# 과제: Reinforcement learning

**[1] 과제 개괄**

본 과제는 강화학습에 대한 것이다. 다룰 문제는 강의노트에서 소개한 frozen lake 문제이다. 강화학습 기법 중 Q-learning 기술을 사용하도록 한다. 이 기술을 사용하여 optimal policy 를 구한다 (즉 q 함수를 구한다.) 따라서 이 문제는 control 문제이다. 프로그램은 optimal policy (즉 optimal q-values) 를 학습하는 단계와 그 후에 오는 test 단계로 구성된다. test 단계에서는 구한 q values 를 기반으로 greedy 하게 에피소드를 구해 본다(즉 start 에서 terminal state 까지 가는 경로를 구하여 출력한다). 다음 예는 테스트 단계에서의 출력의 일부 모습이다:

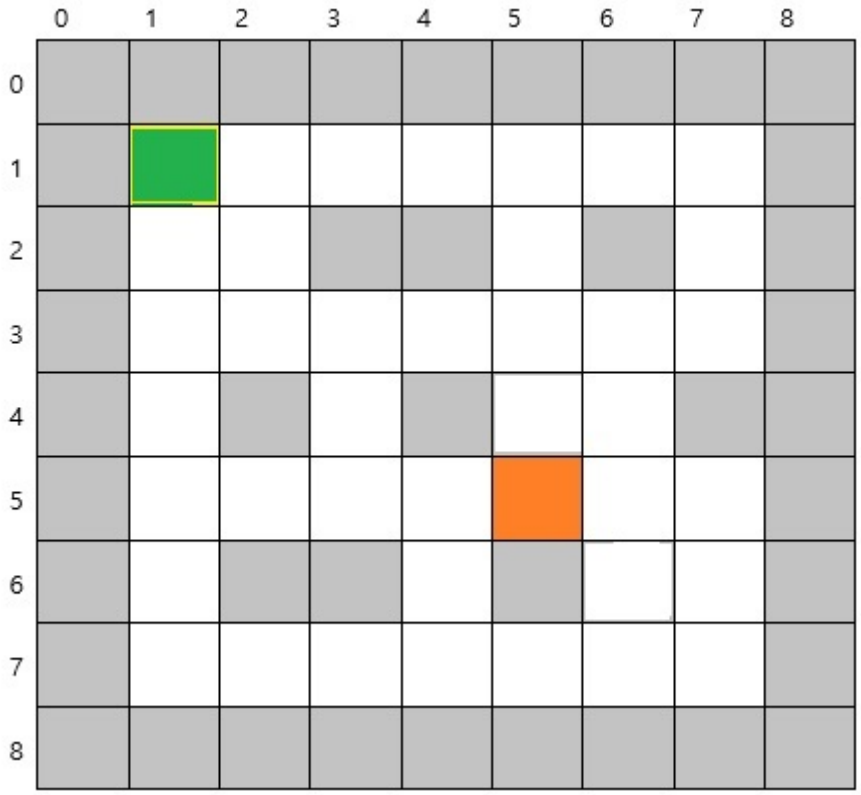
```
Episode: 0  start state: ( 1 ,  1 )
The move is: right  that leads to state ( 1 ,  2 )
The move is: right  that leads to state ( 1 ,  3 )
The move is: down   that leads to state ( 2 ,  3 )
The move is: down   that leads to state ( 3 ,  3 )
The move is: down   that leads to state ( 4 ,  3 )
The move is: right  that leads to state ( 4 ,  4 )
Episode has ended. Total reward received in episode =  9

Episode: 1  start state: ( 1 ,  1 )
The move is: right  that leads to state ( 1 ,  2 )
The move is: right  that leads to state ( 1 ,  3 )
The move is: down   that leads to st
The move is: down   th
The move is: down
```

본 과제는 두 개의 소과제로 구성되어 두번에 걸쳐 진행된다.

- 소과제-1: table 기반의 Q-learning 방법론을 사용하여 개발한다.
- 소과제-2: function approximation 에 기반한 방법론을 사용한다. 이를 위해 DQN(deep Q-network) 을 사용하도록 한다(다음 수업에서 다룰 주제임).

**[2] 문제 정의**

(2-1) 다음의 환경을 사용한다. 즉 9 X 9 의 frozen lake 이다.



cell 정의: 녹색: start; 주황색:goal; 흰색: frozen; 회색: hole.

(2-2) actions:

모든 셀은 다음 4가지 moving 액션이 모두 가능하다: 0(up), 1(right), 2(down), 3(left)

(2-3) rewards:

F/S 로 갈 때 0; G 로 갈 때 9; H 로 갈 때 -9.

**[3] 소과제-1**

table 기반의 Q-learning 알고리즘을 구현하고 실험한다. 강의노트 "RL_2_rev1.pdf"의 20쪽에 소개된 알고리즘을 이용한다. 제공되는 guide 프로그램을 수정하여 사용해도 좋다. guide 프로그램에서 다음 두 가지를 수정하여야 한다.

- **수정요청-1:** 위 그림과 같은 environment 를 사용하도록 가이드 프로그램을 수정한다.
- **수정요청-2:** 한 state s 에서 다음 취할 action 을 선택하는 두 개의 함수 choose_action_with_epsilon_greedy, choose_action_with_greedy 가 있다. 이 함수들은 문장 max_action = np.argmax (q_a_list) 를 사용하여 q 값이 최대인 액션 하나를 선택한다. 입력 q_a_list 는 각 액션에 대한 q 값을 가지는 numpy 배열이다. 여기서 문제가 되는 점은 guide 프로그램은 np.argmax 를 사용한다는 점이다. 예를 들면 state s1 에서 가능한 action 이 6 개라 하자. 그리고 각 q 값은 다음과 같다고 하자:

q(s, a0)= 0.15 / q(s, a1)= 0.25 / q(s, a2)= 0.1 / q(s, a3)= 0.25 / q(s, a4)= 0.1 / q(s, a5)= 0.25

최대값을 가지는 액션이 2개 이상인 경우를 생각하자. 위 경우에 상태 s1 을 주며 위 두 함수를 호출 할 때마다 그 내부에서 호출된 np.argmax 는 항상 같은 액션 a1을 내준다 (최대값을 가지는 첫 액션). 이렇게 되면 문제가 되는 이유는 최대값을 가지는 다른 두 액션 a3, a5 들도 있는데 이들을 통해 갈 수 있는 길은 거의 탐험(explore)되지 않는다는 점이다.

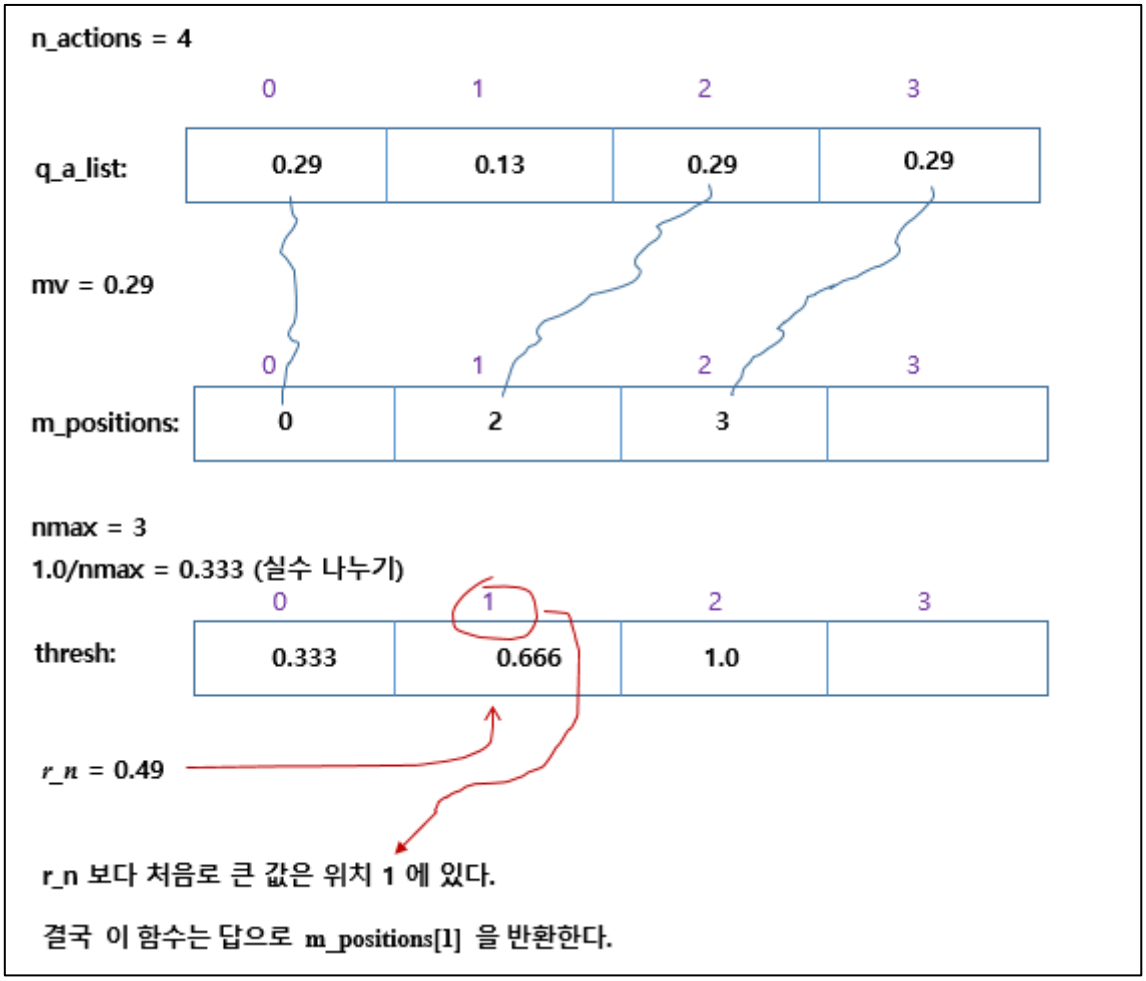
이 문제점을 개선하기 위해서 np.argmax 대신에 my_argmax 함수를 개발하여 사용하자. 이 함수는 다음처럼 동작한다. (n_actions 는 각 상태에서 가능한 총 action 의 수라고 하자.)

def my_argmax(q_a_list):

- 배열 q_a_list 에서 최대값을 가지는 위치들과 그 갯수를 알아 내어 다음 변수에 넣자.
  - 갯수: nmax, 최대값들의 위치: m_positions 라는 리스트.
  - 방법: 먼저 q_a_list 를 스캔하여 가장 큰 값을 알아낸다. 이 값을 mv 라 하자.
    - 그 다음 q_a_list 를 다시 스캔하면서 mv와 같은 값을 가지는 위치들을 m_positions 에 넣는다.
    - m_positions의 길이를 nmax 에 넣는다.
- 길이가 n_actions 인 thresh 라는 numpy 배열을 준비한다. 여기에 다음처럼 nmax 개의 수를 차례로 넣는다: 위치 i 에 $(i+1)\frac{1.0}{nmax}$ (주의: 실수 나누기이어야 함) 를 넣음.
- 0 ~ 1 사이의 random number 를 생성한다(이를 r_n 이라 하자).
- r_n 을 각 구간의 수와 처음부터 차례로 비교한다.
- r_n 보다 크거나 같은 값을 가지는 thresh 의 첫 위치가 어디인지를 알아 내자 (for 루프 이용).
- 이 첫 위치를 함수 my_argmax 의 결과로 반환하고 종료한다.

이 함수의 동작을 설명하는 그림은 다음과 같다.

max_action = np.argmax (q_a_list) 문장 대신에 max_action = my_argmax(q_a_list) 를 사용한다.




- 제출물: 프로그램(.py 파일), 수행 중에 나오는 출력을 캡쳐하여 넣은 파일 (출력이 많으면 여러번 캡쳐하여도 됨).

**[4] 소과제-2**

여기에서는 위의 table 기반이 아닌 방식을 사용한다. 즉 table 대신에 function-approximation 기법을 사용한다. function 으로 우리는 neural network 를 사용한다. 특히 유명한 DQN 기법을 사용한다. 프로그램의 절차는 소과제-1 과 동일하게 두 단계 즉 학습과 테스트로 구성된다.

학습단계: deep Q-learning 기술을 이용한다. 특히 주요 기술로 correlation을 해결하는 capture and replay 기법과 nonstationary 문제를 해결하기 위해 두개의 신경망(separate networks)을 이용하는 기법을 사용한다. 이에 대한 자세한 사항은 강의노트와 guide program 을 참고한다.

테스트 단계에서 수행하는 작업 및 출력 방법은 소과제-1 과 완전히 동일하다.

** 소과제-2 을 위한 정보는 앞으로 추가로 제공할 예정이다.