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 **December 4** at 11:59pm
  - Late submissions for 20% deduction
  - No remarking or EAs for empty submissions
- **SPOT** (formerly USRI) surveys are now available
  - Available until **December 10**

Previous Lecture Summary

- **Monte Carlo estimation** estimates values by averaging returns over **sample episodes**
  - Does not require access to full model of **dynamics**
  - Does require access to an entire **episode** for each sample
- 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**
  - **Importance sampling** is one way to perform off-policy learning
  - **Weighted** importance sampling has lower **variance** than **ordinary** importance sampling
- **Off-policy control** is learning the **optimal policy** (target policy) using episodes from a **behaviour policy**

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

	Bootstrapping	No bootstrapping
Learns from experience	TD	MC
Requires full dynamics	DP	

A 2x2 table. "DP" is in the "requires full dynamics" row and "bootstrapping" column; "MC" is in the "Learns from experience" row and "no bootstrapping" column; "TD" (in a star) is in the "learns from experience" row and "bootstrapping" column.

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

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

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

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

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

$$= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s]$$

$$= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s].$$

Dynamic programming:
Approximate because v_π not known

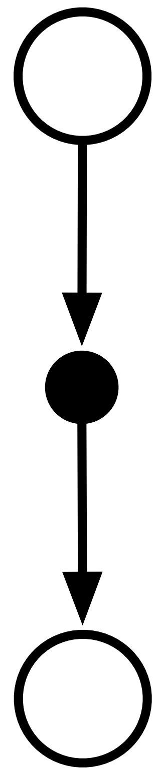
TD(0): Approximate because of $\mathbb{E}$ ***and*** v_π not known

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



An open circle has an arrow to a closed circle, which has an arrow to an open circle.

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

TD for Control

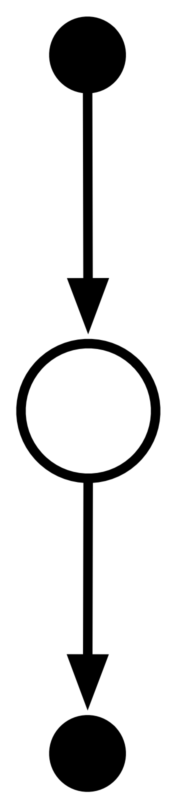
- We can plug TD prediction into the **generalized policy iteration** framework
- **Monte Carlo control loop:**
  1. Generate an **episode** using estimated π
  2. Update estimates of Q and π
- **On-policy TD control loop:**
  1. Take an **action** according to π
  2. Update estimates of Q and π

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 $Q(terminal, \cdot) = 0$

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



A closed circle has an arrow to an open circle, which has an arrow to a closed circle.

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

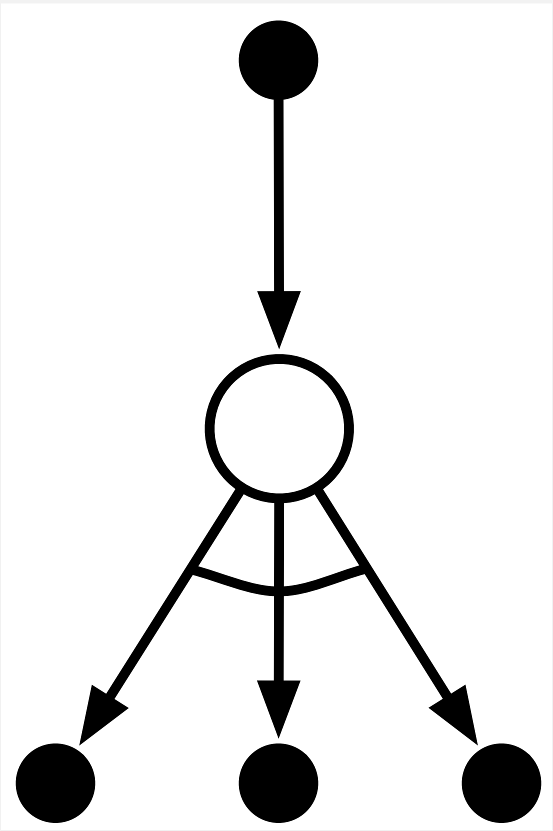
- Just as with on-policy Monte Carlo control, Sarsa does not converge to the **optimal** policy, because it always chooses an **ϵ-greedy action**
  - And the estimated Q-values are with respect to the **actual actions**, which are ϵ-greedy
- **Question:** Why is it necessary to choose ϵ-greedy actions?
- 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 $Q(\mathit{terminal}, \cdot) = 0$

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



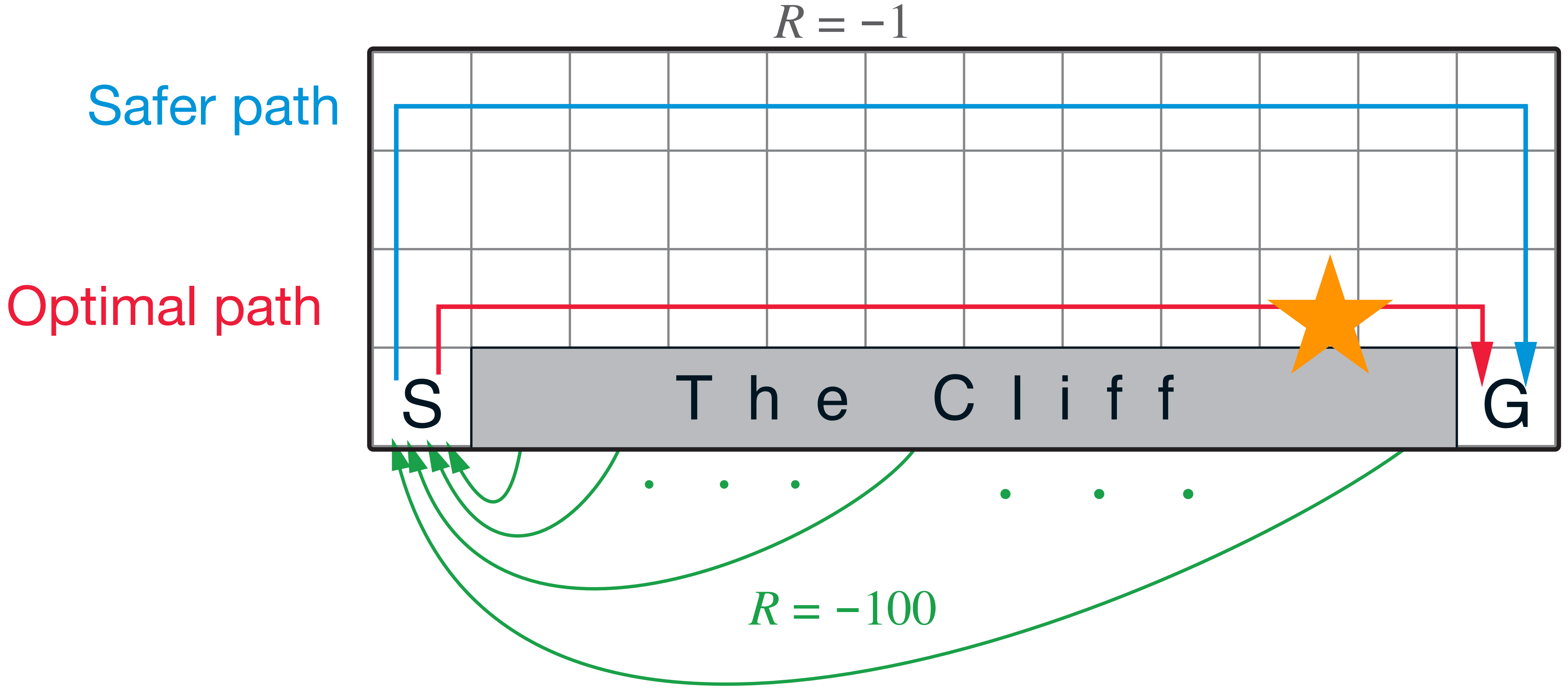
A closed circle has an arrow to an open circle, which has multiple arrows to closed circles; the multiple arrows are connected by an arc.

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

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

Example: The Cliff

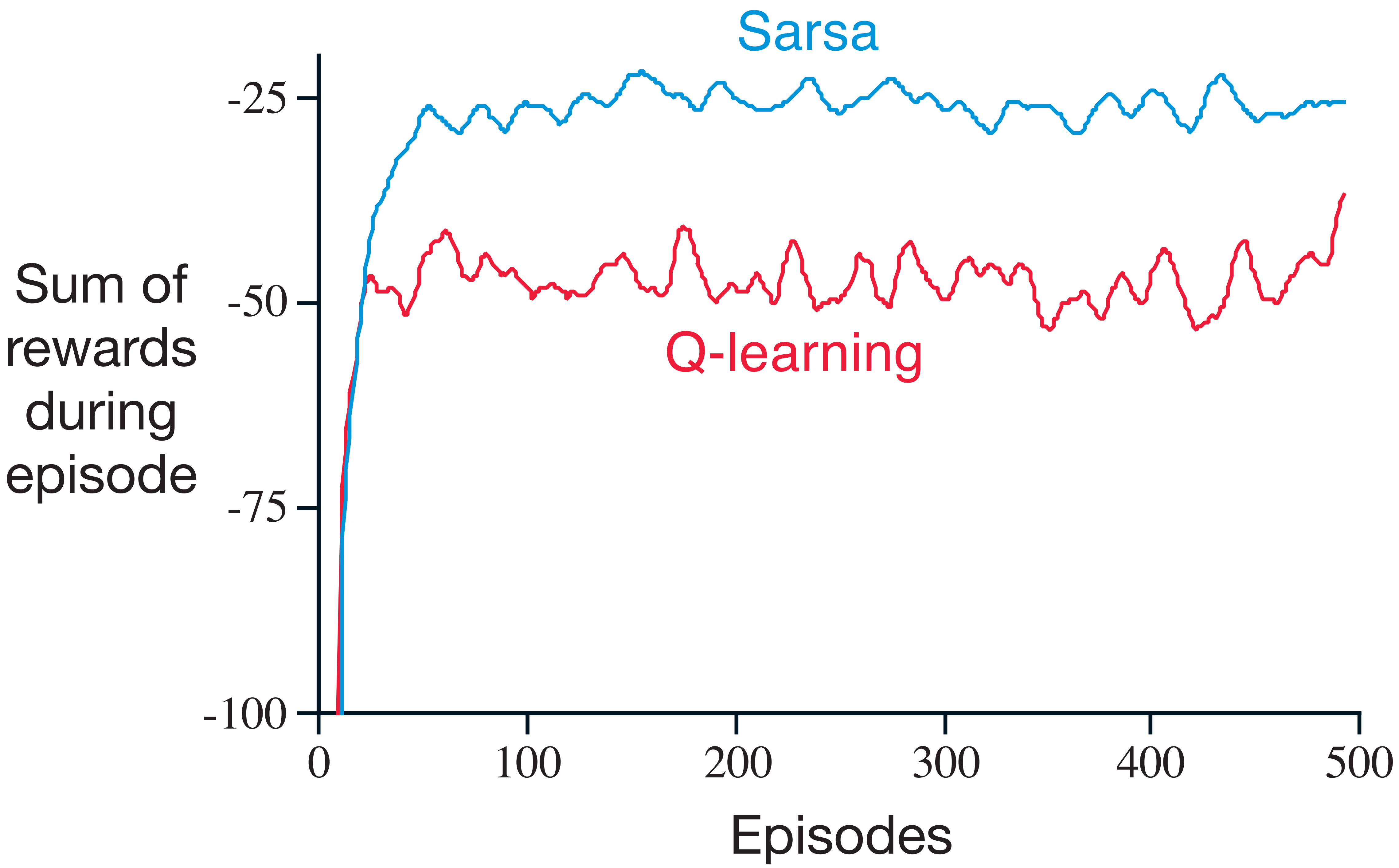
γ=1 (undiscounted)



A gridworld; the bottom-left cell is labelled S and the bottom right cell is labelled G. The rest of the bottom row is a grey and labelled "The Cliff". A path labelled "optimal path" goes up one from S, right immediately beside the cliff, and then down one to G. Another path labelled "safer path" goes all the way to the top of the grid from S, all the way across to the right boundary, and then back down again to G. Each cell of The Cliff has an arrow labelled "R=-100" leading back to S. There is a star on the state just up from the cliff and two steps left of the right boundary.

- 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



The y-axis is labelled "Sum of rewards during episode" and the x-axis is labelled "Episodes". The Sarsa trace climbs quickly to about -25 and then appears to converge. The Q-learning trace climbs at the same rate, but instead converges at around -50.

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 $v_\pi(S_{t+1})$:

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

Expected Sarsa takes **expectation** over every possible action:

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

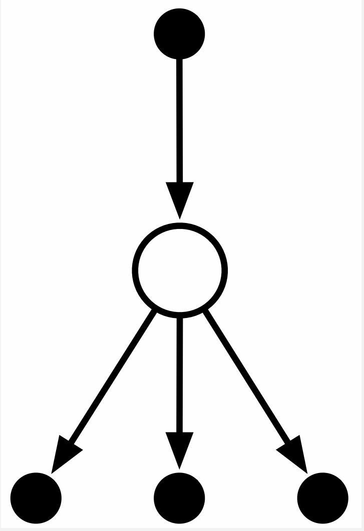
$$= Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \sum_{\mathbf{a} \in \mathscr{A}(S_{t+1})} \left[\pi(a \mid S_{t+1}) Q(S_{t+1}, \mathbf{a})\right] - Q(S_t, A_t)\right]$$

Expected Sarsa

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

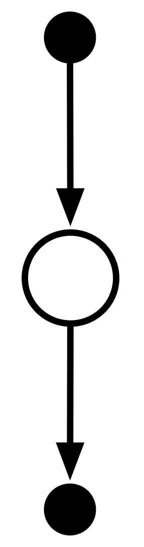
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

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'
        $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma\left(\sum_a \pi(a \mid S')Q(S', a)\right) - Q(S, A)\right]$
        $S \leftarrow S'$
    until S is terminal

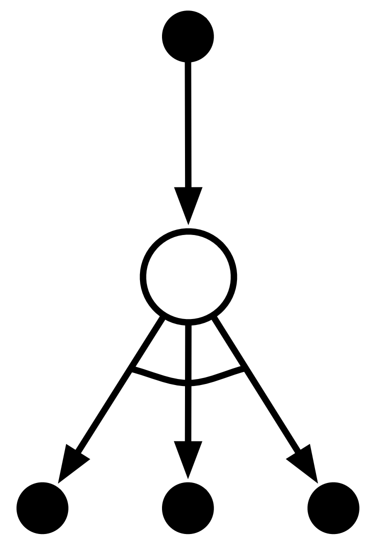


Information Usage

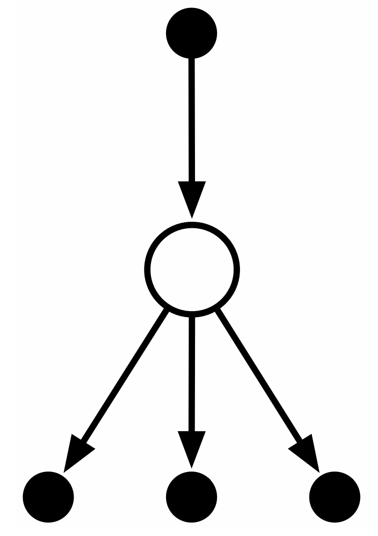
- **Sarsa** uses the actual reward R_t of the actual action A_t taken from an actual state S_t, and the estimated value of the **actual action** A_{t+1} to be taken in the actual next state S_{t+1}



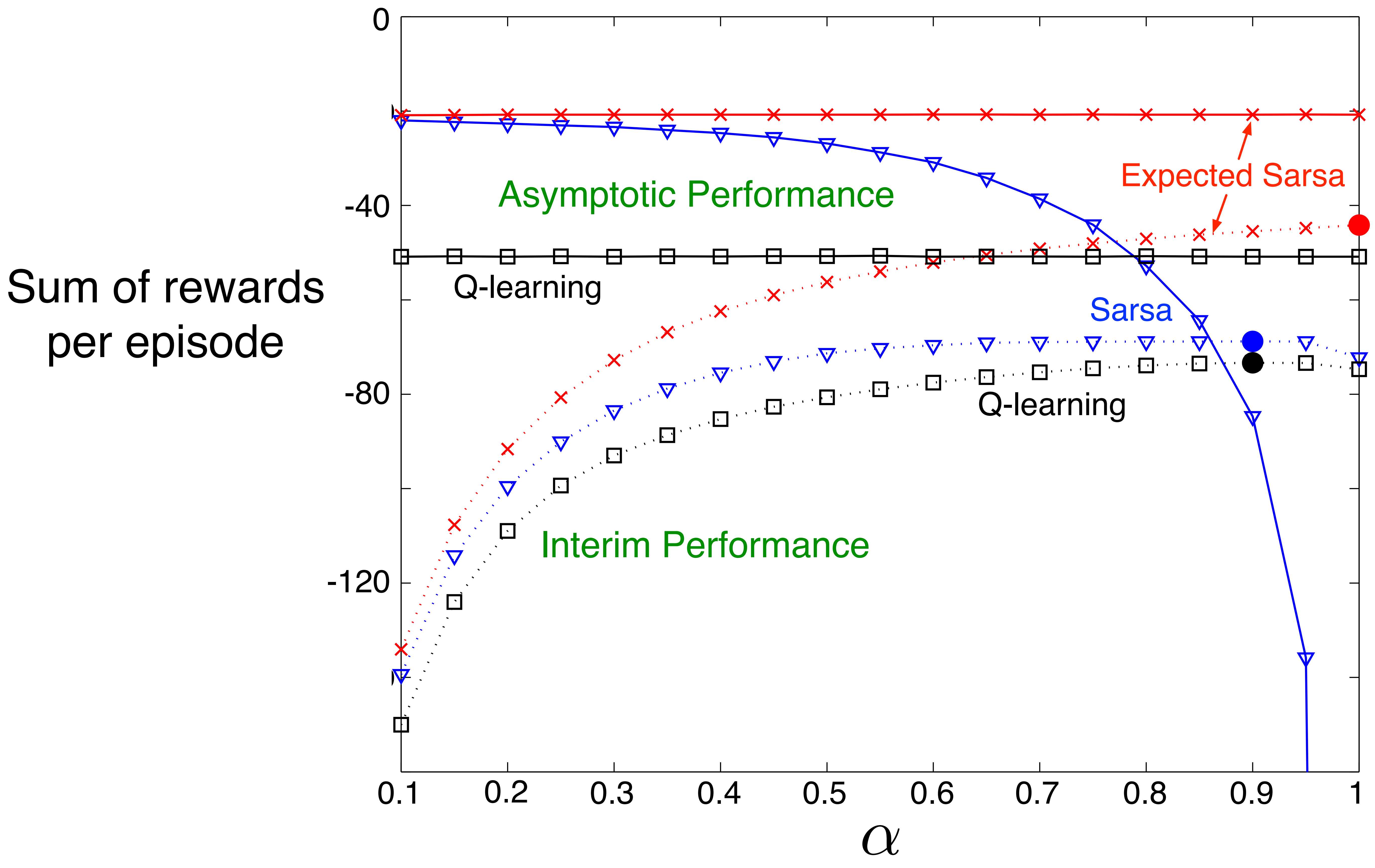
- **Q-Learning** uses the actual reward R_t of the actual action A_t taken from an actual state S_t, and the value of the **highest-estimated-value action** in the actual next state S_{t+1}



- **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



The y-axis is labelled "sum of rewards per episode" and the x-axis is labelled "alpha". Performance on the cliff environment is plotted for Q-learning, Sarsa, and Expected Sarsa for both asymptomatic performance and interim performance.

- 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