上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Lecture 31: Temporal Difference Learning & Q-Learning

Tao Huang

John Hopcroft Center, School of Computer Science, Shanghai Jiao Tong University

https://taohuang.info/cs3317

https://oc.sjtu.edu.cn/courses/89538

AI tools assisted in generating some figures in these slides. All such content has been reviewed, and the instructor is responsible for its accuracy.

Where We Left Off

- **L29** gave us the **MDP** — states, actions, rewards, the Bellman equation.
- **L30** showed how to *solve* it — value iteration and policy iteration — when we know $P(s'|s,a)$ and $R(s,a)$.

But the world doesn't hand us P and R. A robot doesn't know its own dynamics. An Atari agent doesn't get the rules in equation form.

DP world (L30)

$$P(s' \mid s, a)$$

$$R(s, a, s')$$

given as equations

RL world (L31)

Robot Atari Cards

must be discovered through interaction

Big Idea

Learn by trying. Update your estimate using your next estimate.

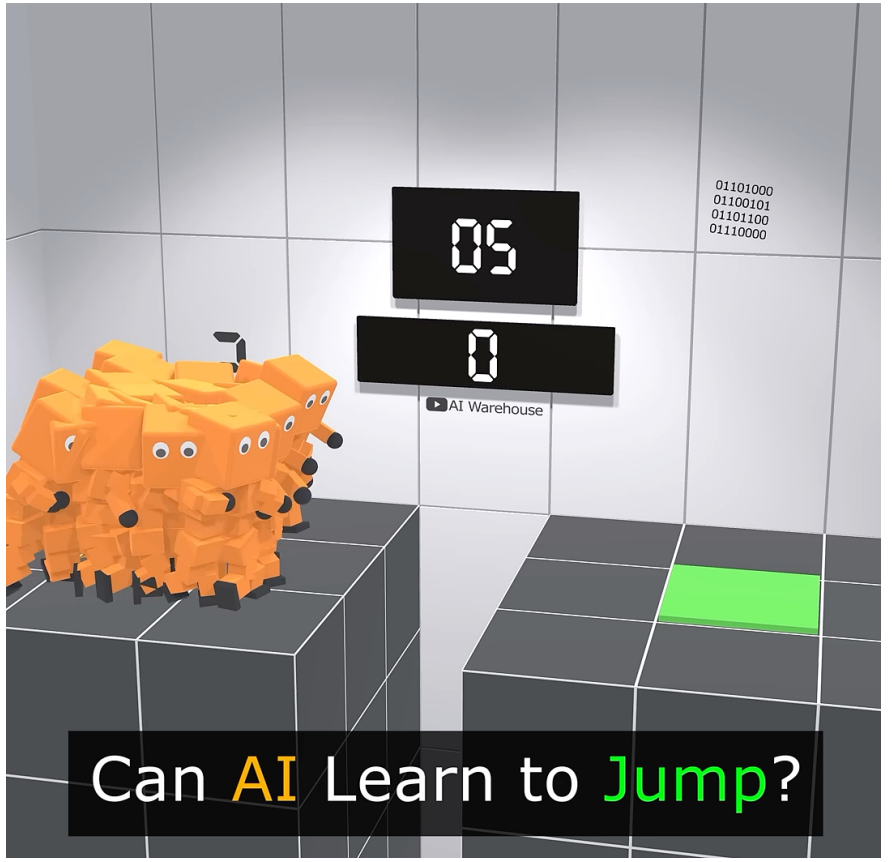
- **Monte Carlo:** play out the whole episode, then average returns.
  Honest but slow.
- **Dynamic Programming:** assume you know the world, lookahead one step.
  Fast but unrealistic.
- **Temporal Difference:** act, observe, bootstrap from your current estimate.
  Best of both.

Question & Objectives

By the end of this lecture, you will be able to:

- Explain what bootstrapping means and why it's not circular.
- Compute a TD(0) update by hand on a gridworld.
- Distinguish SARSA (on-policy) from Q-learning (off-policy) in one sentence each.
- Apply ε-greedy and explain why exploration matters.
- Identify which RL situations Q-learning fits, and which it doesn't.
- Trace the line from Q-learning (1989) to the modern PPO/RLHF stack.

The Hook — DeepMind plays Atari



In 2015, DeepMind's DQN learned to play 49 Atari games from raw pixels using ONE algorithm — a deep version of Q-learning.

https://www.youtube.com/shorts/hgjsLmFSkxo

1. From planning to learning

Situation: DP needs the model

Recall the value iteration update from L30:

$$V_{k+1}(s) \;=\; \max_{a} \sum_{s'} P(s' \mid s, a) \left[R(s, a, s') + \gamma \, V_k(s') \right]$$

Knows the transition probabilities.
Sums over all s'.

Knows the reward function.
Reads R(s, a, s') directly.

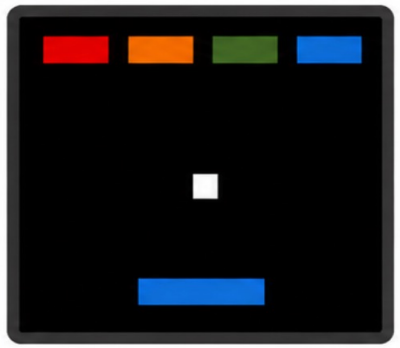
Both assumptions break the moment we leave the textbook.

Complication: The Real-World Setting

Robot

I moved my motors — where did I end up? I don't know P.

Atari Agent



I see pixels. Each frame is the state. Nobody gave me P or R in equations.

Poker

I know the rules. But my opponent's policy is part of P — and it's hidden.

In all three: we can act and observe, but never enumerate P, R.

Question: What Can We Still Do?

Can we estimate $V^{\pi}(s)$ using only observed trajectories?

$$(s_0,\ a_0,\ r_1,\ s_1,\ a_1,\ r_2,\ s_2,\ \ldots)$$

We have no equation for P. We have something better — experience.

Monte Carlo: The Obvious First Idea

Update V toward the observed return G_t :

$$V(s) \leftarrow V(s) + \alpha \left[G_t - V(s) \right]$$

where the return is

$$G_t = r_{t+1} + \gamma\, r_{t+2} + \gamma^2\, r_{t+3} + \dots$$

✓ **Pro:** unbiased — uses real returns, no assumptions about P.

✗ **Con:** must wait until the episode ends. What about Go? Continuing tasks?

Monte Carlo is honest but slow. Can we update BEFORE the episode ends?

The TD Insight: Bootstrap From Your Own Estimate

Replace the unknown future return G_t with a one-step lookahead:

$$V(s_t) \leftarrow V(s_t) + \alpha \left[\underbrace{r_{t+1} + \gamma V(s_{t+1})}_{\text{TD target}} - V(s_t) \right]$$

TD target = use your CURRENT estimate of the next state's value.

TD error δ_t : *how surprised were we?*

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

This looks circular — updating an estimate using itself. Spoiler: it works. (See Sutton & Barto Ch. 6.)

🧠 Think — Why Isn't This Circular?

We're using $V(s_{t+1})$ — an estimate we haven't finished learning — to update V(s_t). Isn't that a bug?

Discuss with your neighbor:

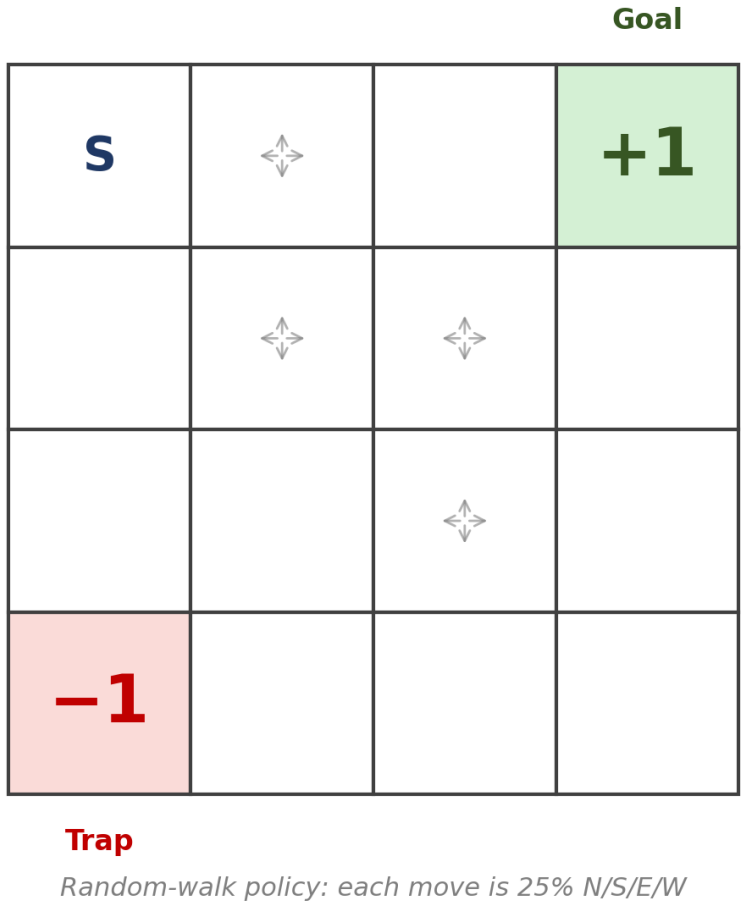
(a) Where does the **non-circular signal** come from in the update?

(b) What happens at **terminal states** (goal reached, game ends)?

(c) If all estimates start at zero, what gets the **first non-zero update?**

2. Worked example — a tiny gridworld

Setup: A 4×4 Gridworld



Random-walk policy: each move is 25% N/S/E/W

Parameters

γ = 0.9 *(discount)*

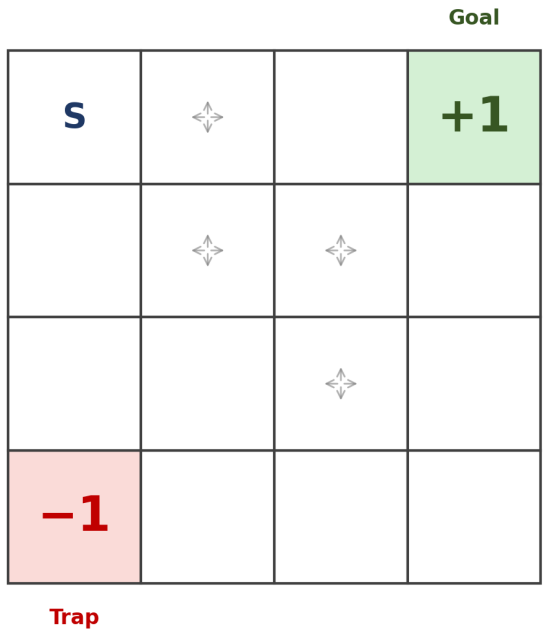
α = 0.1 *(learning rate)*

V(s) = 0 *for all s*

We'll watch V update as the agent walks. By hand. Eight steps.

Trajectory: One Episode, 8 Steps

Step	State (x, y)	Action a_t	Reward r_{t+1}	Next state (x, y)	Notes
1	(0, 0) [S]	right (→)	0	(1, 0)	move right
2	(1, 0)	down (↓)	0	(1, 1)	move down
3	(1, 1)	right (→)	0	(2, 1)	move right
4	(2, 1)	right (→)	0	(3, 1)	move right
5	(3, 1)	down (↓)	0	(3, 2)	move down
6	(3, 2)	down (↓)	0	(3, 3)	move down
7	(3, 3)	left (←)	0	(2, 3)	move left
8	(2, 3)	right (→) → goal	+1	terminal	reached goal (+1)



Reward column: mostly zeros, then +1 at the very end.

TD(0) Updates, Step by Step

Apply $V(s_t) \leftarrow V(s_t) + \alpha \cdot [r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$ at each step:

- **Step 1:** $V(s_0) \leftarrow 0 + 0.1 \cdot (0 + 0.9 \cdot 0 - 0) = \mathbf{0}$ *no update — no information yet.*
- **Steps 2–6:** same — each TD target is (0 + 0.9·0) = 0, and V doesn't move.
- **Step 7:** $V(s_6) \leftarrow 0 + 0.1 \cdot (0 + 0.9 \cdot 0 - 0) = \mathbf{0}$ *still nothing.*
- **Step 8 (terminal):** $V(s_7) \leftarrow 0 + 0.1 \cdot (1 + 0 - 0) = \mathbf{0.10}$ ***first non-zero update — the reward!***

After ONE episode, only s_7 learned anything. The signal hasn't propagated yet.

Value Function Propagate

Value information flows backward through state-space, one step at a time. That's TD.

3. From predicting to controlling — Q-learning

Prediction vs. Control

- **What we just did:**

Estimated $V^{\pi}(s)$ for a FIXED policy π. That's **prediction**.

- **What we actually want:**

Find the OPTIMAL policy π*. That's **control**.

V(s) doesn't tell us WHAT ACTION TO TAKE. We need values for state–action pairs.

The Action-Value Function Q

$\boldsymbol{Q^{\pi}(s,a)}$ **=** the value of taking action a in state s, then following π thereafter.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right]$$

max_a' — always look at the BEST next action, even if our current policy wouldn't take it.

This is **Q-learning** (Watkins, 1989).

One line of pseudocode. One of the most important algorithms in AI.

Off-Policy Magic: SARSA vs. Q-Learning

SARSA (on-policy):

$$r_{t+1} + \gamma\, Q(s_{t+1},\ a_{t+1})$$

uses the action we ACTUALLY took

Q-learning (off-policy):

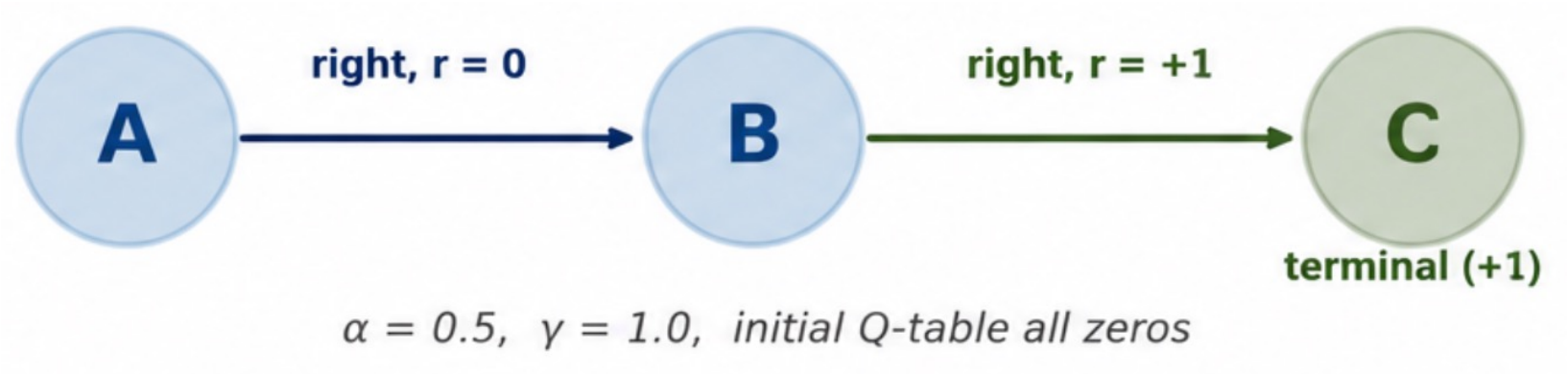
$$r_{t+1} + \gamma\ \max_{a'} Q(s_{t+1},\ a')$$

uses the action we WOULD optimally take

Q-learning learns about the OPTIMAL policy while behaving with a DIFFERENT policy (e.g., random exploration). We can explore freely and still converge to optimal.

On-policy: learn what you do. *Off-policy: learn what you'd ideally do.*

🧠 Think — Q-learning by Hand

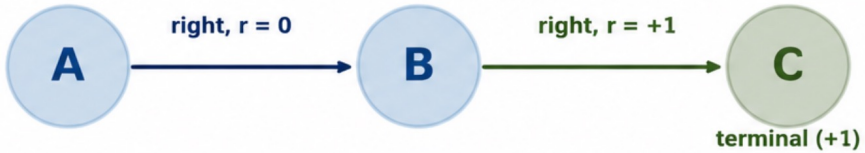


Observed episode: A →(right, r=0)→ B →(right, r=+1)→ C.
What is Q(B, right) after this episode? What is Q(A, right)?

Think — Q-learning by Hand

Observed episode: A →(right, r=0)→ B →(right, r=+1)→ C.
What is Q(B, right) after this episode? What is Q(A, right)?

MDP



$\alpha = 0.5,\ \gamma = 1.0,$ *initial Q-table all zeros*

Episode 1 (start at A)

Step	Transition	Target	Q update	Q-table after step		
				A (right)	B (right)	C (terminal)
1	A → B, $r = 0$	$0 + 1 \cdot \max_a Q(B, a) = 0$	Q(A,right): 0 + 0.5 (0 − 0) = 0	0	0	0
2	B → C, $r = +1$	$1 + 1 \cdot \max_a Q(C, a) = 1$	Q(B,right): 0 + 0.5 (1 − 0) = 0.5	0	0.5	0
3	at C (terminal)	—	—	0	0.5	0

Episode 2 (start at A)

Step	Transition	Target	Q update	Q-table after step		
				A (right)	B (right)	C (terminal)
1	A → B, $r = 0$	$0 + 1 \cdot \max_a Q(B, a) = 0.5$	Q(A,right): 0 + 0.5 (0.5 − 0) = 0.25	0.25	0.5	0
2	B → C, $r = +1$	$1 + 1 \cdot \max_a Q(C, a) = 1$	Q(B,right): 0.5 + 0.5 (1 − 0.5) = 0.75	0.25	0.75	0
3	at C (terminal)	—	—	0.25	0.75	0

Episode 3 (start at A)

Step	Transition	Target	Q update	Q-table after step		
				A (right)	B (right)	C (terminal)
1	A → B, $r = 0$	$0 + 1 \cdot \max_a Q(B, a) = 0.75$	Q(A,right): 0.25 + 0.5 (0.75 − 0.25) = 0.50	0.50	0.75	0
2	B → C, $r = +1$	$1 + 1 \cdot \max_a Q(C, a) = 1$	Q(B,right): 0.75 + 0.5 (1 − 0.75) = 0.875	0.50	0.875	0
3	at C (terminal)	—	—	0.50	0.875	0

4. The explorer's dilemma

The Dilemma

You can't learn the value of an action you never try.

Restaurant: favorite (exploit) or new place (explore)?

Doctor: standard treatment (exploit) or new clinical trial (explore)?

Atari agent: known reward direction (exploit) or untried actions (explore)?

Too much exploration → never converges. *Too little → stuck in a local optimum.*

ε-greedy: The Simplest Solution

At each step, with probability:

ε → pick a **random action** *(explore)*

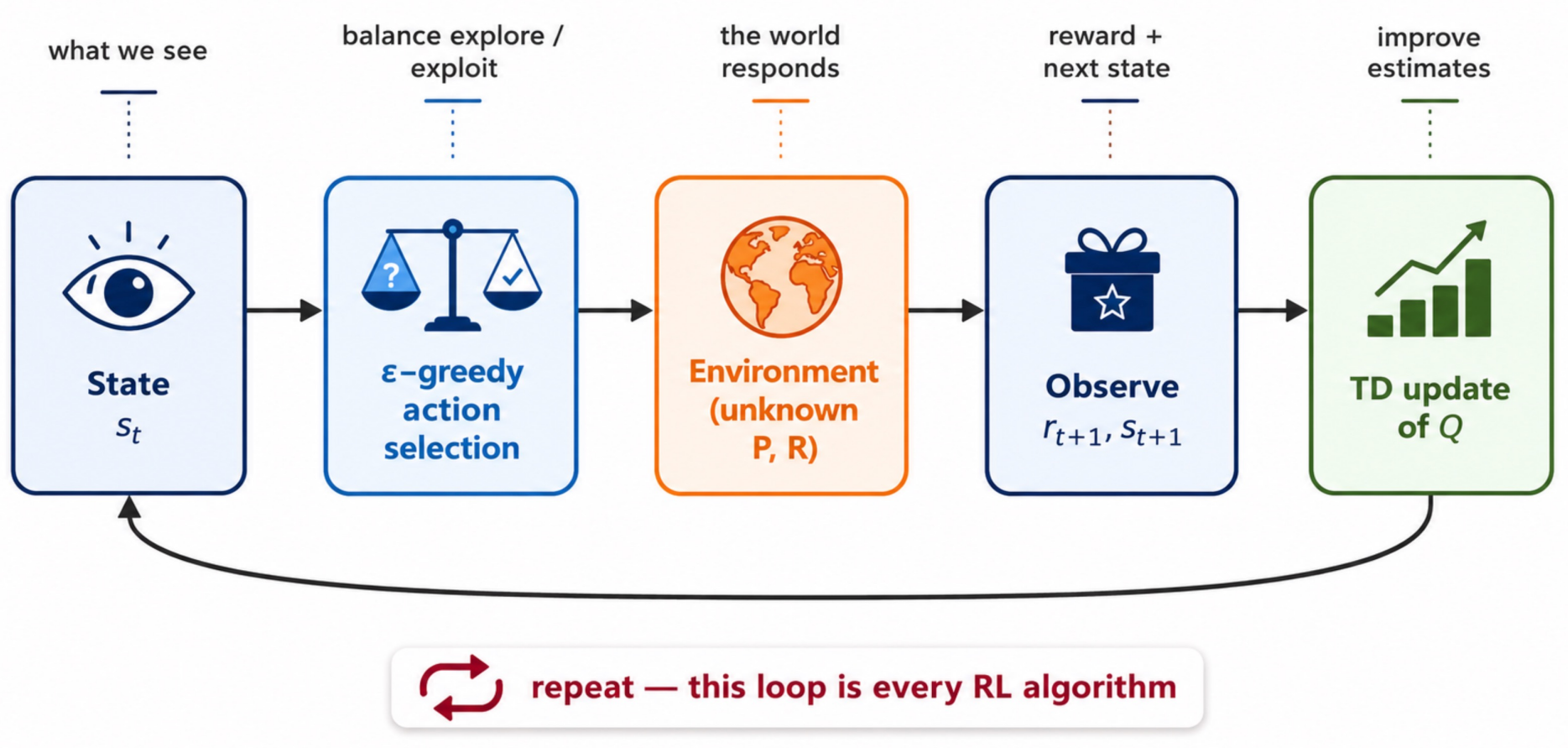
1 – ε → pick argmax_a Q(s, a) *(exploit)*

Schedule: start ε = 1.0, decay toward ε = 0.05 over training.

Exploration Emerging — OpenAI Hide-and-Seek

OpenAI Hide-and-Seek (2019). Agents learned six distinct strategies in sequence — none designed by the engineers.

Connecting the Threads — The Universal RL Loop



This loop is EVERY RL algorithm — DQN, PPO, AlphaGo, RLHF. Only the function approximator changes.

Summary

- **Monte Carlo** waits till episode end. **TD** bootstraps after one step.
- **TD target** = real reward + discounted estimate of next state.
- **Q-learning** is TD on action-values, ***off-policy*** (learn optimal while behaving non-optimally).
- **ε-greedy** is the simplest way to keep exploring while improving.
- **observe → act → update** is the universal RL skeleton.