Many of the slides in this presentation are from R. Sutton and A. Barto, as noted at the bottom of the slides
The Agent-Environment Interface

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: $s_{t+1}$

---

R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
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.
Goals and Rewards

☐ 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 maximizing a received scalar signal
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.
Returns for Continuing Tasks

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:

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

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

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.

\[
\text{reward} = -1 \text{ for each step where not at top of hill} \\
\Rightarrow \text{return} = - \text{number of steps before reaching top of hill}
\]

Return is maximized by minimizing number of steps to reach the top of the hill.
A Unified Notation

- 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 a reward of zero:

![Diagram](image)

- 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 \( s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0 \).
Markov Decision Processes

- 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:
    \[ P^a_{ss'} = Pr \{ s_{t+1} = s' \mid s_t = s, a_t = a \} \] for all \( s, s' \in S, a \in A(s) \).
  - expected rewards:
    \[ R^a_{ss'} = E \{ r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s' \} \] 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 it 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

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

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

\[ R^\text{search} > R^\text{wait} \]
The value-function hypothesis

- All efficient methods for solving sequential decision problems determine (learn or compute) value functions as an intermediate step
Value Functions

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

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

- 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 \( \pi \):

  \[
  Q^\pi(s, a) = E_\pi\{R_t | s_t = s, a_t = a\} = E_\pi\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \bigg| s_t = s, a_t = a \right\}
  \]
Optimal Value Functions

- For finite MDPs, policies can be partially ordered: \( \pi \succeq \pi' \) if and only if \( V^\pi(s) \geq V^{\pi'}(s) \) 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: \( V^*(s) = \max_{\pi} V^\pi(s) \) for all \( s \in S \).

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

Robot Pseudocode

Leader:

loop: on valid input signal from environment
    choose an action to perform
    choose a signal to send to follower via
    the radio boards
    wait for reinforcement signal
    on reinforcement signal
    increment variables for action and signal
    goto loop

Follower:

loop: on valid input signal from leader
    choose an action to perform
    wait for reinforcement signal
    on reinforcement signal
    increment variables for action
    goto loop
**Example run with 2 robots**

<table>
<thead>
<tr>
<th></th>
<th>Appropriate action</th>
<th>Leader’s action</th>
<th>Leader’s signal</th>
<th>Follower’s action</th>
<th>Reinforcement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>↑↑</td>
<td>spin</td>
<td>low</td>
<td>spin</td>
<td>−</td>
</tr>
<tr>
<td>2.</td>
<td>□□□</td>
<td>spin</td>
<td>low</td>
<td>straight</td>
<td>−</td>
</tr>
<tr>
<td>3.</td>
<td>↑↑</td>
<td>straight</td>
<td>high</td>
<td>spin</td>
<td>−</td>
</tr>
<tr>
<td>4.</td>
<td>□□□</td>
<td>straight</td>
<td>high</td>
<td>straight</td>
<td>−</td>
</tr>
<tr>
<td>5.</td>
<td>□□□</td>
<td>spin</td>
<td>low</td>
<td>spin</td>
<td>+</td>
</tr>
<tr>
<td>6.</td>
<td>↑↑</td>
<td>straight</td>
<td>high</td>
<td>spin</td>
<td>−</td>
</tr>
<tr>
<td>7.</td>
<td>□□□</td>
<td>spin</td>
<td>low</td>
<td>spin</td>
<td>+</td>
</tr>
<tr>
<td>8.</td>
<td>□□□</td>
<td>spin</td>
<td>low</td>
<td>spin</td>
<td>+</td>
</tr>
<tr>
<td>9.</td>
<td>□□□</td>
<td>spin</td>
<td>low</td>
<td>spin</td>
<td>+</td>
</tr>
<tr>
<td>10.</td>
<td>↑↑</td>
<td>spin</td>
<td>low</td>
<td>spin</td>
<td>−</td>
</tr>
<tr>
<td>11.</td>
<td>↑↑</td>
<td>straight</td>
<td>high</td>
<td>straight</td>
<td>+</td>
</tr>
<tr>
<td>12.</td>
<td>↑↑</td>
<td>straight</td>
<td>high</td>
<td>straight</td>
<td>+</td>
</tr>
<tr>
<td>13.</td>
<td>□□□</td>
<td>spin</td>
<td>low</td>
<td>spin</td>
<td>+</td>
</tr>
</tbody>
</table>
Curse of dimensionality

<table>
<thead>
<tr>
<th>Size of Language</th>
<th>Number of Iterations to Convergence</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>15.34</td>
<td>10</td>
<td>24</td>
</tr>
<tr>
<td>3</td>
<td>110.30</td>
<td>33</td>
<td>501</td>
</tr>
<tr>
<td>4</td>
<td>340.38</td>
<td>53</td>
<td>990</td>
</tr>
<tr>
<td>5</td>
<td>906.62</td>
<td>255</td>
<td>2472</td>
</tr>
<tr>
<td>10</td>
<td>15011.61</td>
<td>2868</td>
<td>51031</td>
</tr>
<tr>
<td>20</td>
<td>232267.82</td>
<td>44196</td>
<td>1241767</td>
</tr>
</tbody>
</table>

Table 2: Learning times for a two member troupe. Experiments for each language size were run 100 times.

<table>
<thead>
<tr>
<th>Size of Language</th>
<th>Number of Iterations to Convergence</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>27.21</td>
<td>10</td>
<td>80</td>
</tr>
<tr>
<td>3</td>
<td>327.71</td>
<td>35</td>
<td>1211</td>
</tr>
<tr>
<td>4</td>
<td>1530.12</td>
<td>340</td>
<td>6666</td>
</tr>
<tr>
<td>5</td>
<td>4415.60</td>
<td>652</td>
<td>17533</td>
</tr>
<tr>
<td>10</td>
<td>163530.62</td>
<td>37130</td>
<td>705029</td>
</tr>
</tbody>
</table>

Table 3: Data above is for a three member troupe and was collected over 100 runs for each language size.