3.4 Unified Notation for Episodic and Continuing Tasks

이전까지 두 가지 강화학습 tasks 를 봤지. 기억나는감?

1. agent-environment interaction이 개별 episodes의 sequence로 쪼개지는 episodic tasks.
2. 그렇지 않은 continuing tasks.

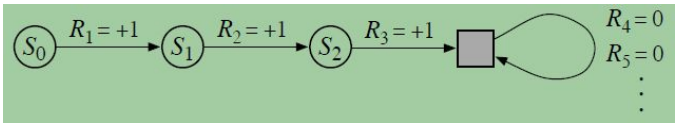
1번이 수학적으로는 쉽지. 각 action이 episode 동안 이후에 받을 finite number의 rewards에만 영향을 미치니까.
이 책에서는 문제에 따라 한 가지씩 또는 둘 다 할거야.
그러니 두 case를 동시에 정확히 묘사하는 notation을 만들어놓으면 좋겠지.

episodic tasks를 정확히 표현하려면 추가적인 notation이 필요.
time steps의 하나의 긴 sequence가 아니라,
episodes의 series가 필요. episodes는 각각 time steps의 finite sequence로 구성됨.
episode마다 time steps을 0부터 다시 세야지.
그래서 S_t가 아니라
$S_{t,i}$ 로 표시해야해. episode i 의 time t 에서의 state.
나머지도 다 마찬가지.

$$A_{t,i},\ R_{t,i},\ \pi_{t,i},\ T_i,\ \text{etc.}$$

그러나, episodic tasks를 볼 때 episodes 끼리 구별할 필요가 거의 없더라고.
우린 거의 항상 특정 single episode를 보거나,
모든 episodes에 참인 어떤 것을 얘기할거야.
따라서, 실제로는, episode number 떼고 막 써.
그러니 S_t 가 곧 $S_{t,i}$ 라고 생각해. 나머지도 마찬가지.

episodic과 continuing tasks를 커버하는 single notation이 하나 또 있어.
우리 위에서 return을 finite하게, 또 infinite 하게 정의했던 거 기억나니.
그거 합칠 수 있어. episode termination 을 할 거야.
absorbing state라는 데 들어가면 자기 자신으로 돌아가면서 rewards를 0만 생성하는 거지.
예를 들어 state transition diagram을 그리면:



요런 느낌.
색칠된 사각형이 special absorbing state야. episode의 end지.
S_0부터 시작해서 reward sequence +1, +1, +1, 0,0,0,... 을 받는겨.
infinite sequence 합친 거나 first T rewards 합친 거나 같겠지? 여기선 T=3이고.
discounting이 있어도 이건 true.
그래서 return을 일반적으로 정의할 수 있어.

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

이렇게 infinite하게 했던 것처럼. episode number는 뺀다고 합의.

합계가 정의된 상태에서 γ = 1일 가능성도 포함. 왜냐면 모든 episodes가 종료되니까

$$G_t \doteq \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$$

이렇게 쓸 수도 있지.

$$T = \infty \text{ or } \gamma = 1 \text{ (but not both)}$$

이런 가능성도 포함.

앞으로 책에서 이 합의들을 쓸겨. notation도 간단히 하고 episodic과 continuing tasks 사이의 close parallels도 표현할 수 있어서 말이지.

3.5 Policies and Value Functions

거의 모든 강화학습 알고리즘은 value functions 을 estimating 하는 걸 갖고 있지.
그 functions은 state 혹은 state-action pairs 의 함수이고,
뭘 estimate하냐면, given state에서 그게 agent에게 얼마나 좋은지.
또는 given state에서의 given action에서 perform하기에 얼마나 좋은지 estimate.
"얼마나 좋은지"는 expected future rewards로 정의. 그러니까 expected return로 정의.
물론 agent가 미래에 받길 기대하는 rewards는 취할 actions에 달려있지.
따라서, value functions은 특정 ways of acting에 대해 정의돼.
그걸 policies라고 불러.

공식적으로, policy는 states에서, 가능한 각 action을 선택할 probabilities로 가는 mapping이야.
agent가 time t에서 policy π를 따른다면, π(a|s) 는 S_t = s 일 때 A_t = a 일 확률.
p 처럼, π도 걍 평범한 함수.
π(a|s) 안의 "|" 이것도 각 s 에서 a 의 probability distribution을 나타내는 거라는군.
강화학습 methods는 agent의 policy가 경험에 따라 어떻게 바뀌는지 specify 하는거다.

$v_\pi(s)$: policy π 일 때 state s 의 value. s에서 시작해 π를 따른 후의 expected return.

MDPs 에서, v_π 를 공식으로 정의하면,

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right], \text{ for all } s \in \mathcal{S}$$

이리 된다.
E_π[.] 는 agent가 policy π를 따를 때 random variable의 expected value. t는 아무 time step.
terminal state가 있다면, 값은 0.
v_π를 policy π 일 때 state-value function이라 한다.

비슷하게, policy π일 때 state s에서 action a 할 때의 value를 정의할 수 있다.
그것이 $q_\pi(s,a)$.
s에서 시작해 action a 를 취한 후 policy π에 따른 expected return.

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t \mid S_t=s, A_t=a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1} \,\middle|\, S_t=s, A_t=a\right]$$

q_π를 action-value function for policy π라 한다.

value function v_π, q_π 는 경험에 의해 estimate 됨.
예를 들어, agent 가 policy π를 따르고, 발생한 각 state마다 그 state를 따르는 actual returns의 평균을 유지한다면, 그 평균은 state의 value인 v_π 에 converge 한다. state가 발생하는 횟수가 무한대로 갈수록.
각 state에서 취해진 각 action마다 개별 평균이 유지된다면, 이 평균들은 비슷하게 action values $q_\pi(s,a)$로 converge한다.
이런 종류의 estimation methods를 Monte Carlo methods라고 한다.
왜냐면 actual returns의 many random samples 를 평균하는 게 involve 돼 있으니까.
ch5에서 할 거.
물론, state가 엄청 많이 있으면, 각 state마다 개별 average를 갖는 게 실용적이지 않겠지.
대신, agent는 v_π, q_π를 parameterized functions로 유지해야할거야. (parameter 수가 states 보단 작게) 그리고 parameter를 조정해. observed returns에 더 잘 맞게.
이것도 정확한 estimates를 만들어. parameterized function approximator의 nature에 많이 depend하지만.
이런 가능성은 책 파트2에서 얘기하자.

강화학습과 dynamic programming 에서 쓰이는 value functions의 fundamental property는, recursive 관계를 만족하는 거야.

$$\begin{aligned} G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma\big(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots\big) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

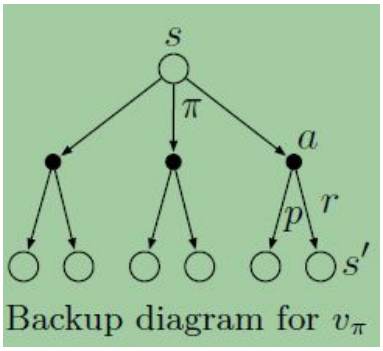
이거 했던 것처럼.
모든 policy π, state s 에 대해, s의 value와 possible successor states의 value 사이에 다음 consistency condition 이 만족함.

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t=s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t=s] && \text{(by (3.9))} \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)\Big[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1}=s']\Big] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_\pi(s')\Big], \quad \text{for all } s \in \mathcal{S}, && (3.14) \end{aligned}$$

a는 A(s) set에서, s'은 S set에서(episodic 이면 S^+), r은 R set에서 나옴.
마지막 equation에서 어떻게 두 sum을 합쳤는지 보자. r과 s'에 대해 합쳤쥬?
이런 거 자주 쓸 거야.
a, s', r 세 variables에 대해 모든 값들을 다 sum 한 거야.
각 triple마다 확률 π(a|s)p(s',r | s,a)를 계산해. 그리고 괄호 안의 양을 계산해서 weight 주고, 모든 가능성을 합해서 expected value를 얻지.

(3.14) 식을 Bellman equation for v_π 라고 불러.
state의 value와 이어지는 states의 values의 관계를 표현하지.

한 state에서 가능한 이어지는 states를 생각해봐.



Backup diagram for v_π

이 그림처럼.
빈 동그라미는 state, 까만 동그라미는 state-action pair를 나타내.
꼭대기 root node인 state s에서 시작해 agent는 any set of actions 할 수 있어. 그림에선 3개지. policy π를 기반으로.
이 각각으로부터, environment 는 여러 next states s' 중 하나로 연결돼. (그림에선 두 개 표현) reward r과 같이. function p에 의해 주어진 dynamics에 depend 해서.

Bellman equation은 모든 가능성에 발생확률을 각각 곱해서 average 하지.
start state의 value
= (discounted) value of the expected next state
= reward expected
이런 뜻을 내포하고 있다는군.

value function v_π는 Bellman equation의 unique solution이야.
Bellman equation이 어떻게 v_π 를 compute, approximate, learn 하는 수많은 방법들의 basis가 되는지 뒤에서 보여줄게.

위에서 본 diagrams을 backup diagrams이라고 부르는 이유는,
reinforcement learning methods의 심장이라 할 수 있는 backup operations이나 update 의 basis를 이루는 관계를 diagram 하기 때문이지.
이런 operatoins이 value information을 다시 state (또는 state-action pair) 에 전달해.
successor states(또는 state-action pair)로부터 말이야.
(transition graphs와는 다르게, backup diagrams 에서 state nodes는 distinct states를 나타낼 필요는 없어. 예를 들어 state는 자기 자신의 successor가 될 수 있지.)

(Example3.5) Gridworld

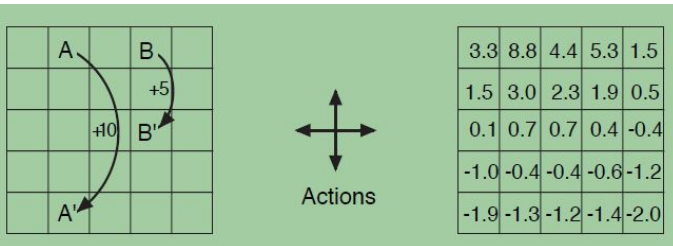


Figure 3.2: Gridworld example: exceptional reward dynamics (left) and state-value function for the equiprobable random policy (right).

왼쪽 그림은 간단한 finite MDP를 나타낸 사각 gridworld이다.

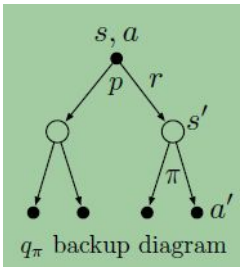
grid의 cells은 environment의 states에 해당한다.
각 cell에서 동서남북 네 가지 actions이 가능하고, deterministically 하게 agent가 각 방향으로 움직인다.(action이 동쪽이면 agent가 동쪽 cell로 간다는 말)
agent가 밖으로 나갈 것 같으면 그 자리에 그대로 있으면서 reward만 -1.
다른 actions은 reward 0인데, A랑 B states는 다름.
A에서는 모든 actions이 reward +10 이면서 agent를 A'로 옮김.
B에서는 모든 actions이 reward +5 면서 agent를 B'로 옮김.

agent가 모든 states에서 같은 확률로 4 actions를 선택한다고 해보자. 각 0.25겠지 그럼?
위 그림 (fig3.2)에서 오른쪽 그림은, 이 policy에서, γ = 0.9일 때, value function v_π를 나타내.
이 value function은 (3.14) 식을 계산한 거야.
아래쪽의 negative 값들이 보이니. 얘들은 random policy 상에서 grid의 외곽을 칠 확률이 높아서 그래.
state A 는 이 policy에서 있기에 best state야. 하지만 expected return은 10보다 작지. 10이 immediate reward인데 말이야. 왜냐면 A에서 A'으로 가면 grid의 edge로 갈 확률이 높아져서 그래.
state B는, 반대로, value가 5보다 높네? 왜냐면 positive value를 가진 B'으로 가기 때문.
B'에서 edge로 가서 받을 expected penalty는 A나 B로 가서 받을 expected gain으로 보상되고도 남는거지.

(Exercise 3.12)
Bellman equation(3.14)은 (fig3.2)에서처럼 must hold for each state for the value function.
numerically 하게 보여보렴. equation이 가운데 state의 +0.7이 성립하는지.
동서남북 이웃 states에 대해서 말이야.
(소수점은 아래 한 자리까지만 정확)

(임시답안)
아주 흥미로운 예제들이 자꾸 나온다잉? 구래요잉~? ㅇ0ㅇ

(Exercise 3.13)
action values q_π에 대한 Bellman equation은 뭘까.
q_π(s,a)를 줘야해. (s,a) pair의 possible successors 의 action values q_π(s', a') 관점에서.
Hint: 요 backup diagram이 equation에 맞아.



(3.14)랑 유사하게 sequence of equations을 보여주렴. action values 용으로 말이야.

(임시답안)
더욱 흥미롭군.
근데 왜 안하고있니.

(Exercise 3.14)

(임시답안)

(Exercise 3.15)

(임시답안)

Example 3.6: Golf
golf hole에 공 넣는 걸 강화학습으로 한다고 하자.
hole 안에 공이 들어가기 전까지 매 stroke 마다 -1.
state는 공의 location.
value of a state는 해당 위치에서 hole 까지의 stroke 수의 음수이다. 뭔말?
actions은 어떻게 aim, swing하는지와 어떤 club을 쓰는지다.
aim, swing은 주어진 걸로 하고, club만 putter 나 driver 중에 고르는 걸로 하자.

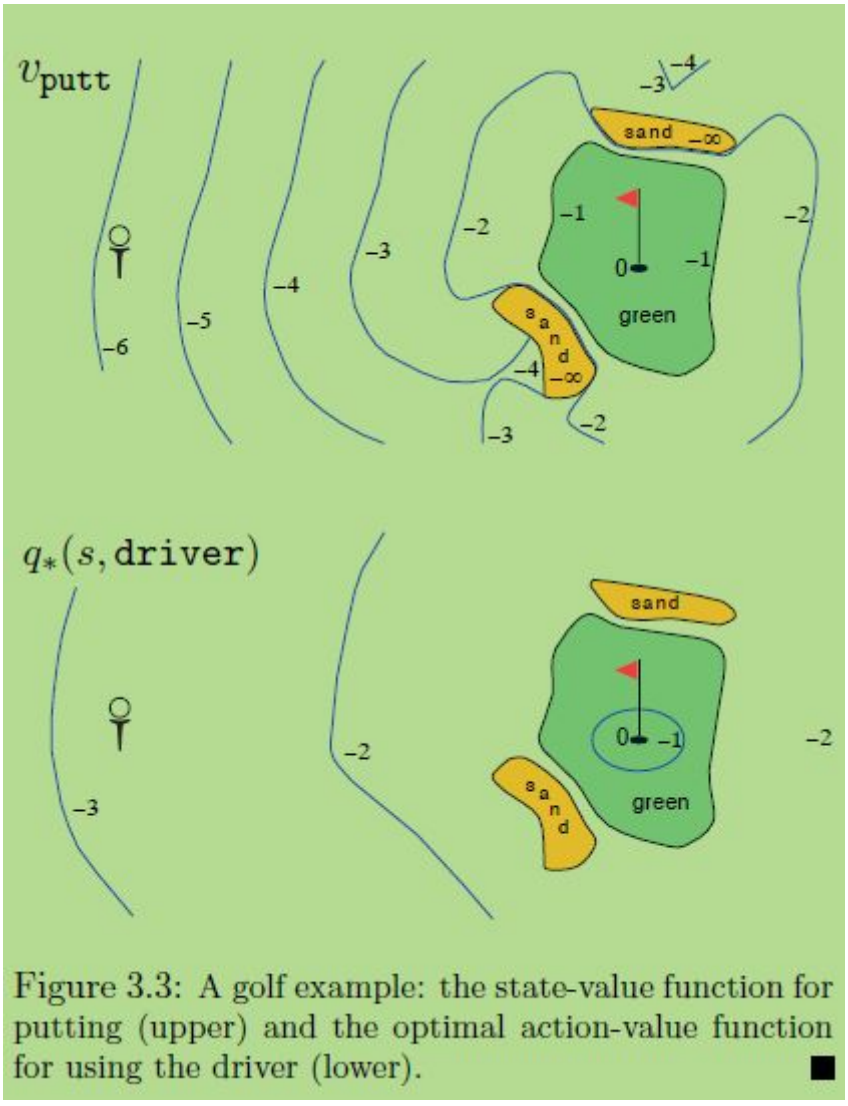


Figure 3.3: A golf example: the state-value function for putting (upper) and the optimal action-value function for using the driver (lower). ■

Fig 3.3의 윗 그림은 항상 putter 를 쓰는 policy의 possible state-value function $v_{\text{putt}}(s)$ 이다.
terminal state인 in-the-hole은 value가 0.

green에서는 항상 putt를 성공할 수 있다고 가정해서, 이 state에선 value가 -1.
green 밖에서는 putting 으로 hole에 못 가고 value가 커. 작은 게 아니고?
만약 어떤 state에서 putting으로 green에 도달하면 해당 state는 green보다 value가 하나 작을거야. 그래서 -2.
putt를 정확하고 deterministically하게 한다고 가정하자. 실력이 그정도는 되잖아?
대신 limited range로.
이렇게 가정하면 -2가 label 된 sharp contour를 얻을 수 있어. 그 라인 안에서 치면 hole까지 무조건 2 strokes.
마찬가지로 나머지 -3, -4, … contour도 얻을 수 있겠지.
putting은 sand traps 밖으로 나올 수 있게 못하니 value는 -∞.
그래서, tee 에서 hole로 putting을 이용해 가는 데는 6 strokes가 필요하다. 이 말이야.

(Example 3.16)

(임시답안)

(Example 3.17)

(임시답안)

3.6 Optimal Policies and Optimal Value Functions

강화학습 문제를 푼다는 건 roughly하게 말해서, policy를 찾는 거야. long run에서 많은 reward를 받는 policy.
finite MDPs에선, optimal policy를 다음과 같이 정확히 정의할 수 있다.
value functions은 partial ordering over policies를 정의한다. 이게 뭔 말이여.

policy π는 policy π' 보다 좋거나 같다고 말한다. 언제?
모든 states에서 expected return 이 크거나 같으면.

$$\pi \geq \pi' \text{ if and only if } v_\pi(s) \geq v_{\pi'}(s) \text{ for all } s \in \mathcal{S}.$$

이렇단 말이쥬.

그 중 best인 policy가 있겠지?
모든 다른 policies 보다 좋거나 같은 policy가 최소한 하나 있단 말이여.
이거슬 optimal policy라 한드아.

하나 이상일지라도 potimal policy를 π_*라고 할 것이여.
걔들은 같은 state-value function을 공유해.
그걸 optimal state-value function 이라고 하지.
v_*라고 할 거여.

$v_*(s) \doteq \max_\pi v_\pi(s)$ 모든 state s 에 대해.

optimal policies 는 마찬가지 optimal action-value function도 share하지.
q_*라고 할 거시다.

$q_*(s,a) \doteq \max_\pi q_\pi(s,a)$ 모든 s, a에 대해.
state-action pair (s,a)에 대해, 이 함수는 state s 에서 action a를 취하고 그 후 optimal policy를 따르며 expected return을 준다.
q_*를 v_*의 관점에서 쓰면

$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a]$$

이리 되지.

Example 3.7: Optimal Value Functions for Golf
바로 예시로 설명해준다.
아까 위에서 봤던 Golf 예시에서 아래 그림 있었지?
그게 possible optimal action-value function q_*(s, driver)의 contours야.
각 state의 values를 보여주고 있지. 처음에 driver로 치고 다음부턴 driver나 putter 중에 좋은 애를 골라 칠 때의 value야.
driver는 공을 멀리, 하지만 조금 부정확하게 보내준다.
충분히 가까우면 한 방에 hole에 보낼 수도 있겠지. 그래서 -1짜리 q_*(s, driver) contour는 green의 조그만 부분만 차지하고 있어.
우리에게 2 strokes 가 있다면 더 멀리서 hole에 접근할 수 있겠지? 그래서 -2 contour가 그려짐.
이번 경우엔 저 작은 -1 contour 안에 들어가려고 drive 하지 않아도 돼. green에만 올려도 되지. 그 다음은 putter로 치면 되니까.
optimal action-value function은 특정 첫 번째 action(여기선 driver)을 하고 value를 준다.
이후엔 어떤 actions이든 best인 걸 하고.
-3 contour는 더 멀고, starting tee까지 포함.
tee에서 actions 의 best sequence는 2 drives + 1 putt. 3 strokes 만에 넣는 거지.
예시가 별 도움이 안 되는 거 같기도 하고?∎

v_*가 value function for a policy라서, self-consistency condition을 만족해야한다.
self-consistency condition은 Bellman equation for state values에 의해 주어짐.

$v_\pi = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\left[r + \gamma v_\pi(s')\right], \quad \text{for all } s \in \mathcal{S}.$

Bellman equation for state values가 이랬는데 기억 나시는지.

v_*는 좀 달라. optimal value function이거든.
consistency condition에 특정 policy에 대한 reference가 없어야해.
이걸 Bellman Equation for v_* 라고 해. Bellman optimality equation 이라고도 하고.
직관적으로, Bellman optimality equation은 optimal policy 상에서 value of a state가 해당 state에서 best action을 할 때의 expected return과 같아야 한다는 걸 나타낸다.

$$
\begin{aligned}
v_*(s) &= \max_{a \in A(s)} q_{\pi_*}(s,a) \\
&= \max_a \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] && \text{(by (3.9))} \\
&= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] && (3.18) \\
&= \max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma v_*(s')\big]. && (3.19)
\end{aligned}
$$

마지막 두 equations은 Bellman optimality equation for v_* 의 두 가지 폼이다.

Bellman optimality equation for q_*는

$$
\begin{aligned}
q_*(s,a) &= \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \;\Big|\; S_t = s, A_t = a\Big] \\
&= \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma \max_{a'} q_*(s', a')\Big]. && (3.20)
\end{aligned}
$$

바로 이것.

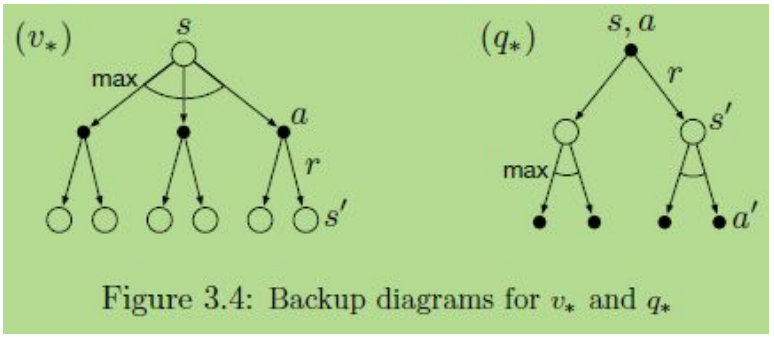
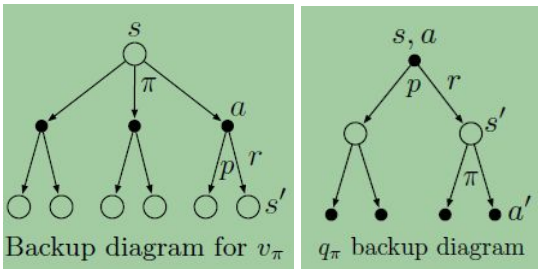


Figure 3.4: Backup diagrams for v_* and q_*

fig 3.4 의 backup diagrams은 future states와 actions의 확장을 보여준다.
Bellman optimality equations for v_*, q_* 가 적용됐을 때.
이전에 봤던



요거랑 비슷하지?
다른 점이 max 로 나타낸 arc. agent의 choice points.
해당 choice 상에서 maximum을 취한다. policy에 따른 expected value로 하는 게 아니라.