**강화학습 프로젝트 보고서**

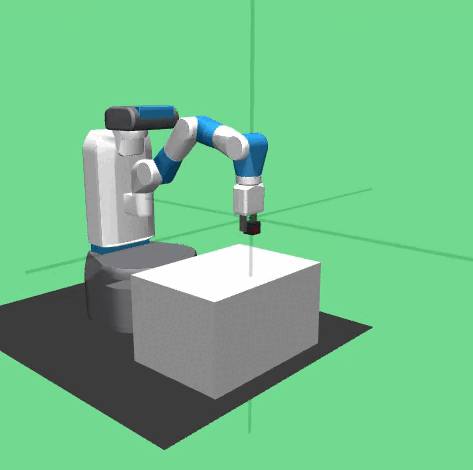
Team Ultron

12194114 구현규, 12191962 하대훈

**-프로젝트 개요**

pick and place 환경을 목표 환경으로 수정하고 목표환경에 맞게 알맞은 강화학습 알고리즘을 선택하여 임무를 수행하라.

* 기존 환경 및 목표



테이블위에 Box와 Target(Site)지점이 생성되며 로봇팔이 Box를 잡고 움직여 Target(Site)지점 위치시키는 것이다.

* 바뀐 환경 및 목표

그래픽, 만화 영화, 스크린샷, 도표이(가) 표시된 사진

자동 생성된 설명

|  |  |  |
| --- | --- | --- |
| **Object** | **Size** | **Position** |
| Table | 0.25 0.25 0.2 | 1.3 0.45 0.2 1.3 1.05 0.2 |
| Box (x10) | 0.02 0.02 0.02 | On the Right Table |
| Border | Height:0.05 왼쪽 테이블 전체 O | On the Left Table |

다음과 같이 table을 나누고 왼쪽 테이블에는 검은 Border 생성하기 그 안에 Target(Site) 생성, 오른쪽 Table에 검은 Box 10개를 생성한다. 목표는 위와 마찬가지로 Target(Site)에 Box를 위치하는 것이다.

* 환경 설정

pick\_and\_place.xml, fetch\_env.py 2가지 파일을 수정하여 환경을 바꾸었다.

먼저 pick\_and\_place.xml에서 위 환경에 맞게 각종 object 변경 및 수정을 한다. 이후 fetch\_env.py에서 바뀐 환경에 맞추어 Target(Site)와 Box 생성 방식을 변경하는데, 이때 핵심은 fetch\_env.py의 sample\_goal과 reset\_sim method에서 Target(Site)와 Box의 위치를 맞게 설정하였고 step method를 오버라이딩하여 총 10개의 box를 옮길 수 있도록 episode 종료 조건을 수정했다. (코드 부분 참조)

-**강화학습 알고리즘 선택**

* 고려사항

1. **행동 공간**: 로봇 팔의 연속적인 움직임 제어가 필요함.
2. **상태 공간**: 로봇 팔의 위치, 속도, 가속도, 그리퍼의 상태, 카메라로 인식한 물체의 위치 등 다양한 변수가 포함하여 효과족으로 다룰 수 있는 알고리즘.
3. **샘플 효율성**: 샘플 효율성이 높은 알고리즘은 적은 데이터로도 효과적으로 학습할 수 있음.
4. **보상 함수**: 적절한 보상 함수를 설계

위는 알고리즘 선택 간에 고려해볼 사항을 정리한 것으로 먼저 수행환경이 연속전이 움직임 제어가 필요한 환경임을 고려해야 한다. 특히 pick and place의 환경은 reward가 sparse reward이기 때문에 샘플 생성과 보상의 설정에 신중해야 한다.

* DDPG+HER

이번 학습에 선택한 모델이다.

1. DDPG

DDPG는 DQN에서 성공적이었던 부분들을 continuous 한 영역으로 확장한 것이다.

1. Off policy
2. DQN 기법 도입
3. OU(Ornstein-Uhlenbeck)\_process
4. Batch normalization

DDPG는 다음과 같은 특징을 가지고 있는데 1에 의해 target policy와 behavior policy가 같지 않아도 학습이 가능하다. 또한 2에 의해 리플레이 메모리와 target network를 사용하여 샘플 효용성 및 안정성을 향상시키고 3,4를 통해 continuous action에서 좀 더 효과적이 탐색과 효과적인 학습이 가능하다.

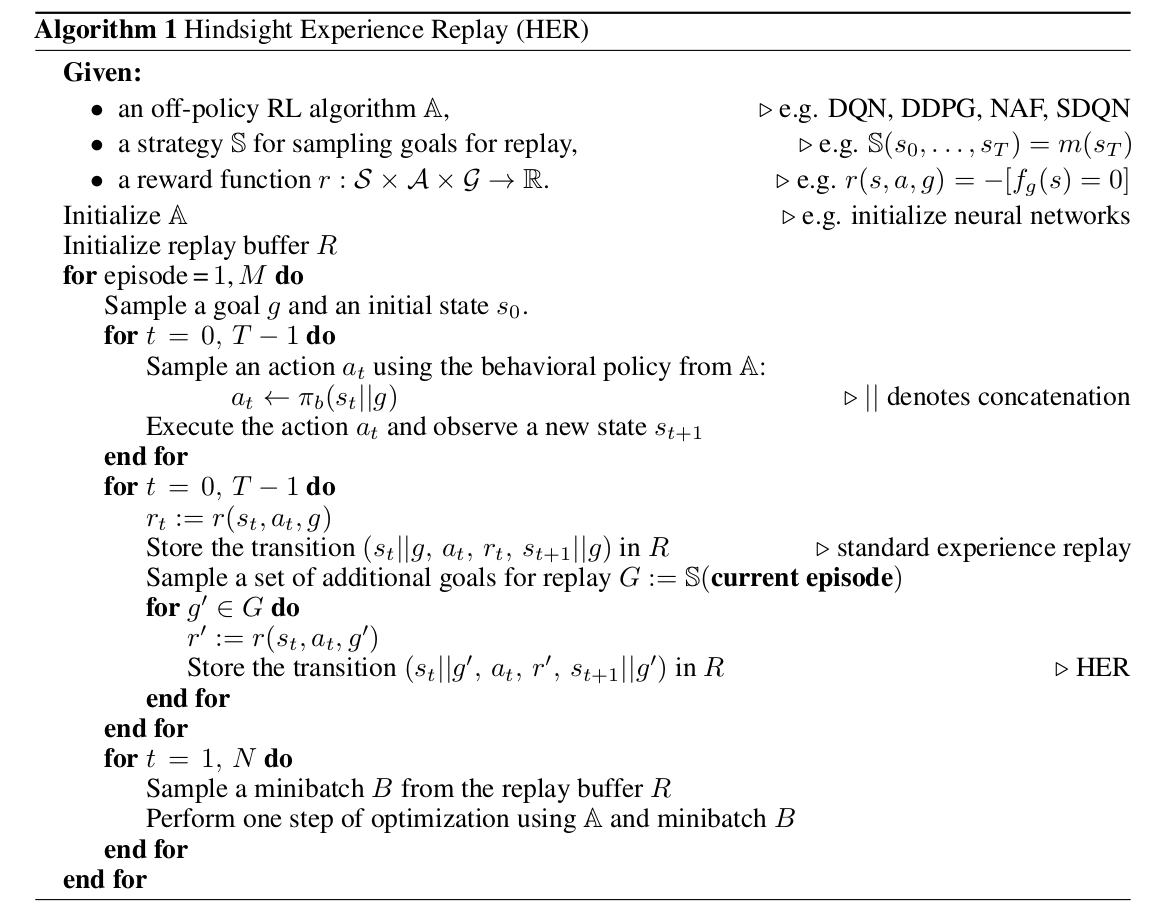
텍스트, 스크린샷, 폰트, 문서이(가) 표시된 사진

자동 생성된 설명

2. HER

sparse and binary reward 환경에서 효과적으로 학습하기 위한 알고리즘으로 사람이 축구공을 차서 골을 넣는 것에 피드백을 골을 넣는 것에만 학습을 하는 것이 아닌 넣지 못하는 상황에서도 학습을 한다는 것에 영감을 얻어 만들어진 알고리즘이다.

에이전트는 re-examine하는 trajectory의 모든 transition에 대해 원래의 goal 뿐만 아니라 다른 goal도 같이 replay buffer에 저장시켜 여러 개의 goal을 학습 즉 실패에서도 학습할 수 있게 된다.



DQN, DDPG 등의 off policy algorithm 을 선택하고 현재 episode내에서 HER 전략 S에 따라 additional goal을 샘플링한다. 현재 episode의 sample(s, a, r, s')에 대해 새로이 sampling된 goal g'을 통해 reward r'을 계산 후 저장, 기존의 goal g에 도달하지 못해 reward를 받지 못한 sample도 additional goal g'에 도달한 경우도 reward signal이 발생할 수 있게 된다

* 이 두 알고리즘을 통해 학습을 수행하는데, 샘플의 경우 향상 HER는 실패한 에피소드에서도 유용한 학습 정보를 얻을 수 있게 되고. DDPG의 경우 리플레이스 메모리를 통해 학습 효율성을 향상시킨다. 또한 프로젝트의 희소 보상 문제도 HER 을 통해 해결이 가능해진다. 즉 효율적인 탐색과 수렴 및 성능과 안정성을 높일 수 있게 된다.

**-코드 확인**

**초기화**

*# 각 네트워크 파라미터에 대해 옵티마이저 초기화*

*self*.actor\_optim = torch.optim.Adam(*self*.actor\_network.parameters(), *lr*=*self*.args.lr\_actor)

*self*.critic\_optim = torch.optim.Adam(*self*.critic\_network.parameters(), *lr*=*self*.args.lr\_critic)

*#her 모듈 초기화*

*self*.her\_module = her\_sampler(*self*.args.replay\_strategy, *self*.args.replay\_k, *self*.env.compute\_reward)

*# replay buffer 초기화*

*self*.buffer = replay\_buffer(*self*.env\_params, *self*.args.buffer\_size, *self*.her\_module.sample\_her\_transitions)

초기화를 통해 actor와 critic, 안정적인 학습을 위해 target network와 main network 모두 선언한다. 학습을 위해 Adam optimizer를 사용했고, her\_sampler를 초기화할때 default arguments로서 future strategy를 사용했다. Mpi를 통해 여러 에피소드에서 각각의 에피소드에 대해 minibatch transitions를 얻었다.

**Learn**

각 에포크에 대해 샘플을 모으기 위한 에피소드 50번을 통해 replay buffer에 샘플을 저장한다.

*for* t *in* range(*self*.env\_params['max\_timesteps']):

*with* torch.no\_grad():

*#현재 상태 obs와 goal g를 이용해 input tensor를 만든다. 즉, obs와 goals를 같이 입력으로 넣어준다. == HER*

input\_tensor = *self*.\_preproc\_inputs(obs, g)

*#actor 네트워크를 이용해 action을 선택한다.*

pi = *self*.actor\_network(input\_tensor)

action = *self*.\_select\_actions(pi)

**Action을 선택하기 위해서 actor network를 사용했는데 이때 입력으로 HER구현을위해 observation과 goal모두를 넣어준다. 여기서 noise를 더해 최종적으로 action을 선택한다. 환경과 상호작용해서 얻은 observations와 goals들을 미니배치에 저장후 buffer에 저장한다. 버퍼에 저장된 데이터들로 수어번의 업데이트 수행후 target network를 soft하게 교체한다.**

**Her**

her\_indexes = np.where(np.random.uniform(*size*=batch\_size) < *self*.future\_p)

*#여기서 선택된 트랜지션은 같은 에피소드 상에서 어떤 future timestep에서 achived goal을 goal로 대체된다.*

*#현 에피소드의 total timesteps에서 샘플링된 각 트랜지션의 현재 timesteps을 빼서 남아있는 timesteps를 구한다음에 random하게 그중에서 offset을 선택한다.*

future\_offset = np.random.uniform(*size*=batch\_size) \* (T - t\_samples)

future\_offset = future\_offset.astype(int)

*#future\_offset을 더해서 future timesteps를 구한다.*

future\_t = (t\_samples + 1 + future\_offset)[her\_indexes] *#남아있는 에피소드에서 랜덤하게 선택된 future timesteps, 그중에서 her\_indexes에 해당하는 것만 선택된다.*

*# 샘플링된 her\_indexes에 해당하는 트랜지션의 goal을 future timesteps에서의 ag로 대체한다.*

future\_ag = *episode\_batch*['ag'][episode\_idxs[her\_indexes], future\_t]

transitions['g'][her\_indexes] = future\_ag

현재 저장된 버퍼 내에서 랜덤으로 relabel되는 transition을 샘플링하고, 어떤 future timestep의 achived goal로 교체할 것인지를 샘플링한다.

**-결과텍스트, 폰트, 스크린샷, 흑백이(가) 표시된 사진

자동 생성된 설명**

Ddpg+her을 통해 기본적인 태스크에서는 100epoch내에 일반적으로 0.9이상으로 success rate가 수렴했다. 하지만 보다 더 넓은 space에서 action이 필요한 위 프로젝트에서는 더 많은 시간이 필요했다. 먼저 한 테이블 위에서 랜덤하게 생성된 object를 다른 테이블의 target으로 옮기는 기본적인 task에서 학습했을 때, 일반적으로 700epoch이상에서 success rate가 0.1이상이 기록되었고, 1300 epoch이상부터 안정적으로 0.8이상의 success rate가 기록되었다. 학습된 agent를 통해 10개의 object를 연속으로 옮기는 task를 수행하였다.