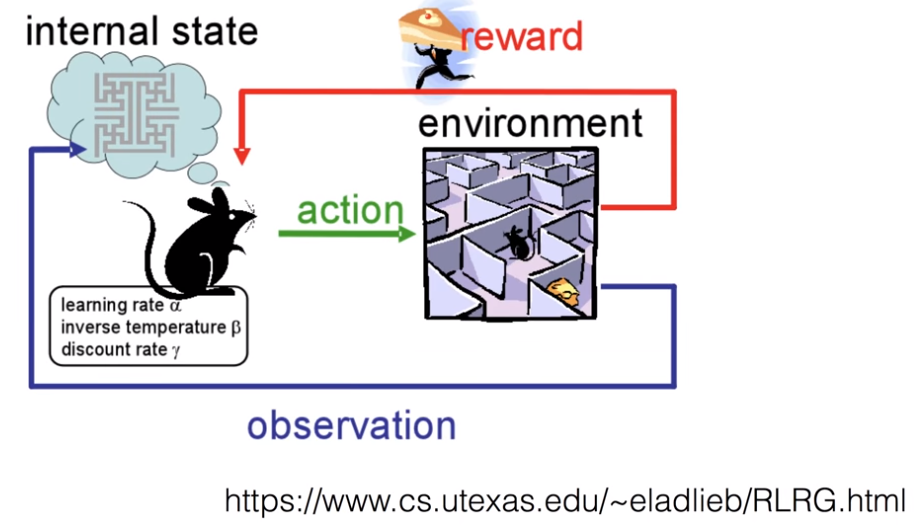
참고자료

홍콩과기대 김성훈 교수님 Reinforcement Learning with TensorFlow&OpenAI Gym강의

<https://www.youtube.com/watch?v=dZ4vw6v3LcA&feature=youtu.be>

강아지 훈련에도 쓰이는 용어. positive reinforcement training. 보상을 바탕으로 한 학습

칭찬과 꾸중을 이용한 자연스러운 학습, 현실에서도 메이저한 학습법



Actor가 Environment에서 Action을 취하면 State가 update되고 잘하면 Reward가 주어진다.

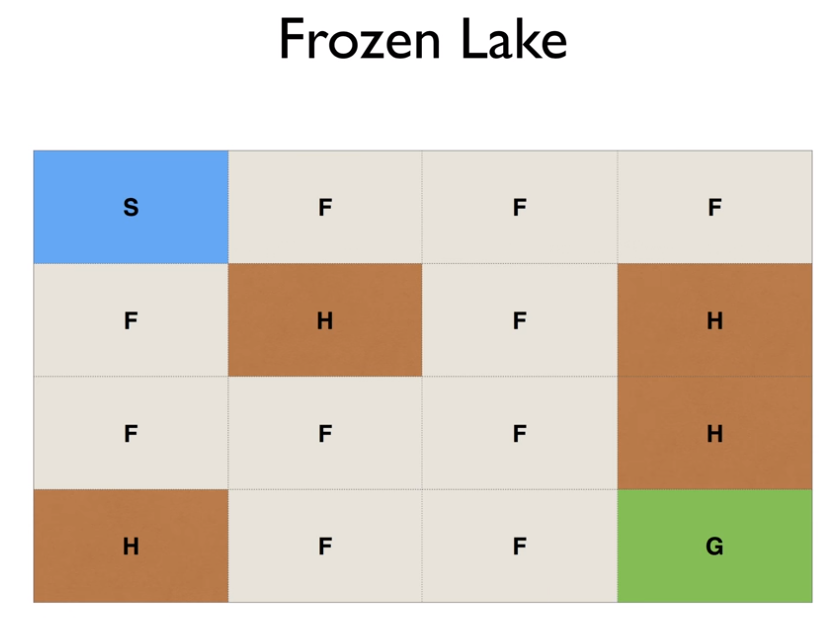
컴퓨터 RL은 1997년도 ML책에도 나올 정도로 오래된 개념. 재평가되어 주목받기 시작한 것은 Deepmind가 등장하면서부터. 한가지 게임이 아니라 모든 아타리 게임에 적용 가능한 학습법!

논문 명 “Human-level control through deep reinforcement learning”, Nature 26 Feb.2015

DeepMind는 게임 뿐만 아니라 구글데이터센터에도 적용되어 냉각에 드는 에너지를 40%절약.

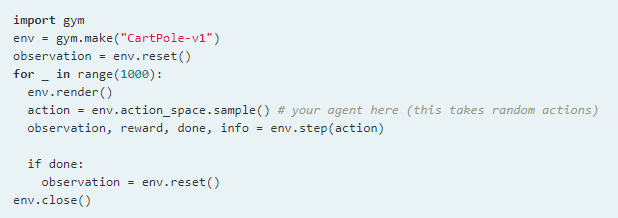
RL의 적용 분야 – 로봇 관절제어, 비즈니스 재고관리, 주식투자, Ecommerce/미디어 광고.

OpenAI GYM: environment를 구현해주는 API이다



예시 : Start, Frozen, Hole, Goal 등의 환경.

Agent는 Action(위,아래,좌,우)를 취하고 Environment는 state와 reward를 주게 된다. 목표는 Goal.



만약 끝났으면 리셋하고 닫는다

gym 불러오고

environment 만들고

첫번째 초기화 한 상태를 가져오고

1000번 반복하는데

render 하고, action을 한 뒤, step하면 상태, 리워드, 끝났는지 여부, 정보 등등등이 들어 오며

궁금한 점: 그래서 RL과 NN의 정확한 차이는 뭐지?

NN은 supervised learning, 즉 입력값과 그에 따른 결과가 이미 알려져있다. 이를 이용해 weight을 정하고, 나름의 loss function을 정하면 이를 바탕으로 loss를 최소화 하는 방향으로 NN은 학습 가능하다. 그러나 RL은 Unsupervised learning으로, loss function이라는 것이 존재하지 않는다. loss를 도출해낼 정답이 애초에 없기 때문이다.

알파고는 CNN을 사용했다고 했는데, RL도 사용했다고 한다. 구체적으로 어떻게 적용된걸까?

딥블루, 왓슨, 알파고(딥마인드)에 대한 글<https://fehani.tistory.com/8> 이것을 보면 알파고는 정책망과 가치망으로 나누어 학습을 하였다고 나와있다. 이때 cnn으로는 문제에 대한 표현 방법을 찾는다고 하는데, 아직은 어려운 것 같다.

<https://trendw.kr/2016-03112466.t1m> 알파고원리에 대한 또다른 글

여기에는 상당히 자세히 나와있다. 19\*19 바둑판을 그 자체로 cnn으로 구현하고, 다음 수를 어디에 놓아야 하는가 (정책망)을 3000만개의 기보를 통해 – 이 기보는 정답이 있다! – 학습시켜서, 돌을 어디에 놓아야 하는 가에 대한 학습을 시켰다. 거기에 MCTS의 Playout을 위해 Rollout policy또한 학습시켰다. 여기에 그치지 않고, “과거의 자신”과의 대결을 시켜서 이기면 현 사용 네트워크 weight을 강화하고 지면 반대방향으로 가게 해서 정책망을 더 강화시켰다. 그리고, MTCS에 필요한 v(s)값 – 현재 상태에서 플레이어의 승률 – 을 찾기 위해 value network 가치망 역시 만들었다. 이것 역시 cnn으로 만든 것인데,아까 자신과의 대결을 통해 얻은 기보를 저장해 놓고 이기는 상태와 지는 상태로 구분해 학습시켰다. 이후, 이들 결과들을 MTCS의 S – E – E - B과정에 적절히 적용하면서 V(s)를 구하는 것을 반복한다. 이러면 일정 시간동안 갱신하게 되면 최종적으로 선택할 수 있는 돌들의 위치 중 가장 많이 방문했던 (확률이 높은?) 위치를 선택하여 돌을 놓는다.

여기서 MCTS가 RL에 속하는 것인가? 그게 아니라 과거의 자신과 대결하는 것이 강화학습인 것 같다.

OpenGym으로 돌아와서, 실습을 시작해 보겠다.

문제점 : 방향키를 입력 받을 때 필요한 헤더인 termios가 unix전용이라는 것 같다. 윈도우 환경인 나는 어떻게 해야 하는가?

msvcrt를 임포트 하면 된다고 한다. msvcrt.getch()를 시도했다.

그런데 문제는 getch()는 1byte만 받는데, 방향키는 2byte라는 것이다.

강의 댓글에 있는 ord()를 사용한 예시를 내가 약간 변형해서 적용해보기로 했다.

잘 작동한다!

random보다 더 나은 알고리즘을 만드는 법은 무엇일까? 어떤 정책(policy)로 갈 것인가?

Q를 가정. state와 action을 넣으면 quality(reward)가 나오는 function = state-action function.

**Q(state, action)**

(MCTS에서 v(s)역할을 하는 것일까?)

상태에서 가장 큰 quality 값은 아래식처럼 표현한다

그리고 이때의 값은 아래식처럼 표현한다.

Q가 존재한다고 가정하자.

나는 s에 있다.

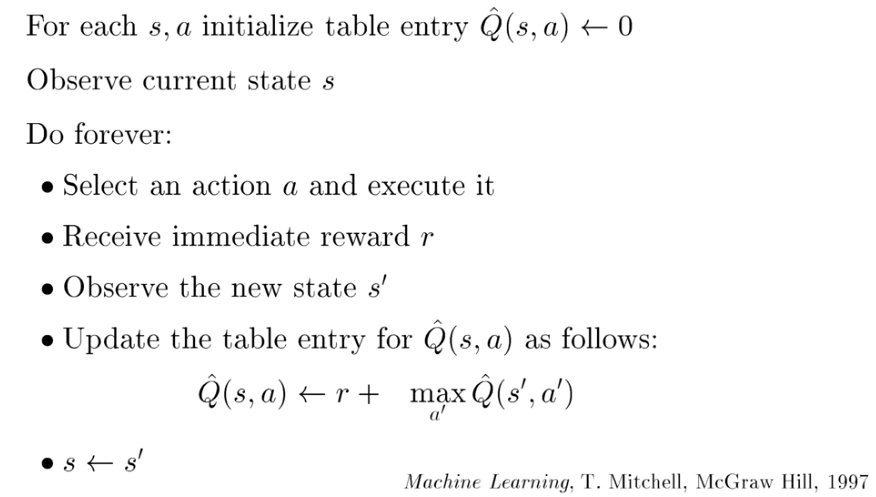
내가 a를 하면 s’으로 간다.

내가 a를 하면 r을 받는다.

Q(s’,a’)는 이미 알고있을 때, Q(s,a)를 Q(s’,a’)으로 어떻게 표현해야 할까?

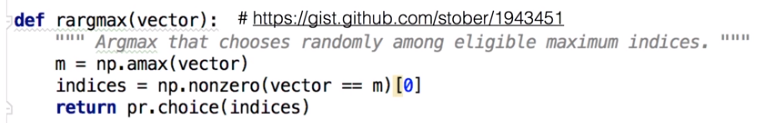
16 state에 4 action을 가진 frozen lake environment에서 Q를 학습한다고 생각해보자.

랜덤한 움직임 마다 계속 Q값을 갱신한다. 처음에는 0+0만 계속 될 것이다. 그러다가 운이 좋게도 goal에 도달해 reward를 받게 되면, 처음으로 Q(s’,a’)에 1이라는 값이 갱신되게 된다. 그리고 이것을 계속 반복한다… 그러다가 s’’에 다다라 a’’을 했더니 s’에 도달하였다면, Q(s’’,a’’)역시 갱신된다. 또 반복한다…결국에는 Goal에 도달할 수 있는 policy를 우리는 얻을 수 있다.



(왜 Q hat이라고 쓰는걸까? 더미Q라서 그런걸까?) -> 완벽한 Q가 아니라 수렴해가는 존재라는 의미이다. 수렴하는가? deterministic world(같은 action에 같은reward보장), finite state일 때 수렴한다는 것이 증명되어있다고 한다.

위 알고리즘을 python으로 구현하면 dummy Q-learning이 구현된다.



예시에서 argmax Q를 구현할 때 위의 메소드를 사용했는데, np.nonezero(verctor == m)[0]의 의미가 잘 이해가 되지 않았다. vector == m은 false, true값 하나만 뱉는 것이 아닌가? 그게 아니었다. true, false의 배열을 뱉는다. m이 numpy.int32이기 때문이다. 단순하게 numpy.nonezero(조건)으로 조건을 참으로 만드는 요소들의 index를 가져올 수 있다고 생각하면 편하겠다. (단, 조건식에 쓰이는 vector와 상수 둘 중 하나는 numpy class 객체이어야 한다.)

실험에 쓰인 파일 : lab3\_npvector.py

뒤에 [0]이 붙은 이유는 np.nonzero가 최소 2개값을 가진 튜플로 값을 뱉기 때문이다.

아직은 완벽한 Q learning이라고 볼 수 없다. 언제나 exploit할 뿐 exploration하지 않기 때문이다. **exploit and exploration**을 구현하는 방법은 여러가지가 있다

E-greedy : e값을 지정해 확률적으로 random한 루트를 찾게 만든다

Decaying E-greedy : 초반에는 높은 확률로, 후반에는 낮은 확률로 exploration을 시도한다

add random noise : 랜덤한 노이즈를 현재 state-action function값에 더해서 결정한다

이중 random noise를 추가하는 방식은 점수가 높을수록 action을 취할 확률도 높아져 자연스럽다.

이렇게 하면 다양한 루트가 개발되겠지만, 에이전트에게 더 빠른 길을 선호시킬 수는 없다. 그렇기에 Discounted future reward라는 개념이 등장한다.

이는 즉

lab3의 실습에 간단하게 덧붙이면 된다. 라고 생각하였으나…

action = rargmax(Q[state, :]+ np.random.randn(1, env.action\_space.n)/(i+1))

이렇게 하니 문제가 생긴다. action이 0에서 도통 움직이질 않는다.

np.random.randn(1, env.action\_space.n)/(i+1)) 이 결과값은 어째서인지 2차원 array를 뱉는다… 따라서 reshape해줘야 될 듯 하다.

지금 보니 2차원 array를 뱉는게 당연했다. 파라미터를 그러라고 넣어놨네.

noise를 reshape해서 더해줬다. 그런데 noise가 영 시원찮은가? 다른 길을 찾지를 않는다.

noise를 10배 늘려주는 작업을 했다. ~~그랬더니 이번엔 왼쪽으로 가는 Q 값이 증가하기 시작…~~

~~noise를 50배 정도 해줬더니 다른 값이 나왔지만, 대체 왜? 왜 왼쪽action Q값이 증가하는지 모르겠다. 동영상 속 교수님이 돌린 결과도 마찬가지더라.~~

벽으로 가면 움직이지 않고, state는 변화하지 않는다. 그러면 자기 자신의 Qmax에 dis를 곱한 값으로 더해지는 것이 자연스러운 현상이다.

Deterministic vs Stochastic(non-deterministic)

지금까지의 환경은 Deterministic이었다. 하지만 FrozenLake의 Slippery옵션을 켜게되면 Stochastic한 환경이 된다. 이러면 지금까지의 Q로는 성공확률이 희박해진다.

해결법이 있는가? 핵심은 Q의 말을 덜 듣는 것이다.

지금까지는 Q의 값을 갱신할 때 전적으로 그 다음 Q의 말을 맹신했다. 자신의 의견 – 기존에 자신이 가지던 Q값 – 은 전혀 고려대상이 아니다. 맹신하지 않으려면 어떻게 해야 하는가? future discount factor gamma와 더불어, learning rate alpha가 등장할 때이다.

action a를 취했더니 r을 얻고, 여기 state의 Q함수는 자신의 최대값이 어느 정도라고 알려온다. deterministic한 환경이라면 의심할 필요가 없지만, stochastic한 환경에서는 곧이곧대로 들어서는 안되는 정보이며, 어느정도 참고사항 정도로만 알아야 한다. 그 역할을 1보다 작은 가 한다.

이렇게 한다고 Qhat이 Q에 converge할까? 한다! 1992년 왓킨스와 대얀 교수가 증명했다고 한다. 실제로 코딩해보면 수렴하는가? 0.63정도의 성공률을 보인다. 예전버전의 noise가 너무 많아서 조금 줄였다.