Chap.6
Temporal-Difference Learning

TD Method

기존의 Monte Carlo Method

$V(S_t) \leftarrow V(S_t) + \alpha\Big[G_t - V(S_t)\Big]$ →G_t를 얻기 위해 한 episode가 끝날 때 까지 대기

Temporal Difference Method (=TD(0))

$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big]$ →t+1 까지만 기다리면 적용 가능

∴ G_t를 Better Estimate $(R_{t+1} + \gamma V(S_{t+1}))$ 로 대체

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

TD Method

$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] && (6.3)\\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] && (\text{from } (3.9))\\
&= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]\,. && (6.4)
\end{aligned}
$$

1. Monte Carlo Method는 (6.3)의 estimate를 target
   -> Expected Value를 모르기 때문

2. DP Method는 (6.4)의 estimate를 target
   -> Expected Value는 알고 있음(환경 모델의 완벽한 제공)
   -> $v_\pi(S_{t+1})$를 모르기 때문에 $V(S_{t+1})$ 로 대체하여 이용하기 때문임.

3. TD도 estimate를 target
   -> (6.4)의 expected value를 sampling 하여 current estimate로 이용.

TD Errors

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

○ Estimate $V(S_t)$ 와, better estimate($R_{t+1} + \gamma V(S_{t+1})$)의 차

○ Monte Carlo Error는 TD Error의 합으로 표현 가능함.
(단, 중간에 V가 update 되지 않을 경우에만 성립)

$$\begin{aligned}
G_t - V(S_t) &= R_{t+1} + \gamma G_{t+1} - V(S_t) + \gamma V(S_{t+1}) - \gamma V(S_{t+1}) && \text{(from (3.9))} \\
&= \delta_t + \gamma\big(G_{t+1} - V(S_{t+1})\big) \\
&= \delta_t + \gamma\delta_{t+1} + \gamma^2\big(G_{t+2} - V(S_{t+2})\big) \\
&= \delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2} + \cdots + \gamma^{T-t-1}\delta_{T-1} + \gamma^{T-t}\big(G_T - V(S_T)\big) \\
&= \delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2} + \cdots + \gamma^{T-t-1}\delta_{T-1} + \gamma^{T-t}\big(0 - 0\big) \\
&= \sum_{k=t}^{T-1} \gamma^{k-t}\delta_k. && (6.6)
\end{aligned}$$

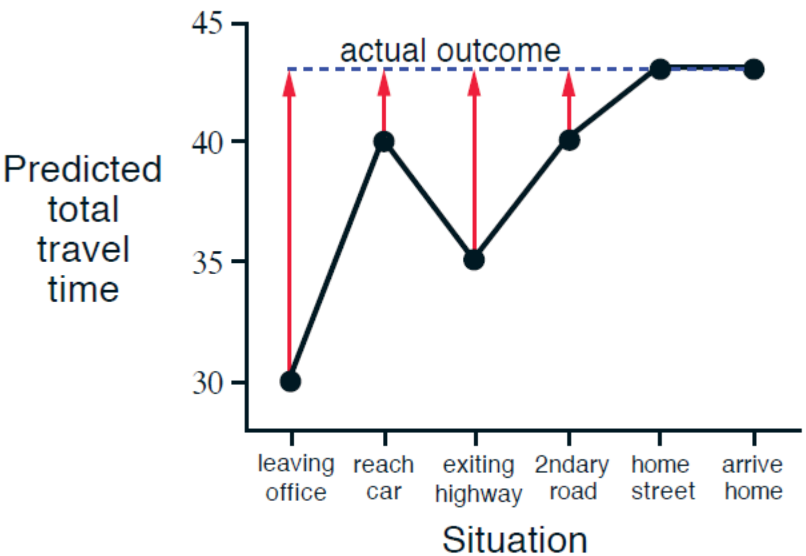
집으로 가는 길

Q. 퇴근을 하면서 도착 예정 시간을 갱신한다.

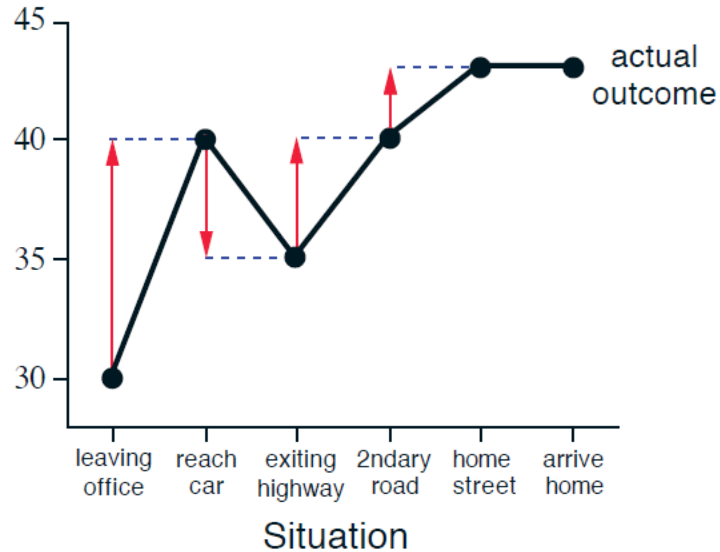
State	*Elapsed Time (minutes)*	*Predicted Time to Go*	*Predicted Total Time*
leaving office, friday at 6	0	30	30
reach car, raining	5	35	40
exiting highway	20	15	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43

Reward : Elapsed Time ($\gamma = 1$, not discounting)
Value of state : Predicted time to go

집으로 가는 길



Monte Carlo Method



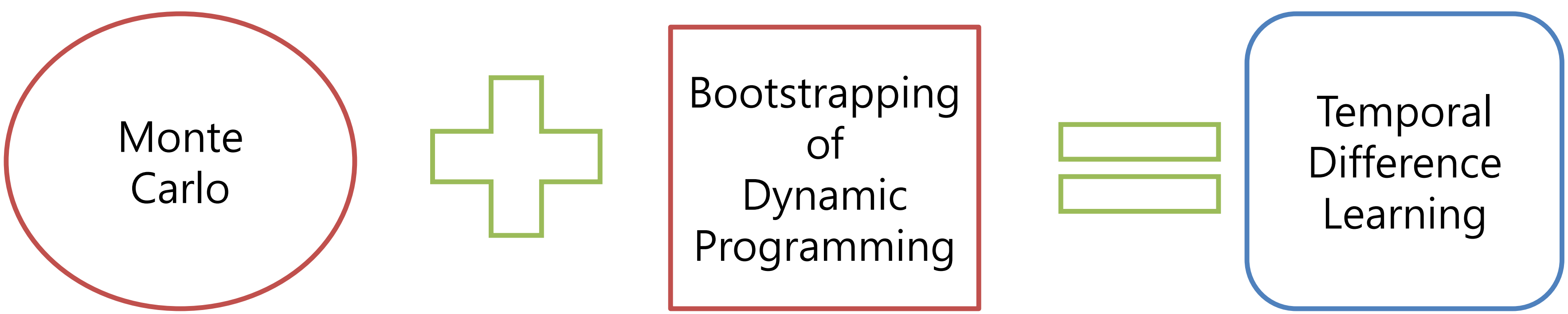
TD Method

TD Method의 이점

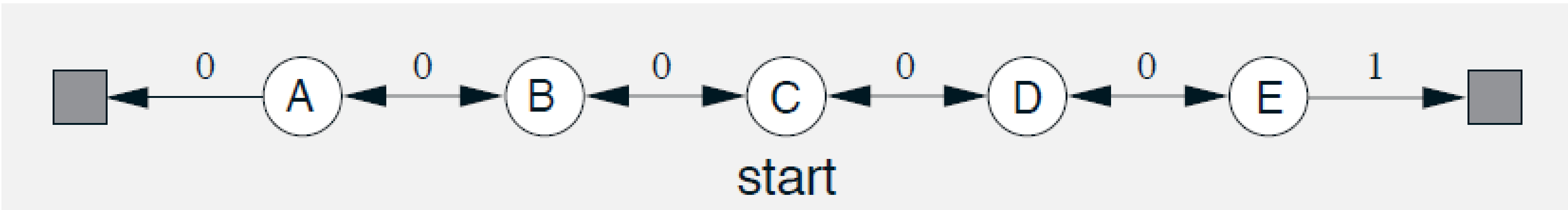
○ DP와 Monte Carlo Method의 이점을 가짐.

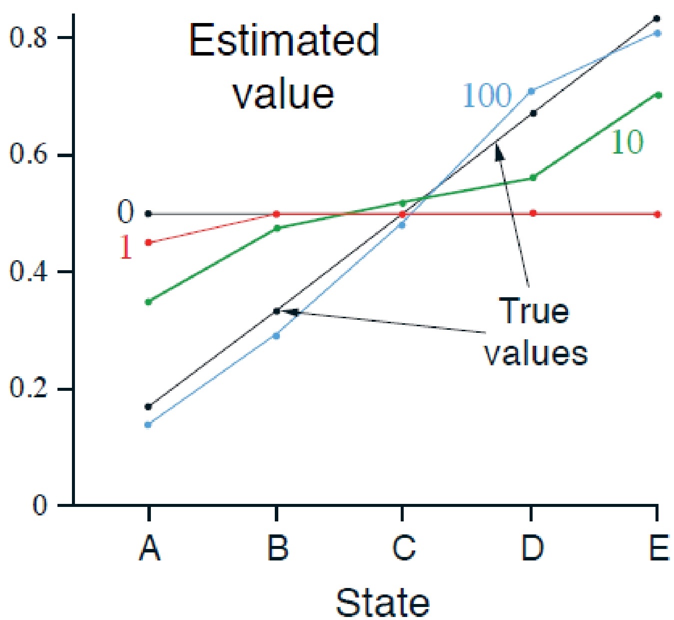
1. update를 위한 환경 모델이 필요하지 않음 (Monte Carlo의 이점)

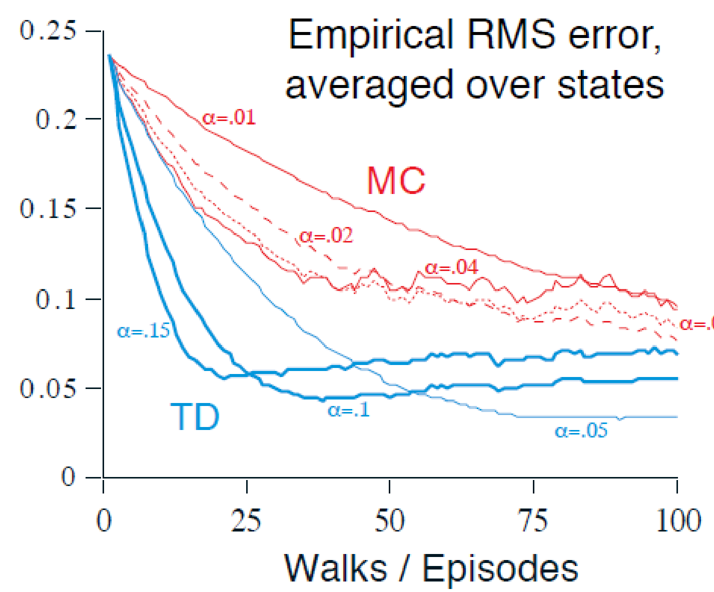
2. Bootstrapping 을 이용하기 때문에 incomplete episode에서 학습 가능
→ Monte Carlo Method와 달리 한 time step 만 대기하면 됨. (learning time 감소)



Random Walk


0
A
0
B
0
C
0
D
0
E
1
start


Estimated value
0.8
0.6
0.4
0.2
0
0
1
10
100
True values
A
B
C
D
E
State


Empirical RMS error, averaged over states
0.25
0.2
0.15
0.1
0.05
0
α=.01
MC
α=.02
α=.04
α=.03
α=.15
TD
α=.1
α=.05
0
25
50
75
100
Walks / Episodes

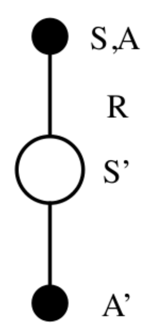
On Policy TD Control(Sarsa)

○ Sarsa (State&Action, Reward, State&Action)

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

○ t+1번째 state-action pair와 reward를 이용해
t번째 state-action pair의 Q function을 개선

○ 각 state의 transition마다 update가 이루어지게 된다.
(다음 state가 terminal state일 경우 $Q(S_{t+1}, A_{t+1}) = 0$ 으로 놓고 update)



$$Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma Q(S',A') - Q(S,A)\right)$$

On Policy TD Control(Sarsa)

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

Off Policy TD Control(Q-Learning)

Sarsa

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

Q-Learning

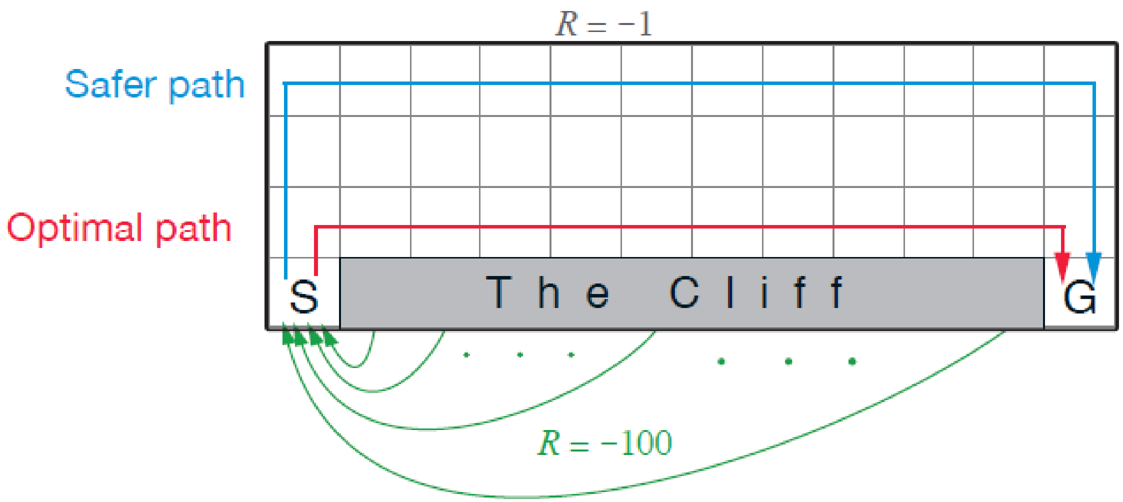
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big].$$

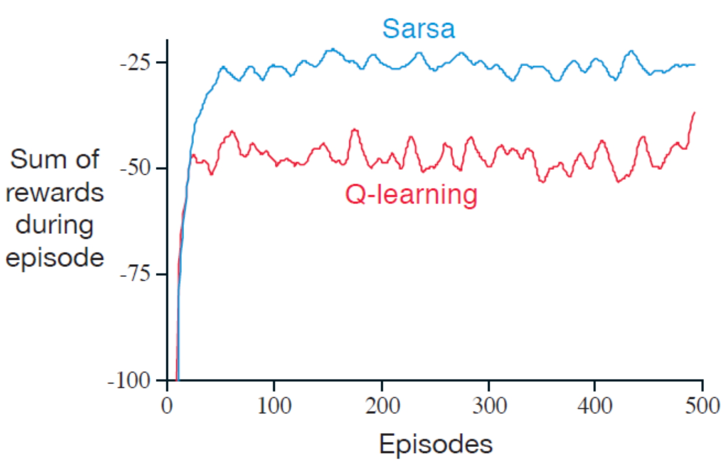
Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

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

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

Cliff Walking Problem





○ 두 경우 모두 $\varepsilon - greedy\ policy$로 접근 $(\varepsilon = 0.1)$

○ Sarsa의 경우 해당 policy를 통해 action을 취함.
→ Cliff 주변의 value를 낮게 잡아 접근 x

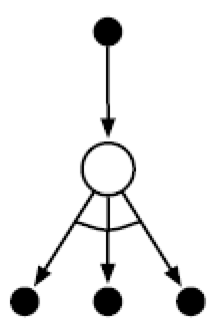
○ Q-Learning의 경우 해당 policy를 통해 학습
→ Q func. 를 통해 cliff로 향하는 action을 지양하도록 학습.

Expected Sarsa

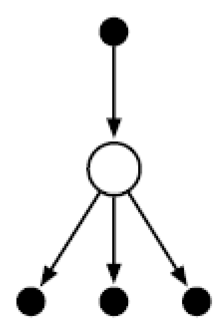
$$
\begin{aligned}
Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma\mathbb{E}_\pi[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t)\Big] \\
&\leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma\sum_a \pi(a|S_{t+1})Q(S_{t+1}, a) - Q(S_t, A_t)\Big],
\end{aligned}
$$

○ 기대값을 이용하여 TD error를 계산하는 방식
→ Sarsa에서 분산을 제거(Random Selection을 하지 않음)하여
동일한 양의 experience 하에서 Sarsa 보다 약간 더 나은 성능을 보임.

○ Expected Sarsa는 Off-Policy

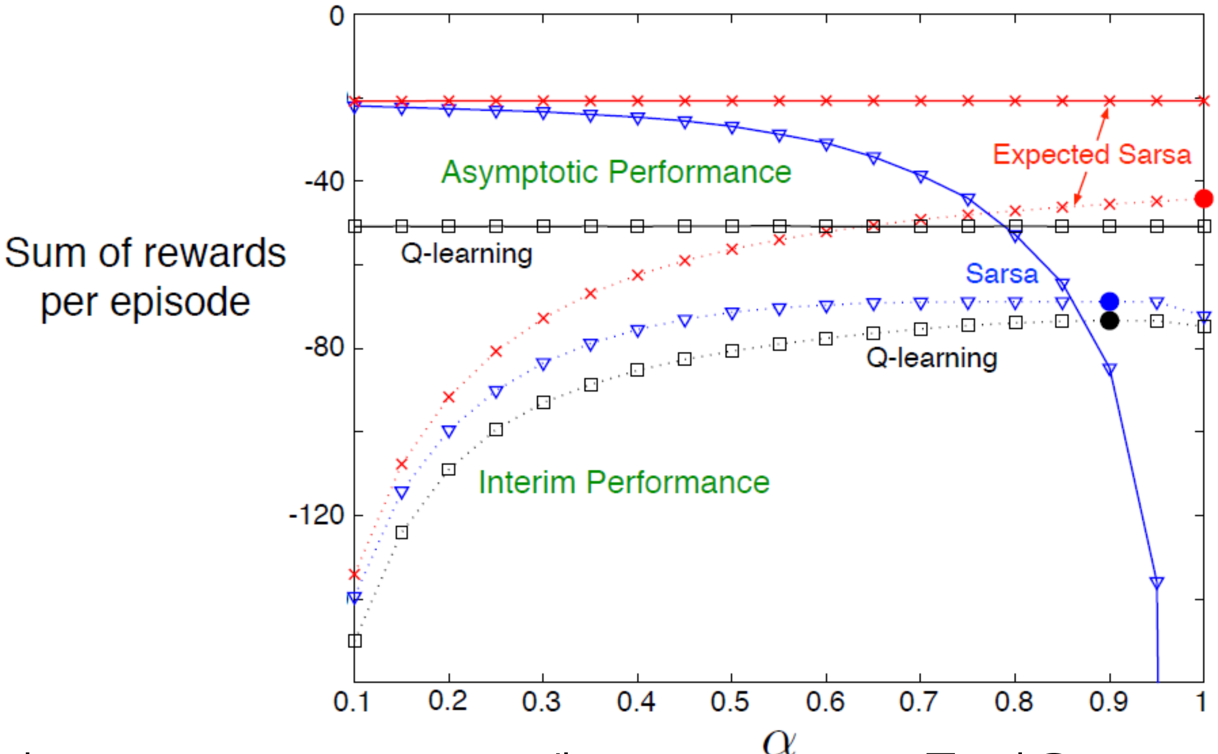


Q-learning



Expected Sarsa

Expected Sarsa



앞의 Cliff-Walking Problem에 Expected Sarsa를 적용

Interim Performance : 100개 episode에 대한 평균
Asymptotic Performance : 100000개 이상 episode에 대한 평균

Double Q-learning

○ Q-learning에는 maximization Operation이 포함되어 있음

-> 학습 초기에 높은 값으로 편중되는 경향(maximization Bias)이 발생하게 됨
-> 하나의 sample로 maximizing action을 선택하고 action value도 구하는 것이 원인

그럼 maximizing action 선택 함수와
Action value 계산 함수를 독립적으로 두면 어떨까?

$$A^* = \arg\max_a Q_1(a)$$

$$Q_2(A^*) = Q_2(\arg\max_a Q_1(a))$$

$$\mathbb{E}[Q_2(A^*)] = q(A^*)$$

Double Q-learning

Double Q-learning

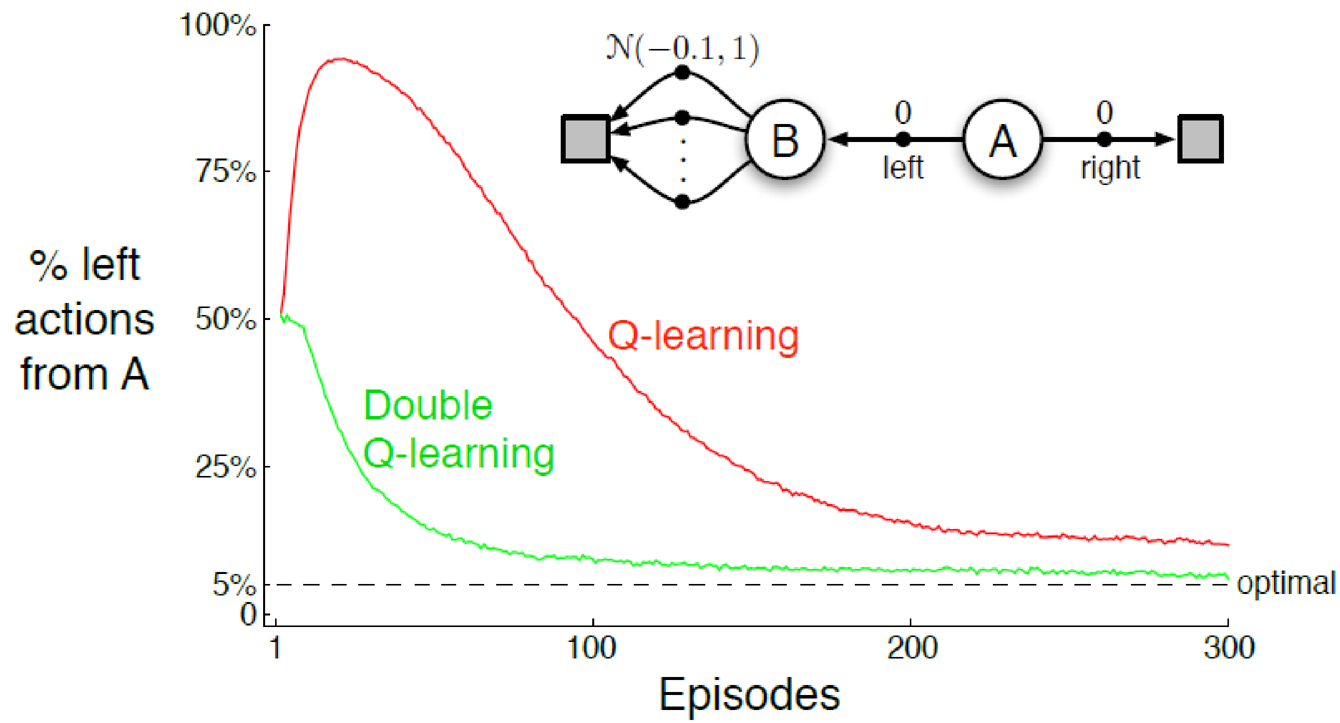
$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q_2\big(S_{t+1}, \arg\max_a Q_1(S_{t+1}, a)\big) - Q_1(S_t, A_t)\Big]$$

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

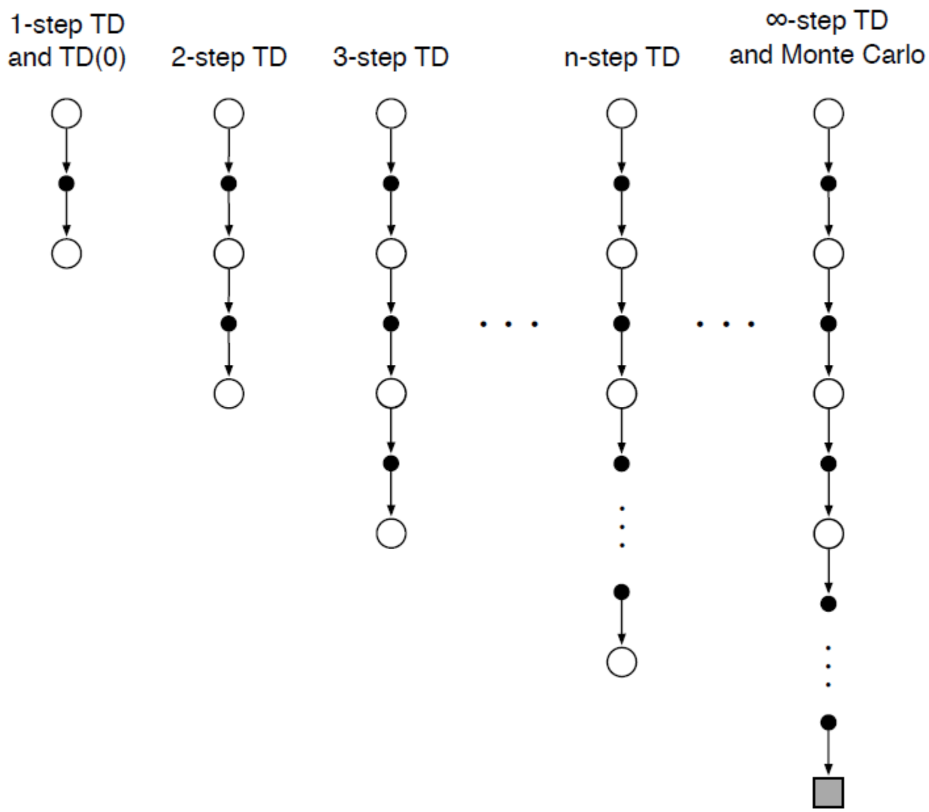
Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using the policy ε-greedy in $Q_1 + Q_2$
    Take action A, observe R, S'
    With 0.5 probabilility:
      $Q_1(S, A) \leftarrow Q_1(S, A) + \alpha\Big(R + \gamma Q_2\big(S', \arg\max_a Q_1(S', a)\big) - Q_1(S, A)\Big)$
    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

Double Q-learning



Chap.7
n-step bootstrapping

n-step TD Prediction



1-step TD : next reward 와 next state의 estimate를 기반으로 update
N-step TD : 다음 n개의 reward와 n번째 state의 estimate를 기반으로 update

n-step TD Prediction

N-step return

$$G_{t:t+1} \doteq R_{t+1} + \gamma V_t(S_{t+1})$$
$$G_{t:t+2} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2}),$$
$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$$

N-step TD의 Value Function 개선

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha\left[G_{t:t+n} - V_{t+n-1}(S_t)\right], \qquad 0 \leq t < T$$

n-step TD for estimating $V \approx v_\pi$

Input: a policy π
Algorithm parameters: step size $\alpha \in (0, 1]$, a positive integer n
Initialize $V(s)$ arbitrarily, for all $s \in \mathcal{S}$
All store and access operations (for S_t and R_t) can take their index mod $n+1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take an action according to $\pi(\cdot|S_t)$
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | If S_{t+1} is terminal, then $T \leftarrow t+1$
  | $\tau \leftarrow t - n + 1$ (τ is the time whose state's estimate is being updated)
  | If $\tau \geq 0$:
  | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n, T)} \gamma^{i-\tau-1} R_i$
  | If $\tau + n < T$, then: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $V(S_\tau) \leftarrow V(S_\tau) + \alpha\left[G - V(S_\tau)\right]$
  Until $\tau = T - 1$

n-step return (VS) $V_t(s)$

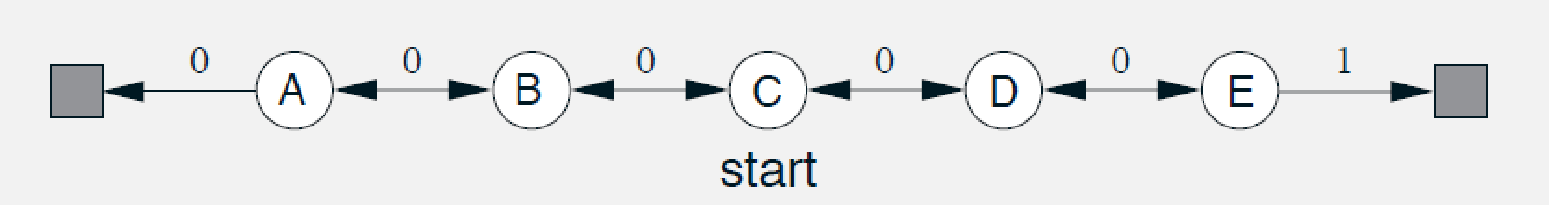
Error Reduction Property

$$\max_s \left| \mathbb{E}_\pi[G_{t:t+n} | S_t = s] - v_\pi(s) \right| \leq \gamma^n \max_s \left| V_{t+n-1}(s) - v_\pi(s) \right|$$

n-step return의 worst error는 $V_{t+n-1}(s)$의 worst error의 γ^n배 보다 작다.

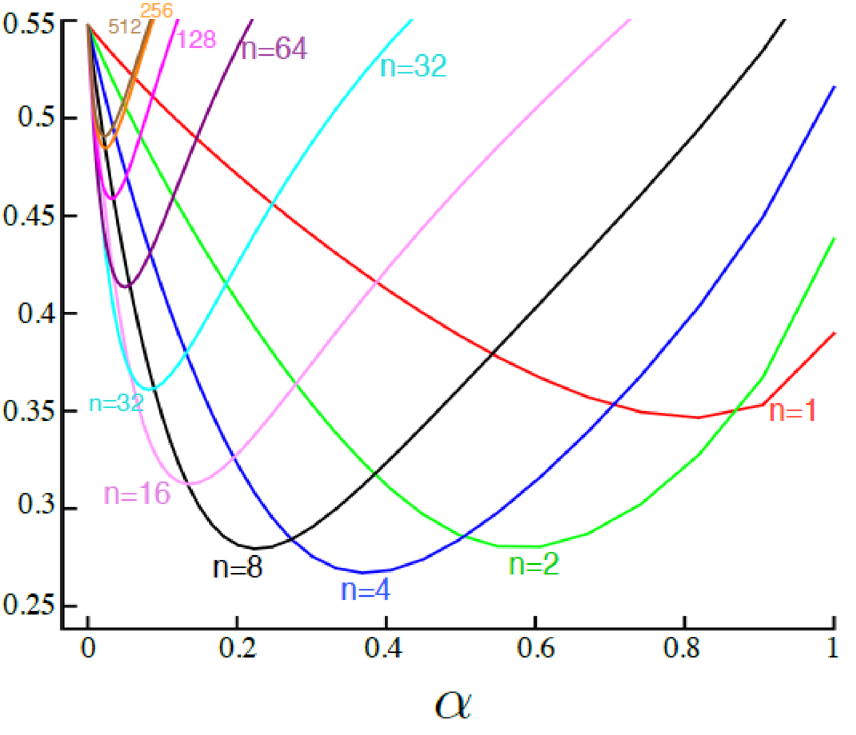
∴ n-step return을 이용하는 것이 좀 더 optimal v(s)에 근접한다.

Random Walk에 n-step TD 적용



△ 5 states

Average RMS error over 19 states and first 10 episodes



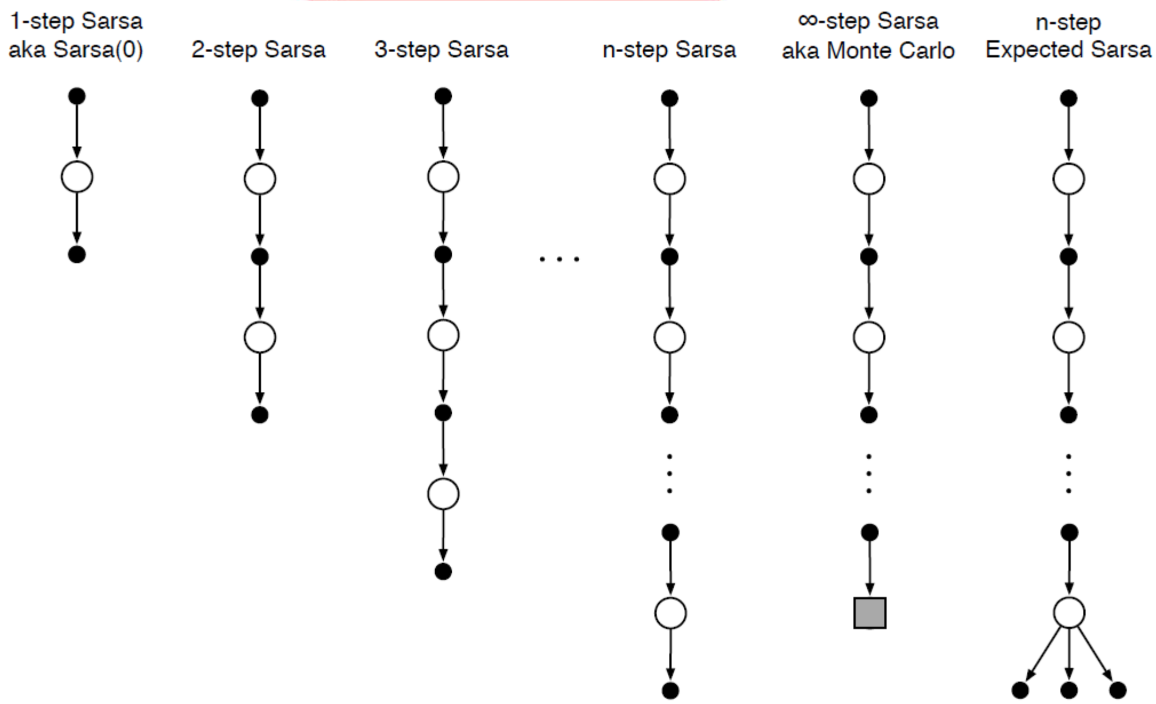
n-step Sarsa

○ Sarsa의 n-step 적용 버전

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}), \quad n \ge 1, 0 \le t < T-n$$

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha \left[G_{t:t+n} - Q_{t+n-1}(S_t, A_t) \right]$$

$$G_t^{(n)} \doteq R_{t+1} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \boxed{\sum_a \pi(a|S_{t+n}) Q_{t+n-1}(S_{t+n}, a),} \quad n \ge 1, 0 \le t \le T-n.$$ (expected Sarsa)



n-step Sarsa

n-step Sarsa for estimating $Q \approx q_*$ or q_π

Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be ε-greedy with respect to Q, or to a fixed given policy
Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$, a positive integer n
All store and access operations (for S_t, A_t, and R_t) can take their index mod $n+1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  Select and store an action $A_0 \sim \pi(\cdot|S_0)$
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take action A_t
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | If S_{t+1} is terminal, then:
  | $T \leftarrow t+1$
  | else:
  | Select and store an action $A_{t+1} \sim \pi(\cdot|S_{t+1})$
  | $\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)
  | If $\tau \geq 0$:
  | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
  | If $\tau + n < T$, then $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\,[G - Q(S_\tau, A_\tau)]$
  | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is ε-greedy wrt Q
  Until $\tau = T - 1$

n-step Off Policy Learning with Importance Sampling

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha\, \rho_{t:t+n-1} \left[G_{t:t+n} - V_{t+n-1}(S_t) \right], \qquad 0 \le t < T$$

b policy에 대한 return

$$\rho_{t:h} \doteq \prod_{k=t}^{\min(h,T-1)} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}.$$

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha\, \rho_{t+1:t+n} \left[G_{t:t+n} - Q_{t+n-1}(S_t, A_t) \right]$$

Off-policy n-step Sarsa for estimating $Q \approx q_*$ or q_π

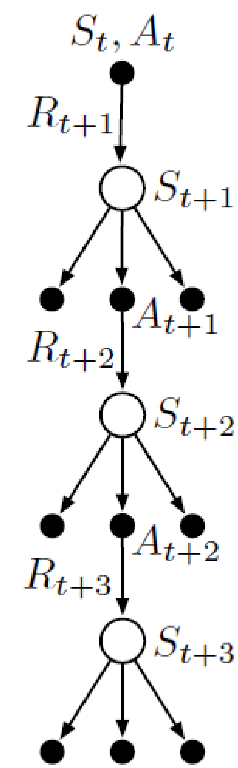
Input: an arbitrary behavior policy b such that $b(a|s) > 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be greedy with respect to Q, or as a fixed given policy
Algorithm parameters: step size $\alpha \in (0,1]$, a positive integer n
All store and access operations (for S_t, A_t, and R_t) can take their index mod $n+1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  Select and store an action $A_0 \sim b(\cdot|S_0)$
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \dots$:
  | If $t < T$, then:
  | Take action A_t
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | If S_{t+1} is terminal, then:
  | $T \leftarrow t+1$
  | else:
  | Select and store an action $A_{t+1} \sim b(\cdot|S_{t+1})$
  | $\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)
  | If $\tau \ge 0$:
  | $\rho \leftarrow \prod_{i=\tau+1}^{\min(\tau+n-1,T-1)} \frac{\pi(A_i|S_i)}{b(A_i|S_i)}$ $(\rho_{\tau+1:t+n-1})$
  | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
  | If $\tau + n < T$, then: $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\rho\left[G - Q(S_\tau, A_\tau)\right]$
  | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt Q
  Until $\tau = T - 1$

Off-Policy Learning Without Importance Sampling

※ Tree-Backup Algorithm

- ○ 선택되지 않은 Action에 대해선 Sample Data가 존재하지 않음.
  → Sampling 이 아닌 Expectation을 이용.



the 3-step
tree-backup
update

Tree-Backup Algorithm

1-step

$$G_{t:t+1} \doteq R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q_t(S_{t+1}, a),$$

(Expected Sarsa와 동일)

2-step

$$\begin{aligned} G_{t:t+2} &\doteq R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{t+1}(S_{t+1}, a) \\ &\quad + \gamma \pi(A_{t+1}|S_{t+1}) \Big(R_{t+2} + \gamma \sum_a \pi(a|S_{t+2}) Q_{t+1}(S_{t+2}, a) \Big) \\ &= R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{t+1}(S_{t+1}, a) + \gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:t+2}, \end{aligned}$$

n-step

$$G_{t:t+n} \doteq R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{t+n-1}(S_{t+1}, a) \; + \; \gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:t+n}$$

n-step written as a sum of TD errors

$$G_{t:t+n} = Q(S_t, A_t) + \sum_{k=t}^{\min(t+n-1, T-1)} \delta_k \prod_{i=t+1}^{k} \gamma \pi(A_i|S_i),$$

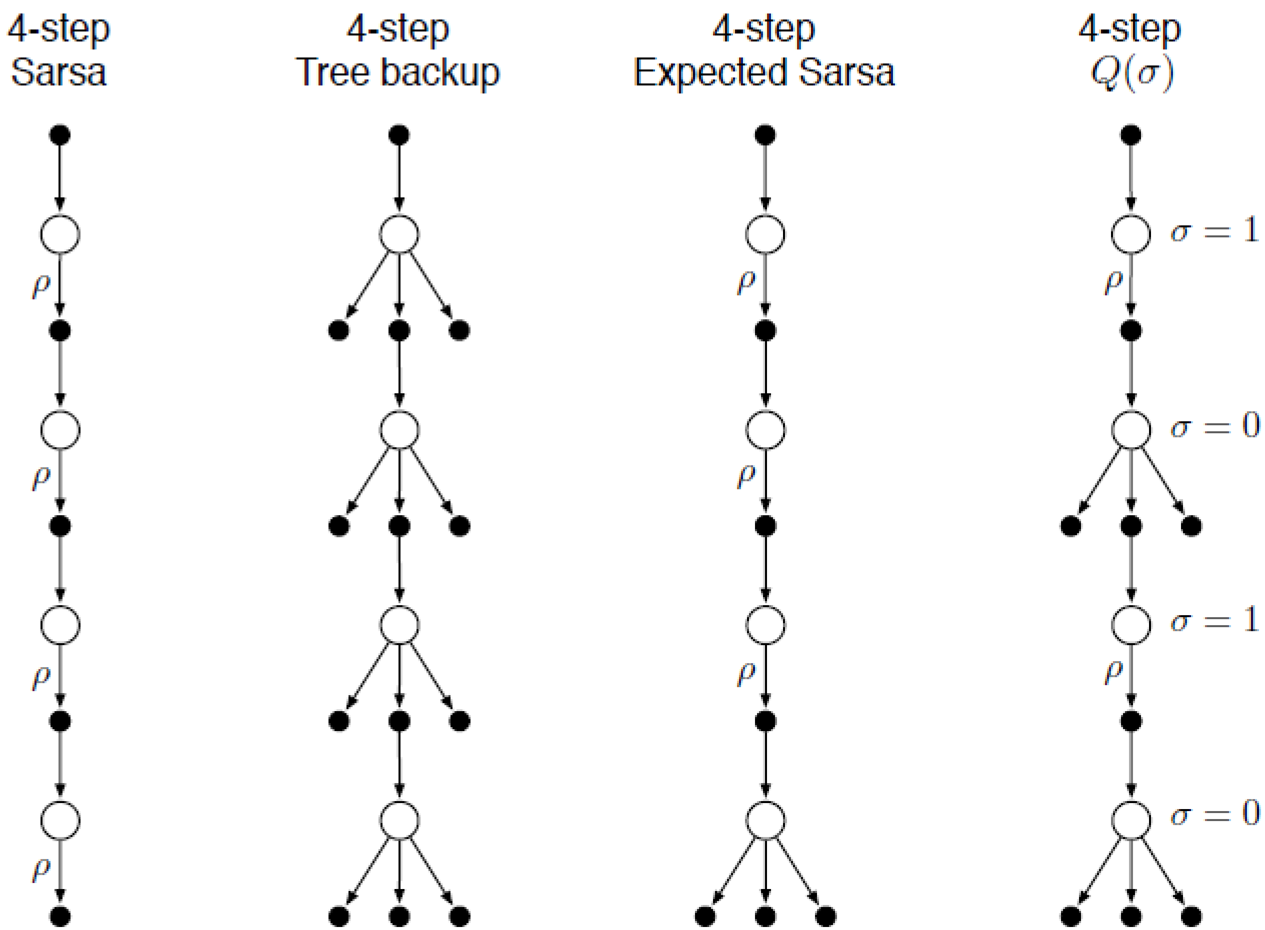
Tree-Backup Algorithm

n-step Tree Backup for estimating $Q \approx q_*$ or q_π

Initialize $Q(s, a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be greedy with respect to Q, or as a fixed given policy
Algorithm parameters: step size $\alpha \in (0, 1]$, a positive integer n
All store and access operations can take their index mod $n+1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  Choose an action A_0 arbitrarily as a function of S_0; Store A_0
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$:
  | Take action A_t; observe and store the next reward and state as R_{t+1}, S_{t+1}
  | If S_{t+1} is terminal:
  | $T \leftarrow t + 1$
  | else:
  | Choose an action A_{t+1} arbitrarily as a function of S_{t+1}; Store A_{t+1}
  | $\tau \leftarrow t + 1 - n$ (τ is the time whose estimate is being updated)
  | If $\tau \geq 0$:
  | If $t + 1 \geq T$:
  | $G \leftarrow R_T$
  | else
  | $G \leftarrow R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a)$
  | Loop for $k = \min(t, T-1)$ down through $\tau + 1$:
  | $G \leftarrow R_k + \gamma \sum_{a \neq A_k} \pi(a|S_k) Q(S_k, a) + \gamma \pi(A_k|S_k) G$
  | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha \left[G - Q(S_\tau, A_\tau)\right]$
  | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt Q
  Until $\tau = T - 1$

n-step $Q(\sigma)$



n-step $Q(\sigma)$

○ σ의 값에 따라서 expectation(tree)을 할 지, sampling(TD)을 할 지 결정.
  σ=0 일 때 sampling 없이 expectation
  σ=1 일 때 하나의 (state, action, state-action 등..)을 sampling

○ ρ의 값에 따라서 on/off policy 여부를 결정.

$$G_{t:h} \doteq R_{t+1} + \gamma\Big(\sigma_{t+1}\rho_{t+1} + (1-\sigma_{t+1})\pi(A_{t+1}|S_{t+1})\Big)\Big(G_{t+1:h} - Q_{h-1}(S_{t+1}, A_{t+1})\Big) + \gamma\bar{V}_{h-1}(S_{t+1}), \tag{7.17}$$

∴ 자유롭게 옵션 설정이 가능한 알고리즘

Off-policy n-step $Q(\sigma)$ for estimating $Q \approx q_*$ or q_π

Input: an arbitrary behavior policy b such that $b(a|s) > 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be ε-greedy with respect to Q, or as a fixed given policy
Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$, a positive integer n
All store and access operations can take their index mod $n+1$

Loop for each episode:
    Initialize and store $S_0 \neq$ terminal
    Choose and store an action $A_0 \sim b(\cdot|S_0)$
    $T \leftarrow \infty$
    Loop for $t = 0, 1, 2, \ldots$:
        If $t < T$:
            Take action A_t; observe and store the next reward and state as R_{t+1}, S_{t+1}
            If S_{t+1} is terminal:
                $T \leftarrow t+1$
            else:
                Choose and store an action $A_{t+1} \sim b(\cdot|S_{t+1})$
                Select and store σ_{t+1}
                Store $\frac{\pi(A_{t+1}|S_{t+1})}{b(A_{t+1}|S_{t+1})}$ as ρ_{t+1}
        $\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)
        If $\tau \geq 0$:
            $G \leftarrow 0$:
            Loop for $k = \min(t+1, T)$ down through $\tau + 1$:
                if $k = T$:
                    $G \leftarrow R_T$
                else:
                    $\bar{V} \leftarrow \sum_a \pi(a|S_k) Q(S_k, a)$
                    $G \leftarrow R_k + \gamma\big(\sigma_k \rho_k + (1 - \sigma_k)\pi(A_k|S_k)\big)\big(G - Q(S_k, A_k)\big) + \gamma \bar{V}$
            $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\left[G - Q(S_\tau, A_\tau)\right]$
            If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt Q
    Until $\tau = T - 1$