Deep Reinforcement Learning

Lecture 5: Q-Learning, SARSA, and Their Variants

Instructor: Chongjie Zhang

Tsinghua University

Review

- Passive learning
  - Monte Carlo direct estimation (model-free)
  - Adaptive dynamic programming (ADP) (model-based)
  - Temporal difference (TD) learning (model-free)

- Active learning
  - ADP-based learning (model-based)
  - TD-based learning
  - Q-learning and its variants (off-policy model-free)
  - SARSA and its variants (on-policy model-free)

Outline

- Passive learning
  - Monte Carlo direct estimation (model-free)
  - Adaptive dynamic programming (ADP) (model-based)
  - Temporal difference (TD) learning (model-free)

- Active learning
  - ADP-based learning (model-based)
  - TD-based learning
  - Q-learning and its variants (off-policy model-free)
  - SARSA and its variants (on-policy model-free)

TD-based Active RL

1. Start with an initial value function
2. Take an action from an exploration/exploitation policy giving new state s’ (should converge to a greedy policy, i.e. GLIE)
3. Update an estimated model
4. Perform TD update

$$V(s) \leftarrow V(s) + \alpha(R(s) + \gamma V(s') - V(s))$$

V(s) is a new estimate of optimal value function at state s.

5. Goto 2

Just like TD for passive RL, but we follow an exploration/exploitation policy

Given the usual assumptions about learning rate and GLIE, TD will converge to an optimal value function!

TD-based Active RL

1. Start with an initial value function
2. Take an action from an exploration/exploitation policy giving new state s'
(should converge to a greedy policy, i.e. GLIE)
3. Update an estimated model
4. Perform TD update

$$V(s) \leftarrow V(s) + \alpha(R(s) + \gamma V(s') - V(s))$$

V(s) is a new estimate of optimal value function at state s.

5. Goto 2

Requires an estimated model. Why?

$\boxed{\text{To compute the exploration/exploitation policy.}}$

TD-Based Active RL

- Exploration/Exploitation policy requires computing argmax Q(s, a) for the exploitation part of the policy
  - Computing argmax Q(s, a) requires T in addition to V
- Thus TD-learning must still maintain an estimated model for action selection
- It is computationally more efficient at each step compared to Rmax (i.e., optimistic exploration)
  - TD-update vs. Value Iteration
  - But a model requires much more memory than a value function
- Can we get a model-fee variant?

Outline

- Passive learning
  - Monte Carlo direct estimation (model-free)
  - Adaptive dynamic programming (ADP) (model-based)
  - Temporal difference (TD) learning (model-free)

- Active learning
  - ADP-based learning (model-based)
  - TD-based learning
  - **Q-learning and its variants (off-policy model-free)**
  - SARSA and its variants (on-policy model-free)

Q: take 1 step & then optimal

Q-Learning: Model-Free RL

definition of Q function

- Instead of learning the optimal value function V, directly learn the optimal Q function.
  - Recall Q(s, a) is the expected value of taking action a in state s and then following the optimal policy thereafter
- Given the Q function we can act optimally by selecting action greedily according to Q(s, a) without a model
- The optimal Q-function satisfies $V(s) = \max_{a'} Q(s, a')$ which gives:

$$Q(s,a) = R(s) + \gamma \sum_{s'} T(s,a,s') V(s')$$

$$= R(s) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(s',a')$$

How can we learn the Q-function directly?

Q-Learning: Model-Free RL

<u>Bellman constraints on the optimal Q-function:</u>

$$Q(s,a) = R(s) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(s,a')$$

- We can perform updates after each action just like in TD.
  - After taking action a in state s and reaching state s' do:
    (note that we directly observe reward R(s))

like a learning rate

$$Q(s,a) \leftarrow Q(s,a) + \alpha\Big(\underbrace{R(s) + \gamma \max_{a'} Q(s',a')}_{\text{(noisy) sample of Q-value based on next state}} - Q(s,a)\Big)$$

take action a in state s

greedy update

Q-Learning

1. Start with an initial Q-function (e.g. all zeros)
2. Take an action from an exploration/exploitation policy giving new state s’ (should converge to a greedy policy, i.e. GLIE)
3. Perform TD update

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

   Q(s, a) is the current estimate of the optimal Q-function.

4. Goto 2

- Does not require model since we learn Q directly!
- Uses explicit |S| x |A| table to represent Q
- **Off-policy learning**: the update does not depend on the actual next action
- The exploration/exploitation policy directly uses Q-values
  - E.g., use Boltzmann exploration.

Q-Learning Convergence

almost sure convergence

- **Theorem 1** : Q-learning converges to the optimal Q-value function in the limit with probability 1, if
  - Every state-action pair is visited infinitely often
  - Learning rate decays just so: $\sum_{n=1}^{\infty} \alpha_n = \infty$ and $\sum_{n=1}^{\infty} \alpha_n^2 < \infty$

Stochastic Approximation

- Recursive update rules to solve optimization problems and fixed point equations
- A simple goal: Find the solution for $\bar{f}(\theta^*) := \mathsf{E}[f(\theta, W)]\Big|_{\theta=\theta^*} = 0$
- What makes this hard?
  - The function f and the distribution of the random vector W may not be known
  - Even if everything is known, computation of the expectation may be expensive. For root finding, we may need to compute the expectation for many values of θ
  - Motivates stochastic approximation: $\theta(n+1) = \theta(n) + \alpha_n f(\theta(n), W(n))$

Stochastic Approximation

Theorem 2. *The random process* $\{\Delta_t\}$ *taking values in* $\mathbb{R}^n$ *and defined as*

$$\Delta_{t+1}(x) = (1 - \alpha_t(x))\Delta_t(x) + \alpha_t(x)F_t(x)$$

converges to zero w.p.1 under the following assumptions:

- $0 \leq \alpha_t \leq 1$, $\sum_t \alpha_t(x) = \infty$ *and* $\sum_t \alpha_t^2(x) < \infty$;
- $\left\|\mathbb{E}\left[F_t(x) \mid \mathcal{F}_t\right]\right\|_W \leq \gamma \left\|\Delta_t\right\|_W$, *with* $\gamma < 1$;
- $\mathbf{var}\left[F_t(x) \mid \mathcal{F}_t\right] \leq C(1 + \left\|\Delta_t\right\|_W^2)$, *for* $C > 0$.

Tommi Jaakkola, Michael I. Jordan, and Satinder P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. Neural Computation, 6(6):1185–1201, 1994.

Proof of Q-Learning Convergence

$$\Delta_{t+1}(x) = (1 - \alpha_t(x))\Delta_t(x) + \alpha_t(x)F_t(x)$$

- $\|\mathbb{E}\left[F_t(x) \mid \mathcal{F}_t\right]\|_W \leq \gamma \|\Delta_t\|_W$, *with* $\gamma < 1$;
- $\mathbf{var}\left[F_t(x) \mid \mathcal{F}_t\right] \leq C(1 + \|\Delta_t\|_W^2)$, *for* $C > 0$.

Q-Learning: Speedup for Goal-Based Problems

- **Goal-Based Problem:** receive big reward in goal state and then transition to terminal state
- Initializing Q(s, a) to zeros and then observing the following sequence of (state, reward, action) triples
  - (s0, 0, a0) (s1, 0, a1) (s2, 10, a2) (terminal,0)
- The sequence of Q-value updates would result in: Q(s0, a0) = 0, Q(s1, a1) =0, Q(s2, a2)=10
- So nothing was learned at s0 and s1
  - Next time this trajectory is observed we will get non-zero for Q(s1, a1) but still Q(s0, a0)=0

Q-Learning: Speedup for Goal-Based Problems

- From the example we see that it can take many learning trials for the final reward to "back propagate" to early state-action pairs

- Two approaches for addressing this problem:
  1. **<u>Trajectory replay</u>**: store each trajectory and do several iterations of Q-updates on each one
  2. **<u>Reverse updates</u>:** store trajectory and do Q-updates in reverse order

- In our example (with learning rate and discount factor equal to 1 for ease of illustration) reverse updates would give
  - Q(s2,a2) = 10, Q(s1,a1) = 10, Q(s0,a0)=10

Improve Convergence Rate of Q-Learning

- Slow convergence of Q-learning: due to the learning rate

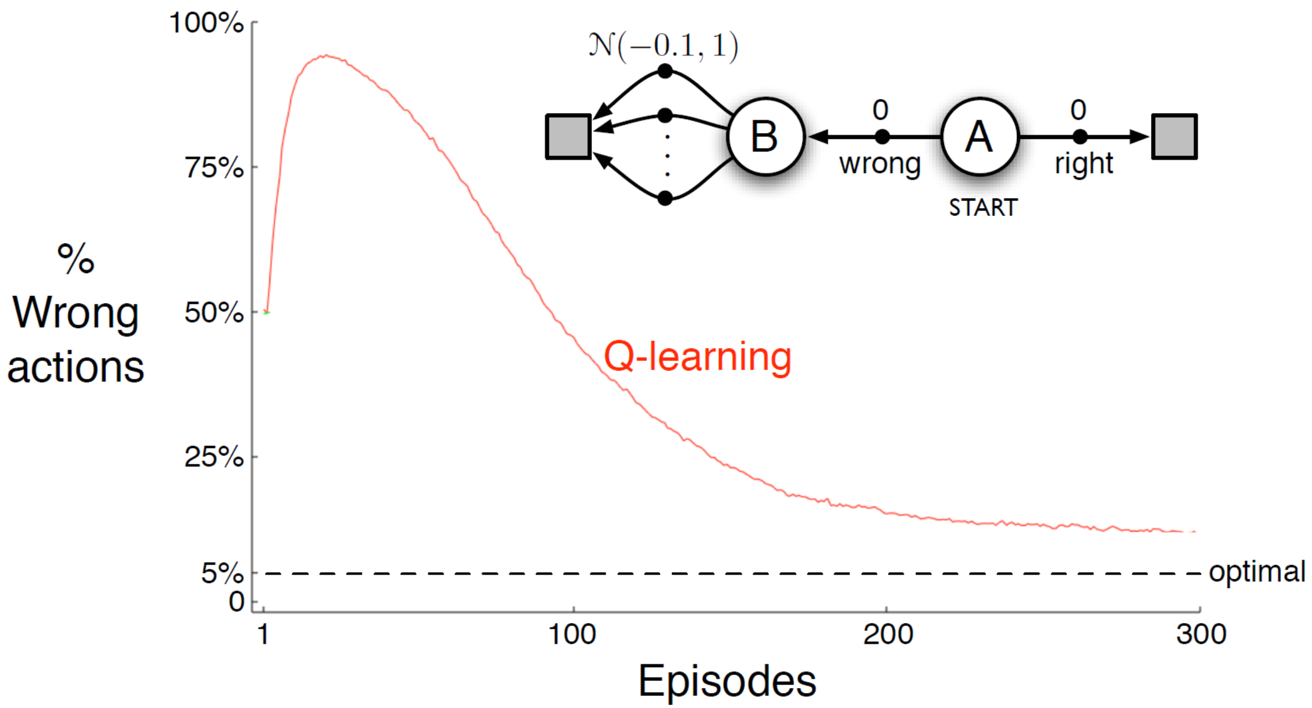
$$Q_{k+1}(x,a) = Q_k(x,a) + \alpha_k \big(\mathfrak{T}_k Q_k(x,a) - Q_k(x,a)\big)$$

- Speedy Q-Learning [Azar et al., NIPS 2011]
  - Use previous estimates of Q value function

$$Q_{k+1}(x,a) = Q_k(x,a) + \alpha_k \big(\mathfrak{T}_k Q_{k-1}(x,a) - Q_k(x,a)\big) + (1-\alpha_k)\big(\mathfrak{T}_k Q_k(x,a) - \mathfrak{T}_k Q_{k-1}(x,a)\big)$$

- Zap Q-Learning [Devraj & Meyn NIPS 2017]
  - Achieving the fastest convergence among Q-learning like algorithms
  - Designing the optimal gain matrix G_n so that the recursion $X_{n+1} = X_n + G_n f(X_n)$ converges the fastest

Maximization Bias of Q-Learning



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

Double Q-Learning (Hado van Hasselt 2010)

- Train 2 action-value functions, Q1 and Q2
- Do Q-learning on both, but
  - never on the same time steps (Q1 and Q2 are indep.)
  - pick Q1 or Q2 at random to be updated on each step
- If updating Q1, use Q2 for the value of the next state:

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha \Big(R_{t+1} + Q_2\big(S_{t+1}, \arg\max_a Q_1(S_{t+1}, a)\big) - Q_1(S_t, A_t) \Big)$$

- Action selections are (say) ε-greedy with respect to the sum of Q1 and Q2

Double Q-Learning (Hado van Hasselt 2010)

Initialize $Q_1(s,a)$ and $Q_2(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily
Initialize $Q_1(\textit{terminal-state}, \cdot) = Q_2(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        Choose A from S using policy derived from Q_1 and Q_2 (e.g., ε-greedy in $Q_1 + Q_2$)
        Take action A, observe R, S'
        With 0.5 probabilility:

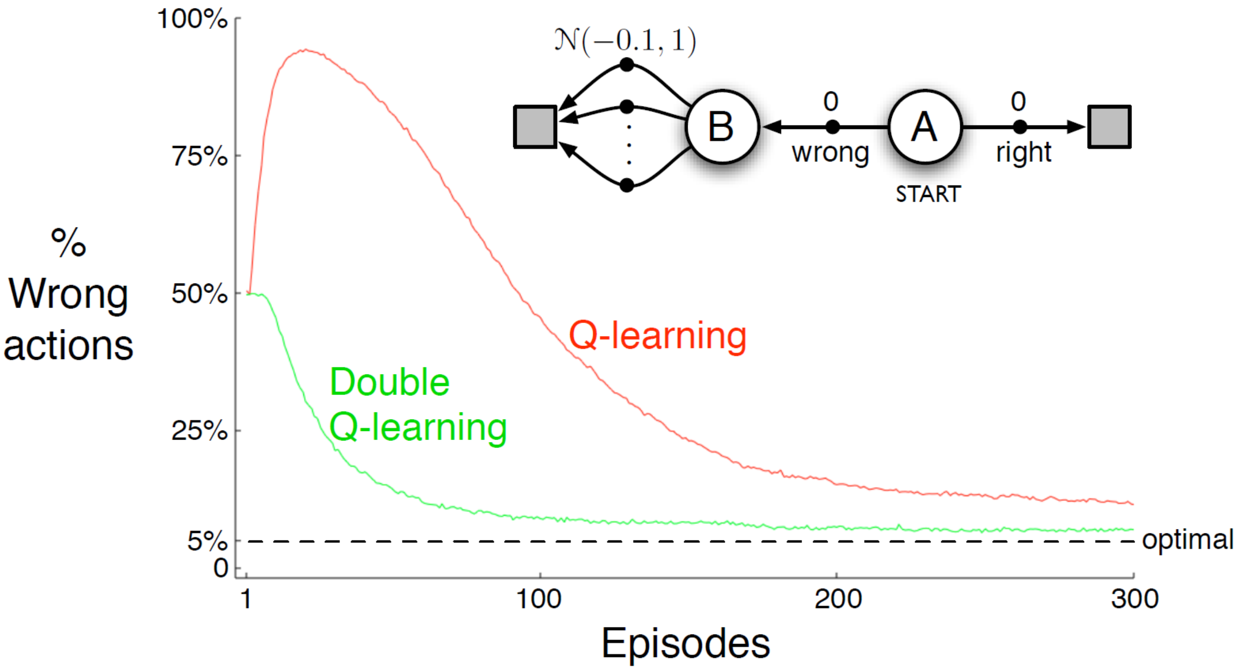
$$Q_1(S,A) \leftarrow Q_1(S,A) + \alpha\Big(R + \gamma Q_2\big(S', \operatorname{argmax}_a Q_1(S',a)\big) - Q_1(S,A)\Big)$$

        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

Example of Maximization Bias

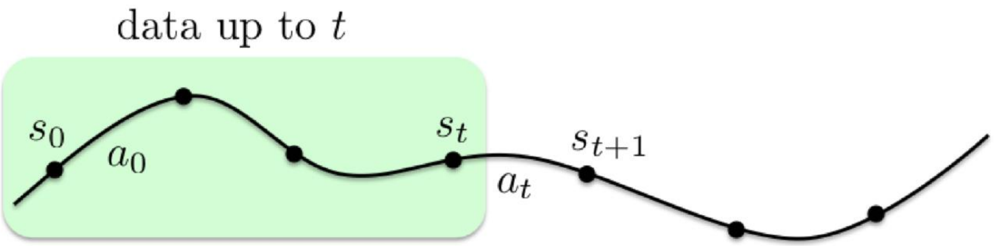


Double Q-learning:

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma Q_2\big(S_{t+1}, \arg\max_a Q_1(S_{t+1}, a)\big) - Q_1(S_t, A_t) \Big]$$

Off vs. On Policy

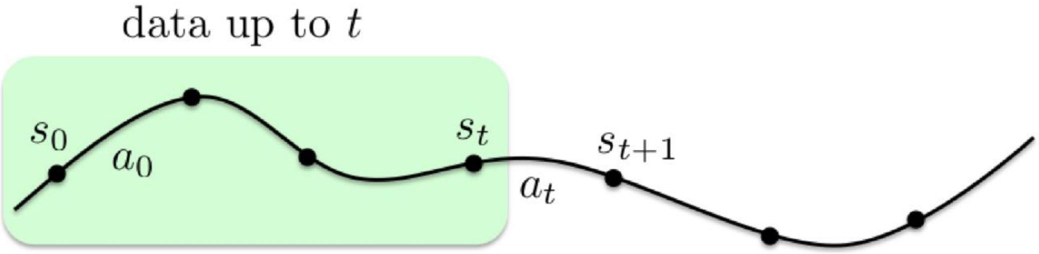
- An **off-policy** learner learns the value of the optimal policy independently of the agent's actions.
- The policy used by the agent is often referred to as the behavior policy, and denoted by π_b.
- Examples: Q-learning is an off-policy



$$s_{t+1} \sim p(\cdot|s_t, \pi_b(s_t))$$
$$\pi_t = \pi_b$$

Off vs. On Policy

- Data: the observations made by the agent along the trajectory of the dynamical system.
- An **on-policy** learner learns the value of the policy being carried out by the agent including the exploration steps.
- The policy used by the agent is computed from the previous collected data. It is an *active learning* method as the gathered data is controlled.



$$s_{t+1} \sim p(\cdot | s_t, \pi_t(s_t))$$

$$\pi_t = F_t(s_0, a_0, \ldots, s_{t-1}, a_{t-1}, s_t)$$

Outline

- Passive learning
  - Monte Carlo direct estimation (model-free)
  - Adaptive dynamic programming (ADP) (model-based)
  - Temporal difference (TD) learning (model-free)

- Active learning
  - ADP-based learning (model-based)
  - TD-based learning
  - Q-learning and its variants (off-policy model-free)
  - **SARSA and its variants (on-policy model-free)**

State-Action-Reward-State-Action (SARSA)

- Update rule

  $Q_{n+1}(s_n, a_n) = Q_n(s_n, a_n) + \alpha_n\, [r(s_n, a_n) + \lambda Q_n(s_{n+1}, a_{n+1}) - Q_n(s_n, a_n)]$

- **On-policy** algorithm:
  - We select actions according to a policy defined through Q_n
  - The Q-value update depends on the actual next action

- ε-greedy policy:
  - w.p. $1 - \varepsilon$, select $a \in \text{argmax}\ Q_n(s_n, b)$
  - w.p. ε, select a uniformly at random

SARSA

1. Start with initial Q-function (e.g. all zeros)
2. Take action a_n on state s_n from an Ɛ-greedy policy giving new state s_{n+1}
3. Take action a_{n+1} on state s_{n+1} from an Ɛ-greedy policy
4. Perform TD update

$$Q_{n+1}(s_n, a_n) = Q_n(s_n, a_n) + \alpha_n \, [r(s_n, a_n) + \lambda Q_n(s_{n+1}, a_{n+1}) - Q_n(s_n, a_n)]$$

5. Goto 2

SARSA Convergence

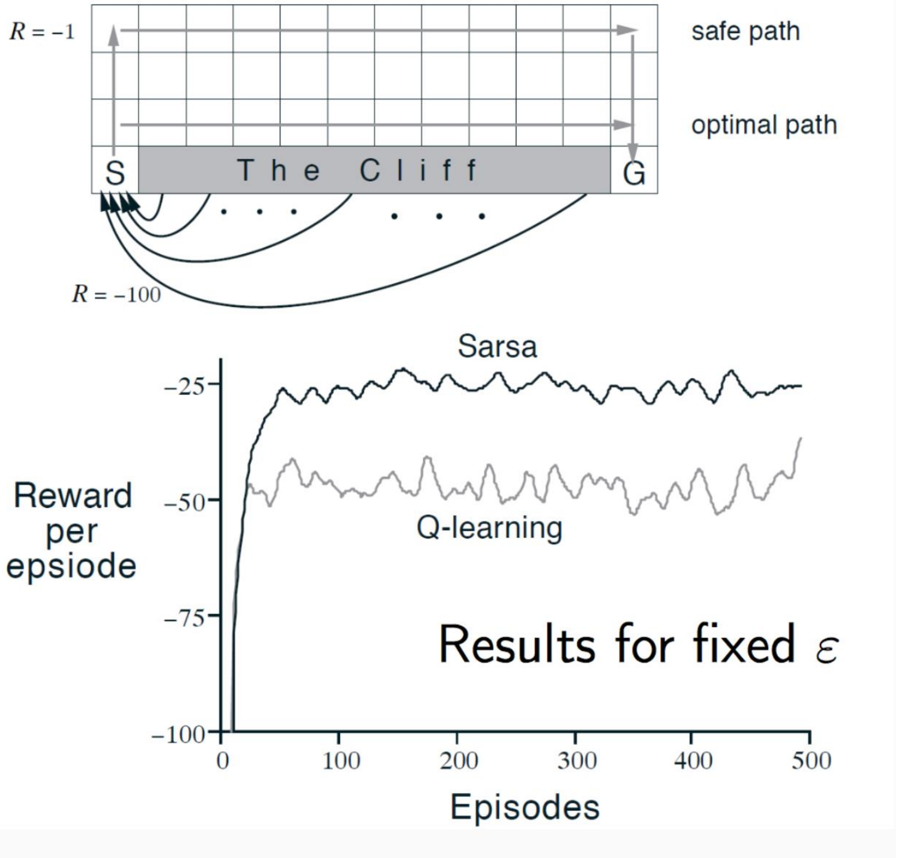
Theorem. Assume that the step sizes (α_n) satisfy (A2), and that SARSA is based on the ϵ-greedy policy. For any discount factor $\lambda \in (0,1)$:

$$\lim_{n\to\infty} Q_n = Q^\epsilon, \quad \text{almost surely}$$

where $Q^\epsilon(s,a) = r(s,a) + \lambda \sum_j p(j|s,a) V^{\star\epsilon}(j)$ and $V^{\star\epsilon}$ is the value function of the policy selecting an optimal action w.p. $1-\epsilon$ and a (uniform) random action w.p. ϵ.

On-policy algorithms are ”safer”, they do not explore (state, action) pairs yielding very negative rewards (for fixed exploration rate Ɛ > 0, SARSA does not converge to the optimal policy)

On-policy vs. off-policy



N-Step SARSA

- Consider the following n-step returns

$$
\begin{array}{lll}
n = 1 & (Sarsa) & q_t^{(1)} = R_{t+1} + \gamma Q(S_{t+1}) \\
n = 2 & & q_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 Q(S_{t+2}) \\
\vdots & & \vdots \\
n = \infty & (MC) & q_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_T
\end{array}
$$

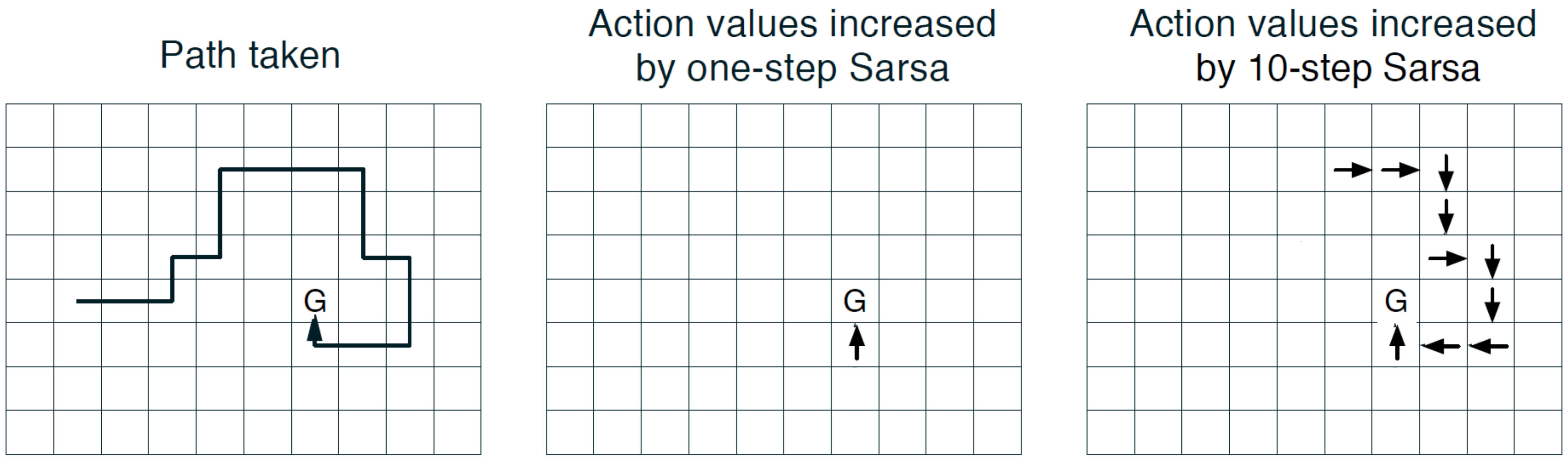
- Define the n-step Q-return

$$q_t^{(n)} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n})$$

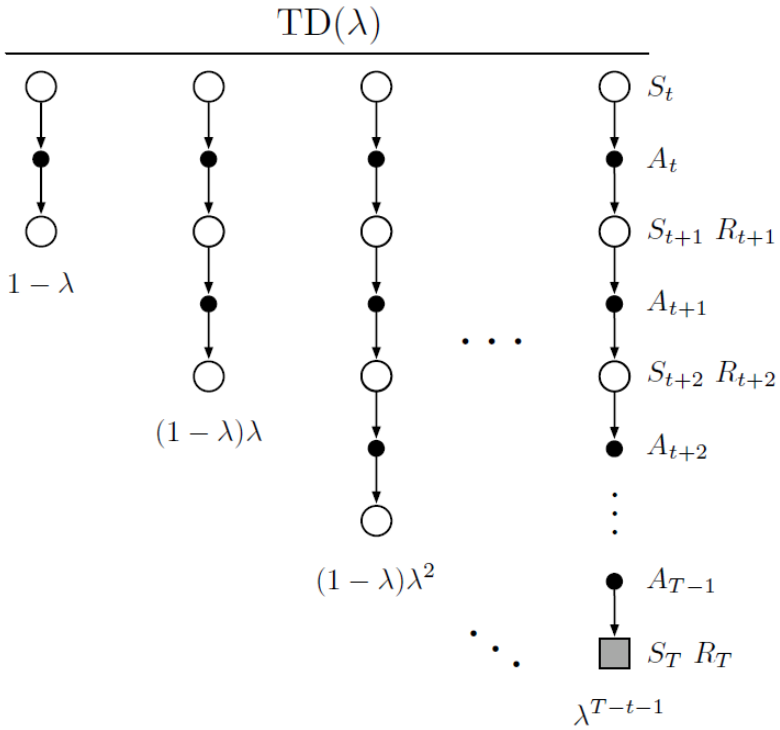
- n-step SARSA updates Q(s, a) towards the n-step Q-return

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(q_t^{(n)} - Q(S_t, A_t) \right)$$

N-Step Methods Can Accelerate Learning



SARSA(λ)



- The q^λ return combines all n-step Q-returns $q^{(n)}$

$$q_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}q_t^{(n)}$$

- Forward View of SARSA(λ)

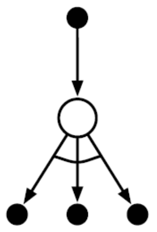
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left(q_t^\lambda - Q(S_t, A_t)\right)$$

Expected SARSA

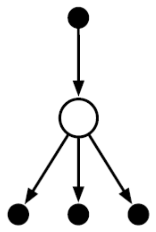
Van Seijen, van Hasselt, Whiteson, & Wiering 2009

- Instead of the *sample* value-of-next-state, use the expectation!

$$
\begin{aligned}
Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \mathbb{E}[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t) \Big] \\
&\leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \Big]
\end{aligned}
$$



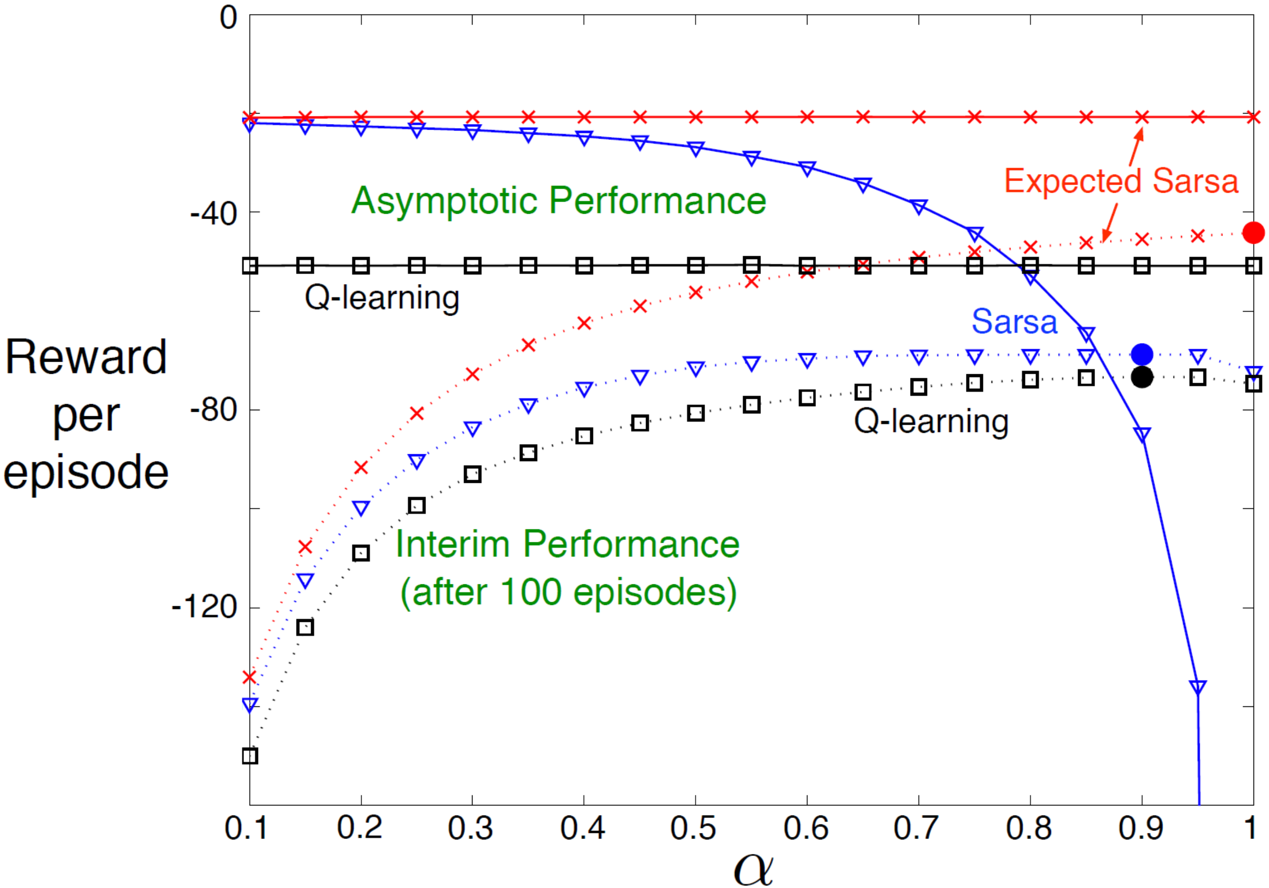
Q-learning



Expected Sarsa

- Expected Sarsa's performs better than Sarsa (but costs more)

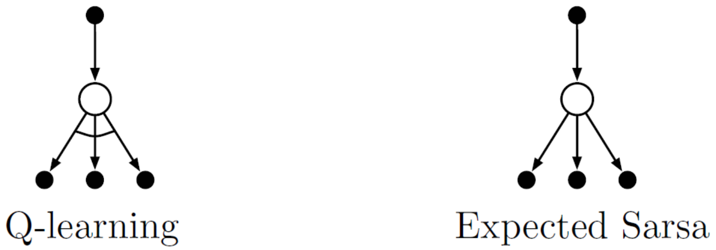
Performance on the Cliff-Walking Task



Off-policy Expected SARSA

- Expected Sarsa generalizes to arbitrary behavior policies
  - in which case it includes Q-learning as the special case in which π is the greedy policy

$$
\begin{aligned}
Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \mathbb{E}[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t) \Big] \\
&\leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \Big]
\end{aligned}
$$



- This idea seems to be new

Active Reinforcement Learning Summary

- Methods
  - ADP-based (model-based) RL
  - Temporal Difference Learning
  - Q-learning and SARSA (model-free)
- All converge to the optimal policy assuming a GLIE exploration strategy
  - Optimistic exploration with ADP can be shown to converge in polynomial time with high probability
- All methods assume the world is not too dangerous (no cliffs to fall off during exploration)
- So far we have assumed small state spaces

ADP vs. TD vs. Q

- Different opinions.
- When state space is small, this is not such an important issue.
- Computation Time
  - ADP-based methods use more computation time per step
- Memory Usage
  - ADP-based methods uses $O(mn^2)$ memory
  - Active TD-learning uses $O(mn^2)$ memory (must store model)
  - Q-learning uses $O(mn)$ memory for Q-table
- Learning efficiency (performance per unit experience)
  - ADP-based methods make more efficient use of experience by storing a model that summarizes the history and then reasoning about the model (e.g. via value iteration or policy iteration)

Large State Spaces

- RL algorithms presented so far have little chance to solve real-world problems when the state (or action) space is large.
  - not longer represent the V or Q functions as explicit tables

- Even if we had enough memory
  - Never enough training data!
  - Learning takes too long

- What to do??

What about large state spaces?

- One approach is to map the original state space S to a much smaller state space S’ via some hashing function.
  - Ideally “similar” states in S are mapped to the same state in S’
- Then do learning over S’ instead of S.
  - Note that the world may not look Markovian when viewed through the lens of S’, so convergence results may not apply
  - But, still the approach can work if a good enough S’ is engineered (requires careful design), e.g.
  - *Empirical Evaluation of a Reinforcement Learning Spoken Dialogue System.* With S. Singh, D. Litman, M. Walker. *AAAI*, 2000
- We will now study three other approaches
  - Value function approximation
  - Policy gradient methods
  - Actor-critic methods