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.
Reinforcement Learning (RL)

- **RL**: A class of learning problems in which an agent interacts with a dynamic, stochastic, and incompletely known environment in order to achieve a goal.
The Agent-Environment Interface

Agent and environment interact at discrete time steps $t = 0, 1, 2, K$

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: $s_{t+1}$
Selective Perception and Hidden State

- An agent interacts with its environment through its sensors and actuators.

- An agent often suffers from two opposite types of perceptual limitations:
  - Too little sensory data (hidden state)
    - Often can be solved by context or memory selective attention
    - Selective attention: what to remember, what to forget
  - Too much sensory data
    - Often can be solved by selective perception
    - Selective perception is like creating hidden states on purpose
Selective Perception and Hidden State

- **Selective perception - selective attention:** agent chooses which features - from present and past sensory data - it will attend to

- **Attend to a feature:** agent distinguishes between situations in which that feature is present and absent *(making distinction)*

- **Agent internal state:** cross product of all distinctions chosen by the agent
  - Agent must find those distinctions (features) relevant to its task at hand
    - difficult - sometimes the agent or its designer may get it wrong
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)
Returns

Suppose the sequence of rewards
\[ r_1, r_2, r_3, \ldots \]
What do we want to maximize?

In general,
we want to maximize the expected return, \( E\{R_0\} \).

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

\[ R_0 = r_1 + r_2 + \cdots + r_T, \]

where \( T \) is a final time step at which a terminal state is reached, ending an episode.
Returns for Continuing Tasks

Continuing tasks: interaction does not have natural episodes.

Discounted return:

\[ R_0 = r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots = \sum_{t=0}^{\infty} \gamma^t r_{t+1}, \]

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

shortsighted 0 ← \( \gamma \) → 1 farsighted
Think of each episode as ending in an absorbing state that always produces reward of zero:

\[ R_0 = \sum_{t=0}^{\infty} \gamma^t r_{t+1}, \quad \text{where } \gamma \]

can be 1 only if a zero reward absorbing state is always reached.
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:
  \[ \text{reward} = -1 \text{ upon failure; 0 otherwise} \]
  \[ \Rightarrow \text{return} = -\gamma^t, \text{ for } t \text{ steps before failure} \]

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 \textbf{not} at top of hill

$\Rightarrow$ return = $-\text{number of steps before reaching top of hill}$

Return is maximized by minimizing number of steps to reach the top of the hill.
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**:

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

for all $s', r$, and histories $s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0$. 
Reinforcement Learning (RL)

- **RL**: a class of learning problems in which an agent interacts with an unfamiliar, dynamic and stochastic environment

- **Goal**: Learn a policy to maximize some measure of long-term reward

- **Interaction**: modeled as a MDP or POMDP
A MDP is defined as a 5-tuple \((S, A, q, P, P_0)\)

- **\(S\)**: state space of the process
- **\(A\)**: action space of the process
- **\(P(\cdot \mid s, a)\)**: probability distribution over next state
  \[
  P(s' \mid s, a) = \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\}
  \]
  for all \(s, s' \in S, a \in A(s)\).
- **\(q(\cdot \mid s, a)\)**: probability distribution over rewards
  \[
  r(s,a) = E\{r_{t+1} \mid s_t = s, a_t = a\}
  \]
  for all \(s \in S, a \in A(s)\).
- **\(P_0\)**: initial state distribution
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

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

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

\[ R^\text{search} > R^\text{wait} \]
Policy and Return

- **A Stationary Policy:** a time-independent mapping from states to actions or distributions over actions

  \[ \pi(s) \in \mathcal{A} \quad \text{or} \quad \pi(a|s) \in \Pr(\mathcal{A}) \]

- **Discounted Return:** a random process (an indexed set of random variables), discounted return for state \( s \) under policy \( \pi \) is a random variable defined as

  \[ R^\pi(s) = \sum_{t=0}^{\infty} \gamma^t r_{t+1} \]

  where

  \[ s_0 = s, \quad a_t \sim \pi(\cdot|s_t), \quad r_{t+1} \sim q(\cdot|s_t, a_t), \quad s_{t+1} \sim P(\cdot|s_t, a_t) \]
Value Functions

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

\[
V^\pi(s) = \mathbb{E}\{R^\pi(s)\} = \mathbb{E}\left\{\sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid s_0 = s, \pi\right\}
\]

- 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 \(\pi\):

\[
Q^\pi(s,a) = \mathbb{E}\{R^\pi(s,a)\} = \mathbb{E}\left\{\sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid s_0 = s, a_0 = a, \pi\right\}
\]
Bellman Equation for $V^\pi$

Show

$$V^\pi(s) = r(s, \pi(s)) + \gamma \sum_{s'} P(s'|s, \pi(s)) V^\pi(s')$$

Proof:

$$V^\pi(s) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r(s_t, \pi(s_t)) | s_0 = s, \pi \right]$$

$$= r(s, \pi(s)) + \mathbb{E} \left[ \sum_{t \geq 1} \gamma^t r(s_t, \pi(s_t)) | s_0 = s, \pi \right]$$

$$= r(s, \pi(s)) + \gamma \sum_{s'} \Pr(s_1 = s'|s_0 = s, \pi(s_0)) \mathbb{E} \left[ \sum_{t \geq 1} \gamma^{t-1} r(s_t, \pi(s_t)) | s_1 = s', \pi \right]$$

$$= r(s, \pi(s)) + \gamma \sum_{s'} P(s'|s, \pi(s)) V^\pi(s').$$

When both reward and policy are stochastic, we have

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left[ r(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^\pi(s') \right]$$
More on the Bellman Equation

\[ V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} \left[ R^a_{ss'} + \gamma V^\pi(s') \right] \]

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

Similarly Bellman equation for \( Q^\pi \) may be written as

**Bellman Equation for \( Q^\pi \)**

\[ Q^\pi(s, a) = r(s, a) + \sum_{s' \in S} P(s'| s, a) \sum_{a' \in A} \pi(a'| s') Q^\pi(s', a') \]
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; $\gamma = 0.9$.
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 \( \pi^* \).

- 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.
Bellman Optimality Equation for $V^*$

Show $V^*(s) = \max_a \left[ r(s, a) + \gamma \sum_{s'} P(s'|s, a)V^*(s') \right]$ Bellman Optimality Equation for $V$

proof:

$V^*(s) = \max_\pi \mathbb{E}\left[ \sum_{t=0}^{\infty} \gamma^t r(s_t, \pi(s_t)) | s_0 = s, \pi \right]$

$\max_\pi \sum_{s'} P(s'|s, a)V^\pi(s) \leq \sum_{s'} P(s'|s, a) \max_\pi V^\pi(s)$

Let $\pi'$ be $\pi'(s') = \arg \max_\pi V^\pi(s')$

$\sum_{s'} P(s'|s, a) \max_\pi V^\pi(s') = \sum_{s'} P(s'|s, a)V^{\pi'}(s')$

$\leq \max_\pi \sum_{s'} P(s'|s, a)V^\pi(s')$

$= \max_a \left[ r(s, a) + \gamma \sum_{s'} P(s'|s, a)V^\pi(s') \right]$
Bellman Optimality Equation for $Q^*$

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

$Q^*$ 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.

$$
\pi^*(s) = \arg\max_{a \in \mathcal{A}} \left[ r(s, a) + \sum_{s' \in \mathcal{S}} P(s'|s,a)V^*(s') \right]
$$

E.g., back to the gridworld:

a) gridworld  

\[
\begin{array}{c|cccc}
   & 22.0 & 24.4 & 22.0 & 19.4 \\
   & 19.8 & 22.0 & 19.8 & 17.5 \\
   & 17.8 & 19.8 & 17.8 & 16.0 \\
   & 16.0 & 17.8 & 16.0 & 14.4 \\
   & 14.4 & 16.0 & 14.4 & 13.0 \\
\end{array}
\]

b) $V^*$

\[
\begin{array}{c|cccc}
   & \uparrow & \uparrow & \uparrow & \leftarrow \\
   & \uparrow & \uparrow & \uparrow & \leftarrow \\
   & \uparrow & \uparrow & \uparrow & \leftarrow \\
   & \uparrow & \uparrow & \uparrow & \leftarrow \\
   & \uparrow & \uparrow & \uparrow & \leftarrow \\
\end{array}
\]

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