, the optimal action-value function allows optimal actions to be selected without having to know anything about possible successor states and their values, that is, without having to know anything about the environment's dynamics

- **Recycling Robot example**

Example 3.9: Bellman Optimality Equations for the Recycling Robot Using (3.19), 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`. Because 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. ■

- 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 desirability in terms of expected rewards
  - This solution relies on at least three assumptions that are rarely true in practice

  - ✧ 1) the dynamics of the environment are accurately known
  - ✧ 2) computational resources are sufficient to complete the calculation
  - ✧ 3) the states have 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

- In reinforcement learning, one typically has to settle for ***approximate solutions***.
  - ➢ Many different decision-making methods can be viewed as ways of approximately solving the Bellman optimality equation
    - ✧ E.g. heuristic search methods can be viewed as expanding the right-hand side of equation 3.19 several times, up to some depth, forming a "tree" of possibilities, and then using a heuristic evaluation function to approximate v_* at the "leaf" nodes.
  - ➢ Many reinforcement learning methods can be clearly understood as approximately solving the Bellman optimality equation, using actual experienced transitions in place of knowledge of the expected transitions.

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.
  - ➢ Optimal policies can be generated only with extreme computational cost
    - ✧ 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 important constraint.
    - ✧ A larger amount of memory is often required to build up approximations of value functions, policies, and models

- 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 behavior, 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.