Lecture 7

Temporal Difference Learning and Stochastic Approximation

Goals of this Lecture

- Understand the motivation and formulation of Temporal-Difference (TD) methods.
- Learn the TD(0) algorithm for policy evaluation and compare it with Monte Carlo.
- Introduce stochastic approximation and the role of contraction mappings in convergence.
- Generalize TD(0) to n-step and TD(λ) using eligibility traces.
- Learn TD-based control algorithms: SARSA (on-policy) and Q-learning (off-policy).
- Analyze empirical performance and practical challenges in using TD methods.

7.1 From Monte Carlo to Temporal-Difference

Motivation. Monte Carlo (MC) methods estimate value functions by averaging complete returns from sampled episodes:

$$\hat{v}_\pi(s) = \frac{1}{N(s)} \sum_{i=1}^{N(s)} G_t^{(i)}.$$

These methods are model-free and easy to implement, but have two major limitations:

- They require episodes to terminate before updating any estimates, making them suitable only for *episodic* tasks.
- Since they use full returns G_t, they often suffer from high variance.

While it is possible to compute the empirical average incrementally without storing all returns—e.g., using:

$$\hat{v}_\pi(s) \leftarrow \hat{v}_\pi(s) + \alpha \left(G_t - \hat{v}_\pi(s)\right),$$

this update still depends on the full return G_t, which can only be calculated after an episode ends. Thus, despite being memory-efficient, Monte Carlo methods remain delayed in practice.

Idea of TD. Temporal-Difference (TD) methods address these issues by *bootstrapping*: they update estimates based on other learned estimates, combining ideas from both Monte Carlo and Dynamic Programming:

- **Like MC:** TD methods learn directly from raw experience (samples), without requiring a model of the environment.
- **Like DP:** TD methods update value estimates using other current estimates, rather than waiting for full returns.

This enables online, incremental updates that are more efficient and broadly applicable. TD methods can operate in continuing tasks and make updates at every time step—without needing to wait for episodes to finish. Moreover, by bootstrapping, they tend to reduce variance compared to Monte Carlo estimation.

7.2 TD(0) for Policy Evaluation

Definition. The simplest TD method is TD(0), which updates the estimate of $V(S_t)$ using the one-step return:

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

This is a stochastic approximation of the Bellman equation:

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

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

Figure 7.1: TD(0) algorithm: tabular policy evaluation with step size α.

Comparison with Monte Carlo. Monte Carlo and TD(0) both aim to estimate v^π from experience, but differ in their update targets and behavior:

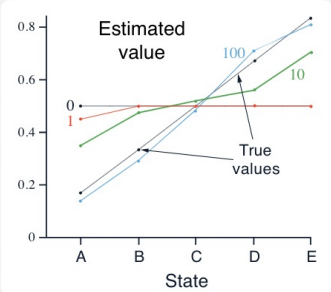
- **Monte Carlo (MC)** uses complete returns $G_t = R_{t+1} + \gamma R_{t+2} + \dots$ and updates only after episode termination. This provides unbiased estimates but with high variance.
- **TD(0)** uses the one-step target $R_{t+1} + \gamma V(S_{t+1})$, enabling online and incremental updates. The estimates are biased (due to bootstrapping) but generally exhibit lower variance.

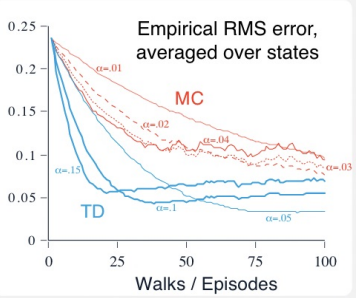
TD(0) typically learns faster than MC, especially in long or infinite-horizon tasks, due to earlier and more frequent updates. The following figure illustrates this difference empirically:

Example 6.2 Random Walk

In this example we empirically compare the prediction abilities of TD(0) and constant-α MC when applied to the following Markov reward process:

A *Markov reward process*, or MRP, is a Markov decision process without actions. We will often use MRPs when focusing on the prediction problem, in which there is no need to distinguish the dynamics due to the environment from those due to the agent. In this MRP, all episodes start in the center state, C, then proceed either left or right by one state on each step, with equal probability. Episodes terminate either on the extreme left or the extreme right. When an episode terminates on the right, a reward of +1 occurs; all other rewards are zero. For example, a typical episode might consist of the following state-and-reward sequence: C, 0, B, 0, C, 0, D, 0, E, 1. Because this task is undiscounted, the true value of each state is the probability of terminating on the right if starting from that state. Thus, the true value of the center state is $v_\pi(\mathsf{C}) = 0.5$. The true values of all the states, A through E, are $\frac{1}{6}, \frac{2}{6}, \frac{3}{6}, \frac{4}{6}$, and $\frac{5}{6}$.





The left graph above shows the values learned after various numbers of episodes on a single run of TD(0). The estimates after 100 episodes are about as close as they ever come to the true values—with a constant step-size parameter ($\alpha = 0.1$ in this example), the values fluctuate indefinitely in response to the outcomes of the most recent episodes. The right graph shows learning curves for the two methods for various values of α. The performance measure shown is the root mean-squared (RMS) error between the value function learned and the true value function, averaged over the five states, then averaged over 100 runs. In all cases the approximate value function was initialized to the intermediate value $V(s) = 0.5$, for all s. The TD method was consistently better than the MC method on this task.

Figure 7.2: Comparison of Monte Carlo and TD(0) learning. TD(0) converges faster by bootstrapping.

General Update View. TD methods can be expressed in the general form of stochastic approximation:

$$\hat{v}_{t+1}(S_t) = \hat{v}_t(S_t) + \alpha_t \left[U_t - \hat{v}_t(S_t)\right],$$

where U_t is a target that serves as an estimate of the expected return. The goal is that $\mathbb{E}[U_t \mid S_t = s] \approx v^\pi(s)$. In the case of TD(0), the one-step bootstrapped target is used:

$$U_t := R_{t+1} + \gamma \hat{v}_t(S_{t+1}).$$

While this introduces bias—since U_t depends on the current approximation $\hat{v}_t$—it enables faster and fully online updates at each step, in contrast to the episodic nature of Monte Carlo methods.

7.2.1 Convergence of TD via the ODE Method

From TD(0) to General Stochastic Approximation. In the previous section, we introduced TD(0) as a stochastic approximation method that uses the update:

$$v_{t+1}(S_t) = v_t(S_t) + \alpha_t \left(R_{t+1} + \gamma v_t(S_{t+1}) - v_t(S_t)\right).$$

This update can be expressed in a more general stochastic approximation form as:

$$x_{t+1} = x_t + \alpha_t \left(F(x_t) + \varepsilon_t\right),$$

where, for TD(0), we set $x_t = v_t$ and $F(v) = T_\pi v - v$, and the noise ε_t is given by:

$$\varepsilon_t = [R_{t+1} + \gamma v_t(S_{t+1})] - (T_\pi v_t)(S_t),$$

with $\mathbb{E}[\varepsilon_t \mid \mathcal{F}_t] = 0$.

To analyze convergence rigorously, we introduce a general stochastic approximation convergence theorem that leverages the *ODE method*.

Theorem 7.1 (Stochastic Approximation via ODE Stability)**.** *Consider the stochastic recursion in $\mathbb{R}^n$:*

$$x_{t+1} = x_t + \alpha_t \left(F(x_t) + \varepsilon_t\right),$$

where the following conditions hold:

1. *The function $F : \mathbb{R}^n \to \mathbb{R}^n$ is Lipschitz continuous.*
2. *The step sizes satisfy the Robbins–Monro conditions:*

$$\sum_{t=0}^{\infty} \alpha_t = \infty, \quad \textit{and} \quad \sum_{t=0}^{\infty} \alpha_t^2 < \infty.$$

3. *The noise ε_t is a martingale difference sequence w.r.t. $\mathcal{F}_t = \sigma(x_0, \ldots, x_t, \varepsilon_0, \ldots, \varepsilon_{t-1})$ satisfying:*

$$\mathbb{E}[\varepsilon_t \mid \mathcal{F}_t] = 0, \quad \sup_t \mathbb{E}[\|\varepsilon_t\|^2] < \infty.$$

4. *The ODE associated with the recursion:*

$$\dot{x}(t) = F(x(t)),$$

has a globally asymptotically stable equilibrium x^*.

Then, the sequence (x_t) converges almost surely to x^*:

$$x_t \xrightarrow{a.s.} x^*, \quad \textit{as} \quad t \to \infty.$$

One-Sided Lipschitz Condition. A sufficient condition ensuring global asymptotic stability of the ODE $\dot{x}(t) = F(x(t))$ is that F satisfies a one-sided Lipschitz condition with constant $\lambda > 0$:

$$\langle F(x) - F(y), x - y \rangle \leq -\lambda \|x - y\|^2, \quad \forall x, y \in \mathbb{R}^n.$$

This condition implies exponential convergence of solutions of the ODE to the equilibrium x^*.

Corollary 7.1 (Convergence of TD(0))**.** *Consider the TD(0) update for a fixed policy π, where:*

$$v_{t+1}(s) = v_t(s) + \alpha_t \left(R_{t+1} + \gamma v_t(S_{t+1}) - v_t(s)\right),$$

and the step sizes α_t satisfy Robbins–Monro conditions. If the Markov chain induced by π visits all states infinitely often, then:

$$v_t \xrightarrow{a.s.} v^\pi, \quad as \quad t \to \infty,$$

where v^π is the unique fixed point of the Bellman operator T_π.

Proof. Observe that $F(v) = T_\pi v - v$. The Bellman operator T_π is a γ-contraction under the max norm, which implies the one-sided Lipschitz condition:

$$\langle (T_\pi v - v) - (T_\pi w - w), v - w \rangle \leq -(1-\gamma)\|v - w\|^2,$$

with $\lambda = 1 - \gamma > 0$. Therefore, the associated ODE:

$$\dot{v}(t) = T_\pi v(t) - v(t)$$

has a globally asymptotically stable equilibrium v^π. Theorem 7.1 then implies that $v_t \to v^\pi$ almost surely. □

Relevant Literature. These results derive from classical stochastic approximation literature, notably works by Robbins and Monro (1951), Kushner and Yin (2003), and Borkar (2008). The ODE method provides an intuitive and rigorous framework connecting stability properties of deterministic differential equations with stochastic iterative methods, widely used in reinforcement learning algorithms such as TD(0), SARSA, and Q-learning.

7.3 Multi-Step TD and TD(λ)

Motivation. TD(0) uses only the immediate reward and the value of the next state to update the estimate of the current state. Monte Carlo (MC) methods, on the other hand, wait until the end of the episode and use the full return. Both approaches have advantages and drawbacks:

- TD(0) is low variance and updates incrementally, but is more biased.
- MC is unbiased but has high variance and applies only to episodic tasks.

This motivates a family of *multi-step TD methods* that interpolate between TD(0) and MC by using returns spanning multiple steps.

The n-Step TD Update. Given a trajectory $(S_t, R_{t+1}, S_{t+1}, \ldots, S_{t+n})$, the n-step return is defined as:

$$G_t^{(n)} := \sum_{k=0}^{n-1} \gamma^k R_{t+k+1} + \gamma^n V(S_{t+n}).$$

The n-step TD algorithm then performs the update:

$$V(S_t) \leftarrow V(S_t) + \alpha \left[G_t^{(n)} - V(S_t)\right].$$

- When $n = 1$, this recovers TD(0).
- When $n \to \infty$, $G_t^{(n)} \to G_t$, the Monte Carlo return.

This formulation allows a continuous tradeoff between bias and variance:

- Smaller n reduces variance due to bootstrapping, but increases bias.
- Larger n reduces bias but increases variance, particularly in stochastic environments.

Discussion of Figure 7.3. The figure below compares the performance of n-step TD methods as a function of the learning rate α, across various values of n, on a 19-state random walk task. We observe:

- TD(0) ($n = 1$) is robust to larger step sizes, but converges to higher error.
- Monte Carlo-like updates (large n) perform poorly unless α is very small.
- Intermediate values like $n = 4$ or $n = 8$ strike the best balance between speed and accuracy.

This empirical result highlights the importance of tuning n to balance stability and learning efficiency.

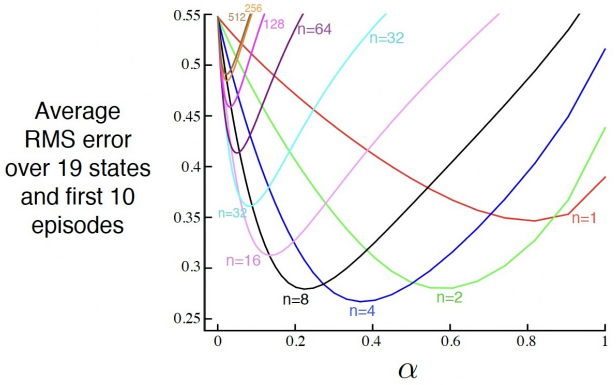


Figure 7.3: Empirical performance of n-step TD methods as a function of α, for various values of n, on a 19-state random walk task.

TD(λ) and Eligibility Traces. TD(λ) combines all n-step returns using an exponentially weighted average with decay factor $\lambda \in [0, 1]$:

$$G_t^\lambda := (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}.$$

This defines the **forward view** of TD(λ), but it is not implementable online since it requires future rewards.

Offline λ-Return Implementation. In the *offline* version of TD(λ), we compute the λ-return G_t^λ for each state visited during an episode *after the episode completes*, and use it to perform a one-time update:

$$V(S_t) \leftarrow V(S_t) + \alpha \left[G_t^\lambda - V(S_t) \right].$$

This method closely matches the forward view of TD(λ) but delays all updates until the episode ends. While accurate, it is less suitable for online learning or real-time applications. This motivates the use of the *backward view*, which provides an efficient and incremental approximation of the same objective.

Eligibility Traces (Backward View). We maintain an auxiliary variable $e_t(s)$ called the *eligibility trace* for each state s:

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) + 1, & \text{if } s = S_t, \\ \gamma\lambda e_{t-1}(s), & \text{otherwise.} \end{cases}$$

Then the value function is updated using:

$$V(s) \leftarrow V(s) + \alpha \cdot \delta_t \cdot e_t(s),$$

where the TD error is:

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t).$$

This update assigns credit to all previously visited states, decaying geometrically according to how long ago they were visited.

Special Cases.

- $\lambda = 0$: Recovers TD(0), only the current state is updated.
- $\lambda \to 1$: Approaches Monte Carlo learning with full episode returns.

Interpretation. TD(λ) unifies temporal-difference and Monte Carlo learning into a single framework that can interpolate between them. It achieves both faster convergence than MC and more stability than TD(0), especially in long or partially observable tasks.

Discussion of Figure 7.4. The figure compares the performance of TD(λ), which implements the backward view with eligibility traces, to the offline λ-return algorithm, which corresponds to the forward view. Both approaches aim to combine multi-step returns to improve learning efficiency, interpolating between TD(0) and Monte Carlo.

The plot shows root mean squared error (RMSE) after 10 episodes on a 19-state random walk task, for varying values of λ and step-size α. We observe that:

- For small α, the performance of both methods is nearly identical.
- For large α, TD(λ) becomes more sensitive, especially at high λ values, potentially due to trace accumulation errors.
- Intermediate λ values (e.g., $\lambda = 0.8$) tend to yield the lowest RMSE across a wide range of α values.

This illustrates that TD(λ) can closely approximate the forward view in practice while being more efficient and fully online, making it well suited for large-scale or continual learning problems.

7.4 TD Control: SARSA and Q-Learning

SARSA (On-Policy TD Control). SARSA is an on-policy TD control method that updates action-value estimates using the actual action taken by the agent. The update rule is:

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

The name "SARSA" comes from the quintuple $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$ used in the update. SARSA improves the policy by continually acting ε-greedily with respect to Q and updating using the same ε-greedy behavior. Thus, it converges to a policy that balances exploration and exploitation.

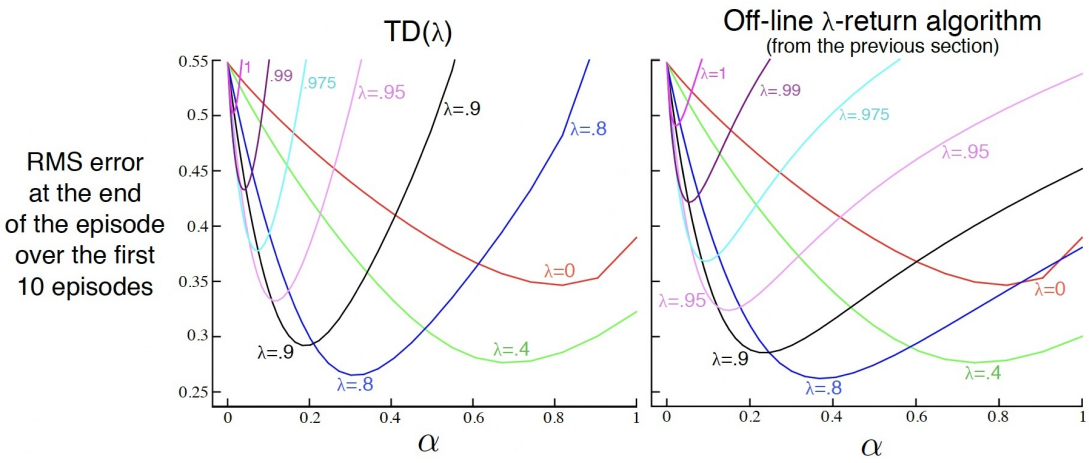


Figure 7.4: Comparison of TD(λ) and offline λ-return algorithm. The backward view (online TD with traces) efficiently approximates the forward view using only incremental updates.

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

Figure 7.5: SARSA algorithm: incremental on-policy TD control using observed actions and transitions.

Q-learning (Off-Policy TD Control). Q-learning is an off-policy TD control algorithm. It learns the optimal action-value function q^*, regardless of the behavior policy used to generate data, by using the greedy action in its update:

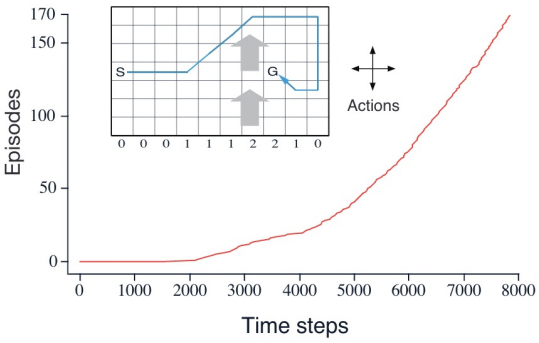
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right].$$

Unlike SARSA, which updates based on the next action actually taken, Q-learning uses the maximum over all possible next actions. This decouples learning from behavior, enabling convergence to the optimal policy even when following an exploratory behavior.

Empirical Comparisons and Takeaways

Example: Cliff Walking. In this domain, the agent must navigate near a "cliff" region that incurs a high negative reward when entered. SARSA, being on-policy, learns a conservative strategy that avoids risky paths during exploration. Q-learning, being off-policy and greedy in its update, may learn a shorter but riskier path.

Example 6.5: Windy Gridworld Shown inset below is a standard gridworld, with start and goal states, but with one difference: there is a crosswind running upward through the middle of the grid. The actions are the standard four—`up, down, right`, and `left`—but in the middle region the resultant next states are shifted upward by a "wind," the strength of which varies from column to column. The strength of the wind is given below each column, in number of cells shifted upward. For example, if you are one cell to the right of the goal, then the action `left` takes you to the cell just above the goal. This is an undiscounted episodic task, with constant rewards of -1 until the goal state is reached.



The graph to the right shows the results of applying ε-greedy Sarsa to this task, with $\varepsilon = 0.1$, $\alpha = 0.5$, and the initial values $Q(s,a) = 0$ for all s, a. The increasing slope of the graph shows that the goal was reached more quickly over time. By 8000 time steps, the greedy policy was long since optimal (a trajectory from it is shown inset); continued ε-greedy exploration kept the average episode length at about 17 steps, two more than the minimum of 15. Note that Monte Carlo methods cannot easily be used here because termination is not guaranteed for all policies. If a policy was ever found that caused the agent to stay in the same state, then the next episode would never end. Online learning methods such as Sarsa do not have this problem because they quickly learn *during the episode* that such policies are poor, and switch to something else. ■

Figure 7.6: SARSA applied to the Windy Gridworld. Despite exploration, it learns a path that avoids risky states.

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 $Q(\mathit{terminal}, \cdot) = 0$

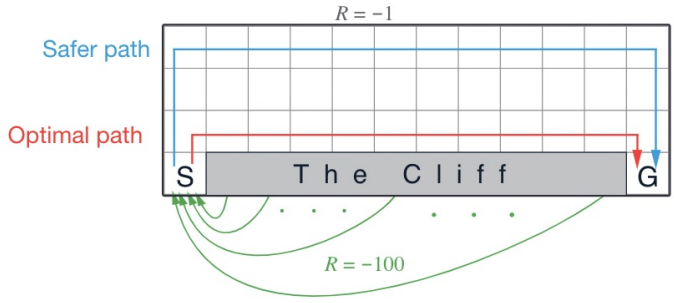
Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    $S \leftarrow S'$
  until S is terminal

Figure 7.7: Q-Learning algorithm: off-policy control using greedy next-action updates.

Summary.

- **SARSA:** On-policy, safer, but converges to a policy that reflects exploratory behavior.
- **Q-learning:** Off-policy, learns optimal greedy policy, but may be riskier during training.
- The choice between them depends on whether safety during learning or optimality at convergence is more important.

Example 6.6: Cliff Walking This gridworld example compares Sarsa and Q-learning, highlighting the difference between on-policy (Sarsa) and off-policy (Q-learning) methods. Consider the gridworld shown to the right. This is a standard undiscounted, episodic task, with start and goal states, and the usual actions causing movement up, down, right, and left. Reward is -1 on all transitions except those into the region marked "The Cliff." Stepping into this region incurs a reward of -100 and sends the agent instantly back to the start.



The graph to the right shows the performance of the Sarsa and Q-learning methods with ε-greedy action selection, $\varepsilon = 0.1$. After an initial transient, Q-learning learns values for the optimal policy, that which travels right along the edge of the cliff. Unfortunately, this results in its occasionally falling off the cliff because of the ε-greedy action selection. Sarsa, on the other hand, takes the action selection into account and learns the longer but safer path through the upper part of the grid. Although Q-learning actually learns the values of the optimal policy, its online performance is worse than that of Sarsa, which learns the roundabout policy. Of course, if ε were gradually reduced, then both methods would asymptotically converge to the optimal policy. ■

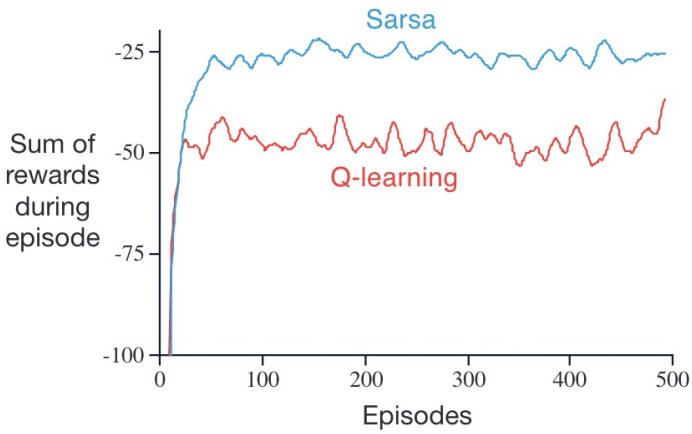


Figure 7.8: Comparison of SARSA and Q-Learning on the Cliff Walking task. SARSA prefers safer trajectories, while Q-learning may risk falling off the cliff during training.