5. Monte Carlo Methods

이 챕터에서 우리의 첫 번째 learning methods를 살펴볼 것이다. value functions을 estimate 하고 optimal policies를 discover 하는 learning methods.
이때까지 한 건 뭐였쥬?
이전 챕터와 다르게, 여기선 environment에 대한 complete knowledge를 가정하지 않는다.
Monte Carlo methods는 오직 experience만 필요로 한다.
experience: sample sequences of states, actions, rewards 인데, actual or simulated interaction with an environment 에서 얻은 것.

actual experience에서 learn 한다는 것은 striking 하다.
environment 의 dynamics에 대한 사전 지식이 필요하지 않지만, 여전히 optimal behavior를 얻을 수 있기 때문이다.
simulated experience 에서 learn 한다는 것 또한 powerful 하다.
model이 필요하긴 하지만, model 은 오직 sample transition만 생성하면 되고,
DP에서 필요했던 것처럼 모든 가능한 transitions의 complete probability distributions은 필요하지 않다.
많은 cases에서, 원하는 확률 분포에 따라 샘플링된 experience를 생성하는 건 쉽지만,
명확한 형태(explicit form)로 분포를 얻는 건 불가능하다.

Monte Carlo methods는 sample returns를 averaging 하는 데 기초해서 강화학습 문제를 푸는 방법이다.
well-defined returns 가 available 함을 확실하게 하기 위해,
여기서는 episodic tasks 에 대해서만 Monte Carlo methods를 정의한다.
그 말은, experience가 episodes로 나눠진다는 걸 가정한다는 말이고,
모든 episodes가 어떤 actions이 선택되든 결국 terminate 된다는 말이다.
episode가 완료됐을 때만 value estimates 와 policies가 변경된다.
그래서 Monte Carlo methods는 episode-by-episode 로 incremental 할 수 있다.
step-by-step(online) 이 아니라.
“Monte Carlo” 라는 용어는 넓게 쓰인다. significant random component 를 포함하는 operation을 사용하는 모든 estimation method에서.
여기서는 구체적으로, averaging complete returns에 기반한 methods에 이 용어를 쓰도록 하겠다. (partial returns로부터 학습하는 methods와 반대로)

Monte Carlo methods는 각 state-action pair 의 returns 을 sample & average 한다.
Ch2 의 bandit methods에서 각 action에 대해 rewards를 sample & average 한 것처럼.
주된 차이는, 지금은 multiple states가 있고, 각각이 서로 다른 bandit problem 처럼 행동한다는 것이다.(associative-search나 contextual bandit 처럼)
그리고 서로 다른 bandit 문제들은 interrelated 돼 있다.
그 말은, 한 state 에서 action을 한 후의 return 이, 같은 episode의 later states 에서 취한 actions에 depend 하다는 말.
모든 action selections이 학습을 겪고, 문제가 earlier state의 관점에서 nonstationary 가 되기 때문이다.

nonstationarity를 다루려면, general policy iteration(GPI) 아이디어를 적용해야 한다.
4챕터에선 value functions 을 MDP 의 knowledge로부터 compute했다면,

여기서는 sample returns with the MDP 로부터 value functions을 learn 한다.
value functions 과 그에 대응하는 policies는 여전히 상호작용한다. optimality를 얻기 위해.
GPI 와 본질적으로 같은 방식으로.
DP 챕터에서처럼, 먼저 prediction problem 을 먼저 본다. (v_π , q_π 계산. fixed arbitrary policy π에서)
그러고 policy improvement를 보고,
마지막으로, control problem 과 GPI 에 의한 그 solution을 보겠다.
DP로부터 가져온 이들 각 ideas는 sample experience만 available한 Monte Carlo case에 확장된다.

5.1 Monte Carlo Prediction

주어진 policy에서 state-value function을 학습하기 위한 MC methods를 생각해 보자.
state 의 value 는 expected return이다. 리멤버?
정확하게는 expected cumulative future discounted reward 지. 그 state에서부터 시작하는.
그러면, experience로 이걸 estimate 하는 확실한 방법은, 단순히 해당 state 방문 이후에 관측된 returns 을 average 하는 거다.
더 많은 returns이 관측될수록, average가 expected value 에 converge한다.
이 idea가 모든 Monte Carlo methods에 underlie 하고 있다.

특히, v_π(s) 를 estimate 하고 싶다고 상상해 봐. v_π(s) 는 policy π 하에서 state의 value지.
주어진 episodes set (following π 와 passing through s에 의해 얻어진 episodes set) 에서.
episode에서 각 state s의 발생은 *visit* to s 라고 부른다.
당연히, s는 한 episode에서 여러 번 visit 될 수 있다.
한 episode에서 처음 visit 된 걸 first visit to s 라고 부르자.

first-visit MC method는 first visits to s 이후의 returns 의 average 로 v_π(s) 를 estimates 한다.
반면, every-visit MC method 는 모든 visits to s 에 따르는 returns 을 average 한다.
두 MC methods는 매우 비슷하지만 이론적 properties가 조금 다르다.
First-visit MC 가 1940년대에 넓게 연구됐고, 이 챕터에서 중점적으로 다룰 거다.
Every-visit MC 는 function approximation 과 eligibility traces 로 더 자연스럽게 확장된다.
그건 챕터 9, 12에서 얘기할 것.
First-visit MC 가 procedural form 으로 아래 box에 나와 있다.
Every-visit MC는 S_t 가 episode 에서 이전에 발생했는지 확인하는 것 외엔 같을 거다.

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated
Initialize:
  $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$
  $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):
  Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow G + R_{t+1}$
    Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:
      Append G to $Returns(S_t)$
      $V(S_t) \leftarrow$ average$(Returns(S_t))$

first-visit MC나 every-visit MC나 v_π(s) 로 converge 한다.
visits(or first visits) to s 횟수가 무한대로 가면.
first-visit MC case 가 보기 쉽다.
이 case에서 각 return은 finite variance의 IID estimate of v_π(s) 이다.
큰 수의 법칙에 의해, 이 estimates의 averages 의 sequence는 그들의 expected value로 converge한다.
각 average는 unbiased estimate이고 error의 standard deviation은 $1/\sqrt{n}$ 이다.
n은 average된 returns 수.
Every-visit MC 는 less straightforward 하지만, estimates는 quadratically v_π(s) 로 수렴한다.

MC methods의 use는 example 로 보는 게 best.

(Example 5.1) Blackjack
blackjack은 합이 21을 넘지 않는 가장 큰 수의 카드를 얻는 게임이다.
모든 face cards는 10으로 치고, ace는 1이나 11로 친다.
각 player가 dealer와 독립적으로 붙는 버젼을 보자.
게임은 dealer와 player가 두 장의 카드를 갖고 시작한다.
dealer의 카드 중 하나를 face up 하고 다른 건 face down.
player가 21을 바로 가지면 natural이라고 한다. 그럼 dealer가 natural이 아닌 이상 이긴다.
같이 natural이면 draw.
player가 natural이 아니면, 추가 cards를 요구할 수 있다.
하나씩(hits), 그가 stop 할 때까지(sticks), 또는 21 넘을 때까지(goes bust)
bust 되면 지는 거. stick하면 dealer 턴.
dealer 는 선택권 없이 고정된 전략에 따라 hit or stick 한다.
17 이상이면 stick, 아니면 hits.
dealer가 bust 하면 player win.
그 외엔 final sum이 21에 가까운 사람이 이긴다.
blackjack은 episodic finite MDP 이다.
blackjack의 각 game은 episode 다.
rewards는 +1, -1, 0 이 winning, losing, drawing에 각각 주어진다.
game 내의 모든 rewards 는 0이고 discount 하지 않는다. (γ = 1)
그러므로 terminal rewards 도 returns 이다.
actions 은 hit or stick.
states 는 player의 cards 와 dealer의 showing card에 depend.

infinite deck with replacement 로 가정하자. 이미 나온 카드를 추적하는 게 소용없게.
player가 ace를 11로 카운트해도 bust 되지 않는 선에서 갖고 있으면, ace 가 usable 이라 부름
이 case에서는 항상 11로 count 한다. 왜냐면 1로 count하면 sum이 11 이하가 되니 player가 무조건 hit 할 거기 때문. 이게 뭔 말일까.
그래서, player 는 3 변수에 근거해서 결정을 내린다.
player's current sum (12-21)
dealer's one showing card (ace-10)
player가 usable ace를 들고 있는지.
이러면 총 200 states가 나온다.
player's sum이 20 or 21 이면 stick, 아니면 hit 하는 policy를 생각해 보자.
MC approach로 이 policy에 대한 state-value function 을 찾으려면,
이 policy 를 이용해 많은 game 을 simulate 하고 각 state에 따른 returns 을 average 한다.
이 task에서 한 episode 안에서 같은 state는 다시 발생하지 않는다.
그래서 first-visit 과 every-visit MC methods는 차이가 없다.
이 방법으로, state-value function의 estimates를 얻을 수 있고, (Fig 5.1) 에 나와 있다.
usable ace 가 있는 states의 estimates 는 less certain, less regular 하다.
왜냐면 이런 states 는 less common 하기 때문.
어떤 event든 500,000 games 이후엔 value function이 매우 잘 approximate 된다.

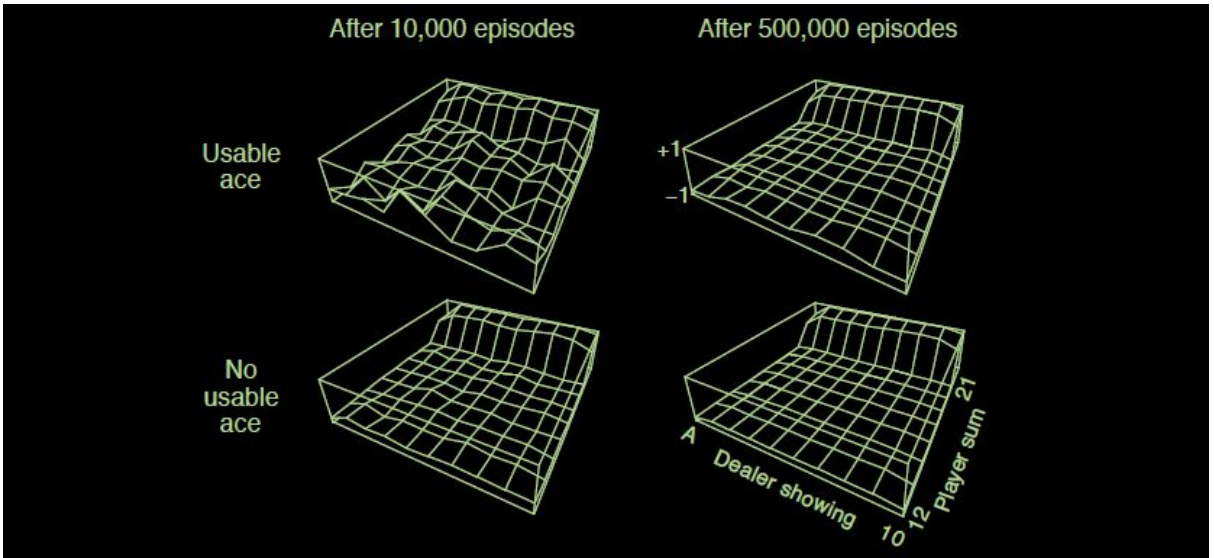


Figure 5.1: Approximate state-value functions for the blackjack policy that sticks only on 20 or 21, computed by Monte Carlo policy evaluation.

(Exercise 5.1) (Fig 5.1) 오른쪽의 diagrams 을 보자. 왜 estimated value function이 마지막 두 rows 에서 jump up 하는 걸까? 왜 왼쪽의 마지막 row 에서 drop off 하는거야?
왜 frontmost values 가 lower 보다 upper diagrams 에서 더 높니?

(Exercise 5.2) first-visit MC 대신 every-visit MC가 쓰였다고 쳐 보자. 결과가 많이 다를까? 이유는?

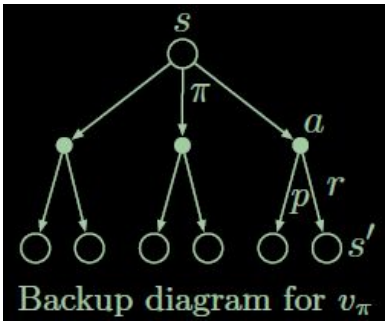
blackjack 에선 environment에 대해 complete knowledge 가 있었지만, value function을 계산하는 데 DP methods 를 적용하는 건 쉽지 않다.
DP methods는 next events의 distribution이 필요하다. 특히, four-argument function p 에 의해 주어진 environments dynamics 가 필요하다.
그리고 blackjack에 대해 이걸 결정하는 건 쉽지 않다.

예를 들어, player의 sum 이 14 이고 stick 하기로 했다하자.
dealer의 showing card 의 함수로, +1 reward를 받고 종료할 확률은 얼마일까?
모든 확률은 DP 가 적용되기 전에 계산돼야하고, 그 계산은 주로 복잡하고 에러나기 쉽다.

이와 대조적으로, MC methods에서 필요로 하는 sample games 을 생성하는 건 쉽다.
이런 case는 매우 자주 일어나며, MC methods의 sample episodes 만으로 작업할 수 있는 ability 아주 큰 장점이 될 수 있다. environment's dynamics에 대한 complete knowledge를 갖고 있어도.

MC algorithms의 backup diagrams idea 를 일반화할 수 있을까?
backup diagram의 general idea 는,
위에 update될 root node 를 표시하고,
아래엔 모든 transitions, 그리고 rewards 와 estimated values 가 update 에 기여하는 leaf nodes 를 표시한다.
v_π 의 MC estimation 은, root가 state node 이고, 아래엔 terminal state 에서 끝나는 특정 single episode 에 따른 전체 trajectory 가 나온다.

DP diagram 은 모든 transition을 보여주는 반면,
MC diagram은 한 episode에서 sample 된 애들만 보여준다.



왼쪽이 MC, 오른쪽이 DP

DP가 one-step transitions 만 포함하고 있는 반면,
MC diagram은 episode의 끝을 향해 간다.

이런 diagrams 에서의 차이점들은 알고리즘간의 근본적 차이를 정확하게 reflect한다.

MC methods 의 중요한 사실은, 각 state에 대한 estimates가 independent 하다는 것.
한 state 에 대한 estimate 는 다른 state의 estimate 위에 세워지는 게 아니다.

DP의 경우와 같이.
다르게 말하면, MC methods 는 bootstrap 하지 않는다. 이전 챕터에서 정의했던 것처럼.

특히, single state의 value를 estimate 하는 계산 비용은 states 수에 independent 하다.
이건 MC methods를 특히 attractive하게 한다. 한 state 나 states의 subset의 value만 필요할 때.
관심을 갖고 있는 states에서 시작하면서, 이 states 에서 얻은 returns 을 averaging 하고, 다른 애들은 무시하는 sample episodes를 많이 생성할 수 있다.
이게 DP methods 보다 MC methods가 더 좋은 세 번째 장점이다.
actual and simulated experience 에서 학습하는 ability 이후에?

(Example 5.2) Soap Bubble
wire frame 이 closed loop 를 형성하고 있다. 얘를 비눗물에 담가 비눗물 표면이나 와이어 가장자리에 거품을 형성한다고 해 보자.
wire frame의 geometry가 irregular 하지만 known 이면, surface 의 shape을 어떻게 계산할래?
shape은, neighboring points가 각 point 에 주는 총 힘이 0 (아니면 shape이 바뀌겠지) 이라는 특성이 있다.
이건 어떤 점에서든 surface 의 height 가 그 점에서 조그만 원을 그렸을 때 height 들의 평균이라는 뜻.
더해서, surface는 boundaries 에서 wire frame 과 만나야 한다.
이런 종류의 문제에 usual approach는, surface로 덮인 영역 위에 grid 를 놓고 iterative computation으로 grid points에서 height를 구하는 거다.
boundary 의 grid points 는 wire frame에 강제되고? 다른 모든 점들은 가장 가까운 4 점의 평균 높이로 조정된다.
이 process를 반복한다. DP 의 iterative policy evaluation 처럼.
그리고 궁극적으로 desired surface 의 close approximation에 converge 한다.
이건 뭔 예제여..

이건 MC methods 가 처음 design 됐을 때의 문제들과 비슷하다.
위에서 말한 iterative computation 대신,
surface 위에 서서 random walk 한다고 상상해봐. grid point 에서 neighboring grid point로. equal probability로. boundary에 도착할 때까지.
경계에서 height의 expected value는,
시작점에서 desired surface의 height에 대한 close approximation이다.
(사실, 정확히 위에서 말한 iterative 방법으로 계산한 value 이다.)
그래서, point에서의 surface의 height를 closely approximate 할 수 있다. 그 point 에서 시작한 많은 walks 의 boundary heights를 averaging 해서.
한 point, 혹은 points 의 fixed small set 에서의 value 만 관심있으면,
이 MC method 가 local consistency에 기반한 iterative method 보다 더 효율적이다.

5.2 Monte Carlo Estimation of Action Values

model이 not available이면, action values (the values of state-action paris) 를 estimate 하는 게 state values 를 하는 것보다 특히 useful 하다.
model 이 있으면, state values만으로 policy를 결정하기에 충분하다.

한 스텝 앞을 내다보고 reward 와 next state의 best 조합으로 이끄는 action 을 선택한다.
DP 에서 했던 것처럼.
하지만 model 이 없으면, state values 만으론 충분하지 않다.
policy를 suggest 할 때 values가 useful 하려면 각 action의 value 를 explicitly estimate 해야한다.
그래서, MC methods 에서 primary goals 중 하나는 q_* 를 추정하는 일이다.
이를 위해, 먼저 action values를 위한 policy evaluation problem을 보자.

action values의 policy evaluation problem은 $q_\pi(s, a)$ 을 estimate 하는 것이다.
$q_\pi(s, a)$: state s 에서 시작해 action a 를 취하고, 이후 policy π 를 따를 때의 expected return.
이를 위한 Monte Carlo methods 는 state values 에 대해 할 때와 본질적으로 같다.
다만 지금은 visits to a state 대신 visits to a state-action pair 에 대해 얘기하는 거다.
state s 가 visit 되고 그 안에서 action a 가 취해지면, state-action pair s, a 는 한 episode 안에서 visit 됐다고 한다.
every-visit MC method 는 이어지는 모든 visits의 returns을 average한 걸로 state-action pair 의 value를 estimate 한다.
first-visit MC method 는 각 episode에서 state 가 visit 되고 action이 select 된 첫 번째에 따른 returns 을 average 한다.
이런 methods 들은 각 state-action pair 에 visit 수가 무한에 가까워질수록 true expected values 에 quadratically converge 한다. 이전처럼.

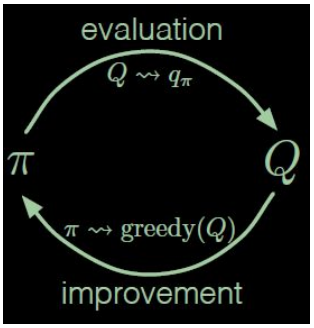
한 가지 상황을 복잡하게 만드는 건, 많은 state-action pairs 가 한 번도 visit 되지 않을 수 있다는 거다.
π 가 deterministic policy 면, π를 따를 때 각 state 에서 actions 중 하나에 대해서만 returns 을 observe 할 것이다.
average 할 returns 이 없다면, 다른 actions의 MC estimates 는 experience에 따라 개선되지 않을거다.
이건 심각한 문제인데, action values 를 학습하는 목적이 각 state에서 available한 actions 중에 고르는 걸 돕는 것이기 때문이다.
alternatives 를 비교하기 위해, 우리는 각 state 에서 모든 actions 의 value를 estimate 해야한다. 우리가 현재 관심있는 하나만 하는 게 아니라.

이건 maintaining exploration의 일반적인 문제다. Ch 2에서 k-armed bandit problem 의 문맥에서 논의했던 것처럼.
policy evaluation 이 action values 에 대해 작동하려면, continual exploration 을 보장해야한다.
이를 하기 위한 한 가지 방법은, episodes가 state-action pair 로 시작하고 모든 pair 가 start 로 선택될 확률이 0이 아니도록 지정하는 것이다.
이게 episodes 수가 무한대가 되는 됨에 따라 모든 state-action pairs 가 무한 번 visit 되는 걸 보장한다.
이걸 exploring starts 의 assumption 이라고 한다.

assumption of exploring starts 는 종종 useful 하지만, 당연히 일반적으로 믿을 순 없다.
특히 environment 와의 actual interaction에서 directly 학습할 때.
그 case에선 starting condition 이 별로 도움이 안 된다.
모든 state-action pairs 가 만난다는 걸 보장하는 가장 일반적인 대체 approach 는,

각 state 에서 모든 actions을 선택할 확률이 0이 아닌 stochastic 한 policy 를 고려하는 것이다.
이 approach의 두 가지 중요한 변형을 뒤 sections 에서 논의한다.
지금은, assumption of exploring starts 를 유지하고, full Monte Carlo control method 를 완성해 보자.

5.3 Monte Carlo Control
우린 이제 준비가 됐다. Monte Carlo estimation 이 control 에서 어떻게 사용될 지 알아볼.
control 은 optimal policies 를 approximate 하는 거인 듯..
전반적인 idea는 DP chapter에서와 같은 pattern에 따라 진행하는 것이다.
그러니까, generalized policy iteration(GPI) 의 idea에 따라서.
GPI 에선 approximate policy 와 approximate value function 을 모두 유지한다.
value function이 current policy 에 대한 value function에 더 가깝게 approximate 하도록 반복적으로 바뀐다.
그리고 policy는 current value function에 대해 반복적으로 개선된다.
고것이 이 그림.



이런 두 종류의 changes 는 어느 정도 서로 work against 한다. 서로가 moving target 을 만들거든. 하지만 함께 optimality 에 다가가게 해.

시작해볼까?
classical policy iteration의 Monte Carlo version을 보자.
이 method에선, policy evaluation 과 policy improvement 의 alternating complete steps를 수행한다. 임의의 policy π_0 에서 시작해 optimal policy 와 optimal action-value function으로 끝나는.

$$\pi_0 \xrightarrow{\text{E}} q_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} q_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} q_{*},$$

policy evaluation 은 앞선 section에서 설명했듯이 정확히 실시된다.
많은 episodes 들이 experienced 된다. approximate action-value function이 true function에 점근적으로 다가가면서.
현재로서는, 무한 개의 episodes 를 관찰한다고 가정하자. 추가로, episodes 가 exploring starts로 생성된다고 하고.
이 가정들 하에서, MC methods는 임의의 π_k 에 대해 각 $q_{\pi k}$ 를 정확히 계산할 것이다.

Policy improvement 는 policy 를 current value function 에 대해 greedy 하게 만들어 수행된다.
이 case에서는 action-value function이 있지.
그러니 greedy policy를 construct 하는 model 이 필요가 없지.

any action-value function q 에 대해, 대응하는 greedy policy는 각 s에 대해 maximal action-value 를 가진 action을 deterministically 선택하는 애다.

$$\pi(s) \doteq \arg\max_a q(s,a).$$

그러면 각 π_{k+1} 를 $q_{\pi k}$ 에 대한 greedy policy 로 construct 함으로써 policy improvement 가 이루어질 수 있다.
그러면 policy improvement theorem (section 4.2) 을 π_k , π_{k+1} 에 적용할 수 있다.
왜냐면, 모든 s에 대해

$$\begin{aligned} q_{\pi_k}(s, \pi_{k+1}(s)) &= q_{\pi_k}(s, \arg\max_a q_{\pi_k}(s,a)) \\ &= \max_a q_{\pi_k}(s,a) \\ &\geq q_{\pi_k}(s, \pi_k(s)) \\ &\geq v_{\pi_k}(s). \end{aligned}$$

이렇기 때문이지.

이전 챕터에서 논의했듯, theorem 이 각 π_{k+1} 이 π_k 보다 uniformly better, or just as good as π_k 함을 보장한다. (같은 건 둘 다 optimal policies 일 때)
이건 결국 전반적인 process가 optimal policy와 optimal value function으로 converge 함을 보장한다.
이 방법으로 MC methods 는 다른 environment's dynamics 에 대한 지식 없이 주어진 sample episodes만 가지고 optimal policy를 찾는 데 사용될 수 있다.

있을 것 같지 않은, unlikely assumption 두 개를 했지?
MC methods 에서 convergence 에 대한 보장을 쉽게 얻기 위해서.
하나는 episodes 가 exploring starts를 갖는다는 거였고,
다른 하나는 policy evaluation 이 infinite number of episodes 에서 수행될 수 있다는거지.
practical algorithm을 얻으려면 두 가정을 없애야 해.
첫 가정을 고려하는 건 챕터에 나중으로 미루자.

지금은 policy evaluation 이 infinite number of episodes 에서 수행된다는 가정에 초점을 맞추자.
이 가정은 상대적으로 없애기 쉽다.
사실, 같은 issue 가 classical DP methods 에서도 발생한다.
iterative policy evaluation 같은 거. true value function에 점근적으로만 converge 한단 말이지.
DP와 MC 모두, 이 문제를 해결하는 두 가지 방법이 있다.
하나는 각 policy evaluation에서 $q_{\pi k}$ 를 approximate 하는 idea를 확실히 고수하는 것이다.
measurements 와 assumptions 은 estimates 에서의 error의 magnitude 와 probability 의 bounds를 얻기 위해 만들어졌다.
그런 다음 이 bounds 가 충분히 작음을 assure 하기 위한 각 policy evaluation 동안 충분한 steps 이 취해진다.
이 approach는 어느 정도의 approximation 까지 정확한 convergence를 보장한다는 의미에서 완전히 만족할 수 있을 거야.
하지만, 실제에 유용하게 적용하기 위해선 작은 문제에도 너무 많은 episodes를 필요로 할 것 같다.

두 번째 approach가 있다.
명목상 policy evaluation에 요구되는 infinite number of episodes를 피하는 approach.
그 policy evaluation은 policy improvement 로 돌아가기 전에 policy evaluation을 마무리하는 걸 포기하는 애야.
각 evaluation step에서 우리는 value function을 $q_{\pi k}$ 로 이동시키지만,
많은 steps 을 밟기 전에 실제로 close 할 거라곤 기대하지 않는다.
우린 이 idea를 GPI idea 를 처음 소개할 때 썼다. 4.6에서.
idea의 한 extreme form 은 value iteration 이다. 그 value iteration 은 iterative policy evaluation의 오직 한 iteration 이 각 policy improvement의 step 사이에 수행된다.
value iteration 의 in-place 버전은 더 extreme 하다.
거기서 우린 single states에 대한 improvement와 evaluation steps 을 번갈아 한다.

Monte Carlo policy evaluation 에서 episode 별로 evaluation과 improvement를 왔다갔다 하는 게 자연스럽다.
각 episode 이후에, observed returns 은 policy evaluation에 이용되고,
그 후 policy는 그 episode에서 visit 된 모든 states 에서 improve 된다.
이런 lines 을 따라 complete simple algorithm을 만들면
Monte Carlo with Exploring Starts 가 나온다. 아래 박스가 설명.

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Initialize:
  $\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$
  $Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
  $Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Loop forever (for each episode):
  Choose $S_0 \in \mathcal{S}$ and $A_0 \in \mathcal{A}(S_0)$ such that all pairs have probability > 0 (exploring starts)
  Generate an episode starting from S_0, A_0, following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow G + R_{t+1}$
    Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
      Append G to $Returns(S_t, A_t)$
      $Q(S_t) \leftarrow \text{average}(Returns(S_t, A_t))$
      $\pi(S_t) \leftarrow \text{argmax}_a\, Q(S_t, a)$

Monte Carlo ES 에서, 각 state-action pair의 모든 returns 은, returns이 관측됐을 때 어떤 policy가 시행 중인지에 상관없이 accumulated, averaged 된다.
Monte Carlo ES 가 어느 suboptimal policy에도 converge 하지 않는 걸 보는 건 쉽다.
만약 converge 한다면, value function 이 그 policy의 value function 에 결국 converge 할 거고, 결국 policy 가 바뀌게 할 거다.
stability 는 policy 와 value function 모두 optimal 할 때만 얻을 수 있다.
이 optimal fixed point 에 Converge 하는 건 inevitable 해 보인다. action-value function 의 변화가 시간에 따라 줄어듦에 따라. 하지만 공식적으로 증명되진 않았다.
우리 의견으로는, 이건 강화학습에서 가장 fundamental open theoretical questions 중 하나다.

(Example 5.3) Solving Blackjack
Monte Carlo ES 를 blackjack에 적용하는 건 straightforward 하다.
왜냐면 episodes 가 모두 simulated games 이고
모든 가능성을 포함한 exploring starts로 arrange 하는 건 쉽기 때문.

이 case에선 dealer's cards, player's sum, player가 usable ace를 갖고 있는지 모두 같은 확률로 random하게 선택한다.
initial policy 로 이전 blackjack example 에서 evaluated 된 policy를 쓴다.
20 이나 21 에서만 stick 했던 거 있잖아.
initial action-value function 은 모든 state-action pairs 에 0 일 수 있다.

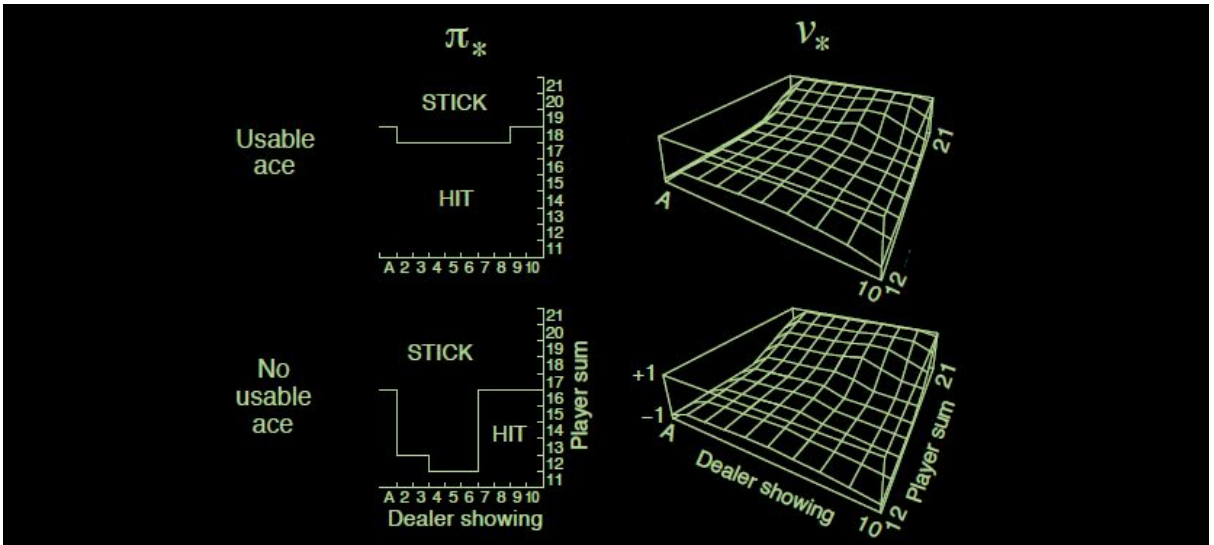


Figure 5.2: The optimal policy and state-value function for blackjack, found by Monte Carlo ES (Figure 5.4). The state-value function shown was computed from the action-value function found by Monte Carlo ES.

(Fig 5.2)는 Monte Carlo Es 로 찾은 blackjack의 optimal policy다.
이 policy 가 Thorp(1966)의 기본 전략과 같다. usable ace의 제일 왼쪽 부분만 빼고.
이 차이가 왜 있는지는 모르겠는데 이게 우리가 설명한 버전의 blackjack의 optimal policy는 확실하다.

5.4 Monte Carlo Control without Exploring Starts

exploring starts 라는 unlikely assumption을 어떻게 피할 수 있을까?
모든 actions 이 infinitely often 하게 선택되는 걸 ensure 하는 only general way는 agent가 계속 actions 을 선택하는 거다.
이걸 ensure 하는 두 가지 approaches 가 있는데,
on- policy methods 와 off-policy methods 라는 결과를 낳는다.

on-policy methods 는 의사 결정에 사용되는 policy를 evaluate 하거나 improve 하려고 한다.
off-policy methods 는 data 를 생성하는 데 사용되는 policy와 다른 policy를 evaluate 하거나 improve 한다.

위에서 얘기한 Monte Carlo ES method 는 on-policy method의 한 예이다.
이 section 에서는 on-policy Monte Carlo control method 가 exploring starts 라는 unrealistic assumption을 사용하지 않고 어떻게 design 되는지 보여줄게 완전히 달라진 나.
off-policy methods 는 다음 section에서 보자.

on-policy control methods 에선, policy가 일반적으로 soft 하다.
무슨 뜻이냐면, 모든 s, a에 대해 $\pi(a|s) > 0$ 하다는 말.
하지만 점차 deterministic optimal policy로 가까이 shift 된다.

챕터 2에서 논의했던 많은 methods 들이 이를 위한 mechanisms 을 제공한다.
이 section에서 보여줄 on-policy method 는 ε-greedy policies 를 쓴다.
대부분 maximal estimated action value 를 갖는 action을 선택하고 ε 확률로 랜덤하게 선택하는 거 알지?
그러니까, 모든 nongreedy actions 은 minimal probability of selection ε / $|\mathcal{A}(s)|$ 을 갖고, greedy action은 나머지 확률인 1 - ε + ε / $|\mathcal{A}(s)|$ 을 갖는다.
ε-greedy policies 는 ε-soft policies 의 예들인데,
모든 states 와 actions, 어떤 ε > 0 에 대해 $\pi(a|s) \geq \varepsilon / |\mathcal{A}(s)|$ 인 policies 로 정의된다.
ε-soft policies 들 중, ε-greedy policies 가 어떤 면에서 greedy 에 가장 가까운 애들이다.

on-policy Monte Carlo control 의 전반적인 idea는 여전히 GPI의 idea다.
Monte Carlo ES 에서처럼, current policy 의 action-value function을 estimate 하는 데 first-visit MC methods 를 쓴다.
하지만 exploring starts 가정 없이는 현재 value function에 대해 greedy하게 만듦으로써 policy를 improve 할 수 없다.
왜냐하면 그건 nongreedy actions에 대한 further exploration 을 막을 테니까.
다행히도, GPI 는 policy 가 완전히 greedy 하길 요구하진 않고 moved toward a greedy policy 하길 요구하는 정도다.
우리의 on-policy method 에선 ε-greedy policy로만 옮길 거다. policy를 옮긴다는 말인가.
For any ε-soft policy π 에서, q_π 에 대한 어떤 ε-greedy policy든 π 보다 더 좋거나 같음이 보장된다.
complete algorithm 은 아래 박스.

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$
Initialize:
  $\pi \leftarrow$ an arbitrary ε-soft policy
  $Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
  $Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):
  Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow G + R_{t+1}$
    Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
      Append G to $Returns(S_t, A_t)$
      $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
      $A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)
      For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

알고리즘 한 번 볼까?
π는 임의의 ε-soft
$Q(s, a)$ 는 임의의 실수.
Returns(s, a) 는 빈 칸이고 채워나갈거다.
각 episode마다 loop 가 영원히 도는데.. :
  π: S_0, A_0, R_1, …, S_{T-1}, A_{T-1}, R_T 를 따르는 episode 를 생성한다.
  G는 0으로 시작.
  episode 의 각 step t = T-1, T-2, …, 0 에 따라 Loop:
    $G = G + R_{t+1}$

S_t, A_t 가 S_0, A_0, R_1, …, S_{t-1}, A_{t-1} 에 나타나지 않는 한: (마지막이$_t$ 아닐 때?)
G를 Returns(S_t, A_t) 에 append
Returns(S_t, A_t) 의 평균을 $Q(S_t, A_t)$ 로.
$\text{argmax}_a Q(S_t, a)$ 를 A^* 로.
모든 $a \in \mathcal{A}(S_t)$에 대해
$\pi(a|S_t)$ = 저렇다.

q_π 에 대한 어떤 ε-greedy policy 도 어떤 ε-soft policy π 보다 낫다. policy improvement theorem에 의해 보장됨.

준혁샘 대단하시네여!

π' 가 ε-greedy policy 라고 하자.
policy improvement theorem 의 conditions 이 apply 하는데,
왜냐면 for any $s \in \mathcal{S}$:

$$
\begin{aligned}
q_\pi(s, \pi'(s)) &= \sum_a \pi'(a|s) q_\pi(s,a) \\
&= \frac{\epsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s,a) + (1-\varepsilon) \max_a q_\pi(s,a) \qquad (5.2) \\
&\geq \frac{\epsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s,a) + (1-\varepsilon) \sum_a \frac{\pi(a|s) - \frac{\epsilon}{|\mathcal{A}(s)|}}{1-\varepsilon} q_\pi(s,a)
\end{aligned}
$$

(the sum is a weighted average with nonnegative weights summing to 1, and as such it must be less than or equal to the largest number averaged)

$$
\begin{aligned}
&= \frac{\epsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s,a) - \frac{\epsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s,a) + \sum_a \pi(a|s) q_\pi(s,a) \\
&= v_\pi(s).
\end{aligned}
$$

(sum 은 합이 1인 nonnegative weights의 weighted average 이다. average 된 가장 큰 수보다 작거나 같아야 함.)

그래서, policy improvement theorem 에 따라, $\pi' \geq \pi$ 이다.
그러니까, 모든 s에 대해 $v_{\pi'}(s) \geq v_\pi(s)$ 라는 뜻.
이제 equality 가
ε-soft policies 중 π', π 모두 optimal 일 때,
그러니까, 다른 모든 ε-soft policies 보다 더 좋거나 같을 때,
만 성립한다는 걸 증명한다.

new environment 를 생각해보자. original environment랑 같은데, 뭐만 다른?
policies 가 ε-soft "moved inside" the environment 해야하는 조건이 있어.
new environment는 original과 같은 action, state set 을 갖고 있다. 다음과 같이 behave하는.
state s 에서 action a 를 하면, 1-ε 확률로 old environment와 똑같이 behave 한다.
ε 확률로 action을 같은 확률의 랜덤으로 다시 뽑는다.
그리고 새로운 랜덤 action으로 old environment 처럼 behave 한다.
general policies 를 가진 new environment 에서 할 수 있는 best 는 ε-soft policies 를 가진 original environment에서 할 수 있는 best 와 같다.
v_*, q_* 위에 물결 표시한 게 new env. 에서 optimal value functions 라고 하자.

policy π는 ε-soft policies 중 optimal 이다. if and only if v_π = v_*~ .
v_*~ 의 정의로부터, 이게

$$\begin{aligned}\tilde{v}_*(s) &= (1-\varepsilon)\max_a \tilde{q}_*(s,a) + \frac{\epsilon}{|\mathcal{A}(s)|}\sum_a \tilde{q}_*(s,a) \\ &= (1-\varepsilon)\max_a \sum_{s',r} p(s',r|s,a)\Big[r+\gamma\tilde{v}_*(s')\Big] \\ &\quad + \frac{\epsilon}{|\mathcal{A}(s)|}\sum_a\sum_{s',r} p(s',r|s,a)\Big[r+\gamma\tilde{v}_*(s')\Big].\end{aligned}$$

에 대한 unique solution 임을 안다. 이건 뭔 말일까..

equality 가 성립하고 ε-soft policy π 가 더이상 improve 되지 않으면,
(5.2)로부터

$$\begin{aligned}v_\pi(s) &= (1-\varepsilon)\max_a q_\pi(s,a) + \frac{\epsilon}{|\mathcal{A}(s)|}\sum_a q_\pi(s,a) \\ &= (1-\varepsilon)\max_a \sum_{s',r} p(s',r|s,a)\Big[r+\gamma v_\pi(s')\Big] \\ &\quad + \frac{\epsilon}{|\mathcal{A}(s)|}\sum_a\sum_{s',r} p(s',r|s,a)\Big[r+\gamma v_\pi(s')\Big].\end{aligned}$$

하지만, 이 equation 은 이전 것과 같다. v_π 를 v_*~ 대신 쓴 거 빼곤.
왜냐면 v_*~ 는 unique solution 이고 v_π = v_*~ 이어야 하기 때문.

본질적으로, 이전 몇 페이지에서 policy iteration이 ε-soft policies 에 대해 작동함을 봤다.
ε-soft policies 를 위한 greedy policy 의 자연스러운 개념을 사용하면,
모든 step 에서 improvement가 보장된다.
ε-soft policies 중 best policy 를 찾았을 때 빼곤.
이 분석은 각 state 에서 action-value function이 어떻게 결정되는지에 independent 하다.
하지만 그들이 정확히 계산됐다는 가정 하에.
이게 이전 section 에서처럼 같은 포인트로 우릴 데려다 놓는다.
이제 ε-soft policies 중에 best policy만 achieve 한다.
하지만 반대로, exploring starts 가정을 없앴다.

5.5 Off-policy Prediction via Importance Sampling

모든 learning control methods 는 dilemma 에 빠져 있다.
그들은 subsequent optimal behavior 를 조건으로 한 action values 를 학습하려 한다.
하지만 non-optimally 하게 behave 해야 한다. 모든 actions 을 explore 하기 위해. (optimal actions을 찾기 위해)
exploratory policy를 따라 behave 하면서 어떻게 optimal policy 에 대해 학습할 수 있을까?
이전 section 의 on-policy approach 는 사실 compromise 이다.
그건 optimal policy 에 대한 action values 가 아니라 여전히 explore 하는 near-optimal policy 에 대한 action values 를 학습한다.

더 straightforward approach는 두 개의 policies 를 이용하는 거다.
하나는 학습되고 optimal policy 가 되는 애.
하나는 more exploratory 하고 behavior 를 generate 하는 데 쓰이는 애.
학습되는 policy 는 target policy 라고 부르고,
generate behavior 하는 데 쓰이는 policy는 behavior policy 라고 부른다.
이 경우에 target policy를 벗어난 data로부터 학습이 이루어진다고 얘기하고,
전반적인 process 를 off-policy learning 이라 부른다.

이 책의 나머지 내내 on-policy, off-policy methods 를 고려한다.
On-policy methods 는 일반적으로 더 간단하고 먼저 고려된다.
Off-policy methods 는 추가적인 개념과 표기가 필요하다. 그리고 data가 다른 policy 에서 나오기 때문에 off-policy methods 가 주로 더 큰 variance 를 갖고 converge도 더 느리다.
반면, off-policy methods 가 더 powerful 하고 더 일반적이다.
걔들은 on-policy methods 를 target 과 behavior policies 가 같은 special case 로 포함하고 있기 때문.
Off-policy methods 가 적용도 더 다양하게 된다.
예를 들어, 걔들은 전통적인 non-learning controller 나 human expert 에 의해 생성된 data로부터 학습하는 데 적용될 수 있다.
Off-policy learning 은 world's dynamics 의 multi-step predictive models 을 학습하는 키로도 여겨진다.

이 section 에서 prediction problem 을 고려하며 off-policy methods 를 공부해 보자.
이 prediction problem 은 target & behavior policies 가 둘 다 fixed 돼 있다.
즉, 우리가 v_π, q_π를 estimate 하고 싶다 해보자. 하지만 우리가 가진 건 다른 policy $b \neq \pi$ 를 따르는 episodes 뿐이야.
이 경우에, π 는 target policy이고 b는 behavior policy 야. 두 policies 모두 fixed, given 이고.

b 에서의 episodes 를 사용해 π 일 때의 values 를 estimate 하려면, π 하에서 취해진 모든 action이 또한 취해져야 한다. 최소한 occasionally 라도. b 아래서.
그러니까, $\pi(a|s) > 0$ 이 $b(a|s) > 0$ 를 imply하는 게 필요하다.
이게 assumption of coverage 라 불림.
coverage 에서 b 가 π와 같지 않은 states 에서는 b 는 stochastic 이어야 한다.(?)
반면, target policy π 는 deterministic 일 수 있고, 사실, 이게 control applications 에서 특히 관심있는 case 이다.
control 에서, target policy 는 보통 action-value function 의 current estimate 에 대한 deterministic greedy policy 이다.
이 policy 가 deterministic optimal policy 가 된다. behavior policy 가 stochastic 하고 더 exploratory 하는 동안. 예를 들어, ε-greedy policy.
하지만 이 section에서 prediction problem 을 볼 거다. π가 unchanging 이고 given인.

대개 모든 off-policy methods 는 importance sampling 을 활용한다.
importance sampling: 다른 분포에서 표본이 주어진 분포 하에서 expected values 를 estimating 하는general technique.
importance-sampling ratio: target & behavior policies 하에서 발생하는 궤적의 상대적 확률에 따라 returns 을 가중시켜 off-policy 학습에 importance sampling 을 적용한다.

starting state S_t 이 주어졌을 때, any policy π 아래서 그 후의 state-action 궤적, A_t, S_{t+1}, A_{t+1}, S_T 의 확률은

$$\begin{aligned}
&\Pr\{A_t, S_{t+1}, A_{t+1}, \dots, S_T \mid S_t, A_{t:T-1} \sim \pi\} \\
&\quad = \pi(A_t|S_t)p(S_{t+1}|S_t, A_t)\pi(A_{t+1}|S_{t+1}) \cdots p(S_T|S_{T-1}, A_{T-1}) \\
&\quad = \prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k),
\end{aligned}$$

p 는 (3.4)에 의해 정의된 state-transition probability function.
그래서, target & behavior policies 하에서 상대적 확률의 궤적, 그러니까 importance-sampling ratio 는

$$\rho_{t:T-1} \doteq \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k)}{\prod_{k=t}^{T-1} b(A_k|S_k)p(S_{k+1}|S_k, A_k)} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}. \tag{5.3}$$

이거시다.

trajectory probabilities 가 일반적으로 unknown인 MDP의 transition probabilities 에 depend 돼 있다 하더라도, numerator 와 denominator 에 identically 나타난다. 그래서 cancel 됨.
importance sampling ratio 는 MDP 가 아니라 두 policies 와 sequence에 의해서만 결정된다.

기억나니. target policy 하에서 expected returns (values) 를 estimate 하고자 했었지. 하지만 우리가 가진 건 behavior poilcy 로 얻은 G_t 뿐이다.
이 returns 들은 wrong expectation $\mathbb{E}[G_t|S_t] = v_b(S_t)$ 을 갖고 있다. 그래서 v_π 를 얻기 위해 평균낼 수 없다.
여기서 importance sampling 이 들어온다.
ratio $\varrho_{t:T-1}$ 이 returns 을 right expected value 로 바꿔준다.

$$\mathbb{E}[\rho_{t:T-1}G_t \mid S_t] = v_\pi(S_t). \tag{5.4}$$

요로케.

이제 우리는 준비가 됐다. policy b 를 따르는 observed episodes 의 batch에서 나온 returns 을 average 하는 Monte Carlo algorithm 으로 $v_\pi(s)$ 를 estimate 할 준비.
여기서는 에피소드의 경계를 넘어 time steps 을 늘리는 게 편리하다.
즉, time 100 인 terminal state 에서 batch의 첫 episode 가 끝나면,
다음 episode 가 t = 101 에서 시작한다.
이건 우리가 특정 episodes 에서 특정 steps 을 참조하는 time-step numbers 를 사용할 수 있게 해준다.
특히, state s 가 visit 되는 모든 time steps 의 집합을 정의할 수 있다.
그 집합을 $\mathcal{T}(s)$ 라 쓴다.
이건 every-visit method 용이다.
first-visit method 에선, $\mathcal{T}(s)$ 가 오직 해당 episodes 내에서 s 에 first visits 한 time steps 만 포함할 것이다.
또한, $T(t)$ 는 time t 이후의 termination의 첫 번째 time 을 나타낸다.
G_t 는 t 이후 $T(t)$ 까지의 return 을 나타낸다.
그러면 $\{G_t\}_{t\in\mathcal{T}(s)}$ 가 state s 와 관련있는 returns 이다.

$\{\varrho_{t:T-1}\}_{t\in\mathcal{T}(s)}$ 가 대응하는 importance-sampling ratios 이고.
$v_\pi(s)$ 를 estimate 하기 위해서, 우린 단순히 returns 을 ratios 로 scale 하고 결과를 average 한다.

$$V(s) \doteq \frac{\sum_{t\in\mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{|\mathcal{T}(s)|}. \tag{5.5}$$

importance sampling 이 이렇게 simple average 로 되면, ordinary importance sampling 이라고 부름.
중요한 alternative 는 weighted importance sampling 인데, weighted average 를 쓴다.

$$V(s) \doteq \frac{\sum_{t\in\mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{\sum_{t\in\mathcal{T}(s)} \rho_{t:T(t)-1}}, \tag{5.6}$$

이렇게 쓰고 말이지.
denominator 가 0이면 0.
importance sampling 의 이 두 varieties 를 이해하기 위해, single return을 관찰한 후 그들의 estimates를 생각해 보자.
weighted-average estimate 에선, single return 에 대한 ratio $\varrho_{t:T-1}$ 가 cancel 된다.
그래서 estimate 가, 관측된 return(ratio 와 무관) 과 같다. (ratio 가 0이 아니라 가정)
이 return 이 관측된 단 하나면, 이건 reasonable estimate 이다.
하지만 얘의 expectation 이 $v_\pi(s)$ 보단 $v_b(s)$ 이고,
이 통계적 관점에서 biased 돼 있다.
이와 대조적으로, (5.5)의 simple average 는 항상 $v_\pi(s)$ 를 expectation 으로 한다? (unbiased)
하지만 extreme 할 수 있지.
ratio 가 10이었다 치자. 관찰된 궤적이 behavior policy 때보다 target policy 에서 10 배 더 likely 하다. 가능성이 있다.
이 case 에서 ordinary importance-sampling estimate 는 observed return 의 10배일 거다.
그러니까, episode 의 trajectory가 target policy를 아주 잘 나타낸다고 해도, observed return 에서 꽤 멀 것이다.

Formally, 두 가지 importance sampling 의 차이는 biases 와 variances 에서 나타난다.
ordinary importance-sampling estimator 는 unbiased 인 반면
weighted importance-sampling estimator 는 biased 이다. (bias 는 0에 asymptotically 수렴)

반면,ordinary importance-sampling estimator 의 variance는 일반적으로 unbounded. 제한이 없다. 왜냐면 ratios 의 variance 는 unbounded니까. 왜? 어째서여.
weighted estimator 는 어떤 single return 의 largest weight 도 1이다.

사실, bounded returns 을 가정하면, weighted importance-sampling estimator 의 variance는 0으로 수렴한다. ratios 의 variance 가 그 자체로 infinite 이라 해도.

실제로는, weighted estimator 는 dramatically lower variance 를 갖고 매우 선호된다.
그럼에도 불구하고, ordinary importance sampling 을 완전히 배제하진 않을 거다. 왜냐면 function approximation 을 사용한 approximate methods 로 확장하기 더 쉬워서. 이건 책의 second part 에서 볼 거야.

weighted importance sampling 을 이용한 off-policy evaluation 에 대한 every-visit MC algorithm 의 complete 내용이 88페이지 박스에 나와. 근데 왜 지금 말해주는거지.
이따가 보자.

(Example 5.4) Off-policy Estimation of a Blackjack State Value
off-policy data 에서 나온 single blackjack state 의 value 를 estimate 하는 데 ordinary와 weighted importance-sampling methods 둘 다 적용해봤다.
돌이켜보렴. Monte Carlo methods 의 장점 중 하나가 뭐였는지.
다른 states 에 대한 estimates 를 형성하지 않고도 single state 를 평가하는 데 걔들을 쓸 수 있었지.
이 예에서, dealer 가 deuce를 보여주는 state 를 evaluate 했다.
player 의 cards 의 합이 13이고 player 가 usable ace 를 갖고 있는 거지.
그러니까, player 가 ace 와 deuce를 갖고 있거나, equivalently 3 aces 를 갖고 있다.
data 가 이 state 에서 시작해 생성되고, 같은 확률로 hit or stick 고른다.(behavior policy)
target policy 는 sum 이 20 이나 21일 때만 stick.
target policy 하에서 이 state 의 value 는 대략 -0.27726 이다. (이건 결정됐다. 개별적으로 1억 개의 episodes 를 생성해서. episodes 는 target policy 와 returns 을 averaging 하는 걸 이용.)
both off-policy methods 가 random policy 를 이용한 1000 개의 off-policy episodes 이후 이 value 를 closely approximated 했다.
얘들이 이걸 reliably 했다는 걸 확실하게 하기 위해, 100 independent runs 을 수행했고, 각각 0부터 10000 episodes 까지 학습했다.
(Fig 5.3) 은 resultant learning curves 다. number of episodes 의 함수로 각 method 의 estimates 의 squared error 를 100 runs 동안 average 함.
두 algorithms 모두 error가 0으로 다가간다. 하지만 weighted importance-sampling method 가 초반에 훨씬 낮은 error 를 보여준다. 실제에서도 이럼.

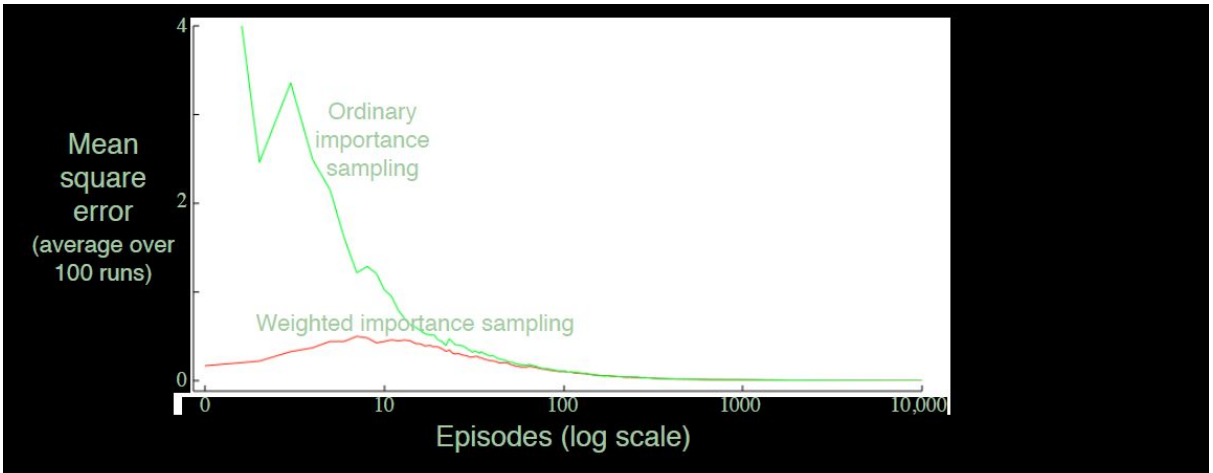


Figure 5.3: Weighted importance sampling produces lower error estimates of the value of a single blackjack state from off-policy episodes (see Example 5.4).

(Example 5.5) Infinite Variance
ordinary importance sampling 의 estimates 는 보통 infinite variance를 가질 거다.
그래서 unsatisfactory convergence properties. scaled returns 가 infinite variance 를 가질 때마다.

이건 off-policy learning 에서 쉽게 발생한다. 궤적이 loops 를 포함할 때.
(Fig 5.4) 의 inset에서 간단한 예를 볼 수 있다.
한 개의 nonterminal state s 와 두 actions, right & left 가 있다.
right action 하면 deterministic 하게 termination 으로 옮겨가고,
left action 하면, 0.9 확률로 s로 돌아오고 0.1 확률로 termination 으로 간다.
rewards 는, 후자일 때 +1, 나머진 0. 후자가 뭔데.
항상 left 만 선택하는 target policy를 생각해 보자.
이 policy 하에서 모든 episodes 는 어떤 숫자(아마 0)로 구성돼 있다. 어떤 숫자?
s로 옮겨간 후 reward 와 +1 return 으로 termination 하는 transitions 의 수.
그래서 target policy 하에서 s 의 value 는 1이다. (γ = 1)
이 value 를 right & left 를 같은 확률로 선택하는 behavior policy 를 사용하는 off-policy 로부터 estimate 한다고 가정하자.

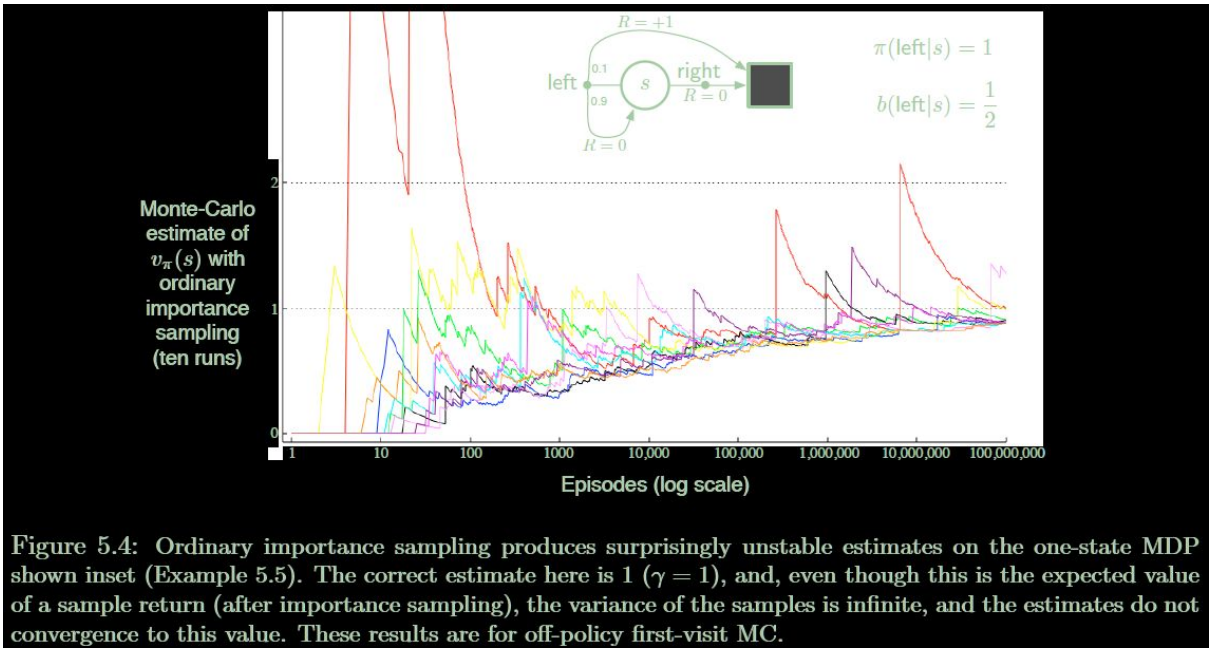


Figure 5.4: Ordinary importance sampling produces surprisingly unstable estimates on the one-state MDP shown inset (Example 5.5). The correct estimate here is 1 ($\gamma = 1$), and, even though this is the expected value of a sample return (after importance sampling), the variance of the samples is infinite, and the estimates do not convergence to this value. These results are for off-policy first-visit MC.

(Fig 5.4) 의 아래 부분은 ordinary importance sampling 을 사용한 first-visit MC algorithm 의 independent runs 10개를 보여준다. 수백만 episodes를 지난 후에도 estimates가 correct value 1에 수렴하는 게 실패한다.
이와 대조적으로, weighted importance-sampling algorithm 은 left action 으로 끝나는 첫 번째 episode 이후 정확히 1 의 estimate 를 줄 거다.
1이 아닌 모든 returns(right action으로 끝나는) 은 target policy 와 일치하지 않을 거다.
그래서 $\rho_{t:T(t)-1}$ 는 0이 될 거고 (5.6) 식의 분모, 분자에 영향을 미치지 않을 거다.
weighted importance-sampling algorithm 은 target policy 와 일치하는 returns 만의 weighted average를 만들어 내고, 이 모두는 정확히 1일 거야.
importance-sampling-scaled returns 의 variance가 infinite 라는 걸 확인할 수 있다.

$$\text{Var}[X] \doteq \mathbb{E}\left[(X - \bar{X})^2\right] = \mathbb{E}\left[X^2 - 2X\bar{X} + \bar{X}^2\right] = \mathbb{E}\left[X^2\right] - \bar{X}^2.$$

이거 알지?
그래서 우리 case에서처럼 평균이 finite 면,
variance 가 infinite ⇔ random variable 의 square의 expectation 이 infinite
그래서, importance-sampling-scaled return 의 expected square 가 infinite 인 것만 보이면 된다.

$$\mathbb{E}_b\left[\left(\prod_{t=0}^{T-1}\frac{\pi(A_t|S_t)}{b(A_t|S_t)}G_0\right)^2\right]$$

요거.
이 expectation을 계산하기 위해, 이걸 쪼갤 거야. episode length 와 termination에 따른 cases로.
먼저, right action으로 끝나는 어떤 episode든, importance sampling ratio 는 0이야.
왜냐하면 target policy가 이 action 을 절대 안할 거거든.
그래서 이 episodes 들이 expectation에 영향을 안 미쳐서(괄호 안이 0이 될 거라?) 무시될 수 있지.
우린 left actions 의 수(아마 0) 를 포함한 episodes만 고려하면 된다. 여기서 left actions 은 nonterminal state 로 돌아가는 transition 이후 left action으로 termination으로 가는 transition.
이 모든 episodes 들은 1의 return 을 갖고, 그래서 G_0 factor는 무시될 수 있다.
expected square 를 얻기 위해 episode의 각 length 만 고려하며, episode 의 발생 확률을 importance-sampling ratio 의 제곱에 곱하고, 합치면 된다.

$$\begin{aligned}
&= \frac{1}{2}\cdot 0.1\left(\frac{1}{0.5}\right)^2 && \text{(the length 1 episode)}\\
&\quad + \frac{1}{2}\cdot 0.9\cdot\frac{1}{2}\cdot 0.1\left(\frac{1}{0.5}\frac{1}{0.5}\right)^2 && \text{(the length 2 episode)}\\
&\quad + \frac{1}{2}\cdot 0.9\cdot\frac{1}{2}\cdot 0.9\cdot\frac{1}{2}\cdot 0.1\left(\frac{1}{0.5}\frac{1}{0.5}\frac{1}{0.5}\right)^2 && \text{(the length 3 episode)}\\
&\quad + \cdots\\
&= 0.1\sum_{k=0}^{\infty} 0.9^k\cdot 2^k\cdot 2 = \ 0.2\sum_{k=0}^{\infty} 1.8^k = \ \infty. && \blacksquare
\end{aligned}$$

(Exercise 5.4) state values $V(s)$ 대신 action values $Q(s, a)$에 대한 (5.6)과 유사한 식이 뭐니? b를 이용해 생성된 다시 주어진 returns 인가? 이게 뭔 말인공.. again given returns generated using b?

(Exercise 5.5) (Fig 5.3)과 같은 learning curves 에서, error는 일반적으로 training 에 따라 감소한다. ordinary importance-sampling method 에서도 그랬던 것처럼.
하지만 weighted importance-sampling method 에선 처음엔 error가 증가했다가 다시 감소한다. 왜 이럴까?

(Exercise 5.6) (Example 5.5 의 결과와 (Fig 5.4)에서 first-visit MC method 를 사용했다.
같은 문제에 every-visit MC method 가 사용됐다고 가정해 보자. estimator 의 variance 가 여전히 infinite 일까? 이유는?

5.6 Incremental Implementation

Monte Carlo prediction methods 는 incrementally 적용될 수 있다. episode-by-episode basis 로. 챕터 2에서 나왔던 기술들의 확장된 버전을 써서.

챕터 2에서는 rewards 를 평균낸 거라면, Monte Carlo methods 에선 returns 을 평균낸다.
다른 모든 측면에서는 챕터2에서 쓰인 methods 와 같은 걸 on-policy Monte Carlo methods 에 쓸 수 있다.
off-policy Monte Carlo methods 에선, ordinary importance sampling 을 쓰는 애들과 weighted importance sampling 을 쓰는 애들을 개별적으로 고려해야 한다.

ordinary importance sampling 에서는, returns 이 importance sampling ratio $\varrho_{t:T(t)-1}$ (5.3) 으로 scale 됐다. 그리고 단순 평균냈지.
이런 methods 에 대해 우린 챕터2의 incremental methods 를 다시 사용할 수 있다.
하지만 rewards 대신 scaled returns 을 사용해서.
이게 weighted importance sampling을 이용하는 off-policy 의 case 를 남긴다.
여기서 우린 returns 의 weighted average 를 형성해야 하고, 약간 다른 incremental algorithm 이 필요하다.
returns $G_1, G_2, \ldots, G_{n-1}$ 의 sequence가 있다고 가정해 보자. 전부 같은 state 에서 시작하며, 각각 random weight W_i (e.g., $W_i = \varrho_{t:T(t)-1}$) 에 대응한다.
우린 estimate

$$V_n \doteq \frac{\sum_{k=1}^{n-1} W_k G_k}{\sum_{k=1}^{n-1} W_k}, \qquad n \geq 2, \tag{5.7}$$

를 form 하고 싶고, single additional return G_n 을 얻을 때 최신 상태로 유지하길 바란다.
V_n 을 추적하는 거 외에도, 각 state 에 대해 cumulative sum C_n을 유지해야 한다.
뭐에 대한 cumulative sum? 첫 번째 n 개 returns 에 주어진 weights 의 cumsum.
V_n 에 대한 update rule 은

$$V_{n+1} \doteq V_n + \frac{W_n}{C_n}\Big[G_n - V_n\Big], \qquad n \geq 1, \tag{5.8}$$

이거시고, C_n 은 요러하다.

$$C_{n+1} \doteq C_n + W_{n+1}$$

$C_0 = 0$ (V_1 은 임의로.)

아래 박스는 Monte Carlo policy evaluation 을 하는 episode 별 incremental algorithm 이다.
알고리즘은 nominally, weighted importance sampling 을 사용하는 off-policy case 용이지만, on-policy 에도 적용된다. target & behavior policies 를 같게 고르면.($\pi = b$ 이고, W 가 항상 1 인 case)
approximation Q 는 q_π 로 수렴한다. (모든 encountered state-action pairs 에 대해)
potentially 다른 policy b 에 따라 actions 이 선택되는 동안.

Off-policy MC prediction (policy evaluation) for estimating $Q \approx q_\pi$

Input: an arbitrary target policy π
Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
  $Q(s,a) \in \mathbb{R}$ (arbitrarily)
  $C(s,a) \leftarrow 0$

Loop forever (for each episode):
  $b \leftarrow$ any policy with coverage of π
  Generate an episode following b: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  $W \leftarrow 1$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
    $W \leftarrow W \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$
    If $W = 0$ then exit For loop

박스 내용:
Input: 임의의 target policy π
Initialize, 모든 $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$ 에 대해:
  $Q(s, a) \in R$ (arbitrary)
  $C(s, a) = 0$
Loop forever (for each episode):
  b = any policy with coverage of π
  b를 따르는 episode 생성: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G = 0$
  $W = 1$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G = \gamma G + R_{t+1}$
    $C(S_t, A_t) = C(S_t, A_t) + W$
    $Q(S_t, A_t) = Q(S_t, A_t) + W / C(S_t, A_t) * (G - Q(S_t, A_t))$
    $W = W * \pi(A_t | S_t) / b(A_t | S_t)$
    $W = 0$ 이면 loop 빠져나간다.

(Exercise 5.7) section 5.1의 first-visit MC policy evaluatoin에 대한 알고리즘을 수정해봐라. section 2.4 의 incremental implementation for sample averages 를 이용해서.

(Exercise 5.8) 5.7로부터 5.8 weighted-average update rule 을 이끌어내보렴.
2.3 의 unweighted rule 의 미분의 패턴을 따라가 봐.

5.7 Off-policy Monte Carlo Control

이제 learning control methods 의 두 번째 class 의 예를 보여줄 준비가 됐어.
떠올려보렴. on-policy methods 의 구별되는 특징은, 그걸 control 에 사용하는 동안 policy의 value를 estimate 한다는 거지.
off-policy methods 에서는 이 두 함수가 분리돼 있다.
behavior 를 생성하는 데 쓰이는 policy, 그러니까 behavior policy 는 사실 evaluated 되고 improved 되는 policy(target policy) 와 관련이 없을 수 있다.

이 분리의 장점은, target policy 가 deterministic (e.g., greedy) 할 수 있다는 거다. behavior policy 가 모든 가능한 actions 을 계속 샘플링 할 수 있는 동안.

off-policy Monte Carlo control methods 는 이전 두 섹션에서 나온 테크닉들 중 하나를 쓴다.
그들은 학습하고 target policy를 개선해 나가면서 behavior policy 를 따른다.
이런 테크닉들은 behavior policy 가 target policy (coverage) 에 의해 선택될 수 있는 모든 actions 을 선택할 확률이 0이 아니길 요구한다.
모든 가능성을 explore 하기 위해, behavior policy 가 soft 해야 해. (그러니까 0이 아닌 확률로 모든 states 에서 모든 actions 을 선택한다는 말)

다음 박스는 GPI 와 weighted importance sampling 에 근거해 π_*, q_* 를 estimating 하는 off-policy Monte Carlo method 를 보여준다.
target policy $\pi \approx \pi_*$ 는 Q에 대한 greedy policy 이다. Q는 q_π 의 estimate 이지.
behavior policy b 는 뭐든 될 수 있는데, π가 optimal policy 에 수렴하는 걸 보장하기 위해선 returns 무한 개가 각 state-action pair 에 대해 구해져야 한다.
이건 b 를 ε-soft 하게 고르면 보장된다.
policy π 가 모든 encountered states 에서 optimal 로 수렴한다. actions 이 다른 soft policy b 에 따라 선택된다 하더라도. b는 episodes 간, 또는 episodes 내에서도 변할 수 있고.

Off-policy MC control, for estimating $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
  $Q(s,a) \in \mathbb{R}$ (arbitrarily)
  $C(s,a) \leftarrow 0$
  $\pi(s) \leftarrow \operatorname{argmax}_a Q(s,a)$ (with ties broken consistently)

Loop forever (for each episode):
  $b \leftarrow$ any soft policy
  Generate an episode using b: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  $W \leftarrow 1$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
    $\pi(S_t) \leftarrow \operatorname{argmax}_a Q(S_t, a)$ (with ties broken consistently)
    If $A_t \neq \pi(S_t)$ then exit For loop
    $W \leftarrow W \frac{1}{b(A_t|S_t)}$

박스 내용을 좀 볼까?
Initialize, 모든 $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$에 대해:
  $Q(s, a) \in R$ (arbitrarily)
  $C(s, a) = 0$
  $\pi(s) = \operatorname{argmax}_a Q(s, a)$ (with ties broken consistently)
  임의의 $Q(s, a)$ 들이 배치됐고, 그 중 max 들의 actions 을 $\pi(s)$ 에 배정.
Loop forever (for each episode): 각 에피소드마다 loop
  b = any soft policy
  b를 사용하는? 이용한? episode 생성: S_0, A_0, R_1, …, S_{T-1}, A_{T-1}, R_T
  $G = 0$
  $W = 1$
  Loop for each step of episode, $t = T$-1, T-2, …, 0: 에피소드의 각 step 마다 loop

$G = \gamma G + R_{t+1}$
$C(S_t, A_t) = C(S_t, A_t) + W$
$Q(S_t, A_t) = Q(S_t, A_t) + W / C(S_t, A_t) * (G - Q(S_t, A_t))$
$\pi(S_t) = \text{argmax}_a\ Q(S_t, a)$ (tie 면 consistently 쪼개)
$A_t \neq \pi(S_t)$ 이면 for loop 빠져나감.
$W = W / b(A_t \mid S_t)$

잠재적 문제는, 이 method 가 episode 의 모든 remaining actions 가 greedy 일 때인 episodes 의 끝에서만 학습한다는 거다.
nongreedy actions 이 흔하면, 학습이 느려질 거고, 긴 episodes 의 앞 부분에서 나타나는 states 에 대해서는 특히 더 느려지겠지.
잠재적으로, 이건 학습을 엄청 느리게 할 수 있다.
이 문제가 얼마나 심각한지 평가하는 off-policy Monte Carlo methods 의 경험이 불충분했다.
이게 심각하면, temporal-difference learning 을 결합시키는 게 아마 가장 중요한 해결책일 거다. 다음 챕터의 algorithmic idea 야. 기대되지?
대안으로, γ가 1보다 작으면, 다음 section 에서 나올 idea 도 아주 도움이 될 수 있다.

(Exercise 5.9) 박스 안의 off-policy MC control 알고리즘에서, W를 update 할 때 importance-sampling ratio 인 $\pi(A_t \mid S_t) / b(A_t \mid S_t)$ 를 쓰리라 예상했을 수 있어.
근데 $1 / b(A_t \mid S_t)$ 였지? 왜 그래도 이게 correct 일까?

(Exercise 5.10) Racetrack (programming)
이건 나중에 하자.

5.8 Discounting-aware Importance Sampling
지금까지 본 off-policy methods 는 unitary wholes 로 여겨지는 returns 에 대한 importance-sampling weights 를 형성하는 것에 기초한다. discounted rewards 의 합으로 returns 의 내부 구조를 고려하지 않고.
이제 off-policy estimators 의 variance 를 크게 줄이기 위해 이 구조를 사용하는 cutting-edge research ideas 를 간략히 살펴보자.
예를 들어, episodes 가 길고 γ 가 1보다 매우 작은 case를 생각해봐.
정확히 말하자면, episodes 가 100 steps. $\gamma = 0$.
그러면 time 0 에서 return 은 $G_0 = R_1$, 하지만 importance sampling ratio 는 100 factors 의 product

$$\frac{\pi(A_0|S_0)}{b(A_0|S_0)}\frac{\pi(A_1|S_1)}{b(A_1|S_1)}\cdots\frac{\pi(A_{99}|S_{99})}{b(A_{99}|S_{99})}$$

이다.
ordinary importance sampling에선, return 은 entire product 로 scale 될 거다.
하지만 첫 번째 factor $\pi(A_0 \mid S_0) / b(A_0 \mid S_0)$ 로 scale 되는 것만 필요하다.
나머지 99 factors 는 irrelevant 하다. 왜냐면 첫 번째 reward 이후, return 은 결정됐기 때문이다.
이 뒤쪽 factors 는 모두 return, expected value 1 과 무관하다.
그들은 expected update 를 바꾸지 않는다.
하지만 variance 에 엄청 add 한다.
일부 cases 에선 variance 를 infinite 로 만들 수도 있다.

이제 이 큰 extraneous variance를 피하는 idea 를 생각해 보자.

idea 의 정수는 discounting 을 termination 확률을 결정하는 걸로 생각하는 거다.
또는, equivalently, a degree of partial termination 으로 생각한다.
for any $\gamma \in [0,1)$, 우린 return G_0 를 one step 에서 partly terminating 으로 생각할 수 있다.
degree 1 - γ 로, 첫 번째 reward R_1 만의 return 을 produce 하면서.
2 steps 이후 partly terminating 하며 degree$(1 - \gamma)\gamma$ 로, $R_1 + R_2$ 의 return 을 produce.
요런 방식으로다가 계속 가는겨.
latter degree는 second step 에서 terminating 하는 1 - γ ? 그리고 first step에서 이미 terminated 되지 않은 γ에 대응한다.
그래서 third step 에서 termination 의 degree 는 $(1 - \gamma)\gamma^2$. γ^2 은 첫 두 steps 중 어느 하나에서도 일어나지 않은 termination을 reflecting.
여기서 partial returns 를 flat partial returns 라 부른다:

$$\bar{G}_{t:h} \doteq R_{t+1} + R_{t+2} + \cdots + R_h, \qquad 0 \le t < h \le T,$$

flat 은 discounting 이 없음을 나타내고,
partial 은 이들 returns 이 termination 까지 확장되는 게 아니라 h 에서 멈추는 걸 나타내고 horizon 이라 부름. (T는 episode 의 termination 의 time)

conventional full return Gt 는 flat partial returns 의 합으로 볼 수 있다. 다음과 같이.

$$\begin{aligned}
G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T \\
&= (1-\gamma) R_{t+1} \\
&\quad + (1-\gamma)\gamma \left(R_{t+1} + R_{t+2}\right) \\
&\quad + (1-\gamma)\gamma^2 \left(R_{t+1} + R_{t+2} + R_{t+3}\right) \\
&\quad \vdots \\
&\quad + (1-\gamma)\gamma^{T-t-2} \left(R_{t+1} + R_{t+2} + \cdots + R_{T-1}\right) \\
&\quad + \gamma^{T-t-1} \left(R_{t+1} + R_{t+2} + \cdots + R_T\right) \\
&= (1-\gamma) \sum_{h=t+1}^{T-1} \gamma^{h-t-1} \bar{G}_{t:h} \quad + \quad \gamma^{T-t-1} \bar{G}_{t:T}.
\end{aligned}$$

이제 flat partial returns 를 비슷하게 truncated 된 importance sampling ratio 로 scale 해야한다.
$G_{t:h}$ (G 위에 bar) 가 horizon h 까지의 rewards 만 포함하듯이, 확률의 ratio 도 h 까지만 필요하다.
ordinary importance-sampling estimator 를 정의하자. (5.5)와 유사하게.

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \left((1-\gamma) \sum_{h=t+1}^{T(t)-1} \gamma^{h-t-1} \rho_{t:h-1} \bar{G}_{t:h} \quad + \quad \gamma^{T(t)-t-1} \rho_{t:T(t)-1} \bar{G}_{t:T(t)} \right)}{|\mathcal{T}(s)|}, \qquad (5.9)$$

weighted importance-sampling estimator 는 아래와 같다. (5.6)과 유사하게.

$$V(s) \doteq \frac{\sum_{t\in\mathcal{T}(s)} \left((1-\gamma) \sum_{h=t+1}^{T(t)-1} \gamma^{h-t-1} \rho_{t:h-1} \bar{G}_{t:h} \quad + \quad \gamma^{T(t)-t-1} \rho_{t:T(t)-1} \bar{G}_{t:T(t)} \right)}{\sum_{t\in\mathcal{T}(s)} \left((1-\gamma) \sum_{h=t+1}^{T(t)-1} \gamma^{h-t-1} \rho_{t:h-1} \quad + \quad \gamma^{T(t)-t-1} \rho_{t:T(t)-1} \right)}. \tag{5.10}$$

이 두 estimators 를 discounting-aware importance sampling estimators 라 부른다.

γ = 1 이면 효과는 없다. (section 5.5 의 off-policy estimators 랑 같지.)

5.9 *Per-decision Importance Sampling

5.10 Summary
Monte Carlo methods 는 sample episodes 라는 형태로 experience로부터 value functions 과 optimal policies 를 학습한다.
이건 DP methods 보다 최소한 세 종류의 장점을 준다.
첫째, environment 와의 interaction으로부터 직접적으로 optimal behavior 를 학습하는 데 쓰일 수 있다. environment 의 dynamics 의 model 없이.
둘째, simulation 이나 sample models 와 함께 쓰일 수 있다.
매우 많은 applications 에서, sample episodes 를 simulate 하는 건 쉽다. DP methods 에 의해 요구되는 transition 확률의 explicit model 종류를 construct 하기 어렵다 하더라도.
셋째, Monte Carlo methods 를 states 의 작은 subset 에 집중시키기 쉽고 효율적이다.
special interest 한 region 은 정확하게 평가될 수 있다. state set 의 나머지 부분에 대해 정확히 평가하는 비용 없이.(자세한 건 8챕터에서)

네 번째 장점은 나중에 얘기할 건데, Markov property 의 violations 에 의한 타격이 작다.
이유는 그들의 value estimates 를 successor states 의 value estimates 에 근거해 update 하지 않기 때문이다.
그러니까, bootstrap 하지 않는다고.

Monte Carlo control methods 를 design 할 때, GPI 의 전반적인 schema 를 따랐다.
GPI 는 policy evaluation과 policy improvement 의 interacting processes 를 포함한다.
Monte Carlo methods 는 alternative policy evaluation process 를 제공한다.
각 state 의 value를 compute 하는 model을 사용하기보다,
state 에서 시작한 많은 returns 을 단순히 average한다.
state 의 value 가 expected return 이기 때문에, 이 평균은 value에 대해 좋은 approximation이다.
control methods 에서, 우린 특히 action-value functions 를 approximating 하는 데 관심이 있다.
왜냐하면 얘들은 environment의 transition dynamics 의 모델을 필요로 하지 않고 policy 를 improve 하는 데 쓰일 수 있기 때문이다.
Monte Carlo methods 는 episode-by-episode basis 로 policy evaluation 와 policy improvement 를 혼용한다. 그리고 episode-by-episode basis 로 incrementally implemented 될 수 있다.

sufficient exploration 을 유지하는 건 Monte Carlo control methods 의 이슈다.

그냥 현재 best 로 estimated 된 actions 을 선택하는 걸로는 부족하다.
그러면 alternative actions 에 대한 returns을 얻지 못할 거고, 걔들이 실제로 더 나은지 학습할 수 없을 것이기 때문이다.
한 가지 접근법은, episodes 가 모든 가능성을 cover 하기 위해 랜덤하게 선택된 state-action pairs 로 시작한다고 가정해 이 문제를 무시하는 거다.
그런 exploring starts 는 가끔 시뮬레이션 된 episodes 에서의 applications 에서 이루어질 수 있지만, real experience 에서의 학습에서는 그럴 수 없다.
on-policy methods 에서, agent는 항상 exploring 하고, explore 하는 best policy를 찾으려 한다.
off-policy methods 에서는, agent가 explore 하지만, 따르는 policy 와 관련없을 수 있는 deterministic optimal policy 를 학습한다.

off-policy prediction 은, 다른 behavior policy로 생성된 데이터로부터, target policy의 value function 을 학습하는 걸 말한다.
그런 learning methods 는 importance sampling 의 형태에 기초한다. 그러니까, 두 policies 하에서 관측된 actions 을 취할 확률의 ratio 로 얻은 weighting returns 에 기초해서, behavior policy 에서 target policy 로 그들의 expectations 를 변환한다.

ordinary importance sampling 은 weighted returns 의 simple average 를 쓰는 반면, weighted importance sampling 은 weighted average 를 쓴다.

ordinary importance sampling 은 unbiased estimates를 생성하지만, 더 큰, 아마 infinite 인 variance를 만든다.
weighted importance sampling 은 항상 finite variance를 갖지만 실제로는 더 선호된다.

개념적으로 간단함에도, off-policy Monte Carlo methods 로 하는 prediction과 control은 unsettled 로 남아있고 ongoing research 의 주제다.

이 장에서 다뤘던 Monte Carlo methods 는 이전 챕터에서 다룬 DP 와 두 가지 길에서 다른 점이 있다.
하나는, sample experience 에서 operate 된다는 점. 그래서 model 없이 direct learning 에 쓰일 수 있다.
둘째는, bootstrap 하지 않는다는 점. 즉, 다른 value estimates 에 근거해 value estimates를 update 하지 않는다.
이 두 차이점이 긴밀하게 연결돼 있진 않다.그리고 분리될 수 있지.
다음 챕터에서 experience 로부터 학습하는 methods 를 공부한다. Monte Carlo methods 처럼. 하지만 bootstrap 도 하지. DP methods 처럼.

$\gamma r \varepsilon \varrho$
$\pi(a|s)$
$v_\pi(s)$
$q_\pi(s, a)$
$ACSQRTVWGtabpni\boldsymbol{h}$
$\mathcal{APT}$

$\ni\in\geq\leq\neq\Leftrightarrow\ s\in\mathcal{S}a\in\mathcal{A}(s)$

$\varrho_{t:T(t)\text{-}1}$