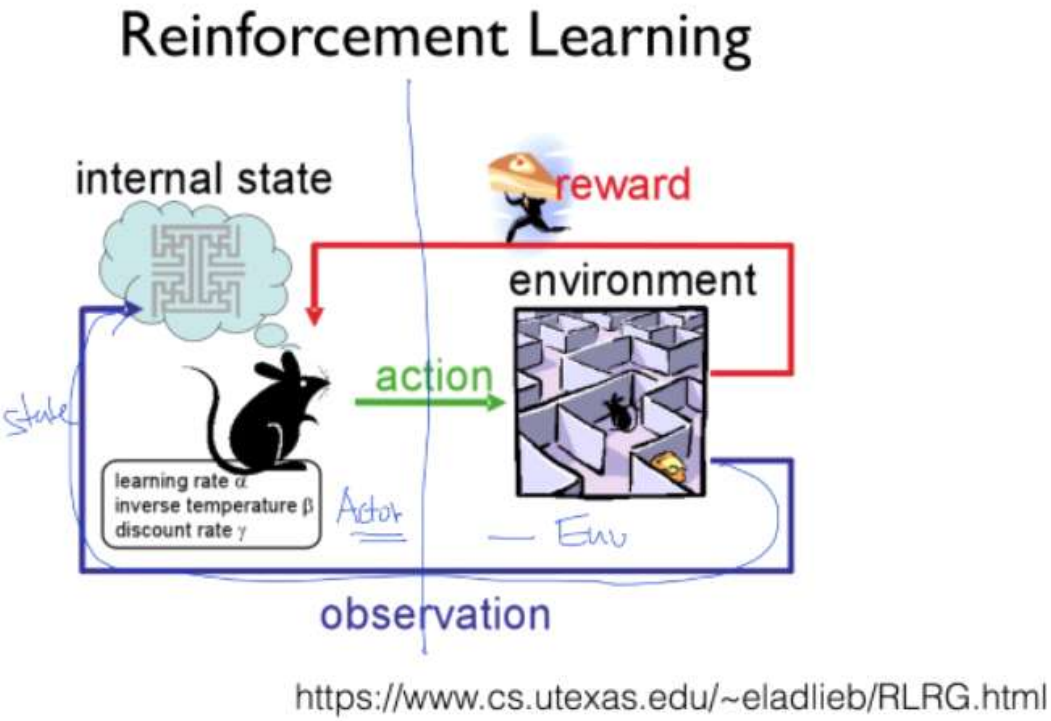
환경 : openAI에서 배포한 GYM 패키지



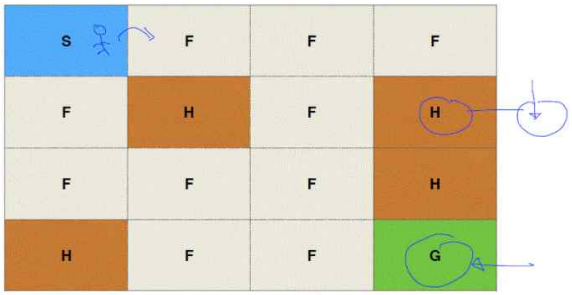
복습)
actor와 env가 나뉘어 있다 환경에서 행동하는 actor가 있습니다
액터가 env속에서 action(행동)을 한다 행동을 할때마다 환경에서 상태가 변경됩니다
그 변경된 상태 state가 update가 된다. 매 순간 순간 마다 보상이 있다.

오래전에 제시된 알고리즘이다. 97년 tom의 machine learning책에 소개되었습니다.
15년도 딥마인드의 벽돌깨기로 화려하게 부활하였습니다.

Q = r + Q 식이 기본이다
Exploit 현재 있는 값을 이용한다
exploration 안가본 곳(값)을 이용한다

Q-learning은 결국 Q는 주어진 state에서 어떤 action을 취할 것인가에 대한 길잡이인 Q를 학습하는 것입니다. 이 문제에서 Q라는 2차원 배열에는 현재 state에서 action을 취할 때 얻을 수 있는 reward를 저장하고 있습니다. 이 Q 2차원 배열에서 argmax 함수를 이용하면 어떤 action을 취할지를 얻어낼 수 있는 것입니다. 이 문제에서 state는 16, action은 4입니다. (4x4 Frozen Lake 그리고 action은 위, 아래, 왼쪽, 오른쪽 4개)
 * Frozen Lake 문제는 얼어있는 호수에서 Robot을 조종해서 호수 중간중간에 있는 구멍을 피해서 목표지점까지 가는 문제입니다. 4x4크기의 얼어있는 호수에 Robot이 있습니다. 구멍에 빠지면 문제의 목표달성에 실패하게 됩니다. Robot에는 4방향(상,하,좌,우)으로 움직이는 명령을 내릴 수 있는데 호수가 얼어있어서 Robot이 가고자 하는 방향과 다른 방향으로 미끄

러져서 이동할 수도 있습니다. 그래도 Robot이 미끄러지더라도 가고자 하는 방향의 반대방향으로 이동하지는 않습니다. 즉, 왼쪽으로 가려고 하는데 오른쪽으로 이동하는 경우는 없습니다.



s:시작지점 , G: 목표지점, H: 구멍, F: 얼어있는 호수

Q-learning 알고리즘에서 Q를 업데이트하는 것
Q(state, action) = R + max(Q(new state)) 로 업데이트하게 됩니다. R은 reward로 게임 내부에서 지정되는 값입니다. 현재 state에서의 어떤 action을 취할 때의 Q 값은 그 action을 통해 얻어지는 reward와 action으로 변화된 state에서 얻을 수 있는 reward의 최댓값을 더한 것입니다. 즉 의미는 현재 리워드와 미래에 가능한 리워드의 최대치를 더하는 것입니다.

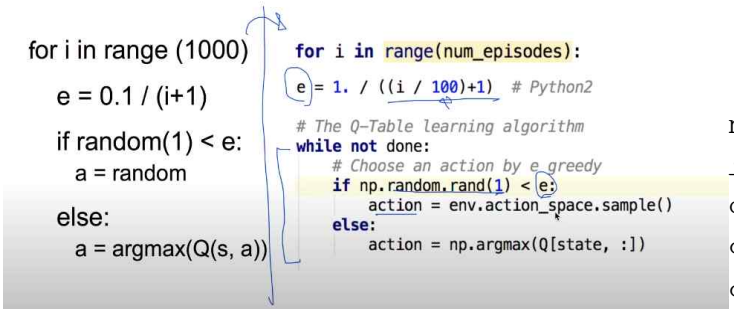
근데 이 때, Q(state, action) = R + discount * max(Q(new state)) 로 미래 가능한 리워드에 1 미만의 discount factor를 곱해주어(이 실습에서는 0.99) 미래 리워드에 패널티를 주어 Q가 조금 더 optimal한 방법으로 learning 될 수 있습니다. 이 때 discount는 하이퍼파라미터(hyperparameter)로 여러개를 선택하면서 좋은 값을 찾을 수 있습니다.
액션을 선택할 때 2가지 방식이 있다. 1)랜덤값을 더해 기존의 크기의 값의 선택을 랜덤하게 만드는 경우, 2)e-greedy를 써서 랜덤값과 e를 비교해 가면서 액션의 종류를 결정한다.

Q-learning 알고리즘에서 Action을 고르는 것
강화학습에서 action 을 고르는 것은 이슈 중 하나입니다. 액션을 고르는 방법 중에 첫 번째, random noise 방식, 두 번째, E-Greedy 방식입니다. 둘 다 exploit & exporation 방법을 구현한 것으로 볼 수 있는데 약간의 차이가 있습니다. 둘의 공통적인 아이디어는 '무조건 Q가 시키는대로만 가지말고 새로운 길로도 가보자는 것' 입니다. 왜냐하면 Q가 시키는 것이 최선이 아닐 수 있기 때문입니다.

E-greedy 방식
E-Greedy 방식은 어떠한 확률값 e를 주어, e의 확률로 exploration한다는 것입니다. 예를 들어 e=0.99 이면 99%의 확률로 exploration 하고, 1%의 확률로 exploit해서 새로운 길을 찾게 됩니다. 이 값을 (i/100) 으로 나눈 것은 위 1번과 마찬가지로 여러번 해보면서 Q 값이 optimal 해질 것이기 때문에 exploration을 줄이면서 exploit 위주로 한다는 뜻입니다.

decay- 학습하는 초기에는 많이 랜덤하게 하고 후반부에는 e값을 작게하자.
e = 1/((i/100)+1)로 I가 커질수록 e가 작아져 랜덤하게 하는 것이 작아진다.

Exploit VS Exploration: decaying E-greedy



random값보다 e가 크면 샘플값으로 액션 취한다
아니라면
액션을 q상태에서 가장 큰값으로 액션을 취하겠다.

random 값을 Q에 더해서 기존의 큰 값만 선택하는 것이 아닌 크기가 작은 값들도 선택할 수 있게 된다.

noise를 주는 방식은 액션을 선택할 때 아래 그림과 같이 랜덤한 값을 줘서 기존 Q값에 최선의 선택 대신 2번째, 3번째 값들도 선택 할 수 있게 하는 것이다.

Exploit VS Exploration: add random noise

```
a = argmax(Q(s, a) + random_values)

a = argmax([0.5  0.6  0.3  0.2  0.5]+[0.1  0.2  0.7  0.3  0.1])
```

decay를 여기서도 할 수 있다
for I in range(1000)
a = argmax(Q(s,a) + random_value)/i+1)

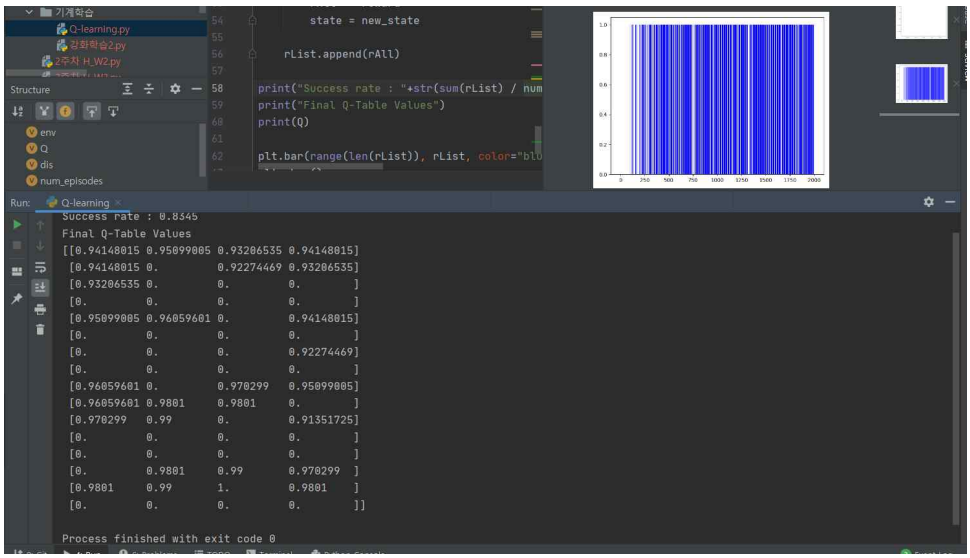
이렇게 하여 가장 좋은 값만 선택이 아니라 2번째 3번째 것도 선택할 수 있게 된다.

register 부분은 제시된 그대로 설정.

for I in range(num_episodes):
- 2000번 정도 돌면서 1. 초기화 2. 루프를 돌린다.
2-> 액션에 노이즈 추가
q 값에 노이즈를 뿌린다
그다음 아그맥스를 구한다

or
for I in range(num_episodes):
2000번 정도 돌면서 1. 초기화 2. 루프를 돌린다
2-> e값과 랜덤하게 생성한 값 1개를 비교하여
랜덤값이 크면 액션을 랜덤하게 취한다 -> action_space.sample() (환경에서 있는것중에 하나 가져오는 함수)
그렇지 않다면 Q상태에서 가장큰 것을 고른다(argmax())

이후에 액션을 실행시켜서 새로운 상태와 reward를 받는다
Q를 업데이트 = 현재 리워드 + 미래의 리워드*discount



Success rate는 Goal까지 실패 안하고 갈 확률입니다. 그래프를 보면 초반에만 실패하고 나중에는 다 성공하도록 Q 가 학습되어 나간다는 것을 볼 수 있습니다.