기초 인공지능

2023 Second Semester CSE 4185
Assignment#06

1. Requirements

- python version >= 3.6
- numpy >= 1.15

2. 문제 설명

주어진 폴더에는 데이터 파일(model.json), main.py, hw6.py, utils.py가 있다. 데이터 파일은 이번 과제에서 사용할 wall, state, reward, action, gamma에 대한 정보가 json 파일로 저장되어 있다.

[그림 1] 데이터 파일 예시

utils.py는 데이터 파일을 불러와 각 정보 별로 분리(load_MDP())하고, grid환경, 벽, terminal state, utility를 시각화(visualize())하는데 사용된다. main.py는 데이터를 읽고 각 함수를 실행하여 결과를 확인할 수 있는 코드들이 작성되어있다. 작성해야 하는 함수는 모두 hw6.py에 정의되어 있다. hw6.py의 주석을 참고하여 주어진 함수를 모두 작성해야 한다.

결과 값은 아래 command를 콘솔에 입력하여 확인한다.

```
python main.py
```

함수마다 출력 예시가 주어지며, 출력 예시와 본인이 작성한 코드의 결과가 같은지 확인하고 다음 문제로 넘어가기를 권장한다.

작성해야하는 함수는 총 3개로 이루어져있다. 오류를 전파하는 기본 코드(raise RuntimeError("You need to write this part!"))를 지우고 주어진 주석에 맞게 함수를 구현해야한다.

각 문제와 문제별로 구현해야할 각 함수에 대한 설명은 아래와 같다. input과 output에 대한 상세한 설명은 hw6.py의 주석을 참고하면 된다.

▶ 환경 설명

main.py에 작성되어 있는 model=utils.load_MDP('model.json')은 아래와 같이 활용된다.

- **model.M, model.N** : 주어진 환경은 $M \times N$ 차원의 grid world이며 r-th row, c-th column에 해당하는 셀은 각 state (r, c) 정보를 의미한다.
- **model.gamma** : discount factor를 의미한다.
- **model.W** : $M \times N$ boolean matrix로, 주어진 환경에 존재하는 벽을 나타낸다. model.W[r, c] == True이면 (r, c) state는 벽을 의미하므로, agent가 해당 state로 이동할 수 없다.

wall 예시)

```
array([[False, False, False, False],
       [False,  True, False, False],
       [False, False, False, False]])
```

- **model.T** : $M \times N$ boolean matrix로 model.T[r, c] == True이면 terminal state를 의미한다.

  terminal state 예시)

```
array([[False, False, False,  True],
       [False, False, False,  True],
       [False, False, False, False]])
```

- **model.R** : $M \times N$ matrix로 reward를 의미한다. agent가 이동한 next state가 terminal state에 해당한다면 big reward $+1, -1$, 해당하지 않는 셀은 small negative number가 부여된다.

  reward 예시)

```
array([[-0.04, -0.04, -0.04,  1.  ],
       [-0.04, -0.04, -0.04, -1.  ],
       [-0.04, -0.04, -0.04, -0.04]])
```

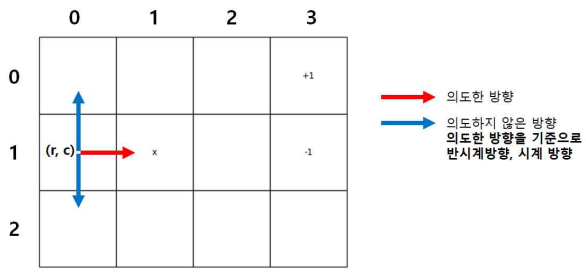
- **model.D :** $M \times N \times 3$ numpy 배열이며, 각 state에서 agent의 움직임에 대한 확률을 정의한다. non-terminal state (r, c)에서 agent는 left(0), up(1), right(2), down(3) 총 4가지 방향으로 움직일 수 있다. 하지만, 언제나 의도한 방향으로 움직이지는 않으며, 확률에 근거하여 움직이게 된다.
  즉, (r, c)에서 agent가 **의도한 방향으로 움직일 확률** $D[r,c,0]$, **의도한 방향의 반시계 방향으로 움직일 확률** $D[r,c,1]$, **의도한 방향의 시계 방향으로 움직일 확률** $D[r,c,2]$을 나타낸다.
  (r,c) state에서 모든 action을 취할 확률의 합은 1이며, 의도한 방향과 반대 방향으로는 이동하지 않는다. 테스트 케이스에서 각 action에 대한 확률은 달라질 수 있다.

  action probability 예시)

```
array([[[0.8, 0.1, 0.1],
        [0.8, 0.1, 0.1],
        [0.8, 0.1, 0.1],
        [0.8, 0.1, 0.1]],

       [[0.8, 0.1, 0.1],
        [0.8, 0.1, 0.1],
        [0.8, 0.1, 0.1],
        [0.8, 0.1, 0.1]],

       [[0.8, 0.1, 0.1],
        [0.8, 0.1, 0.1],
        [0.8, 0.1, 0.1],
        [0.8, 0.1, 0.1]]])
```

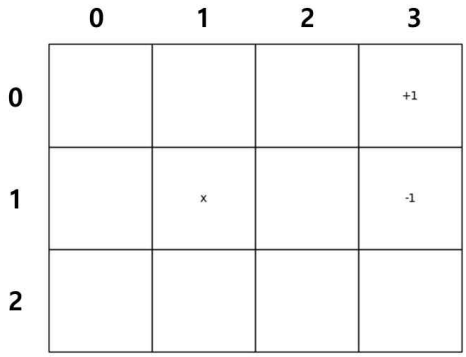


[그림 2] model.D 예시

그림 2의 경우, 해당 state (1,0)에서 agent가 **의도한 방향**이 **right(2)**면
- D[1,0,0] : right 방향으로 이동할 확률
- D[1,0,1] : up 방향으로 이동할 확률
- D[1,0,2] : down 방향으로 이동할 확률

마찬가지로, 해당 state (1,0)에서 agent가 의도한 방향이 left(0)면
- D[1,0,0] : left 방향으로 이동할 확률
- D[1,0,1] : up 방향으로 이동할 확률
- D[1,0,2] : down 방향으로 이동할 확률

참고로, 주어진 grid world 환경은 왼쪽 위 모서리 (0,0), 왼쪽 아래 모서리 (2,0), 오른쪽 위 모서리 (0,3), 오른쪽 아래 모서리 (2,3)으로 구성되며,
agent가 특정 state (r, c)에서 left 방향으로 이동하고 싶은 경우 (r, c-1), up 방향으로 이동하고 싶은 경우 (r+1, c)로 계산한다.



[그림 3] $M \times N$ grid world

▶ 함수1. Transition Probability 계산

value iteration 수행 전 방대한 연산량을 방지하기 위해, transition probability $P(s'|s,a)$를 미리 계산한다. 각 state (r,c)에서 action $a \in \{0,1,2,3\}$ (left(0), up(1), right(2), down(3))를 취했을 때, state (r',c')로 갈 확률 $P[r, c, a, r', c']$를 계산하는 **compute_transition_matrix** 함수를 구현해야 한다.

함수의 input은 앞서 설명했던 action, state, 등의 환경 정보가 들어있는 model이며, transition probability P를 반환하면 된다.

만약, 각 state가 특정 action을 취할 때, 이동하는 다음 state가 grid 환경의 경계 범위를 벗어나거나, 벽에 해당한다면 현재 state를 유지한다.

편의를 위해 해당 state가 terminal state인 경우는 $P[r, c, :, :, :] = 0$으로 설정한다.
해당 state가 벽인 경우의 이동 확률은 계산되어야 한다.

계산된 P에 대한 정답 확인을 위해 solution_P.npy 파일을 첨부하였으며, main.py에서 확인할 수 있다.

```
# 문제 1. Transition Probability
P = hw6.compute_transition_matrix(model)
sol_P = np.load("solution_P.npy")
if not np.array_equal(P, sol_P):
    raise ValueError(
        "The computed transition matrix P does not match the ground truth."
    )
```

[그림 4] Transition Probability 정답 확인 방법

▶ 함수2. Next Utility 계산

가능한 discounted rewards의 expected sum을 가장 크게 만들도록 utility $U(s)$를 업데이트하는 **update_utility** 함수를 구현한다. 구현에 필요한 수식은 아래와 같다.

수식)

$$U_{i+1}(s) = \max_{a \in A(s)} \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma U_i(s')]$$

이때, 본 과제에서 reward는 특정 state (r,c)에 도달했을 때, 주어지는 즉각적인 보상이므로 action a나 next state s'에 의존하지 않는다. 따라서 $R(s,a,s') = R(s)$로 표현된다.

수식으로 표현하면 다음과 같다.

$$U_{i+1}(s) = \max_{a \in A(s)} \sum_{s'} P(s'|s,a)[R(s) + \gamma U_i(s')]$$

$R(s)$는 상수로, 각 action에 대해 최대 utility를 구할 때 변하지 않으므로 최대값 연산에 영향을 주지 않는다. 따라서 최대값 연산 밖으로 분리할 수 있으며, 수식은 다음과 같다.

$$U_{i+1}(s) = R(s)\max_{a \in A(s)} \sum_{s'} P(s'|s,a) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U_i(s')$$

이때, 모든 action을 취할 확률의 합은 1이 되므로 $R(s)\max_{a \in A(s)} \sum_{s'} P(s'|s,a)$에서 $\max_{a \in A(s)} \sum_{s'} P(s'|s,a) = 1$이 되어 생략하면 최종적으로 아래 수식이 된다.

최종 수식)

$$U_{i+1}(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U_i(s')$$

update_utility 함수 구현은 최종적으로 간소화된 수식을 활용해 구현하면 된다.

함수의 input은 model, 앞서 계산된 transition probability(P), 현재 utility 정보(U_current)이며 계산된 U_next를 반환하면 된다. U_current는 $M \times N$ matrix로 0으로 초기화된다.

```
# 문제 2. Update utility
U_current = np.zeros((model.M, model.N))
U_next = hw6.update_utility(model, P, U_current)
model.visualize(U_next, save_path=True, figname="U_next")
```

[그림 5] update_utility

$U_{current}(s)$를 0으로 초기화 후 한 번 업데이트 된 $U_{next}(s)$의 예시는 다음과 같다.

-0.040	-0.040	-0.040	1.000
-0.040	x	-0.040	-1.000
-0.040	-0.040	-0.040	-0.040

[그림 6] 초기화된 Utility 한 번 업데이트

효율성은 점수 산정에 포함되지 않으므로 np.dot(), for문 등 구현 방식의 제한이 없다.

▶ 함수3. Value Iteration 구현

update_utility 함수를 반복적으로 호출하면서 모든 state에 대해 $|U_{i+1}(s) - U_i(s)| < \epsilon$이면, 수렴되었다고 판단하여 iteration을 종료하도록 **value_iteration** 함수를 구현한다.
transition probability P는 **value_iteration** 함수 내에서 **compute_transition_matrix** 함수를 호출하여 사용한다. iteration은 수렴되지 않아도 최대 100회로 제한한다. ($\epsilon = 1e-3$로 코드 내 정의 되어 있음)

함수의 input은 model로 최종 업데이트 된 utility를 반환한다.

```
# 문제 3. Value iteration
U = hw6.value_iteration(model)
model.visualize(U, save_path=True, figname="result")
```

[그림 7] value iteration

실행 결과) value_iteration 함수를 구현 후 정상적으로 실행이 되면, 아래와 같은 결과 이미지가 생성된다.

0.812	0.868	0.918	1.000
0.762	x	0.660	-1.000
0.705	0.655	0.611	0.387

[그림 8] 최종 utility 업데이트 결과

3. 보고서

보고서 분량 제한은 없으나, 반드시 다음과 같은 내용이 포함되어야한다.

1. 각 함수마다 구현한 방법에 대한 간략한 설명
2. 실행 결과 후 저장된 사진 첨부

참고) model.visualize()
해당 함수에 utility 정보를 넣은 후 저장된 그림 6, 8과 동일한 결과를 보고서에 첨부한다.

```
# 문제 2. Update utility
U_current = np.zeros((model.M, model.N))
U_next = hw6.update_utility(model, P, U_current)
model.visualize(U_next, save_path=True, figname="U_next")

# 문제 3. Value iteration
U = hw6.value_iteration(model)
model.visualize(U, save_path=True, figname="result")
```

[그림 9] model.visualize() 활용

4. 주의사항

- 코드 실행시 출력 화면과 보고서에 첨부된 화면 캡처 내용이 반드시 동일해야 한다. (다를 경우 코드 실행 시의 결과를 기준으로 점수를 산정할 것)
- 함수 구현에 대한 라이브러리는 numpy만 가능하며 추가적인 test case가 있을 수 있음.
- 본인이 **작성한 코드에 대하여 annotation**을 작성할 것. (미 작성 시 감점)
- **copy check 적발시 0점 처리.**

5. 제출

아래 두 가지 파일만 압축하여 AI분반_학번_이름.zip으로 사이버 캠퍼스에 업로드 한다.
- python file: hw6.py
- report: AI분반_학번_이름.pdf