Lecture 16: Reinforcement Learning



Gabriel Poesia
Assistant Professor, CSE
University of Michigan, Ann Arbor
poesia@umich.edu
https://gpoesia.com

Plan

1. Blackjack
2. Monte Carlo prediction
3. Monte Carlo control
4. Temporal-difference learning: SARSA and Q-learning

Administrivia

Proposal feedback posted last week - check Gradescope

Project milestone out last Friday, deadline pushed by 8 days (Nov 12th)

See handout on Ed

You should work towards a baseline result at least, + completion plan

Learning to act

Recall our goal: **learn to act optimally.**

Choose actions to collect as much reward as possible over time.

Last time: plan using a model.

Today: learn from experience.

Bandits and exploration

> 🧠 **Recall**
> Multi-armed bandits: no state, actions with random rewards but unknown means.

Canonical example of the exploration/exploitation trade-off.

Arm A

?

Unknown μ_A

Arm B

?

Unknown μ_B

Arm C

?

Unknown μ_C

Need to balance:

Exploitation

Take the action that looks best.

Exploration

Try an action to learn about it.

Markov decision processes

We can generalize to add states, leading us to MDPs:

Definition (Markov decision process)

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R)$$

- **States** $\mathcal{S}$: where are we?
- **Actions** $\mathcal{A}$: what can we do?
- **Transitions** $P(s' \mid s, a)$: where might that take us?
- **Rewards** R: what reward will we get?

Markov: next-state distribution depends only on current state + action.

Model-based planning

If we know P and R, we can find an optimal policy by planning.

P **and** R	→	**Plan**	→	V^*, π^*
Known environment model		Value / policy iteration		Optimal policy and value function

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

Evaluate π ⇄ **Improve π**

Today: model-free RL

Today we won't assume a model. We'll learn just from experience.

Still tabular RL:

Finite states.
Finite actions.

But no transition table

Need to act and see what happens.

We'll interact with the environment, get trajectories, learn, repeat

Our policy determines the experience we get. Need to explore again!

Blackjack

Blackjack

Let's learn some gambling!

Goal: beat the dealer by getting closer to 21, without going over.



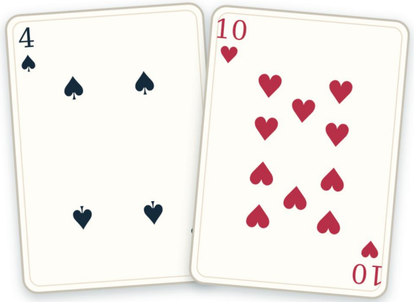
Each player starts with two cards.

One dealer card is face up; the other is hidden.

Going over 21 is a **bust**: player loses immediately.

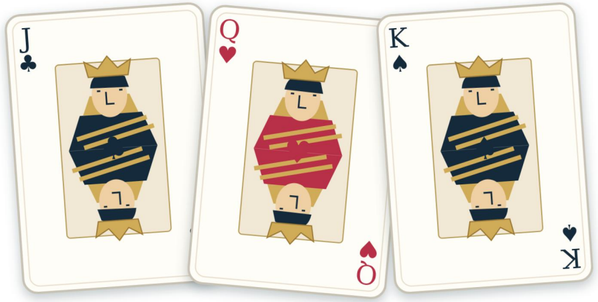
Card values

Hand's total is the sum of card values. Suits don't matter.



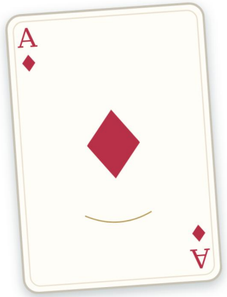
Face value

2, ..., 10



10

J, Q, K



1 or 11

Whichever doesn't bust

Blackjack rules

On each turn, you can take another card or stop. Two actions:

> **Hit**
>
> Draw one card. If the total exceeds 21, we lose.

> **Stick**
>
> Stop. Dealer plays out their hand.

The dealer reveals the hidden card and draws until reaching 17 or more.

If the dealer busts, we win. Otherwise, the higher total wins; equal totals draw.

Reward:



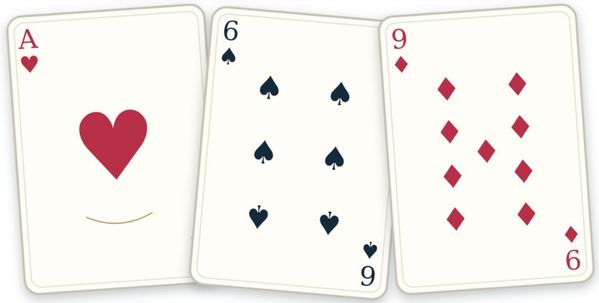
Reward is 0 until the game ends.

Simplified infinite-deck blackjack; see Sutton & Barto §5.1 and Gymnasium.

Usable aces

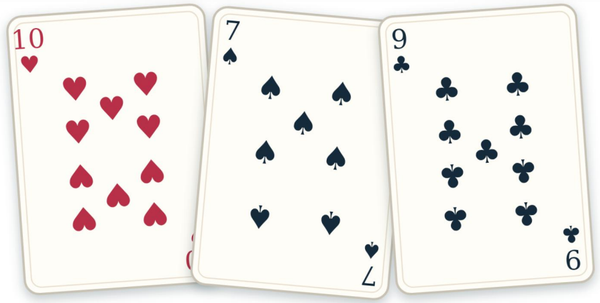
Definition (Usable ace)

A usable ace in your hand is an ace currently counted as 11 without making the total exceed 21.
If needed, it can be converted to 1 instead.



16 · still in the game!

1 + 6 + 9



26 · bust!

10 + 7 + 9

Q: Both hands start at 17. What if we draw a 9?

The sum alone doesn't tell us what can happen next.

Infinite-deck assumption

To keep the state small, we make one simplifying assumption:

Infinite-deck blackjack: Cards are drawn independently, with replacement.



$$\Pr(\text{ace}) = 1/13$$
$$\Pr(\text{10-valued card}) = 4/13$$

So past cards do not change the distribution of the next card.

So don't need to remember exact cards.

Blackjack states

With infinite deck, state only needs a triple:

> Player sum: 12 through 21
>
> Usable ace?
>
> Dealer showing card: A through 10

Below 12: automatically play "hit". We can't bust yet.

$10 \times 2 \times 10 = 200$ decision states

+ a terminal state.

Actions: hit or stick



Example: $s = (13,\ \text{no},\ 6)$

Blackjack decision



Example

Take another card?
Or stop here?

HIT **STICK**

Q: What would you do?

Episodic and continuing tasks

The game always ends, so blackjack is an **episodic environment**.

Episodic

$$\text{Act} \rightarrow \text{Act} \rightarrow \cdots \rightarrow \text{End}$$

Each episode ends after $\leq T$ steps.

Continuing

$$\text{Act} \rightarrow \text{Act} \rightarrow \cdots$$

No terminal step, e.g. a robot that keeps operating.

For an episode, the return from time t is:

$$G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$$

In our blackjack game, $\gamma = 1$, so G_t is the final outcome: $+1$, 0, or -1.

Environment interaction

We know the rules now, but our learner will only gets sampled outcomes.

State + action
$(13, \text{no}, 6) \cdot \text{hit}$

→

Play



→

Next state + reward
$(18, \text{no}, 6) \cdot 0$

$$(S_t, A_t, R_{t+1}, S_{t+1})$$

We can sample what happens, but not sum over P as before.

Policy iteration

🧠 **Recall**

Evaluate the current policy, then improve it, & repeat.

Policy π	→	**Evaluate**	→	**Improve**
How we play at each s		Estimate $V^{\pi}(s)$		Choose better actions

We will first learn to evaluate a fixed policy from experience, then improve it.

We don't have P now, but we can use samples instead.

Monte Carlo prediction

Policy evaluation

First, given a fixed policy π, how can we evaluate it?

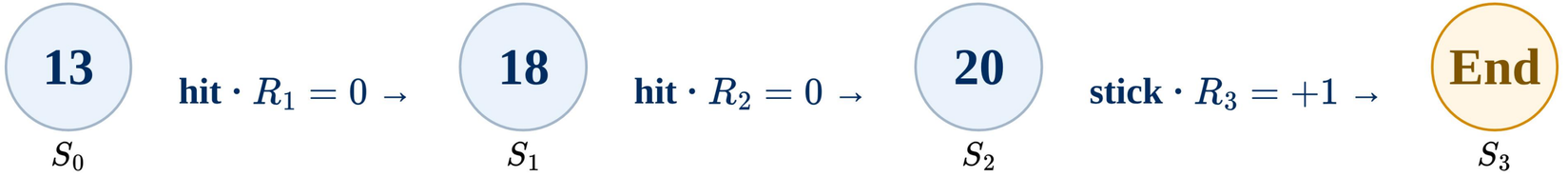
$V^\pi(s)$ is the expected return when we follow π from s:

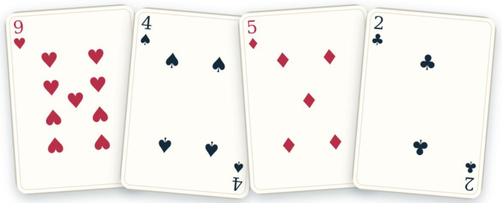
$$V^\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$$

Idea: use π to play games & collect data.

Episode returns

Suppose we play one game with π and get:





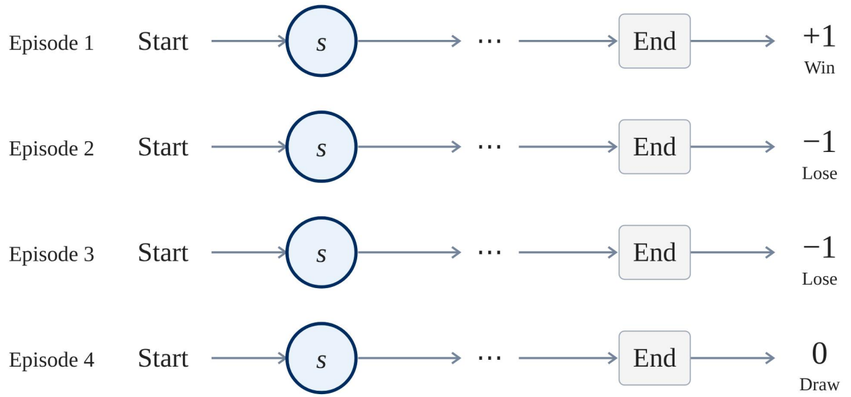
Say $\gamma = 1$, so only final reward is nonzero, $G_0 = G_1 = G_2 = +1$.

Monte Carlo estimates

Now play many games with the same π. Keep the returns from episodes that visit s.



Each return comes from a **complete trajectory** passing through s:



$$\hat{V}(s) = \frac{1 + (-1) + (-1) + 0}{4} = -0.25$$

> **Insight**
> Monte Carlo estimation of $V^\pi(s)$: average returns after visits to s.

Monte Carlo prediction

First-visit MC: play the fixed policy, compute returns, then update each visited state.

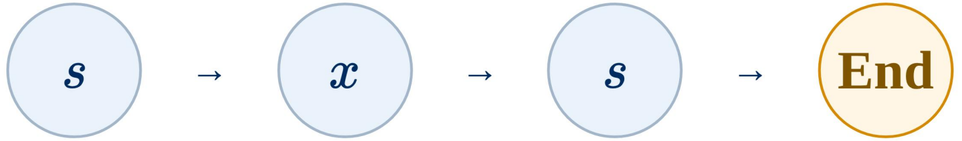
```
V, N = zeros(S), zeros(S)
for episode in range(num_episodes):
   trajectory = play(pi)
   returns = discounted_returns(trajectory, gamma)
   visited = set()
   for t, s in enumerate(trajectory.states[:-1]):
      if s not in visited:
         visited.add(s)
         N[s] += 1
         V[s] += (returns[t] - V[s]) / N[s]
```

The $1/N(s)$ stepsize makes $V(s)$ the sample average of returns.

Sutton & Barto, Reinforcement Learning: An Introduction, 2nd ed., chs. 5–6.

First-visit and every-visit MC

In a general episodic MDP, the same state may appear more than once in one episode.



(Although not in blackjack)

Two ways to compute returns:

First-visit MC

One return per state per episode.
Simpler convergence proof.

Every-visit MC

Use returns from all visits.
Returns within a game can be correlated.

Both converge for fixed-policy prediction with enough visits. (Just easier to prove for first-visit.)

Singh & Sutton (1996), Reinforcement Learning with Replacing Eligibility Traces, §3.

Example policies

We now have an evaluation method.

Two example policies we'll work through:

> π_{20}
>
> Hit below 20, stick if 20 or 21.

> π_{17}
>
> Hit below 17, stick if $\geq$ 17.

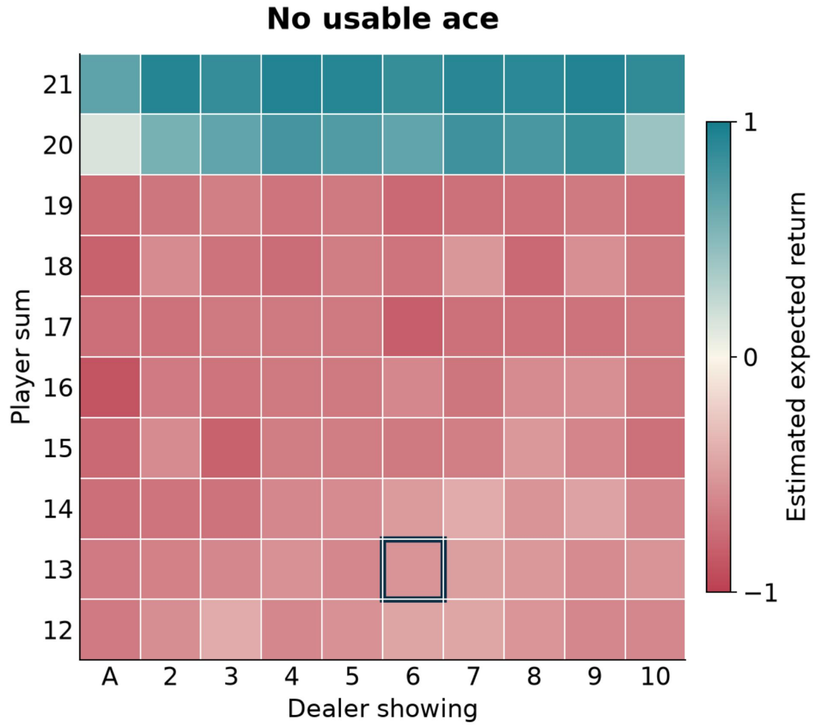
Both ignore the dealer's card and usable-ace flag.

Can play games under each policy and estimate V^{π} for visited states.

Value tables

Table shows estimates computed from 10,000 games.

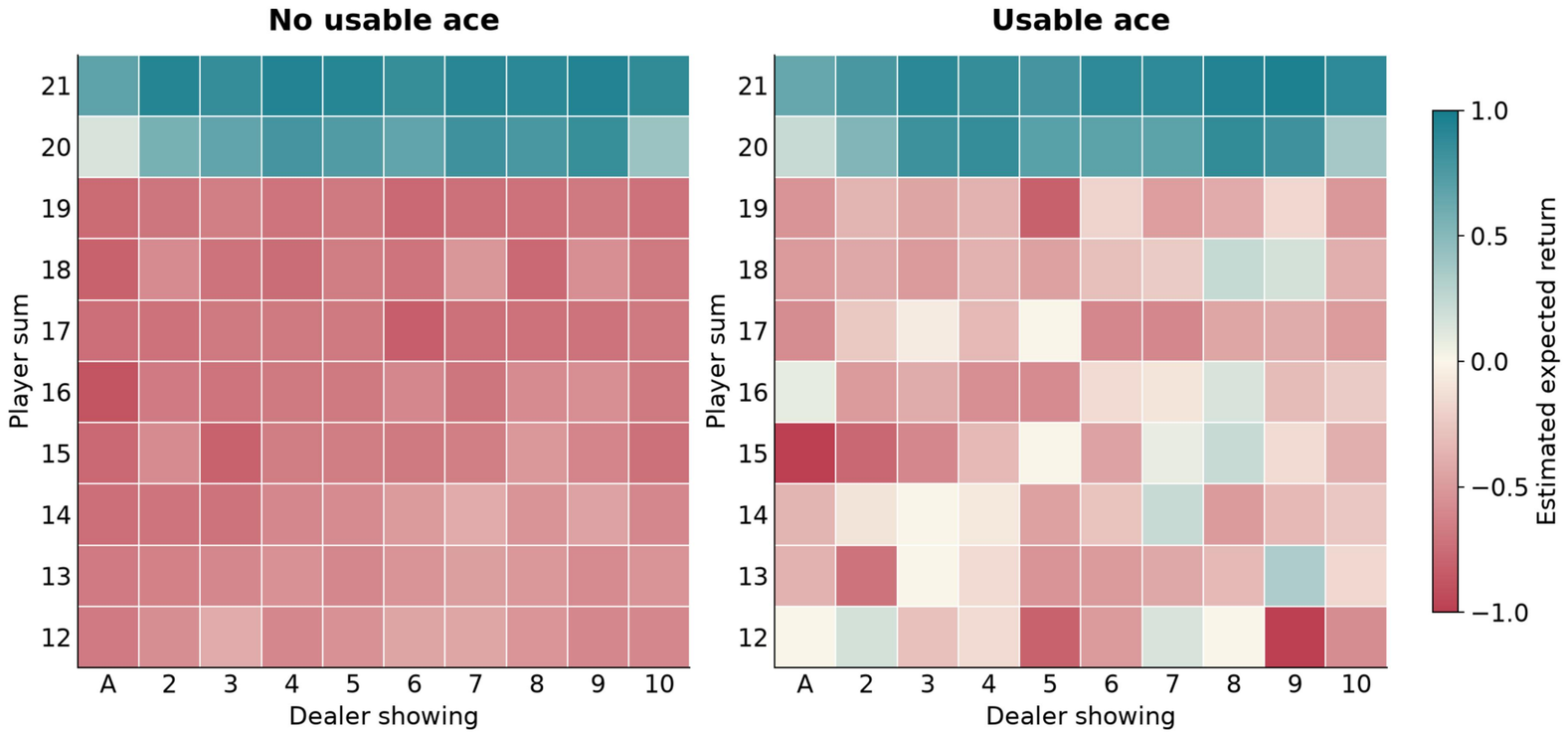




94 returns under π_{20} for this state: $\hat{V}(13, \text{no}, 6) \approx$ -0.53

Full MC prediction: 10,000 episodes

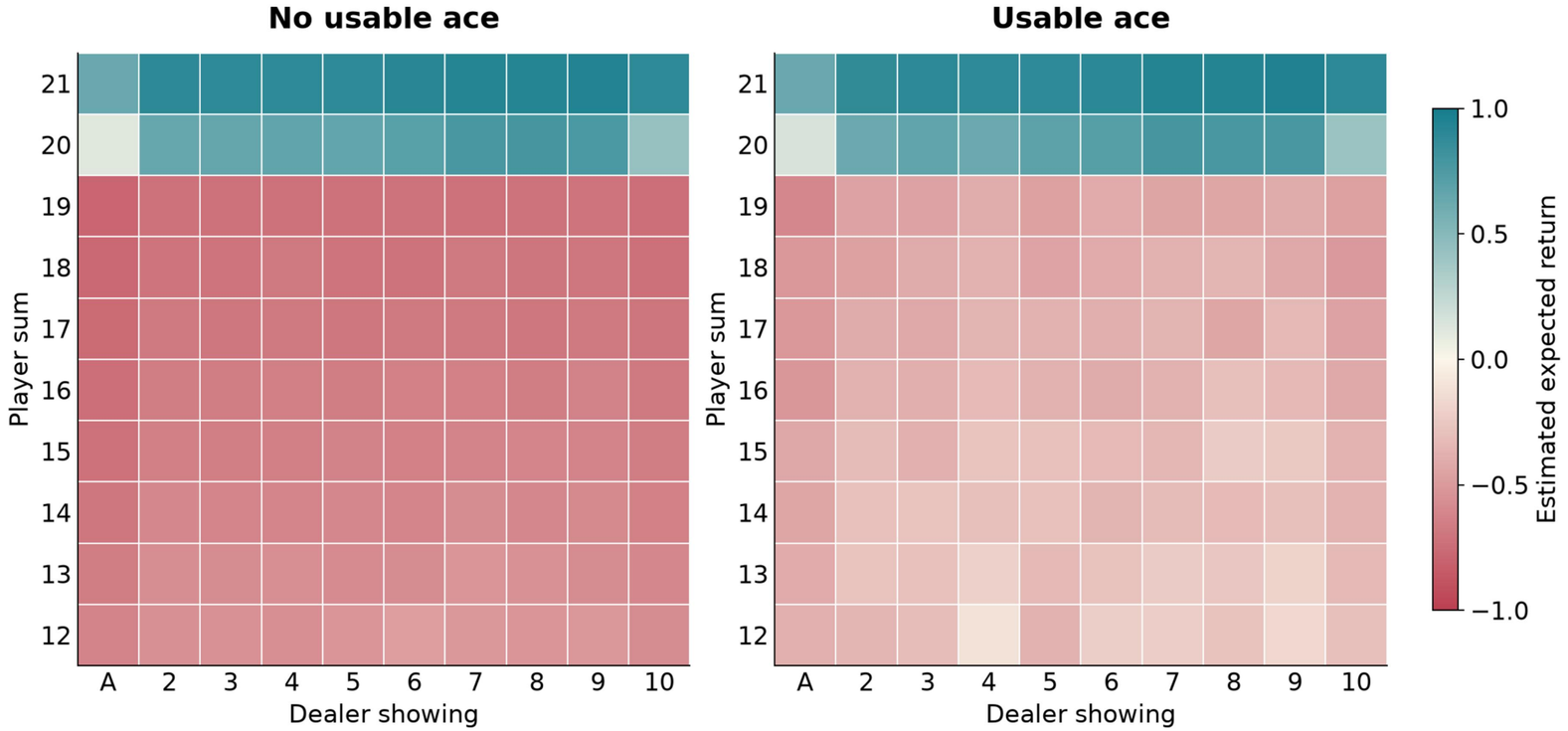
Showing with both usable aces and no usable aces:



Q: Which estimates should we trust least? Why?

MC prediction: 500,000 episodes

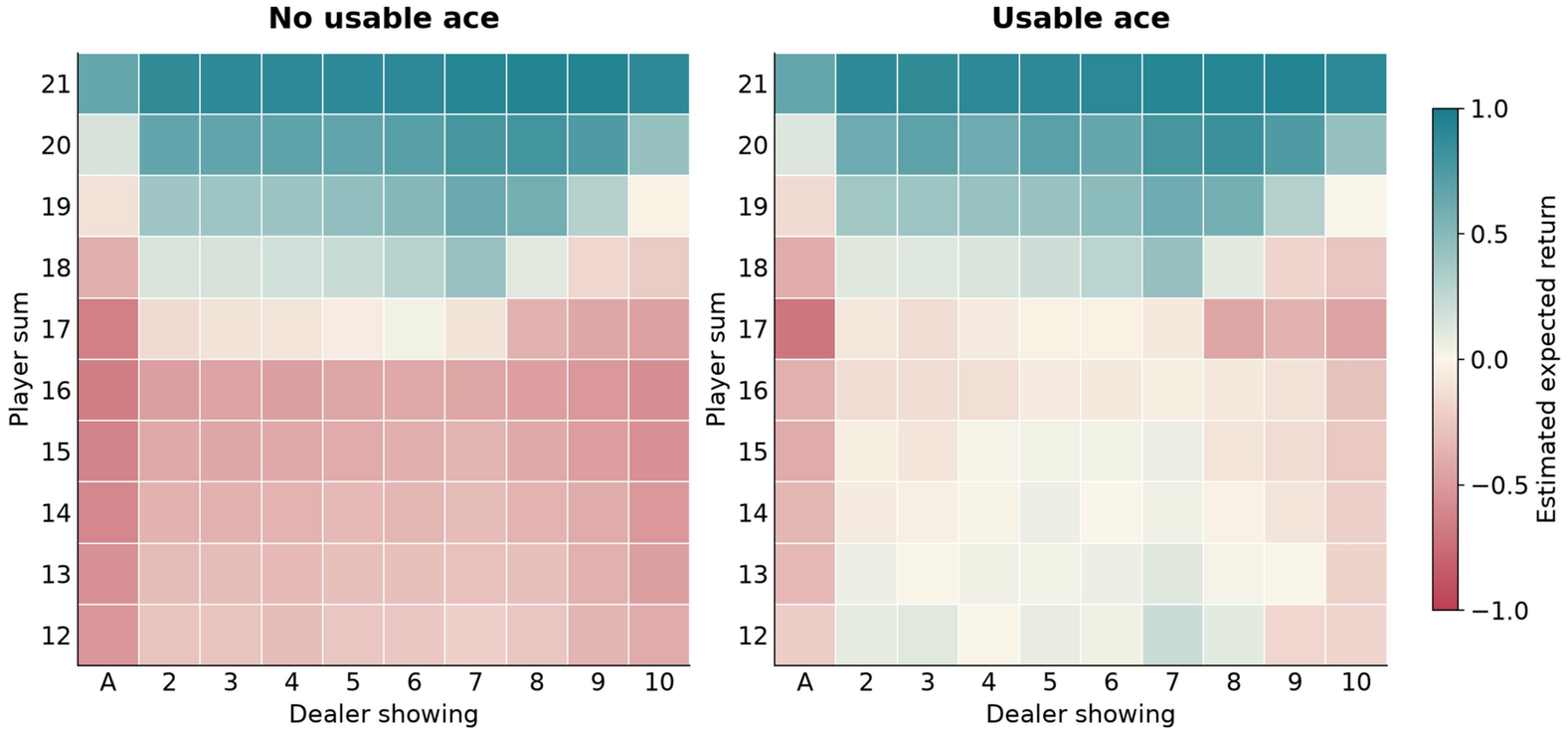
Keep π_{20} fixed and collect more experience: 500,000 complete games.



Better estimates. Same policy: we still hit until 20.

Policy comparison

Now evaluate π_{17} on 500,000 new games. It stops earlier than π_{20}.



Q: Where did the values change most?

Action values

We estimated V^π. Can we use it to improve π?



How good is hit?
How good is stick?

HIT **STICK**

$V^\pi(s)$ evaluates our policy's choice. We need to compare *both* actions.

MC action values

Same idea as before: choose action a in s, then follow π for the rest of the game.

Play action a in s → **Follow π** → **Observe G_t**

The expected return of that first action, followed by π, is the Q-function:

$$Q^\pi(s, a) = \mathbb{E}[G_t \mid S_t = s, A_t = a,\ \pi]$$

We can estimate $Q^\pi(s, a)$ incrementally as:

$$Q(s, a) \leftarrow Q(s, a) + \frac{1}{N(s, a)}[G_t - Q(s, a)]$$

Monte Carlo control

Exploration

How can we improve π now? We can compare $Q(s, a)$ across actions.

Caveat: need data even for actions π never chooses!



Suppose we played with π_{20}. It never tries sticking here!

Q: How would we discover that sticking might be better?

Exploring starts

Simple idea: start MC evaluation from all possible states and try all first actions.



Start at any state.
Try either action.

HIT **STICK**

Even if π always hits here, we can try sticking.

Give every starting (s, a) a chance. Then follow π.

This is called **exploring starts**.

Sutton & Barto, Reinforcement Learning: An Introduction, 2nd ed., chs. 5–6.

Policy improvement

Exploring starts gives us data for both first actions, following π_{20} after.



Try either first action then follow π_{20}

First action	Estimated $Q^{\pi_{20}}$
Hit	-0.56
Stick	**-0.15**

Q: How would you change π_{20}?

Stick here. Better estimated return.

MC control with exploring starts

MC control ES: Alternate MC evaluation with greedy policy improvement, using exploring starts.

```
Q, N, pi = zeros(S, A), zeros(S, A), initial_policy()
for episode in range(num_episodes):
   s0, a0 = random_state(), random_action()
   trajectory = play(pi, start=s0, first_action=a0)
   returns = discounted_returns(trajectory, gamma)
   for s, a, G in first_visits(trajectory, returns):
      N[s, a] += 1
      Q[s, a] += (G - Q[s, a]) / N[s, a]
   for s in visited_states(trajectory):
      pi[s] = greedy_action(Q[s], prefer=pi[s])
```

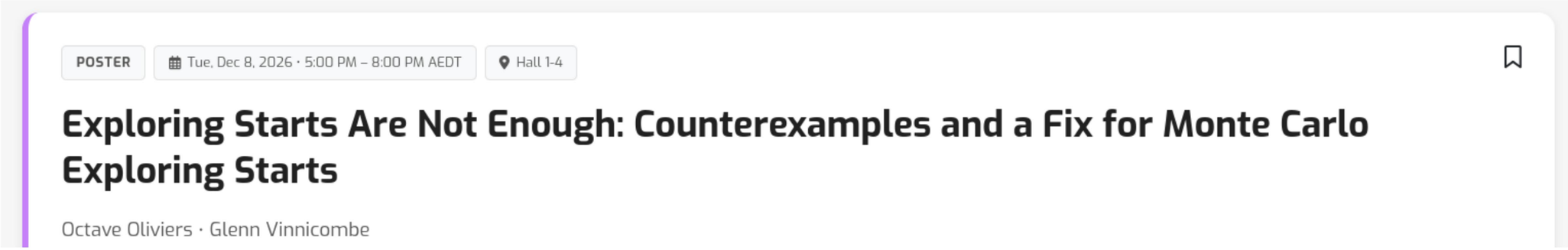
Here first_visits returns the first occurrence of each state-action pair, on ties we keep the old action.

Sutton & Barto, Reinforcement Learning: An Introduction, 2nd ed., chs. 5–6.

MC control: convergence

Does this converge to π^*?

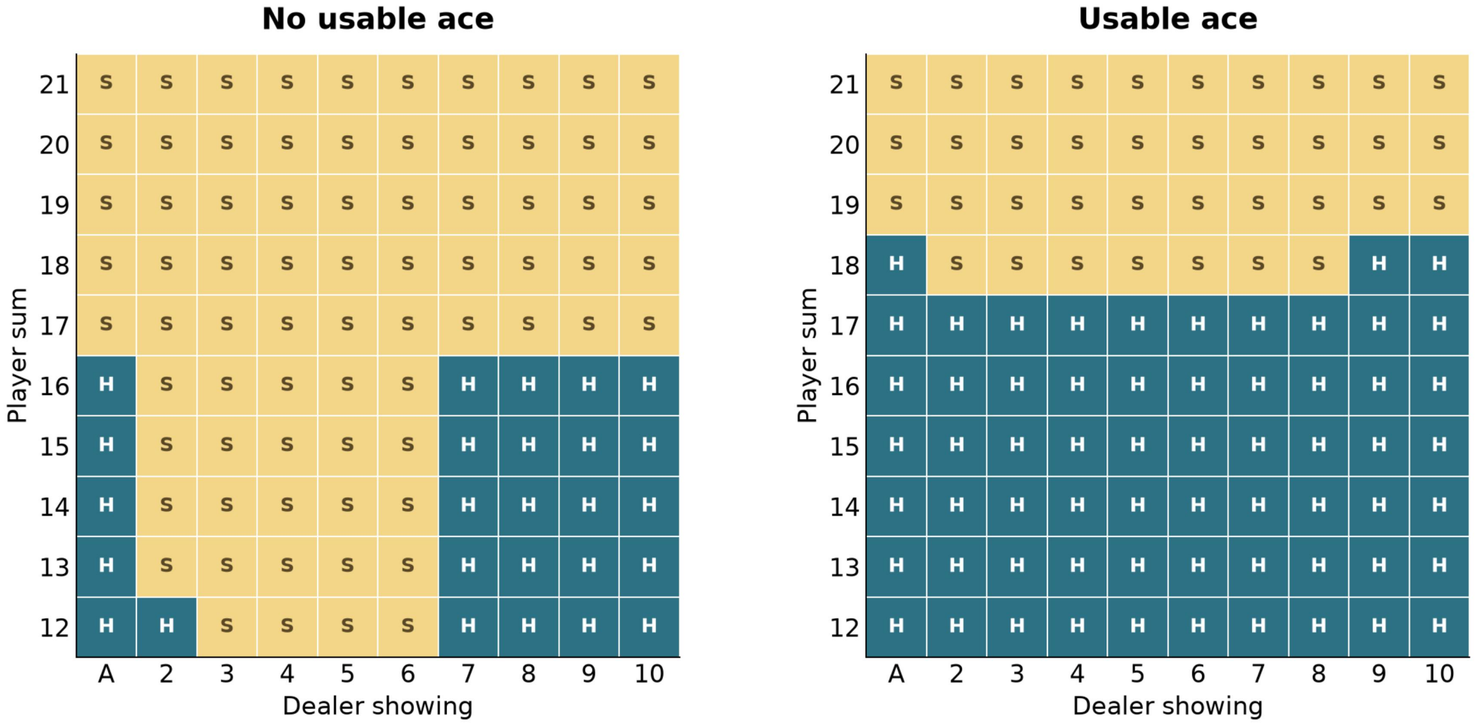
Works well in practice, but convergence was an open problem until this year!!!



Learned blackjack policy

After 2,000,000 exploring-start episodes, the greedy policy uses both the player and dealer cards.

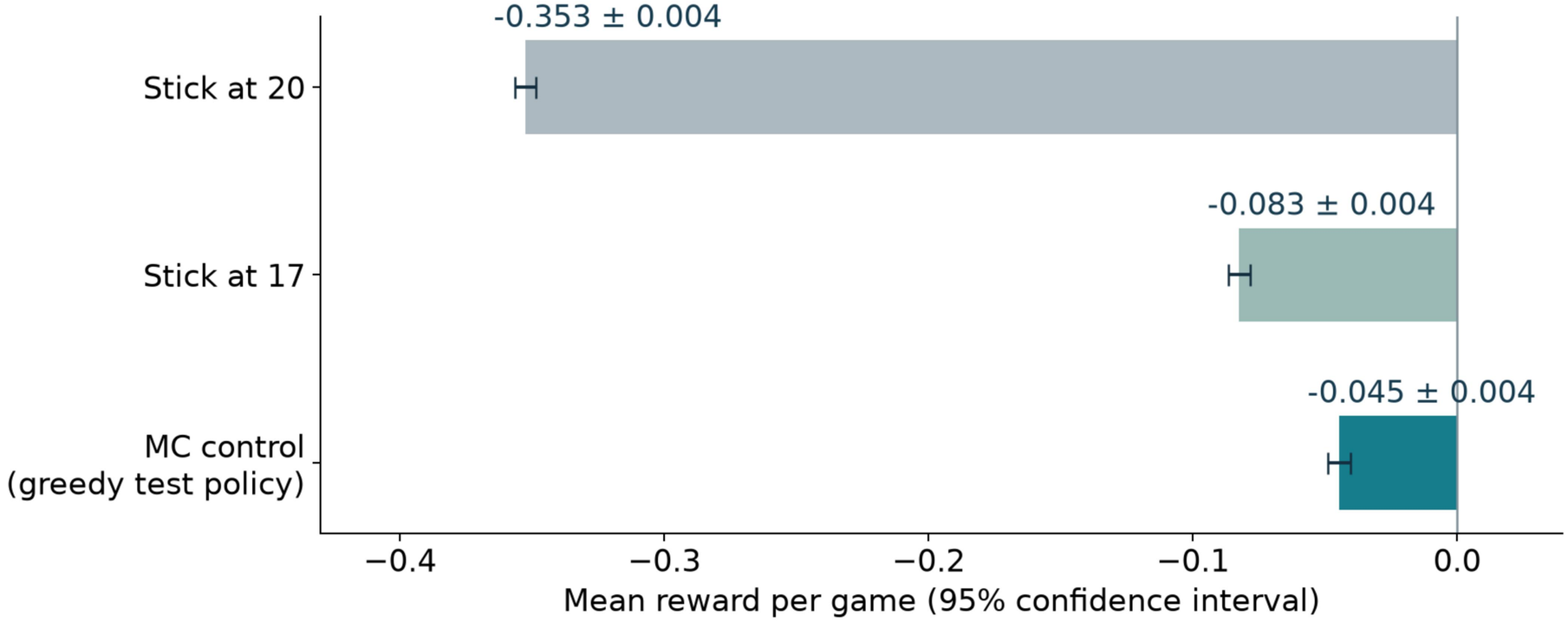
H = hit, S = stick



Q: In our example opening hand, we had sum 13, no ace, dealer had 6. What do we play now?

Policy returns

Comparing learned greedy policy with our two fixed policies on 200k games:



Mean reward is $\Pr(\text{win}) - \Pr(\text{lose})$.

Even with close to optimal policy, the dealer has an edge. The house always wins!

Epsilon-greedy control

We can't always decide where to start. Can we avoid that?

Keep a greedy policy π, but act with an ϵ-greedy **behavior policy** b:

Generally exploit	**Sometimes explore**
Take $\pi(s)$: the action with the best current Q.	Choose a random action.
Probability $1 - \epsilon$	Probability ϵ

With 2 actions and $\epsilon = 0.1$: b takes $\pi(s)$ 95% of the time (why?).

This is **off-policy learning**: collect data with b, to learn about a different policy π.

Complication for MC: returns include later exploration by b. Averaging them estimates Q^b, not Q^π.

Need to correct for the different policies, e.g. with **importance sampling**.

Another limitation of MC: we need the game's complete return before updating.

Sutton & Barto, Reinforcement Learning: An Introduction, 2nd ed., chs. 5–6.

Temporal-difference learning

Learning after each transition

First, let's go back to evaluating a fixed π, using experience from π itself.

MC can update after each game, but still has to wait for the game to end.

Could we learn incrementally, even from just seeing one transition?



Monte Carlo

Wait for the complete return G_t.

But we already have a guess...

$V(18, \text{no}, 6) = 0.30$

Could it teach us something about 13?

TD update example

Suppose these are our current value estimates for π. We can use the observed transition to revise $V(s)$!



New guess for 13

Reward 0 + estimated future 0.30

(with $\gamma = 1$)

Move α toward that guess

$0.10 + \alpha(0.30 - 0.10)$

$= 0.12$ (say, with $\alpha = 0.1$)

We learned something before finishing the game.

This is the idea of **temporal-difference learning**.

TD prediction

Generalizing, we replace the unknown future return by current estimate of the next state's value.

Recall we have:

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

We don't know G_{t+1} yet, but we do have a current estimate of it: $V(S_{t+1})$.

Observed reward	+	**Estimated future**
R_{t+1}		$\gamma V(S_{t+1})$

$$V(S_t) \leftarrow V(S_t) + \alpha \underbrace{[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]}_{\delta_t \text{ : TD error}}$$

This is the $TD(0)$ update: one sampled transition, model-free

Bootstrapping

Model-free like MC

Unlike MC, TD can learn from incomplete episodes.

Also unlike MC, TD **bootstraps**:

> **Definition (Bootstrapping)**
> Updating one estimate using another estimate.

TD estimates the future from V, whereas MC waited for the complete return.

TD can learn online, including in continuing tasks (i.e., non-episodic case).

TD control

Back to control: learn Q instead of V, improve the policy, and keep exploring with b.

Same choice as before: learn about b itself, or a separate greedy target π?

For TD, this changes which next action's Q-value we put in the target.

Exploratory actions

Suppose we explore hit at 13 and arrive at 18 with reward 0. Now the policy chooses another action.



Illustrative current Q estimates:

Next action	$Q(s', a)$
Stick	$+0.28$
Hit	-0.45

Behavior b explores and chooses **hit**.

Q: Which value should teach the previous decision?

SARSA

SARSA learns about the exploring behavior b: use the next action b actually selected.

13	**Hit**	**0**	**18**	**Hit**
State S_t	Action A_t	Reward R_{t+1}	State S_{t+1}	Action A_{t+1}

Use the action we actually selected next: $Q(s', \text{hit}) = -0.45$.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \right]$$

State, Action, Reward, State, Action: SARSA

Sutton & Barto, Reinforcement Learning: An Introduction, 2nd ed., chs. 5–6.

On-policy learning

With SARSA, the behavior and target policies are the same: $\pi = b$.

Behavior policy b

Which actions generate the data?

ϵ-greedy

Target policy π

Whose continuation are we learning?

The same exploring policy: $\pi = b$

SARSA's target includes our future exploratory actions.

Q-learning

Q-learning keeps the two policies separate: act with b, learn a greedy target π.

Or use the best next action:
$\max_a Q(s', a) = +0.28$.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

Behavior

Explore, e.g. ϵ-greedy.

Target

Use the greedy next action.

SARSA and Q-learning

With the same table and the same observed transition, the two methods use different targets.

Example: on $13 \xrightarrow{\text{hit},\ R=0} 18$ transition, suppose behavior policy chooses hit from state 18

SARSA	**Q-learning**
$y = 0 + Q(s', \text{hit})$	$y = 0 + \max_a Q(s', a)$
-0.45	$+0.28$
The next action actually selected	The best estimated next action

$$Q(s, \text{hit}) = 0.20,\ \alpha = 0.1 \quad \Rightarrow \quad 0.135 \text{ vs. } 0.208$$

Off-policy learning

Same separation as in off-policy MC: the policy generating data differs from the target.

Behavior policy b

Produces the data.

May choose exploratory actions.

Target policy π

The continuation we learn about.

Q-learning: greedy with respect to Q.

$$b \text{ can differ from } \pi$$

One-step Q-learning uses a greedy next-state estimate, so it needs no full-trajectory importance weights.

The learner still needs experience of every relevant state-action pair.

Control targets

All three methods update the chosen action toward a target y; they differ in how they form it.

$$Q(s,a) \leftarrow Q(s,a) + \alpha[\, y - Q(s,a) \,]$$

Method	Target y	When?
On-policy MC control	G_t	End of game
SARSA	$R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$	After a step + next action choice
Q-learning	$R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$	After a step

For TD at a terminal transition, $y = R_{t+1}$.

Choosing a control method

When should we use SARSA vs. Q-learning?

SARSA

Improve the behavior policy, including its exploration.

Useful when reward matters **while learning**.

Example: a robot learning in a house.

Q-learning

Learn values for a greedy policy while exploring.

Useful when we'll **switch to greedy actions later**.

Example: train a game agent in a simulator, then play greedily.

Summary and next lecture

Model-free RL: learn values and improve a policy directly from experience.

Monte Carlo	**Temporal-difference learning**
Learn from complete returns, after the episode ends.	Bootstrap from current estimates, after each transition.

Today: tables for finite state and action spaces.

Next lecture: function approximation and policy gradients for deep RL.