2월 2주차 레포트

GYM 예제 분석

1. Texi

(1). Environment Setting

```
class TaxiEnv(Env):

  def __init__(self):
  >> 기본적인 설정 + 보상 + 액션들의 상호 연관성 관련 정보들을 내포

  def encode(self,taxi_row,taxi_col,pass_loc,dest_idx):
  >> 현재의 택시의 위치 좌표를 부호화하여 하나의 숫자 or 리스트로 보내는 것으로 보인다.
  >> 하나의 숫자로 만들어서 보내는 것으로 보인다.

  def decode(self,i):
  >> 부호화된 하나의 숫자 or 리스트를 해석하는 역할이다.

  def step(self,a):
  >> 매 스탭마다 주어진 확률에 따른 각각의 액션들을 수행한다.

  def reset(self,*,seed:Optional[int]=None,return_info:bool = False,options:Optional[dict]=None,):

  def render(self,mode='human'):
  >> step마다 액션이 진행되는 것들을 Window상에 표기한다.
```

(2). Reward Function Setting

```
def __init__(self):
        self.desc = np.asarray(MAP, dtype="c")
        >> np.asarray : 주어진 행렬이 이미 존재하는 배열과 차이점(데이터 형태 등등)이 있을 경우 복사한다.
        >> 렌더링 업데이트로 생각해도 된다.

        >> MAP : 주어진 택시가 이동하는 이미지 렌더링
        >> dtype
            i : 부호가 있는 정수형
            u : 부호가 없는 정수형
            f : 실수형
            c : 복소수형


        self.locs = locs = [(0, 0), (0, 4), (4, 0), (4, 3)]
        >> 얘가 뜻하는 건 뭐지??
          >> 나타내는 것들은 위치 정보이다
            (0,0) : Red
            (0,4) : Green
            (4,0) : Yellow
            (4,3) : Blue

        num_states = 500
        >> 전체 액션이 진행되는 정도??
```

```
num_rows = 5
>> 차량이 앞뒤로 움직일 수 있는 거리 5

num_columns = 5
>> 5차선 도로

max_row = num_rows - 1
>> 코드상에서는 5행일시 4로 읽혀야 하므로 이렇게 표시한 것으로 이해

max_col = num_columns - 1
>> 코드상에서는 5열일시 4로 읽혀야 하므로 이렇게 표시한 것으로 이해

self.initial_state_distrib = np.zeros(num_states)
>> 초기 상태의 분포를 500개의 빈 리스트로 설정
>> 왜 초기 상태의 분포를 500개로 만들고 이것들로 빈 리스트 만들까?
  >> 초기 상태의 분포가 아니라 step이 500까지 진행되는 것이 아닌가?

num_actions = 6
>> 액션의 개수 6개
  이동
    상/하/좌/우
  이벤트
    상차/하차

self.P = {
    state: {action: [] for action in range(num_actions)}
    for state in range(num_states)
}
>> 매 스탭에 각각의 액션들이 들어가게 된다.
```

```
for row in range(num_rows):
    for col in range(num_columns):
        for pass_idx in range(len(locs) + 1):  # +1 for being inside taxi
        >> 왜 +1이 붙는가?
        >> pass_idx : 0, 1, 2, 3, 4


            for dest_idx in range(len(locs)):
            >> 왜 +1이 붙지 않는가?
            >> dest_idx : 0, 1, 2, 3


                state = self.encode(row, col, pass_idx, dest_idx)
                >> row,col : 택시의 위치
                >> pass_idx : 출발지(손님이 계신 위치)
                >> dest_idx : 목적지

                if pass_idx < 4 and pass_idx != dest_idx:
                    self.initial_state_distrib[state] += 1
                    >> step마다 액션이 진행되는 정도가 저장되는 것으로 보임
                    >> 그렇다면 왜 pass_dix = 4까지이고, dest_idx는 3까지인가?

                for action in range(num_actions):
                    # defaults
                    new_row, new_col, new_pass_idx = row, col, pass_idx
                    reward = (
                        -1
                    )  # default reward when there is no pickup/dropoff
                    done = False
                    taxi_loc = (row, col)
                    >> row, col은 택시의 위치를 나타낸다.
                    >> forans을 통해서 모든 구역을 돌아다니는 것이다.
                      >> 왜 목적지를 향하여 가는 것이 아니라 전부다를 돌아다니는 것인가?
                      >> 목적지에 도달하기 위해서는 전체를 돌아다니며 학습을 해야 가능하기 때문인가?
```

```
>> action 0&1 : 위/아래 이동
>> action 2&3 : 좌/우 이동
>> action 4&5 : 상차/하차 움직임(이벤트)

if action == 0:
    new_row = min(row + 1, max_row)
    >> 아래쪽으로 이동하였을 시, 5행을 벗어나면 안된다.
      >> 코드상에선 5행은 4로 인식되어진다.
      >> 그렇다면 4를 벗아나선 안된다.
      >> 아래쪽으로 1칸 이동하였을 시(row+1), 4보다 작아야 한다.

elif action == 1:
    new_row = max(row - 1, 0)
    >> 위쪽으로 이동하였을 시, 1행을 벗어나면 안된다.
      >> 코드상에서는 1행은 0로 인식되어진다.
      >> 그렇다면 0을 벗아나선 안된다.
      >> 위쪽으로 1칸 이동하였을 시(row-1), 0보다 커야한다.

if action == 2 and self.desc[1 + row, 2 * col + 2] == b":":
    new_col = min(col + 1, max_col)
    >> 오른쪽으로 이동하였을 시, 5열을 벗어나선 안된다.
      >> 코드상에서는 5열은 4로 인식되어 진다.
      >> 그렇다면 4를 벗어나선 안된다.
      >> 오른쪽으로 1칸 이동하였을 시 (col+1), 4보다 작아야 한다.

    >> 지나갈 수 있는 길(:)이어야 한다.
      >> 1+row의 이유 : MAP의 첫 행은 "+---------+" 이기에 이것들을 무시하기 위해
      >> 2*col+2의 이유
        >> texi가 MAP의 행렬로서의 위치가 아닌
        >> 일반적인 지도처럼 생각하였을 시 (a,b)에 있다면
        >> 내가 오른쪽으로 가려고 할 때 확인해야하는 건
        >> 일반적인 지도처럼 생각하였을 시 (a,b+1)의 길을 봐야한다.
          >> 일반적인 지도의 (a,b+1)의 열은 실제 MAP 행렬의 몇번째 열인가?
          >> 만약에 (1,1)에 위치했다고 한다면, 실제론 MAP에선 (2,3)에 위치
          >> 그리고 확인해야할 위치는 (2,4)
          >> 현재 texi의 MAP에서의 위치 : 2*col+1
          >> 확인해야할 위치 : (2*col+1)+1

elif action == 3 and self.desc[1 + row, 2 * col] == b":":
    new_col = max(col - 1, 0)
    >> 왼쪽으로 이동하였을 시 , 1열을 벗어나선 안된다.
      >> 코드상에서는 1열은 0으로 인식되어 진다.
      >> 그렇다면 0을 벗어나선 안된다.
      >> 왼쪽으로 1칸 이동하였을 시 (col-1), 0보다 커야한다.

    >> 지나갈 수 있는 길(:)이어야 한다.
      >> 1+row의 이유 : action 2와 동일
      >> 2*col의 이유
        >> 현재 texi의 MAP에서의 위치 : 2*col+1
        >> 확인해야할 위치 : texi에서의 왼편 = (2*col+1)-1

elif action == 4:  # pickup
    if pass_idx < 4 and taxi_loc == locs[pass_idx]:
        new_pass_idx = 4
        >> pass_idx < 4 의 이유 :
          >> pass_idx 0~3까지 손님들이 계신 위치들을 돌아가며 진행되고
          >> pass_idx 4에서는 끝나야하기 때문
          >> 그렇다면 그냥 pass_idx 3까지만 진행하면 되는 것 아닌가?
          >> 왜 pass_idx 4를 종료 신호로 두었을 까?
            >> pass_idx = 4 : 하차 신호


        >> taxi_loc = locs[pass_idx] 의 이유:
          >> 택시의 위치와 손님이 계신위치가 같아야 태울 수 있기 때문

    else:  # passenger not at location
        reward = -10
        >> 택시의 위치에 손님이 계시지 않을 경우
        >> pass_idx = 4 일경우
          >> 왜 이 경우도 보상이 -10인가?

elif action == 5:  # dropoff
```

```
                                if (taxi_loc == locs[dest_idx]) and pass_idx == 4:
                                    new_pass_idx = dest_idx
                                    done = True
                                    reward = 20
                                    >> 하차의 위치가 왜 새로운 승객의 위치가 되는가?
                                    >> done = True : 하차 완료
                                    >> reward = 20 : 옳은 하차 보상

                                elif (taxi_loc in locs) and pass_idx == 4:
                                    new_pass_idx = locs.index(taxi_loc)
                                    >> 지금 현재 택시가 손님의 하차 목록 중에 있는 위치에 있을 시
                                      >> 출발점은 현재의 택시 위치로 새로 설정
                                else:  # dropoff at wrong location
                                    >> 완전히 이상한 위치에 내려줬을 시
                                    reward = -10
                            new_state = self.encode(
                                new_row, new_col, new_pass_idx, dest_idx
                            )
                            >> 액션이 진행된 현재의 상태를 encode화 시킨다.
                              >> list화시키는 것보다 더 효율적인가?
                              >> 현재의 상태를 분류시키기 위해선 list보단 단순 숫자로 표기된 encode가 더 효율적일 것으로 보인다.

                            self.P[state][action].append((1.0, new_state, reward, done))
                            >> 각각의 state에 따른 액션에 정보들이 저장된다.
        self.initial_state_distrib /= self.initial_state_distrib.sum()
        >> 분포도를 구하기 위한 것이므로, 전체에서 각각들을 나누어 준다.
        self.action_space = spaces.Discrete(num_actions)
        self.observation_space = spaces.Discrete(num_states)
        >> Action(6), State(500)의 각각의 space 설정
```

(3). Encoding & Decoding

```
    def encode(self, taxi_row, taxi_col, pass_loc, dest_idx):
        # (5) 5, 5, 4
        i = taxi_row
        i *= 5
        i += taxi_col
        i *= 5
        i += pass_loc
        i *= 4
        i += dest_idx
        return i
        >> 단순 산수를 통해서 4가지의 정보를 하나의 정보로 합쳐놓는다.

    def decode(self, i):
        out = []
        out.append(i % 4)
        i = i // 4
        out.append(i % 5)
        i = i // 5
        out.append(i % 5)
        i = i // 5
        out.append(i)
        assert 0 <= i < 5
        return reversed(out)
        >> 주어진 하나의 정보(i)에서 encode에서 진행하였던 단순 산수들의 역순을 통하여
        >> 반대 순서로 각각의 정보들을 추출한다.
        >> 최종적으로, list를 역순으로 정렬시켜 원하고자하는 정보를 얻을 수 있게 된다.

>> 의문점
>> 그렇다면 encoding과 decoding을 하여 얻는 이점은 무엇인가?
>> 이러한 과정을 통하면서도 보안이 필요한가?
>> 아니면 계산되는 과정에서 조금 더 효율적으로 되기 때문인가?
```

(4). Step & Reset

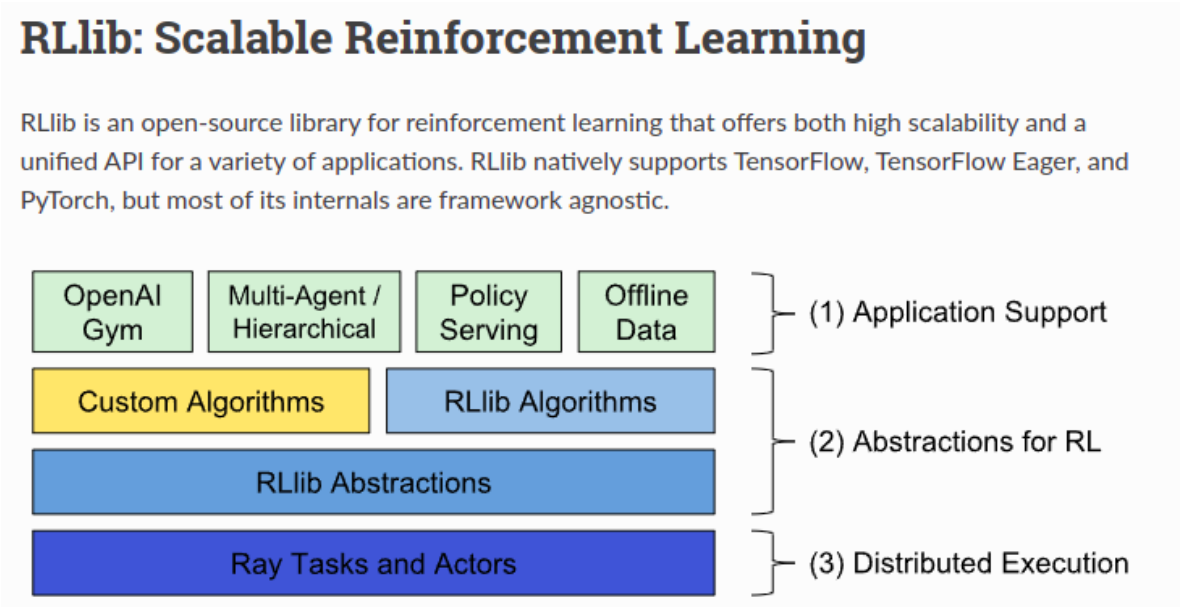
```
    def step(self, a):
        transitions = self.P[self.s][a]
        >> 주어진 State의 특정 액션에 대한 정보들을 추출한다.

        i = categorical_sample([t[0] for t in transitions], self.np_random)
        p, s, r, d = transitions[i]
        self.s = s
        self.lastaction = a
        return (int(s), r, d, {"prob": p})

    def reset(
        self,
        *,
        seed: Optional[int] = None,
        return_info: bool = False,
        options: Optional[dict] = None,
    ):
        super().reset(seed=seed)
        self.s = categorical_sample(self.initial_state_distrib, self.np_random)
        self.lastaction = None
        if not return_info:
            return int(self.s)
        else:
            return int(self.s), {"prob": 1}
```

@@ GYM FUNCTION - categorical_sample

```
def categorical_sample(prob_n, np_random):
    """
    Sample from categorical distribution
    Each row specifies class probabilities
    """
    prob_n = np.asarray(prob_n)
    csprob_n = np.cumsum(prob_n)
    return (csprob_n > np_random.rand()).argmax()

>> 카테고리 분포(Categorical Distribution)
  >> 베르누이 분포의 확장판
    베르누이 분포는 0이나 1 (또는 -1이나 1)이 나오는  확률변수의 분포로 동전을 던져서
    나오는 경우, 동전을 던져 나오는 결과를 묘사할 때 사용할 수 있다.

    동전이 아닌 주사위를 던져서 나오는 경우를 묘사할 때는 카테고리 분포를 사용할 수 있다.
    카테고리 분포는 1부터 K까지의 K개의 정수 값 중 하나가 나오는 확률변수의 분포이다.
    따라서 주사위를 던져 나오는 눈금의 수는 K=6 인 타케고리 분포가 된다.

  >> 원-핫-인코딩(One-Hot-Encoding)
    카테고리 분포를 가진 확률변수는 원래 카테고리인 스칼라 값을 출력하는 확률변수지만
    1과 0으로만 이루어진 다차원 벡터로 변형하여 사용한다.
    이러한 인코딩 방식을 원-핫-인코딩이라고 한다.

  >> 주사위의 원-핫-인코딩
    경우의수
      x=1 x=(1,0,0,0,0,0)
      x=2 x=(0,1,0,0,0,0)
          ......
      x=6 x=(0,0,0,0,0,1)
```

RLlib 분석

1. GYM과의 차이점



Application Support와 Abstratctions for RL의 차이점은 무엇인가?