Reinforcement Learning
Markov Decision Processes

Stefano V. Albrecht, Michael Herrmann
23 January 2024
Lecture Outline

- Markov decision process
- Policies, goals, rewards, returns
- Value functions and Bellman equation
- Optimal value functions and policies
Agent and environment interact at discrete time steps: $t = 0, 1, 2, 3, ...$

- Agent observes environment state at time $t$: $S_t \in S$
- and selects an action at step $t$: $A_t \in \mathcal{A}$
- Environment sends back reward $R_{t+1} \in \mathcal{R}$ and new state $S_{t+1} \in S$
The Agent-Environment Interface

Agent → Environment

Agent receives state $S_t$ and reward $R_t$ from Environment.

Environment sends action $A_t$ to Agent.

Agent receives state $S_{t+1}$ and reward $R_{t+1}$ from Environment.

Diagram:

- State sequence: $S_t, S_{t+1}, S_{t+2}, S_{t+3}, \ldots$
- Action sequence: $A_t, A_{t+1}, A_{t+2}, A_{t+3}, \ldots$
- Reward sequence: $R_{t+1}, R_{t+2}, R_{t+3}, \ldots$
Markov Decision Process

Markov decision process (MDP) consists of:

- State space $S$
- Action space $A$
- Reward space $\mathcal{R}$
- Environment dynamics:

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

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

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

MDP is finite if $S, A, \mathcal{R}$ are finite.
Markov property:
Future state and reward are independent of past states and actions, *given the current state and action*:

\[
\Pr\{S_{t+1}, R_{t+1} \mid S_t, A_t, S_{t-1}, A_{t-1}, \ldots, S_0, A_0\} = \Pr\{S_{t+1}, R_{t+1} \mid S_t, A_t\}
\]

- State $S_t$ is *sufficient summary* of interaction history
  \[\Rightarrow\] Means optimal decision in $S_t$ does not depend on past decisions
- Designing compact Markov states is “engineering work” in RL
Example: Recycling Robot

- Mobile robot must collect cans in office
- States:
  - high battery level
  - low battery level
- Actions:
  - search for can
  - wait for someone to bring can
  - recharge battery at charging station
- Rewards: number of cans collected
### Example: Recycling Robot

<table>
<thead>
<tr>
<th>$s$</th>
<th>$a$</th>
<th>$s'$</th>
<th>$p(s' \mid s, a)$</th>
<th>$r(s, a, s')$</th>
</tr>
</thead>
<tbody>
<tr>
<td>high</td>
<td>search</td>
<td>high</td>
<td>$\alpha$</td>
<td>$r_{\text{search}}$</td>
</tr>
<tr>
<td>high</td>
<td>search</td>
<td>low</td>
<td>$1 - \alpha$</td>
<td>$r_{\text{search}}$</td>
</tr>
<tr>
<td>low</td>
<td>search</td>
<td>high</td>
<td>$1 - \beta$</td>
<td>$-3$</td>
</tr>
<tr>
<td>low</td>
<td>search</td>
<td>low</td>
<td>$\beta$</td>
<td>$r_{\text{search}}$</td>
</tr>
<tr>
<td>high</td>
<td>wait</td>
<td>high</td>
<td>1</td>
<td>$r_{\text{wait}}$</td>
</tr>
<tr>
<td>high</td>
<td>wait</td>
<td>low</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>low</td>
<td>wait</td>
<td>high</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>low</td>
<td>wait</td>
<td>low</td>
<td>1</td>
<td>$r_{\text{wait}}$</td>
</tr>
<tr>
<td>low</td>
<td>recharge</td>
<td>high</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>low</td>
<td>recharge</td>
<td>low</td>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>
Example: Recycling Robot
MDP is controlled with a policy:

\[ \pi(a|s) = \text{probability of selecting action } a \text{ when in state } s \]

| \( \pi(a|s) \) | search | wait | recharge |
|----------------|--------|------|----------|
| high           | 0.9    | 0.1  | 0        |
| low            | 0.2    | 0.3  | 0.5      |

Special case: deterministic policy \( \pi(s) = a \)

Remark: MDP coupled with fixed policy \( \pi \) is a “Markov chain”
Agent’s goal is to learn a policy that maximises cumulative reward

**Reward hypothesis:**
All goals can be described by the maximisation of the expected value of cumulative scalar rewards.

Rewards specify *what* the goal is

- Rewards do not specify *how* to achieve goal
- But if done carefully, good reward design may help to learn faster
  - Like state design, reward design is “engineering work” in RL
Formally, policy should maximise expected return:

\[ G_t = R_{t+1} + R_{t+2} + R_{t+3} + \ldots + R_T \]

\[ = R_{t+1} + G_{t+1} \]

where \( T \) is final time step

Assumes *terminating* episodes:

- e.g. Chess game: terminates when one player wins
- e.g. Furniture building: terminates when furniture completed
- Can enforce termination by setting number of allowed time steps
Discounted Return

For non-terminating (infinite) episodes, can use discount rate $\gamma \in [0, 1)$:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+1+k}$$

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

- e.g. Financial portfolio management
- e.g. Server monitoring and maintenance

low $\gamma$ is shortsighted
high $\gamma$ is farsighted
Discounted Return

For non-terminating (infinite) episodes, can use discount rate $\gamma \in [0, 1)$:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+1+k}$$

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

- Sum is finite for $\gamma < 1$ and bounded rewards $R_t \leq r_{\text{max}}$

$$\sum_{k=0}^{\infty} \gamma^k R_{t+1+k} \leq r_{\text{max}} \sum_{k=0}^{\infty} \gamma^k = r_{\text{max}} \frac{1}{1-\gamma}$$

low $\gamma$ is shortsighted

high $\gamma$ is farsighted
Discounted Return

For non-terminating (infinite) episodes, can use discount rate $\gamma \in [0, 1)$:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+1+k}$$

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

- Sum is finite for $\gamma < 1$ and bounded rewards $R_t \leq r_{\text{max}}$:

$$\sum_{k=0}^{\infty} \gamma^k R_{t+1+k} \leq r_{\text{max}} \sum_{k=0}^{\infty} \gamma^k = r_{\text{max}} \frac{1}{1-\gamma}$$

- Definition also works for terminating episodes if terminal states are “absorbing”: absorbing state always transitions into itself and gives reward 0
State Value Function

Given policy $\pi$, can quantify expected return in any state $s$ with state-value function:

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... | S_t = s]$$
State Value Function

Given policy $\pi$, can quantify expected return in any state $s$ with state-value function:

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... | S_t = s]$$

In general, assuming terminating episodes (need more math for non-terminating episodes), let $\mathcal{H}(s)$ be the set of all possible episodes starting in $s$:

$$\mathcal{H}(s) = \{ h = (s^t, a^t, r^{t+1}, s^{t+1}, a^{t+1}, r^{t+2}, s^{t+2}, ..., r^T, s^T) | s^t = s \}$$
State Value Function

Given policy \( \pi \), can quantify expected return in any state \( s \) with state-value function:

\[
v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | S_t = s]
\]

In general, assuming terminating episodes (need more math for non-terminating episodes), let \( \mathcal{H}(s) \) be the set of all possible episodes starting in \( s \):

\[
\mathcal{H}(s) = \{ h = (s^t, a^t, r^{t+1}, s^{t+1}, a^{t+1}, r^{t+2}, s^{t+2}, \ldots, r^T, s^T) | s^t = s \}
\]

Each \( h \in \mathcal{H}(s) \) has associated probability of occurring and cumulative reward:

\[
\Pr(h|\pi) = \prod_{\tau=t}^{T-1} \pi(a^\tau|s^\tau) \rho(s^{\tau+1}, r^{\tau+1}|s^\tau, a^\tau) \quad \text{and} \quad G(h) = \sum_{\tau=t}^{T} \gamma^{\tau-t} r^\tau
\]
Given policy $\pi$, can quantify expected return in any state $s$ with state-value function:

$$v_{\pi}(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... | S_t = s]$$

In general, assuming terminating episodes (need more math for non-terminating episodes), let $\mathcal{H}(s)$ be the set of all possible episodes starting in $s$:

$$\mathcal{H}(s) = \{ h = (s^t, a^t, r^{t+1}, s^{t+1}, a^{t+1}, r^{t+2}, s^{t+2}, ..., r^T, s^T) | s^t = s \}$$

Each $h \in \mathcal{H}(s)$ has associated probability of occurring and cumulative reward:

$$\Pr(h|\pi) = \prod_{\tau=t}^{T-1} \pi(a^\tau|s^\tau) \rho(s^{\tau+1}, r^{\tau+1}|s^\tau, a^\tau) \quad \text{and} \quad G(h) = \sum_{\tau=t}^{T} \gamma^{T-\tau} r^\tau$$

Then compute state value as $v_\pi(s) = \sum_{h \in \mathcal{H}(s)} \Pr(h|\pi) G(h)$
Because of Markov property, can write state-value function in recursive form with Bellman equation:

\[ v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] \]

Markov: past states/actions don’t matter given current state
Because of Markov property, can write state-value function in recursive form with Bellman equation:

$$v_{\pi}(s) \triangleq \mathbb{E}_{\pi}[G_t|S_t = s]$$

$$= \mathbb{E}_{\pi}[R_{t+1} + \gamma G_{t+1}|S_t = s]$$

*Markov: past states/actions don’t matter given current state*
State Value Function and the Bellman equation

Because of Markov property, can write state-value function in recursive form with Bellman equation:

\[ v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] \]

Markov: past states/actions don’t matter given current state

\[ = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] \]

\[ = \sum_a \pi(a | s) \sum_{s', r} p(s', r | a, s) \left[ r + \gamma \mathbb{E}_\pi[G_{t+1} | S_{t+1} = s'] \right] \]
Because of Markov property, can write state-value function in recursive form with Bellman equation:

\[ v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] \]

\[ = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] \]

\[ = \sum_a \pi(a|s) \sum_{s', r} p(s', r|a, s) \left[ r + \gamma \mathbb{E}_\pi[G_{t+1} | S_{t+1} = s'] \right] \]

\[ = \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a) \left[ r + \gamma v_\pi(s') \right] \]

Markov: past states/actions don’t matter given current state
State Value Function and the Bellman equation

Because of Markov property, can write state-value function in recursive form with Bellman equation:

\[ v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] \]

\[ = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] \]

\[ = \sum_a \pi(a|s) \sum_{s', r} p(s', r|a, s) \left[ r + \gamma \mathbb{E}_\pi[G_{t+1} | S_{t+1} = s'] \right] \]

\[ = \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a) \left[ r + \gamma v_\pi(s') \right] \]

Markov: past states/actions don’t matter given current state

One-step look-ahead tree
Because of Markov property, can write state-value function in recursive form with Bellman equation:

\[ v_\pi(s) = \mathbb{E}_\pi \left[ G_t | S_t = s \right] \]

\[ = \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a) \left[ r + \gamma v_\pi(s') \right] \]
Because of Markov property, can write state-value function in recursive form with Bellman equation:

\[
v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] \\
= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) [r + \gamma v_\pi(s')]
\]

Can also define action-value function:

\[
q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \\
= \sum_{s',r} p(s', r|s, a) [r + \gamma v_\pi(s')]
\]
Policy $\pi$ is optimal if

$$v_\pi(s) = v_*(s) = \max_{\pi'} v_{\pi'}(s)$$

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

Because of the Bellman equation, this means that for any optimal policy $\pi$:

$$\forall \hat{\pi} \forall s : v_\pi(s) \geq v_{\hat{\pi}}(s)$$
Optimal Value Functions and Policies

We can write optimal value function without reference to policy:

\[ v_*(s) = \max_a \sum_{s', r} p(s', r|s, a) \left[ r + \gamma v_*(s') \right] \]

\[ q_*(s, a) = \sum_{s', r} p(s', r|s, a) \left[ r + \gamma \max_{a'} q_*(s', a') \right] \]

Bellman optimality equations
Example: Gridworld

Gridworld:

- States: cell location in grid
- Actions: move north, south, east, west
- Rewards: -1 if off-grid, +10/+5 if in A/B, 0 otherwise

State-value function $v_\pi(s)$ for policy $\pi(a|s) = \frac{1}{4}$ for all $s, a$, with $\gamma = 0.9$
Example: Gridworld

Gridworld:

- States: cell location in grid
- Actions: move north, south, east, west
- Rewards: -1 if off-grid, +10/+5 if in A/B, 0 otherwise

Optimal policy and state-value function
Solving the Bellman Equation

Bellman equation for $v_\pi$ forms a system of $n$ linear equations with $n$ variables, where $n$ is number of states (for finite MDP):

\[
v_\pi(s_1) = \sum_a \pi(a|s_1) \sum_{s',r} p(s', r|s_1, a) \left[ r + \gamma v_\pi(s') \right]
\]

\[
v_\pi(s_2) = \sum_a \pi(a|s_2) \sum_{s',r} p(s', r|s_2, a) \left[ r + \gamma v_\pi(s') \right]
\]

\[\vdots\]

\[
v_\pi(s_n) = \sum_a \pi(a|s_n) \sum_{s',r} p(s', r|s_n, a) \left[ r + \gamma v_\pi(s') \right]
\]

- Value function $v_\pi$ is unique solution to system
- Solve for $v_\pi$ with any method to solve linear systems (e.g. Gauss elimination)
Solving the Bellman Optimality Equation

Bellman optimality equation for \( v_\star \) forms a system of \( n \) non-linear equations with \( n \) variables

- Equations are non-linear due to \text{max} operator
- Optimal value function \( v_\star \) is unique solution to system
- Solve for \( v_\star \) with any method to solve non-linear equation systems

Can solve related set of equations for \( q_\pi / q_\star \)

\[ \text{Once we have } v_\star \text{ or } q_\star, \text{ we know optimal policy } \pi_\star \text{ (why?)} \]
Solving for $v_*$ in recycling robot example (states: h/l, actions: s, w, re):

$$v_*(h) = \max \left\{ \begin{array}{l}
p(h|h, s)[r(h, s, h) + \gamma v_*(h)] + p(l|h, s)[r(h, s, l) + \gamma v_*(l)], \\
p(h|h, w)[r(h, w, h) + \gamma v_*(h)] + p(l|h, w)[r(h, w, l) + \gamma v_*(l)]
\end{array} \right\}$$

$$= \max \left\{ \begin{array}{l}
\alpha[r_s + \gamma v_*(h)] + (1 - \alpha)[r_s + \gamma v_*(l)], \\
1[r_w + \gamma v_*(h)] + 0[r_w + \gamma v_*(l)]
\end{array} \right\}$$

$$= \max \left\{ \begin{array}{l}
r_s + \gamma[\alpha v_*(h) + (1 - \alpha)v_*(l)], \\
r_w + \gamma v_*(h)
\end{array} \right\}.$$ 

$$v_*(l) = \max \left\{ \begin{array}{l}
\beta r_s - 3(1 - \beta) + \gamma[(1 - \beta)v_*(h) + \beta v_*(l)], \\
r_w + \gamma v_*(l), \\
\gamma v_*(h)
\end{array} \right\}$$

Choose numbers for $r_s, r_w, \alpha, \beta, \gamma$ and solve for unique $v_*(h) / v_*(l)$ pair.
• Markov decision process is the fundamental model in RL

• MDPs can be solved exactly if we know all components of the MDP (i.e. $S, A, R, p(s', r|a, s)$)
  ⇒ But number of states/actions is problem for scalability

• We will discuss RL techniques which learn optimal policy by interacting with MDP
  ⇒ Methods try to find good approximate solutions with reasonable effort
Required:

- RL book, Chapter 3 (3.1–3.7)

Optional:

- *Dynamic Programming*  
  by Richard Bellman (university library has copies)
- *Markov Decision Processes: Discrete Stochastic Dynamic Programming*  
  by Martin Puterman (university library has copies)
For finite MDP and non-terminating episode, any policy $\pi$ will produce an ergodic set of states $\hat{S}$:

- Every state in $\hat{S}$ visited infinitely often
- Steady-state distribution: $P_\pi(s) = \lim_{t \to \infty} \Pr\{S_t = s \mid A_0, \ldots, A_{t-1} \sim \pi\}$

Performance of $\pi$ can be measured by average reward:

$$r(\pi) \doteq \lim_{h \to \infty} \frac{1}{h} \sum_{t=1}^{h} \mathbb{E}[R_t \mid S_0, A_0, \ldots, A_{t-1} \sim \pi]$$

$$= \sum_s P_\pi(s) \sum_a \pi(a \mid s) \sum_{s', r} p(s', r \mid s, a) r$$

Independent of initial state $S_0$!
Maximising discounted return over steady-state dist. is same as maximising average reward!

\[
\sum_s P_\pi(s) v_\pi(s) = \sum_s P_\pi(s) \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) [r + \gamma v_\pi(s')] \\
= r(\pi) + \sum_s P_\pi(s) \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) [\gamma v_\pi(s')] \\
= r(\pi) + \gamma \sum_{s'} P_\pi(s') v_\pi(s') \\
= r(\pi) + \gamma [r(\pi) + \gamma \sum_{s'} P_\pi(s') v_\pi(s')] \\
= r(\pi) + \gamma r(\pi) + \gamma^2 r(\pi) + \gamma^3 r(\pi) + \cdots \\
= r(\pi) \frac{1}{1 - \gamma} \quad \Rightarrow \gamma \text{ has no effect on maximisation!}
\]
We will focus on discounted return since:

- Most of current RL theory was developed for discounted return
- Discounted and average setting give same limit results for $\gamma \to 1$
  $\Rightarrow$ This is why most often people use $\gamma \in [0.95, 0.99]$
- Discounted return works well for finite and infinite episodes