

2월 4주차 레포트

간단한 예제 실행 - 1차원 바에서의 이동

[학습까지의 문제점들]

```
## AssertionError
try :
    assert self.action_space.contains(action)
except AssertionError :
    print("Action 값이 Action Space 범위를 벗어났습니다. : ", action)

>> A_space.contains(B)
>> A와 B가
>> np.ndarray의 형태
>> 동일한 dtype
>> 동일한 shape
>> 만족해야 AssertionError가 작동하지 않는다.

## Reset
self.position=np.array(self.observation_space.sample())
>> seeding을 사용하는 것보다
>> sample()을 사용하는 것이 더 편리하다

## State와 Action
이번의 Step Function의 경우에는
State를 Position으로 옮기고,
Position과 Action을 계산한 후,
다시 Position을 State로 옮기는 형식으로 진행하였다.
>> 이때, 최종결과물인 State는 Observation Space에서 지정한 형태여야한다.
>> 이때 편하게 하는 방법, position이 단일 숫자일시
>> self.position = self.state
>> self.state.shape
>> (1,)
>> self.position과 action 계산
>> self.position[0]과 action으로 계산
>> self.state=self.position
>> 이렇게 진행할 시, self.state에 대한 shape 오류가 뜨지 않는다.
```

[학습]

1. 환경설정 - 변수들

```

# 위치 설정 & MAP 설정
LF_MIN = 1
RT_MAX = 10
>> { - - - - - }
>> 위와 같은 1~10자리의 실수들로 가득찬 바로 구성된 MAP
>> LF_MIN : 제일 왼쪽은 위치상 1로 지정
>> RT_MAX : 제일 오른쪽은 위치상 10로 지정

# 보상 지정 & MAX_STEPS 지정
MAX_STEPS = 10
>> 최대로 이동할 수 있는 횟수 제한

def REWARD_STEP(self):
    REWARD = -self.count
    if self.count == self.MAX_STEPS :
        REWARD += - self.count
    return float(REWARD)

>> MAX_STEPS에 가까워질 수록 높은 패널티가 부과된다.

def REWARD_CONFIG(self,action):
    return float(abs(abs(self.config) - abs(action)))

>> 한칸 이동할 때, 주어진 Config에 맞춰서 이동할 시 높은 보상이 주어진다

def REWARD_GOAL(self):
    return 50

>> Goal 지점에 도달할 시, 가장 큰 보상이 주어진다.

def REWARD_AWAY(self):
    return -float(self.count)

>> Left와 Right 양 끝에 도달할 시, 매우 큰 패널티가 주어진다

# Error 범위 설정
error = 0.1
>> Goal까지의 위치 범위를 0.1로 설정

```

1. 환경설정 - 본격적

```

def __init__(self,config):
    self.config = config
    >> 원하는 이동거리 간격을 config로 설정

    self.action_space=gym.spaces.Box(-5.0,5.0,shape=(1,),dtype = np.float32)
    >> action의 범위를 -5.0 ~ 5.0으로 제한
    >> 너무 빨리 크게 움직이는 것을 방지하기 위해

    self.observation_space = gym.spaces.Box(1,10,shape=(1,),dtype=np.float32)

```

```
>> 1 ~ 10까지의 실수들로 구성된 state 형성

self.goal = ((self.LF_MIN + self.RT_MAX -1)/2)
>> goal지점 설정 : 5.0

self.reset()
>> 초기화
```

2. Reset Function

```
def reset(self):
    self.count = 0
    >> MAX_STEPS와 비교하기 위한 Step Function에 대한 Count 변수

    self.position=np.array(self.observation_space.sample())
    >> 주어진 Observation Space 상에서 Sample를 추출하여 초기 State로 설정

    self.reward = 0
    self.done = False
    self.info= {}
    >> 초기 Reward와 done(성공 여부), info를 비워둔다

    self.state = self.position

    return self.state
```

3. Step Function - 보상함수만 바꾼 Step Function입니다.

```
def step(self,action):
    if self.done:
        self.done=False

    elif self.count == self.MAX_STEPS:
        self.done=True
        self.count==0

        self.reward = self.REWARD_STEP()

    else :
        try :
            assert self.action_space.contains(action)
        except AssertionError :
            print("Action 값이 Action Space 범위를 벗어났습니다. : ", action)

        self.reward=self.REWARD_STEP() + self.REWARD_CONFIG(action)
        self.count +=1
```

```

self.position = self.state

if action < 0:
    if self.position+action<= self.LF_MIN :
        self.reward += self.REWARD_AWAY()
        self.position[0] = 1
    else :
        self.position +=action

        if abs(self.position-self.goal) <self.error :
            # on goal now
            self.reward += self.REWARD_GOAL()
            self.done = True

elif action > 0 :
    if self.position+action >= self.RT_MAX :
        #invalid
        self.reward += self.REWARD_AWAY()
        self.position[0] = 10

    else :
        self.position +=action

        if abs(self.position-self.goal) <self.error:
            # on goal now
            self.reward = self.REWARD_GOAL()
            self.done = True

self.state=self.position

try :
    assert self.observation_space.contains(self.state)
except AssertionError :
    print("State 값이 Observation Space 범위를 벗어났습니다. :", self.state)
return self.state,self.reward,self.done,self.info

```

>> Step Function의 순서

1. True 인가 False인가?

>> 임무가 끝났는가? 실행중인가?

2. 지금 현재 Step은 몇 단계인가?

>> MAX_STEPS일시 임무 종료

3. 1번과 2번을 통과하였을 시, 임무를 진행 중인 것으로 판단

>> 왼쪽 이동인가?

>> 제일 끝점에 도달하였는가?

>> 아니라면, 왼쪽으로 이동

>> 이동 후 Goal 지점에 도달하였는가?

>> 오른쪽 이동인가?

>> 제일 끝점에 도달하였는가?

>> 아니라면, 오른쪽으로 이동

>> 이동 후 Goal 지점에 도달하였는가?

Step Function의 순서도를 따라가면 다소 엉켜있다는 느낌

>> 수정 필요

```
## State에 Action이 진행된 상태

1. 이동을 시킨다.
  >> REWARD_STEP에 대한 패널티 부여
  >> 주어진 Config(걸음걸이)에 대한 패널티 부여
  >> 주어진 Config와 차이가 클수록 패널티 부여

2. Goal지점에 도달하였는가?
  >> Yes
    >> REWARD_GOAL 부여
    >> Goal 지점으로 갈 때도 보상을 부여하는가?
    >> 방향이 Goal으로 향할 때 보상?

  >> No
    >> 제일 끝단에 도착하였는가?
    >> YES
    >> REWARD_AWAY 부여

  >> NO
    >> Nothing
```

>> 수정 동영상

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/6529e91d-d2eb-447e-9c82-94f0566d5031/My_Viewer-2022-02-25_11.09.49.mp4

[실행까지의 문제점들]

```

Initial Position : [6.6541204]
2022-02-24 23:45:57,971 WARNING deprecation.py:45 -- DeprecationWarning: `compute_action` has been de
precated. Use `compute_single_action` instead. This will raise an error in the future!
Action Error : [16.916906]
Count : 1
Episode Reward : -2
Action Error : [48.512814]
Count : 2
Episode Reward : -4
Action Error : [82.1243]
Count : 3
Episode Reward : -6
Action Error : [-9.381205]
Count : 4
Episode Reward : -8
Action Error : [48.030582]
Count : 5
Episode Reward : -10
Action Error : [35.388756]
Count : 6
Episode Reward : -12
Action Error : [60.450768]
Count : 7
Episode Reward : -14
Action Error : [54.380825]
Count : 8
Episode Reward : -16
Action Error : [44.118965]
Count : 9
Episode Reward : -18
Action Error : [-24.877132]
Count : 10
Episode Reward : -20
Count : 10
Episode Reward : -22
Success
tot4766@tot4766:~/intern/python/rllib/gridworld$

```

>> 보상함수를 변경하시 전에, 주어진 Action Space를 벗어난 Action 값들이 자주 나타난다

[실행]

1. 순서

- (1). 학습된 Policy를 불러온다.
- (2). 임무가 완료될 때까지 Step를 진행한다.
- (3). 매 Step마다 Rendering을 실행한다.
- (4). 임무가 완료될 시, 구분될 수 있는 Rendering을 진행한다.

2. 코드

```

class Draw:

    def __init__(self):
        >> 학습된 파일에서 Environment를 불러온다.
        >> 저장된 Policy를 불러온다.
        >> Done과 Reward를 초기화시켜준다.

    def MyDisplay(self):
        >> 주어진 Goal의 위치에 따라 점을 그린다.
        >> 주어진 agent의 위치에 따라 점을 그린다.

    def MyTimer(self,time):
        >> 임무(Done)이 안 끝날 시, Step(Update_Step Func)를 진행시킨다.

    def Update_Step(self):
        >> Step을 진행한다.

    def Change(self):
        >> agent의 State를 Window 위치에 맞게 재설정한다.

    def main(self):
        >> OpenGL main Func

```