Ch2. Multi-armed Bandits. 슬롯머신

강화학습이 다른 학습과 구별되는 가장 중요한 특징은
action을 evaluate한 정보를 쓴다는 것임.
correct action을 가르쳐주는 게 아님.
이것이 active exploration이 필요한 이유임. 좋은 행동을 explicit하게 찾는다는 게야.

evaluative feedback은 action이 얼마나 좋은지 가리키지만 가능한 action 중 best인지 worst 인지는 모르지.

instructive feedback은 correct action to take 를 주지만 실제 취한 action과는 독립적이야.
이건 기존의 supervised learning의 basis지.

evaluative feedback: 취한 action에 depend.
instructive feedback: 취한 action에 independent.

이 챕터에서 강화학습의 evaluative 한 면을 단순화한 세팅에서 보겠어. one that does not involve learning to act in more than one situation.

이런 nonassociative setting은 대부분의 사전 작업이 된 상태라 full 강화학습 문제의 복잡함을 피할 수 있어.

이 케이스를 보면 evaluative feedback 과 instructive feedback 이 어떻게 다른지 볼 수 있을거야.

우리가 살펴볼 nonassociative, evaluative feedback 문제는 k-armed bandit 문제야.

여기서 본 learning methods들을 확장해서 나중에 쓸 테니까 잘 봐. 나중에 full reinforcement learning problem 을 볼 생각을 하니 가슴이 설레지?

챕터 끝에서는 bandit problem 이 associative, 그러니까 action들이 한 가지 이상의 situation에서 발생하는, 그런 상태가 되면 무슨 일이 일어나는지 보자.

2.1 A k-armed Bandit Problem

다음 learning problem 을 보자.
반복적으로 k 개의 different option 또는 action 중 선택해야하는 문제.
매 choice마다 numerical reward를 받아. 근데 reward는 stationary prob. dist. 에서 나오고 그 dist.는 선택한 action에 depend해.
objective 는 maximize 하는 거야. 뭘? expected total reward를. 특정 time period 동안. 예를 들어 1000번 action selection을 하고 난 뒤지. time steps 이라고도 하고.
슬롯머신이 one-armed bandit 인데 이건 k 개지? 그래서 k-armed bandit.
반복적으로 action selection 을 하면 best lever를 땡겨서 상금을 maximize 하는 게 좋겠지?

여기선 각 k action들 마다 해당 action이 선택됐을 때의 expected reward가 있다고 하자.

A_t: time step t에서 선택한 action.
R_t: 거기에 대응하는 reward.
$q_*(a)$: a 라는 action이 선택됐을 때 expected reward. a의 value.

$q_*(a) := E [R_t | A_t=a]$

각 action의 value를 알고 있으면 문제는 쉽겠지. 항상 젤 value가 높은 걸 땡기면 되니까.
그럼 모른다고 가정을 해볼까?
근데 estimate는 할 수 있는겨.
$Q_t(a)$를 estimated value of action a at time step t로 하자.
$Q_t(a)$를 $q_*(a)$ 에 가깝게 하는 게 목표.

estimates of the action values를 알고 있으면, 매 time step 마다 estimated value가 최대인 게 최소한 하나는 있겠지. 이걸 greedy actions라고 해. 이 중에 하나 선택하는 걸 exploiting 한다고 해. 니가 현재 알고 있는 values of the actions 를 exploiting 한다고.
nongreedy actions 중에 선택하는 걸 exploring 한다고 하지.
왜냐면 너의 estimate를 향상시켜줄 테니까. 뭐에 대한 estimate?
nongreedy action's value 에 대한 estimate.

Exploitation은 one step 의 expected reward를 maximize할 수 있지.
Exploration은 total reward를 더 크게 만들 수도 있어. 빅픽쳐?

뒤에 time steps이 많이 남았으면 nongreedy action을 explore해서 greedy action보다 더 좋은 걸 발견해낼 수 있겠지?
더 좋은 걸 한 번 발견하고 나면 그걸 계속 exploit 할 수 있잖아. cool?

explore 할지 exploit 할지는 estimate, uncertainties, number of remaining steps 등 다양하고 복잡한 요인들에 얽혀 있어. 둘의 밸런스를 계산해낼 수학적 계산도 있지. 근데 stationarity 나 prior knowledge에 걸려 있는 가정이 넘나 강력… full reinforcement learning에는 적용하기 어려워.
여기선 간단한 balancing methods만 살펴보자.
exploit 만 하는 것보단 낫다는 걸 볼 수 있어.

2.2 Action-value Methods

시작해볼까? ㅋㅋ 뭘?
estimating the values of actions 하는 방법,
그 estimates를 action selection decision 할 때 사용하는 방법.

일단 true value of an action은 그 action이 선택됐을 때의 mean reward야. 기억하니?

estimate 첫 번째 방법. 받았던 reward averaging.

$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}}$$

이렇게.

저 라지1은 i 번째 step 에서 action이 a 일 때 1이고 아니면 0이겠지.

분모가 0이면, 그러니까 t 시점 이전에 a 한 적이 없으면 default로 $Q_t(a) = 0$ 으로 한다든지.

분모가 무한대로 가면 the law of large numbers로 $Q_t(a)$는 $q_*(a)$로 converge.

이걸 sample-average method for estimating action values 라고 한다.

다양한 estimation method들이 있겠지만 우선 이 estimate 가 action을 select 할 때 어떻게 사용되는지 보자.

가장 간단한 action selection rule 은 highest estimated value를 선택하는 거야. 그니까 아까 봤던 greedy action 중에 하나를 고르는거지. 한 가지 이상의 greedy action이 있으면 그 중에 하나를 랜덤하게 고른다든지 하는 거야.
이 greedy action selection method를

$$A_t \doteq \arg\max_a Q_t(a),$$

라고 합시다?
$Q_t(a)$ 를 가장 크게하는 a 가 A_t
그러니까, time step t에서 선택할 action이 $Q_t(a)$를 가장 크게하는 a 라는 말.

greedy action selection은 항상 현재 알고 있는 바를 exploit 해서 당장의 reward를 maximize 하지. inferior action에는 전혀 시간을 쓰지 않아. 더 좋아질 수 있는데 말이야.

여기서 초큼 더 좋아지는 방법을 생각해볼까?
대부분 greedily 하게 행동하다가 가끔, 확률 ε 정도로 가능한 action들 중 랜덤하게 뽑는거지. action-value estimates와 독립적으로.
요러케 하면 ε-greedy methods.
step이 증가할수록 모든 action이 무한하게 sampling 되니 $Q_t(a)$는 $q_*(a)$로 converge.
optimal action 을 선택할 확률이 1-ε 보다 큰 숫자로 수렴한다는 뜻이기도 함. 그러니까 거의 확실하쥬?

Exercise2.1
ε-greedy action selection에서, action이 두 개 있고 ε = 0.5 일 때, greedy action 이 선택될 확률은?

임시답안) 0.5 아닌가? estimate of values of action 이 있으니 greedy action은 정해져있는데 ε의 확률로 새로운 걸 explore 하니까?

Excercise2.2
k-armed bandit problem 에서 k=4라 치자. ε-greedy action selection을 하고 sample-average action-value estimates를 한다고 쳐. 모든 a에 대해 $Q_1(a) = 0$ 이라고 하고.
$A_1 = 1, R_1 = 1,$
$A_2 = 2, R_2 = 1,$
$A_3 = 2, R_3 = 2,$
$A_4 = 2, R_4 = 2,$
$A_5 = 3, R_5 = 0$ 라고 하자.
이 과정 중에 ε case가 나타나겠지? action 이 random하게 선택되는?
어떤 time step에서 definitely 발생했나?
어떤 time step에서 possibly 발생했나?

임시답안)
A1이 1인 건 random이겠지?
A2 에서 이전 estimate에 따르면 1이 선택돼야 하는데 2가 선택됐으니 ε 발생.
A3가 2가 된 건 1, 2가 R이 1이었으니 둘 중 하나 random 하게 택? possibly ε 가능성. 근데 왜 R이 2지? action에 따라 보상이 정해져 있는 게 아닌가?
A4는 2인 게 A3에서 R이 2가 나왔으니 그런 듯. sample average니까.
A5는 새로운 3이 나왔으니 ε case.

2.3 The 10-armed Testbed
greedy와 ε-greedy 효율을 대강 비교해보자.
k=10일 때 2000번 돌린 걸로.
action values $q_*(a)$는 N(0,1)을 따르는 분포에서 랜덤 픽. 표준정규분포임.
그러면 R_t는 $N(q_*(A_t), 1)$ 에서 뽑힘.
이걸 10-armed testbed라고 부르겠어.

어떤 learning method든 한 bandit problem에서 1000번 이상 경험하면서 향상시키면 performance와 behavior를 measure할 수 있다.
이걸 one run 이라고 부르자.

이걸 2000번 independent 하게 run 하면 각 bandit problem 마다 learning algorithm의 average behavior를 measure 할 수 있다.

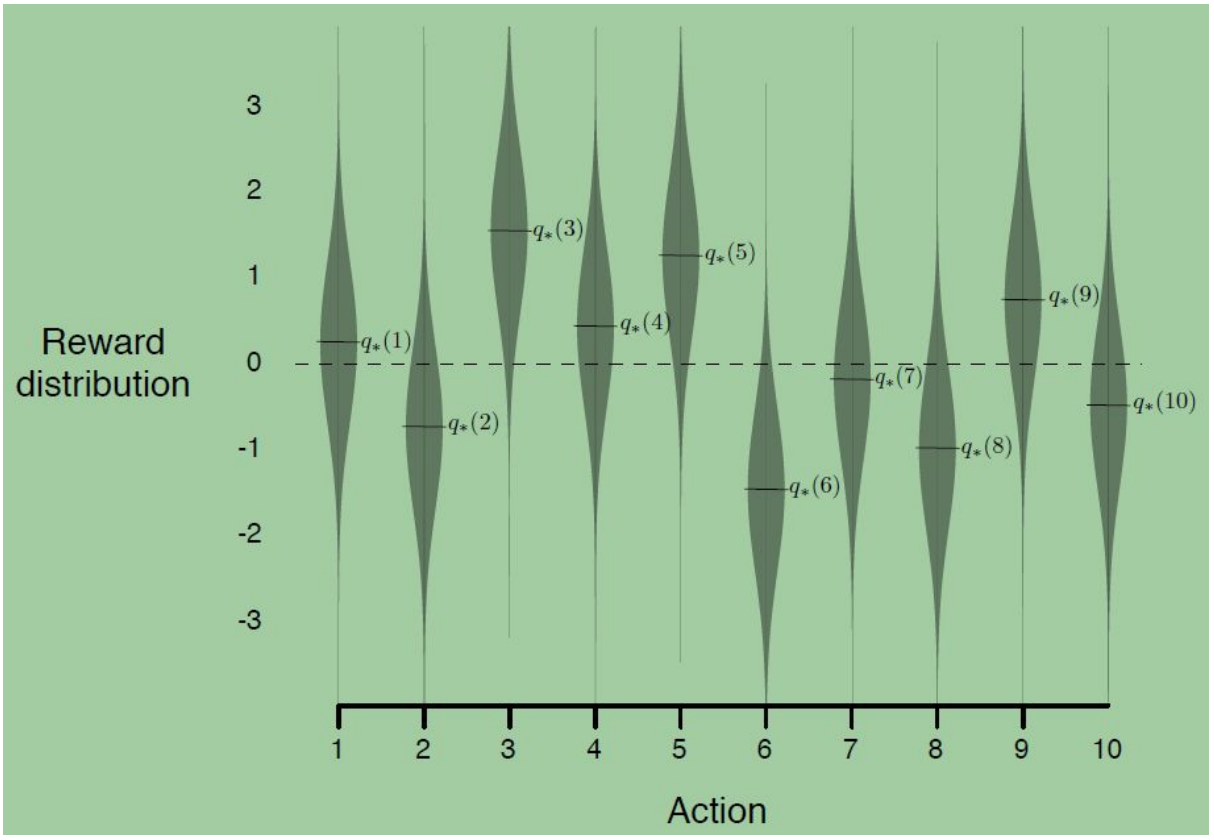


figure 2.1: 이런 분포로 뽑히는 중.

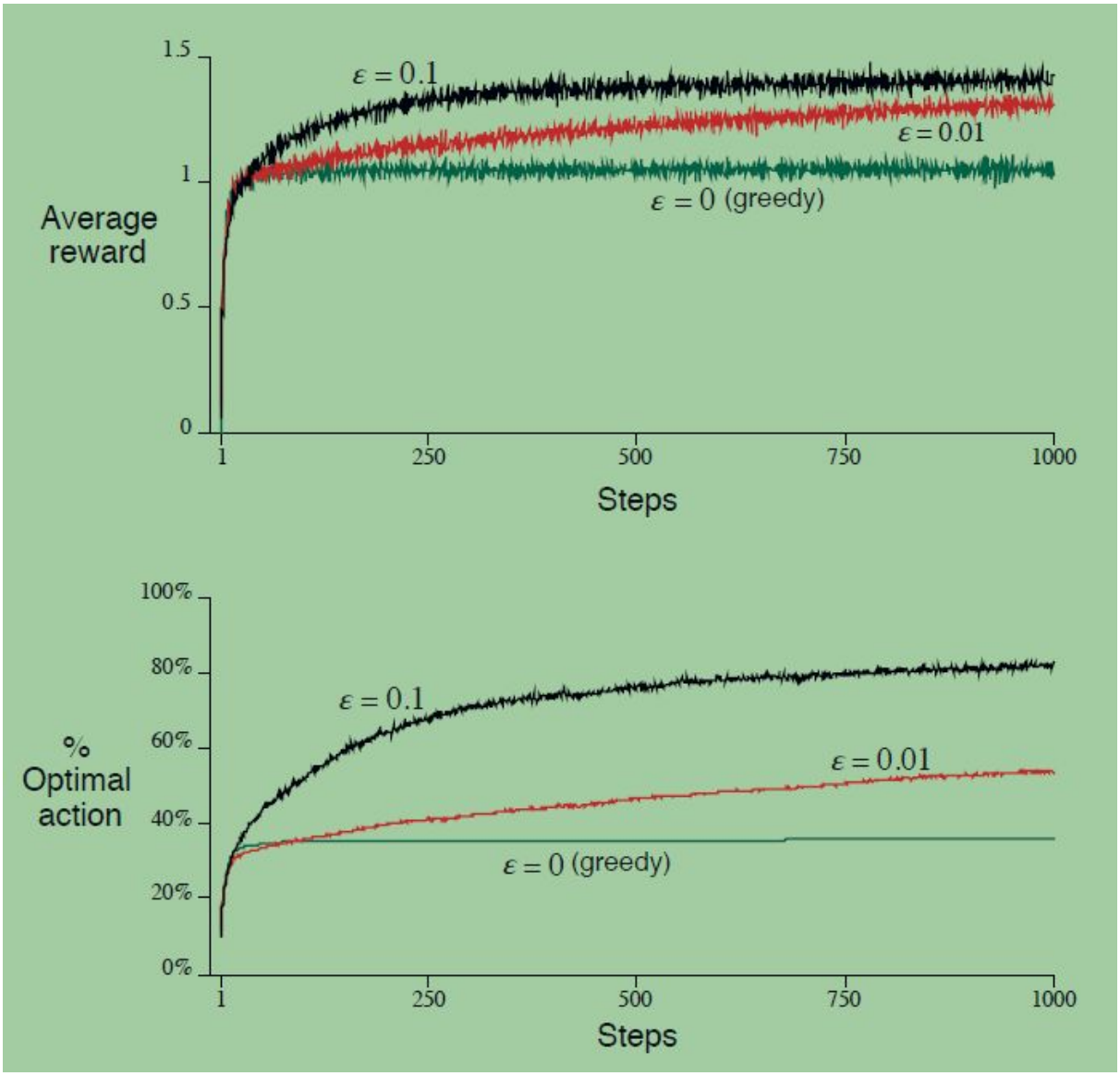


figure 2.2. Average performance of ε-greedy action-value methods.
서로 다른 bandit problem에서 2000 run 넘게 돌린 거 average.
모든 methods는 sample average로 action-value estimate.

결과 비교를 해 볼까?
greedy method와 ε-greedy methods 두 개를 비교한 거야.
전부 action-value estimates 는 sample-average 를 사용하고 있지.

fig 2.2 첫 번째 그래프는 experience에 따라 expected reward가 증가하고 있쥬?
greedy method가 극초반에는 다른 애들보다 살짝 improve가 빠르쥬?
근데 lower level에서 변동이 없어유.
reward-per-step이 1이유. 딴 애들은 1.55까지 가는디 말여유.
suboptimal action에 stuck 되기 때문이지.

fig 2.2 두 번째 그래프는, greedy method가 optimal action을 ⅓ 정도만 하고 있어.
나머지 ⅔ 는 optimal action의 initial samples가 잘못 된 거고 다시 돌아올 수가 없었지.
ε-greedy는 계속 explore 해서 optimal action을 찾을 확률을 향상시켰어.

ε=0.1로 하면 optimal action을 더 빨리 찾아. 하지만 91% 이상 선택하진 못하네.
ε=0.01로 하면 improve는 느리지만 더 낫다고?
그림이 잘못 된 거니 설명이 잘못 된 거니 내가 잘못 된 거니
ε를 time에 따라 감소시키는 것도 한 가지 방법.

reward의 variance가 클수록 ε-greedy가 좋겠지?
반대로 reward의 variance가 0이면 greedy method가 한 번만 action해도 true value를 아니까 금방 optimal action을 찾을 거고 explore하지 않으니 good.

하지만 deterministic case 에서도 exploring 하는 게 음청 좋아.
예를 들어, bandit task가 nonstationary라고 해 보자. 그러니까 true values of actions 가 매 time 바뀌는 거지. 이런 경우엔 exploration이 필요해. nongreedy action이 greedy one 보다 좋아졌을지 알아야하니까.

앞으로는 nonstationary 가 주된 환경이 될 거야.
설령 stationary하고 deterministic한 case라고 해도, decision task가 시간에 따라 바뀌고 policy가 바뀌는 상황을 볼 거야.

강화학습은 exploration과 exploitation의 balance가 중요해.

Exercise 2.3
fig 2.2를 봤을 때 어떤 method가 제일 좋을까? long run, cumulative reward, probability of selecting the best action의 관점에서. 얼마나 좋을까? quantitatively하게 대답해보렴.

임시답안)
저거 선형으로 기울기 계산해서 역전하는 지점 찾으면 되나? ε=0.01이 젤 좋겠네.
역전하는 지점의 step, reward, prob 찾아봅시다?
아니면 막 확률 계산해서?
근데 greedy method의 optimal action prob이 왜 10%가 넘어 30%가 되는거지?
최초 탐험은 어떻게 하나…

2.4 incremental implementation. 점진적 구현?
여태까지 봤던 action-value method는 action value를 estimate 하는 방법으로 sample averages of observed rewards 를 썼지. 리멤버?

이제 이 average들을 어떻게 효율적으로 계산하는지 보자.
constant memory, constant per-time-step 계산한다네.

우선 single action 에 초점을 맞춰보자. notation을 간단하게 하려고잉.

R_i: reward. 이 action의 i번째 selection을 했을 때 받는 reward.
Q_n: estimate of action value. n-1 번 select 됐을 때의 estimate.

$$Q_n \doteq \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}.$$

모든 reward에 대해 기록을 보관하고 필요할 때마다 계산하겠지?
근데 그러면 메모리랑 계산량이 시간에 따라 늘어날 거야.
추가되는 reward는 추가로 메모리를 잡아먹고 계산도 추가시켜.

근데 뭐 꼭 그럴 필요는 없어.
incremental formulas로 update 시키면 돼.

$$
\begin{aligned}
Q_{n+1} &= \frac{1}{n}\sum_{i=1}^{n} R_i \\
&= \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\left(R_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\Big(R_n + (n-1)Q_n\Big) \\
&= \frac{1}{n}\Big(R_n + nQ_n - Q_n\Big) \\
&= Q_n + \frac{1}{n}\Big[R_n - Q_n\Big],
\end{aligned}
$$

바로 이거시다. 아주 trivial 하지? 고등학교 때 한 점화식 생각남.
new average of all n rewards는 이렇게 계산이 된다.
심지어 n=1일 때도 성립.
메모리는 Q_n, n, small computation for new reward. 만 있으면 돼.

A simple bandit algorithm

Initialize, for $a = 1$ to k:
  $Q(a) \leftarrow 0$
  $N(a) \leftarrow 0$

Loop forever:
  $A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$
  $R \leftarrow bandit(A)$
  $N(A) \leftarrow N(A) + 1$
  $Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\big[R - Q(A)\big]$

Pseudocode.
k개의 action에 대해.
Q, N은 0부터.
평생 돌려보자. 뭘?
Q가 최대가 되는 action. 가끔은 random action.
해당 action을 했을 때의 보상을 R에 저장.
해당 action에 대한 N은 +1 해서 Q(A) 다시 계산.

NewEstimate <- OldEstimate + StepSize [Target - OldEstimate]

이런 형태는 책 전반에 걸쳐 나오니 기억해 두도록.

[Target - OldEstimate]는 estimate의 error. old estimate 가 평균이니까 말이지?
target을 향해 step을 밟을수록 줄어들 거야.
target은 움직이고 싶은 방향을 추정.
위의 case에서는 n-th reward가 target.

step-size는 time step 마다 바뀌는데,
여기서는 action a의 n-th reward를 처리하는 데 step-size parameter로 1/n 를 쓴다.
앞으로 step-size parameter를
α라고 하겠슈.
$\alpha_t(a)$ 는 뭐라고? t 시점에서 action a를 했을 때의 step-size. 지금은 1/n.

2.5 Tracking a Nonstationary Problem

averaging method는 stationary bandit problem 에 적합했다. 그러니까 reward probabilities가 시간에 따라 바뀌지 않았잖아.
근데 실제로는 nonstationary 가 많다고 했지?
그럴 땐 recent rewards에 weight 를 주는 게 맞겠지? long-past rewards는 적게 주고잉?
보통 constant step-size parameter를 써서 이걸 실현하지.
예를 들어, 아까 봤던 incremental update rule 있지? n-1 past rewards의 average Q_n 을 계산했던. 그땐 최근 게 오히려 작아졌잖아. 그게 문젠가본데.

그게

$$Q_{n+1} \doteq Q_n + \alpha\Big[R_n - Q_n\Big]$$

이렇게 바뀌는 거지. α 가 constant여. (0,1] 이여.
Q_{n+1} 은 weighted average of past rewards 가 되는겨.

$$\begin{aligned}
Q_{n+1} &= Q_n + \alpha\Big[R_n - Q_n\Big] \\
&= \alpha R_n + (1-\alpha)Q_n \\
&= \alpha R_n + (1-\alpha)\left[\alpha R_{n-1} + (1-\alpha)Q_{n-1}\right] \\
&= \alpha R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha)^2 Q_{n-1} \\
&= \alpha R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha)^2 \alpha R_{n-2} + \\
&\qquad \cdots + (1-\alpha)^{n-1}\alpha R_1 + (1-\alpha)^n Q_1 \\
&= (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i .
\end{aligned} \tag{2.6}$$

알겠지?
이걸 wieghted average라고 부르는 이유는 weights 의 합이 1, 그러니까

$$(1-\alpha)^n + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} = 1$$

이기 때문이야. 확인 해 보슈. 난 안 했음. father sutton을 믿으니까. 근데 암산해도 될듯?

$\alpha(1-\alpha)^{n-i}$: 이 weight는 얼마나 이전에 받은 reward인지에 depend.
$1-\alpha$ 이 1보다 작으니 intervening rewards 수가 늘어날수록 R_i에 붙은 weight는 줄어들거야.
weight 는 exponentially decay. $1-\alpha$ 만큼?
$1-\alpha$ =0 이면 모든 weight 가 last reward R_n에 갈 것.
그래서 exponential recency-weighted average 라고 부름.

step-size parameter를 step마다 다르게 하는 것도 convenient하다.
$\alpha_n(a)$: 이건 step-size parameter야. 어떤? action a의 n-th selection 이후 reward 받을 때 쓰일 step-size parameter.
$\alpha_n(a) = 1/n$ 일 때 sample-average method라고 했고, 큰 수의 법칙으로 반드시 수렴한다고 했지.
하지만 당연히, sequence $\{\alpha_n(a)\}$의 모든 choice에 대해 수렴을 보장하진 않아.
stochastic approximation theory 에 의하면 수렴을 보장하는 어떤 조건들이 있지.

$$\sum_{n=1}^{\infty} \alpha_n(a) = \infty \qquad \text{and} \qquad \sum_{n=1}^{\infty} \alpha_n^2(a) < \infty$$

바로 이것.
첫 번째 조건은 step이 어떤 초기 조건이나 random 변동도 이겨낼 정도로 크다는 걸 보장하기 위해 필요.
두 번째 조건은 step이 수렴을 보장할 정도로 충분히 작다는 걸 보장.

sample-average case에서는 두 가지 조건이 다 충족되지만 constant step-size parameter $\alpha_n(a) = \alpha$ 에서는 아냐. 노트에 적어놔.
constant에서는 두 번째 조건이 안 맞지. 그러니까 estimate가 절대 수렴 안한다는 뜻인데, 대신 가장 최근에 받은 reward에 대응해서 계속 달라져.
이건 nonstationary 환경에 아주 좋지.
게다가, 위 수렴조건을 만족하는 step-size parameter들의 sequences 는 엄청 느리게 converge하거나 tuning을 많이 해야해. 만족스러운 convergence rate를 얻으려면. 그래서 이론적 연구에서나 쓰지 실제에선 잘 안 써.

Exercise 2.4
step-size parameter α_n 이 constant가 아니면, Q_n 이 weighted average of previously received rewards with a weighting 인데 위에서 봤던 (2.6)식이랑은 다른 거지.
각 prior reward의 weighting 이 어떻게 될까? general case에서. (2.6)이랑 비슷하게.
sequence of step-size parameters의 관점에서.

임시 답안)
constant가 아니면 매 step마다 바뀐다는 건데… 그럼 n으로 된 함수? 1/n 처럼?
그럼 (2.6)에서 α 대신 n으로 된 함수가 들어간다는 건데 뭐가 다른가?
α의 범위에 따라 구분지어주면 되는건가. 0과 1사이일 때, 1보다 클 때, 이렇게?
α가 매번 달라져서 매 스텝 다른 α를 써서 보여주는 것도 방법.

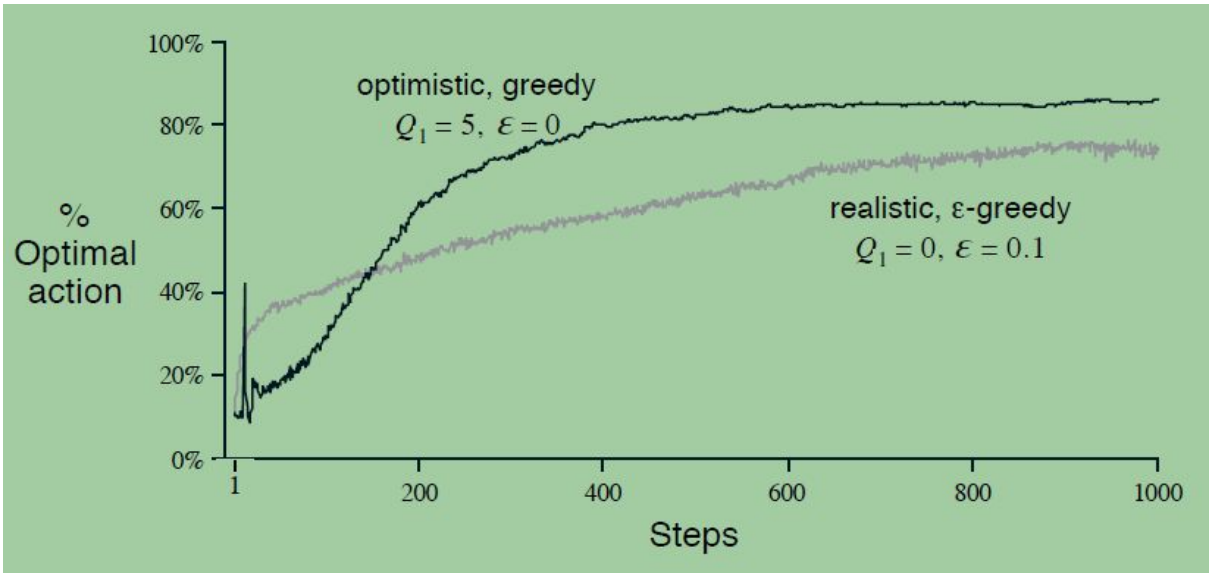
Exercise 2.5(programming)
헐. sample-average method가 nonstationary problem에서 갖는 어려움을 묘사하는 실험을 design and conduct 해보렴? 10-armed testbed를 변형해서 말이지. $q_*(a)$가 전부 똑같이 시작하면서 independent random walks(그러니까 N(0,0.01)에서 $q_*(a)$가 나오는겨)
fig2.2처럼 plot도 그려보렴? sample average를 쓰는 action-value method도 해 보고, constant step-size parameter를 쓰는 action-value method도 해 봐. α=0.1 이다. ε=0.1이고 10,000 step 짜리로다가. ㅇㅋ?

2.6 Optimistic Initial Values

휴. 힘들었다. exercise 다 하심? 이 문제를 풀지 않고는 다음 문제들은 엄두도 안 나리라는 선견지명으로 시간 좀 들였다.
하다보면 눈치 챘겠지만 여태까지 한 methods 는 initial action-value estimates에 dependent한 면이 있었어. 그러니까 $Q_1(a)$에 말이지.
통계학 용어로는 이 methods 들이 initial estimates에 biased 돼 있다고 하지.

sample-average method 에선 모든 action이 최소 한 번 선택된 순간부터 bias가 사라져.
하지만 constant α method 에선 bias 가 사라지지 않아. (2.6) 식처럼 시간에 따라 줄어들긴 하지.
실제론 이런 종류의 bias는 문제가 되진 않아. 가끔 도움이 될 때도 있지.
사실 단점은 initial estimate를 전부 0으로 하면 initial estimate는 user가 pick 해야하는 parameter set이라는 거야.
장점은 어느 정도 수준의 reward를 기대할 수 있을지 사전 지식을 쉽게 얻을 수 있다는 거지.

initial action values는 exploration을 encourage하는 데 쓰이기도 해.
위에서 했던대로 다 0으로 하는 대신 +5로 세팅한다고 쳐.
$q_*(a)$ 이 N(0, 1)에서 나왔던 거 기억하니?
그럼 initial estimate가 +5라는 건 아주 optimistic 하지.
하지만 이런 optimism 이 action-value methods가 explore 하도록 encourage해.
어떤 action이 선택됐든 reward는 starting estimate보다 작겠지. 그럼 learner가 실망하면서 다른 action으로 바꿀거야.
결과는, 모든 action이 여러 차례 시도될거야. value estimates가 converge하기 전에.
system 은 greedy action이 매번 선택된다해도 exploration을 충분히 하게 되겠지.



(fig2.3) 10-armed testbed에서 optimistic initial action-value estimates 의 영향.
둘 다 α = 0.1

(fig2.3)은 performance를 보여주고 있다. 무슨 performance? 10-armed bandit testbed인데 greedy method. 모든 a가 $Q_1(a)$ = +5 다.
비교한 건 ε-greedy method에 $Q_1(a)$ = 0.
초반에는 optimistic이 안 좋지. explore를 더 많이 하니까. 근데 결국 더 나아져. 왜냐면 exploration이 점차 줄고 greedy selection을 하니까.
우린 이걸 exploration optimistic initial values 를 encouraging 하는 테크닉이라 불러.

stationary problem 에선 아주 효율적인 트릭.
하지만 일반적으로 encouraging exploration하는 방법으론 별로.
예를 들어, nonstationary problem 에는 구려. 왜냐면 exploration을 위한 추진력이 일시적이기 때문이야.
task가 바뀌면 exploration을 위한 새로운 need를 만들어 내는 데 이 method는 도움이 안 돼.
특수한 경우의 initial condition에 focus한 method들은 일반적인 nonstationary case에 도움이 안 돼.
beginning of time은 한 번만 발생하지. 그러니 여기 너무 focus하면 안 돼.
이런 비판은 sample-average에도 마찬가지로 적용되지. 그것도 beginning of time 을 special event로 취급하고 뒤따르는 reward들을 동일 weight로 averaging하니까.

그래도 이 method 들은 simple해서 좋아. 하나씩 또는 조합해서 쓰면 실제로 좋을 때도 있어.
이 책에서도 이런 simple exploration techniques를 자주 쓸 겨.

(exercise 2.6)
(fig2.3)는 reliable 하지. 왜냐면 2000번의 individual, randomly chosen 10-armed bandit tasks 를 average 한 거니까. 그럼, 왜 optimistic method 초반에 oscillation과 spikes 가 나타날까? 그러니까, 뭣이 이 method가 평균적으로, 특히 early steps 에서 요상한 better or worse가 나타나게 만드는 걸까?

(임시답안)

그러게. 왜그럴까..
과정을 천천히 밟아보자. 10개 전부 5점에서 하나가 랜덤리 선택돼. 걘 보상이? 짜.
N(0,1)이니까. 그러면 해당 action은 Q가 마이너스가 더해지겠지. 그 다음에 9개 중에 하나가 선택 돼. 걔도 Q가 마이너스. 11 step에선 $Q_2(a)$가 높은 애가 다시 선택되겠지.
weighted average로 해야하나. (1-α)가 5점에 계속 곱해지고. N(0,1)에서 뽑히는 애가 (1-α) 곱해져서 더해진다.. 마찬가지 10step 까진 10개 다 돌아갈거고. reward로 더해지는 애가 워낙 적으니.. Q가 줄어드는 폭보다 reward가 더해지는 게 커지기 전까지 한동안은 다 돌아가겠지. 그러면서 충분히 작아지면 기본 ε-greedy 처럼 행동하겠지.
돌아가면서 다 하니까 optimal value 를 할 때 spikes 치는 거 아닌가.

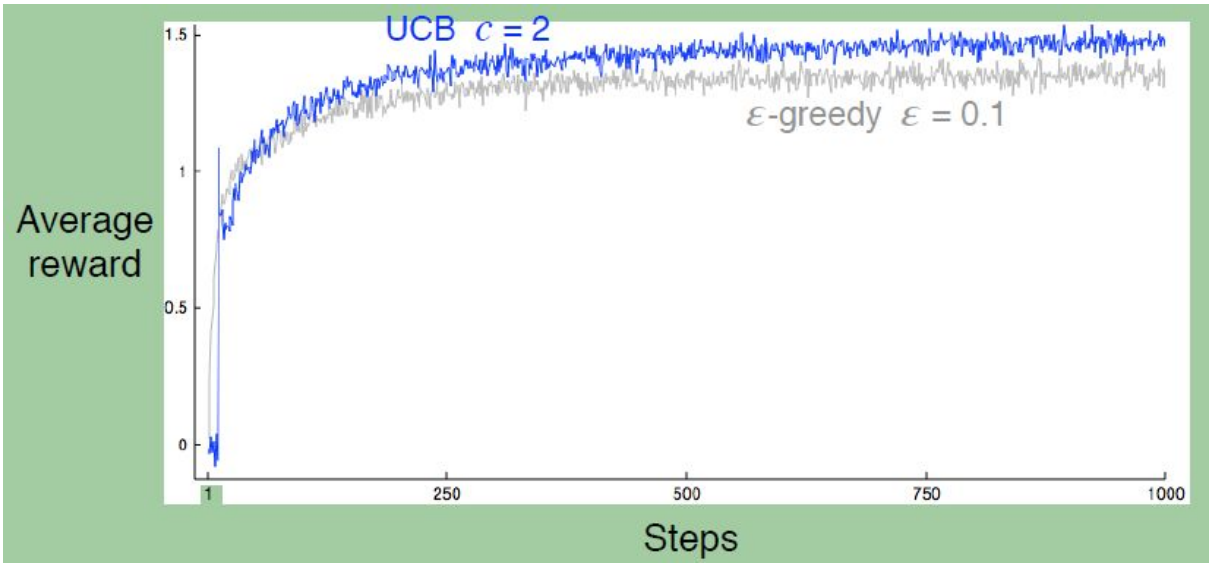
2.7. Upper-Confidence-Bound(UCB) Action Selection

Exploration은 필요해. 왜? action-value estimate는 정확하지 않으니까.
greedy action은 현재엔 best로 보이지만 다른 action이 더 좋았을 수도 있지.
ε-greedy action selection은 non-greedy action을 시도하지만 greedy에 가깝거나 특히 불확실한 것을 선택하진 못한다.
potential optimal 을 생각해서 non-greedy action을 선택하는 게 좋을거야. 근데 그 optimal은 estimate가 maximal이나 uncertainty에 얼마나 가까운지를 고려해야겠지.
action 을 선택하는 한 가지 방법을 소개해줄게.

$$A_t \doteq \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right]$$

바아로 이거시다. $ln\ t$ 는 자연로그. natural logarithm.
$N_t(a)$는 t시점 이전에 action a를 한 횟수.
c>0 는 exploration의 degree를 control 하지.
$N_t(a)$=0 이면 a는 maximizing action이라 하게쒀. 그럼 한 번도 한 적 없으면 반드시 하겠구먼.

이 UCB action selection의 idea는 square-root term 이 a value의 estimate가 지닌 uncertainty 또는 variance를 알 수 있는 measure라는 게지. 에헴.
그래서 최대가 되는 quantity는 일종의 upper bound야. 어떤 upper bound? action a의 가능한 true value의 upper bound.
c는 confidence level 을 결정하지.
a가 선택될 때마다 uncertainty는 줄어들 거야.
그러니까, $N_t(a)$가 증가하면 uncertainty term은 줄어들어.
반대로, a 말고 다른 애들이 선택되면, t는 증가하지만 $N_t(a)$는 그대로. 그러면 uncertainty는 증가하지.
자연로그를 쓴 건 시간에 따른 증가가 점차 작아지게, 하지만 unbounded 되게 만든 거고.
모든 action이 결국 선택되겠지만, lower estimate의 action이나 이미 자주 선택된 action은 시간이 지남에 따라 적게 선택된대. 이게 다 한 식에 들어있다니.



(fig2.4) UCB selection의 Average performance. 10-armed testbed임.
ε-greedy 보다 일반적으로 좋다. 초반 k step 빼고는. 그땐 아직 untried actions 중에 랜덤으로 고르는 중이니까.

근데 UCB는 ε-greedy보다 extend하기 어려워. 일반적인 강화학습 세팅에.

어려움 1. nonstationary problem을 상대할 땐

$$Q_{n+1} \doteq Q_n + \alpha\left[R_n - Q_n\right]$$

이거보다 복잡한 method가 필요해.

어려움 2. large state spaces 상대할 때.
특히 function approximation 할 때. 나중에 할 거임.

2.8 Gradient Bandit Algorithms

여태까진 뭐했다? 눈을 감고 생각해보고 다음 줄로 넘어가자.
action value를 estimate하는 method를 생각해보고 그 estimate로 action selection 했지.
근데 이 방법만 있는 건 아냐. 그럴 줄 알았지.
이번에는 각 action의 numerical preference를 학습하는 방법을 알아볼까?
$H_t(a)$라고 쓸 거야. 뭐를? 각 action의 numerical preference!

preference가 커질수록 action이 자주 취해지는거지.
근데 preference는 reward의 관점에서 해석되진 않아.
다른 action에 대해 relative preference 가 중요할 뿐.
그러니까 모든 preference에 100을 더한다고 action probabilities에 영향을 주진 않겠지?
action probabilities는 soft-max distribution에 따라 결정돼.(i.e., Gibbs or Boltzmann dist.)
식은 다음과 같지.

$$\Pr\{A_t = a\} \doteq \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \doteq \pi_t(a)$$

$\pi_t(a)$ 는 첨봤지? action a를 할 확률이야. time t에서 말이지.
preference를 계산해서 0에서 1 scale로 만들어 확률처럼 쓰자는 말?

처음 preference는 전부 같아. $H_1(a)=0$ 처럼. 모든 action이 같은 확률을 가지게.

여기서 갑자기 퀴즈?
(Exercise2.7)
action이 2개일 때, soft-max distribution과 logistic, sigmoid 처럼 통계학이나 신경망에서 자주 쓰는 함수들로 만든 게 같다는 걸 보여라.

(임시답안)
이건 찾아보면 금방 나올 듯? 나도 어디선가 본 듯.

다시 내용으로 돌아와서.
이 세팅을 위한 natural learning algorithm이 있어. 여기서 말하는 세팅은 preference 세팅.
뭐냐면 stochastic gradient ascent. 야. 허메 descent가 아니네. 첨본다.
각 스텝마다 action A_t 가 선택되고 reward R_t를 받고 나면 preferences가 update 돼쥬. 그럼 $\pi_t(a)$ 도 update 되것네.

$$H_{t+1}(A_t) \doteq H_t(A_t) + \alpha\big(R_t - \bar{R}_t\big)\big(1 - \pi_t(A_t)\big), \qquad \text{and}$$
$$H_{t+1}(a) \doteq H_t(a) - \alpha\big(R_t - \bar{R}_t\big)\pi_t(a), \qquad \text{for all } a \neq A_t$$

수식 쓰다가 위에 bar 나오길래 포기하고 캡처.
$\alpha > 0$ 인 step-size parameter.

R_t bar는 실수인데 average야. 어떤? t time까지 포함한 모든 reward의.
incrementally 계산할 수 있었지? 리멤버? 기억안날까봐 가져옴.

$$\begin{aligned}
Q_{n+1} &= \frac{1}{n}\sum_{i=1}^{n} R_i \\
&= \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\left(R_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\Big(R_n + (n-1)Q_n\Big) \\
&= \frac{1}{n}\Big(R_n + nQ_n - Q_n\Big) \\
&= Q_n + \frac{1}{n}\Big[R_n - Q_n\Big],
\end{aligned}$$

stationary 일 때.

$$Q_{n+1} \doteq Q_n + \alpha\Big[R_n - Q_n\Big]$$

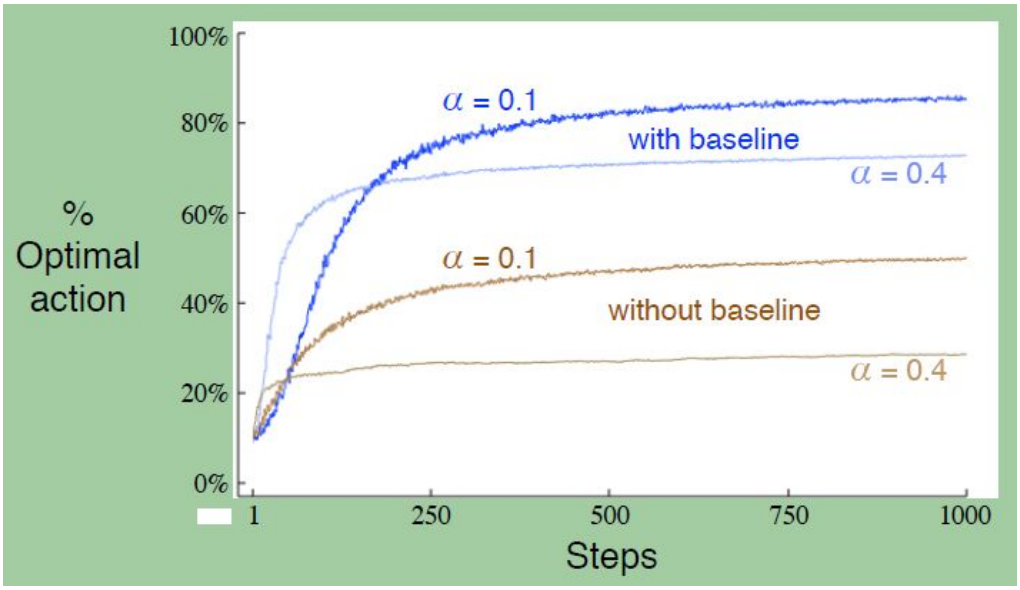
nonstationary 일 때. 이렇게 했었음.

R_t bar 는 reward가 비교되는 baseline 역할이야.

만약 reward가 baseline보다 높으면, 미래에 taking A_t 할 확률이 커지는 거야. 해봤더니 보상이 크네? 굿. 담에 또 하자.
만약 reward가 baseline보다 낮으면, 확률은 줄어들지. 해봤더니 보상이 짜네? 뱉. 담엔 별로.

그리고 non-selected actions는 반대로 움직인대.
딴 걸 했더니 보상이 크네? 이건 하지 말까?
딴 걸 했더니 보상이 짜네? 이걸 해볼까?

이제보니 α 앞의 부호가 반대네.
제일 뒤에 확률은 왜 곱해주는 걸까..



(fig2.5) average performance야. gradient bandit algorithm 의. 하나는 baseline 있는 거고 하나는 없는 거. 아직 10-armed-testbed 문제.
q.(a)가 0에 가까울 때보다 +4에 가깝게 선택될 때.
이게 뭔 말이여? 설명을 좀 더 보자.

(fig2.5)는 gradient bandit algorithm의 결과. variant of the 10-armed testbed에서.
true expected rewards가 N(+4,1) 에서 뽑혔을 때. 아 위에 그림 설명이 이 말이었군여.
이렇게 모든 reward를 shifting up 하는 건 gradient bandit algorithm에 1도 영향이 없다.
왜냐면 baseline term 역할을 하는 reward 때문이지. 동시에 새로운 레벨로 맞춰지지 않겠니.
다만, baseline이 없어지면, 그러니까 R_t bar 가 constant 0이면 performance는 심히 degraded 될 거야. 기준이 없어져서 단순히 R_t 에 따라 업데이트 돼서 그런가?

The Bandit Gradient Algorithm as Stochastic Gradient Ascent 에 대한 본격적인 설명.
알고보니 증명.

deeper insight를 얻을 수 있을 거야. 뭐에 대한? gradient bandit algorithm에 대한.
어떻게? 이걸 stochastic approximation to gradient ascent로 이해하면.

gradient ascent에선, 각 preference $H_t(a)$는 increment 될 겨. performance에 대한 increment에 비례해서.

$$H_{t+1}(a) \doteq H_t(a) + \alpha \frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)}$$

여기서 measure of performance는 expected reward임. 아래 식.

$$\mathbb{E}[R_t] = \sum_x \pi_t(x) q_*(x)$$

measure of the increment's effect는 performance measure를 preference로 편미분한거.

근데 지금은 $q_*(x)$를 모르니까 gradient ascent를 적용하진 못하겠지?
근데 사실

$$H_{t+1}(A_t) \doteq H_t(A_t) + \alpha\big(R_t - \bar{R}_t\big)\big(1 - \pi_t(A_t)\big), \qquad \text{and}$$
$$H_{t+1}(a) \doteq H_t(a) - \alpha\big(R_t - \bar{R}_t\big)\pi_t(a), \qquad \text{for all } a \neq A_t$$

이 알고리즘의 업데이트는

$$H_{t+1}(a) \doteq H_t(a) + \alpha \frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)}$$

이거나 같아. expected value를 활용하는 면이? expected value 측면에서?
그래서 알고리즘을 stochastic gradient ascent 의 instance로 만들지.
calculations는 beginning calculus만 필요하지만 몇 스텝 걸리긴 해.
먼저 exact performance gradient 를 자세히 살펴보자.

$$\begin{aligned}
\frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} &= \frac{\partial}{\partial H_t(a)} \left[\sum_x \pi_t(x) q_*(x) \right] \\
&= \sum_x q_*(x) \frac{\partial \pi_t(x)}{\partial H_t(a)} \\
&= \sum_x \big(q_*(x) - B_t\big) \frac{\partial \pi_t(x)}{\partial H_t(a)},
\end{aligned}$$

B_t는 baseline이야. 아무 scalar나 가능.
x에 depend하지 않는 걸로. 근데 x가 action 아닌가? 아 뭐, 가능.
baseline을 equality 에 영향 없이 추가할 수 있었지?
이유는 모든 action에 대해 gradient 합이 0이 되기 때문.

$$\sum_x \frac{\partial \pi_t(x)}{\partial H_t(a)} = 0$$

이렇게.
$H_t(a)$가 변함에 따라 어떤 action은 확률이 올라가고 어떤 건 떨어지는데
sum of the changes는 반드시 0이어야 해. 왜냐면 확률의 합은 항상 1이니까.

다음 단계는 $\pi_t(x)/\pi_t(x)$ 를 곱해주는 거여.

$$\frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} = \sum_x \pi_t(x)\big(q_*(x) - B_t\big)\frac{\partial \pi_t(x)}{\partial H_t(a)}/\pi_t(x)$$

요로코롬.

이제 방정식은 expectation 꼴이 된다. random variable A_t의 x 값을 다 합쳐서 확률 곱하는거지.

$$\begin{aligned}
&= \mathbb{E}\left[\big(q_*(A_t) - B_t\big)\frac{\partial \pi_t(A_t)}{\partial H_t(a)}/\pi_t(A_t)\right] \\
&= \mathbb{E}\left[\big(R_t - \bar{R}_t\big)\frac{\partial \pi_t(A_t)}{\partial H_t(a)}/\pi_t(A_t)\right],
\end{aligned}$$

이렇게 됐네?
여기선 baseline B_t = R_t bar 로 정했고 $q_*(A_t)$를 R_t 로 바꿨지. 막 바꿔도 되냐고?

$$\mathbb{E}[R_t|A_t] = q_*(A_t)$$

이런 식이 기다리고 있었다 요놈아. 이게 애초의 정의였나?

$$\frac{\partial \pi_t(x)}{\partial H_t(a)} = \pi_t(x)\big(\mathbb{1}_{a=x} - \pi_t(a)\big)$$

이제 이거는 일단 맞다 치고. 1 저건 a=x 일 때만 1이고 아니면 0인 거.

$$\begin{aligned}
&= \mathbb{E}\big[\big(R_t - \bar{R}_t\big)\pi_t(A_t)\big(\mathbb{1}_{a=A_t} - \pi_t(a)\big)/\pi_t(A_t)\big] \\
&= \mathbb{E}\big[\big(R_t - \bar{R}_t\big)\big(\mathbb{1}_{a=A_t} - \pi_t(a)\big)\big].
\end{aligned}$$

그러면 이게 되지.

뭐하고 있는지 까먹으면 안 되지.
우리 계획은 performance gradient를 뭔가 매 step 샘플링할 수 있는 것의 기대값으로 쓰는거지. 방금 한 것처럼? 방금 뭘 했지?ㅋㅋ
그러고 매 스텝 sample 에 비례해서 update 하는 거지.

$$H_{t+1}(a) \doteq H_t(a) + \alpha\frac{\partial \mathbb{E}\left[R_t\right]}{\partial H_t(a)}$$

위 식의 performance gradient 를 a sample of the expectation 으로 바꾸면

$$H_{t+1}(a) = H_t(a) + \alpha\big(R_t - \bar{R}_t\big)\big(\mathbb{1}_{a=A_t} - \pi_t(a)\big), \qquad \text{for all } a,$$

이래 돼요.
전에 봤던

$$H_{t+1}(A_t) \doteq H_t(A_t) + \alpha\big(R_t - \bar{R}_t\big)\big(1 - \pi_t(A_t)\big), \qquad \text{and}$$
$$H_{t+1}(a) \doteq H_t(a) - \alpha\big(R_t - \bar{R}_t\big)\pi_t(a), \qquad \text{for all } a \neq A_t$$

이 original algorighm과 equivalent 한 것이 보이느뇨.

이제 남은 건 우리가 가정했던

$$\frac{\partial \pi_t(x)}{\partial H_t(a)} = \pi_t(x)\big(\mathbb{1}_{a=x} - \pi_t(a)\big)$$

이걸 보이는 일이지.

standard quotient rule for derivatives가 기억나느뇨.

$$\frac{\partial}{\partial x}\left[\frac{f(x)}{g(x)}\right] = \frac{\frac{\partial f(x)}{\partial x}g(x) - f(x)\frac{\partial g(x)}{\partial x}}{g(x)^2}$$

바로 이거시다. 이걸 이용하면

$$\begin{aligned}
\frac{\partial \pi_t(x)}{\partial H_t(a)} &= \frac{\partial}{\partial H_t(a)}\pi_t(x) \\
&= \frac{\partial}{\partial H_t(a)}\left[\frac{e^{H_t(x)}}{\sum_{y=1}^{k} e^{H_t(y)}}\right] \\
&= \frac{\frac{\partial e^{H_t(x)}}{\partial H_t(a)}\sum_{y=1}^{k} e^{H_t(y)} - e^{H_t(x)}\frac{\partial \sum_{y=1}^{k} e^{H_t(y)}}{\partial H_t(a)}}{\left(\sum_{y=1}^{k} e^{H_t(y)}\right)^2} && \text{(by the quotient rule)} \\
&= \frac{\mathbb{1}_{a=x}e^{H_t(x)}\sum_{y=1}^{k} e^{H_t(y)} - e^{H_t(x)}e^{H_t(a)}}{\left(\sum_{y=1}^{k} e^{H_t(y)}\right)^2} && \text{(because } \tfrac{\partial e^x}{\partial x} = e^x\text{)} \\
&= \frac{\mathbb{1}_{a=x}e^{H_t(x)}}{\sum_{y=1}^{k} e^{H_t(y)}} - \frac{e^{H_t(x)}e^{H_t(a)}}{\left(\sum_{y=1}^{k} e^{H_t(y)}\right)^2} \\
&= \mathbb{1}_{a=x}\pi_t(x) - \pi_t(x)\pi_t(a) \\
&= \pi_t(x)\big(\mathbb{1}_{a=x} - \pi_t(a)\big). && \text{Q.E.D.}
\end{aligned}$$

이거시 가능.

증명하겠다는 건 줄 알았으면 마음의 준비를 좀 했을텐데.
지금 본 게 뭐냐면,
expected update of the gradient bandit algorithm 이
gradient of expected reward와 같다는 것이야.
그래서 algorithm이 stochastic gradient ascent의 instance라는 거지.
이 사실로 algorithm 이 robust한 convergence properties를 갖고 있다는 걸 알 수 있지.

옆에 공책에 적어두렴. 어서. 이 과정에서 우린 reward baseline의 어떤 properties 도 필요하지 않았어. selected action에 의존하지 않는다는 걸 빼곤 말이지.

예를 들어, reward baseline 을 0이나 1000으로 세팅해도 algorithm은 여전히 stochastic gradient ascent의 instance일 거란 말이지.
baseline을 선택하는 건 algorithm의 expected update에 영향을 미치지 않아.
하지만 update의 variance에 영향을 미치고, 그래서 rate of convergence에도 영향이 있지.
baseline을 average of the rewards로 고르는 건 best는 아닐지도 모르지만, 간단하고 실용적이야.

2.9 Associative Search (contextual bandits)

지금까지 2챕터에선 nonassociative task만 다뤘지. 긍께 여태까진 different actions과 different situations 를 associate 할 필욘 없었단 말이여.
이런 task들에선 learner는 single best action을 찾기 위해 노력해. task가 stationary 할 때.
또는 task가 nonstationary일 때 time 에 따라 바뀌는 best action을 찾으려고 노력하지.
하우에버, general reinforcement learning task에선, 한 가지 이상의 situation이 있고, 목표는 policy를 learn 하는 거야.
policy가 뭐냐고?
a mapping from situations to the actions that are best in those situations.
어떤 situation에서 best인 actions으로의 mapping.

full problem 에 해당하는 stage를 set 하기 위해,
nonassociative task를 associative setting으로 확장하는 가장 간단한 방법을 discuss해보자.

예를 들어보자.
서로 다른 k-armed bandit task가 여러 개 있다 쳐.
매 스텝 랜덤하게 선택해서 얘들 중 하나를 만나는거지.
그래서 bandit task가 매 스텝 랜덤하게 바뀌는거야.
single, nonstationary k-armed bandit task인, true action values가 매 스텝 random하게 바뀌는, 그런 task 로 보일 거야.
앞에서 봤던 method들로 nonstationarity 를 상대할 순 있겠지. 하지만 true action values 가 천천히 바뀌는 경우가 아니면 이 method들은 별로야.

하우에버, 가정을 해 보자.
bandit task가 selected 됐을 때, 넌 이 task의 identity에 대한 distinctive clue를 받은 거야.
action value에 대한 clue는 아니고. identity.
니가 action values가 바뀔 때마다 display color가 바뀌는 slot machine을 보고 있다고 생각해도 될 듯.
넌 이제 policy를 배울 수 있어. associating each task 하는 policy.
니가 보는 color에 signaled 된. 해당 task에서 best action이 있고.
예를 들면, red면 arm1, green이면 arm2. 이런 식.
policy가 잘 맞으면 bandit들을 구분하는 정보가 없어도 훨씬 결과가 좋겠지?

이게 associative search task의 한 예야. 왜냐면 best action을 찾기 위한 trial-and-error learning, 특정 situation에서 action들의 association이 관련돼 있기 때문이지.
associative search task는 contextual bandits 라고 불림.

associative search tasks는 k-armed bandit problem과 full reinforcement learning problem의 중간계야. 호빗이지.
learning a policy 측면에선 full reinforcement 이고,
each action이 immediate reward에만 영향을 준다는 건 k-armed 느낌.

action이 reward뿐만 아니라 next situation에 영향을 주게 돼 있다면, full reinforcement learning problem임.
이 문제를 다음 챕터에서 다뤄보자. 그리고 책의 나머지에 미치는 영향도.

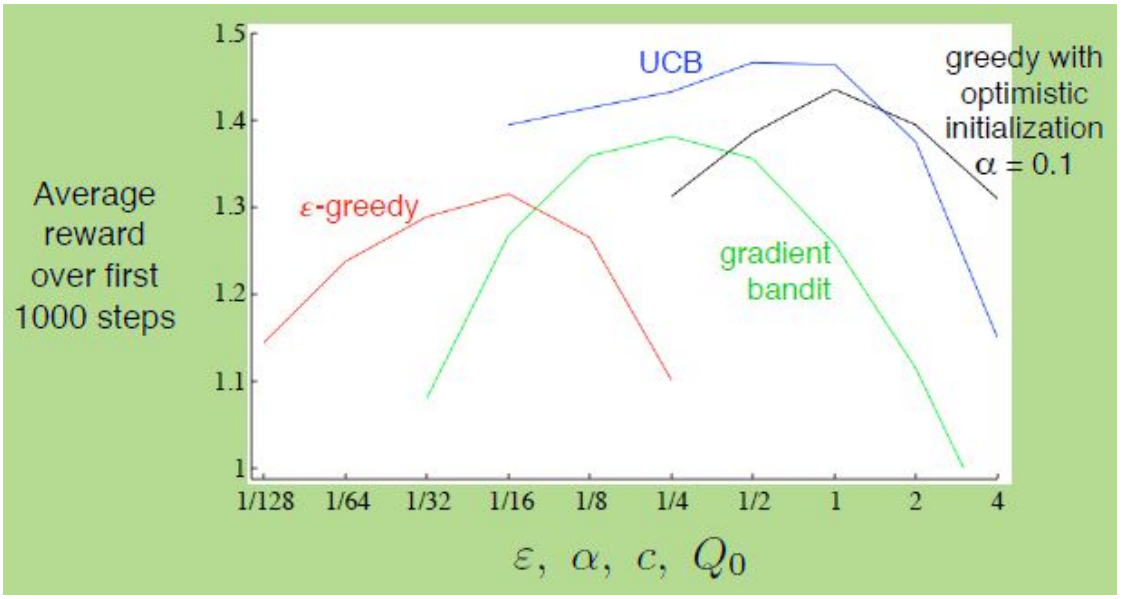
2.10 summary

오호.. 드디어 2챕터 써머리인가. 뒤에 한참 남았지만 한 챕터 끝나니 기분이 좋구만. 마저 달려보도록 하자.

이 챕터에선 몇 가지 simple ways를 봤어? 뭐였냐고? exploration과 exploitation을 balancing하는 ways.
ε-greedy methods는 랜덤하게 골라. 뭘? a small fraction of the time을.
UCB는 deterministically 골라. but exploration을 하지. 어떻게? 매 step subtly favoring the actions 해서. 여태 fewer samples 를 받은 action.
Gradient bandit algorithms은 action value를 estimate하는 게 아냐. action preferences를 estimate하지. 그리고 더 선호되는 action을 favor하지. soft-max distribution을 이용하는 graded, probabilistic manner로.
initializing estimates를 달리하는 편법은 optimistically 하게 greedy methods 조차 explore를 많이 하게 만들어.

이 중 어떤 게 best냐. 대답하기 어려운데. 사실 다 돌려보는거야. 그러고는 performance를 비교하는거지. 복잡한 건, 얘들이 다 parameter를 갖고 있다는 거야.
의미있는 비교를 하려면 얘들의 parameter로 만든 함수로 performance를 측정해야해.

우리 그래프 몇 개 봤잖아? 각 algorithm 과 setting 별로, 시간에 따른 learning course가 그려진. learning curve를 그리려면 고게 필요해.
모든 algorithms 와 모든 parameter setting에서 learning curve를 그리면, graph 는 너무 복잡하겠지? 막 겹쳐있을 거 아냐.
1000 step 동안의 average value로 complete learning curve를 요약하는 것 대신,
learning curve 아래 영역에 비례해. 뭐가? learning curve 다 그렸을 때 말하는 듯.



(fig2.6) parameter study. 각 포인트들은 average reward obtained over 1000 steps.

(fig2.6)은 이 measure를 보여주고 있는겨. 엥. average로 하겠다는 말이었어? 좀 더 보자.
그림은 여러 algorithms의 각자 parameter의 함수인겨. parameter를 x축으로 놓고 그렸슈.
이런 graph를 parameter study라고 불러.
parameter value들은 two factors에 따라 다르고 log scale로 나와있슈. 무슨 factor지?

전부 inverted-U 모양인 게 보이니? 모든 algorithms이 각자의 intermediate value에서 best네.

method를 평가할 땐, best parameter setting에서만 보는 게 아니라 parameter value에 얼마나 sensitive한지도 봐야혀.
이 algorithm들은 충분히 insensitive해. parameter가 크기 순으로 나와있을 때 잘 구분되게 performance가 분포된다는 뜻인듯.
전반적으로 UCB가 젤 좋네?

이 챕터에서 소개한 것들이 간단하긴 한데 아주 좋아. the state of the art 야 아주.
sophisticated methods들도 있는데 걔네 complexity와 assumption은 아주 impractical해. full reinforcement learning problem에. 5챕터가면 이번에 배운거 쓰니까 잘 기억해두도록.

지금까지 본 simple methods들은 지금까지는 best지만, balancing exploration and exploitation 하기에 완전 만족스러운 건 아냐.

한 가지 잘 연구된 approach가 있어. Gittins indices라는 함수를 계산하는겨.
특정 bandit problem 에 optimal solution을 제공하지.
근데 이 approach는 prior distribution of possible problems를 알아야 한다는 가정이 있어.
안타깝게도, 이론적으로나 계산적으로나 일반화하긴 힘들어.

Bayesian methods는 known initial distribution over action values 를 가정해. 그리고 매 step 이후 이 distribution을 update하지. true action values는 stationary라고 가정해.

일반적으로 update 계산이 아주 복잡하지만 특정 distribution(conjugate priors라고 부른대)에서는 쉽다네.
한 가지 가능성은, 매 step마다 best action의 posterior probability에 따라 select actions 하는거야. 이 방법은 posterior sampling 또는 Thompson sampling이라고 불리는데, 이 챕터에서 봤던 distribution-free methods의 best만큼 성능을 자주 보여줘.

Bayesian setting에선 optimal balance를 계산도 conceivable.
가능한 모든 action에 대해 각각의 immediate reward probability, action value에 대한 resultant posterior distribution를 구할 수 있대.
이 진화하는 분포는 information state of the problem이 된대.

Given a horizon, 1000 steps라고 치자. 우린
all possible
actions,
resulting rewards,
next actions,
next rewards를 다 고려할 수 있는거지. 1000 steps 전부.
주어진 가정에서, rewards 와 probabilities of each possible chain of events는 determined 될 수 있어. 그리고 그 중 best만 고르면 돼.
하지만 tree of possibilities는 엄청 빠르게 커져. 2개의 actions과 2개의 rewards만 있어도. 2^2000 개 leaves 가 있겠지.
이건 방대한 계산을 정확하게 하긴 어렵지. 하지만 approximated efficiently 될 순 있어.
이런 approach 는 bandit problem을 효율적으로 full reinforcement learning problem 의 instance로 바꿀 수 있을 거야.
마지막엔 approximate reinforcement learning methods들을 쓸 수 있을 거야. 파트 2에 나와 있는 optimal solution에 approach하는 방법 같은 것들이지.
근데 이건 사실 research를 위한 주제고. introductory book의 scope에는 넘어가지. 이 지긋지긋한 introduction의 굴레.

(exercise 2.9) (programming)
(fig2.6)같은 걸 만들어보렴? (exercise2.5)에 나와있는 non-stationary case로.
constant-step-size ε-greedy algorithm with α=0.1 이야.
200,000 step으로 돌리고, performance measure로는 average reward over the last 100,000 steps로 해 보게. 허헣.