Temporal Difference Learning

CMPUT 261: Introduction to Artificial Intelligence

S&B §6.0-6.2, §6.4-6.5

Lecture Overview

1. Recap & Logistics
2. TD Prediction
3. On-Policy TD Control (Sarsa)
4. Off-Policy TD Control (Q-Learning)
5. Expected Sarsa

After this lecture, you should be able to:

- trace an execution of the TD(0) algorithm
- trace an execution of the Q-learning algorithm
- trace an execution of the Sarsa algorithm
- define bootstrapping
- explain why bootstrapping is useful
- trace an execution of the Expected Sarsa algorithm
- describe the advantages of Expected Sarsa over Sarsa

Logistics

- **Assignment #4** is due **April 11** at 11:59pm
  - Late submissions for 20% deduction until **April 13** at 11:59pm
- **SPOT** (former USRI) surveys are now available: https://p20.courseval.net/etw/ets/et.asp?nxappid=UA2&nxmid=start
  - Available until **April 14**

Recap: Monte Carlo RL

- **Monte Carlo** estimation: Estimate **expected returns** to a state or action by averaging **actual returns** over **sampled trajectories**
    - Estimating **action values** requires either **exploring starts** or a **soft policy** (e.g., ϵ-greedy)
- **Off-policy learning** is the estimation of value functions for a **target policy** based on episodes generated by a different **behaviour policy**
- **Off-policy control** is learning the **optimal policy** (target policy) using episodes from a **behaviour policy**

Importance Sampling

Monte Carlo: For $X \sim f$, can use a sample $x_1, \ldots, x_n$ drawn from f :

$$\mathbb{E}[X] \doteq \sum_x f(x)x \approx \frac{1}{n}\sum_{i=1}^{n} x_i$$

Importance Sampling: For $X \sim f$, can use a sample $x_1, \ldots, x_n$ drawn from g :

$$\mathbb{E}[X] \doteq \sum_x f(x)x = \sum_x g(x)\boxed{\frac{f(x)}{g(x)}}x \approx \frac{1}{n}\sum_{i=1}^{n} \boxed{\frac{f(x_i)}{g(x_i)}}x_i$$

Importance ratio

Recap: Off-Policy Monte Carlo Prediction

Off-policy MC prediction (policy evaluation) for estimating $Q \approx q_\pi$

Input: an arbitrary target policy π
Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
    $Q(s,a) \in \mathbb{R}$ (arbitrarily)
    $C(s,a) \leftarrow 0$

Loop forever (for each episode):
    $b \leftarrow$ any policy with coverage of π
    Generate an episode following b: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    $W \leftarrow 1$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$, while $W \neq 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t,A_t)} [G - Q(S_t, A_t)]$
        $W \leftarrow W \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$

Recap: Off-Policy Monte Carlo Control

Off-policy MC control, for estimating $\pi \approx \pi_*$

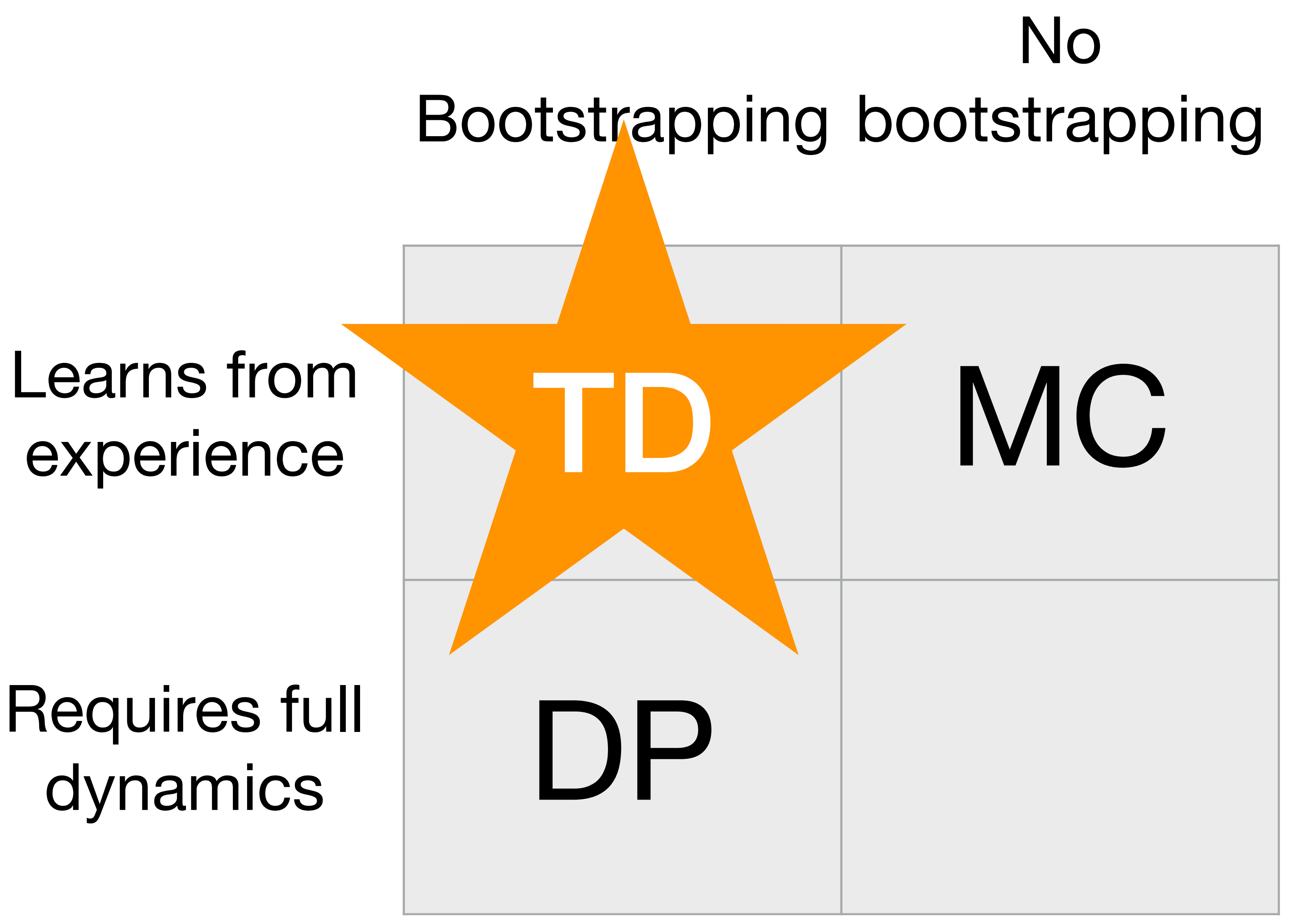
Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
- $Q(s, a) \in \mathbb{R}$ (arbitrarily)
- $C(s, a) \leftarrow 0$
- $\pi(s) \leftarrow \arg\max_a Q(s, a)$ (with ties broken consistently)

Loop forever (for each episode):
- $b \leftarrow$ any soft policy
- Generate an episode using b: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
- $G \leftarrow 0$
- $W \leftarrow 1$
- Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
  - $G \leftarrow \gamma G + R_{t+1}$
  - $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
  - $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
  - $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken consistently)
  - If $A_t \neq \pi(S_t)$ then exit inner Loop (proceed to next episode)
  - $W \leftarrow W \frac{1}{b(A_t|S_t)}$

Learning from Experience

- Suppose we are playing a blackjack-like game **in person**, but we **don't know the rules**.
  - We know the actions we can take, we can see the cards, and we get told when we win or lose
- **Question:** Could we compute an optimal policy using **dynamic programming** in this scenario?
- **Question:** Could we compute an optimal policy using **Monte Carlo**?
  - What would be the **pros and cons** of running Monte Carlo?

Bootstrapping



- Dynamic programming **bootstraps**: Each iteration's estimates are based partly on **estimates from previous iterations**
- Each Monte Carlo estimate is based only on **actual returns**

Updates

Dynamic Programming: $V(S_t) \leftarrow \sum_a \pi(a \mid S_t) \sum_{s',r} p(s', r \mid S_t, a)\left[r + \gamma V(s')\right]$

Monte Carlo: $V(S_t) \leftarrow V(S_t) + \alpha \left[G_t - V(S_t)\right]$

TD(0): $V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right]$

$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] .
\end{aligned}
$$

Monte Carlo: Approximate because of $\mathbb{E}$

Dynamic programming: Approximate because v_π not known

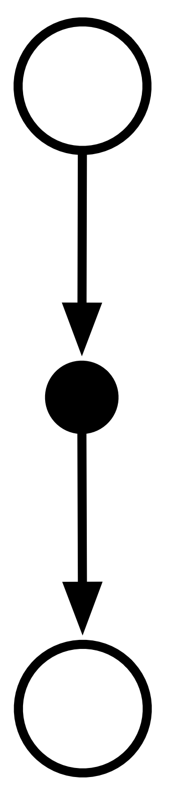
TD(0): Approximate because of $\mathbb{E}$ ***and*** v_π not known

TD(0) Algorithm

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\textit{terminal}) = 0$

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        $A \leftarrow$ action given by π for S
        Take action A, observe R, S'
        $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
        $S \leftarrow S'$
    until S is terminal



Question: What information does this algorithm use?

TD for Control

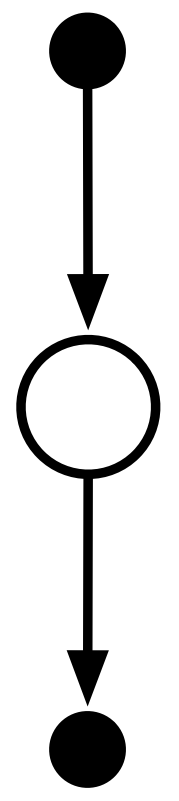
- We can plug TD prediction into the **generalized policy iteration** framework
- **Monte Carlo control loop:**
    1. Generate an **episode** using estimated π
    2. Update estimates of Q and π
- **On-policy TD control loop:**
    1. Take an **action** according to π
    2. Update estimates of Q and π

On-Policy TD Control

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
    Initialize S
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Loop for each step of episode:
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ε-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until S is terminal



Question: What information does this algorithm use?

Question: Will this estimate the Q-values of the optimal policy?

Actual Q-Values vs. Optimal Q-Values

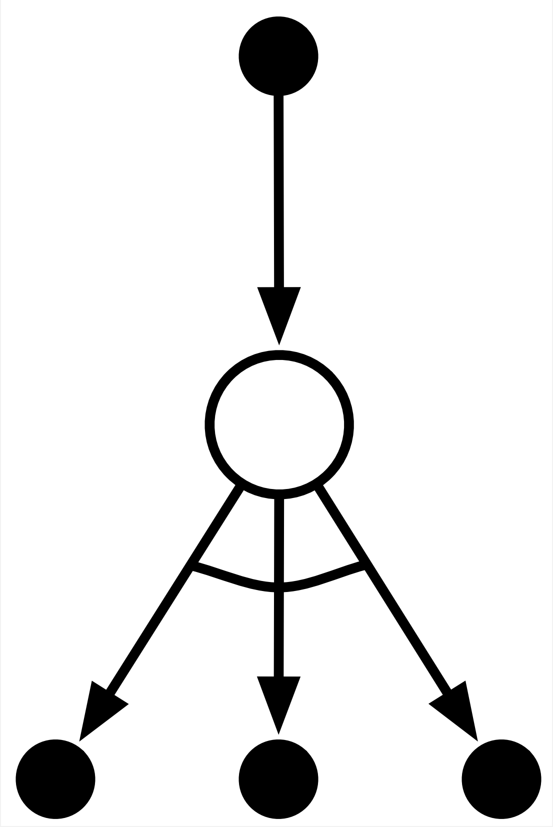
- Just as with on-policy Monte Carlo control, Sarsa does not converge to the **optimal** policy, because it always chooses an **ϵ-greedy action**
    - And the estimated Q-values are with respect to the **actual actions**, which are ϵ-greedy
- **Question:** Why is it necessary to choose ϵ-greedy actions?
- What if we **acted** ϵ-greedy, but **learned the Q-values** for the optimal policy?

Off-Policy TD Control

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

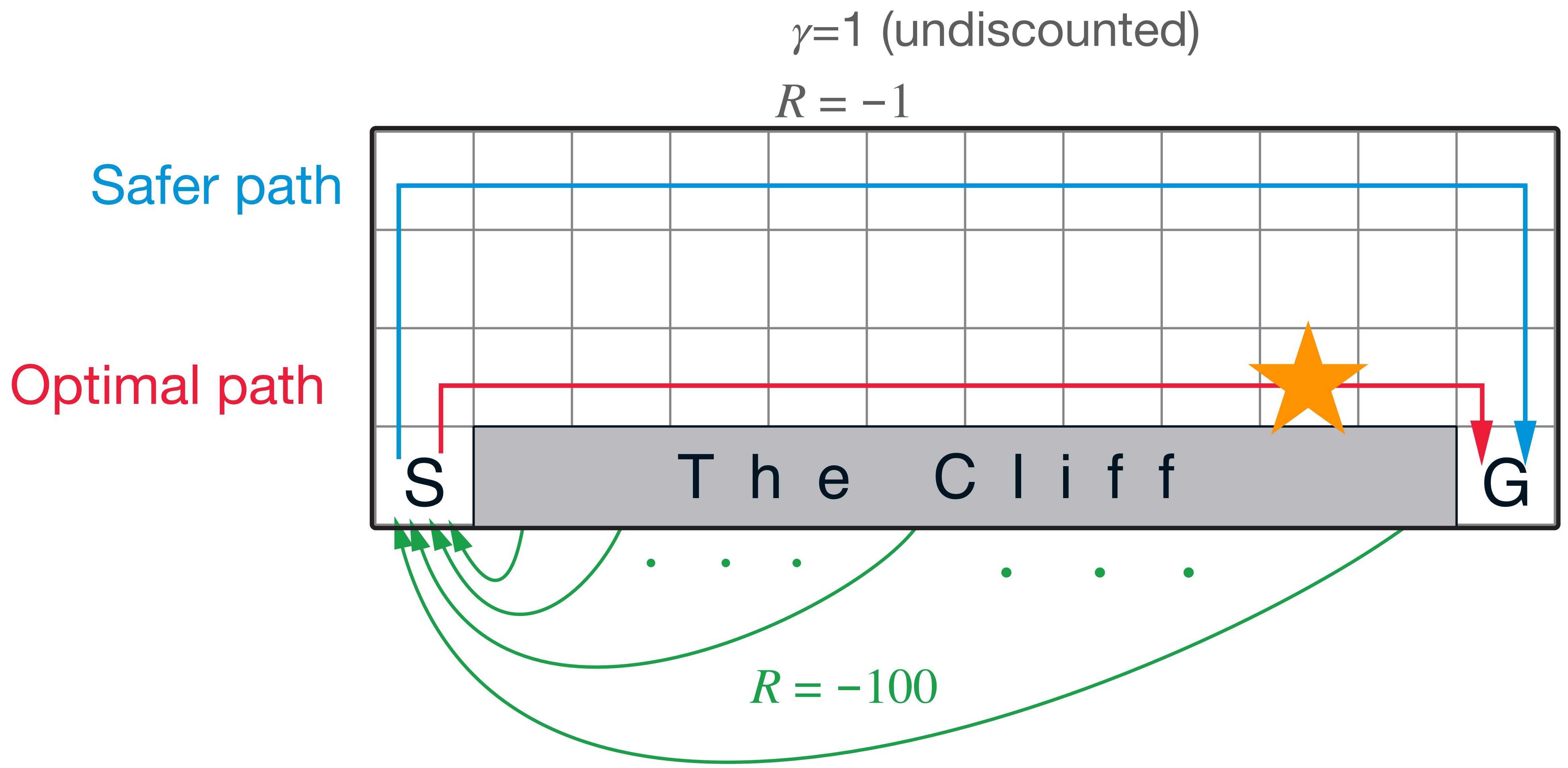
Loop for each episode:
    Initialize S
    Loop for each step of episode:
        Choose A from S using policy derived from Q (e.g., ε-greedy)
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
        $S \leftarrow S'$
    until S is terminal



Question: What **information** does this algorithm use?

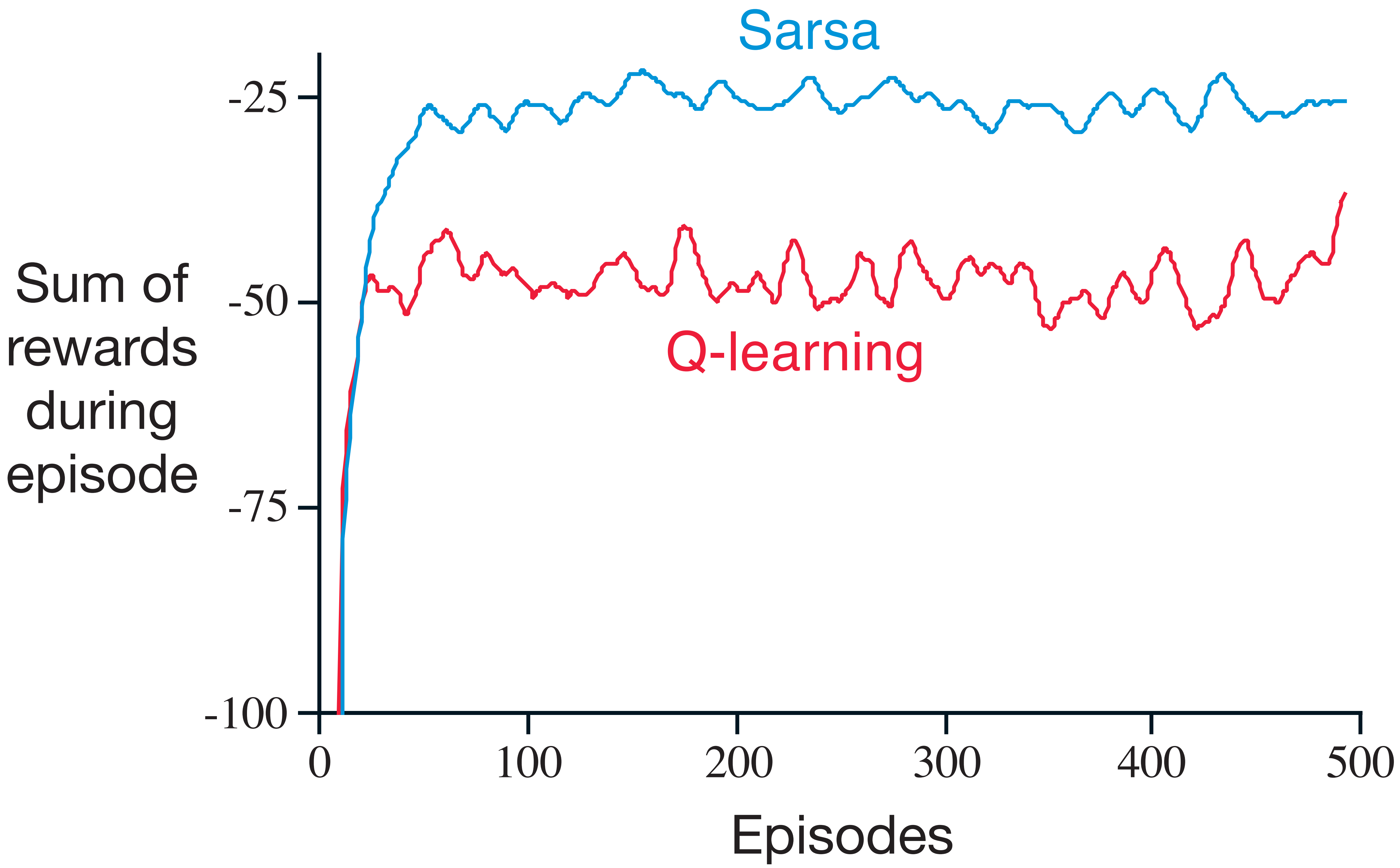
Question: Why aren't we estimating the **policy** π explicitly?

Example: The Cliff



- Agent gets -1 reward until they reach the goal state
- Step into the Cliff region, get reward -100 and go back to start
- **Question:** How will **Q-Learning** estimate the value of **this** state?
- **Question:** How will **Sarsa** estimate the value of **this** state?

Performance on The Cliff



Q-Learning estimates **optimal policy**, but Sarsa consistently **outperforms** Q-Learning. (**why?**)

Sarsa Uses Sampled Actions

- Sarsa updates the value of $Q(S_t, A_t)$ based on the **estimated value** of the next action that will **actually be taken** in the next state:

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

estimate of $v_\pi(S_{t+1}) = \mathbb{E}_\pi \left[Q(S_{t+1}, A_{t+1})\right]$

- *BUT:*
  - We know the **distribution** of A_{t+1} (**what is it?**)
  - The **estimated value** of that action **doesn't depend** on what happens after it is taken (**why?**)
  - Why not estimate $\mathbb{E}_\pi \left[Q(S_{t+1}, A_{t+1})\right]$ by taking **expectation** over A_{t+1}?

Expected Sarsa

Sarsa uses a single sample from $\pi(\cdot \mid S_t)$ to estimate $v_\pi(S_{t+1})$:

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

Expected Sarsa takes expectation over every possible action:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \mathbb{E}_{\mathbf{a} \sim \pi(\cdot|S_{t+1})} \left[Q(S_{t+1}, \mathbf{a})\right] - Q(S_t, A_t)\right]$$

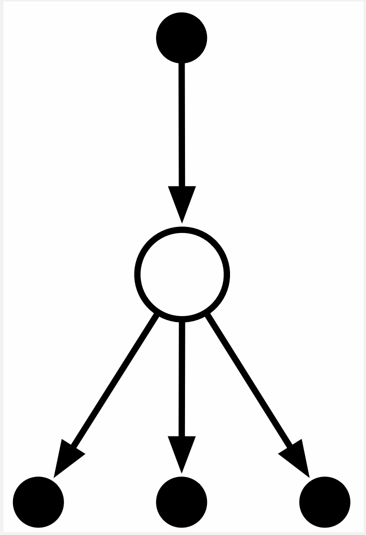
$$= Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \sum_{\mathbf{a} \in \mathcal{A}(S_{t+1})} \left[\pi(a \mid S_{t+1}) Q(S_{t+1}, \mathbf{a})\right] - Q(S_t, A_t)\right]$$

Expected Sarsa

Expected Sarsa (on-policy TD control) for estimating $\pi \approx \pi_*$

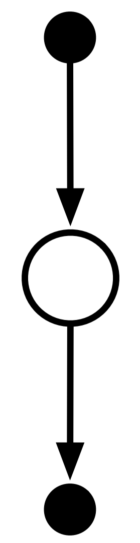
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        Choose A from S using policy derived from Q (e.g., ε-greedy)
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha\Big[R + \gamma\Big(\sum_a \pi(a \mid S')Q(S', a)\Big) - Q(S, A)\Big]$
        $S \leftarrow S'$
    until S is terminal

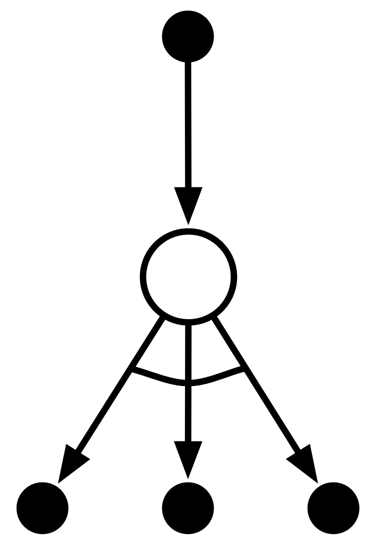


Information Usage

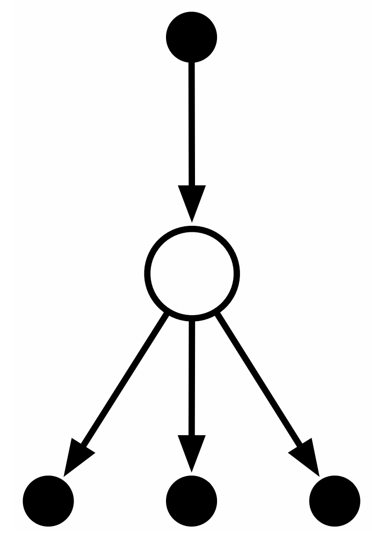
- **Sarsa** uses the actual reward R_t of the actual action A_t taken from an actual state S_t, and the estimated value of the **actual action** A_{t+1} to be taken in the actual next state S_{t+1}



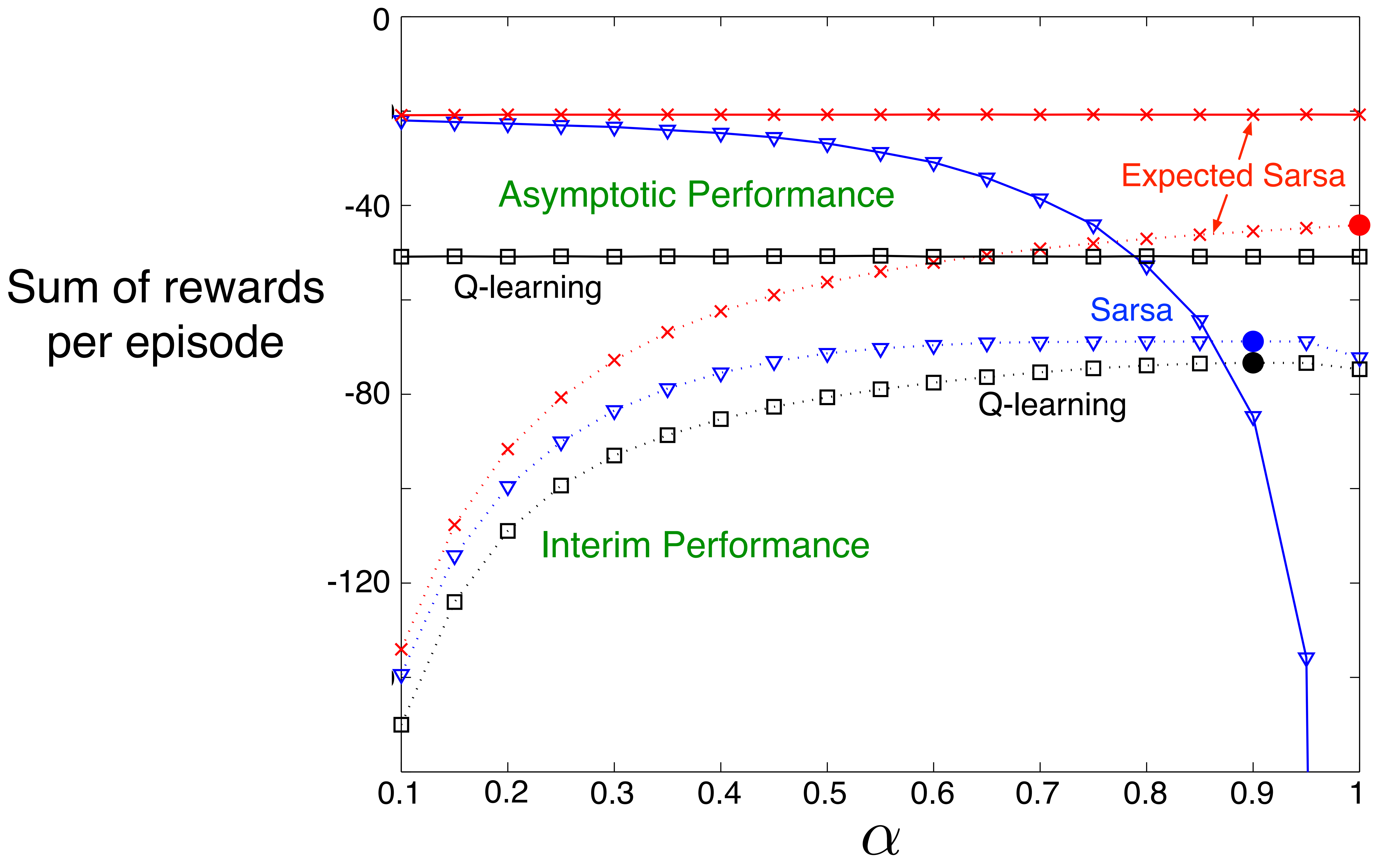
- **Q-Learning** uses the actual reward R_t of the actual action A_t taken from an actual state S_t, and the value of the **highest-estimated-value action** in the actual next state S_{t+1}



- **Expected Sarsa** uses the actual reward R_t of the actual action A_t taken from an actual state S_t, and the **expected estimated value** of next action A_{t+1} to be taken in the actual next state S_{t+1}



Performance on The Cliff, revisited



- For small enough α, Sarsa and Expected Sarsa have same **asymptotic** performance
- For larger α, Expected Sarsa has increasingly high **interim** performance, whereas Sarsa has increasingly poor interim performance (**why?**)

Summary

- Temporal Difference Learning **bootstraps** *and* learns from **experience**
    - Dynamic programming bootstraps, but doesn't learn from experience (requires full dynamics)
    - Monte Carlo learns from experience, but doesn't bootstrap
- Prediction: **TD(0) algorithm**
- **Sarsa** estimates action-values of **actual ϵ-greedy policy**
    - **Expected Sarsa** estimates action-values of ϵ-greedy policy
- **Q-Learning** estimates action-values of **optimal** policy while **executing** an **ϵ-greedy** policy