

Chapter 6

# Temporal-Difference Learning

2018/11/13

Dev3B 강준하

# Index

6.1 TD Prediction

6.2 Advantages of TD Prediction

6.3 Optimality of TD(0)

6.4 Sarsa : On-Policy TD Control

6.5 Q-learning : Off-Policy TD Control

6.6 Expected Sarsa

6.7 Maximization Bias and Double Learning

6.8 Games, Afterstates, and Other Special Cases

# Introduction

- Temporal-Difference = Dynamic Programming + Monte Carlo
  - MC method : learn directly from raw experience *without a model*
  - DP method : update estimates based in part on other learned estimates, *without waiting for a final outcome*
- RL problem : prediction problem + control problem
  - prediction problem : policy evaluation, estimating value function  $v_\pi$  for a given policy  $\pi$
  - control problem : finding an optimal policy, mainly use generalized policy iteration (GPI)

# 6.1 TD Prediction

- MC method value function update rule

$$V(S_t) \leftarrow V(S_t) + \alpha [G_t - V(S_t)], \quad (6.1)$$

- TD method value function update rule

$$V(S_t) \leftarrow V(S_t) + \alpha [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \quad (6.2)$$

- $G_t = R_{t+1} + \gamma G_{t+1}$  ( $= R_{t+1} + \gamma v_\pi(S_{t+1})$ )
- $G_{t+1} \leftarrow V(S_{t+1})$ 
  - $v_\pi$  : true value function
  - $V$  : estimated value function
- 이를 통해 한 step만 지나도 update 가능하도록 함

# 6.1 TD Prediction

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] \quad (6.3)$$

$$= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \quad (\text{from (3.9)})$$

$$= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]. \quad (6.4)$$

- DP의 bootstrapping, MC의 sampling concept를 동시에 사용!

## Tabular TD(0) for estimating $v_\pi$

Input: the policy  $\pi$  to be evaluated

Algorithm parameter: step size  $\alpha \in (0, 1]$

Initialize  $V(s)$ , for all  $s \in \mathcal{S}^+$ , arbitrarily except that  $V(\text{terminal}) = 0$

Loop for each episode:

  Initialize  $S$

  Loop for each step of episode:

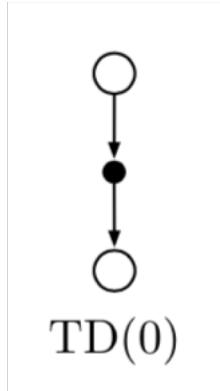
$A \leftarrow$  action given by  $\pi$  for  $S$

    Take action  $A$ , observe  $R, S'$

$V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$

$S \leftarrow S'$

  until  $S$  is terminal

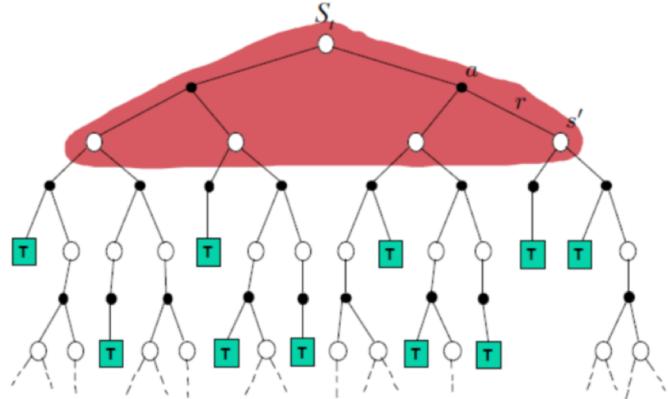


Backup diagram

# 6.1 TD Prediction

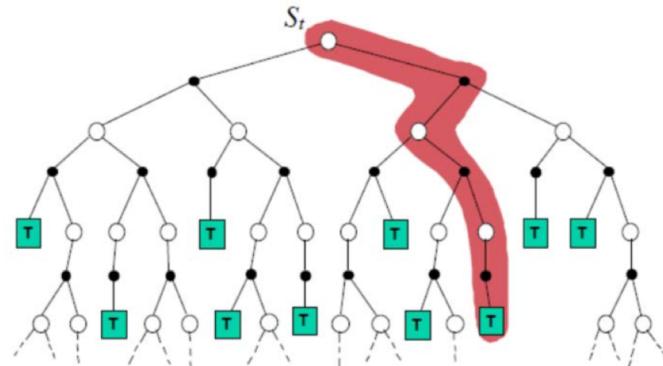
cf. Dynamic Programming

$$V(S_t) \leftarrow E_{\pi} [R_{t+1} + \gamma V(S_{t+1})] = \sum_a \pi(a|S_t) \sum_{s',r} p(s',r|S_t,a)[r + \gamma V(s')]$$



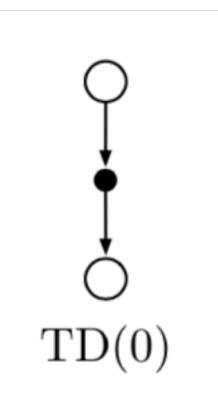
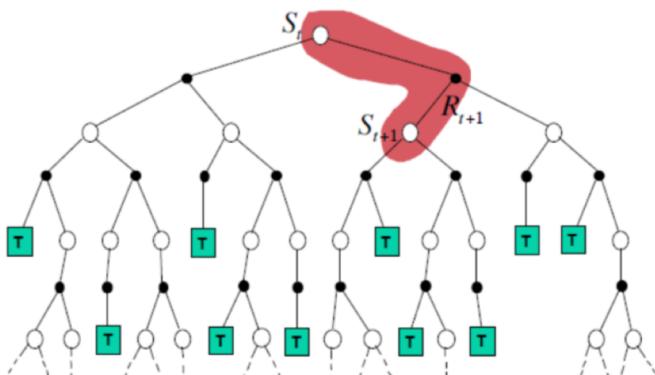
Simple Monte Carlo

$$V(S_t) \leftarrow V(S_t) + \alpha [G_t - V(S_t)]$$



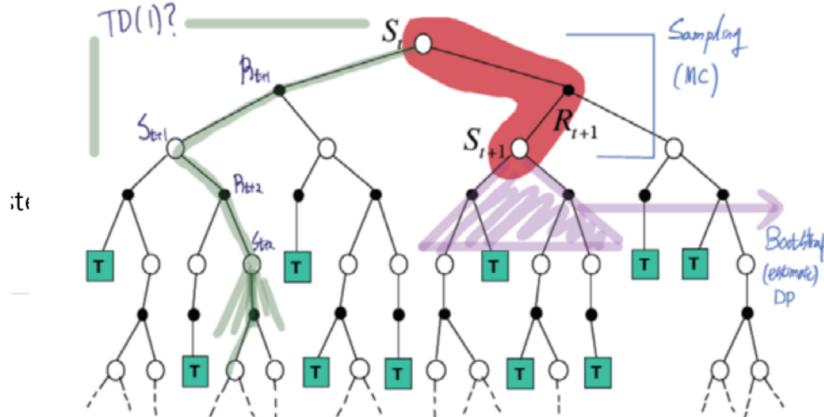
Simplest TD Method

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



Simplest TD Method

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



# 6.1 TD Prediction

- MC와 TD 사이의 차이 : TD Error

$$\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t). \quad (6.5)$$

- TD Error의 합은 Monte Carlo Error

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

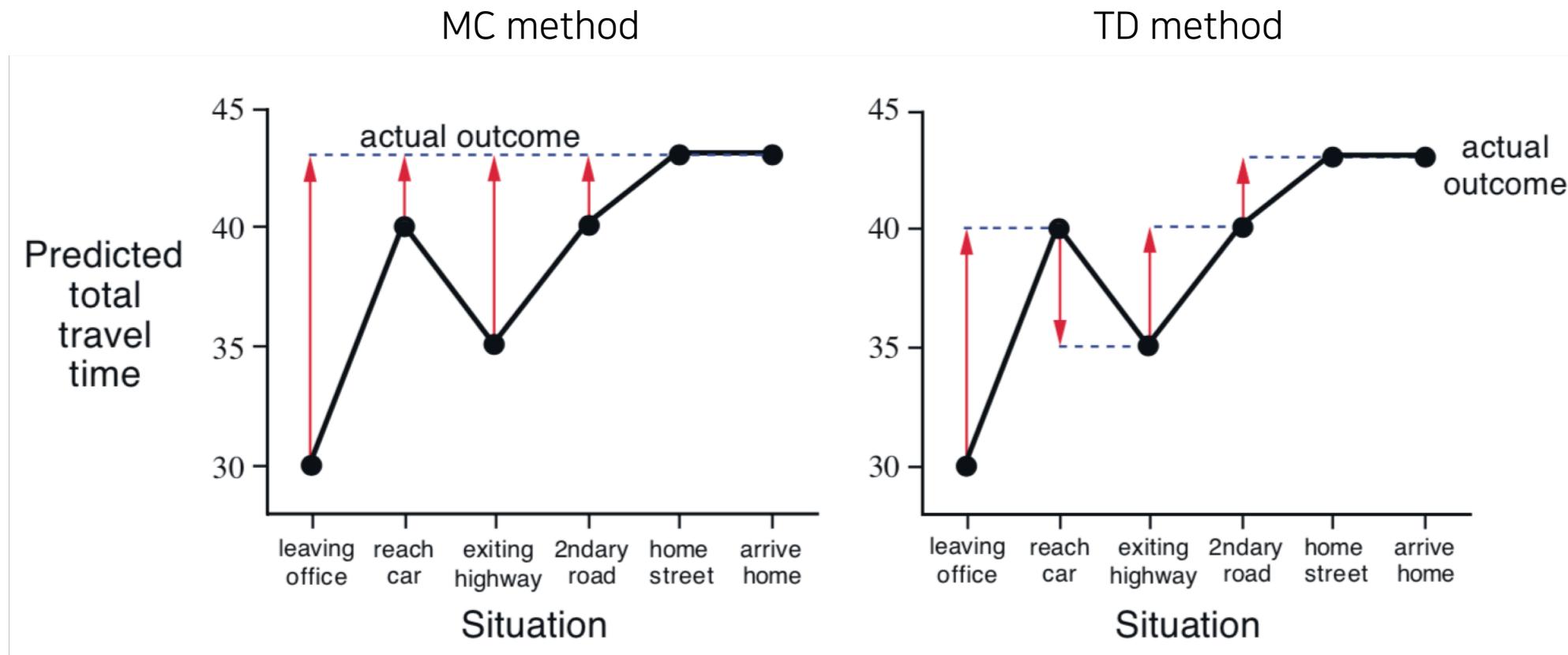
- 한 episode를 끝까지 거치면 TD와 MC의 error는 동일하다!

# 6.1 TD Prediction – Example 6.1

- 퇴근 후 차를 몰고 집에 가는 중
- 집까지 가는데 걸리는 시간을 퇴근 시간, 요일, 날씨, 도로 상황 등을 고려해서 예측하고자 함
- state value function : 집에 가는데 걸리는 시간

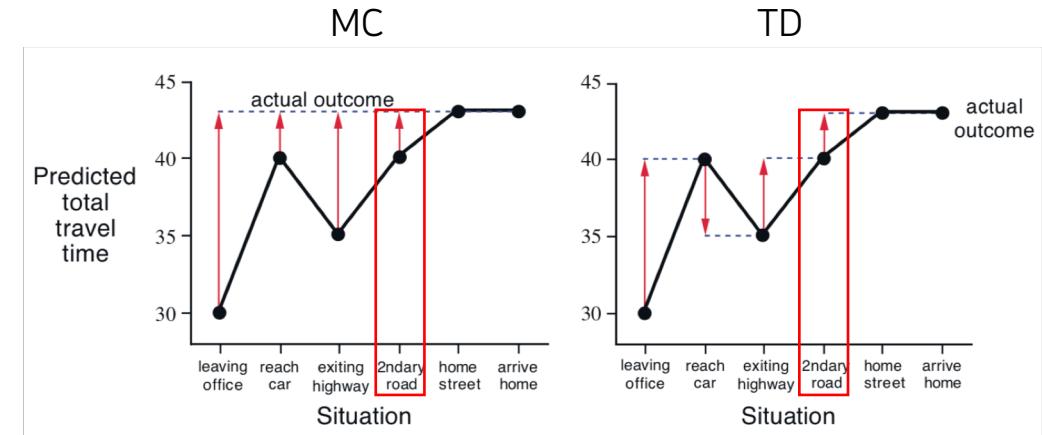
<i>State</i>	<i>Elapsed Time (minutes)</i>	<i>Predicted Time to Go</i>	<i>Predicted Total Time</i>
leaving office, friday at 6	0	30	30
reach car, raining	5	35	40
exiting highway	20	15	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43

# 6.1 TD Prediction – Example 6.1



# 6.1 TD Prediction – Example 6.1

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



- MC에서는 예상 소요 시간을 업데이트하기 위해 actual outcome을 봐야 하기 때문에 기다려야 함
- TD에서는 즉각적으로 예상 소요 시간을 업데이트함
  - *Temporal differences!*

# 6.1 TD Prediction – Example 6.1

- Exercise 6.2 : 왜 이런 문제(example 6.1)에서 MC보다 TD가 좋은가?
  - example 6.1과 같은 문제는 state가 많이 바뀌지 않음
  - 내일 출근할 때에도 비슷한 상황을 거치고, 설령 다른 건물로 출근한다 해도 비슷한 과정을 거침
  - TD는 우리의 initial intuition이 훌륭한 추정을 갖춘 상태라면, 훨씬 수렴이 빨라짐 (실제로 이러한 효과가 나타나는 이유를 example 6.2에서 확인할 예정)
  - 현실에서 우리가 생각할 때(*original learning task*)에도, 초기 *estimate value function*이 *true value function*과 가깝다면 비슷한 현상이 발생한다는 것을 확인할 수 있다.



## 6.2 Advantages of TD Prediction Methods

$$V(S_t) \leftarrow V(S_t) + \alpha [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \quad (6.2)$$

- TD는 추정을 바탕으로 다른 추정량을 업데이트한다.
  - $V$  : *estimated value function*
  - They learn a guess from a guess! (bootstrap)

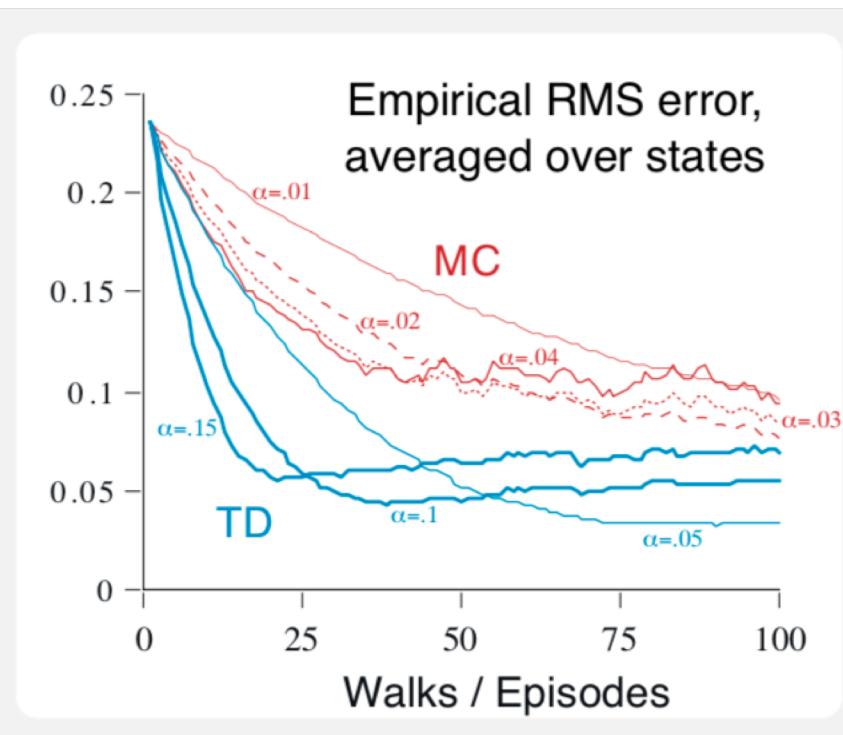
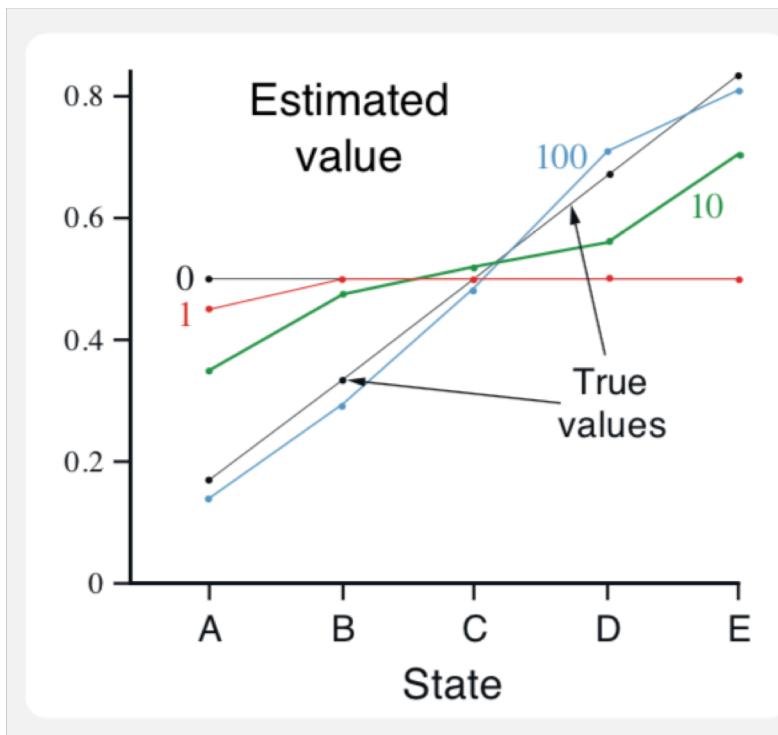
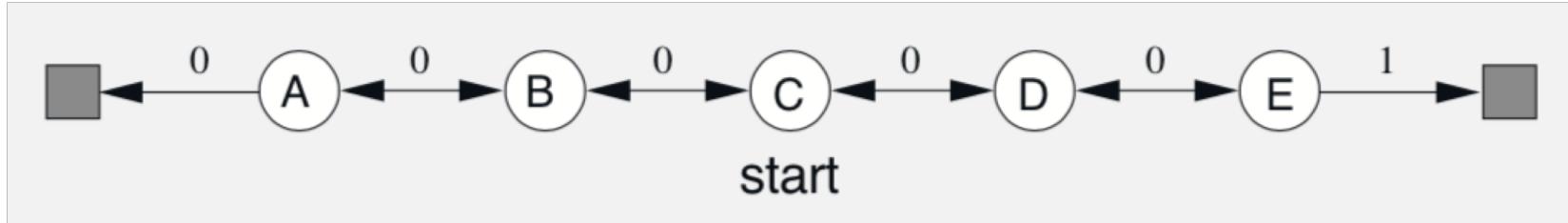
## 6.2 Advantages of TD Prediction Methods

- DP에서는 next-state와 reward, 즉 model을 요구하지만 TD에서는 요구하지 않는다. (MC의 model-free advantage)
- MC는 episode의 terminal을 봐야 하지만 TD는 그럴 필요가 없다. (DP의 bootstrap advantage)
  - episode가 매우 길면 learning이 매우 느려진다.
  - episode가 끝이 나지 않는 continuing task일 경우에는 MC를 사용할 수 없다.
  - MC의 경우 terminal이 명확히 존재해야 하고, terminal이 존재한다고 해도 여러 계산이 어긋나는 경우를 보정하기 위해 복잡한 보정(5.8, 5.9의 discount factor 도입과 horizon 도입)이 필요하지만, TD는 그럴 필요가 없다.

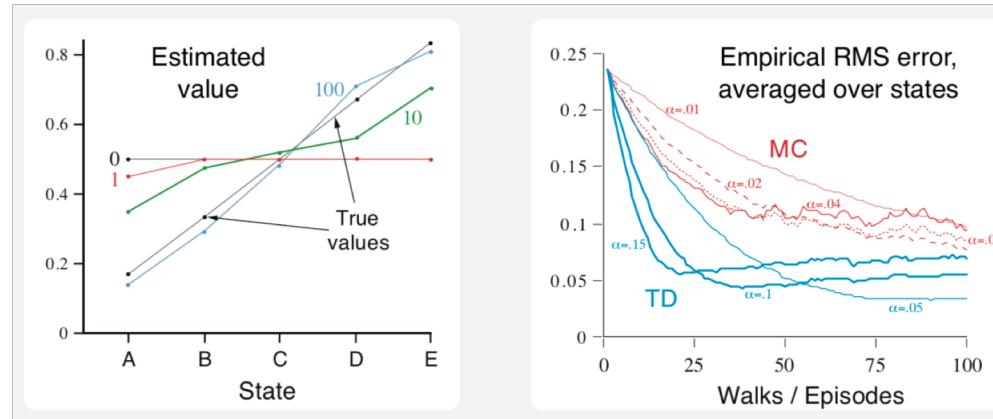
## 6.2 Advantages of TD Prediction Methods

- But are TD methods sound?
  - MC처럼 terminal state를 보지 않아도 fixed policy의 true value function을 찾을 수 있다는 것이 증명됨 (Chapter 9?)
  - MC와 TD의 수렴 속도에 대한 비교는 증명된 바는 없지만, 경험적으로 TD가 빠르다는 추측은 존재한다.

## 6.2 Advantages of TD – Example 6.2

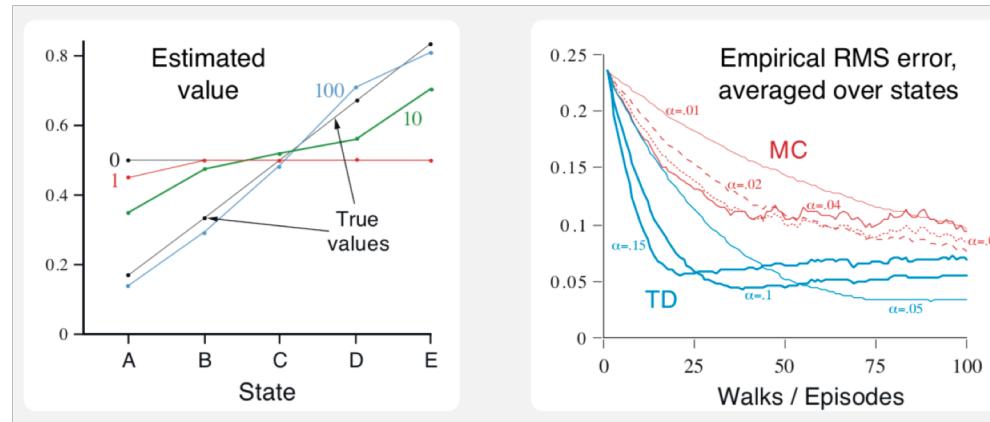


## 6.2 Advantages of TD – Example 6.2



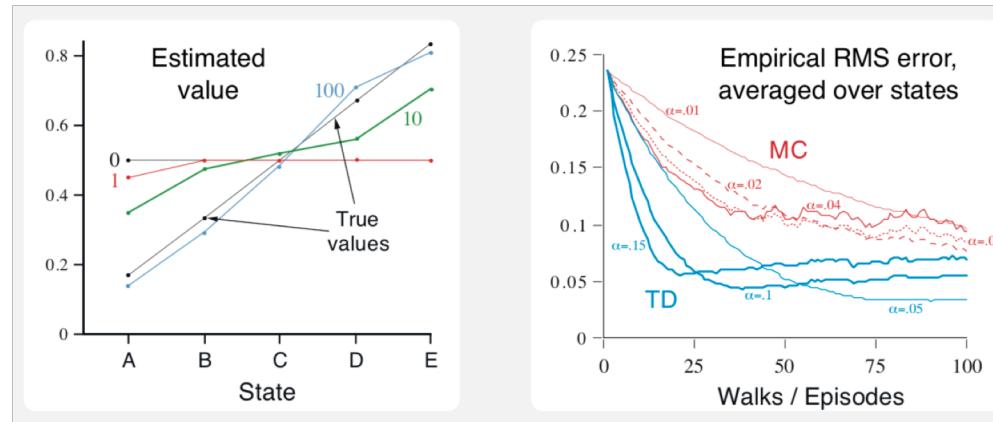
- Exercise 6.3 : 왼쪽 그래프의 첫 스텝에서 왜  $V(A)$ 만 바뀌었는가? 정확히 몇 으로 바뀌었는가?
  - $V(A) \leftarrow V(A) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(A)]$ 
    - $S_{t+1}$  : terminal,  $R_{t+1}, V(S_{t+1}) = 0$  (by initialization)
  - $V(A) \leftarrow V(A) + \alpha[-V(A)]$
  - $V(A) \leftarrow (1 - 0.1) \times 0.5 = \mathbf{0.45}$

## 6.2 Advantages of TD – Example 6.2



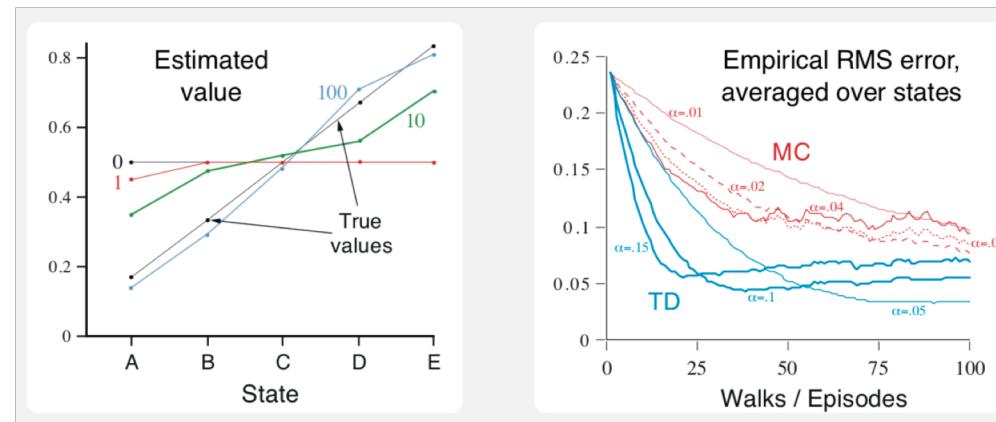
- Exercise 6.5 : 오른쪽 그래프의 TD에서 RMS error가 높은  $\alpha$ 값에서 위아래로 출렁거리는데, 무엇 때문인가? 이 출렁임이 항상 일어나는가? 아니면 value function의 초기값에 따라 달라질 수 있다고 생각하는가?
  - $\alpha$ 값을 크게 잡는다는 것 : 매 과정에서 일어나는 일들을 크게 반영하므로 specific한 경우의 반영이 커진다.
  - TD는 완전한 상황을 보지 않고 한 스텝단위로 update하니깐 초기값이 true value와 멀 경우 크게 흔들린다. 즉, 초기값이 true value와 가까울수록 흔들리지 않는다.

## 6.2 Advantages of TD – Example 6.2



- Exercise 6.4 : 오른쪽 그래프에서 MC와 TD가  $\alpha$ 값에 의존적임을 알 수 있다. 보다 넓은 범위의  $\alpha$ 값을 사용했을 때 MC, TD 중 어느 것이 더 좋은 영향을 받을 것이라 생각하는가? 실험에서 사용한  $\alpha$ 값보다 더 좋은 성능을 보여주는  $\alpha$  값이 있는가?
  - TD가 더 유리하다.
  - MC보다는 TD에서 더 작은  $\alpha$ 값을 시도해볼 필요가 있다.

## 6.2 Advantages of TD – Example 6.2



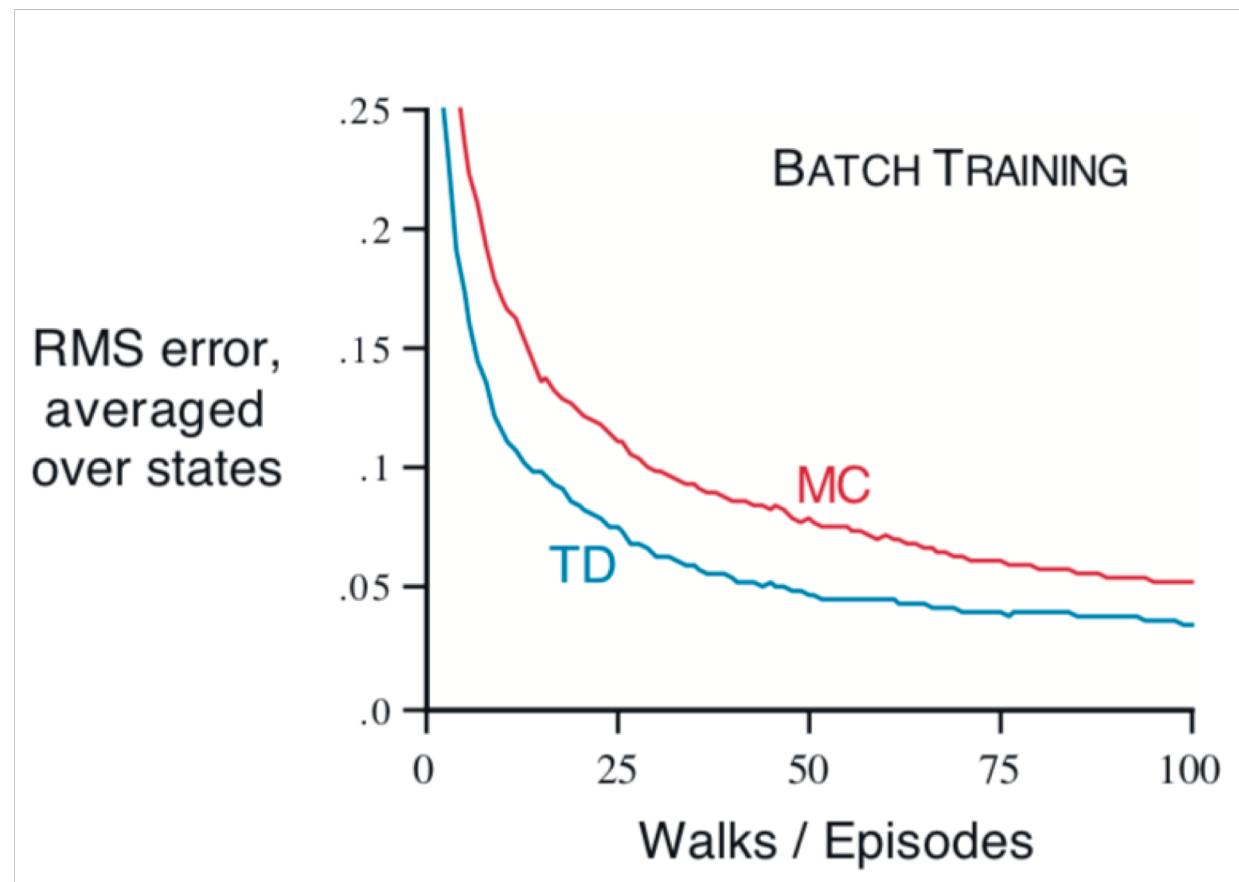
- Exercise 6.6 : A-E state들이 각각  $\frac{1}{6}, \frac{2}{6}, \frac{3}{6}, \frac{4}{6}, \frac{5}{6}$ 의 true value를 갖는데, 이를 계산하는 두 가지 다른 방법을 제시하라. 책에서 사용한 방법은 무엇이며 왜 그렇다고 생각하는가?

## 6.3 Optimality of TD(0)

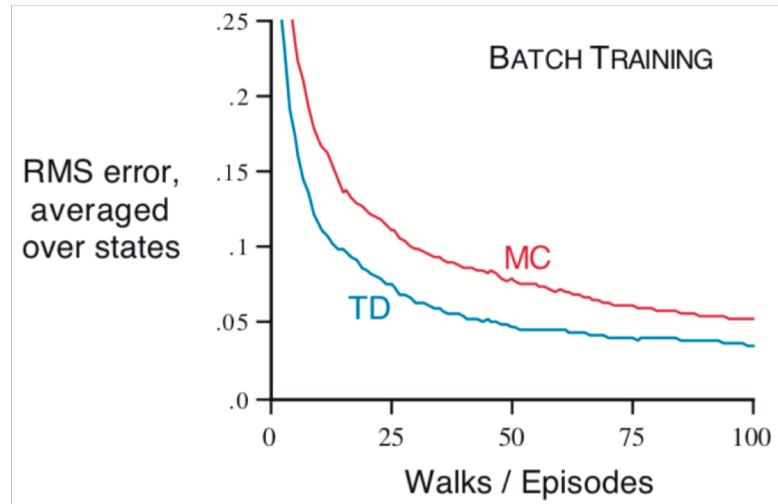
- batch updating : training completely on a finite amount of data, e.g. train repeatedly on 10 episodes until convergence
  - 10개의 에피소드를 하나의 *batch*로 모아 수렴할 때까지 학습 후 value update
  - Finite markov prediction task에서 충분히 작은  $\alpha$ 값으로 batch updating을 하면 TD는 반드시 수렴하며, constant- $\alpha$  MC 또한 수렴하지만 각각 수렴값이 다르다.
  - MC와 TD의 수렴값 차이를 이해하는 것이 두 방법의 차이를 이해하는 데 결정적인 역할을 하므로 간단한 예제(Example 6.3, 6.4)로 이유를 알아보자.

## 6.3 Optimality of TD(0) – Example 6.3

- Example 6.2와 같은 문제에 batch training MC & TD를 적용



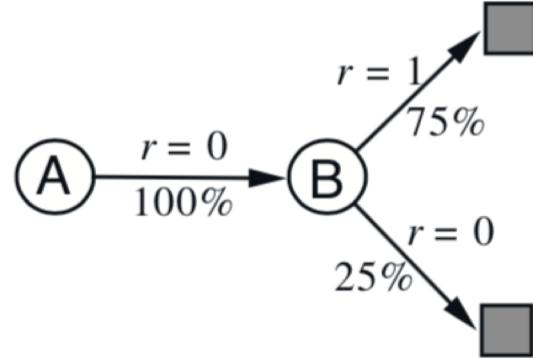
## 6.3 Optimality of TD(0) – Example 6.3



- 두 방법 모두 RMS error 값이 수렴하는 것을 확인할 수 있다.
- 수렴값에 차이가 나는 것 또한 확인할 수 있다.
  - MC는 제한된 경험에서 최적값을 찾는 반면, TD는 실제 return을 예측하고자 하기에 이러한 결과가 나타난다.

## 6.3 Optimality of TD(0) – Example 6.4

- 다음과 같은 문제를 가정



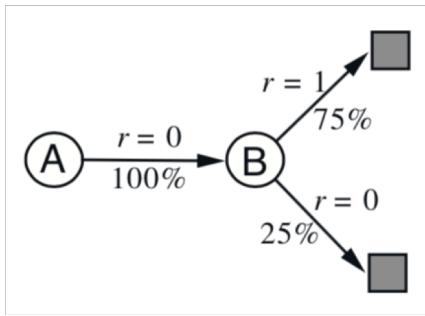
- 8개의 episode가 다음과 같이 나온 상황

A, 0, B, 0	B, 1
B, 1	B, 1
B, 1	B, 1
B, 1	B, 0

- $V(B)$ 는  $\frac{1 \times 6 + 0 \times 2}{8} = \frac{3}{4}$ 로 계산 가능

- $V(A)$ 는?

## 6.3 Optimality of TD(0) – Example 6.4



A, 0, B, 0	B, 1
B, 1	B, 1
B, 1	B, 1
B, 1	B, 0

- batch TD(0)의 경우

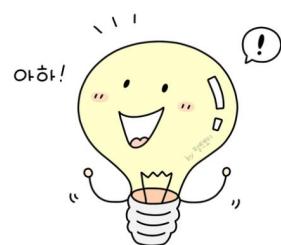
1.  $V(B)$ 를  $\frac{3}{4}$ 로 결정
2.  $A$ 에서  $B$ 로 가는 episode를 확인했으므로  $V(A) \leftarrow V(B)$

- batch MC의 경우

1.  $A$ 가 존재하는 episode는 하나고, 그 episode의 return은 0
2.  $V(A) \leftarrow 0$

## 6.3 Optimality of TD(0) – Example 6.4

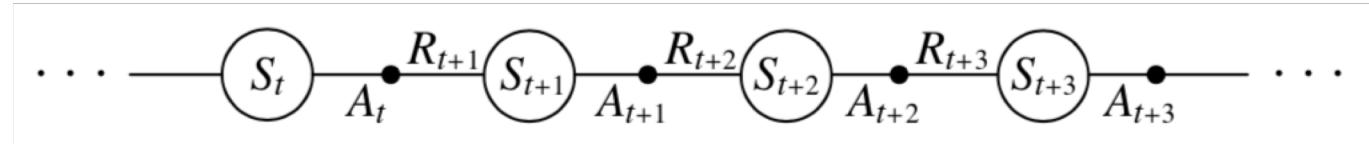
- batch MC의 경우는 항상 training set에서의 MSE를 줄이는 방향으로 간다 (like overfitting).
- batch TD(0)은 주어진 markov process가 가질 수 있는 가장 가능성이 높은 형태를 추정(maximum likelihood estimate)한다.
  - maximum likelihood estimate : 주어진 데이터를 만들어낼 확률이 가장 높은 parameter
  - 만들어낸 모델이 정확하다면 주어진 모델로 계산한 value function이 정확할 것이며, 이렇게 나온 추정치를 certainty-equivalence estimate(CEE)라고 부른다.
  - TD(0)은 CEE에 수렴한다고 한다.



## 6.4 Sarsa : On-policy TD control

- 이제까지 푼 문제 : policy evaluation
- 앞으로 풀 문제 : control problem
  - 평소처럼 GPI의 패턴을 이용하고, GPI의 evaluation part에서 TD methods를 이용할 것이다.
  - MC처럼 state value function 대신 state-action value function을 이용한다.
  - MC에서 exploration과 exploitation의 tradeoff를 조정하기 위해 on-policy와 off-policy 접근을 각각 알아보았던 것처럼 TD 또한 두 가지 방향에서의 접근을 각각 알아보자.

## 6.4 Sarsa : On-policy TD control



- (State-Action pair, Return, State-Action pair) → SARSA!

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]. \quad (6.7)$$

- state to state transition과 state-action pair to state-action pair transition은 둘다 markov chains with a reward process라는 측면에서 동일하다.
- 이는 TD에서 state value function이 수렴한다는 특성이 action-state value function에도 적용된다는 것을 의미한다.

# 6.4 Sarsa : On-policy TD control

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

Loop for each episode:

    Initialize  $S$

    Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\varepsilon$ -greedy)

    Loop for each step of episode:

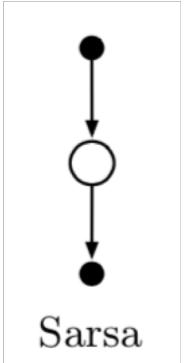
        Take action  $A$ , observe  $R, S'$

        Choose  $A'$  from  $S'$  using policy derived from  $Q$  (e.g.,  $\varepsilon$ -greedy)

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$$

$S \leftarrow S'; A \leftarrow A'$ ;

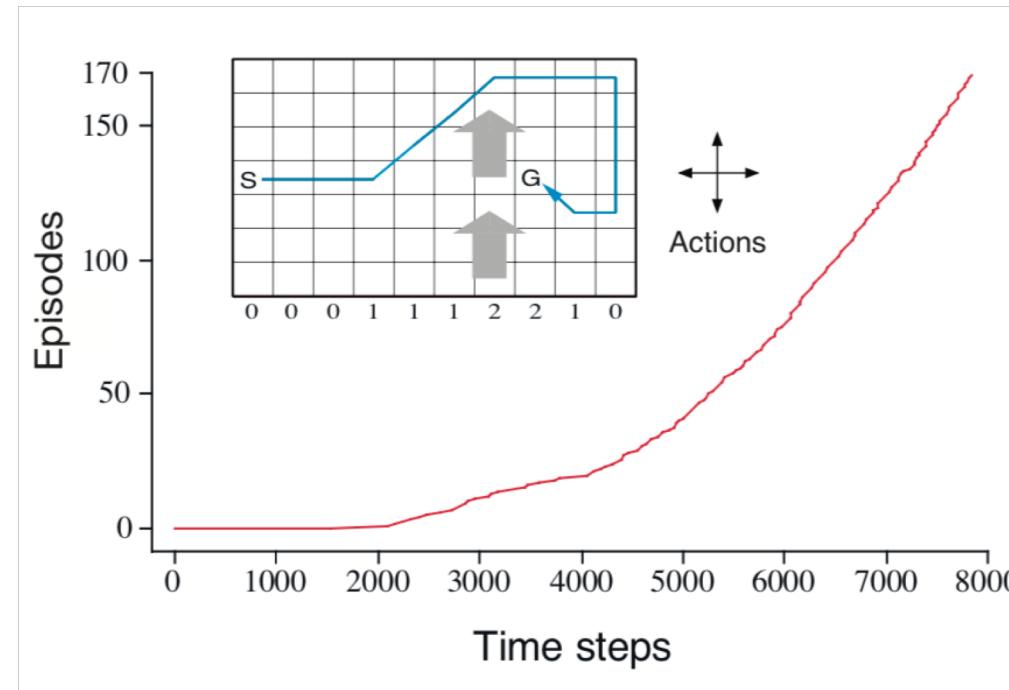
    until  $S$  is terminal



Backup Diagram

- sarsa의 convergence property는 policy의 Q 의존성에 기반한다.
- $\varepsilon$ -greedy(or  $\varepsilon$ -soft) policy 이용에 의존할 수밖에 없다.

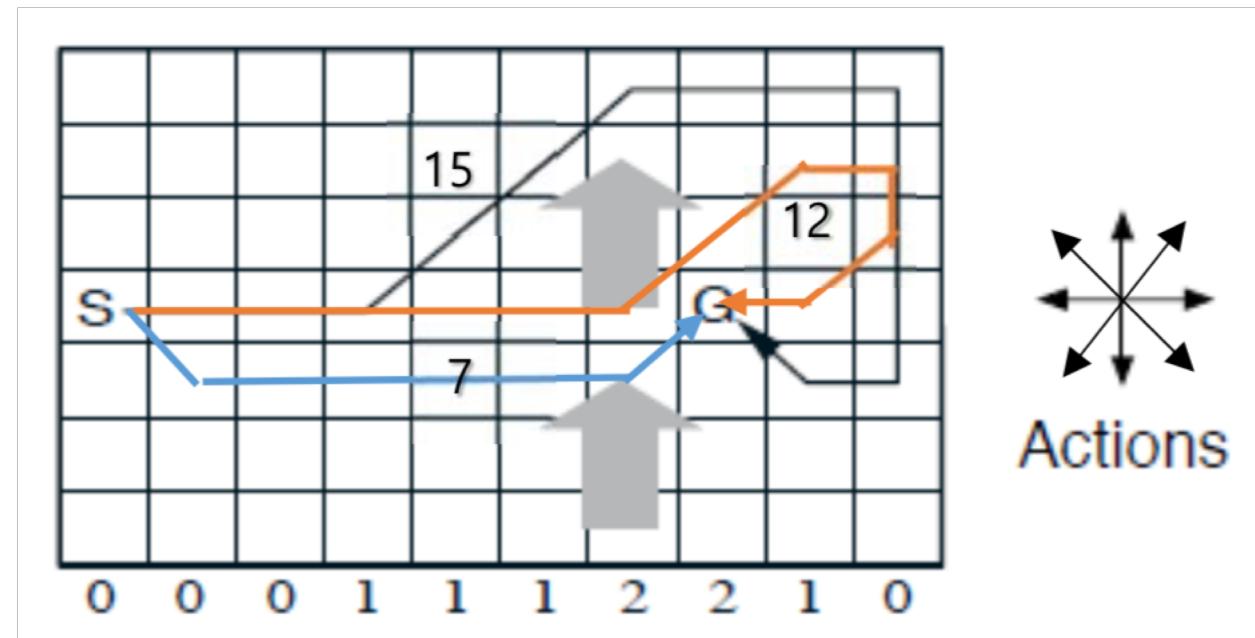
## 6.4 Sarsa – Example 6.5



- Gridworld 아래의 숫자는 바람의 세기를 의미한다(저 세로줄에서 움직이면 강제로 바람의 세기만큼 위로 이동).
- 그래프가 가팔라지는 것은 한 에피소드마다 스텝 수가 줄어들어 최적의 경로를 찾아간다는 것을 의미한다.

## 6.4 Sarsa – Example 6.5

- Exercise 6.9 : King's move(대각선 이동)가 가능해지면 optimal route는 바뀌는가?
  - Maybe...?



# 6.5 Q-learning : Off-policy TD Control

- On-policy : 현재 policy대로 움직이면서 같은 policy를 평가한다. 즉, 현재 policy 위에서 control을 해결한다.
- Off-policy : 움직이는 policy와 평가하는 policy가 다른 경우이다.
  - 초창기 강화학습의 큰 성과는 off-policy TD control인 Q-learning의 개발이라고 한다.

## 6.5 Q-learning : Off-policy TD Control

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]. \quad (6.8)$$

- 학습된 state-action value function인  $Q$ 가 따르고 있는 policy와는 독립적으로  $q_\pi$ (optimal action-value function)의 근사치를 구한다.
- $\max_a Q(S_{t+1}, a)$ 를 이용하는 이유 :  $Q$ 를 maximizing하는 것을 searching 한다는 의미이다. 이를 통해  $Q$ 를 maximizing하는 방향으로 update를 시행한다.

# 6.5 Q-learning : Off-policy TD Control

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

Loop for each episode:

    Initialize  $S$

    Loop for each step of episode:

        Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\varepsilon$ -greedy)

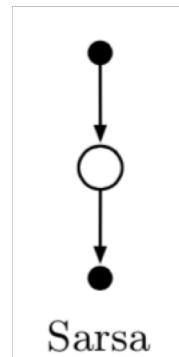
        Take action  $A$ , observe  $R, S'$

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

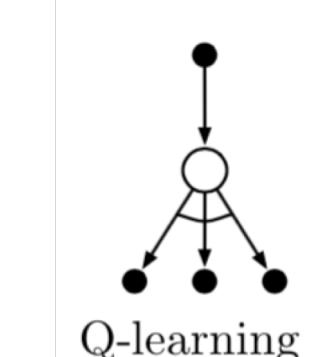
$S \leftarrow S'$

    until  $S$  is terminal

Backup diagram :



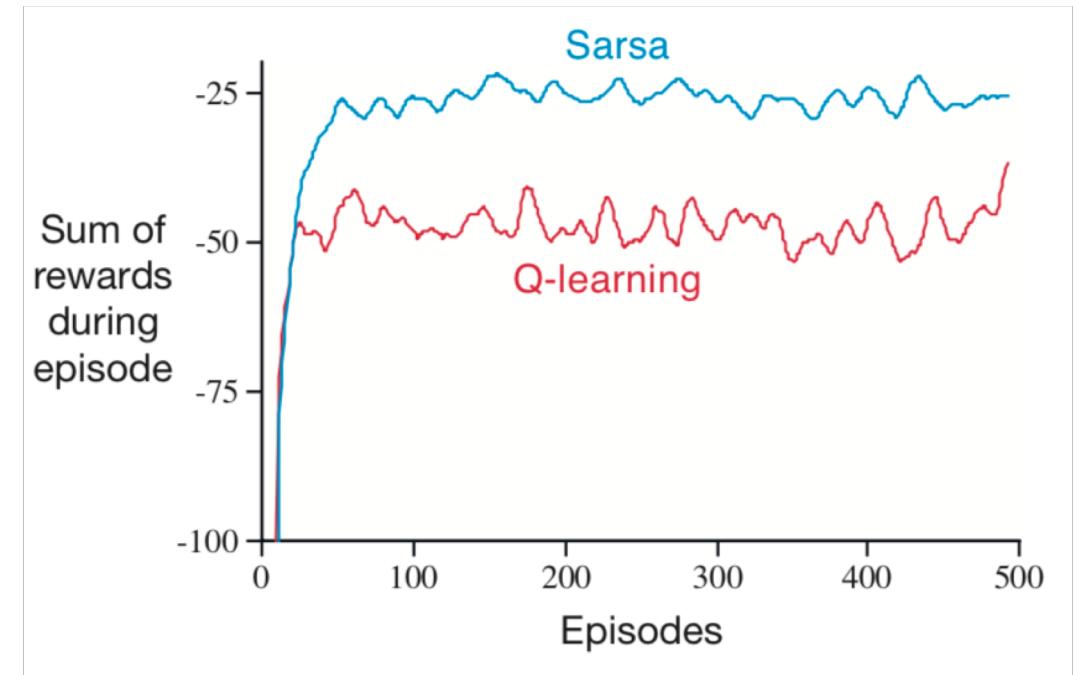
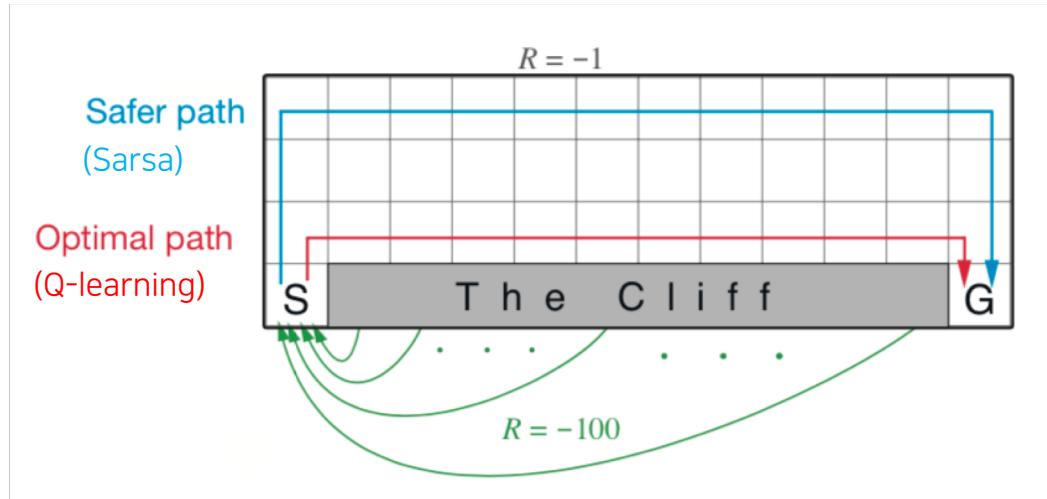
Sarsa



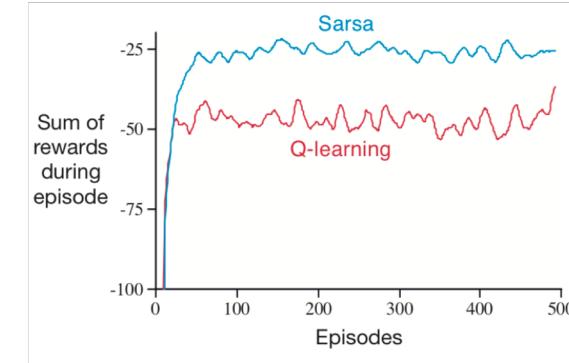
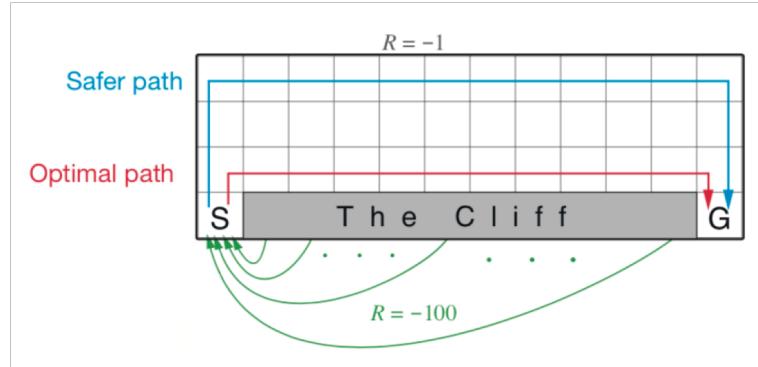
Q-learning



# 6.5 Q-learning – Example 6.6



# 6.5 Q-learning – Example 6.6



- Why?
  - Sarsa : 움직이는 policy를 학습하므로 각 칸에 대한 value를 계산하기 때문에 절벽 쪽 칸의 value를 낮게 계산하여 safer path를 선택한다.
  - Q-learning : 움직이는 policy와는 별개로 return을 최대화하는 방향으로 다른 policy를 학습하기 때문에 optimal path를 찾을 수 있다.

## 6.5 Q-learning – Exercises

- Exercise 6.11 : 왜 Q-learning을 off-policy control method라 할 수 있는가?
  - Because the action selection is taken with respect to an  $\epsilon$ -greedy algorithm while the state-action value function update is determined using a *different* policy.
  - i.e. One that is directly greedy with respect to the action value function.

# 6.5 Q-learning – Exercises

- Exercise 6.12 : Action selection이 greedy하다고 생각해 보자. Q-learning은 sarsa와 같아지는가? 그들은 같은 action selection과 weight update를 할 것인가?
  - The new method is off-policy.
  - The action selection is epsilon-greedy while the action value update is not it is an expectation.
  - If the learning task is stationary(i.e. does not change over time), I would expect this algorithm to perform slightly worse.
  - The tradeoff is again withholding greedy action selection (which would seem to get the best immediate reward) with exploration (which might give better rewards in the future).
  - Here our action value function update explicitly continues to explore by sampling all action value functions in its neighborhood.
  - i.e. The expression

$$\sum_a \pi(s_t, a)Q(s_{t+1}, a)$$

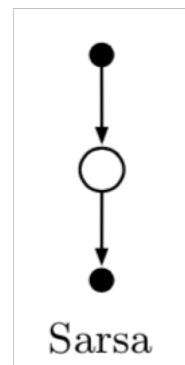
- by using an average rather than taking the explicitly greedy selection.

## 6.6 Expected Sarsa

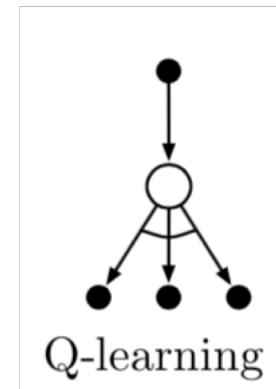
- Sarsa는 random하게 policy에 따라 선택된 episode를 기반으로 update
- Q-learning은 maximizing하는 policy에 따라 update
- Expected sarsa는 모든 가능한 경우의 수의 기댓값을 구해 update

$$\begin{aligned} Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \mathbb{E}_\pi [Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t) \right] \\ &\leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \sum \pi(a|S_{t+1})Q(S_{t+1}, a) - Q(S_t, A_t) \right], \quad (6.9) \end{aligned}$$

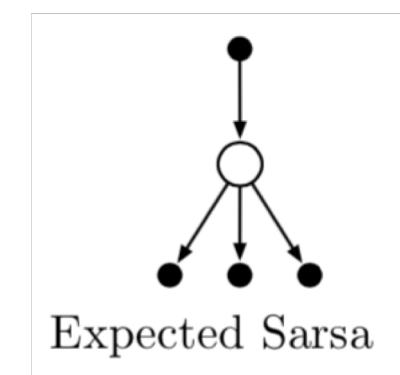
Backup diagram :



Sarsa



Q-learning



Expected Sarsa



## 6.6 Expected Sarsa

- sarsa보다 연산량이 많아지긴 하지만, sarsa에서 하나의 action을 임의로 선택해 업데이트하는 것보다 variance를 낮춰줄 수 있다.
- 같은 experience를 가지고 학습할 때 sarsa보다 더 좋은 performance를 얻을 수 있다.
- sarsa에 비해 다양한 범위의 step-size  $\alpha$  값에 대해 더 좋은 성능을 얻을 수 있다.
  - deterministic reward를 가지는 cliff case에서는  $\alpha$ 를 1로 사용 가능하다(short performance가 별로라면 생각해볼 필요 있음).
- 어떤 관점에서는 expected sarsa는 q-learning의 generalized 버전이라 볼 수 있다.

## 6.7 Maximization Bias and Double Learning

- control algorithm들은 모두 target policy를 만드는데 maximize 기법을 이용한다. 그러나 maximize 기법은 positive bias를 가지고, 이를 maximization bias라 한다.
- 예를 들어, 여러 action들에 대해 Q value  $q(s, a)$ 가 모두 0이라고 가정하자. 추정된 value들은 0보다 높은 값을 가지기도 하고 0보다 낮은 값을 가지기도 하지만, control 알고리즘들은 모두 가장 큰 값을 선택하기 때문에 positive bias가 발생한다.

# 6.7 Maximization Bias and Double Learning

- 이를 해결하기 위해 Double Q-learning 알고리즘이 등장한다.

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \max_a Q_2(S_{t+1}, a) - Q_1(S_t, A_t) \right]. \quad (6.10)$$

- 두 개의 Q-function을 설정하고, 번갈아가며 서로의 estimation을 구한다.
- 이러한 방식이 반복되면 마치 expectation을 구하는 효과를 얻어 unbiased estimate를 구할 수 있다.

# 6.7 Maximization Bias and Double Learning

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

Loop for each episode:

    Initialize  $S$

    Loop for each step of episode:

        Choose  $A$  from  $S$  using the policy  $\varepsilon$ -greedy in  $Q_1 + Q_2$

        Take action  $A$ , observe  $R, S'$

        With 0.5 probability:

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

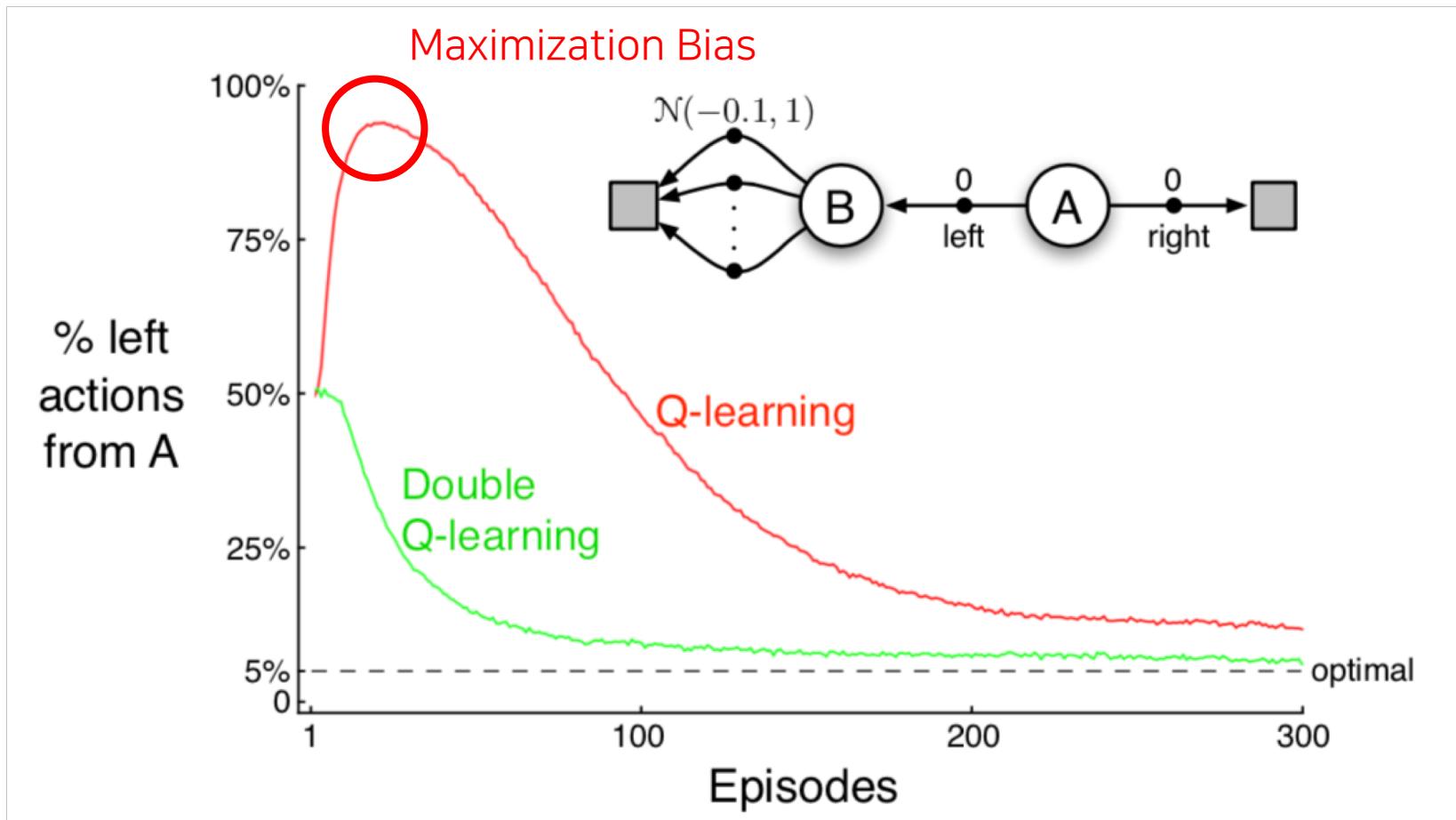
        else:

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

$S \leftarrow S'$

    until  $S$  is terminal

# 6.7 Double Learning – Example 6.7



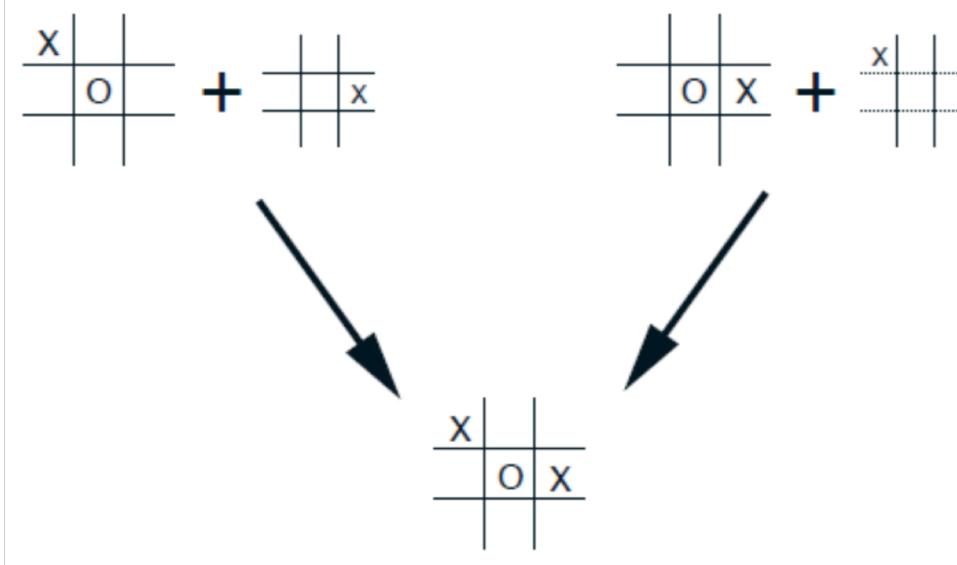
- B 쪽으로 가서 첫 번째 reward가 +값이 나왔을 경우에 발생



## 6.8 Games, Afterstates, and Other Special Cases

- 책에서 되도록 general한 상황에서 다양한 분류의 task를 설명했지만, 특별한 방법으로 처리해야만 하는 예외적인 task가 있기 마련이다.
  - 1장의 tic-tac-toe 게임을 설명할 때 일반적인 state-value function으로 설명했지만 실제와는 다르다.
  - tic-tac-toe에 사용되는 state-value function은 agent가 action을 한 다음에 board position을 평가한다. 이 board position을 afterstates라 하며, 상응하는 value function을 afterstates value functions라 부른다.
  - afterstates는 시작 state에서 environment가 어떻게 변화하는 지에 대한 정보가 있을 때 유용하게 사용된다. 체스나 장기를 시작할 때 어떤 말이 어디로 움직일 수 있을 지 아는 것처럼 말이다.

## 6.8 Games, Afterstates, and Other Special Cases



- 기존 action-value function은 위 2가지 경우를 각각 계산해서 value를 추정 한다.
- 반면 afterstate를 이용하면 두 가지 시나리오 모두 아래의 state로 귀결되므로 관찰 후 아래의 것만 사용한다.

# 6.9 Summary

- DP + MC = TD!
  - DP : bootstrapping
  - MC : model-free
- TD Control
  - On-policy control : Sarsa
  - Off-policy control : Q-learning
  - Generalized Q-learning : Expected sarsa
  - Double Q-learning
- Afterstates

Thank you for listening!