SMU Lyle School of Engineering

Reinforcement Learning

Temporal-Difference Learning

Sutton/Barto* Chapter 6

Michael Hahsler, SMU

With figures from Sutton/Barto*

*Sutton and Barto, Reinforcement Learning: An Introduction, 2nd edition, MIT Press, Cambridge, MA, 2018

Topics of this Course

- Introduction to reinforcement learning
- Markov decision processes
- **Part I: Tabular Methods**
  - Dynamic programming
  - Monte Carlo methods
  - **Temporal-difference learning**
  - Multi-step bootstrapping
  - Planning and learning with tabular methods
- Part II: Approximate Solution Methods
  - Prediction and Control using Approximation
  - Eligibility Traces
  - Policy Gradient Methods
- Part III: Modern RL Methods
  - Deep Reinforcement Learning
  - Current Applications

Summary of Notation

General

X	capital letters: random variables
x, p	lower-case letters: realizations of random variables or scalar functions
$\boldsymbol{w}$	Bold lower-case letters: real-valued vectors (even if random variables)
$\boldsymbol{W}$	bold capitals: matrices
α	Greek letters: parameters (vectors if in bolt)
$\Pr\{X = x\}$	probability that a random variable X takes on the value x
$X \sim p$	random variable X selected from distribution $p(x) = \Pr\{X = x\}$
$\mathbb{E}[X]$	expectation of a random variable X, i.e., $\mathbb{E}[X] = \sum_x p(x)x$
$\text{argmax}_\text{a}\, f(a)$	a value of action a at which $f(a)$ takes its maximal value

Value Function

G_t	return (cumulative reward) following time t
$G_{t:h}$	return from t to h (discounted and corrected)
$v_\pi(s)$	value of state s under policy π (expected return)
$v_*(s)$	value of state s under the optimal policy
$q_\pi(s, a)$	value of taking action a in state s under policy π
$q_*(s, a)$	value of taking action a in state s under the optimal policy
V, V_t	array estimates of state-value function v_π or v_*
Q, Q_t	array estimates of action-value function q_π or v_*

MDP

s, s'	states
a	an action
r	a reward
$\mathcal{S}$	set of all (nonterminal) states, $\mathcal{S}^+$ are all states
$\mathcal{A}(s)$	set of all actions available in state s
γ	discount-rate parameter
t	discrete time step
T	final time step of an episode (a.k.a. horizon)
A_t	random variable for the action at time t
S_t	random variable for the state at time t
R_t	random variable for the reward at time t
$p(s', r \mid s, a)$	probability of transition to state s' and receiving reward r, from state s taking action a.
$p(s' \mid s, a)$	probability of transition to state s' fom state s taking action a.
$r(s, a)$	expected immediate reward from state s after action a.
$r(s, a, s')$	expected immediate reward from state s to s' with action a.
$\pi(a \mid s)$	probability of taking action a in state s under stochastic policy π
$\pi(s)$	action taken in state s under deterministic policy π

Temporal Difference Learning

U_t	target for estimate at time t
δ_t	TD error

Recap: MDPs, DP and MC

- **Finite MDPs**: **Bellman Expectation Equations** convert expected return maximization into an easier value function consistency problem.
- **Dynamic Programming (DP)**: **Policy evaluation** solves the Bellman Expectation Equations by updating the value function iteratively (planning).
- **Generalized Policy Iteration (GPI)**: Iterates between policy evaluation and **greedy policy improvement** to find the optimal policy.
- **Monte Carlo Methods (MC)**: Uses **sample-averaging for policy evaluation** in GPI (update after complete sample episode).

Reward hypothesis: "All goals and purposes of an agent can be represented as the maximization of the expected cumulative reward."

Expected Bellman update operator

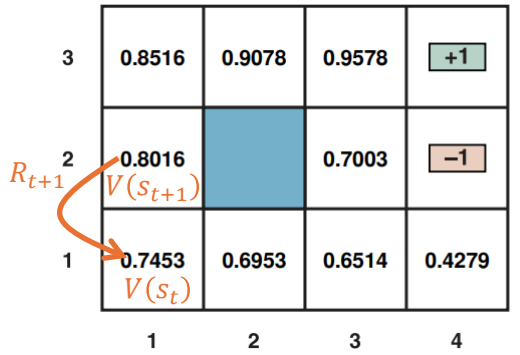
Policy Improvement Theorem: Greedy action choice leads to policy improvement and convergence.

Greedy in the Limit with Infinite Exploration (GLIE): Guarantees convergence.

Temporal-Differencing provides another mechanism for **policy evaluation.**

Temporal-Difference (TD) Learning

$\gamma = 1; r = -0.04$



+ stochastic transition model

- We will first focus on:
  - On-step
  - Tabular
  - Model-free TD

- We know from the Bellman equations, that the value function has a recursive relationship between state values and the value of its successors.

$$v_\pi(s_t) \stackrel{\text{def}}{=} \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s_t]$$

- The error can be **sampled** using the value of the state at time t, the observed immediate reward and next state at time $t+1$. The **TD error** is defined between two time steps as:

$$\delta_t = \underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\text{We call this the target } U_t} - V(S_t)$$

Note: The TD error is only a sample from the error distribution because we ignore the expectation!

- **Idea**: Reduce the TD error by making the tabular state value estimate $V(S_t)$ more similar to the TD target:

$$V(S_t) \leftsquigarrow U_t = R_{t+1} + \gamma V(S_{t+1})$$

- Combines ideas from MC and DP
  - From MC: learn from **raw experience** (observed $S_t, A_t, R_{t+1}, S_{t+1}$ tuples)
  - From DP: calculates the error between two value estimates without waiting for the end of the episode via **bootstrapping**.

A truly online process that can update after each action

TD Prediction

Estimate the value function for a policy by minimizing the TD error.

TD Prediction

- Follow π and update tabular estimate V of v_π

TD Error for one step

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

Uses reward + estimate for next state = target U_t

TD uses local search with α as a learning parameter to reduce the TD error.

- Updating an estimate using another estimate is called **"bootstrapping"** and introduces an estimation bias.

- Similartiy to MC: $V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$

MC Error over complete episode

In MC methods, $\alpha = \frac{W}{C}$ represents the weight for incremental updates.
No Bootstrapping!
Unbiased estimate!

Alg. Tabular TD(0)

We will later learn what 0 means

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 $V(terminal) = 0$

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'
    $V(S) \leftarrow V(S) + \alpha\left[R + \gamma V(S') - V(S)\right]$
    $S \leftarrow S'$
  until S is terminal

Note: We can only update for time t at time $t+1$ since we need to observe a complete S, A, R, S' interaction and S' is observed at $t+1$.

Convergence of TD Methods

- Regular conditions on the MDP apply (finite, $\gamma < 1$, etc.)

- The observed TD error $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is a noisy sample of the true difference.

  Stochastic approximation theory guarantees convergence if the **learning rate α decreases slowly enough.** Then the updates

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

  lead to an expected TD error of 0, i.e., the Bellman consistency conditions are met which means convergence.

- What is slow enough?

Robbins-Monro condition:

$$\sum_{t=1}^{\infty} \alpha_t = \infty \quad \text{and} \quad \sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

Keeps learning forever

Noise variance shrinks fast enough to suppress noise

- Example: $\alpha_t = \frac{1}{t}$ (equivalent to incremental averaging)

Advantages of TD Prediction Methods

- **Online updates** at each step. Bootstraps using estimate $V(S')$ for the value of the next state. The estimate is biased.

TD plans and MC has to wait for the end of the episode but is unbiased.

- **Model-free:** Only needs samples. DP needs the transition probabilities!

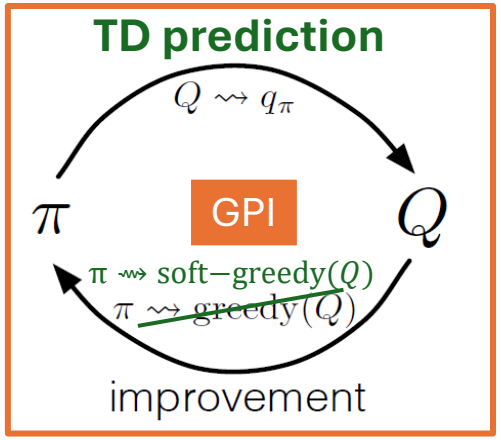
Same for MC, but TD needs the model.

- **Convergence**: Converges typically faster than MC prediction.

On-Policy TD Control: Sarsa

Find a good policy online while you follow the policy.

Sarsa



- **Idea**: Use **GPI** with TD prediction for policy evaluation. Exploration is guaranteed by learning a **soft policy.**

- **TD Evaluation**: Estimate $q_\pi(s, a)$ using a Q-table with updates:

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

$Q(S_{t+1}, A_{t+1})$ for a terminal state is defined as 0.

- **Improvement**: Lean a soft (exploring) policy based on Q.
- **Properties**:
  - **On-policy** means that it learns a soft policy and not the optimal deterministic policy!
  - Update uses $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$ which leads to the name **Sarsa** and requires us to know what A_{t+1} will be.

Alg. Sarsa: On-Policy TD Control

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 $V(terminal) = 0$

Loop for each episode:
  Initialize S
  Loop for each s
    $A \leftarrow$ action giv
    Take action A, obs
    $V(S) \leftarrow V(S) + \alpha$
    $S \leftarrow S'$
  until S is terminal

The learned and followed policy is expressed as a Q-function.

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

Follow and learn a soft policy. On-policy means b = π

We need A'

Update the Q-function with the observed next action.

Off-policy TD Control: Q-Learning

Learn a deterministic policy while following an exploring behavior policy.

Q-Learning

- An extremely influential method developed by Watkins (1989).
- Update rule only uses Sars observed by following the behavior policy:

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

The next action has not yet been observed. Updates with the greedy/optimal action for the target policy!

- **Off–policy**: Q directly approximates q_* independent of the behavior policy being followed because it ignores the next behavior action and updates with the next action for the greedy target policy instead.
- **Converges** to the optimal value function if:
  a) State-action pairs are represented discretely (e.g., in a table).
  b) All action are repeatedly sampled in all states (behavior policy keeps exploring).

Alg. Q-Learning

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
  Loop for each step
    Take action A, obse
    Choose A' from S'
    $Q(S, A) \leftarrow Q(S, A)$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

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

Exploring behavior policy

Does not use the actual behavior A' for the update! Uses instead what Q thinks would be the optimal next action. Learns q_* and a deterministic target π_*!

Q-Learning's Maximization Bias

- Q-learning update:

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

- The next action is not observed, but the best action according to Q is used in the update of Q.

- This cycle creates a **maximization bias** that leads to overestimating the Q- values.

- Solution:
  - **Double learning** by learning two independent Q-functions and using one for the action selection and the other one for the bootstrap estimate.

Double Q-Learning

- Learns two independent Q-functions. One is used for action selection and the other one for the bootstrap estimate.

Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q_1(s, a)$ and $Q_2(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, such that $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using the policy ε-greedy in $Q_1 + Q_2$
    Take action A, observe R, S'
    With 0.5 probabilility:

$$Q_1(S, A) \leftarrow Q_1(S, A) + \alpha\Big(R + \gamma Q_2\big(S', \operatorname{argmax}_a Q_1(S', a)\big) - Q_1(S, A)\Big)$$

    else:

$$Q_2(S, A) \leftarrow Q_2(S, A) + \alpha\Big(R + \gamma Q_1\big(S', \operatorname{argmax}_a Q_2(S', a)\big) - Q_2(S, A)\Big)$$

    $S \leftarrow S'$
  until S is terminal

Bootstrap estimate

Action selection

Eventually Q_1 and Q_2 should become equal.

Note: All TD methods have a similar bias, and double learning extensions for these algorithms also exist.

On-policy TD Control: Expected Sarsa

Reduce action sample variation for Sarsa.

Expected Sarsa

- Learn a soft policy and use the expectation over all possible next actions instead of the observed (Sarsa) or best (Q-Learning) action:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \mathbb{E}_\pi[\, Q(S_{t+1}, A_{t+1}) |\, S_{t+1}] - Q(S_t, A_t)]$$

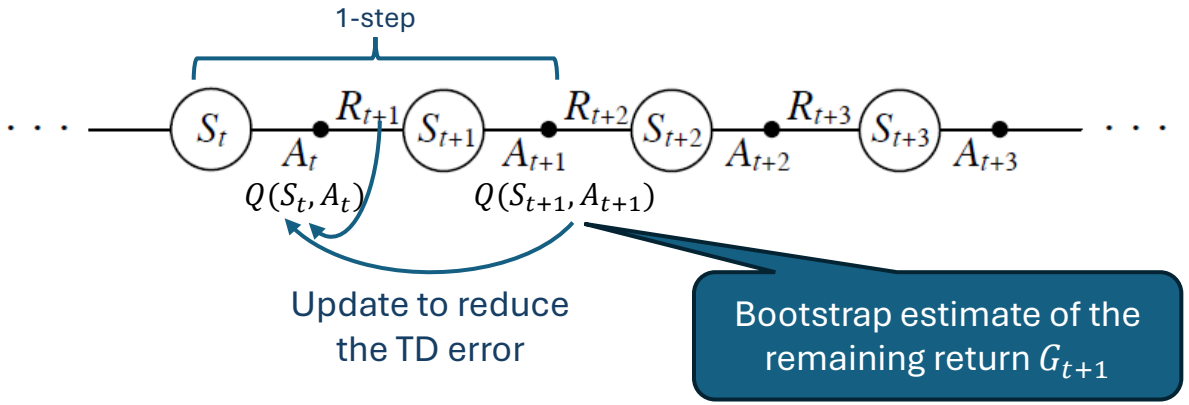
$$= Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

- **Properties:**
  - On-policy like Sarsa.
  - Eliminates Sarsa's variance due to the selection of a single A_{t+1} from a soft policy.
  - Typically uses a learning rate $\alpha = 1$ since it already uses the expectation!

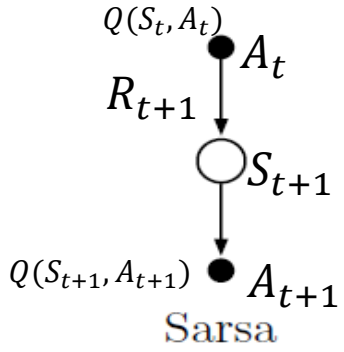
Comparison

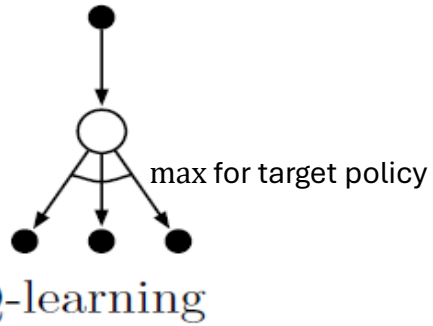
Sarsa vs. Q-Learning vs. Expected Sarsa

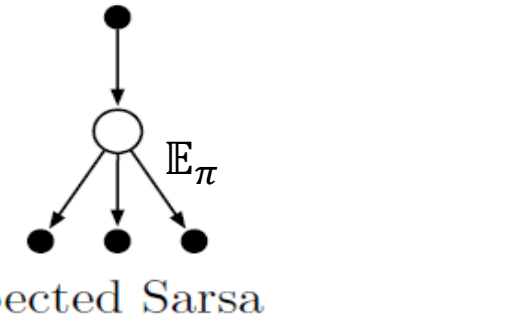
Comparison: 1-step Backup Diagrams



Target: $U = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$ $U = R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$ $U = R_{t+1} + \gamma \mathbb{E}_\pi[Q(S_{t+1}, A_{t+1})|S_{t+1}]$







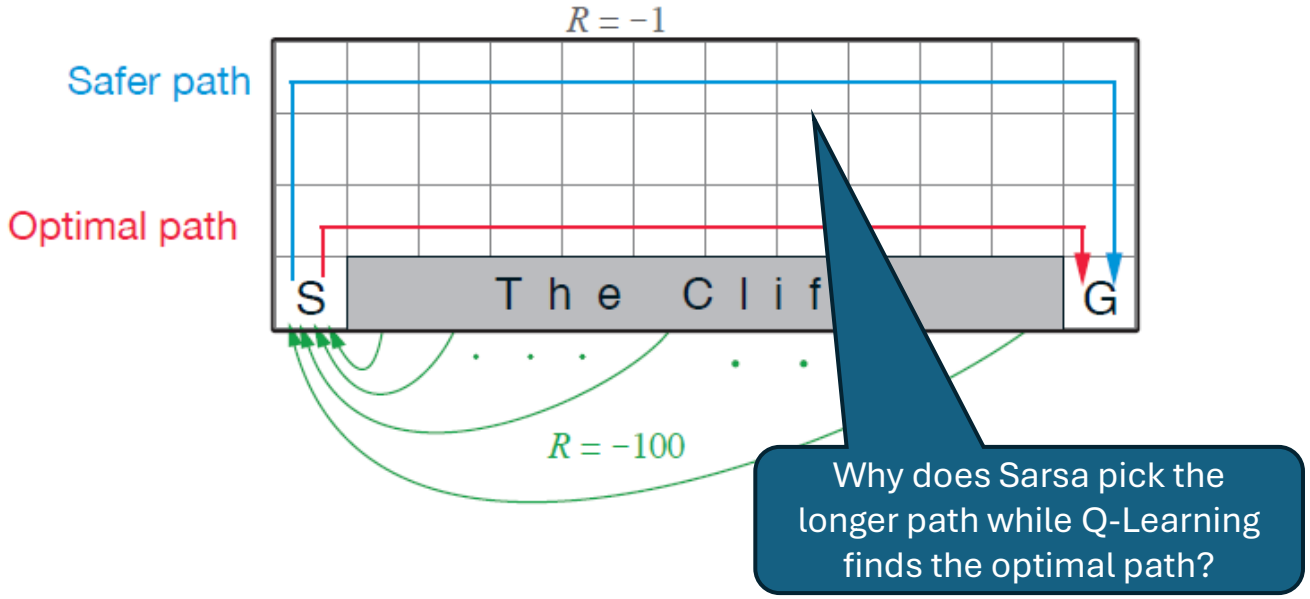
On-policy: Updates with a next action following the learned policy.

Off-policy: Updates with the best next action for the target policy.

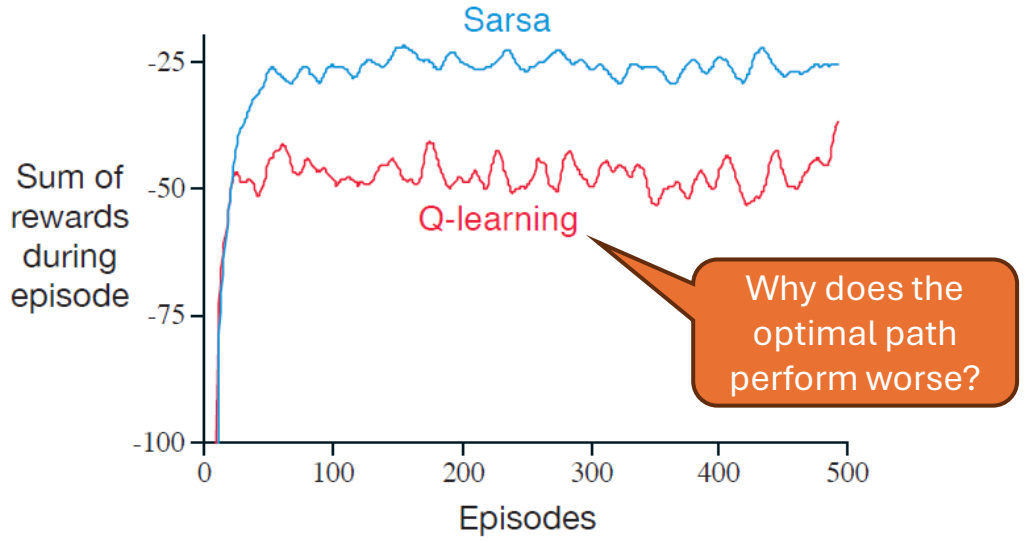
On-policy: Updates with the expectation over the action for the learned policy.

Sarsa vs. Q-Learning

- **Cliff Walking**: Consider the grid world shown to the right.
- This is a standard, undiscounted, episodic task with start and goal states.
- 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.

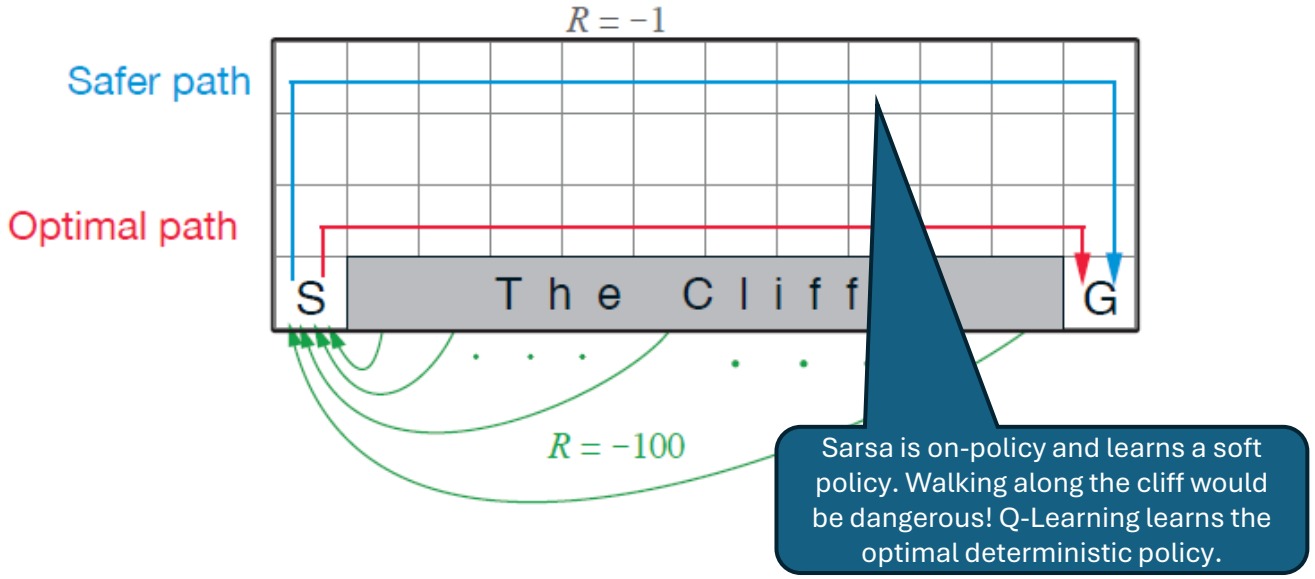


Performance of the Sarsa and Q-learning with ϵ-greedy action selection, $\epsilon = 0.1$.

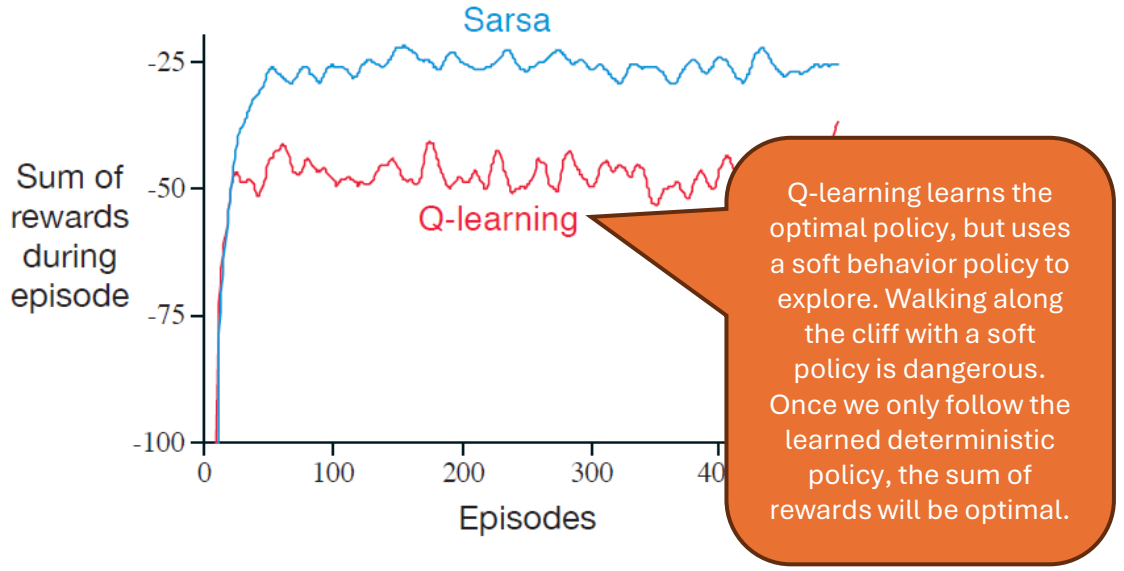


Sarsa vs. Q-Learning

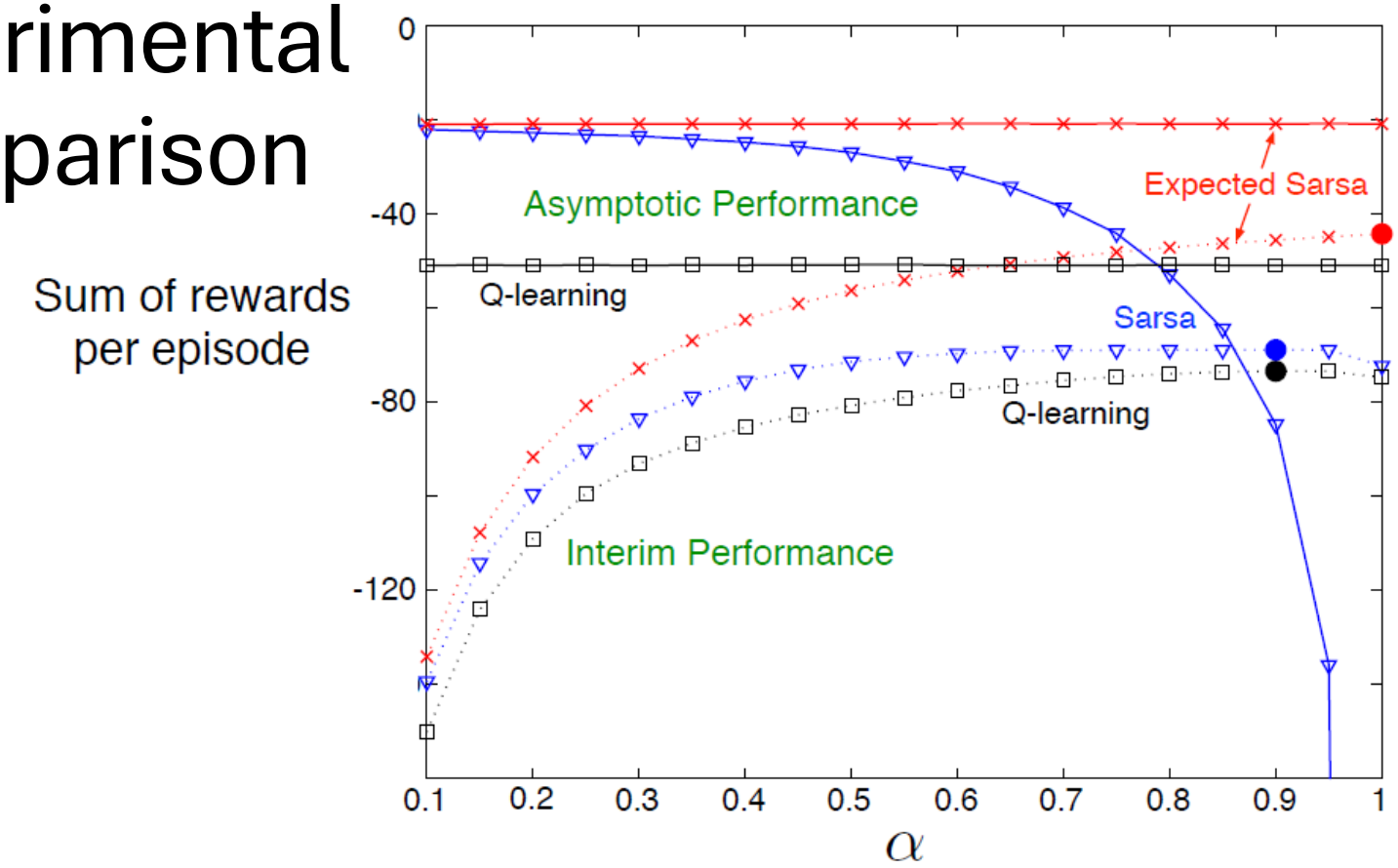
- **Cliff Walking**: Consider the grid world shown to the right.
- This is a standard, undiscounted, episodic task with start and goal states.
- Actions causing movement up, down, right, and left.
- Reward is −1 on all transitions except those into the region marked “The Cliff.” Stepping into this region incurs a reward of −100 and sends the agent instantly back to the start.



Performance of the Sarsa and Q-learning methods with ϵ-greedy action selection, $\epsilon = 0.1$.



Experimental Comparison



Observations:

- Performance measure is for the behavior policy. Q-Learning learns the optimal policy.
- Sarsa is very sensitive to α! (asymptotic performance).
- Expected Sarsa should use $\alpha = 1$. For the other methods, α is a problem-dependent tuning parameter (solid circles).

Figure 6.3: Interim and asymptotic performance of TD control methods on the cliff-walking task as a function of α. All algorithms used an ε-greedy policy with $\varepsilon = 0.1$. Asymptotic performance is an average over 100,000 episodes whereas interim performance is an average over the first 100 episodes. These data are averages of over 50,000 and 10 runs for the interim and asymptotic cases respectively. The solid circles mark the best interim performance of each method. Adapted from van Seijen et al. (2009).

What You Should Know

- Time difference (TD) learning is an alternative to MC for the prediction problem (policy evaluation).
- **Method**: Stepwise reduces the TD error for the next step.
- **Popular and most widely used RL methods**:
  - On-policy: Sarsa
  - Off-policy: Q-learning
- **Advantages**:
  - Model-free: Uses only experience.
  - Truly online learning after each observation.
  - Simple with minimal computation.
- **Challenges**:
  - Bootstrapping introduces bias.
  - Maximation bias (especially Q-learning).
  - Behavior policy needs to keep exploring.
  - Not sample efficient: uses observations only once.
  - Tabular: Requires too much memory and does not generalize between states.

- We will extend the one-step tabular TD ideas from this chapter to ***n*-steps** and **function approximation**.