Multi-Agent Systems

Introduction to Reinforcement Learning:

Part 2: Model-based Prediction and Control

Eric Pauwels (CWI & VU)

December 7, 2023

Reading

- Sutton & Barto: chapters 3 & 4

Outline

Optimal Policy and Bellman Optimality Equations

Taxonomy of RL problems

Model-based Prediction and Control

Optimal value functions

- Value functions define a partial ordering over policies:

$$\pi \succ \pi' \quad \iff \quad v_\pi(s) \geq v_{\pi'}(s), \ \forall s \in S$$

- There can be multiple optimal policies but they all share the same **optimal state-value function**:

$$v^*(s) = \max_\pi v_\pi(s), \quad \forall s \in S$$

- They also share the same **optimal action-value function**:

$$q^*(s, a) = \max_\pi q_\pi(s, a), \quad \forall s \in S, a \in A$$

Backup Diagram for Bellman Optimality Equations

Optimize over actions!

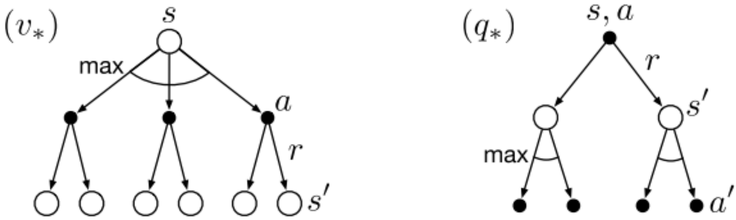


Figure 3.5: Backup diagrams for v_* and q_*

Bellman optimality equation in matrix form

$$
\begin{aligned}
v^*(s) &= \max_{a \in A} \sum_{s'} p(s' \mid s, a) \left[r(s, a, s') + \gamma v^*(s') \right] \\
&= \max_a \left(R(s, a) + \gamma \sum_{s'} \underbrace{p(s' \mid s, a)}_{T_a(s,s')} v^*(s') \right) \\
&= \max_a \left(R(s, a) + \gamma \sum_{s'} T_a(s, s') v^*(s') \right)
\end{aligned}
$$

or in matrix notation:

$$
\mathbf{v}^* = \max_a \left(R_a + \gamma T_a \mathbf{v}^* \right)
$$

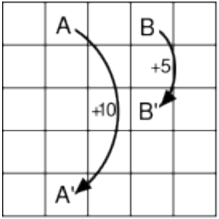
q^* versus v^*

$$V^*(s) = \max_a Q^*(s,a)$$

Why optimal value functions are useful

An optimal policy is **greedy** with respect to v^* or q^*:

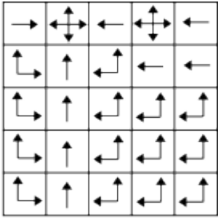
$$\pi^*(s) \in \arg\max_a q^*(s,a) = \arg\max_a \left[\sum_{s'} p(s' \mid a, s)(r(s,a,s') + \gamma v^*(s')) \right]$$



a) gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

b) V^*



c) π^*

Outline

Optimal Policy and Bellman Optimality Equations

Taxonomy of RL problems

Model-based Prediction and Control

Model-based vs model-free

- **Model-based (planning):** the MDP $= (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ is completely specified;
  - Solve the Bellman (optimality) equations
  - Suffices to focus on state value function $v(s)$;

- **Model-free (learning):** only **direct experience**, i.e. sample paths (states, actions and rewards) are given. Put differently, only experience-based information is given!
  - Focus on state-action value function $q(s, a)$
  - Random search but Bellman equations allow to propagate values!

Taxonomy of RL problems

	Prediction *Estimation:* *Given π, what is v?*	**(Optimal) Control** *Optimisation:* *What is optimal π^*?*
model-based (MDP given)	Policy evaluation using Dyn. Programming (DP)	Policy improvement (+ Policy evaluation) = Policy iteration
model-free (MDP unknown)	Monte Carlo (MC) Temporal Diff^{ing} (TD) = "impatient MC" *bootstrapping!*	Q-learning Generalized Policy Iteration *"simultaneous"*

Outline

Optimal Policy and Bellman Optimality Equations

Taxonomy of RL problems

Model-based Prediction and Control

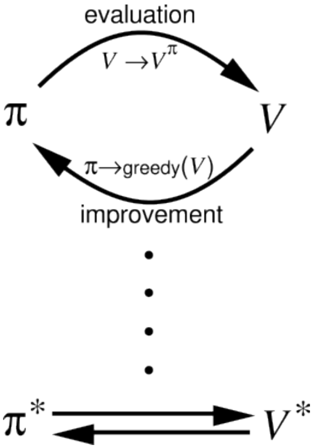
Model-based Prediction and Control

1. **Dynamic Programming (DP):** For **completely specified MDP**, solve

$$\mathbf{v}^* = \max_a (R_a + \gamma T_a \mathbf{v}^*)$$

2. **Policy iteration**
   - **Policy evaluation:** given a policy π compute value functions $v_\pi(s)$ and $q_\pi(s, a)$;
   - **Policy improvement:** given a policy π and corresponding value function v_π, can we find a better policy π' such that $v_{\pi'} \geq v_\pi$? (Spoiler alert: **Greedification!**)
   - **Policy iteration:** iteratively alternate between policy evaluation and improvement to find an optimal policy.

Policy evaluation, improvement and iteration



Policy **evaluation** (1)

Main idea: Iteratively solve the Bellman equation for v_π:

$$\mathbf{v}_\pi = \gamma P_\pi \mathbf{v}_\pi + \mathbf{r}_\pi$$

Policy Evaluation Algo:

- Initial value function v_0 is chosen arbitrarily
- Evaluate value function under the policy update rule:

$$\mathbf{v}_\pi^{k+1} = \gamma P_\pi \mathbf{v}_\pi^k + \mathbf{r}_\pi$$

  or explicitly:

$$v_{k+1}(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s'} p(s' \mid s, a) \Big[r(s, a, s') + \gamma v_k(s') \Big]$$

- Apply to every state in each **sweep** of the state space
- Repeat till convergence to fixed point $\lim_{k\to\infty} v^k = v_\pi$

Policy **evaluation (2)**: Algorithm

Input π, the policy to be evaluated;

Initialize $v(s) = 0$, for all $s \in \mathcal{S}$

Repeat:

$\Delta \leftarrow 0$;

for each $s \in S$:   # single sweep over all states

$v \leftarrow v(s)$

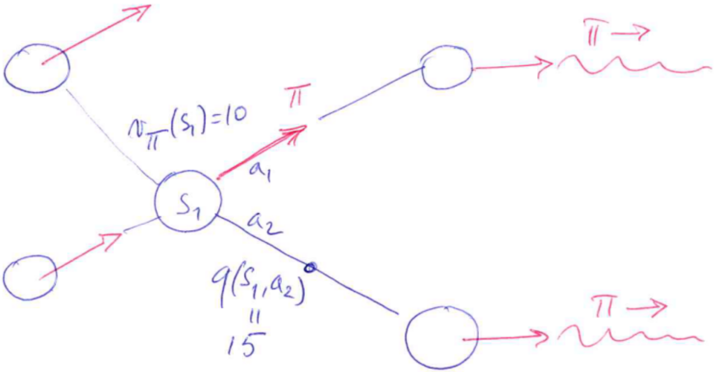
$v(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s'} p(s' \mid s, a)(r(s, a, s') + \gamma v(s'))$

$\Delta \leftarrow \max(\Delta, |v - v(s)|)$

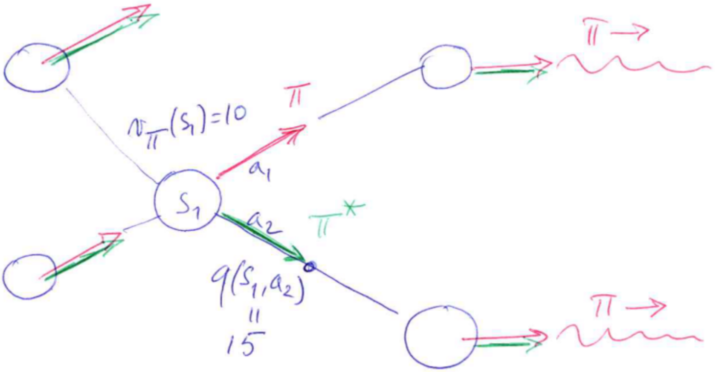
until $\Delta \leq$ small positive number;

Output: $v \approx v_\pi$

Policy **improvement**



Policy **improvement**



Policy **improvement** (1)

- Policy evaluation yields $v_\pi(s)$, and use one-step look-ahead to compute $q_\pi(s, a)$;
- Use this to incrementally improve the policy by considering whether for some state s_0 there is a better action $a \neq \pi(s_0)$:

$$q_\pi(s_0, a) > v_\pi(s_0)?$$

- If so, then the **policy improvement theorem** tells us that defining new policy π' by changing π to take a in s_0 will increase its value:

$$\forall s \in S, v_{\pi'}(s) = q_\pi(s, \pi'(s)) \geq v_\pi(s)$$

- In our case, $\pi = \pi'$ except that $\pi'(s_0) = a \neq \pi(s_0)$

Policy **improvement** (2)

- Applying to all states yields the **greedy** policy w.r.t. v_π:

$$\pi'(s) \leftarrow \arg\max_a q_\pi(s, a)$$

- In that case: $v_{\pi'}(s) = \max_a q_\pi(s, a)$ or again:

$$v_{\pi'}(s) = \max_a \sum_{s'} p(s' \mid s, a)\Big[r(s, a, s') + \gamma v_\pi(s')\Big]$$

- If $\pi = \pi'$, then $v_\pi = v_{\pi'}$ and for all $s \in S$:

$$v_{\pi'}(s) = \max_{a \in A} \sum_{s'} p(s' \mid s, a)\Big[r(s, a, s') + \gamma v'_\pi(s')\Big]$$

- This is equivalent to the Bellman optimality equation, implying that $v_\pi = v_{\pi'} = v^*$ and $\pi = \pi' = \pi^*$

Policy **iteration** (1)

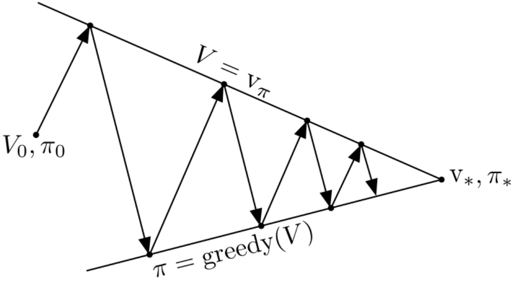
Policy iteration = policy evaluation + policy improvement

$$\pi_0 \xrightarrow{\mathrm{E}} V^{\pi_0} \xrightarrow{\mathrm{I}} \pi_1 \xrightarrow{\mathrm{E}} V^{\pi_1} \xrightarrow{\mathrm{I}} \pi_2 \xrightarrow{\mathrm{E}} \cdots \xrightarrow{\mathrm{I}} \pi^* \xrightarrow{\mathrm{E}} V^*,$$

- Policy improvement makes result of policy evaluation obsolete
- Return to policy evaluation to compute $v_{\pi'}$
- Converges to the fixed point $v_\pi = v^*$

Policy **iteration** (2): geometric analogy

A geometric metaphor for convergence of GPI:



Compare to **EM-algorithm** in ML.

Policy iteration (3): Algorithm (for deterministic policy)

1. Initialization
   $V(s) \in \Re$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
   $\quad \Delta \leftarrow 0$
   $\quad$ For each $s \in \mathcal{S}$:
   $\quad\quad v \leftarrow V(s)$
   $\quad\quad V(s) \leftarrow \sum_{s'} \mathcal{P}_{ss'}^{\pi(s)} \left[\mathcal{R}_{ss'}^{\pi(s)} + \gamma V(s') \right]$
   $\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

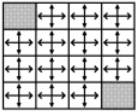
3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $\quad b \leftarrow \pi(s)$
   $\quad \pi(s) \leftarrow \arg\max_a \sum_{s'} \mathcal{P}_{ss'}^{a} \left[\mathcal{R}_{ss'}^{a} + \gamma V(s') \right]$
   $\quad$ If $b \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop; else go to 2

Stopping policy evaluation early

V_k for the Random Policy | Greedy Policy w.r.t. V_k

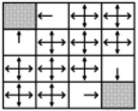
$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



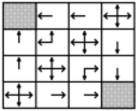
$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0



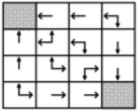
$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0



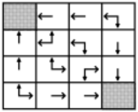
$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0



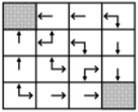
$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0



$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



Value iteration

- Compute optimal v^* first (iteratively), then derive optimal policy π^*:

$$v_1 \longrightarrow v_2 \longrightarrow v_3 \longrightarrow \ldots \longrightarrow v^* \longrightarrow q^*(s,a) \longrightarrow \pi^*$$

- value iteration:

$$\begin{aligned} q_{k+1}(s,a) &\leftarrow \sum_{s'} p(s' \mid s,a)\Big[r(s,a,s') + \gamma v_k(s')\Big] \\ v_{k+1}(s) &\leftarrow \max_a q_{k+1}(s,a), \end{aligned} \tag{1}$$

- Turns Bellman optimality equation into an update rule:

$$v_{k+1}(s) \leftarrow \max_a \sum_{s'} p(s' \mid s,a)\Big[r(s,a,s') + \gamma v_k(s')\Big]$$

Efficiency of dynamic programming

- An MDP has $|A|^{|S|}$ deterministic policies
- But the worst-case computational complexity of dynamic programming is polynomial in $|S|$ and $|A|$
- MDP planning can also be done with **linear programming**, which has better worst-case guarantees, but is impractical for large MDPs
- In very large MDPs, where even doing one sweep is infeasible, **asynchronous dynamic programming** must be used
- Convergence in the limit is guaranteed as long as every state is backed up infinitely often

Summary of terminology

- **Value iteration** algorithms search for optimal value function v^* from which policy is deduced:

$$v_1 \longrightarrow v_2 \longrightarrow \ldots \longrightarrow v^* \longrightarrow \pi^*$$

- **Policy iteration** algorithms evaluate the policy π by computing the (corresponding) value function v_π and uses v_π to improve the policy: va

$$\pi_1 \longrightarrow v_1 \longrightarrow \pi_2 \longrightarrow v_2 \longrightarrow \ldots \longrightarrow \pi^*$$

- **Policy search** algorithms use optimisation techniques to directly search for an optimal policy:

$$\pi_1 \longrightarrow \pi_2 \longrightarrow \ldots \longrightarrow \pi^*$$