7. n-step Bootstrapping

이 챕터에선 이전 두 챕터에서 본 MC methods 와 one-step temporal-difference(TD) methods 를 합친다.
둘 다 항상 best 는 아니다.
여기서 두 methods 를 일반화하는 n-step TD methods 를 보겠어.
그러면 task 에 맞춰서 하나에서 다른 하나로 부드럽게 바꿀 수 있어.
n-step methods는 MC 와 one-step TD 를 양 끝으로 하는 스펙트럼에 걸쳐 있어.
best는 주로 두 극단의 가운데 쯤이야.

n-step methods 의 또다른 장점은 time step 의 tyranny 에서 자유롭게 해준다는 것이다.

one-step TD methods 를 사용하면,
action 이 얼마나 자주 바뀔지와
bootstrapping 이 수행되는 time interval 을
같은 time step이 결정한다.
많은 applications 에서, 바뀐 사항들을 고려하여 action을 매우 빠르게 update 하고 싶을거다.
하지만 significant, recognizable state change 가 발생할 만큼의 시간이 지나면
bootstrapping 이 제일 좋다.

one-step TD methods 를 쓰면,
이런 time intervals 이 다 같아서, compromise must be made.
n-step methods 는 bootstrapping 이 multiple steps 에서 일어날 수 있게 하고, single time step 의 단점에서 벗어날 수 있게 해준다.

n-step methods 의 idea는 챕터 12의 eligibility traces 의 알고리즘적 idea에 대한 introduction으로 쓰인다.
eligibility traces 가 multiple time intervals 에서 bootstrapping 이 동시에 가능하게 해 줌.
여기서는 n-step bootstrapping idea 만 보자.
이게 문제를 더 간단하게 만들어줄거야.

평소처럼, prediction problem 을 보고 그다음 control problem 을 보자.
그러니까,
먼저 n-step methods 가 어떻게 fixed policy 에서 state 의 함수로 returns 을 예측하는지 보고,(i.e., v_π 추정)
그다음 action values 와 control methods 로 idea를 확장한다.

7.1 n-step TD Prediction

MC 와 TD methods 사이에 놓인 methods 의 공간은 뭘까?
v_π 를 π로 생성된 sample episodes 로부터 estimating 하는 걸 생각해보자.
MC methods 는 각 state 에서 episode 의 끝까지 관측된 rewards 의 전체 sequence에 기반해서 각 state 를 update 한다.
반면, one-step TD methods 는 바로 다음 reward 에만 기반하며, one step 뒤의 state 의 value 에서 bootstrapping 한 걸 remaining rewards 를 대신해 쓴다.

그럼 intermediate method 의 한 종류는, intermediate number of rewards 에 기반한 update 를 수행하겠지. 하나보단 많지만, termination 까지의 모두를 포함하진 않는.
예를 들어, two-step update 는 첫 두 rewards 와 2 step 뒤의 estimated value of the state 에 기반한다.
비슷하게, 3 step updates, 4 step updates 다 할 수 있겠지.
아래 Fig 7.1 은 v_π을 위한 n-step updates 의 spectrum의 backup diagrams 이다.

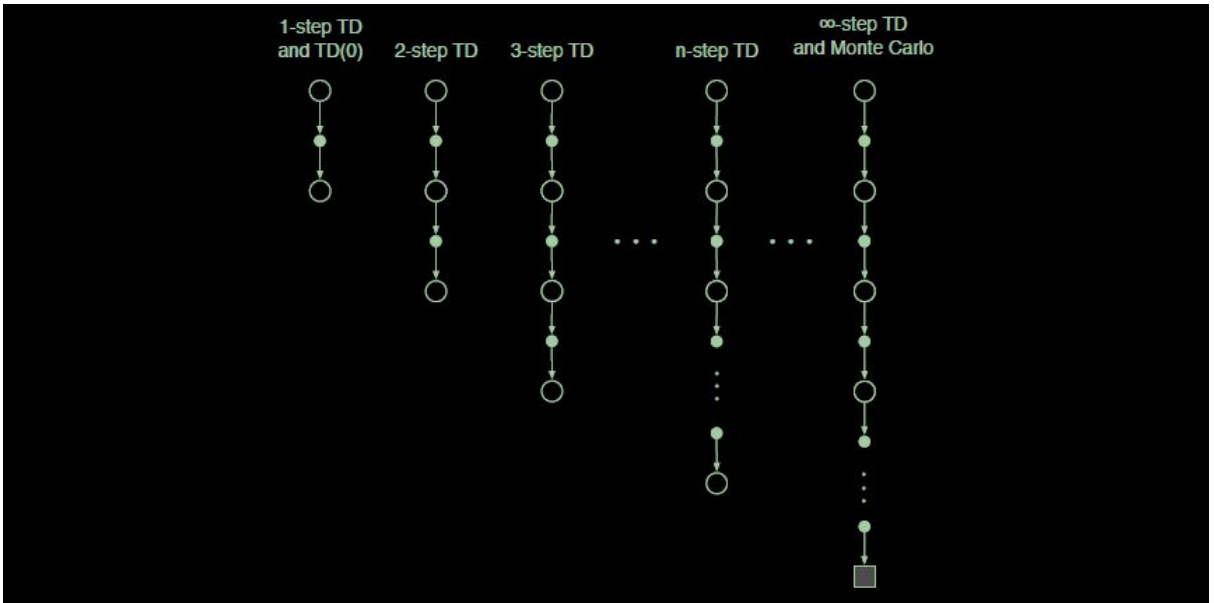


Figure 7.1: The backup diagrams of n-step methods. These methods form a spectrum ranging from one-step TD methods to Monte Carlo methods.

몇 개의 스텝까지 보고 업데이트를 할 거냐 문제인가봐.

n-step updates 를 쓰는 methods는 여전히 TD methods 이다. 왜냐면 later estimate 가 어떻게 바뀌냐에 기반해서 earlier estimate 를 바꾸니까.
이제 later estimate 가 one step 이 아니라 n-step later 이다.
temporal difference가 n-step 으로 확장한 methods 가 n-step TD methods이다.
이전 챕터에서 소개된 TD methods 가 전부 one-step updates를 썼다.
그래서 one-step TD methods 라고 했지.

더 formally, state-reward sequence, $S_t, R_{t+1}, S_{t+1}, R_{t+2}, \ldots, S_T, R_T$ (actions 은 생략) 의 결과로 state S_t 의 estimated value 를 update 하는 걸 생각해보자.
MC methods에서 S_t 의 v_π 의 estimates 는 complete return의 방향으로 업데이트 된다:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T,$$

T는 episode의 마지막 time step이고.
이 quantity 를 update 의 target 이라고 해볼까.
MC 에서는 return 이 target 이었는데,
one-step updates 에서는 target 이 first reward + discounted estimated value of next state이다. 이걸 one-step return 이라 부르고:

$$G_{t:t+1} \doteq R_{t+1} + \gamma V_t(S_{t+1}),$$

이거지.
Vt: $S \to \mathscr{R}$ 은 여기선 v_π의 time t에서의 estimate.
$G_{t:t+1}$ 은 time t+1 까지의 rewards 를 이용해 time t 에 대한 truncated return 이다.

full return 대신 저 discounted 된 거 하나만 쓰는 게 포인트.
더 중요한 포인트. two steps 해도 one step 만큼 말이 될 거란 말이지.
two-step update 의 target 은 two-step return:

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

임의의 n-step return 도 비슷하겠지?

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

모든 n-step returns 는 full return 의 approximations 으로 볼 수 있다.
n steps 이후가 잘려나가고 remaining missing terms이 마지막 항으로 대체되는거지.
t+n ≥ T 이면, 그러니까 n-step return 이 termination 을 넘어가면, 모든 missing terms 을 0으로 처리하고 n-step return 은 보통의 full return 과 같다고 본다.

$$(G_{t:t+n} \doteq G_t \text{ if } t+n \geq T)$$

요로케.

n>1 인 n-step returns 는 t 에서 t+1 로 가는 transition 의 time 에 not available한 future rewards 와 states 를 포함한다.
R_{t+n}을 관측하기 전까진, V_{t+n-1} 을 계산하기 전까진 n-step return 을 쓸 수 없다.
가장 먼저 얘들을 쓸 수 있는 때는 $t+n$ 일 때이다.
그러니, n-step returns 을 쓸 수 있는 state-value learning algorithm 은 이렇게 된다.

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha[G_{t:t+n} - V_{t+n-1}(S_t)], \qquad 0 \leq t < T, \tag{7.2}$$

모든 다른 states 의 values 가 remain unchanged 일 때:

$$V_{t+n}(s) = V_{t+n-1}(s), \text{ for all } s \neq S_t.$$

이 알고리즘을 n-step TD 라고 한다.

각 episode 의 첫 n-1 steps 에서는 아무런 changes 가 없다.
이를 보완하기 위해, episode 의 마지막에 같은 수의 추가적인 updates 를 한다. termination 이후, 다음 episode 를 시작하기 전에.
전체 pseudo code 가 아래 박스.

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 each step of episode, $t = 0, 1, 2, \ldots$:
  | 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: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $V(S_\tau) \leftarrow V(S_\tau) + \alpha [G - V(S_\tau)]$
  Until $\tau = T - 1$

위 박스 설명.
Input 은 a policy π 이다.
알고리즘의 parameters 는 step size α 와 양의 정수 n
$V(s)$ 는 임의로 초기화한다. 모든 s에 대해.
S_t, R_t 의 모든 store, access operations 은 index를 mod n+1 로 취할 수 있다?
각 episode마다 loop 돌 것..:
  non-terminal S_0 을 initialize 하고 store한다.
  T는 무한대로.
  episode의 각 step에서 loop:
    $t < T$ 일 때:
      $\pi(\cdot|S_t)$에 따라 action 취함.
      R_{t+1}, S_{t+1} 를 observe and store.
      S_{t+1} 이 terminal 이면, $T \leftarrow t+1$
    $\tau \leftarrow t - n + 1$ (τ는 estimate 가 update 되고 있는 state의 time)
    $\tau \geq 0$ 일 때:
      G 는 discounted return 맞나? 입력.
      terminal 안 넘어가면 G update.
      그 G에서 이전 value 빼고 α곱해 더해주면 새로운 value.
  $\tau = T$ 될 때까지.(terminal)

(Exercise 7.1)
챕터 6에서는

(Exercise 7.2)

n-step return 은 R_{t+n} 을 넘어서는 missing reward 를 correct 하기 위해 V_{t+n-1}를 쓴다.

n-step returns 의 중요한 특성은 그 expectation 이 V_{t+n-1} 보다 더 나은 estimate 를 보장하는 것이다. worst-state sense에서.
그러니까, expected n-step return 의 worst error 가 V_{t+n-1}하에서의 worst error 에 곱하기 γ^n 보다 작거나 같음을 보장한다:

$$\max_s \left| \mathbb{E}_\pi[G_{t:t+n} | S_t = s] - v_\pi(s) \right| \leq \gamma^n \max_s \left| V_{t+n-1}(s) - v_\pi(s) \right|, \tag{7.3}$$

모든 n ≥ 1에 대해서.
이걸 error reduction property of n-step returns 라고 부른다.
이것때메 모든 n-step TD methods 가 적절한 기술적 조건 하에서 correct predictions 으로 수렴한다.
그래서 n-step TD methods 는 좋은 methods 의 family 를 형성한다.
one-step TD 와 Monte Carlo methods 를 extreme members 로 두면서.

(Example 7.1) n-step TD methods on the Random Walk
n-step TD methods 를 example 6.2의 5-state random walk task 에 쓰는 걸 생각해 보자.
첫 번째 episode 가 C에서 오른쪽으로 바로 D, E 하고 return 1 받으면서 terminate 됐다고 가정하자.
모든 states 의 estimated values 가 intermediate value $V(s)$ = 0.5로 시작했던 거 기억나니?
이 experience 의 결과로, one-step method 는 마지막 state 의 estimate 만 바꿀 거야.
V(E)가 observed return 1로 증가할 거야.
반면, two-step method 는 termination 앞의 두 states 의 value 를 증가시킨다.
V(D), V(E) 가 모두 1로 증가하겠지.
three-step method 나 어떤 n>2인 n-step method 는 방문한 3개 states 의 value 모두 1로 증가시킬 거다.

n 으로 어떤 값이 더 좋을까?
fig 7.2 는 간단한 empirical test 의 결과다.
좀 더 큰 random walk process 를 진행했지. 5개 대신 19 states. 왼쪽에 -1. 전부 0으로 initialize.
이 챕터에서 running example 로 쓸 거래.
n, α 를 조절하며 n-step TD methods 한 결과가 Fig7.2에 나와 있어.
따로 찾아보렴. 눈부셔서 뺐다.
각 parameter 세팅의 performance measure는 episode의 마지막에 predictions 과 19 states 의 true values 의 average squared error 에 root 씌운 거야. 그걸 또 첫 10개의 episodes 를 100번 반복한 average를 쓰네.
n이 중간 정도일 때 best 야.
TD 와 MC methods 를 n-step methods 로 일반화하는 게 두 극단보다 잠재적으로 더 나을 수 있다는 걸 보여주지.

(Exercise 7.3)

7.2 n-step Sarsa

어떻게 n-step methods 가 prediction 뿐만 아니라 control 에도 쓰일 수 있을까?
n-step method 가 Sarsa 와 어떻게 엮이는지 보자.

on-policy TD control method 를 생성하기 위한 straightforward한 방법이야.
Sarsa의 n-step 버전은 n-step Sarsa 라고 불러.
이전 챕터의 원래 버전은 그럼 one-step Sarsa 또는 Sarsa(0) 이겠지?

main idea 는 states for actions (state-action pairs) 를 바꾸는거야.
그리곤 ε-greedy policy 를 쓰는거지.
n-step Sarsa 의 backup diagrams 은 n-step TD와 같이 states 와 actions 의 연속이야.
Sarsa 가 action 에서 시작해 action 에서 끝나는 걸 빼곤 말이야.

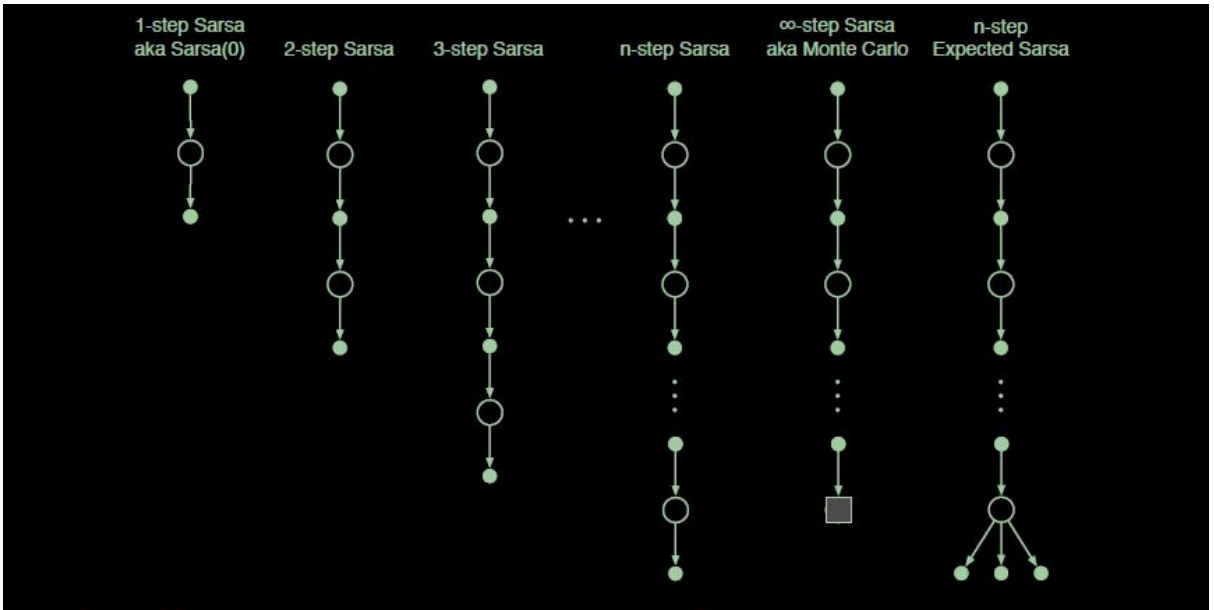


Figure 7.3: The backup diagrams for the spectrum of n-step methods for state–action values. They range from the one-step update of Sarsa(0) to the up-until-termination update of the Monte Carlo method. In between are the n-step updates, based on n steps of real rewards and the estimated value of the nth next state–action pair, all appropriately discounted. On the far right is the backup diagram for n-step Expected Sarsa.

n-step returns (update targets) 를 estimated action values 관점에서 재정의하자:

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}), \quad n \geq 1, 0 \leq t < T - n, \quad (7.4)$$

with $G_{t:t+n} \doteq G_t$ if $t + n \geq T$.

natural algorithm 은 그럼 이렇게 되지.

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha \left[G_{t:t+n} - Q_{t+n-1}(S_t, A_t) \right], \qquad 0 \leq t < T, \quad (7.5)$$

다른 모든 states 의 values 는 remain unchanged.
$Q_{t+n}(s, a) = Q_{t+n-1}(s, a)$, $s \neq S_t$ or $a \neq A_t$ 인 모든 s, a 에 대해.
이게 우리가 n-step Sarsa 라고 부르는 알고리즘이다.
밑에 수도코드도 나와있고, 왜 one-step methods 보다 학습을 빠르게 하는지 Fig 7.4 에 나와 있다.

Figure 7.4: Gridworld example of the speedup of policy learning due to the use of n-step methods. The first panel shows the path taken by an agent in a single episode, ending at a location of high reward, marked by the G. In this example the values were all initially 0, and all rewards were zero except for a positive reward at G. The arrows in the other two panels show which action values were strengthened as a result of this path by one-step and n-step Sarsa methods. The one-step method strengthens only the last action of the sequence of actions that led to the high reward, whereas the n-step method strengthens the last n actions of the sequence, so that much more is learned from the one episode.

위 Fig 7.4: n-step methods 로 policy learning 을 빠르게 한 예제.
첫 번째 그림은, single episode 에서 agent 가 지나간 길을 보여준다. high reward 를 받은 위치에서 끝났고 G 로 표시했다. 모든 values 는 0으로 초기화했고, 모든 rewards 는 0이다. G에서만 positive reward.
다른 두 그림의 화살표는, 어떤 action values 가 one-step, n-step Sarsa methods 로 강화됐는지 보여준다.
one-step method 는 G를 받기 바로 마지막 action 만 강화.
n-step method 는 마지막 n actions 을 강화.
그래서 한 episode 에서 더 많이 학습할 수 있다.

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

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 each step of episode, $t = 0, 1, 2, \ldots$:
  | 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 $\tau = T - 1$

위 박스를 한 번 볼까?
q_* 또는 q_π 를 approximate 하는 Q 를 estimating 하는 n-step Sarsa 되시겠다.
$Q(s, a)$를 initialize arbitrarily. 모든 s, a에 대해.
π도 initialize. Q나 fixed given policy에 대해 ε-greedy 하도록.
알고리즘 파라미터: α, ε, n
episode마다 loop:
  terminal 이 아닌 S_0을 initialize 하고 저장.
  $\pi(\cdot|S_0)$를 따르는 A_0를 선택하고 저장.
  T는 우선 무한대.
  episode의 각 step 마다 loop, t=0,1,2,...:
    $t < T$ 이면:
      action A_t 를 하고
      R_{t+1}, S_{t+1} 을 관찰, 저장.
      S_{t+1}이 terminal이면 T를 t+1 로.
      평소엔 $\pi(\cdot|S_{t+1})$를 따르는 A_{t+1}를 선택하고 저장
    $\tau = t - n + 1$
    $\tau \geq 0$ 이면:
      G 계산.
      $\tau + n < T$ 이면, 뒤에꺼 추가. 의미가 뭐였지.
      $Q(S_\tau, A_\tau)$ 업데이트.
      π 가 학습되고 있으면 $\pi(\cdot|S_\tau)$ 이 Q에 대해 ε-greedy 하도록 확실하게!
  τ 가 T - 1 이 될 때까지.

Expected Sarsa 는 어떨까?
Expected Sarsa 의 n-step version 에 대한 backup diagram 은 Fig 7.3 의 가장 오른쪽에 나와 있다.
n-step Sarsa 처럼 sample actions, states 의 linear string 이 있고, 마지막에 π에 의해 weighted 된 모든 action 확률로 갈라져 있다.
이 알고리즘은 n-step return 만 바꿔서 n-step Sarsa 와 같은 방정식으로 표현된다.

$$G_{t:t+n} \doteq R_{t+1} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \sum_a \pi(a|S_{t+n}) Q_{t+n-1}(S_{t+n}, a), \tag{7.6}$$

for all n and t such that $n \geq 1$ and $0 \leq t \leq T - n$.

7.3 n-step Off-policy Learning by Importance Sampling

off-policy learning 은 b를 따르는 동안 π로 value function 을 학습했던 거 기억나? 그땐 참 좋았지.
보통,
π는 현재 action-value-function estimate 에 대해 greedy policy 이고,
b는 좀 더 exploratory policy 야. 아마 ε-greedy 겠지.
b로부터 얻은 data를 사용하기 위해 두 policies 의 차이를 고려해야 해. 취해진 actions의 상대적 확률을 이용해서. section 5.5 를 볼래?
n-step methods 에서, returns 은 n steps 으로 만들어지고, 그러니까 그 n actions 의 상대적 확률에만 관심이 있지.
예를 들어, n-step TD 의 간단한 off-policy 버전을 만들려면,
time t에서의 update(time t+n 에서 만들어지는) 는 $\rho_{t:t+n-1}$ 로 weighted 될 수 있다:

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha \rho_{t:t+n-1} \left[G_{t:t+n} - V_{t+n-1}(S_t) \right], \quad 0 \leq t < T, \tag{7.7}$$

$\rho_{t:t+n-1}$ 는 importance sampling ratio 지?
A_t 부터 A_{t+n-1} 까지 n actions 를 취하는 두 policies 하에서의 상대적 확률이야.

$$\rho_{t:h} \doteq \prod_{k=t}^{\min(h, T-1)} \frac{\pi(A_k|S_k)}{b(A_k|S_k)} \tag{7.8}$$

예를 들어, π에 따라 어떤 action 도 취해지지 않을 거라면,
그러니까 $\pi(A_k|S_k)$ = 0 이면,
n-step return 은 zero weight 가 주어지고 완전히 무시된다.
반면, b보다 더 큰 확률로 π가 취할 action 이 취해지면(무슨말?),
weight 를 증가시킬거야. 안그랬으면 return 에 주어졌을걸. (무슨말인지..)
이건 말이 되는데, 왜냐면 그 action 이 π의 characteristic 이지만(그래서 우리가 학습하고 싶은 거지) b에 의해 선택되는 경우는 거의 없고, 그래서 data에서 잘 안 나타나기 때문이다.
이걸 보완하기 위해, 이게 발생했을 때 이걸 over-weight 해야해.

두 policies 가 같으면(on-policy case면)
importance sampling ratio 가 항상 1이야.
그래서 (7.7)의 new update 가 일반화하면서 n-step TD methods 를 완전히 대체할 수 있기도 하지.
비슷하게, n-step Sarsa update 가 간단한 off-policy form 으로 완전히 대체됨:

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha \rho_{t+1:t+n-1} \left[G_{t:t+n} - Q_{t+n-1}(S_t, A_t) \right], \tag{7.9}$$

여기서 importance sampling ratio 는 위의 n-step TD 보다 한 step 나중에 시작한다.
왜냐면 여기선 state-action pair 를 update 하기 때문.
action 을 선택할 가능성이 얼마나 되는지 신경 쓸 필요가 없다.
우리가 이걸 선택했기 때문에 발생한 일로부터 fully 학습하길 원하기 원한다.
subsequent actions 에 대해서만 importance sampling 해서.
아래에 수도코드가 나온다.

n-step Expected Sarsa 의 off-policy 버전은 n-step Sarsa 와 같은 update 를 쓰겠지만
importance sampling ratio 의 factor 가 하나 작다.
그러니까, (7.9)에서 $\rho_{t+1:t+n-1}$ 대신 $\rho_{t+1:t+n-2}$ 를 쓰는거고,
당연히 (7.6)의 n-step return 의 Expected Sarsa 버전을 쓸 거다.
왜냐면 Expected Sarsa 에선 마지막 state에서 모든 가능한 actions 이 고려되기 때문이다.
실제로 취해진 action 은 영향이 없고 수정될 필요가 없다.

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 each step of episode, $t = 0, 1, 2, \ldots$:
    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 $A_{t+1} \sim b(\cdot|S_{t+1})$
    $\tau \leftarrow t-n+1$ (τ is the time whose estimate is being updated)
    If $\tau \geq 0$:
      $\rho \leftarrow \prod_{i=\tau+1}^{\min(\tau+n-1,T-1)} \frac{\pi(A_i|S_i)}{b(A_i|S_i)}$ ($\rho_{\tau+1:t+n-1}$)
      $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
      If $\tau+n < T$, then: $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ ($G_{\tau:\tau+n}$)
      $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\rho[G - Q(S_\tau, A_\tau)]$
      If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt Q
  Until $\tau = T-1$

위 박스 설명:

7.4 *Per-decision Off-policy Methods with Control Variates

이전 section 의 multi-step off-policy methods 는 매우 간단하고 개념적으로 명료하다.
하지만 가장 효율적인 건 아냐.
좀 더 세련된 approach 는 per-decision importance sampling ideas 를 쓰는 거겠지.

section 5.9 에서 소개했던 것처럼.
이걸 이해하기 위해, 먼저
n-step return (7.1) 이 recursively 쓸 수 있다는 걸 기억해. 다른 모든 returns 처럼.

$G_{t:h} = R_{t+1} + \gamma G_{t+1:h}$

이제 $b \neq \pi$ 인 behavior policy를 따르는 효과를 생각해보자.
모든 결과 experience는, 첫 reward R_{t+1} 과 다음 state S_{t+1} 을 포함해서,
time t 에 대한 importance sampling ratio

$$\rho_t = \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$$

에 의해 weighted 돼야 한다.
간단히 위 방정식의 오른쪽 부분만 weight 하는 유혹을 받겠지만, 더 좋은 방법이 있다.
time t 에서의 action 이 절대 π에 의해 선택되지 않는다 치면 ϱ_t는 0이다.
그럼 simple weighting 은 n-step return 을 0으로 만들거고, 이건 target 으로 사용됐을 때 high variance를 초래한다.
대신, n-step return 의 off-policy definition 을 써 보자.

$$G_{t:h} \doteq \rho_t \left(R_{t+1} + \gamma G_{t+1:h}\right) + (1 - \rho_t) V_{h-1}(S_t), \qquad t < h \leq T, \tag{7.10}$$

이렇게.
where $G_{t:t} \doteq V_{t-1}(S_t)$.
ϱ_t 가 0이면, target 이 0이고 estimate 가 shrink 되는 대신,
target 이 estimate 랑 같고 estimate 가 변화가 없을거다.
importance sampling ratio 가 0이라는 건 sample 을 무시해야함을 의미해.
estimate 가 바뀌지 않도록 두는 게 좋아 보인다.

(7.10)의 두 번째, 추가적인 term 은 모호한 이유로 control variate 라 불린다.
control variate 가 expected update 를 바꾸지 않는 걸 보렴.
importance sampling ratio 가 expected value one 을 가지고, estimate 와 uncorrelated 이다.
그래서 control variate 의 expected value 는 0이다.
off-policy definition (7.10) 은 earlier on-policy definition of the n-step return(7.1)의 strict generalization 이다. ϱ_t 가 항상 1인 on-policy case 에선 둘이 identical 하기 때문이다.

conventional n-step method 에서,
(7.10)과 함께 사용할 learning rule 은 n-step TD update(7.2)로,
return 에 embedded 돼 있는 것들 말곤 explicit importance sampling ratios 가 없다.

(Exercise 7.4)
위에 나온 off-policy state-value prediction algorithm 의 수도코드를 써보렴.

action values에 대해,
n-step return의 off-policy definition 은 조금 다르다.
왜냐면 첫 action 이 importance sampling 에서 아무 역할도 하지 않기 때문이다.
우린 그 action 의 value를 학습하고 있고, target policy 하에서 unlikely 하거나 impossible 하더라도 중요하지 않다.
action 은 취해졌고 이제 full unit weight 가 뒤따르는 reward 와 state 에 주어져야한다.
importance sampling 은 뒤따르는 actions 에만 적용될 거다.

natural control variate 를 포함한, action values 에 대한 n-step return 의 off-policy recursive definition 은

$$G_{t:h} \doteq R_{t+1} + \gamma \left(\rho_{t+1} G_{t+1:h} + \bar{Q}_{t+1} - \rho_{t+1} Q_t(S_{t+1}, A_{t+1}) \right), \quad t+1 < h \leq T, \tag{7.11}$$

with

$$\bar{Q}_{t+1} \doteq \sum_a \pi(a|S_{t+1}) Q_t(S_{t+1}, a). \tag{7.12}$$

이렇게 된다.
$h = T$ 이면, (7.11)의 recursion 은 $G_{T-1:T} \doteq R_T$로 끝나고,
$h < T$ 이면, recursion 은 $G_{h-1:h} \doteq R_h + \gamma Q^{\text{bar}}{}_h$ 로 끝난다.
그에 따른 결과로 나온 prediction algorithm((7.5)랑 합친 후) 는 Expected Sarsa와 유사하다.

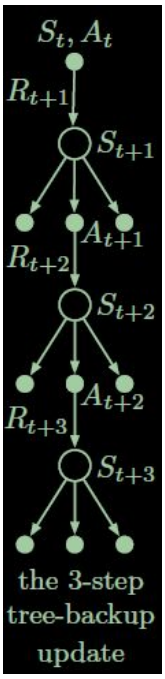
(Exercise 7.5)
(Exercise 7.6)
(Exercise 7.7)
(Exercise 7.8)

여기서, 또 이전 section, 그리고 챕터 5에서 쓴 importance sampling 은 좋은 off-policy learning 을 가능하게 해준다.
하지만 high variance updates 를 초래하기도 한다.
small step-size parameter 를 쓰도록 강제하고, 그래서 학습이 느려지게 한다.
off-policy 가 on-policy training 에 비해 느린 건 피할 수 없을거다.
결국 data 는 무엇이 학습되고 있는지에 덜 상관이 있다.
하지만 이런 methods 들을 개선할 수도 있다.
control variates 는 variance 를 줄이는 한 가지 방법이다.
다른 하나는 observed variance에 step sizes 를 빠르게 적용하는 거다. Autostep method에서처럼. 논문이 있네?
다른 유망한 approach 는 또 다른 논문이 있네?
논문이 하나 더 있고.
다음 section 에서 우리는 off-policy learning 인데 importance sampling 을 사용하지 않는 method 를 보겠다.

7.5 Off-policy Learning Without Importance Sampling: The n-step Tree Backup Algorithm

off-policy learning 이 importance sampling 없이도 가능할까?
챕터 6의 Q-learning 과 Expected Sarsa 가 one-step case 에서 이걸 한다.
하지만 거기 대응하는 multi-step algorithm 이 있나?
이 section 에서 그 n-step method 를 보여줄게.
tree-backup algorithm 이야.

idea는 3-step tree-backup backup diagram 에 의해 제안됨.



central spine 을 따라 내려가면 3 sample states, rewards, 그리고 2 sample actions 이 있다.
얘들은 random variables 인데, initial state-action pair S_t, A_t 이후 발생하는 events 를 표현한다. 측면에 달려있는 건 선택되지 않은 actions 이다.
마지막 state 에선 모든 actions 이 아직 선택되지 않은 걸로 간주.
unselected actions 에 대한 sample data는 없기 때문에,
update 를 위한 target 을 형성하는 데 그들의 values 의 estimates 를 bootstrap 하고 쓴다.
이건 backup diagram 의 idea를 살짝 확장한 거다.

지금까지 우리는 항상 diagram의 꼭대기 노트의 estimated value 부터 update 해서 rewards 를 결합하는? target 까지 갔다. 적절히 discount 해서.
bottom 노드의 estimated value 까지 해서.
대충 뭔 말인지 알겠나?
tree-backup update에선, target 은 이런 모든 것들과 달려있는 모든 action 의 estimated value 까지 포함한다. 모든 levels 에서.
이게 tree-backup update 라고 부르는 이유다.
estimated action values 의 전체 트리로부터 update 한다.

더 정확하게는, update가 트리의 leaf nodes의 estimated action values 로부터 나온다.
실제로 취해진 actions 에 대응하는 내부의 action nodes 는 끼지 않는다.
각 leaf node 는 target policy π 하에서 일어나는 확률에 비례하는 가중치로 target 에 기여한다.
그래서 각 first-level action a 가 $\pi(a|S_{t+1})$ 의 weight 로 contribute.
실제 일어난 action A_{t+1} 은 전혀 contribute 하지 않는다.
그 확률 $\pi(A_{t+1}|S_{t+1})$ 은 모든 second-level action values 를 weight 하는 데 사용된다.
그래서, 각 non-selected second-level action a' 은 weight $\pi(A_{t+1}|S_{t+1})\pi(a'|S_{t+2})$ 로 contribute.
각 third-level action 은 weight $\pi(A_{t+1}|S_{t+1})\pi(A_{t+2}|S_{t+2})\pi(a''|S_{t+3})$ 로 contribute.
그렇게 계속.
이건 마치 diagram 에서 action node 로 향하는 각 화살표가 target policy 하에서 action이 선택될 확률로 weight 되는 것이다.
그리고 action 아래에 tree 가 있으면 그 weight 는 트리의 모든 leaf nodes 에 적용된다.

3-step tree-backup update 가 6 half-steps 로 이루어져있다고 볼 수 있다.
action 에서 다음 state 로 가는 sample half-steps 와
그 state에서 policy 를 따르는 actions의 확률로 발생하는 모든 가능한 actions 을 고려하는 expected half-steps
를 교대로 하는 6 half-steps.

이제 n-step tree-backup algorithm 의 방정식을 만들어 보자.
one-step return(target) 은 Expected Sarsa 의 그것과 같다.

$$G_{t:t+1} \doteq R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q_t(S_{t+1}, a), \qquad t < T-1, \tag{7.13}$$

two-step tree-backup return 은

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

latter form 은 tree-backup n-step return 의 일반적인 recursive definition 을 준다:

$$G_{t:t+n} \doteq R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{t+n-1}(S_{t+1}, a) + \gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:t+n}, \quad t+1 < T, n > 1, \tag{7.14}$$

n = 1일 땐 $G_{T-1:T} \doteq R_T$ 와 (7.13) 적용.
그런 다음 이 target 은 n-step Sarsa 의 usual action-value update rule 과 함께 쓰인다.

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha \left[G_{t:t+n} - Q_{t+n-1}(S_t, A_t) \right], \qquad 0 \leq t < T, \tag{7.5}$$

모든 다른 state-action pairs 의 values 가 unchanged 로 남아있는 동안:
$Q_{t+n}(s, a) = Q_{t+n-1}(s, a)$ for all s, a s.t $s \neq S_t$ or $a \neq A_t$

수도코드는 아래 박스.

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 each step of episode, $t = 0, 1, 2, \ldots$:
    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 $\tau + 1$:
        $G \leftarrow R_k + \gamma \sum_{a \neq A_k} \pi(a|S_k) Q(S_k, a) + \gamma \pi(A_k|S_k) G$
      $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha [G - Q(S_\tau, A_\tau)]$
      If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt Q
  Until $\tau = T - 1$

(Exercise 7.9)

7.6 *A Unifying Algorithm: n-step $Q(\sigma)$

지금까지 이 챕터에서 3 가지 다른 종류의 action-value algorithms 을 봤다.
아래 Fig 7.5 에 처음 세 개의 backup diagrams 이 바로 그거시다.
n-step Sarsa 는 all sample transitions 을 갖고,
tree-backup algorithm 은 fully branched, without sampling 인 all state-to-action transitions 을 갖는다.
n-step Expected Sarsa 는 all sample transitions 을 갖는데, 마지막 state-to-action 만 expected value 로 fully branched 이다.
얘들을 어떻게 통합하냐..

한 idea 는 fourth backup diagram 이다. Fig 7.5 의 젤 오른쪽.

이게 뭐냐면, Sarsa에서처럼 action 을 sample 로서 취할지, tree-backup update 에서처럼 모든 actions 으로 expectation 할 지 단계별로 결정할 수 있다는 거지.
그러면, 만약 항상 sample 하길 선택한다면, Sarsa 를 얻을거고
절대 sample 안 할거면 tree-backup algorithm 을 얻을거야.
마지막만 빼고 모든 steps 에서 sample 하면 Expected Sarsa.
물론 다른 많은 가능성들이 있다.
sampling 과 expectation 의 continuous variation 을 생각하면 더 가능성이 커지지.
0에서 1 사이 값인 σ_t를 step t에서의 degree of sampling 이라고 하자.
1이면 full sampling. 0이면 sampling 없는 pure expectation
σ_t는 random variable 이고, time t 에서의 state, action, state-action pair 의 함수로 설정될 수 있다.
이걸 n-step $Q(\sigma)$ 라고 부른다.

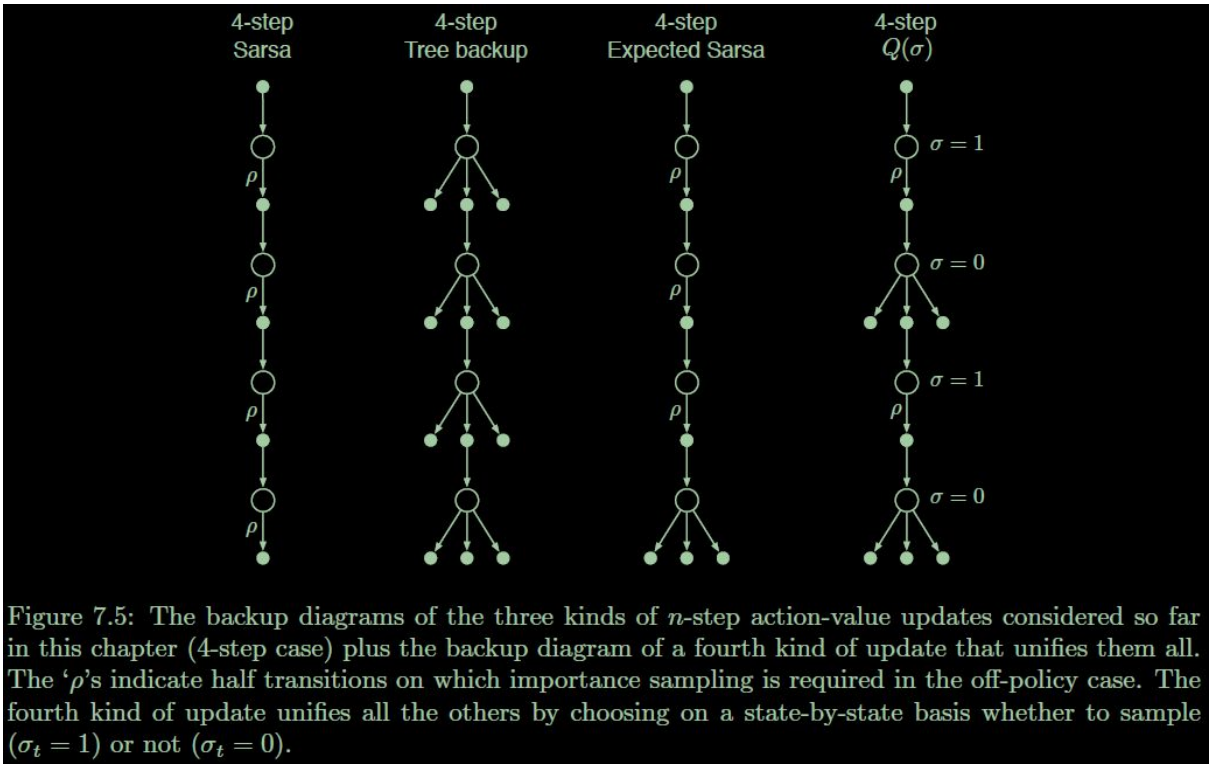


Figure 7.5: The backup diagrams of the three kinds of n-step action-value updates considered so far in this chapter (4-step case) plus the backup diagram of a fourth kind of update that unifies them all. The 'ρ's indicate half transitions on which importance sampling is required in the off-policy case. The fourth kind of update unifies all the others by choosing on a state-by-state basis whether to sample ($\sigma_t = 1$) or not ($\sigma_t = 0$).

Fig 7.5: n-step action-value updates 들이다. 마지막 게 모두를 통합하는 거.
ϱ는 off-policy case 에서 importance sampling 이 필요한 half transitions 을 가리킨다.
네 번째 종류의 update가 sample 을 할 지 (σ_t = 1) 안 할지(σ_t = 0) state-by-state basis 로 선택해서 다른 모든 걸 통합한다.

n-step $Q(\sigma)$ 방정식을 만들어볼까?
Sarsa 의 n-step return (7.4)은 그 자체의 pure-sample-based TD error 관점에서 쓸 수 있다:

$$G_{t:t+n} = Q_{t-1}(S_t, A_t) + \sum_{k=t}^{\min(t+n,T)-1} \gamma^{k-t} \left[R_{k+1} + \gamma Q_k(S_{k+1}, A_{k+1}) - Q_{k-1}(S_k, A_k) \right] \tag{7.15}$$

TD error 를 σ_t 를 써서 expectation 에서 sampling form 으로 바꾸도록 일반화하면
두 cases 모두 다룰 수 있게 해준다:

$$\delta_t \doteq R_{t+1} + \gamma \left[\sigma_{t+1} Q_t(S_{t+1}, A_{t+1}) + (1 - \sigma_{t+1}) \bar{Q}_{t+1} \right] - Q_{t-1}(S_t, A_t), \tag{7.16}$$

이를 이용해 $Q(\sigma)$ 의 n-step returns 을 다음과 같이 정의한다:

$$
\begin{aligned}
G_{t:t+1} &\doteq R_{t+1} + \gamma\big[\sigma_{t+1} Q_t(S_{t+1}, A_{t+1}) + (1-\sigma_{t+1})\bar{Q}_{t+1}\big] \\
&= \delta_t + Q_{t-1}(S_t, A_t), \\
G_{t:t+2} &\doteq R_{t+1} + \gamma\big[\sigma_{t+1} Q_t(S_{t+1}, A_{t+1}) + (1-\sigma_{t+1})\bar{Q}_{t+1}\big] \\
&\quad - \gamma(1-\sigma_{t+1})\pi(A_{t+1}|S_{t+1}) Q_t(S_{t+1}, A_{t+1}) \\
&\quad + \gamma(1-\sigma_{t+1})\pi(A_{t+1}|S_{t+1})\big[R_{t+2} + \gamma\big[\sigma_{t+2} Q_t(S_{t+2}, A_{t+2}) + (1-\sigma_{t+2})\bar{Q}_{t+2}\big]\big] \\
&\quad - \gamma\sigma_{t+1} Q_t(S_{t+1}, A_{t+1}) \\
&\quad + \gamma\sigma_{t+1}\big[R_{t+2} + \gamma\big[\sigma_{t+2} Q_t(S_{t+2}, A_{t+2}) + (1-\sigma_{t+2})\bar{Q}_{t+2}\big]\big] \\
&= Q_{t-1}(S_t, A_t) + \delta_t \\
&\quad + \gamma(1-\sigma_{t+1})\pi(A_{t+1}|S_{t+1})\delta_{t+1} \\
&\quad + \gamma\sigma_{t+1}\delta_{t+1} \\
&= Q_{t-1}(S_t, A_t) + \delta_t + \gamma\big[(1-\sigma_{t+1})\pi(A_{t+1}|S_{t+1}) + \sigma_{t+1}\big]\delta_{t+1} \\
G_{t:t+n} &\doteq Q_{t-1}(S_t, A_t) + \sum_{k=t}^{\min(t+n-1, T-1)} \delta_k \prod_{i=t+1}^{k} \gamma\big[(1-\sigma_i)\pi(A_i|S_i) + \sigma_i\big]. \qquad (7.17)
\end{aligned}
$$

on-policy training 에서, 이 return 은 n-step Sarsa (7.5) 와 같은 update 에 사용될 준비가 됐다.

off-policy case 에선, σ를 importance sampling ratio 에서 고려해야 한다.
더 일반적으로 재정의하면

$$
\rho_{t:h} \doteq \prod_{k=t}^{\min(h, T-1)} \left(\sigma_k \frac{\pi(A_k|S_k)}{b(A_k|S_k)} + 1 - \sigma_k\right). \qquad (7.18)
$$

이후 n-step Sarsa(7.9)에 대한 보통의 일반적인 (off-policy) update 를 쓸 수 있다.
다음이 complete algorithm.

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

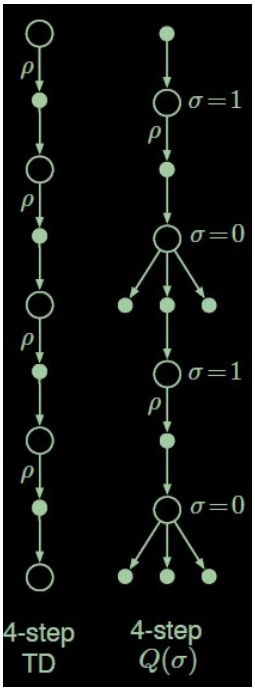
Input: an arbitrary behavior policy b such that $b(a|s) > 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be ε-greedy with respect to Q, or as a fixed given policy
Algorithm parameters: step size $\alpha \in (0,1]$, 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
  Select and store an action $A_0 \sim b(\cdot|S_0)$
  Store $Q(S_0, A_0)$ as Q_0
  $T \leftarrow \infty$
  Loop for each step of episode, $t = 0, 1, 2, \ldots$:
    If $t < T$:
      Take action A_t
      Observe the next reward R; observe and store the next state as S_{t+1}
      If S_{t+1} is terminal:
        $T \leftarrow t+1$
        Store $\delta_t \leftarrow R - Q_t$
      else:
        Select and store an action $A_{t+1} \sim b(\cdot|S_{t+1})$
        Select and store σ_{t+1}
        Store $Q(S_{t+1}, A_{t+1})$ as Q_{t+1}
        Store $R + \gamma\sigma_{t+1}Q_{t+1} + \gamma(1-\sigma_{t+1})\sum_a \pi(a|S_{t+1})Q(S_{t+1},a) - Q_t$ as δ_t
        Store $\pi(A_{t+1}|S_{t+1})$ as π_{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 $\tau \geq 0$:
      $\rho \leftarrow 1$
      $Z \leftarrow 1$
      $G \leftarrow Q_\tau$
      Loop for $k = \tau, \ldots, \min(\tau + n - 1, T - 1)$:
        $G \leftarrow G + Z\delta_k$
        $Z \leftarrow \gamma Z\left[(1-\sigma_{k+1})\pi_{k+1} + \sigma_{k+1}\right]$
        $\rho \leftarrow \rho(1 - \sigma_k + \sigma_k\rho_k)$
      $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\rho\left[G - Q(S_\tau, A_\tau)\right]$
      If π is being learned, then ensure that $\pi(a|S_\tau)$ is ε-greedy wrt $Q(S_\tau, \cdot)$
  Until $\tau = T - 1$

(Exercise 7.10)
(Exercise 7.11)

7.7 Summary
드디어 7단원이 끝나네.
이번 챕터에선 temporal-difference learning methods 의 range 를 develop 해 봤다.
이전 챕터의 one-step TD methods와 그 전의 Monte Carlo methods 사이에 뉘여져 있지.
bootstrapping 의 intermediate amount 를 포함하는 methods 는 아주 중요.
보통 양 극단보다 성능이 좋기 때문!

이 챕터에서 우리의 초점은 n-step methods 에 맞춰져 있었다.
다음 n rewards, states, actions 를 보는 방법이었지.



두 4-step backup diagrams 은 소개된 대부분의 methods 를 요약하고 있다.
state-value update 는 n-step TD with importance sampling 에 대한 것이고,
action-value update 는 n-step $Q(\sigma)$ 대한 것이다. Expected Sarsa 와 Q-learning 을 일반화하는 n-step $Q(\sigma)$ 말이다.

모든 n-step methods 는 update 하기 전에 n time steps 의 delay 가 있다.
그때서야 모든 필요한 future events 를 알게 되기 때문이지.
더 큰 결함은 이전의 methods 보다 time step 당 계산이 더 많다는 것이다.
또 one-step methods 에 비해 n-step methods 는 states, actions, rewards 를 기록한 memory 가 필요하다. 가끔은 마지막 n steps 에 대한 다른 변수까지 포함할 수도 있지.
챕터 12에서 우리는 어떻게 multi-step TD methods 가 minimal memory와 computational complexity 로 적용될 수 있을지 볼 거다. eligibility traces 로.
하지만 항상 one-step methods를 넘는 추가적인 계산은 있을 거야.
그래도 그만한 가치가 있지.

n-step methods 가 eligibility traces 를 쓰는 애들보다 복잡하긴 해도,
개념적으로 clear 하다는 엄청난 장점이 있어.
n-step case 에서 off-policy learning 에 대한 두 가지 접근법을 만들어 이를 활용하고자 한다.
하나는, importance sampling 에 기반한 건데, 개념적으로 간단하지만 high variance 일 수 있지.
target 과 behavior policies 가 많이 다르면 효율성과 실용성을 위해 새로운 idea 가 필요할 거야.
다른 하나는, tree-backup updates 에 기반한 건데, stochastic target policies 를 쓰는 Q-learning 을 multi-step case에 자연스럽게 확장한거야.
importance sampling 은 안 쓰지만, 마찬가지로 target 과 behavior policies 가 많이 다르면, n 이 크더라도 bootstrapping 이 몇 steps 만 span 할 거야.

$\alpha\gamma r\varepsilon\varrho\lambda\delta\tau\phi\sigma$
$\pi(a|s)\ b \neq \pi$
$v_\pi(s)\ Q(s,\ a)$
$q_\pi(s,\ a)$
$ACSQRTVWGtabpn\hbar ijk\boldsymbol{h}t{+}n$
$\mathscr{ASJ}$
$\ni\in\geq\leq\neq\doteq\Leftrightarrow\longleftarrow\ s\in\mathscr{S},\ a\in\mathscr{A}(s)\mathscr{R}$
$\blacksquare'\cdot$
$\varrho_{t:T(t)\text{-}1}$