CS 486/686
Reinforcement Learning

Yuntian Deng

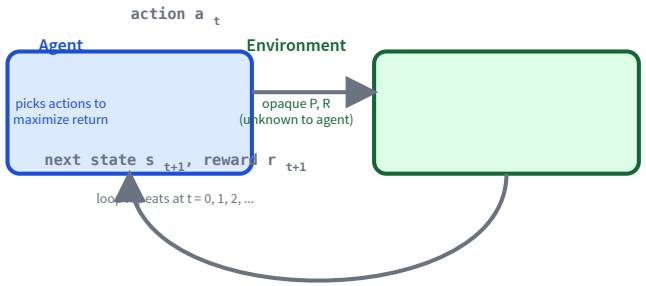
Lecture 15

RN 22.1–22.3 · PM 12.1, 12.5, 12.8

Learning goals

- Trace + implement **passive adaptive dynamic programming** (passive ADP).
- Explain the **exploration vs exploitation** trade-off.
- Trace + implement **active ADP**.
- Trace + implement **Q-learning**.
- Trace + implement **SARSA**.
- Tell **on-policy** from **off-policy** learning.

The RL setting: no P, no R up front



An MDP whose **transition** P and **reward** R are unknown. The agent sees state s and reward r, picks an action a, and observes s'. Goal: maximize discounted return.

Three new challenges:

- **Credit assignment:** which past action caused this reward?
- **Long-term impact:** how will this action affect future returns?
- **Explore vs exploit:** try new actions or stick with the best known?

Passive learning: just evaluate a fixed policy

Fix a policy π. Goal: learn $V^{\pi}(s)$ for every state s.

Adaptive Dynamic Programming (ADP)

1. **Learn** the transition $P(s' \mid s, a)$ and reward $R(s)$ from observed experience.
2. **Solve** the Bellman policy-evaluation equations:

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

A **model-based** approach — it builds an explicit model of the world.

The passive ADP algorithm

passive-ADP(π)

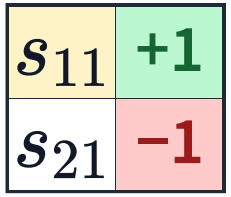
1. Initialize $R(s),\ N(s,a),\ N(s,a,s'),\ V^{\pi}(s)$.
2. Follow π and observe an experience $\langle s, a, s', r \rangle$.
3. Update reward: $R(s) \leftarrow r$.
4. Update counts and transition probability:

$$N(s,a) \mathrel{+}= 1, \quad N(s,a,s') \mathrel{+}= 1, \quad P(s' \mid s,a) = \frac{N(s,a,s')}{N(s,a)}.$$

5. Re-solve Bellman to get $V^{\pi}(s)$ (exactly or iteratively).

Each new experience refines the model; each Bellman solve costs $O(n^3)$ or a few iterations.

Passive ADP: a worked example



$\pi(s_{11}) = \text{down}, \pi(s_{21}) = \text{right}, \gamma = 0.9.$

Before: $N(s, a) = 5$ for all s, a; $N(s, a, s') = 3$ for intended direction, 1 for each side.

New experience: $\langle s_{11}, \text{down}, s_{21}, -0.04 \rangle$.

Update counts: $N(s_{11}, \text{down}) = 6, N(s_{11}, \text{down}, s_{21}) = 4$.

Solve Bellman with the refreshed model:

$$V(s_{11}) = -0.4378, \quad V(s_{21}) = -0.8034.$$

Quick check: probability update

Q1. In a 3x4 grid, suppose $N(s_{23}, \text{down}) = 23$ and $N(s_{23}, \text{down}, s_{24}) = 2$. The agent tries to move *down* but slips and lands in s_{24}. What is the updated $P(s_{24} \mid s_{23}, \text{down})$?

A. $2/23 \approx 0.087$

B. $3/23 \approx 0.130$

C. $3/24 = 0.125$

D. $2/24 \approx 0.083$

> **C — 0.125.** First increment the counts: $N(s_{23}, \text{down}) \to 24$ and $N(s_{23}, \text{down}, s_{24}) \to 3$. Then $P = 3/24 = 0.125$.

Active learning: explore vs exploit

Now the agent *chooses* its actions. Two competing pressures:

Exploit

Take the action that maximizes the current $V(s)$ (or $Q(s, a)$). Reap known rewards.

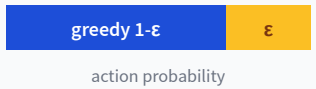
Explore

Try a different action. Gather data to learn a better model / better Q-values.

A purely greedy agent *seldom* converges to the optimal policy — the learned model is biased toward states the agent has already preferred.

Three exploration strategies

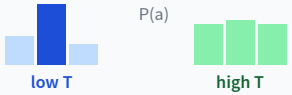
ε-greedy



Random with prob ε;
greedy with prob $1 - \varepsilon$.
Decay ε over time.

Simple, but treats all sub-optimal actions equally.

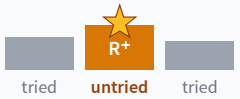
Softmax / Boltzmann



$$P(a) = \frac{e^{Q(s,a)/T}}{\sum_{a'} e^{Q(s,a')/T}}$$

Low T: peaky / greedy. High T: uniform / exploratory.

Optimistic init



$$f(u, n) = \begin{cases} R^+ & n < N_e \\ u & \text{else} \end{cases}$$

"Shiny new toy" bonus until each action has been tried N_e times.

The active ADP algorithm

active-ADP(N_e, R^+)

1. Initialize $R(s),\ V^+(s),\ N(s,a),\ N(s,a,s')$.
2. Repeat until every (s,a) is visited $\geq N_e$ times and V^+ converged:
   - Pick action $a = \arg\max_a f(\sum_{s'} P(s' \mid s,a)\, V^+(s'),\ N(s,a))$.
   - Execute a; observe $\langle s, a, s', r \rangle$; update $R(s) \leftarrow r$, counts, and $P(s' \mid s,a)$.
   - Update $V^+(s) \leftarrow R(s) + \gamma\ \max_a f(\sum_{s'} P(s' \mid s,a)\, V^+(s'),\ N(s,a))$.

Same model-based loop as passive ADP, plus the exploration bonus f baked into action selection.

From V to Q: learn without a model

Bellman for V and Q are both fixed-point equations:

Bellman for $V^\star$

$$V^\star(s) = R(s) + \gamma \max_a \sum_{s'} P(s' \mid s, a)\, V^\star(s')$$

To pick $\pi^\star$ you still need to know P and combine it with $V^\star$.

Bellman for $Q^\star$

$$Q^\star(s, a) = R(s) + \gamma \sum_{s'} P(s' \mid s, a) \max_{a'} Q^\star(s', a')$$

If we know $Q^\star$, then $\pi^\star(s) = \arg\max_a Q^\star(s, a)$ — **no P needed.**

model-free!

Learning Q directly is the trick that makes RL practical at scale.

Temporal difference (TD) error

After one transition $\langle s, a, r, s' \rangle$, pretend that transition was certain ($P(s' \mid s, a) = 1$). Then by Bellman:

$$Q(s,a) \overset{?}{=} \underbrace{R(s) + \gamma \max_{a'} Q(s', a')}_{\text{target}}$$

TD error

$$\delta = \left(R(s) + \gamma \max_{a'} Q(s', a')\right) - Q(s,a)$$

Idea: nudge $Q(s,a)$ toward the target by an amount proportional to δ.

The Q-learning algorithm

Q-learning(α, γ)

1. Initialize $Q(s, a)$ arbitrarily, $R(s), N(s, a)$.
2. Repeat until Q converges:
   - Pick a from s using *any* exploration strategy (ε-greedy, softmax, optimistic).
   - Execute a; observe $\langle s, a, s', r \rangle$; update $R(s) \leftarrow r$.
   - $Q(s, a) \leftarrow Q(s, a) + \alpha\big(r + \gamma \max_{a'} Q(s', a') - Q(s, a)\big)$.
   - $s \leftarrow s'$.

- **Model-free**: never estimates P explicitly.
- **Learning rate**: smaller α = slower but more accurate; e.g. $\alpha(N) = 10/(9 + N(s, a))$.
- Converges to $Q^\star$ if every (s, a) is visited infinitely often.

Q-learning: a single update

3x4 grid. Experience: $\langle s_{32}, \text{right}, s_{33}, -0.04 \rangle$. Take $\gamma = 1, \alpha = 0.1$.

	s_{32}	s_{33}
up	0.55	0.4
right	0.7	0.9
down	0.5	0.8
left	0.4	0.5

Yellow cell = the one we update; green = the max-value action in s_{33}.

target = r + γ · $\max_{a'}$ Q(s_{33}, a') = −0.04 + 1.0 · 0.9 = 0.86

δ = target − Q(s_{32}, right) = 0.86 − 0.7 = 0.16

Q(s_{32}, right) ← 0.7 + 0.1 · 0.16 = 0.716

SARSA: on-policy temporal difference

Observe a 5-tuple $\langle s, a, r, s', a' \rangle$ where a' is the action **actually taken** in s' (per the current policy):

Q-learning (off-policy)

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big)$$

Target uses the *greedy* next action. Learns about the optimal policy regardless of what's actually played.

SARSA (on-policy)

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big(r + \gamma\, Q(s',a') - Q(s,a)\big)$$

Target uses the *actually-played* next action a'. Learns about the policy the agent is following (exploration and all).

SARSA = State, Action, Reward, State, Action.

SARSA vs Q-learning: a side-by-side update

At s_{13}, update $Q(s_{13}, \text{down}) = 0.4$. Next state s_{23}: $Q = (\text{L}{:}0.3,\ \text{R}{:}-0.4,\ \text{D}{:}0.7,\ \text{U}{:}0.3)$, $r = -0.04$, $\gamma = 0.9$, $\alpha = 0.4$. The agent's ε-greedy step explores $a' = \text{right}$ (into the bad cell).

SARSA: uses $a' = \text{right}$

```
target = −0.04 + 0.9 · (−0.4) =
−0.4
δ = −0.4 − 0.4 = −0.8
Q(s13, down) ← 0.4 + 0.4 · (−0.8)
```

= 0.08 (punished for the bad step)

Q-learning: uses $\max_{a'} = 0.7$

```
target = −0.04 + 0.9 · 0.7 = 0.59
δ = 0.59 − 0.4 = 0.19
Q(s13, down) ← 0.4 + 0.4 · 0.19
```

= 0.476 (rewarded for the best)

SARSA is **conservative** (learns what really happens); Q-learning is **aggressive** (learns the optimal policy).

ADP vs Q-learning vs SARSA

	ADP	Q-learning	SARSA
Learns transition P?	**yes (model-based)**	**no (model-free)**	**no (model-free)**
On / off policy	*n/a* (eval-only)	off-policy	on-policy
Computation per experience	**heavy (solve Bellman)**	**cheap (one TD update)**	**cheap (one TD update)**
Convergence speed	**fast (few experiences)**	**slower, higher variance**	**slower, higher variance**

ADP is sample-efficient but expensive per step; TD methods (Q, SARSA) flip that trade-off.

When to use what (Q2)

Q2. For a high-stakes *online* learning problem (for example, a self-driving car still in training), which algorithm is safer?

A. Q-learning

B. **SARSA**

C. I don't know

> **B — SARSA.** SARSA's target uses the action that will *actually* be taken while exploring, so it learns to avoid catastrophic outcomes along the exploration path. Q-learning's target uses the greedy action, ignoring the cost of exploration mistakes.

Putting the labels together (Q3)

Q3. Which statement is true?

A. Q-learning is model-free + off-policy; SARSA is model-based + on-policy.

B. Q-learning is model-based + off-policy; SARSA is model-free + on-policy.

C. Q-learning is model-free + on-policy; SARSA is model-based + off-policy.

D. Q-learning is model-based + on-policy; SARSA is model-free + off-policy.

E. **None of the above.**

E — None of the above. Both Q-learning and SARSA are *model-free*. Q-learning is off-policy (target uses $\max_{a'} Q$); SARSA is on-policy (target uses the action actually played).

Learning goals (recap)

✓ Trace + implement **passive ADP**.

✓ Explain the **exploration vs exploitation** trade-off.

✓ Trace + implement **active ADP**.

✓ Trace + implement **Q-learning**.

✓ Trace + implement **SARSA**.

✓ Tell **on-policy** from **off-policy**.

Search › Uncertainty › Decisions › Learning

Next module: Machine Learning and Deep Learning

RL was learning from rewards. The last module turns to *learning from data*.

- **L16** — supervised learning, bias / variance, cross-validation.
- **L17** — unsupervised: k-means, PCA, autoencoders, GANs.
- **L18** — decision trees: entropy, information gain, ID3.
- **L19–L21** — neural networks: forward / backward / optimizers.