Applied Computational Mathematics Lab

TEDDY LAZEBNIK'S COURSE · LECTURE 6

Reinforcement Learning

Lecture 6 · Model-free control: Monte Carlo control and GLIE, SARSA, Q-learning and off-policy learning, double Q-learning, importance sampling and expected SARSA



Teddy Lazebnik · Applied Computational Mathematics Laboratory

BEFORE WE START

Recap, and what this lecture does

- RL is the science of learning to make decisions; agents learn a policy, a value function and/or a model.
- Lecture 5, model-free **prediction**: estimate the value function of an unknown MDP.
- This lecture, model-free **control**: **optimise** the value function of an unknown MDP.



IN PLAIN WORDS

Nobody wants only a critic.
Today the agent learns to act.

EXAMPLE

A robot vacuum in a new flat: no floor plan, no model of where each move leads. It must still learn a good cleaning route.

LECTURE 5

A critic

Give it a policy, and it tells you how good it is.

LECTURE 6

An actor

Learn to act, and get better at it. No model, no map: just experience.

BEFORE WE START

Today's map

PART 1

Monte Carlo control

Policy iteration with q, ε-greedy exploration, and GLIE.

PART 2

TD control: SARSA

Update after every step; a windy gridworld.

PART 3

Off-policy learning and Q-learning

Learn about one policy while following another; a walk along a cliff.

PART 4

Overestimation and double learning

Why the max is too optimistic; a trip to the casino.

PART 5

Importance sampling and expected SARSA

Correcting for the wrong policy; one family of updates.

READING

Sutton & Barto (2018)

Chapter 6 (and 5.3–5.7 for Monte Carlo control and importance sampling).



IN PLAIN WORDS

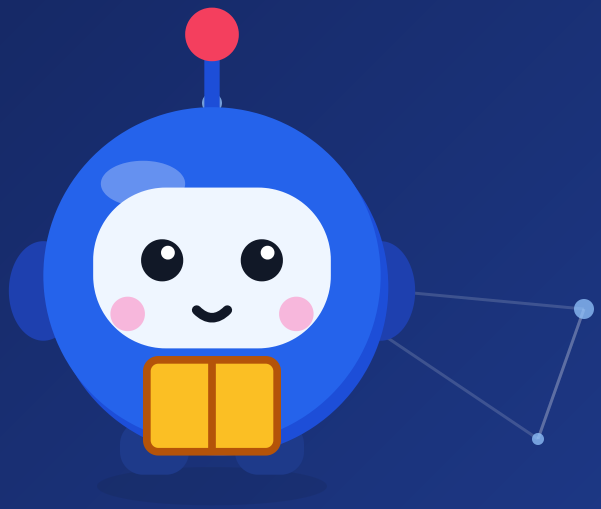
By the end you should be able to run SARSA and Q-learning by hand, explain why they find different paths, and say what double learning and importance sampling fix.

Applied Computational Mathematics Lab

PART 1

Monte Carlo control

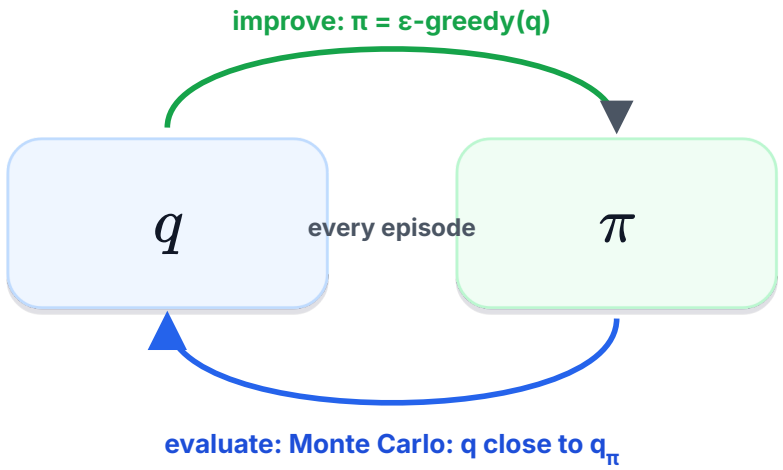
Policy iteration with action values and exploration.



Generalised policy iteration

- Recipe from dynamic programming: **evaluate** the policy, **improve** it, repeat.
- **Policy evaluation**: estimate v_π (or q_π) for all states.
- **Policy improvement**: generate π' with $v_{\pi'}(s) \geq v_\pi(s)$ for all s.
- Today we run this loop **without a model**, using the evaluation methods of Lecture 5:

METHOD	TARGET FOR $v(S_t)$
Monte Carlo	$G_t = R_{t+1} + \gamma G_{t+1}$
TD(0)	$R_{t+1} + \gamma v(S_{t+1})$
n-step TD	$R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^n v(S_{t+n})$
TD(λ)	$R_{t+1} + \gamma[(1-\lambda)v(S_{t+1}) + \lambda G^\lambda_{t+1}]$



Generalised policy iteration: the two steps pull toward q_* and π_*.



IN PLAIN WORDS

In all evaluation methods the goal is v_π for a given π, and the data is generated by π.

Why learn q, not v

- Greedy improvement over $v(s)$ needs a model of the MDP: where each action leads.

$$\pi'(s) = \operatorname*{argmax}_a \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s, A_t = a\right]$$

- Greedy improvement over $q(s, a)$ is model-free:

$$\pi'(s) = \operatorname*{argmax}_a q(s, a)$$

- That makes action values convenient: for control, we learn q.

STATE	UP	DOWN	LEFT	RIGHT
kitchen	2.1	0.4	**3.0**	1.7
hallway	**4.2**	3.9	−1.0	0.5

An illustrative q-table. Improving needs no model: in each row, take the biggest number.



IN PLAIN WORDS

Like choosing a dish from reviews: you don't need to know what happens in the kitchen.

WITH V INSTEAD

To act greedily on v in the kitchen, you'd need to know that "left" leads to the hallway (and with what probability) and what reward it pays: that is p.

Greedy is a trap: explore

- Greedy improvement plus Monte Carlo evaluation? **No exploration**: we can't sample all s, a pairs when we learn by interacting.
- If the first action you tried paid a little, greedy picks it forever. The others never get the chance to show their worth.
- Fix: **ε-greedy**. With probability $1 - \varepsilon$ take the greedy action; with probability ε a uniformly random one:

$$\pi(a \mid s) = \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}| & \text{if } a = \operatorname{argmax}_{a'} q(s, a') \\ \varepsilon/|\mathcal{A}| & \text{otherwise} \end{cases}$$

TRY IT

4 actions, $\varepsilon = 0.1$.
The greedy action: $0.9 + 0.1/4 = 0.925$.
Each other action: $0.1/4 = 0.025$.
The random move can land on the greedy action too: exploring includes the obvious.

LINK

This generalises the ε-greedy bandit algorithm of Lecture 2 to many states.

ε-greedy improvement works

THEOREM

For any ε-greedy (more generally ε-soft) policy π, the ε-greedy policy π' with respect to q_π is an improvement: $v_{\pi'}(s) \geq v_\pi(s)$ for all s.

$$\begin{aligned}\sum_a \pi'(a \mid s)\, q_\pi(s,a) &= \frac{\varepsilon}{|\mathcal{A}|} \sum_a q_\pi(s,a) + (1-\varepsilon) \max_a q_\pi(s,a) \\ &\geq \frac{\varepsilon}{|\mathcal{A}|} \sum_a q_\pi(s,a) + (1-\varepsilon) \sum_a \frac{\pi(a \mid s) - \varepsilon/|\mathcal{A}|}{1-\varepsilon}\, q_\pi(s,a) = v_\pi(s)\end{aligned}$$

- The weights $(\pi(a \mid s) - \varepsilon/|\mathcal{A}|)/(1-\varepsilon)$ are non-negative and add up to 1, and a max beats any weighted average. Then the policy improvement theorem (Lecture 3) finishes the job.



IN PLAIN WORDS

Acting ε-greedily on your own q is never worse than the ε-greedy policy you had.

SOURCE

The proof follows Sutton & Barto (2018), section 5.4.

The Monte Carlo control algorithm

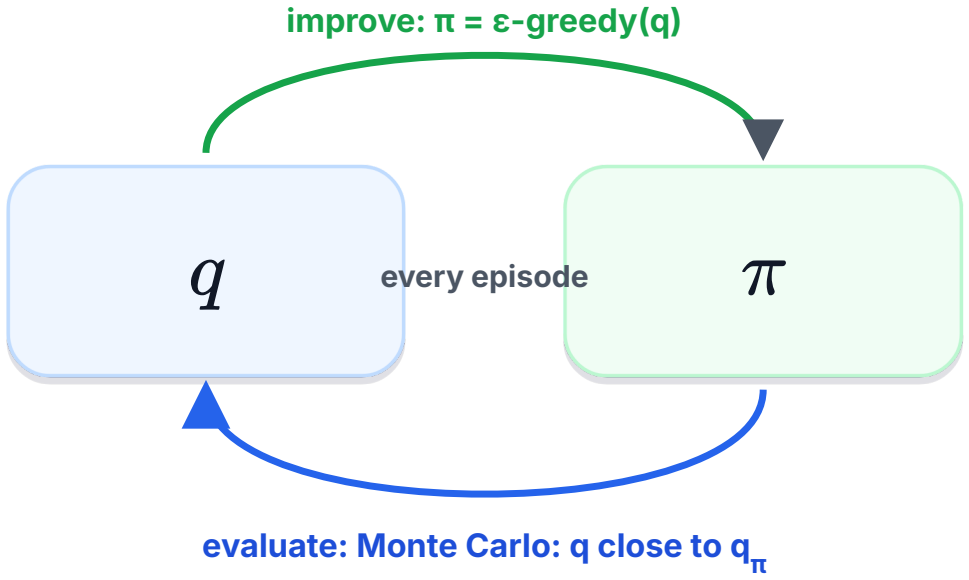
MONTE CARLO CONTROL

1. Sample episode k with π: $S_1, A_1, R_2, \ldots, S_T$.
2. For each S_t, A_t in it: $q(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha_t\big(G_t - q(S_t, A_t)\big)$, for example with $\alpha_t = 1/N(S_t, A_t)$ or $1/k$.
3. Improve: $\varepsilon \leftarrow 1/k$, $\pi \leftarrow \varepsilon\text{-greedy}(q)$.
4. Repeat.



IN PLAIN WORDS

Evaluate a little, improve a little, every episode. We don't wait for the evaluation to finish.



Every episode: Monte Carlo evaluation, then ε-greedy improvement.

GLIE: greedy in the limit

DEFINITION (GLIE)

Greedy in the Limit with Infinite Exploration:

- all state-action pairs are explored infinitely many times: $\lim_{t\to\infty} N_t(s,a) = \infty$ for all s, a;
- the policy converges to a greedy policy: $\lim_{t\to\infty} \pi_t(a \mid s) = \mathbb{I}\big(a = \operatorname{argmax}_{a'} q_t(s,a')\big)$.

For example, ε-greedy with $\varepsilon_k = 1/k$.

THEOREM

GLIE model-free (Monte Carlo) control converges to the optimal action-value function: $q_t \to q_*$.

EPISODE k	$\varepsilon_k = 1/k$
1	1
2	0.5
10	0.1
100	0.01
200	0.005



IN PLAIN WORDS

A cleaning robot with a fixed ε of 0.1 does random things in your living room forever. Let ε shrink: day one it explores a lot; by run 200 it should know the house.

Check yourself: Part 1

CHECK YOURSELF

1. Why can't we improve greedily on v without a model, but can on q?

CHECK YOURSELF

2. With 5 actions and $\varepsilon = 0.2$, what is the probability of the greedy action? Of each other action?

CHECK YOURSELF

3. Is a fixed $\varepsilon = 0.1$ GLIE? Is $\varepsilon_k = 1/k$?

CHECK YOURSELF

4. In the illustrative q-table, which action does the greedy policy take in the hallway?





HINT

Answers are at the end of the deck.

Applied Computational Mathematics Lab

PART 2

TD control: SARSA

Evaluate and improve after every single step.



From Monte Carlo to TD control

- TD has several advantages over Monte Carlo: lower variance, online, learns from incomplete sequences.
- Natural idea: use TD instead of MC for control:
  - apply TD to $q(s, a)$;
  - use, for example, ε-greedy policy improvement;
  - update every time step.

WHY IT MATTERS

Monte Carlo control waits until the end of an episode before improving. In a long task, or one that never ends, TD control improves after every step.



the five things one SARSA update needs

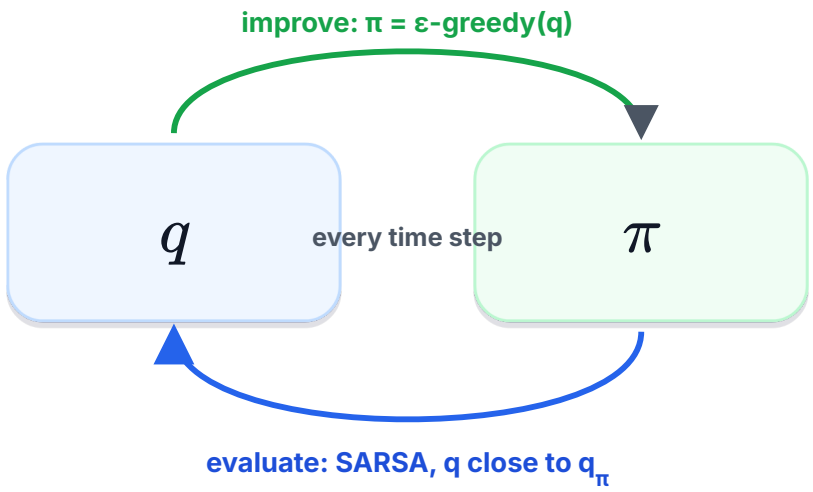
$$q_{t+1}(S_t, A_t) = q_t(S_t, A_t) + \alpha_t\big(R_{t+1} + \gamma q_t(S_{t+1}, A_{t+1}) - q_t(S_t, A_t)\big)$$



IN PLAIN WORDS

TD on q has a name you can pronounce: SARSA, after the data it needs.

SARSA, step by step



Every time step: SARSA evaluation, then ε-greedy improvement.

- Nudge $q(S_t, A_t)$ toward the reward plus γ times q of the **next pair**, where A_{t+1} is the action actually chosen.
- Then act ε-greedily on the new q, and update again.

TRY IT

$q(s, a) = 2$, reward -1, next state has $q(s', \cdot) = (0, 4, 3)$, and the ε-greedy policy happened to pick the first action. $\gamma = 0.9$, $\alpha = 0.5$.

Target $-1 + 0.9 \times 0 = -1$; new $q(s, a) = 2 + 0.5(-1 - 2) = 0.5$.



IN PLAIN WORDS

Update the policy after every step: smoother progress, noisier steps. Which is better for your problem? There's no way to know in advance.

Tabular SARSA converges

TABULAR SARSA

1. Initialise $q(s, a)$, for example to 0.
2. Each episode: take the start state S; pick A ε-greedily from q.
3. Each step: act, observe R, S'; pick A' ε-greedily from q.
4. $q(S, A) \leftarrow q(S, A) + \alpha\big(R + \gamma q(S', A') - q(S, A)\big)$; then $S \leftarrow S'$, $A \leftarrow A'$.

THEOREM

Tabular SARSA converges to the optimal action-value function, $q(s, a) \to q_*(s, a)$, if the policy is GLIE and the step sizes satisfy $\sum_t \alpha_t = \infty$ and $\sum_t \alpha_t^2 < \infty$.

TRY IT

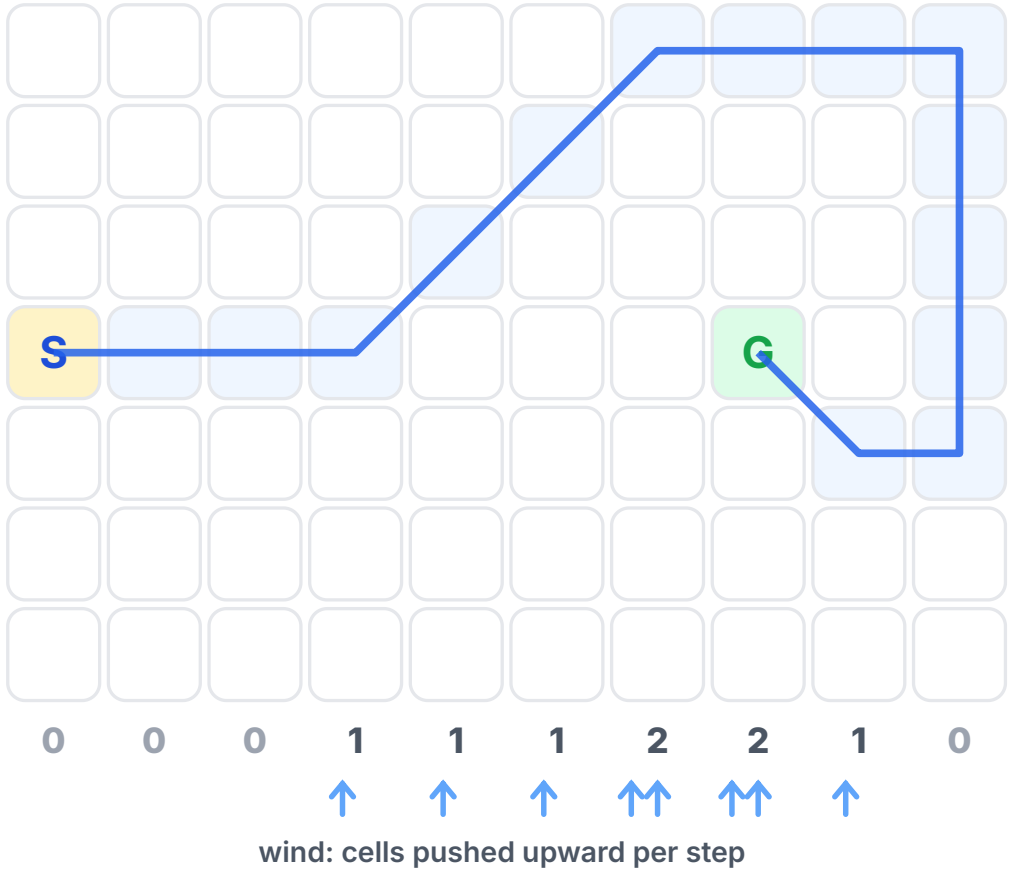
$\alpha_t = 1/t$ satisfies both. After a million steps, $\sum 1/t \approx 14.39$, and it keeps growing forever. The squares add up to $\pi^2/6 \approx 1.645$.



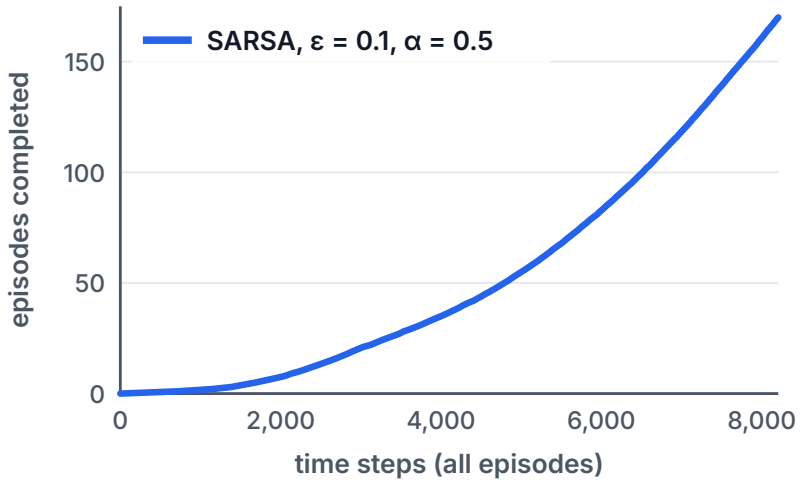
IN PLAIN WORDS

Big enough to get anywhere; small enough to settle down. And in code, SARSA is about ten lines.

Example: windy gridworld



Example 6.5 of Sutton & Barto (2018). The blue line is one final greedy path from our runs (15 moves). Our drawing.



Our runs: 50 of them, 170 episodes each. The first episode takes about 715 steps; the last 20 average 22.7 steps each. The curve steepens as the agent learns.

- Moves: up, down, left, right; reward −1 per step until the goal G.
- Why not 15 steps per episode? ε still adds a random move now and then, and the wind punishes every one.

SARSA is on-policy

- SARSA learns the value of the policy it actually follows, random moves included.
- Its target uses A_{t+1}, drawn from the same ε-greedy policy it is improving.
- For convergence to q_* we need that policy to become greedy (GLIE), for example ε-greedy or softmax with decreasing exploration.



IN PLAIN WORDS

That sounds like a detail. It isn't: it changes which path the agent thinks is best. Part 3 shows how.

INGREDIENT	CHOICE IN OUR WINDY RUN
Exploration	ε-greedy, ε = 0.1 (fixed)
Step size	α = 0.5 (fixed)
Initial values	q = 0 everywhere
Discount	γ = 1, reward −1 per step

With a fixed ε the run is not GLIE, so it learns the best ε-greedy policy, not the greedy optimum.

IN ONE LINE

Learn q, explore, shrink the exploration slowly, and update with SARSA: that is model-free control.

Check yourself: Part 2

CHECK YOURSELF

5. $q(s, a) = 1$, reward 2, next pair $q(s', a') = 4$, $\gamma = 0.5$, $\alpha = 0.25$. What is the SARSA update?

CHECK YOURSELF

6. Does $\alpha_t = 1/\sqrt{t}$ satisfy both step-size conditions? Does $\alpha_t = 1/t^2$?

CHECK YOURSELF

7. In the windy gridworld, why do the episodes stay longer than the shortest path even after learning?



Applied Computational Mathematics Lab

PART 3

Off-policy learning and Q-learning

Learning about one policy while following another.



Dynamic programming and its TD cousins

$$
\begin{aligned}
v_{k+1}(s) &= \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t \sim \pi(S_t)] && \text{(policy evaluation)} \\
v_{k+1}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a] && \text{(value iteration)} \\
q_{k+1}(s,a) &= \mathbb{E}[R_{t+1} + \gamma q_k(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a] && \text{(policy evaluation)} \\
q_{k+1}(s,a) &= \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_k(S_{t+1}, a') \mid S_t = s, A_t = a] && \text{(value iteration)}
\end{aligned}
$$

$$
\begin{aligned}
v(S_t) &\leftarrow v(S_t) + \alpha\big(R_{t+1} + \gamma v(S_{t+1}) - v(S_t)\big) && \text{(TD)} \\
q(S_t, A_t) &\leftarrow q(S_t, A_t) + \alpha\big(R_{t+1} + \gamma q(S_{t+1}, A_{t+1}) - q(S_t, A_t)\big) && \text{(SARSA)} \\
q(S_t, A_t) &\leftarrow q(S_t, A_t) + \alpha\big(R_{t+1} + \gamma \max_{a'} q(S_{t+1}, a') - q(S_t, A_t)\big) && \text{(Q-learning)}
\end{aligned}
$$



IN PLAIN WORDS

Each DP update above has a sampled TD version below, except value iteration for v. Can you explain why?

YOUR TURN

Hint: where does the max sit, relative to the expectation? Answer at the end.

On-policy and off-policy learning

ON-POLICY LEARNING

Learn about the behaviour policy π from experience sampled from π. (SARSA.)

OFF-POLICY LEARNING

Learn about a **target policy** π from experience sampled from a **behaviour policy** μ. Learn "counterfactually" about other things you could do: what if...?

- "What if I turned left?" New observations and rewards.
- "What if I played more defensively?" A different win probability.
- "What if I kept going forward?" How long until I bump into a wall?

THREE EXAMPLES

- **Robot vacuum**: explores with a random walk, but what you want is the best route.
- **Chess**: you only solve two-move puzzles, but want to play good full games.
- **Others**: learn from another agent, a human, or old logs.



IN PLAIN WORDS

In all three, the behaviour gathers the data, and the target is what we actually want.

Why off-policy matters

- Evaluate a target policy $\pi(a \mid s)$, computing v_π or q_π, while a behaviour policy $\mu(a \mid s)$ generates the actions. It lets us:
- learn from observing humans or other agents (for example, from logged data);
- re-use experience from old policies (your own past experience);
- learn about **multiple policies** while following one;
- learn about the **greedy** policy while following an exploratory one.



IN PLAIN WORDS

One stream of experience, many lessons. The same walk teaches the way to the kitchen and to the living room.

THE CATCH

The price: when μ and π are very different, learning is slower. And you can't learn about things you never do.

TRY IT

A hospital has years of logged treatment decisions made by doctors (μ). Off-policy learning can estimate how a new treatment policy π would have done, without trying it on patients, as long as π picks treatments the doctors sometimes picked.

Q-learning

- Q-learning estimates the value of the **greedy** policy, whatever the agent actually does next:

$$q(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha\big(R_{t+1} + \gamma \max_{a'} q(S_{t+1}, a') - q(S_t, A_t)\big)$$

- Acting greedily all the time would not explore enough, so the behaviour is usually ε-greedy.

THEOREM

Q-learning control converges to q_* as long as every action is taken in every state infinitely often, with step sizes $\sum_t \alpha_t = \infty$, $\sum_t \alpha_t^2 < \infty$ (for example $\alpha_t = 1/t^\omega$, $\omega \in (0.5, 1]$). No need for greedy behaviour!

TRY IT

The same step as before: $q(s, a) = 2$, reward -1, $q(s', \cdot) = (0, 4, 3)$, $\gamma = 0.9$, $\alpha = 0.5$.
Q-learning target: $-1 + 0.9 \times 4 = 2.6$; new $q = 2.3$.
SARSA, with the sampled first action: target -1, new $q = 0.5$.



IN PLAIN WORDS

One word changed: next action became best action.

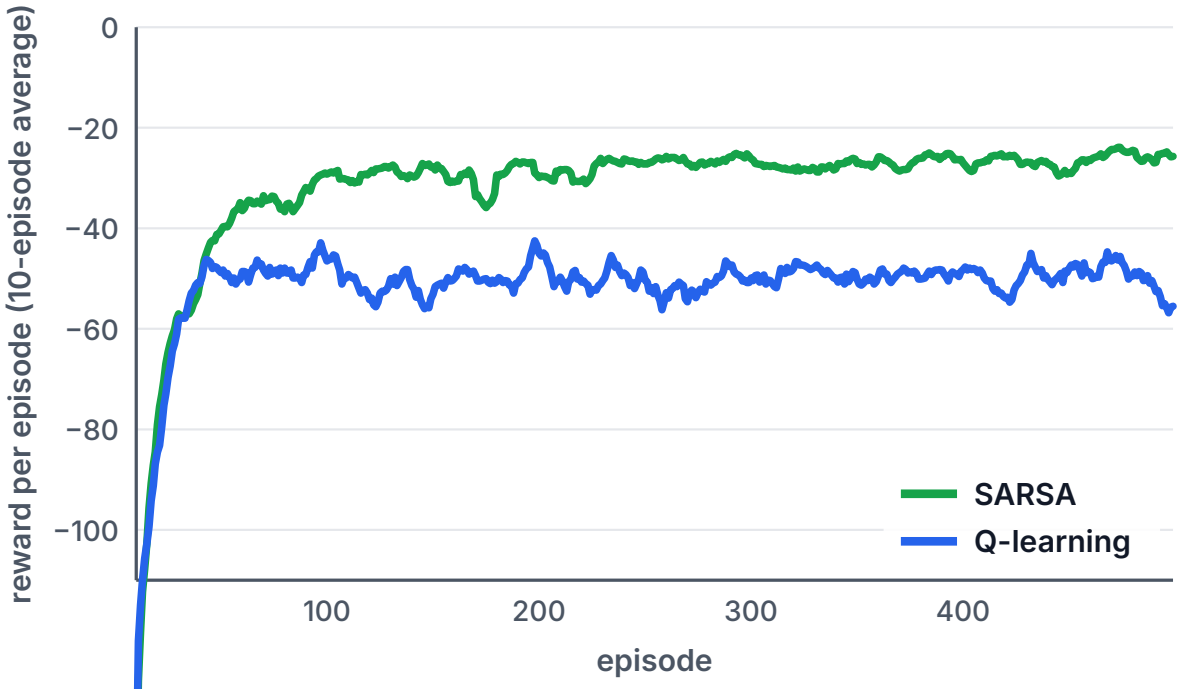
Example: cliff walking



Example 6.6 of Sutton & Barto (2018). Paths from our runs: 100 runs, 500 episodes, ε = 0.1, α = 0.5. Our drawing.

- A 4 × 12 grid. Start bottom left, goal bottom right.
- Stepping into the cliff: −100 and back to the start. Every other step: −1.
- The shortest path hugs the edge: 13 steps, a return of −13.
- Both learners explore with ε = 0.1.
- Q-learning finds the edge path. SARSA picks a path away from the edge: longer, but safe.

Cliff walking: reward while learning



Our runs: 100 runs averaged, then a 10-episode moving average.

LAST 100 EPISODES	SARSA	Q-LEARNING
reward per episode	-26.3	-49.9
falls per episode	0.045	0.333

- Q-learning learned the better (greedy) policy. SARSA earns more **while exploring**.
- Q-learning's values assume a perfect greedy walker, but the real walker still makes random moves, right next to the edge: it falls about once every three episodes.

So which one is better?

- SARSA's values include its own random moves, so it learns that the edge is dangerous for someone like itself.
- It depends on whether the falling is real:

REAL ROBOT

Next to a real staircase

SARSA's caution is worth a lot.

SIMULATOR

Exploration off later

Q-learning's path is the one you want.

- If ε is slowly reduced to 0, both converge to the optimal (edge) path.



IN PLAIN WORDS

There's no way to know in advance which works better on your problem. Many people start with SARSA and switch to Q-learning later; there's no promise that's better either.

FROM CLASS

Starting from scratch, Q-learning sticks with its first guesses until two estimates cross (a "flip"). On a reward curve that can look like noise for a long time, and then, suddenly, it works.

Check yourself: Part 3

CHECK YOURSELF

8. Is Q-learning on-policy or off-policy? What are its target and behaviour policies?

CHECK YOURSELF

9. $q(s, a) = 0$, reward 1, $q(s', \cdot) = (2, 6)$, the agent then takes the first action, $\gamma = 0.5$, $\alpha = 1$. New $q(s, a)$ under SARSA? Under Q-learning?

CHECK YOURSELF

10. In cliff walking, why does Q-learning earn less reward per episode even though it learned the shorter path?



Applied Computational Mathematics Lab

PART 4

Overestimation and double learning

The max is too optimistic. A trip to the casino.

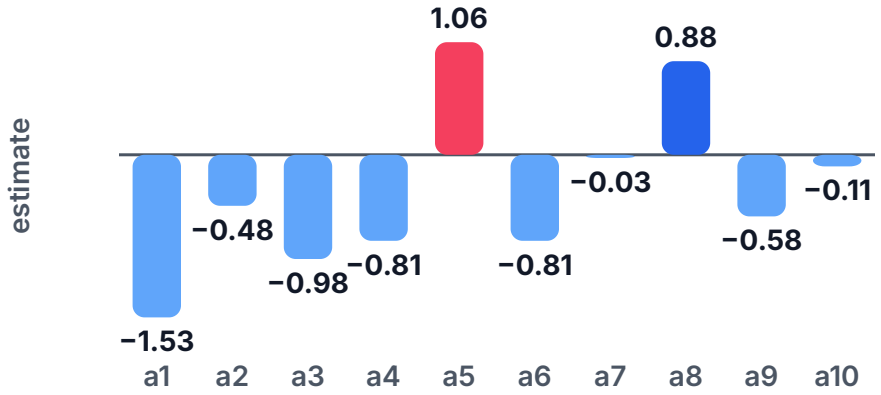


The max is too optimistic

- Q-learning uses the same values to **select** and to **evaluate**:

$$\max_a q(S_{t+1}, a) = q\big(S_{t+1}, \operatorname{argmax}_a q(S_{t+1}, a)\big)$$

- But the values are approximate: more likely to select overestimated values, less likely to select underestimated ones.
- This causes an **upward bias**, and Q-learning bootstraps on it at every step, so the error compounds.



Our run: ten actions all worth exactly 0, each estimated with standard normal noise. The max of the ten is on average 1.54, not 0 (200,000 trials).



IN PLAIN WORDS

Ten noisy zeros, and the winner always looks like a genius.

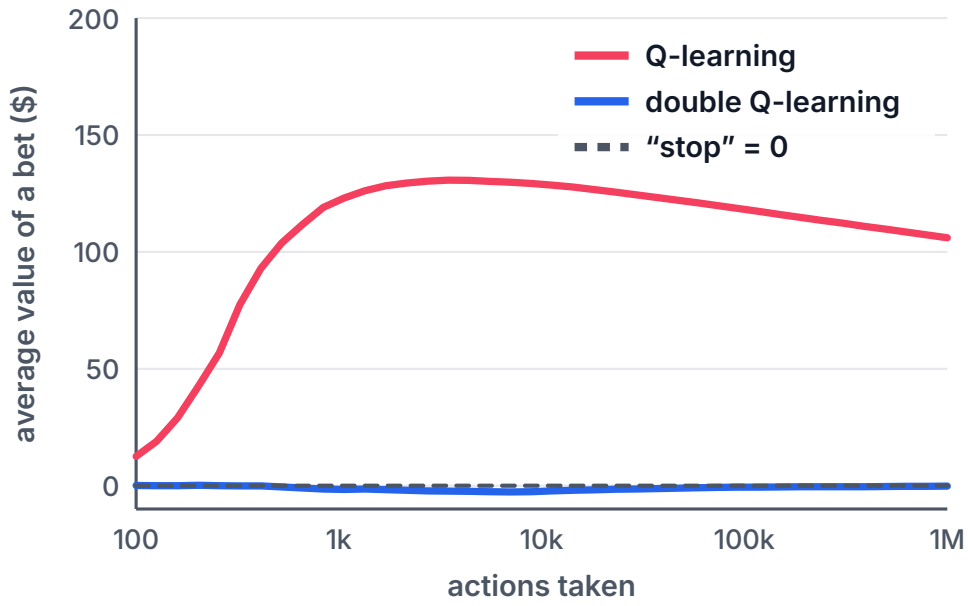
Example: roulette

- Our simplified roulette: one state, 170 actions that each bet $1 on one option, plus “stop”.
- “Stop” ends the episode with $0. Every bet has a high-variance reward with negative mean (about -0.053 per dollar), and the episode goes on.
- The right answer is to stop. We ran Q-learning on 1,000,000 uniformly random actions, $\gamma = 0.95$, $\alpha = 1/N(a)$.



IN PLAIN WORDS

Q-learning remembers the wins: after a million bets it thinks a typical bet is worth about $106. Every bet looks like a winner, so it never stops.



Our run. After a million actions, double Q-learning values a typical bet at -0.25 dollars: a bit gloomy, but its greedy action is “stop”. The setting follows the lecture; the numbers are ours.

Double Q-learning

- Decouple selection from evaluation. Store two action-value functions, q and q':

$$R_{t+1} + \gamma\, q'\big(S_{t+1}, \operatorname*{argmax}_a q(S_{t+1}, a)\big) \quad (1)$$

$$R_{t+1} + \gamma\, q\big(S_{t+1}, \operatorname*{argmax}_a q'(S_{t+1}, a)\big) \quad (2)$$

- Each step, pick q or q' (for example at random) and update it with target (1) for q or (2) for q'.
- Act with both, for example ε-greedy on $(q + q')/2$.
- Double Q-learning also converges to the optimal policy, under the same conditions as Q-learning.

TRY IT

In the next state, $q = (1.2, 0.4)$ and $q' = (-0.3, 0.5)$.
Q-learning bootstraps on $\max q = 1.2$.
Double: q picks action 1, q' scores it: -0.3.



IN PLAIN WORDS

If one table overestimates an action, the other has no reason to agree. So the bias washes out.

Double learning, more broadly

- The same idea scaled up to Atari games as **Double DQN** (van Hasselt, Guez & Silver, 2016), which used a target network (Lecture 8) and improved the scores on many of the games.
- Double learning generalises to other updates: if you are (soft-)greedy, for example ε-greedy, SARSA can also overestimate, and the same fix gives **double SARSA**.
- Double learning is not needed when there is no correlation between the target policy and the value function (for example, pure prediction).
- With a greedy target policy (Q-learning) the correlation is strong; then double learning helps.



IN PLAIN WORDS

Same data. One learner wants to stay at the casino all night. The other goes home.

ASIDE

Atari is what people played at the arcade: a wooden cabinet, a joystick, and another coin every time you lost. In RL research it became a standard test bed of dozens of games.

Check yourself: Part 4

CHECK YOURSELF

11. Why does taking a max over noisy estimates give a biased result, even if each estimate is unbiased?

CHECK YOURSELF

12. In double Q-learning, the next state has $q = (3, 1)$ and $q' = (0, 2)$. What does the update of q bootstrap on?

CHECK YOURSELF

13. Why does Q-learning in the roulette example never choose “stop”?



Applied Computational Mathematics Lab

PART 5

Importance sampling and expected SARSA

Correcting for following the wrong policy.



Importance sampling

- Off-policy learning: learn about π from experience generated by a different μ. Q-learning is one example; there are general tools.
- Goal: estimate the expectation of $f(X)$ under a target distribution d, with samples from a different distribution d'. Weight the data by d/d':

$$\mathbb{E}_{x\sim d}[f(x)] = \sum_x d(x)f(x) = \sum_x d'(x)\frac{d(x)}{d'(x)}f(x) = \mathbb{E}_{x\sim d'}\left[\frac{d(x)}{d'(x)}f(x)\right]$$

- Scale **up** events that are rare under d' but common under d.
- Scale **down** events that are common under d' but rare under d.



IN PLAIN WORDS

Formally, multiply and divide by d'. Multiplying by one never hurt anybody.

CAVEAT

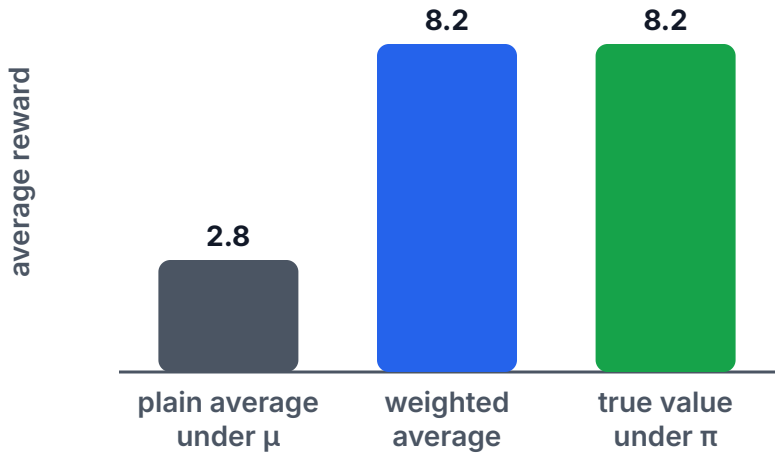
You can't expect to learn about things you never do: $d'(x)$ must be positive wherever $d(x)$ is.

A worked example

- Two actions: left pays 1, right pays 10.
- Behaviour μ: left 80% of the time. Target π: right 80% of the time.

ACTION	REWARD	μ	π	WEIGHT π/μ
left	1	0.8	0.2	0.25
right	10	0.2	0.8	4

- Plain average of μ's rewards: $0.8 \times 1 + 0.2 \times 10 = 2.8$. Wrong for π.
- Weighted: $0.8 \times 0.25 \times 1 + 0.2 \times 4 \times 10 = 8.2$, which is π's true average $0.2 + 8 = 8.2$.
- Under μ, the weight averages to $0.8 \times 0.25 + 0.2 \times 4 = 1$.





IN PLAIN WORDS

A left counts a quarter; a right counts four times.

Importance sampling for RL

- One-step reward, following μ:

$$\mathbb{E}[R_{t+1} \mid S_t = s, A_t \sim \pi] = \sum_a \mu(a \mid s) \frac{\pi(a \mid s)}{\mu(a \mid s)} r(s,a) = \mathbb{E}\left[\frac{\pi(A_t \mid S_t)}{\mu(A_t \mid S_t)} R_{t+1} \,\middle|\, S_t = s, A_t \sim \mu\right]$$

- So $\frac{\pi(A_t|S_t)}{\mu(A_t|S_t)} R_{t+1}$ is an unbiased sample. For **Monte Carlo**, weight the whole return by the probability ratio of the trajectory; the environment's probabilities cancel:

$$\frac{p(\tau_t \mid \pi)}{p(\tau_t \mid \mu)} G_t = \frac{\pi(A_t \mid S_t)\, p(R_{t+1}, S_{t+1} \mid S_t, A_t)\, \pi(A_{t+1} \mid S_{t+1}) \cdots}{\mu(A_t \mid S_t)\, p(R_{t+1}, S_{t+1} \mid S_t, A_t)\, \mu(A_{t+1} \mid S_{t+1}) \cdots} G_t = \frac{\pi(A_t \mid S_t)}{\mu(A_t \mid S_t)} \frac{\pi(A_{t+1} \mid S_{t+1})}{\mu(A_{t+1} \mid S_{t+1})} \cdots G_t$$

TRY IT

Ten steps, each with a ratio of 4: the weight is $4^{10} = 1,048,576$. One unlikely episode gets a weight of a million: the variance explodes.



IN PLAIN WORDS

No model needed: only the two policies' probabilities of the actions taken. But the ratios multiply, so a few rare episodes can dominate everything.

Importance sampling for TD

- Use TD targets generated by μ to evaluate π: weight the TD target by a **single** ratio:

$$v(S_t) \leftarrow v(S_t) + \alpha\left(\frac{\pi(A_t \mid S_t)}{\mu(A_t \mid S_t)}\left(R_{t+1} + \gamma v(S_{t+1})\right) - v(S_t)\right)$$

- Why it is right, in expectation under μ:

$$\begin{aligned}\mathbb{E}_\mu\left[\frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}\left(R_{t+1} + \gamma v(S_{t+1})\right) - v(S_t) \,\middle|\, S_t = s\right] &= \sum_a \pi(a \mid s)\,\mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) \mid s, a] - \sum_a \mu(a \mid s)\, v(s) \\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma v(S_{t+1}) - v(s) \mid S_t = s\right]\end{aligned}$$

- Much lower variance than Monte Carlo importance sampling; the policies only need to be similar over a single step.



IN PLAIN WORDS

That's one reason off-policy methods built on TD are far more common.

DETAIL

$\sum_a \mu(a \mid s)v(s) = \sum_a \pi(a \mid s)v(s) = v(s)$: both are probability distributions.

Expected SARSA

- Off-policy learning of action values needs **no importance sampling**.
- The next action may be chosen by the behaviour policy, $A_{t+1} \sim \mu(\cdot \mid S_{t+1})$, but we average over π:

$$q(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha\Big(R_{t+1} + \gamma \sum_a \pi(a \mid S_{t+1})\, q(S_{t+1}, a) - q(S_t, A_t)\Big)$$

- Called **expected SARSA** (sometimes "general Q-learning").
- Q-learning is the special case with a greedy target policy π: the average becomes the max.



IN PLAIN WORDS

A little more work per step, but no noise from sampling the next action.

TRY IT

$q(s', \cdot) = (2, 5, 1)$, π ε-greedy with $\varepsilon = 0.3$: probabilities $(0.1, 0.8, 0.1)$.
Expected SARSA uses $0.1 \times 2 + 0.8 \times 5 + 0.1 \times 1 = 4.3$.
SARSA, if it sampled action 1: 2. Q-learning: 5.

One family of updates

METHOD	TARGET	TARGET POLICY	ON / OFF
SARSA	$R + \gamma q(S', A')$	a sample of the behaviour	on-policy
Expected SARSA	$R + \gamma \sum_a \pi(a \mid S') q(S', a)$	any π	either
Q-learning	$R + \gamma \max_a q(S', a)$	greedy	off-policy
Double Q-learning	$R + \gamma q'(S', \operatorname{argmax}_a q(S', a))$	greedy, evaluated separately	off-policy

- Q-learning is akin to value iteration: it immediately estimates the current greedy policy.
- (Expected) SARSA is more like policy iteration: evaluate the current behaviour, then (immediately) update it.
- Learning about the greedy policy is a special case of off-policy learning.



IN PLAIN WORDS

SARSA samples, expected SARSA averages, Q-learning takes the max.

WRAP-UP

Summary

- Control without a model: learn q, improve greedily on it, and **explore** (ε-greedy; GLIE for convergence).
- **Monte Carlo control** and **SARSA** are on-policy: they learn the value of what they do, exploration included.
- **Q-learning** is off-policy: a greedy target, any exploring behaviour. Cliff walking shows the difference.
- The max over noisy estimates **overestimates**; **double learning** decouples selecting from evaluating.
- **Importance sampling** corrects for the behaviour policy: cheap for TD (one ratio), explosive for Monte Carlo (a product). **Expected SARSA** needs none.

$$\text{SARSA: } R + \gamma q(S', A')$$

$$\text{Q-learning: } R + \gamma \max_a q(S', a)$$

$$\text{Expected SARSA: } R + \gamma \sum_a \pi(a \mid S') q(S', a)$$

ANSWERS

Answers to check yourself

- **1.** Greedy on v needs p to know where each action leads; greedy on q just takes the largest entry.
- **2.** $0.8 + 0.2/5 = 0.84$; each other: 0.04.
- **3.** Fixed 0.1: no (it never becomes greedy). $1/k$: yes.
- **4.** up (4.2).
- **5.** Target $2 + 0.5 \times 4 = 4$; $q = 1 + 0.25 \times 3 = 1.75$.
- **6.** Neither. $1/\sqrt{t}$: its squares $1/t$ sum to infinity. $1/t^2$: the steps themselves have a finite sum.
- **7.** ε = 0.1 keeps adding random moves, and the wind punishes them.
- **8.** Off-policy: target greedy, behaviour (typically) ε-greedy.
- **9.** SARSA: $1 + 0.5 \times 2 = 2$. Q-learning: $1 + 0.5 \times 6 = 4$.
- **10.** It walks the edge, and its random moves make it fall about once every three episodes.
- **11.** The max picks whichever estimate got the luckiest noise.
- **12.** q picks action 1 (3); q' scores it: 0.
- **13.** Lucky wins inflate the bet values above 0, the value of “stop”.
- **Value iteration for v:** the max sits outside the expectation. One sample of the next state can’t tell us $\max_a \mathbb{E}[\cdot]$ without a model of every action’s outcome.

CREDITS

Sources and credits

- The order of topics follows *Lecture 6: Model-Free Control* of the DeepMind × UCL Reinforcement Learning Lecture Series (2021), by Hado van Hasselt. This deck is not an official DeepMind or UCL material.
- The text, figures, examples and simulations were written, drawn and run for ACML. Numbers labelled “our run” come from our own code; examples taken from books or papers are credited where they appear.
- Sutton & Barto (2018), chapters 5 and 6: the windy gridworld (6.5) and cliff walking (6.6) examples, re-run by us; van Hasselt (2010), double Q-learning; van Hasselt, Guez & Silver (2016), Double DQN. The roulette setting follows the lecture; our version and numbers are our own.
- Companion videos: **ACML Explains**, season 1, episodes 21–23. More from the lab: acml.teddylazebnik.com



Applied Computational Mathematics Lab

END OF LECTURE 6

Next: function approximation

Lecture 7 replaces the tables with functions: linear features, neural networks, and what can go wrong.

