Chapter 3: The Reinforcement Learning Problem

Objectives of this chapter:

☑ describe the RL problem we will be studying for the remainder of the course
☑ present idealized form of the RL problem for which we have precise theoretical results;
☑ introduce key components of the mathematics: value functions and Bellman equations;
☑ describe trade-offs between applicability and mathematical tractability.
Agent and environment interact at discrete time steps: \( t = 0, 1, 2, \ldots \)

- Agent observes state at step \( t \): \( s_t \in S \)
- Produces action at step \( t \): \( a_t \in A(s_t) \)
- Gets resulting reward: \( r_{t+1} \in \mathbb{R} \)
- And resulting next state: \( s_{t+1} \)
The Agent Learns a Policy

Policy at step $t$, $\pi_t$:

a mapping from states to action probabilities

$\pi_t(s, a) = \text{probability that } a_t = a \text{ when } s_t = s$

- Reinforcement learning methods specify how the agent changes its policy as a result of experience.
- Roughly, the agent’s goal is to get as much reward as it can over the long run.
Getting the Degree of Abstraction Right

- Time steps need not refer to fixed intervals of real time.
- Actions can be low level (e.g., voltages to motors), or high level (e.g., accept a job offer), “mental” (e.g., shift in focus of attention), etc.
- States can low-level “sensations”, or they can be abstract, symbolic, based on memory, or subjective (e.g., the state of being “surprised” or “lost”).
- An RL agent is not like a whole animal or robot.
- Reward computation is in the agent’s environment because the agent cannot change it arbitrarily.
- The environment is not necessarily unknown to the agent, only incompletely controllable.
Is a scalar reward signal an adequate notion of a goal?—maybe not, but it is surprisingly flexible.

A goal should specify what we want to achieve, not how we want to achieve it.

A goal must be outside the agent’s direct control—thus outside the agent.

The agent must be able to measure success:
- explicitly;
- frequently during its lifespan.
The reward hypothesis

☐ That all of what we mean by goals and purposes can be well thought of as the maximization of the cumulative sum of a received scalar signal (reward)

☐ A sort of null hypothesis.
  - Probably ultimately wrong, but so simple we have to disprove it before considering anything more complicated
Returns

Suppose the sequence of rewards after step $t$ is:

$$r_{t+1}, r_{t+2}, r_{t+3}, \ldots$$

What do we want to maximize?

In general, we want to maximize the expected return, $E\{R_t\}$, for each step $t$.

**Episodic tasks**: interaction breaks naturally into episodes, e.g., plays of a game, trips through a maze.

$$R_t = r_{t+1} + r_{t+2} + \cdots + r_T,$$

where $T$ is a final time step at which a terminal state is reached, ending an episode.
Continuing tasks: interaction does not have natural episodes.

Discounted return:

\[ R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \]

where \( \gamma, 0 \leq \gamma \leq 1 \), is the discount rate.

shortsighted \( 0 \leftarrow \gamma \rightarrow 1 \) farsighted
An Example

Avoid failure: the pole falling beyond a critical angle or the cart hitting end of track.

As an episodic task where episode ends upon failure:

\[ \text{reward} = +1 \text{ for each step before failure} \]

\[ \Rightarrow \text{return} = \text{number of steps before failure} \]

As a continuing task with discounted return:

\[ \text{reward} = -1 \text{ upon failure; 0 otherwise} \]

\[ \Rightarrow \text{return} = -\gamma^k, \text{ for } k \text{ steps before failure} \]

In either case, return is maximized by avoiding failure for as long as possible.
Another Example

Get to the top of the hill as quickly as possible.

reward = −1 for each step where not at top of hill
⇒ return = − number of steps before reaching top of hill

Return is maximized by minimizing number of steps to reach the top of the hill.
In episodic tasks, we number the time steps of each episode starting from zero.

We usually do not have to distinguish between episodes, so we write $s_t$ instead of $s_{t,j}$ for the state at step $t$ of episode $j$.

Think of each episode as ending in an absorbing state that always produces reward of zero:

We can cover all cases by writing $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$, where $\gamma$ can be 1 only if a zero reward absorbing state is always reached.
The Markov Property

- By “the state” at step \( t \), the book means whatever information is available to the agent at step \( t \) about its environment.
- The state can include immediate “sensations,” highly processed sensations, and structures built up over time from sequences of sensations.
- Ideally, a state should summarize past sensations so as to retain all “essential” information, i.e., it should have the Markov Property:

\[
\Pr \left\{ s_{t+1} = s', r_{t+1} = r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0 \right\} = \\
\Pr \left\{ s_{t+1} = s', r_{t+1} = r \mid s_t, a_t \right\}
\]

for all \( s' \), \( r \), and histories \( s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0 \).
If a reinforcement learning task has the Markov Property, it is basically a Markov Decision Process (MDP).

If state and action sets are finite, it is a finite MDP.

To define a finite MDP, you need to give:

- state and action sets
- one-step “dynamics” defined by transition probabilities:
  \[
  P_{ss'}^a = \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\} \quad \text{for all } s, s' \in S, a \in A(s).
  \]

- reward probabilities:
  \[
  R_{ss'}^a = E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\} \quad \text{for all } s, s' \in S, a \in A(s).
  \]
An Example Finite MDP

Recycling Robot

- At each step, robot has to decide whether it should (1) actively search for a can, (2) wait for someone to bring it a can, or (3) go to home base and recharge.
- Searching is better but runs down the battery; if runs out of power while searching, has to be rescued (which is bad).
- Decisions made on basis of current energy level: high, low.
- Reward = number of cans collected
Recycling Robot MDP

\[ S = \{ \text{high}, \text{low} \} \]

\[ A(\text{high}) = \{ \text{search}, \text{wait} \} \]

\[ A(\text{low}) = \{ \text{search}, \text{wait}, \text{recharge} \} \]

\[ R^{\text{search}} = \text{expected no. of cans while searching} \]

\[ R^{\text{wait}} = \text{expected no. of cans while waiting} \]

\[ R^{\text{search}} > R^{\text{wait}} \]
Value Functions

- The **value of a state** is the expected return starting from that state; depends on the agent’s policy:

\[
V^\pi(s) = E^\pi \left\{ R_t \mid s_t = s \right\} = E^\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s \right\}
\]

- The **value of taking an action in a state under policy** \( \pi \) is the expected return starting from that state, taking that action, and thereafter following \( \pi \):

\[
Q^\pi(s, a) = E^\pi \left\{ R_t \mid s_t = s, a_t = a \right\} = E^\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a \right\}
\]
Bellman Equation for a Policy $\pi$

The basic idea:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} \cdots$$

$$= r_{t+1} + \gamma \left( r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} \cdots \right)$$

$$= r_{t+1} + \gamma R_{t+1}$$

So:

$$V^\pi(s) = E_\pi \{R_t \mid s_t = s\}$$

$$= E_\pi \{r_{t+1} + \gamma V(s_{t+1}) \mid s_t = s\}$$

Or, without the expectation operator:

$$V^\pi(s) = \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a \left[ R_{ss'}^a + \gamma V^\pi(s') \right]$$
More on the Bellman Equation

$$V^\pi(s) = \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a \left[ R_{ss'}^a + \gamma V^\pi(s') \right]$$

This is a set of equations (in fact, linear), one for each state. The value function for $\pi$ is its unique solution.

Backup diagrams:

(a) $s$  
\[\begin{array}{c}
\bullet \\
\bullet \\
\bullet \\
\bullet
\end{array}\]

(b) $s, a$
\[\begin{array}{c}
\bullet \\
\bullet \\
\bullet
\end{array}\
\begin{array}{c}
\bullet \\
\bullet
\end{array}
\begin{array}{c}
\bullet \\
\bullet
\end{array}
\begin{array}{c}
\bullet \\
\bullet
\end{array}
\begin{array}{c}
\bullet \\
\bullet
\end{array}
\]

for $V^\pi$

for $Q^\pi$
Gridworld

- Actions: north, south, east, west; deterministic.
- If would take agent off the grid: no move but reward = −1
- Other actions produce reward = 0, except actions that move agent out of special states A and B as shown.

State-value function for equiprobable random policy; \( \gamma = 0.9 \)
Golf

- State is ball location
- Reward of $-1$ for each stroke until the ball is in the hole
- Value of a state?
- Actions:
  - *putt* (use putter)
  - *driver* (use driver)
- *putt* succeeds anywhere on the green
Optimal Value Functions

- For finite MDPs, policies can be partially ordered:
  \[ \pi \geq \pi' \text{ if and only if } V^{\pi}(s) \geq V^{\pi'}(s) \text{ for all } s \in S \]

- There are always one or more policies that are better than or equal to all the others. These are the optimal policies. We denote them all as \( \pi^* \).

- Optimal policies share the same optimal state-value function:
  \[ V^*(s) = \max_{\pi} V^{\pi}(s) \text{ for all } s \in S \]

- Optimal policies also share the same optimal action-value function:
  \[ Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a) \text{ for all } s \in S \text{ and } a \in A(s) \]
  This is the expected return for taking action \( a \) in state \( s \) and thereafter following an optimal policy.
Optimal Value Function for Golf

- We can hit the ball farther with driver than with putter, but with less accuracy
- $Q^*(s, \text{driver})$ gives the value or using driver first, then using whichever actions are best
Bellman Optimality Equation for $V^*$

The value of a state under an optimal policy must equal the expected return for the best action from that state:

$$V^*(s) = \max_{a \in A(s)} Q^\pi^*(s, a)$$

$$= \max_{a \in A(s)} E \left\{ r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a \right\}$$

$$= \max_{a \in A(s)} \sum_{s'} P_{ss'}^a \left[ R_{ss'}^a + \gamma V^*(s') \right]$$

The relevant backup diagram:

$V^*$ is the unique solution of this system of nonlinear equations.
Bellman Optimality Equation for $Q^*$

$$Q^*(s,a) = E \left\{ r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1},a') \mid s_t = s, a_t = a \right\}$$

$$= \sum_{s'} P_{ss'}^a \left[ R_{ss'}^a + \gamma \max_{a'} Q^*(s',a') \right]$$

The relevant backup diagram:

$Q^*$ is the unique solution of this system of nonlinear equations.
Why Optimal State-Value Functions are Useful

Any policy that is greedy with respect to $V^*$ is an optimal policy.

Therefore, given $V^*$, one-step-ahead search produces the long-term optimal actions.

E.g., back to the gridworld:

![Gridworld and Value Function](image)

- **a)** gridworld
- **b)** $V^*$
- **c)** $\pi^*$
What About Optimal Action-Value Functions?

Given $Q^*$, the agent does not even have to do a one-step-ahead search:

$$\pi^*(s) = \arg \max_{a \in A(s)} Q^*(s, a)$$
Solving the Bellman Optimality Equation

Finding an optimal policy by solving the Bellman Optimality Equation requires the following:
- accurate knowledge of environment dynamics;
- we have enough space and time to do the computation;
- the Markov Property.

How much space and time do we need?
- polynomial in number of states (via dynamic programming methods; Chapter 4),
- BUT, number of states is often huge (e.g., backgammon has about $10^{20}$ states).

We usually have to settle for approximations.

Many RL methods can be understood as approximately solving the Bellman Optimality Equation.
Summary

- Agent-environment interaction
  - States
  - Actions
  - Rewards
- Policy: stochastic rule for selecting actions
- Return: the function of future rewards agent tries to maximize
- Episodic and continuing tasks
- Markov Property
- Markov Decision Process
  - Transition probabilities
  - Expected rewards
- Value functions
  - State-value function for a policy
  - Action-value function for a policy
  - Optimal state-value function
  - Optimal action-value function
- Optimal value functions
- Optimal policies
- Bellman Equations
- The need for approximation