소프트웨어특강2 : 강화 학습 개론

Assignment 1: Planning FrozenLake

1. 목표

- FrozenLake 환경에서의 Optimal Value Function 과 Optimal Policy 찾기

2. 개발 환경

- Python (version 3.5+)

3. 템플릿 파일 및 제출 파일

- 제공되어지는 템플릿 파일 3개
  - frozenlake.py: FrozenLake 환경의 MDP 제공
  - planning.py: FrozenLake 환경의 optimal value function 과 optimal policy 를 찾음
  - main_lib.py: policy iteration 및 value iteration 함수 (비어있음)
- 제출할 파일 1개
  - 위 소개된 파일 중 **main_lib.py** 만 수정가능하며 최종 결과물로 제출한다. Optimal value function 과 optimal policy 를 찾을 수 있도록 주어진 템플릿 함수 코드를 완성한다. main_lib.py 파일 내에선 원하는 만큼 다른 함수, 변수 등을 추가할 수 있다.
  - frozenlake.py 와 planning.py 는 제출하지 않습니다.

4. FrozeLake 환경 소개

*** frozenlake.py 파일은 제출하지 않습니다.**
테스트 목적으로 환경을 다르게 설정해보는 것은 괜찮습니다.

```
MAP = [
        "SFFF",
        "FHFH",
        "FFFH",
        "HFFG"
    ]
```

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

이렇게 4x4 구성으로, S 는 시작점, G 는 목표점, H 는 구멍, F 는 빙판길임.

시작점에서 시작하여, 상하좌우 action 을 통해 움직임. 구멍 이나 목표점에 도달하면 하나의 에피소드가 종료함.

state 는 위 오른쪽 그림과 같이 0 부터 15 로 표현하며,

action 은 LEFT = 0, DOWN = 1, RIGHT = 2, UP = 3 이다.

- **Not Slippery (is_slippery = False)** 인 경우:

  state 0 에서 RIGHT action 을 하면 state 1 로,

  state 0 에서 DOWN action 을 하면 state 4 로 변경 됨.

  반면 state 0 에서 UP 이나 LEFT action 을 하면 갈 수 없으므로 state 0 으로 유지됨.

- **Slippery (is_slippery = True)** 인 경우:

  주어진 action 에 대해서, 미끄러지는 불확실한 state 이동이 발생.

  다음과 같이 (action - 1) % 4, action, (action + 1) % 4 방향으로 1/3 확률로 미끄러짐.

  예: DOWN action 에 대해 LEFT, DOWN, RIGHT 방향으로 1/3 확률로 미끄러짐.

reward 는 목표점 state 15 에 도착한 경우에만 +1 그 이외의 모든 경우 0 이다.

FrozenLakeEnv 클래스에서는 이러한 모든 정보를 self.MDP 에 다음과 같은 형식으로 표현함.

self.MDP 는 4 차원 중첩 리스트.

self.MDP[s][a] 는 state s 에서 action a 를 했을 때 아래 와 같은 2 차원 리스트를 제공함.

- **Not Slippery (is_slippery = False)** 인 경우:

  self.MDP[state][action] = [[action 방향으로 이동 확률 항상 1.0, 도착할 state, reward]]

  예: self.MDP[0][1] = [[1.0, 4, 0.0]]

  => state 0 에서 1 (down) 방향으로 움직이면 100%로 state 4 로 이동하고 reward 는 0 임.

- **Slippery (is_slippery = True)** 인 경우:

  self.MDP[state][action] = [

  [action-1 방향으로 미끄러질 확률 (1/3), 도착한 state, reward]

  [action 방향으로 미끄러질 확률 (1/3), 도착한 state, reward]

  [action+1 방향으로 미끄러질 확률 (1/3), 도착한 state, reward]

  ]

  예: self.MDP[0][1] = [[0.33.., 0, 0.0], [0.33.., 4, 0.0], [0.33.., 1, 0.0]]

  => state 0 에서 1 (down) 액션을 선택했을 때 나올 수 있는 3 가지 경우의 수가 있음.
  state 0, state 4, state 1 로 도착할 확률이 각각 1/3, 그리고 reward 는 모두 0 임.

5. planning.py 설명

*** planning.py 파일은 수정하지 않으며 제출하지도 않습니다.**

해당 파일을 실행하면 아래와 같이 Not Slippery / Slippery 여부와, Policy Iteration / Value Iteration 방식을 선택한다.

```
1.Not Slippery, 2.Slippery : 1
1.Policy Iteration, 2.Value Iteration : 2
```

그림 1. 실행 옵션 설정

선택에 따라 env = FrozenLakeEnv(is_slippery=**False**) 또는 env = FrozenLakeEnv(is_slippery=**True**) 로 환경을 생성하고 수행할 Iteration 방식에 따라 아래 두 함수를 선택적으로 실행한다.

```
policy, V = main_lib.policy_iteration( env )
policy, V = main_lib.value_iteration( env )
```

반환된 State-Value function 과 policy 출력 예시 다음페이지를 참조.

참고로 policy 는 deterministic & greedy 하게 최대 값을 선택하며 최대 값이 동일한 경우 action 인덱스 기준으로 작은 값을 선택한다.

Policy Iteration 과 Value Iteration 선택은 계산 방식이 다를 뿐 최종 결과는 동일함.

6. main_lib.py 설명

*** main_lib.py 파일명 그대로 제출합니다.**

아래 두 함수를 구현. 필요시 다른 함수/변수들을 추가 가능.

def policy_iteration (env, gamma=0.99, theta=1e-8):

def value_iteration (env, gamma=0.99, theta=1e-8):

gamma 는 discount factor 이며, theta 는 iteration 을 멈추는 기준 값으로 사용.

1. Not Slippery, 2. Slippery: **1**

1. Policy Iteration, 2. Value Iteration: **1**

Optimal State-Value Function:

0.951 0.961 0.970 0.961

0.961 0.000 0.980 0.000

0.970 0.980 0.990 0.000

0.000 0.990 1.000 0.000

Optimal Policy [LEFT, DOWN, RIGHT, UP]:

[0. 1. 0. 0.] [0. 0. 1. 0.] [0. 1. 0. 0.] [1. 0. 0. 0.]

[0. 1. 0. 0.] [1. 0. 0. 0.] [0. 1. 0. 0.] [1. 0. 0. 0.]

[0. 0. 1. 0.] [0. 1. 0. 0.] [0. 1. 0. 0.] [1. 0. 0. 0.]

[1. 0. 0. 0.] [0. 0. 1. 0.] [0. 0. 1. 0.] [1. 0. 0. 0.]

그림 2. 결과 예시 1

1. Not Slippery, **2. Slippery: 2**

1. Policy Iteration, **2. Value Iteration: 2**

Optimal State-Value Function:

0.542 0.499 0.471 0.457

0.558 0.000 0.358 0.000

0.592 0.643 0.615 0.000

0.000 0.742 0.863 0.000

Optimal Policy [LEFT, DOWN, RIGHT, UP]:

[1. 0. 0. 0.] [0. 0. 0. 1.] [0. 0. 0. 1.] [0. 0. 0. 1.]

[1. 0. 0. 0.] [1. 0. 0. 0.] [1. 0. 0. 0.] [1. 0. 0. 0.]

[0. 0. 0. 1.] [0. 1. 0. 0.] [1. 0. 0. 0.] [1. 0. 0. 0.]

[1. 0. 0. 0.] [0. 0. 1. 0.] [0. 1. 0. 0.] [1. 0. 0. 0.]

그림 3. 결과 예시2

7. 제출 관련

- 제출 마감일: 10.30 (수) 23:59.
  - 지연 제출은 24시간마다 -15 점, 72시간 초과는 받지 않습니다.
  - 표절은 양쪽 모두 0점 처리 되며, 두 번 이상 반복 시에는 F 입니다.
- "main_lib.py" 파일명을 그대로 사용하고 해당 파일 1개만 iCampus 로 제출합니다.

8. 채점 기준

- Total 100 points
  - 20 points: Not Slippery & Policy Iteration 로 optimal value function & optimal policy 찾음
  - 20 points: Not Slippery & Value Iteration 로 optimal value function & optimal policy 찾음
  - 20 points: Slippery & Policy Iteration 로 optimal value function & optimal policy 찾음
  - 20 points: Slippery & Value Iteration 로 optimal value function & optimal policy 찾음
  - 20 points: 환경을 변화 시켰을 경우에도 optimal value function & optimal policy 찾음