6. Model-free Control

2019 Fall

Yusung Kim
yskim525@skku.edu

Model-free Reinforcement Learning

- Model-free prediction (evaluation)
  - Estimate the value function of an unknown MDP
  - How good is this given policy ?
- Model-free control (improvement)
  - Optimize the value function of an unknown MDP
  - How can we learn a better policy ?

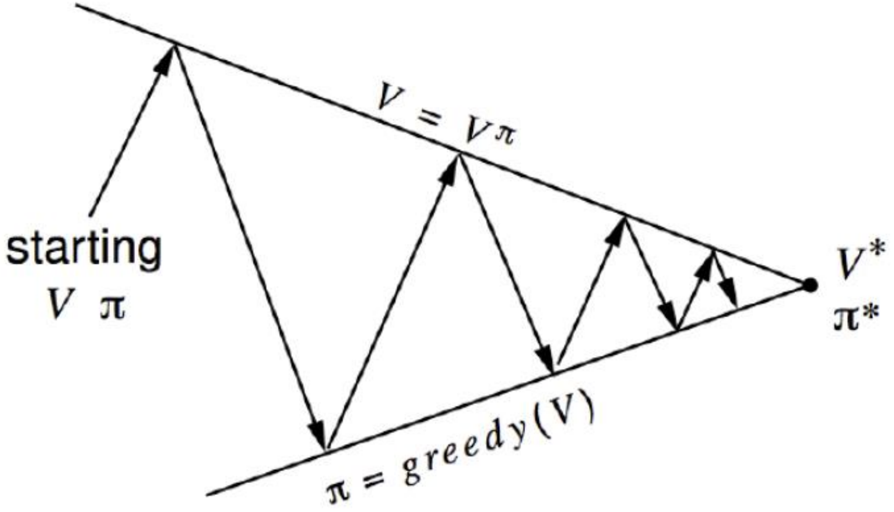
Uses of Model-Free Control

- Many applications can be modeled as a MDP:
  - Backgammon, Go, Robot locomotion, Helicopter flight, Robocup soccer, Autonomous driving, Customer ad selection, Patient treatment
- For most of these problems, either:
  - MDP model is unknown, but experience can be sampled
  - MDP model is known, but is computationally infeasible to use directly, except through sampling
- Model-free control can solve these problems

On and Off-Policy Learning

- On-policy learning
  - Direct experience
  - Learn to estimate and evaluate a policy π from experience obtained from following that policy π
- Off-policy learning
  - Lear to estimate and evaluate a policy π using experience gathered from following a different policy π'

Generalized Policy Iteration (Remind)

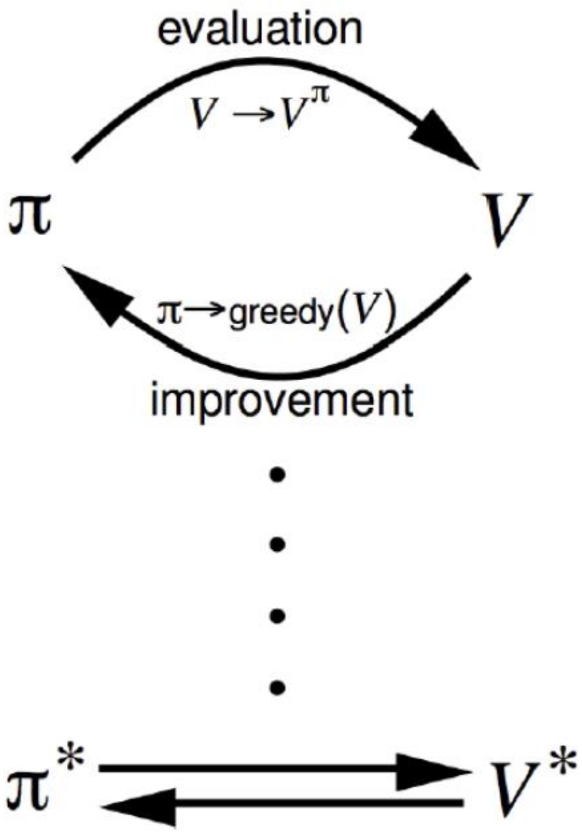


Policy evaluation Estimate v_π

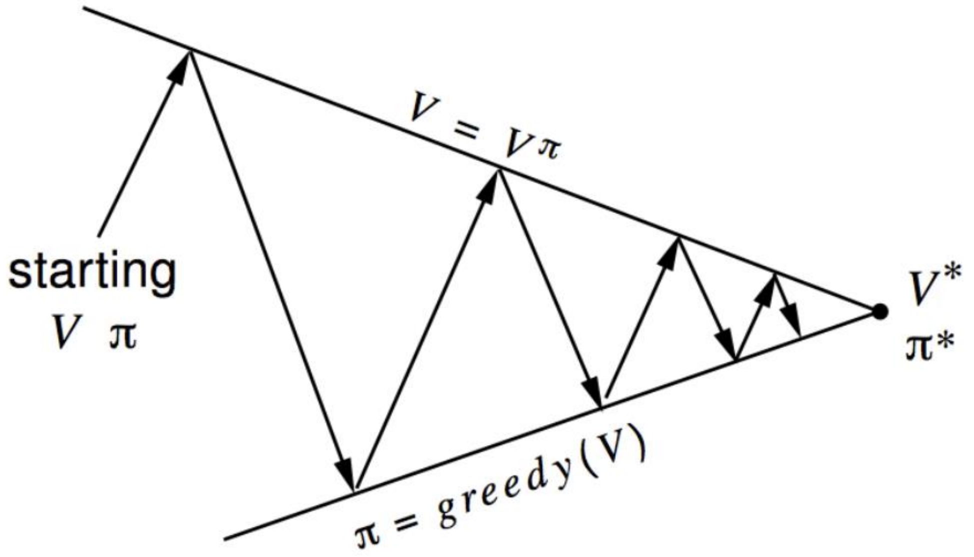
Any policy evaluation algorithm

Policy improvement Generate $\pi' \geq \pi$

Any policy improvement algorithm



Generalized Policy Iteration with MC Evaluation



Policy evaluation Monte-Carlo policy evaluation, $V = v_\pi$?

Policy improvement Greedy policy improvement?

Model-Free Policy Improvement

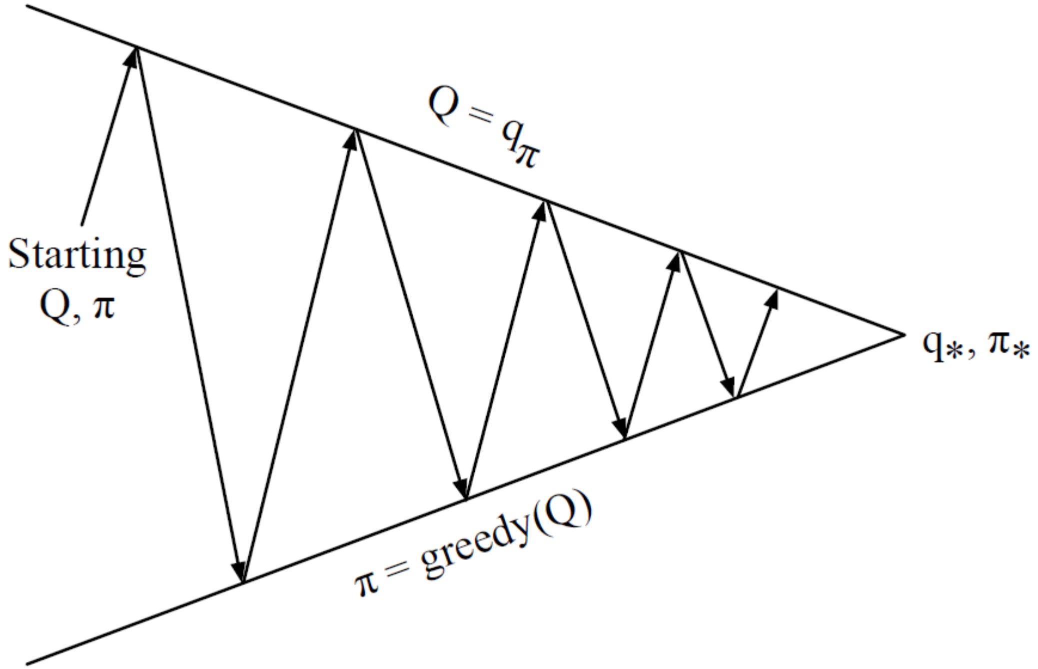
- Greedy policy improvement over $V(s)$ requires model of MDP

$$\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ \mathcal{R}_s^a + \mathcal{P}_{ss'}^a V(s')$$

- Greedy policy improvement over $Q(s,a)$ is model-free

$$\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ Q(s,a)$$

Generalized Policy Iteration with Q-Function



Policy evaluation Monte-Carlo policy evaluation, $Q = q_\pi$

Policy improvement Greedy policy improvement?

Example of Greedy Action Selection

return 0
return 0
return 0
return 0
return 100
return 0
return 0
return 0
return 0
return 100

return 1
return 1
return 3
return 3
return 1
return 1
return 3
return 3
return 1
return 1

Which door would you open ?

Exploration and Exploitation

- Reinforcement learning is like trial-and-error learning
- The agent should discover a good policy from its experiences of the environment without losing too much reward along the way

- Exploration finds more information about the environment
- Exploitation exploits known information to maximize reward
- It is usually important to explore as well as exploit.

ϵ-Greedy Exploration

- ϵ-greedy Exploration
  - simplest idea for ensuring continual exploration
  - all m actions are tried with non-zero probability
  - with probability 1 - ϵ choose the greedy action
  - with probability ϵ choose an action at random

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a^* = \underset{a \in \mathcal{A}}{\text{argmax}}\ Q(s,a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

where m is the number of actions

```
eps = 0.1

if rand() > eps :
      a = argmax( Q(s,a) )
else:
      a = random_action()
```

ϵ-Greedy Policy Improvement

Theorem

For any ϵ-greedy policy π, the ϵ-greedy policy π' with respect to q_π is an improvement, $v_{\pi'}(s) \geq v_\pi(s)$

$$
\begin{aligned}
q_\pi(s, \pi'(s)) &= \sum_{a \in \mathcal{A}} \pi'(a|s) q_\pi(s,a) \\
&= \epsilon/m \sum_{a \in \mathcal{A}} q_\pi(s,a) + (1-\epsilon) \max_{a \in \mathcal{A}} q_\pi(s,a) \\
&\geq \epsilon/m \sum_{a \in \mathcal{A}} q_\pi(s,a) + (1-\epsilon) \sum_{a \in \mathcal{A}} \frac{\pi(a|s) - \epsilon/m}{1-\epsilon} q_\pi(s,a) \\
&= \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a) = v_\pi(s)
\end{aligned}
$$

모든 확률별로 action을 할 수 있는데 가장 높은 것을 뽑는다. 최댓값

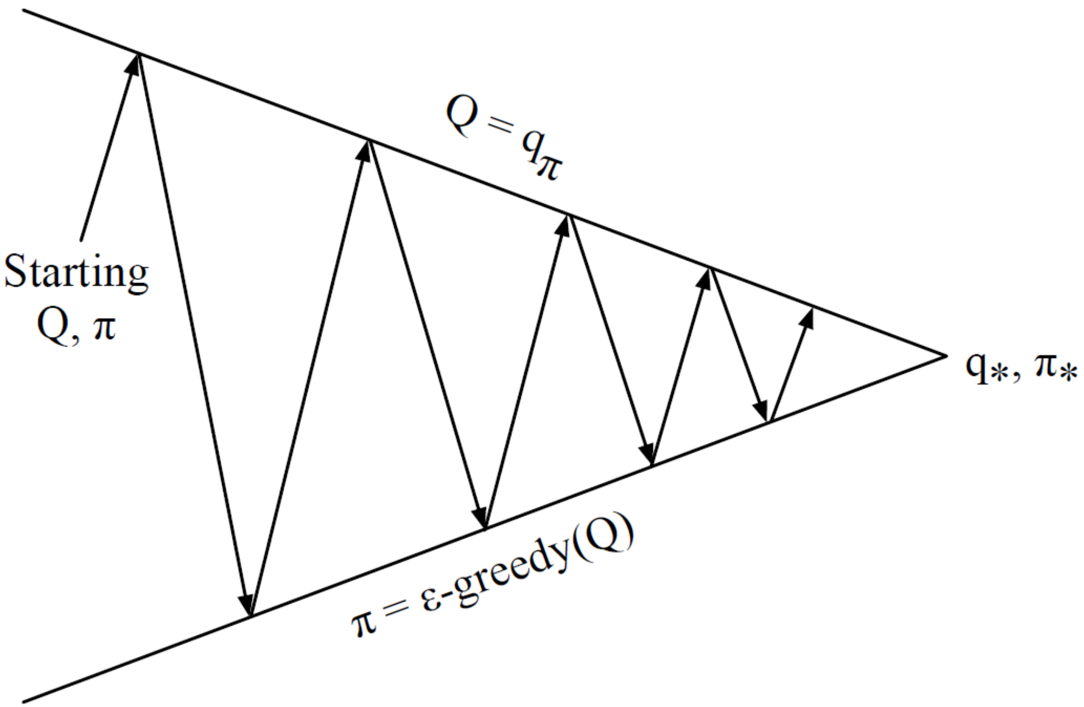
최댓값 >= 기댓값

분모, 분자에 1-epsilon곱해줌.

모든 확률을 구한다. 기댓값

Therefore from policy improvement theorem, $v_{\pi'}(s) \geq v_\pi(s)$

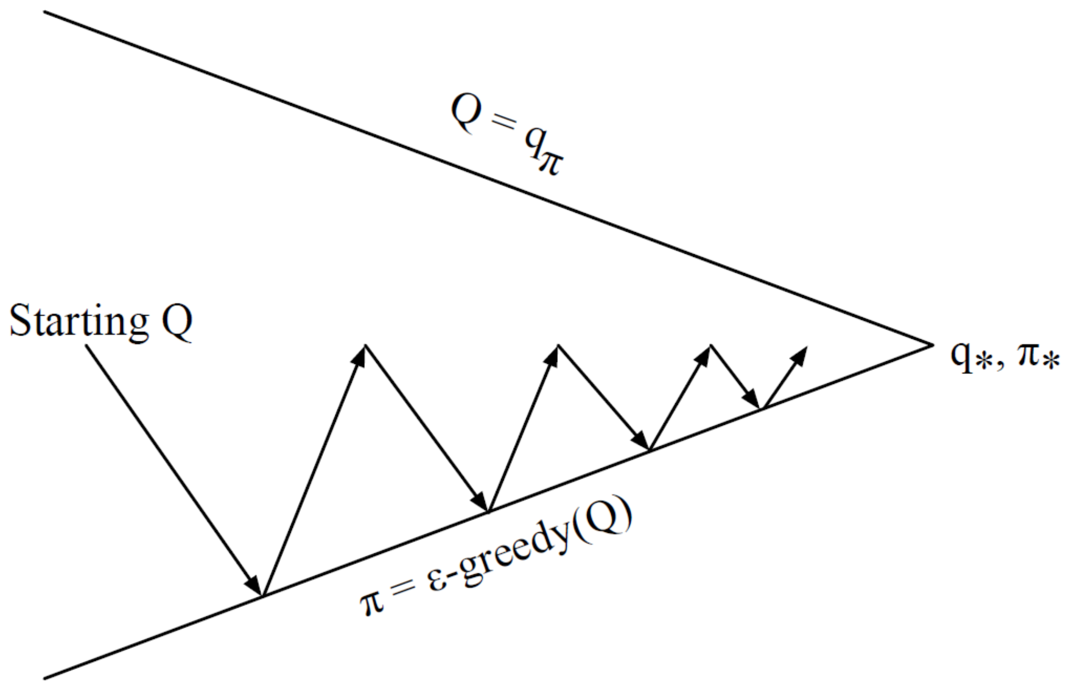
(Model-Free) Monte-Carlo Policy Iteration



Policy evaluation Monte-Carlo policy evaluation, $Q = q_\pi$

Policy improvement ϵ-greedy policy improvement

Variant Monte-Carlo Policy Iteration



Every episode:

Policy evaluation Monte-Carlo policy evaluation, $Q \approx q_\pi$

Policy improvement ϵ-greedy policy improvement

Greedy in the Limit with Infinite Exploration (GLIE)

Definition

Greedy in the Limit with Infinite Exploration (GLIE)

- All state-action pairs are explored infinitely many times,

  모든 state, action 의 pair에 대해서 무한번 돌리면 무한번 방문해야 한다.

$$\lim_{k\to\infty} N_k(s,a) = \infty$$

- The policy converges on a greedy policy,

  개선시키고 있는 policy가 결국은 greedy하게 된다.

$$\lim_{k\to\infty} \pi_k(a|s) = \mathbf{1}(a = \underset{a'\in\mathcal{A}}{\operatorname{argmax}}\ Q_k(s,a'))$$

epsilon값을 1로 놓고 모든 state를 두루두루 보다가 epsilon값을 점점 감소시키면서 greedy하게 만듦.

For example, ϵ-greedy is GLIE if ϵ reduces to zero at $\epsilon_k = \frac{1}{k}$

Monte-Carlo Control

- Sample k^{th} episode using π : {S_1, A_1, R_2, . . . , S_T} ~ π
- For each state S_t and action A_t in the episode,

$$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$$

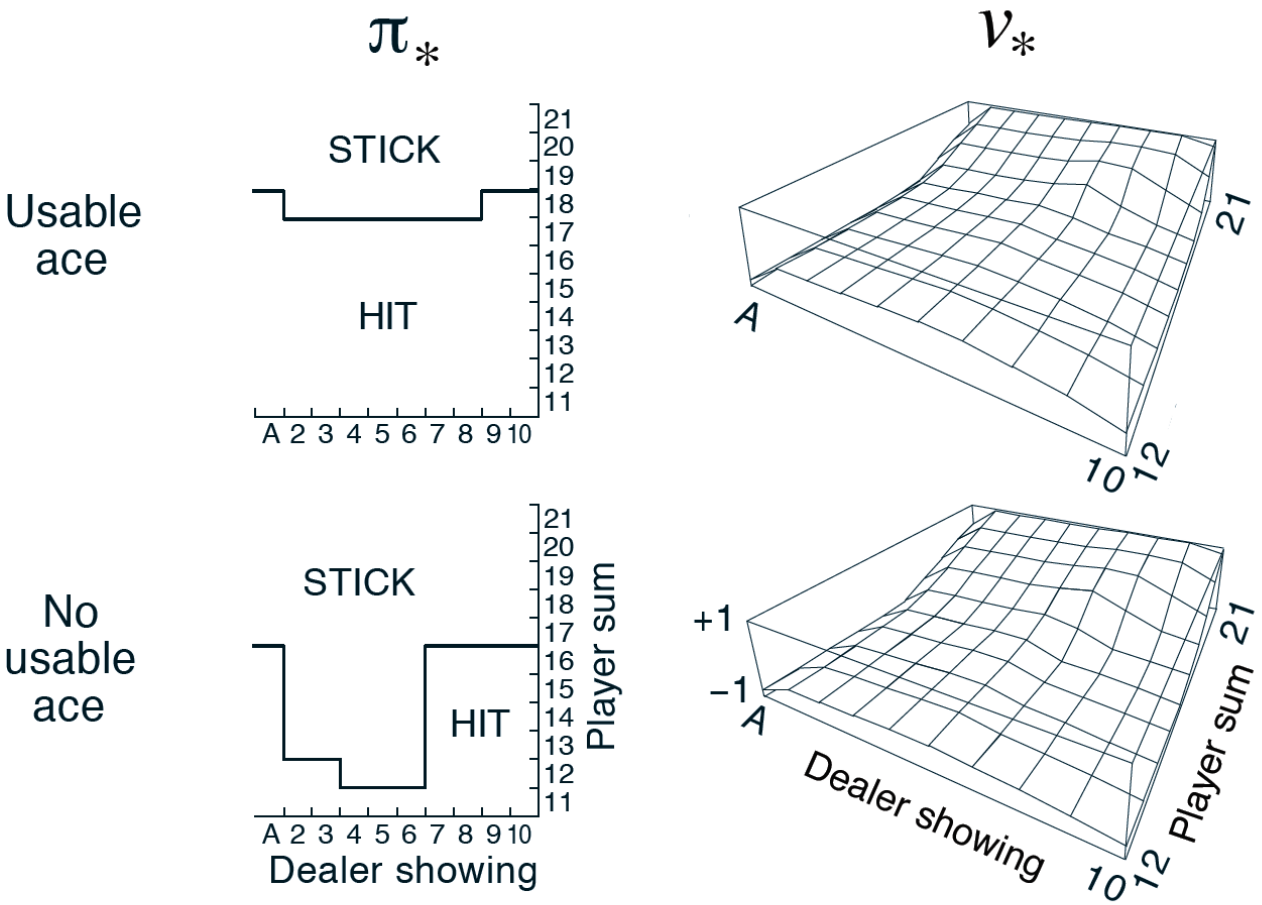
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)}(G_t - Q(S_t, A_t))$$

- Improve policy based on new action-value function

$$\epsilon \leftarrow \frac{1}{k}$$

$$\pi \leftarrow \epsilon - greedy(Q)$$

Monte-Carlo Control in Blackjack



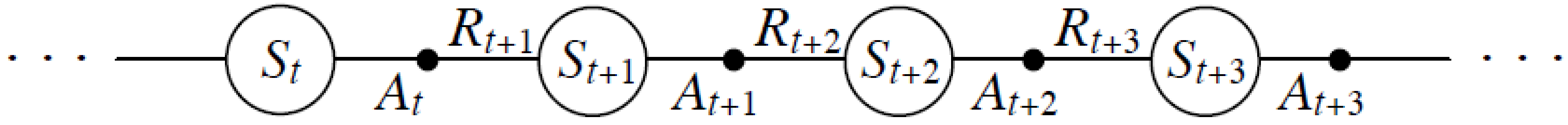
MC vs. TD Control

- TD learning has several advantages over MC learning
  - Lower variance
  - Online
  - Incomplete sequences
- Natural idea: use TD instead of MC
  - Apply TD to $Q(S, A)$
  - Use ϵ -greedy policy improvement
  - Update every time-step

Model-free Policy Iteration with TD Methods

- Use temporal difference methods for policy evaluation step
- Initialize policy π
- Repeat:
  - Policy evaluation: compute Q_π using temporal difference updating with $\epsilon - greedy$ policy
  - Policy improvement: same as Monte Carlo policy improvement, set π to $\epsilon - greedy\ (Q_\pi)$

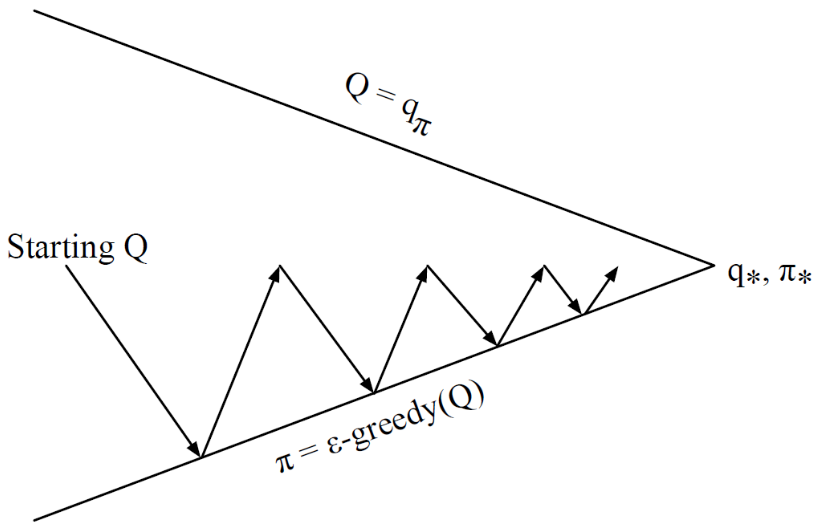
Updating Action-Value Functions with SARSA



$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \underbrace{\Big[\overbrace{R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})}^{\text{TD target}} - Q(S_t, A_t) \Big]}_{\text{TD error}}.$$

Policy Control with SARSA

- Every time-step:
  - policy evaluation SARSA, $Q \approx q_\pi$
  - Policy improvement $\epsilon - greedy$ policy improvement



SARSA Algorithm

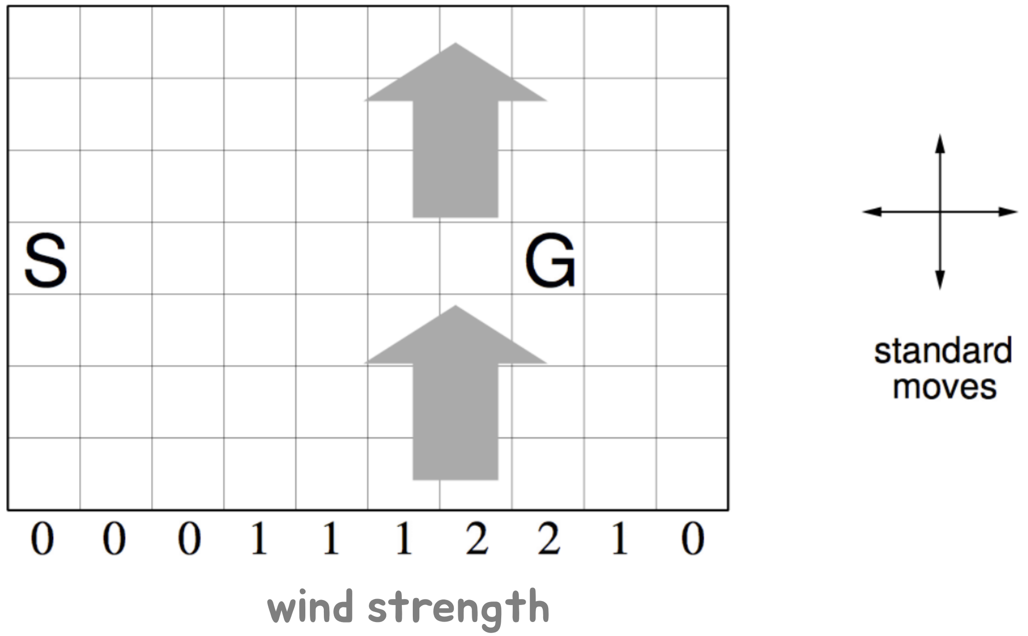
Sarsa (on-policy TD control) for estimating $Q \approx q_*$

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

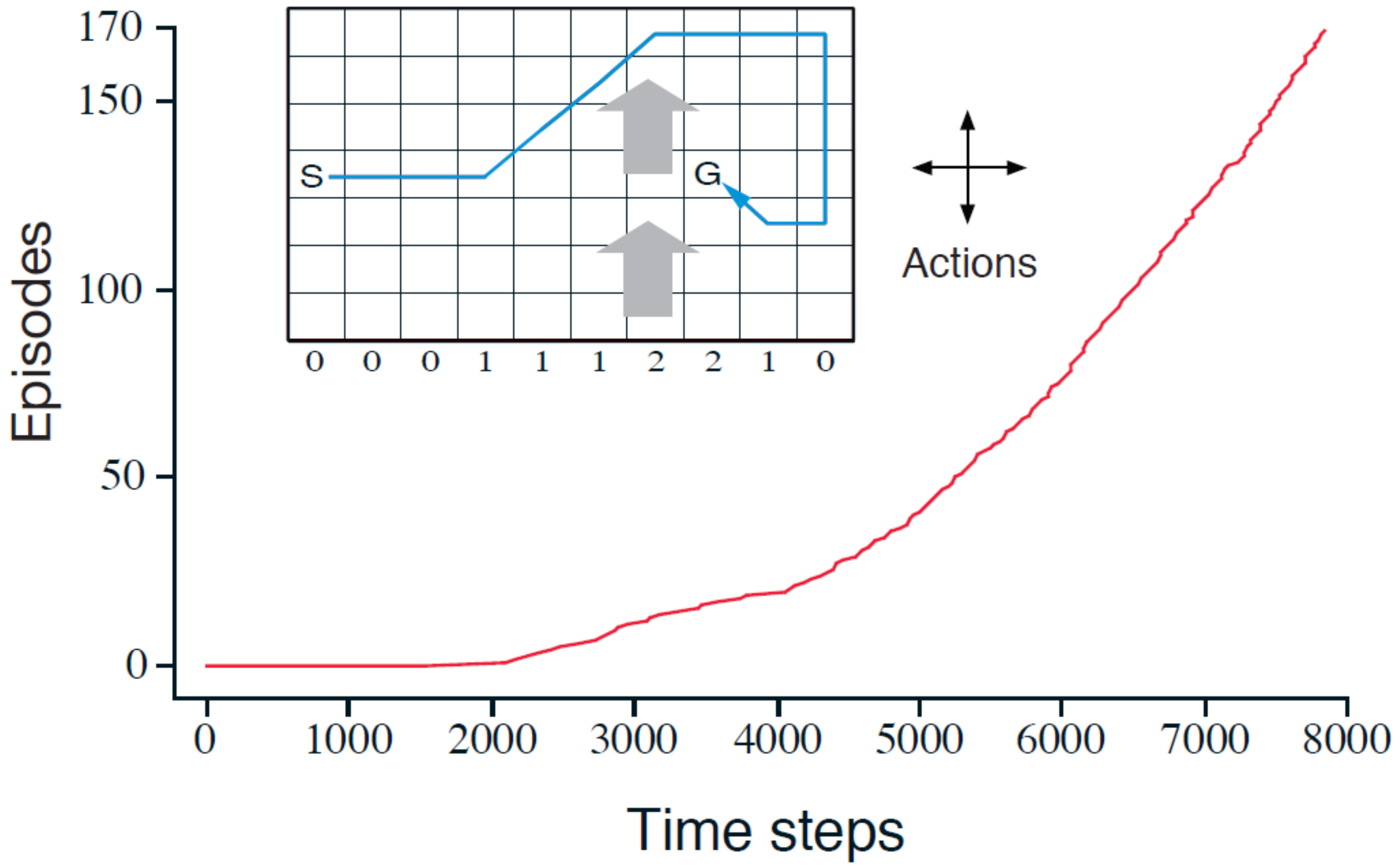
Loop for each episode:
    Initialize S
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Loop for each step of episode:
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ε-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R +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

Windy Gridworld Example

- Reward = -1 per time-step until reaching goal
- $\epsilon = 0.1, \alpha = 0.5$
- Undiscounted



SARSA on Windy Gridworld



one step TD
TD(0)
둘은 거의 동일한 결과를 냄.

n-Step SARSA

- Consider the following n-step returns for $n = 1, 2, ..\infty$:

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

- 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

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

Off-Policy Learning

- Evaluate target policy $\pi(a|s)$ to compute $v_\pi(s)$ or $q_\pi(s,a)$ while following behavior policy $\mu(a|s)$

$$\{S_1, A_1, R_2, \ldots, S_T\} \sim \mu$$

- Why is this important?
  - Learn from observing humans or other agents
  - Re-use experience generated from old policies $\pi_1, \pi_2, \ldots, \pi_{t-1}$
  - Learn about optimal policy while following exploratory policy
- Learn about multiple policies while following one policy

on-policy는 policy를 update하면 그 update된 policy로만 sampling이 가능함.
off는 이전 policy로도 sampling이 가능한가?

Importance Sampling

- Estimate the expectation of a different distribution

$$
\begin{aligned}
\mathbb{E}_{X\sim P}[f(X)] &= \sum P(X)f(X) \\
&= \sum Q(X)\frac{P(X)}{Q(X)}f(X) \\
&= \mathbb{E}_{X\sim Q}\left[\frac{P(X)}{Q(X)}f(X)\right]
\end{aligned}
$$

Off

Importance Sampling for O-Policy Monte-Carlo

- Use returns generated from μ to evaluate π
- Weight return G_t according to similarity between policies
- Multiply importance sampling corrections along whole episode

$$G_t^{\pi/\mu} = \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)} \frac{\pi(A_{t+1}|S_{t+1})}{\mu(A_{t+1}|S_{t+1})} \dots \frac{\pi(A_T|S_T)}{\mu(A_T|S_T)} G_t$$

importance sampling rate을 쭉 곱해주면 현재 policy의 기대치를 구할 수 있음.

- Update value towards corrected return

이론적으로는 맞는 말이지만 쉽게 적용하기 힘듦.
뮤값이 아주 작아지거나 하면 굉장히 큰 값으로 발산할 수도 있음.(variation이 큼.)

$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t^{\pi/\mu} - V(S_t) \right)$$

- Cannot use if μ is zero
- Importance sampling can dramatically increase variance

Importance Sampling for O-Policy TD

- Use TD targets generated from to evaluate π
- Weight TD target $R_{t+1} + \gamma V(S_{t+1})$ by importance sampling
- Only need a single importance sampling correction

$$V(S_t) \leftarrow V(S_t) + \alpha \left(\boxed{\frac{\pi(S_t, A_t)}{\mu(S_t, A_t)}} \left(R_{t+1} + \gamma V(S_{t+1})\right) - V(S_t) \right)$$

- Much lower variance than Monte-Carlo importance sampling

Q-Learning : Off-policy TD Control

- We now consider off-policy learning of action-values $Q(S, A)$
- No importance sampling is required
- Next action is chosen using behavior policy $A_{t+1} \sim \mu(S_t)$
- But we consider alternative successor action $A' \sim \pi(S_t)$
- And update $Q(S_t, A_t)$ towards value of alternative action

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t)\right)$$

Q-Learning : Off-policy TD Control

importance sampling을 사용하지 않고 두가지 policy가 있을 때 q값을 update.

- We now allow both behavior and target policies to improve

- The target policy π is **greedy** w.r.t. $Q(s, a)$

$$\pi(S_t) = \underset{a'}{\operatorname{argmax}}\, Q(S_{t+1}, a')$$

- The behavior policy μ is e.g. $\boldsymbol{\epsilon - greedy}$ w.r.t. $Q(s, a)$

- The Q-learning target then simplifies:

sarsa 같은 경우 실제로 환경과 상호작용해서 값을 얻어오는데
q-learning은 그러지 않고 그냥 갖고 있던 q값 중에 가장 큰 값을 가져옴.

$$R_{t+1} + \gamma Q(S_{t+1}, A') = R_{t+1} + \gamma Q\left(S_{t+1}, \underset{a'}{\operatorname{argmax}}\, Q(S_{t+1}, a')\right)$$

$$= R_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a')$$

value iteration 방식.

Q-Learning Algorithm

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

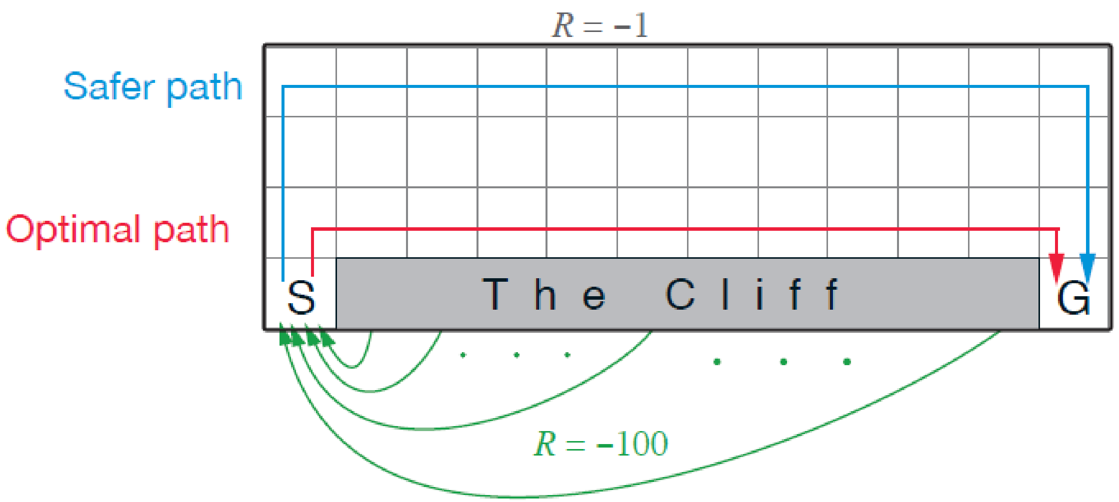
SARSA vs. Q-Learning

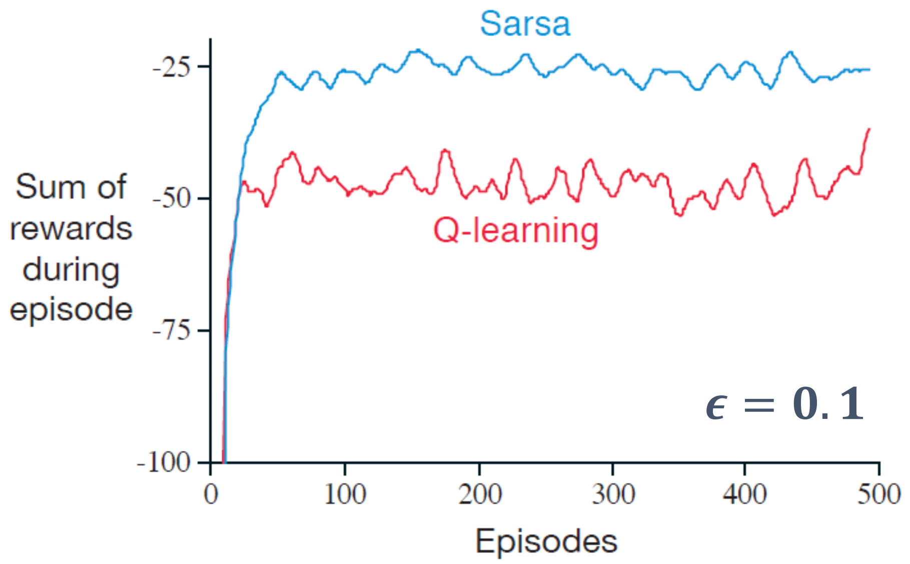
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big]$$

epsilon greedy로 뽑아냄.

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

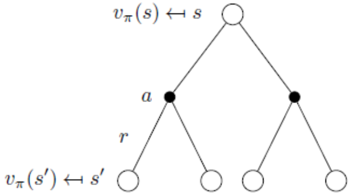
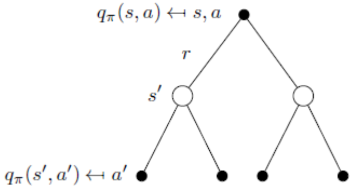
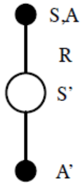
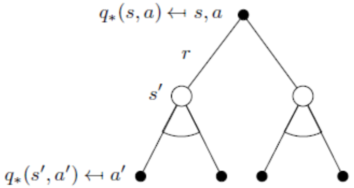
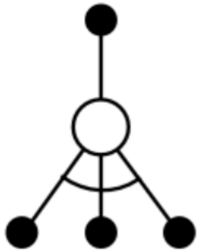
Example: Cliff Walking





Of course, if ϵ were gradually reduced, then both methods would asymptotically converge to the optimal policy

Relationship Between DP and TD (1/2)

		Full Backup (DP)	*Sample Backup (TD)*
	Bellman Expectation Equation for $v_\pi(s)$	 Iterative Policy Evaluation	 TD Learning
policy iteration	Bellman Expectation Equation for $q_\pi(s, a)$	 Q-Policy Iteration	 Sarsa
value iteration	Bellman Optimality Equation for $q_*(s, a)$	 Q-Value Iteration	 Q-Learning

Relationship Between DP and TD (2/2)

알파라는 learning rate가 필요.
sampling한 값은 정확히 믿을 수 없는 값임.

Full Backup (DP)	*Sample Backup (TD)*
Iterative Policy Evaluation	TD Learning
$V(s) \leftarrow \mathbb{E}\left[R + \gamma V(S') \mid s\right]$	$V(S) \xleftarrow{\alpha} R + \gamma V(S')$
Q-Policy Iteration	Sarsa
$Q(s,a) \leftarrow \mathbb{E}\left[R + \gamma Q(S',A') \mid s,a\right]$	$Q(S,A) \xleftarrow{\alpha} R + \gamma Q(S',A')$
Q-Value Iteration	Q-Learning
$Q(s,a) \leftarrow \mathbb{E}\left[R + \gamma \max_{a' \in \mathcal{A}} Q(S',a') \mid s,a\right]$	$Q(S,A) \xleftarrow{\alpha} R + \gamma \max_{a' \in \mathcal{A}} Q(S',a')$

where $x \xleftarrow{\alpha} y \equiv x \leftarrow x + \alpha(y - x)$

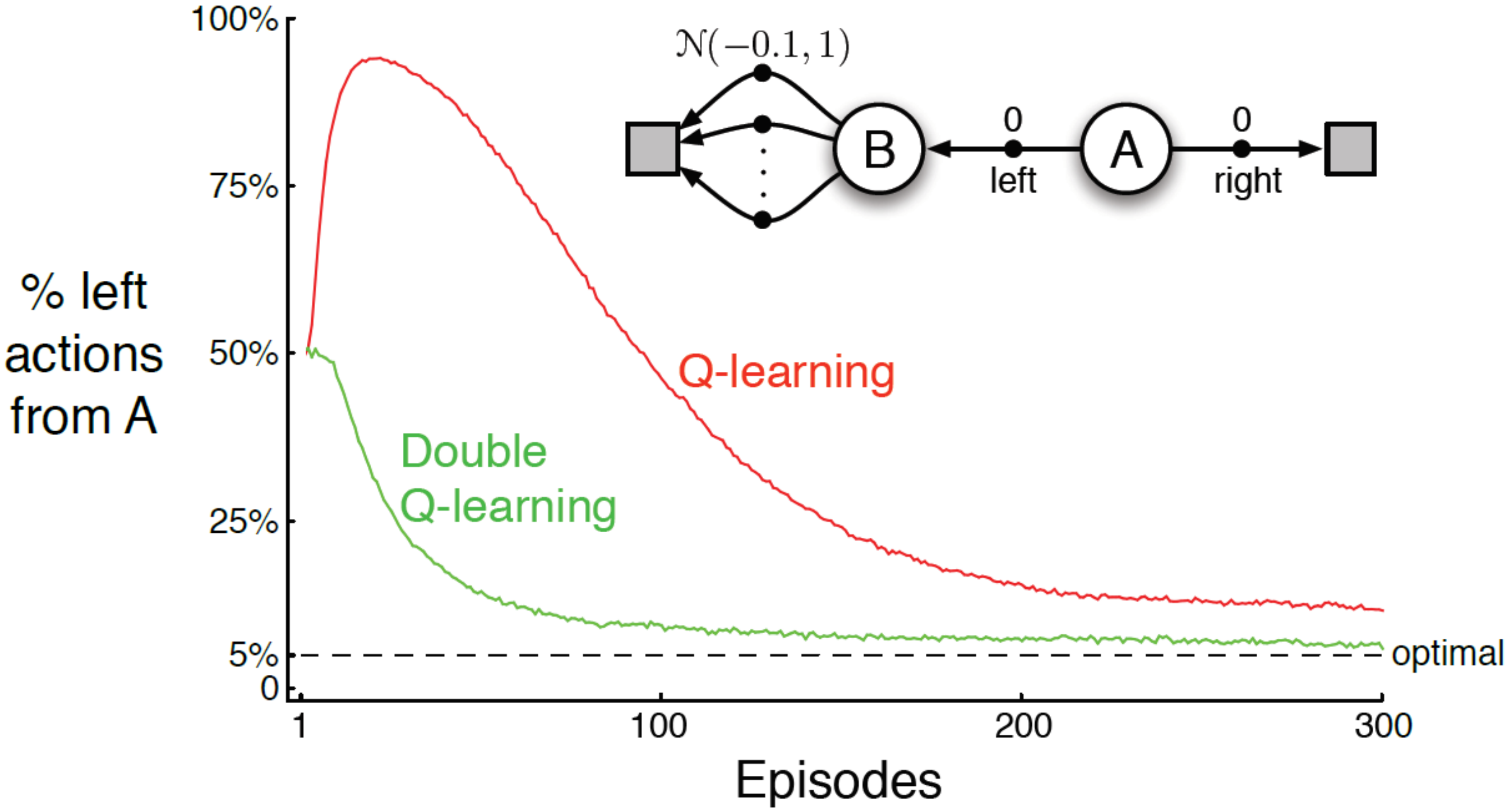
Maximization Bias

- Consider single-state MDP with 2 actions, and both actions have 0-mean random rewards, $\mathbb{E}(r|a = a_1) = \mathbb{E}(r|a = a_2) = 0$.
- Then $\mathrm{Q}(s, a_1) = \mathrm{Q}(s, a_2) = 0$
- Assume there are prior samples of taking action a_1 and a_2
- Let $\widehat{\mathrm{Q}}(s, a_1)$, $\widehat{\mathrm{Q}}(s, a_2)$ be the finite sample estimate of Q
- Let $\hat{\pi} = \underset{a'}{\mathrm{argmax}}\, \widehat{\mathrm{Q}}(s, a)$ be the greedy policy w.r.t. the estimated $\widehat{\mathrm{Q}}$
- The maximum of the estimates would be positive not zero.

Double Learning

- The greedy policy w.r.t. estimated Q values can yield a maximization bias during finite-sample learning
- Due to using Q both to determine the maximizing action and to estimate its value.
- Use two independent unbiased estimates of Q_1 and Q_2
  - Use one estimate to select max action: $a^* = \operatorname*{argmax}_a Q_1(s,a)$
  - Use other estimate to estimate value of a^* : $Q_2(s,a^*)$

Maximization Bias Example



- $\epsilon = 0.1$
- $\alpha = 0.1$
- $\gamma = 1$

Double Learning Algorithm

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

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:

$$Q_1(S,A) \leftarrow Q_1(S,A) + \alpha\Big(R + \gamma Q_2\big(S', \arg\max_a Q_1(S',a)\big) - Q_1(S,A)\Big)$$

        else:

$$Q_2(S,A) \leftarrow Q_2(S,A) + \alpha\Big(R + \gamma Q_1\big(S', \arg\max_a Q_2(S',a)\big) - Q_2(S,A)\Big)$$

        $S \leftarrow S'$
    until S is terminal