copy\_weight부분을 이용해 디버깅을 해보자. variable이 변화하는지, trainnet에서 targetnet으로 복사가 잘 되는지를 출력시켜서 확인해 본 결과 놀랍게도, weight이 변하고 있으며 복사도 착실히 진행됨을 알 수 있었다. 그럼 뭐가 문제일까? 학습 방향 자체가 문제인가?

<https://github.com/hunkim/ReinforcementZeroToAll> 강의의 소스코드를 보려했는데, 여기에 있는 소스는 또 다른 소스인 것 같다.

강의 코드와 약간의 차이점이 있다. 강의 코드는 episode 10번마다 한번 training을 시켜주는데, 10사이즈 minibatch를 50번 반복해 교육시키고 있었다. 반면, 내가 tf2용으로 가져온 코드는 episode 도중 매 step마다 minibatch(32)를 뽑은 뒤 학습을 시키고 있다. 이걸 한 50번 반복시키면 되려나???

일단 다시 origin 소스코드에 hidden layer만 10\*10으로 줄여 학습시켜보고있다. loss가 600에서 줄기 시작했지만 100 이하로 떨어지지를 않는다. average reward는 랜덤보다 못한 수준이다. 15000회 결과 e값이 상당히 줄었음에도 loss는 10~20사이를 와리가리 한다. 잠깐만, 혹시? 최대진행횟수 200제한에 걸려서? 끝나면 done이 되고, 계속 -200점을 먹어서 그런건가?

1. 최대 횟수 늘려보기

생각해보니 rewards는 done을 만날 시 -200과 관계가 없더라… 학습에 걸리는 시간만 늘어난 것 같다. 느낌 상? 근데 그러면 reward가 늘어야 정상인데 왜 reward는 그대로지?

1. reward -200을 -100으로 줄여보기?

loss가 100 이하로 안 떨어지던 것이 50대가 되었다. 하지만 reward가 여전히 끔찍하다.

1. epsilon 강의처럼 바꿔서 해보기

여전히 답 없음.

1. hidden layer [10]으로 줄여보기
2. minibatch 반복학습 시켜 보기 = 또는 Lr늘려보기?

속도를 빠르게 하기 위해 log 빈도 줄여보기. 빨라진 느낌이 살짝 든다.

움직이는 모습 보려고 200번만 학습시키고 render해봤다. 그랬는데… 안넘어진다. 이게 무슨일이야. 10001스텝 찍을때까지 안넘어진다. 대체 뭐가 일어난거지?

근데 또 두번 더 반복하니까 형편없는 결과를 보여준다. 엄청 운 좋게 초기값이 좋았던건가? 뭐지?

최대 step을 3000정도로 줄이고, log빈도를 다시 늘린 뒤 이번에는 N=400정도로 해봐야겠다. epsilon decay도 .9->.99로 늘리고.

너무 불안정하다. reward와 loss가 널뛰기한다.

epsilon decay .9999로 다시 늘려봤다. 여전히 널뛰기 한다. reward는 e가 늘었으니 당연하지만, 테스트 결과도 너무 안좋았다. loss는 그나마 착실히 줄어갔다.

decay를 다시 0.9로 줄여봤다. reward는 바닥을 치고, loss는 500대에서 줄지를 않는다.

1,2)decay를 0.999로 만들고, layer모양을 10 하나로 만들어보자. 여전히 안좋다.

3)layer모양을 100 하나로 만들어보자. 의미 없는 학습곡선이다.

4)layer를 100인채로 둔 채, bias를 없애보자. 엉망진창의 결과가 나왔다.

5)bias를 없앤 채로 initializer를 glorot uniform으로 바꿔봤다. 여전히 엉망진창이다.

6)layer를 10으로 다시 줄여볼까? >bias 하나 없앴다고 이렇게 널뛰기하는 이유가?

7)lr을 늘려보자, 강의처럼. 0.01->0.1그대로 엉망.

8)카피 타이밍도 아예 줄여버리고. 25->10. 성능 좀 나오나? 싶었는데 loss는 여전히 엉망.

9)강의에서는 gamma가 0.9더라고. 0.99->0.9 …..

10)epsilon을 다시 강의처럼 만들었다…. 아무런 변화가 없다.

11) batch를 늘려보자. 32 -> 100

12) bias를 다시 돌려놔보자…loss는 안정화 되었다. reward도 봐줄만 해 졌다. reward가 널뛰기 하하는 것 좀 어떻게 하면…

13) 12에서 lr을 줄이거나, e를 다시 복구시키거나. 먼저 lr부터.0.01

처음엔 정신좀 못차리다가 스무스하게 loss가 줄고, 근데 reward는 여전히… 좀 안정적으로 된 것 같긴 한데.

14) e를 복구하고 decay 0.993으로. loss가 정말 완벽하게 학습된다! reward도 상당히 좋은 것 같다! episode 개수를 좀 더 늘려보면 되지 않을까???

15)통 크게 1000으로 늘려준 episode 개수. 그러나… reward가 떡상하는 일은 없었다…

회색 14번, 주황 15번. 올라가지를 못한다….

왜일까, 뭐가 문제일까? overfitting? overshooting? overestimating???

계속 만져 주는 중 relu로 바꿔보기도 하고 decay값도 만져주고… loss는 어느정도 내려가지만 reward는 항상 널뛰는데, 바가 한 방향으로 쭉 가서 벽에 부딪혀 사라지는 현상이 계속 발생한다.

19) batch 100->10 loss가 감소하다 솟구치기 시작. reward도 증가하다가 급감

reward가 증가한 상태를 유지하지 못하는 이유는 무엇인가?

20) relu->tanh. loss가 다시 이상해졌다

학습에 영향주는 요소들

초기화방법 Glorot uniform

activation function tanh

gamma값 0.9

learning rate 1e-2

decay

(min epsilon)

hidden layer

batch size

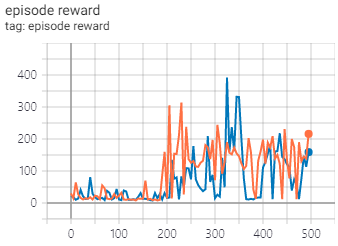
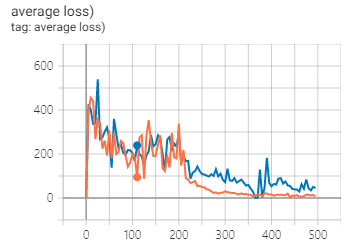
copy step

done penalty

21) batch 10->30, lr 0.002 ->0.01

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 초기화 | AF | gamma | lr | de/min | hid ly | batch | cp step | done | ep n |
| Glorot | tanh | 0.9 | **1e-2** | .993/e-2 | 10 | **30** | 10 | -100 | 500 |

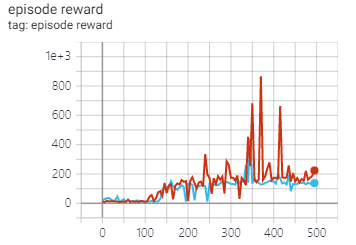
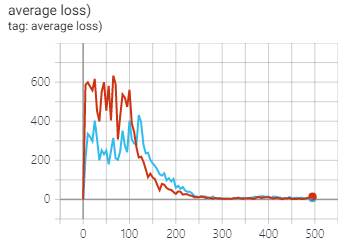
벽 박치기 현상



22) batch 30 -> 32, copy step -> 25

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 초기화 | AF | gamma | lr | de/min | hid ly | batch | cp step | done | ep n |
| Glorot | tanh | 0.9 | 1e-2 | .993/e-2 | 10 | **32** | **25** | -100 | 500 |

벽 박치기 현상 여전히. 특이한 현상으로, 학습도 안 된 모델이 시작부터 3001step을 찍는 현상이 있었다. 학습이 진행되면서 사라졌다.



23) epsilon을 좀 더 급격히 줄여보자.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 초기화 | AF | gamma | lr | de/min | hid ly | batch | cp step | done | ep n |
| Glorot | tanh | 0.9 | 1e-2 | .**99/e-2** | 10 | 32 | 25 | -100 | 500 |

눈에 띄는 변화는 없다.

24) lr 5e-2 5배

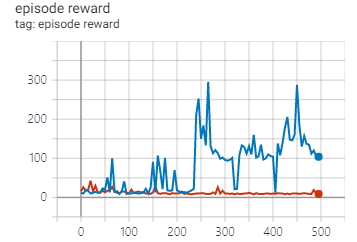
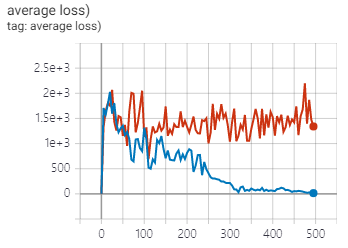
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 초기화 | AF | gamma | lr | de/min | hid ly | batch | cp step | done | ep n |
| Glorot | tanh | 0.9 | **5e-2** | .993/e-2 | 10 | 32 | 25 | -100 | 500 |

loss 줄어드는 속도는 늘었는데, 의미없는 것 같다. 롤백해야겠다.

25) lr 롤백, 벽에 박는 것을 응징하기 위해 done -300?

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 초기화 | AF | gamma | lr | de/min | hid ly | batch | cp step | done | ep n |
| Glorot | tanh | 0.9 | **1e-2** | .993/e-2 | 10 | 32 | 25 | **-300** | 500 |

loss 감소되는 경우도 있고 안되는 경우도 있는 신기한 케이스

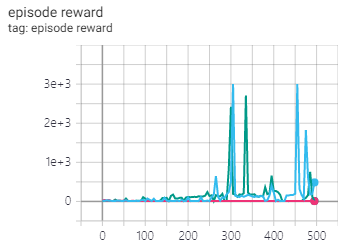
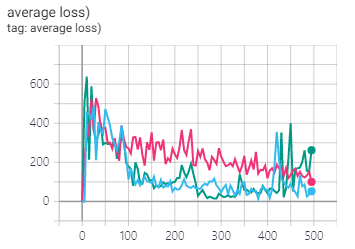


26) penalty를 다시 -100으로 바꾸고, gamma를 늘린다면?

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 초기화 | AF | gamma | lr | de/min | hid ly | batch | cp step | done | ep n |
| Glorot | tanh | **0.99** | 1e-2 | .993/e-2 | 10 | 32 | 25 | **-100** | 500 |

가장 좋은 성능! gamma값이 미래구나. 3000을 찍기도 한다. 아예 학습이 안되는 경우도 있고. 대신 3000을 계속 유지하지는 못하는 것 같다. e때문에 그런가?

3번 반복했는데 두번은 실패. 그중 한번은 아예 학습도 안됨.

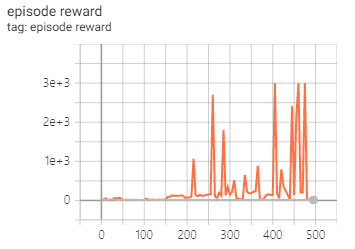


loss 튀는 것을 잡아주면 될까? 아니면 e값을 급격히 감소시키는게 좋을까? gamma값을 999로 만드는 것은 어떨까?

27) copy step을 10으로 감소. loss를 잡기 위해?

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 초기화 | AF | gamma | lr | de/min | hid ly | batch | cp step | done | ep n |
| Glorot | tanh | 0.99 | 1e-2 | .993/e-2 | 10 | 32 | **10** | -100 | 500 |

여전히 아예 학습도 안되는 경우가 있고, 3000reward를 찍는 횟수가 가장 많은 경우도 있다. 유지력이 계속 문제이다. 그래서 생각해보건대, replay memory를 늘리면 뭔가 될 것 같기도 하다.



28) replay memory 1만->5만

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 초기화 | AF | gamma | lr | de/min | hid ly | batch | cp step | done | ep n |
| Glorot | tanh | 0.99 | 1e-2 | .993/e-2 | 10 | 32 | 10 | -100 | 500 |
| memory |  | | | | | | | | |
| **50000** |

별다른 의미는 없었다.

29) gamma 0.99->0.999

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 초기화 | AF | gamma | lr | de/min | hid ly | batch | cp step | done | ep n |
| Glorot | tanh | **0.999** | 1e-2 | .993/e-2 | 10 | 32 | 10 | -100 | 500 |
| memory |  | | | | | | | | |
| 50000 |

어쩔 땐 loss가 미쳐 날뛰고, reward는 더 안 좋아진다

30) 끝부분에 도달했을 때, penalty를 안 줘야 학습이 제대로 될까? play\_game method에서 자체적으로 iter가 일정 이상(2001)이 되면 break한 후 멈추게 하고, reward는 그대로 1이 되게 했다.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 초기화 | AF | gamma | lr | de/min | hid ly | batch | cp step | done | ep n |
| Glorot | tanh | **0.99** | 1e-2 | **.993/**e-2 | 10 | 32 | 10 | **1/-200** | 1000 |
| memory |  | | | | | | | | |
| 10000 |

그래도 여전히 2001step을 유지하지는 못하는 모습이다.

골고루, 정확한 학습이 되지 않아서일까?

31) 격변시켜봤다.

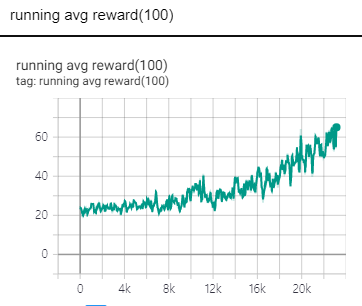
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 초기화 | AF | gamma | lr | de/min | hid ly | batch | cp step | done | ep n |
| Glorot | **relu** | 0.99 | **1e-3** | .9999/e-1 | 100\*50 | 32 | 25 | 1/-200 | 5000 |
| memory |  | | | | | | | | |
| 10000 |

layer개수를 늘렸더니 loss가 학습이 잘 안되더라. 그래서 relu로 바꿨는데, 이번엔 14k를 넘어 발산하더라. lr을 1/10했지만 잘…. 안되는 듯…. 성능 자체는 가장 높게 나왔다. 테스트할 때 1000~1500step 사이를 왔다 갔다 하는 정도?

DQN은 이정도로 하는 게 좋겠다. 시시포스의 돌굴리기도 아니고, 나아지는 방향이 보이면 다시 망하고, 또 나아가면 망하고. 더 이상 만지는 것 자체가 무의미한 것 같다. 마지막으로 original 코드나 한번 돌려보면서 마무리 하자.

31)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 초기화 | AF | gamma | lr | de/min | hid ly | batch | cp step | done | ep n |
| random | tanh | 0.99 | 1e-2 | .9999/e-1 | 200\*200 | 32 | 25 | -200 | 50000 |
| memory |  | | | | | | | | |
| 10000 |

점차적으로 reward 평균이 올라가긴 한다.

걱정되는 점 : DQN으로 Cartpole하는 것도 이렇게 오래 걸리는데, DDQN으로 TORC는 어떻게 돌리냐… 후…

DDPG로 넘어가기 – PG에 대한 자료를 정리중이었는데, 다 날아갔다. 왜 저장 안했지…

<https://reinforcement-learning-kr.github.io/2018/06/29/0_pg-travel-guide/> 참고할 article

<http://karpathy.github.io/2016/05/31/rl> pg개념을 잡을 때 읽으면 좋다고 추천된 글

Q-learning과는 다른 방식의 deep learning 학습 개념 PG. Q-learning은 deterministic한 면이 있고,(오로지 Q value가 높은 방향으로만 정책 결정. e-greedy같은 것을 적용하지 않는다는 가정 하에) Q value가 살짝 변해도 action은 급격하게 변하는 등의 문제점이 있다. PG는 action 선택의 확률에 관한 policy를 기반으로 하기에 이러한 문제들에 상대적으로 자유롭다.

Sutton PG -> DPG -> DDPG순서.

의 정확한 뜻은 무엇인가? x를 넣었을 때 y를 증가시키기 위한 W벡터 원소 각각의 변화량들의 벡터라고 한다 (설명에 따르면). 헷갈릴 필요 없다, 우리는 이것을 이미 loss function을 최소화하기 위한 backpropagation에서 배웠다. 차이점은, 여기는 y=up확률을 늘리기 위해 그래디언트를 더해준다면, loss function은 오히려 loss를 줄이기 위해 그래디언트를 빼주는 것이라는 차이가 있다. loss로 -plog(q)를 사용했던 것을 기억하는가?

위의 경우는 label이 정해진 지도학습의 경우인데, 글은 강화학습의 경우도 이와 비슷한 프로세스를 가진다고 설명한다. 그렇다면 label에 해당하는 값은 어떻게 얻어야 되는가? sampling을 함으로서 얻는다. forward pass를 하면 up, down의 확률이 계산되며 이 확률에 따라 sampling을 하면 (즉 확률이 적용된 동전던지기를 하면) label이 결정된다. (그리고 동시에 정책 역시 결정된다.) down확률이 up확률보다 높다면 아마도 down이 나올 확률이 높을 것이다. 실제로 down이 나왔다고 가정하자. 그러면 back propagation을 통해 구한 는 down값이 커지는 방향을 가리킬 것이다. 그리고 게임의 결과(보상)을 봤는데 어라, 패배해서 -1을 먹었다. 그러면 어떻게 해야 하는가? 간단히 보상과 그래디언트를 곱한 값을 더해주면 된다. 그러면 down값이 커지는 방향의 반대방향, 즉 down 확률이 작아지는 방향으로 정책망이 학습될 것이다.

up이 나왔다면? 똑같이 up확률이 커지는 방향의 gradient를 구해준 뒤, 결과 보상을 곱해준 값을 더해준다.

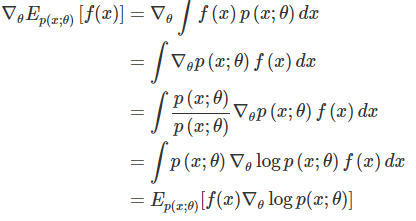
거기에 좀 더 디테일을 더해주자면, 승/패 가 결정되는 시점을 기준으로 가까운 시기의 결정에는 더 큰 보상/패널티를, 먼 시점의 결정에는 더 작은 보상/패널티를 주는 식으로 할 수도 있다.

설명글과 함께 pg논문을 탐구해보자.

Expected return을 구하는 방법엔 크게 두가지가 있다.

“deterministic approximation으로 Markov decision process의 dynamics를 모델링한 후 수식을 통해 구하는 것입니다” = Q learning이 이에 속한다고 보면 될까? 뉘앙스가 그런 것 같다.

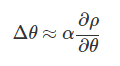
“monte carlo estimation으로 dynamics에 대한 모델을 하지 않고 많은 sample들을 얻은 후 empirical(경험주의적)하게 expected return을 계산하는 방법입니다.” = 우리가 하고자 하는 방식은 이것이다. “dynamics에 대한 모델이 어렵거나 변화가 큰 경우” 이 방식이 더 효과적이나 gradient를 estimate하는 방식이 필요하고 그것이 monte carlo estimation이다. 이것은 무엇인가?



이처럼 log derivate trick을 이용해 식을 변형을 가능하고 최종적으로



이렇게 sample의 평균을 취하는 monte carlo 기법으로 gradient를 근사화 하는 것이 monte carlo gradient estimation이다. 그러나 이 방법은 얻은 샘플에 의존하기 때문에 variance가 클 수 있다….고 한다? 잉? 이게 PG 아니었어?



ρ : 해당 policy들의 성능을 나타내는 척도 (예. average reward per step)

α: positive-definite한 step size

“논문의 가장 중요한 contribution은 특정 조건을 만족하는 function approximator를 이용하여 경험(experience, sample)을 축적하고 이것들을 이용하여 위의 gradient를 unbiased estimate할 수 있음을 증명한 것입니다.”

이러려면 일단 ρ를 표현하는 방법 두가지를 알아야 한다.

2.3 Average Reward Formation

시간의 흐름에 따른 reward를 표현하기 보다 모든 시간의 reward를 평균내서 표현하는 방법.

* Long‐term expected reward per step

등식이 성립하는 조건 : time average 와 ensemble average가 같다는 뜻, ergodic한 시스템에서 성립한다고 한다.

: stationary distribution of states under π

stationary distribution이 뭐지? marcov chain(부루마불!)에서 특정 조건을 만족하면, 어떤 지점에서시작하더라도 무한히 step을 반복하면 각 상태들의 방문횟수 비율이 특정 확률분포로 수렴하게 되게 되는데 이를 stationary distribution이라고 한다.

따라서 모든 policy에 대해서 는 와 independent하다고 한다.

내 자의적으로 종합해 해석해보자면, 정책 에 의한 평균 reward는 곧 특정 state에서 a를 취할 확률에 reward를 곱한 값들을 모두 더한 뒤 ( ) 여기에 특정 state에 도달할 확률을 곱한 값을 state마다 전부 곱해서 더해준다는 의미인 것 같다.

* Value of a state-action pair given a policy

왜 이런식으로 평균reward를 빼주는가? 내 추측이지만, <http://karpathy.github.io/2016/05/31/rl> 아까 봤던 이 글에 설명이 있는 것 같다. reward가 모두 양수의 형태라면 어떻게 되겠는가? 그래디언트는 항상 한쪽 방향으로 더해지기만 하고(실패건, 성공이건!) 결국 \theta값이 발산해버릴 것이기 때문에 state-action value function을 이렇게 지정해줌으로서 일종의 regularization을 해주는 것이 아닐까?

2.4Start-State Formulation

아까는 시작지점에 관계없이 state에 대한 value function이 구해졌지만, 시간이 지남에 따라 reward가 감소되는 것을 표현하기 위해서는 달리 해야한다.

* long-term expected reward per step with a designated start

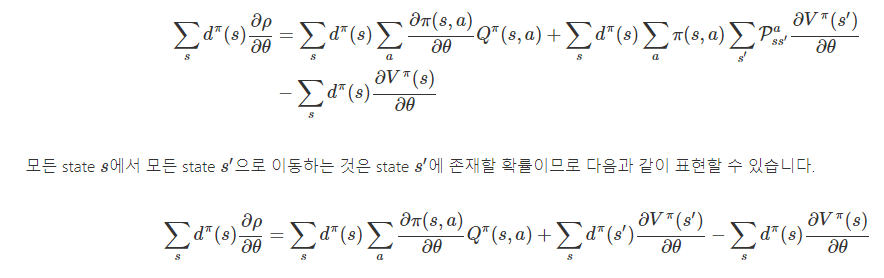
…..

**Theorem 1 (Policy Gradient)** For any MDP, in either the average‐reward or start‐state formulations,

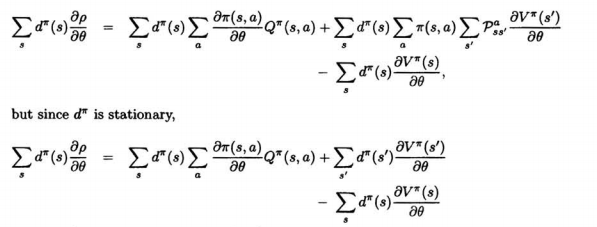
가장 놀라운 점은 espected return의 gradient를 구할 때 를 구할 필요하 없다는 것.

증명은… 일단 너무 복잡하니 넘기자…

average-reward formulation에서의 증명이 다른건 그러려니 하고 넘어가겠는데



이부분이 이해가 안 된다. 논문의 원문은 이렇다.



곰곰히 생각해 보니 이해되는 것 같기도 하다. 빨간 부분으로 표시한 부분을 말로 해석해보면 결국 각 state V(s)의 미분값 평균이라는 의미인데, 간단하게 아래 식처럼 바꿀 수도 있는 거니까. s 가 될 확률 \* s에서 a를 선택할 확률 \* s에서 a를 선택했을 때 s’이 될 확률. 이들을 모두 더하면 결국 s’이 될 확률 이 된다.