

# 인공지능 Project 2

이름 : 이준휘

학번 : 2018202046

교수 : 박철수 교수님

분반 : 금 3, 4

## 1. Introduction

해당 과제는 Dynamic Programming(DP)를 이용하여 Policy evaluation, Policy improvement, 그리고 Value iteration을 구현하는 것이다. 주어진  $7 \times 7$  크기의 grid-world를 사용하며 Action은 상하좌우 4가지 케이스로 이루어져 있다. 만약 grid-world 밖으로 나가는 action을 취할 경우 제자리로 돌아오며 중간중간 reward = -100인 함정이 존재한다. Agent는 최단거리로 종료점까지 가는 경로를 학습한다. Policy evaluation의 수렴값과, 업데이트 되는 과정, 그리고 각각의 state에서의 action을 매트릭스로 만들어 보고서에 작성한다.

## 2. Algorithm

### a. Dynamic Programming

일반적으로 상당수의 분할 정복 알고리즘(Divide-and-Conquer)은 Top-Down 방식으로 이전에 해결한 문제를 다시 풀 수 있다는 단점을 가지고 있다. 이는 심각한 비효율성으로 이어진다. 이를 해결하기 위한 Dynamic Programming은 기본적으로 큰 문제를 작은 문제로 나누어 작은 문제에서 구한 solution을 큰 문제에 사용하는 방식은 동일하나 이 과정에서 Memoization, 즉 이미 계산한 결과는 배열에 저장하는 방법을 통해 계산을 최소화하는 알고리즘이다. 해당 알고리즘은 Top-Down 방식과 Bottom-Up 방식 모두 사용할 수 있으며 주로 Bottom-Up 방식을 활용한다.

### b. MDP & Bellman Equation

Markov Decision Process란 강화학습의 한 종류로 에이전트가 환경과 상호작용을 하며 보상을 최대화하는 방향으로 학습하는 것을 의미한다. 기본적으로 Markov Property에 의해 이전 시점에만 사건의 영향을 받는다. MDP의 목적은 보상, 즉 reward를 최대화하는 것이며 이는  $G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-t} R_T$ 로 나타낼 수 있다. 이 때  $\gamma$ 가 매우 큰 경우 학습이 더딜 수 있기 때문에 discount factor를 이용하여 미래의 reward를 조정한다.

Bellman Equation이란 value function을 정리한 것이다. Value function은 에이전트가 놓인 상태 또는 상태와 행동을 함수로 표현한 것이다.  $v_\pi(s) = \sum_a \pi(s|a) q_\pi(s, a)$ 는 상태에 따른 가치를 나타내며,  $q_\pi(s, a) = \sum P(s', a) \{R + \gamma v_\pi(s')\}$ 의 식으로 행동을 가치를 통해 나타낼 수 있다.

### c. Policy Evaluation

Policy evaluation이란 value가 어떻게 변할지를 예측(prediction)하는 것으로, 현재 주어진 policy에 대한 true value function을 구하는 것이다. 이 과정에서 Bellman equation을 사용하며 현재 Policy를 가지고 이전 value를 이용하여 구하는 one step backup을 이용한다. 현재 상태의 value function을 update하는데 reward와 next state들의 value function을 사용한다. 모든 state를 동시에 한 번씩 업데이트하는 시퀀스를 특정 횟수만큼 반복한다. 이를 무한대까지 계산하게 될 경우 현재 random policy에 대한 true value function을 구할 수 있고 이를 policy evaluation이라 칭한다.

### d. Policy Improvement

Policy improvement란 위의 Policy evaluation을 통해 얻은 true value function을 통해 더 나

은 policy로 update하는 것을 의미한다. 반복의 횟수가 늘어날수록 optimal policy에 가까워질 것이다. Improve의 방법으로 greedy improvement가 있다. 이는 다음 state 중에서 가장 높은 value function을 가진 state로 가는 것으로 max값을 취하는 것과 같다.

#### e. Value Iteration

Value iteration은 Policy iteration과 다르게 Bellman Optimality Equation을 이용한다. 이를 통해 evaluation 과정에서 이동 가능한 state의 모든 value function을 더하여 value를 평가하는 것과 다르게 이 중 max값을 통해 greedy한 방식으로 value function을 구하여 improve를 진행하는 것이다. 즉 기존 식에서 action을 취할 확률을 곱하여 summation하는 과정을 대신하여 max 값을 취하는 것으로 대체하면 된다. Policy improvement와의 차이점으로는 Policy Iteration의 경우 policy에 따라 value 값이 확률적으로 주어지지만 value iteration은 deterministic한 action으로 정해진다.

### 3. Result

#### a. Policy Evaluation

```
Untitled1.ipynb - Colaboratory
colab.research.google.com/drive/16lLZdzNQJ5BY-QWRc9p0NzPmTqWmKcQL#scrollTo=H...

+ 코드 + 텍스트
다시 연결

import numpy as np

def getState(s, a, grid_size = 7):
    A = [(-1, 0), (1, 0), (0, -1), (0, 1)]
    s[0] += A[a][0]
    s[1] += A[a][1]

    if(s[0] < 0):
        s[0] = 0
    elif(s[0] > grid_size - 1):
        s[0] = grid_size - 1

    if(s[1] < 0):
        s[1] = 0
    elif(s[1] > grid_size - 1):
        s[1] = grid_size - 1

    return s[0], s[1]

[ ] def getRewardTable(grid_size = 7):
    reward_t = np.full([grid_size, grid_size], -1)
    reward_t[0][2] = -100
    reward_t[1][2] = -100
    reward_t[grid_size - 1][2] = -100
    reward_t[grid_size - 1][3] = -100
    reward_t[3][4] = -100
    reward_t[3][5] = -100
    reward_t[grid_size - 1][grid_size - 1] = 0

    return reward_t

[ ] def policy_eval(iter, grid_size = 7):
    post_value_t = np.full([grid_size, grid_size], -1.0)
    reward_t = getRewardTable()

    for i in range(iter):
        temp = np.full([grid_size, grid_size], 0.0)
        for j in range(grid_size):
            for k in range(grid_size):
                if j == k and j == grid_size - 1:
                    temp[j][k] = 0
                else:
                    for l in range(4):
                        x, y = getState([j, k], l)
                        temp[j][k] += (reward_t[j][k] + post_value_t[x][y]) * 0.25
        post_value_t = temp
    return post_value_t

0초 오전 12:27에 완료됨
```

해당 알고리즘은 google colab 환경에서 jupyter notebook을 통해 실행한 결과를 나타낸다.

getState() 함수는 방향(a)에 따라 이동된 좌표를 반환하는 함수다. 만약 grid-world 밖으로 나가는 경우에는 멈춘 상태를 유지한다.

GetRewardTable() 함수는 grid-world의 reward table을 만드는 함수다. 기본적으로 reward는 -1로 설정되며 특정 부분에 함정에서는 -100으로 설정된다. 마지막으로 도착지의 reward는 0으로 설정한다.

Policy\_eval() 함수는 policy evaluation을 구현한 함수다. Post\_value\_t에는 grid\_size \* grid\_size 크기의 -1 value 값으로 초기화된 table을 생성한다. 그 후 이전 getRewardTable()함수를 통해 reward table을 생성한다. 입력받은 iter의 횟수만큼 반복문을 반복하게 된다. temp에는 임시로 저장할 다음 value의 값을 나타내는 table이며, 0으로 초기화한다. 그 후 각 state에 대해 다음 반복을 수행한다. 만약 현재 위치가 마지막

위치인 경우 temp의 해당 위치는 0으로 설정하며, 아닐 경우 각 방향(0(up), 1(down), 2(left), 3(right))에 대하여 현재 위치의 reward의 값과 이동한 위치의 value 값을 더하여 이를 0.25(방향이 4개이기에 1/4)를 누적 시킨다. 위의 반복이 끝나면 temp에 저장했던 table을 post\_value\_t에 update하는 과정을 반복한다. 그 후 post\_value\_t를 반환한다.

```

[3]     if(s[1] < 0):
        s[1] = 0
    elif(s[1] > grid_size - 1):
        s[1] = grid_size - 1

    return s[0], s[1]

[4] def getRewardTable(grid_size = 7):
    reward_t = np.full([grid_size, grid_size], -1)
    reward_t[0][2] = -100
    reward_t[1][2] = -100
    reward_t[grid_size - 1][2] = -100
    reward_t[grid_size - 1][3] = -100
    reward_t[3][4] = -100
    reward_t[3][5] = -100
    reward_t[grid_size - 1][grid_size - 1] = 0

    return reward_t

[5] def policy_eval(iter, grid_size = 7):
    post_value_t = np.full([grid_size, grid_size], -1.0)
    reward_t = getRewardTable()

    for i in range(iter):
        temp = np.full([grid_size, grid_size], 0.0)
        for j in range(grid_size):
            for k in range(grid_size):
                if j == k and j == grid_size - 1:
                    temp[j][k] = 0
                else:
                    for l in range(4):
                        x, y = getState([j, k], l)
                        temp[j][k] += (reward_t[j][k] + post_value_t[x][y]) * 0.25
        post_value_t = temp
    return post_value_t

[6] policy_eval(0)

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

```

위의 값은 policy evaluation에서 k=0일 때의 결과를 나타낸 것이다. 이는 기존에 초기화된 -1 table을 출력하는 정상적인 모습을 보인다.

```
Untitled1.ipynb - Colaboratory
colab.research.google.com/drive/16lLZdzNQJ5BY-QWRc9p0NzPmTqWmKcQL#scrollTo=H...

+ 코드 + 텍스트
[3] if(s[1] < 0):
    s[1] = 0
elif(s[1] > grid_size - 1):
    s[1] = grid_size - 1

return s[0], s[1]

[4] def getRewardTable(grid_size = 7):
    reward_t = np.full([grid_size, grid_size], -1)
    reward_t[0][2] = -100
    reward_t[1][2] = -100
    reward_t[grid_size - 1][2] = -100
    reward_t[grid_size - 1][3] = -100
    reward_t[3][4] = -100
    reward_t[3][5] = -100
    reward_t[grid_size - 1][grid_size - 1] = 0

    return reward_t

[5] def policy_eval(iter, grid_size = 7):
    post_value_t = np.full([grid_size, grid_size], -1.0)
    reward_t = getRewardTable()

    for i in range(iter):
        temp = np.full([grid_size, grid_size], 0.0)
        for j in range(grid_size):
            for k in range(grid_size):
                if j == k and j == grid_size - 1:
                    temp[j][k] = 0
                else:
                    for l in range(4):
                        x, y = getState([j, k], l)
                        temp[j][k] += (reward_t[j][k] + post_value_t[x][y]) * 0.25
        post_value_t = temp
    return post_value_t

policy_eval(1)

array([[ -2.,  -2., -101.,  -2.,  -2.,  -2.,  -2.],
       [ -2.,  -2., -101.,  -2.,  -2.,  -2.,  -2.],
       [ -2.,  -2.,  -2.,  -2.,  -2.,  -2.,  -2.],
       [ -2.,  -2.,  -2.,  -2., -101., -101.,  -2.],
       [ -2.,  -2.,  -2.,  -2.,  -2.,  -2.,  -2.],
       [ -2.,  -2.,  -2.,  -2.,  -2.,  -2.,  -2.],
       [ -2.,  -2., -101., -101.,  -2.,  -2.,  0.]])

[6] def policy_improv(iter, grid_size = 7):
```

K = 1로 반복을 하였을 때에는 기존에서 값이 변화된 모습을 볼 수 있다. 모두 이전의 value값을 통해 기대값을 예측한 것을 확인할 수 있으며 특히 함정의 위치의 경우 reward가 -100임으로 -101로 다른 위치에 비해 크게 나온 것을 알 수 있다. 그리고 결승점

```

elif(s[1] > grid_size - 1):
    s[1] = grid_size - 1

return s[0], s[1]

[4] def getRewardTable(grid_size = 7):
    reward_t = np.full([grid_size, grid_size], -1)
    reward_t[0][2] = -100
    reward_t[1][2] = -100
    reward_t[grid_size - 1][2] = -100
    reward_t[grid_size - 1][3] = -100
    reward_t[3][4] = -100
    reward_t[3][5] = -100
    reward_t[grid_size - 1][grid_size - 1] = 0

    return reward_t

def policy_eval(iter, grid_size = 7):
    post_value_t = np.full([grid_size, grid_size], -1.0)
    reward_t = getRewardTable()

    for i in range(iter):
        temp = np.full([grid_size, grid_size], 0.0)
        for j in range(grid_size):
            for k in range(grid_size):
                if j == k and j == grid_size - 1:
                    temp[j][k] = 0
                else:
                    for l in range(4):
                        x, y = getState([j, k], l)
                        temp[j][k] += (reward_t[j][k] + post_value_t[x][y]) * 0.25
        post_value_t = temp
    return post_value_t

policy_eval(2)

array([[ -3. , -27.75, -151.5 , -27.75,  -3. ,  -3. ,  -3. ],
       [ -3. , -27.75, -126.75, -27.75,  -3. ,  -3. ,  -3. ],
       [ -3. ,  -3. , -27.75,  -3. , -27.75, -27.75,  -3. ],
       [ -3. ,  -3. ,  -3. , -27.75, -126.75, -126.75, -27.75],
       [ -3. ,  -3. ,  -3. ,  -3. , -27.75, -27.75,  -3. ],
       [ -3. ,  -3. , -27.75, -27.75,  -3. ,  -3. , -2.5 ],
       [ -3. , -27.75, -151.5 , -151.5 , -27.75, -2.5 ,  0. ]])

[6] def policy_improv(iter, grid_size = 7):
    post_value_t = np.full([grid_size, grid_size], -1.0)
    action_t = np.full([grid_size, grid_size, 4], 0.25, dtype=np.float64)

```

$k=2$ 일 때 결과는 다음과 같다. (0,1)의 값을 분석해 보았을 때  $3 * 0.25 * (-1 - 2) + 0.25 * (-1 - 101) = 2.25 + 25.5 = 27.75$ 로 직접 계산했을 때의 결과와 일치하는 것을 확인할 수 있다. 이를 통해 해당 결과가 정상적으로 나타나는 것을 알 수 있다.

```
Untitled1.ipynb - Colaboratory
colab.research.google.com/drive/16lLZdzNQJ5BY-QWRc9p0NzPmTqWmKcQL#scrollTo=H...

+ 코드 + 텍스트
[4] def getRewardTable(grid_size = 7):
    reward_t = np.full([grid_size, grid_size], -1)
    reward_t[0][2] = -100
    reward_t[1][2] = -100
    reward_t[grid_size - 1][2] = -100
    reward_t[grid_size - 1][3] = -100
    reward_t[3][4] = -100
    reward_t[3][5] = -100
    reward_t[grid_size - 1][grid_size - 1] = 0

    return reward_t

[5] def policy_eval(iter, grid_size = 7):
    post_value_t = np.full([grid_size, grid_size], -1.0)
    reward_t = getRewardTable()

    for i in range(iter):
        temp = np.full([grid_size, grid_size], 0.0)
        for j in range(grid_size):
            for k in range(grid_size):
                if j == k and j == grid_size - 1:
                    temp[j][k] = 0
                else:
                    for l in range(4):
                        x, y = getState([j, k], l)
                        temp[j][k] += (reward_t[j][k] + post_value_t[x][y]) * 0.25
        post_value_t = temp
    return post_value_t

policy_eval(3)

array([[ -10.1875,  -53.5    , -183.4375,  -53.5    ,  -10.1875,  -4.    ,
        -4.    ],
       [ -10.1875,  -41.125 , -158.6875,  -41.125 ,  -16.375 ,  -10.1875,
        -4.    ],
       [  -4.    ,  -16.375 ,  -34.9375,  -28.75   ,  -41.125 ,  -41.125 ,
       -16.375 ],
       [  -4.    ,  -4.    ,  -16.375 ,  -34.9375, -152.5    , -152.5    ,
       -41.125 ],
       [  -4.    ,  -4.    ,  -10.1875,  -22.5625,  -41.125 ,  -41.125 ,
       -16.25  ],
       [  -4.    ,  -16.375 ,  -47.3125,  -47.3125,  -22.5625,  -9.9375,
       -3.125  ],
       [ -10.1875,  -47.3125, -189.625 , -189.625 ,  -47.1875,  -9.3125,
         0.    ]])

[6] def policy_improv(iter, grid_size = 7):
    post_value_t = np.full([grid_size, grid_size], -1.0)
```

해당 값은  $k = 3$ 일 때의 결과를 나타낸다. 이전보다 값이 증가된 것을 볼 수 있으며 goal에 가깝거나 함정에서 멀수록 value값이 작아지는 것을 확인할 수 있다.



Untitled1.ipynb - Colaboratory 새 탭

colab.research.google.com/drive/16ILZdzNQJ5BY-QWRc9p0NzPmTqWmKcQL#scrollTo=H...

+ 코드 + 텍스트

RAM 디스크

0초

[4] def getRewardTable(grid\_size = 7):  
 reward\_t = np.full([grid\_size, grid\_size], -1)  
 reward\_t[0][2] = -100  
 reward\_t[1][2] = -100  
 reward\_t[grid\_size - 1][2] = -100  
 reward\_t[grid\_size - 1][3] = -100  
 reward\_t[3][4] = -100  
 reward\_t[3][5] = -100  
 reward\_t[grid\_size - 1][grid\_size - 1] = 0  
  
 return reward\_t

0초

[5] def policy\_eval(iter, grid\_size = 7):  
 post\_value\_t = np.full([grid\_size, grid\_size], -1.0)  
 reward\_t = getRewardTable()  
  
 for i in range(iter):  
 temp = np.full([grid\_size, grid\_size], 0.0)  
 for j in range(grid\_size):  
 for k in range(grid\_size):  
 if j == k and j == grid\_size - 1:  
 temp[j][k] = 0  
 else:  
 for l in range(4):  
 x, y = getState([j, k], l)  
 temp[j][k] += (reward\_t[j][k] + post\_value\_t[x][y]) \* 0.25  
 post\_value\_t = temp  
 return post\_value\_t

3초

▶ policy\_eval(3000)

array([[ -3337.59452264, -3381.21532552, -3482.45959185, -3268.32013372,  
 -3114.22840277, -3006.49629939, -2947.59890753],  
 [-3289.97378759, -3319.59192929, -3397.84338236, -3204.27247113,  
 -3063.86883823, -2953.6616496 , -2884.70157665],  
 [-3208.73497805, -3205.33528821, -3185.04960235, -3083.05759368,  
 -2979.31289106, -2855.57994417, -2748.84423191],  
 [-3126.89592441, -3103.96470832, -3049.96220872, -2959.59547152,  
 -2910.74524699, -2736.50106072, -2502.25123016],  
 [-3063.98815145, -3029.66547544, -2947.23911399, -2790.61689499,  
 -2567.57161941, -2277.42787287, -2017.40844679],  
 [-3031.40311752, -2999.46998962, -2914.71193568, -2684.06142979,  
 -2287.49651233, -1784.23040848, -1268.54627682],  
 [-3026.75127321, -3018.09948988, -3024.07726592, -2739.42042744,  
 -2110.12263557, -1299.45100641, 0. ]])

<>

0초

[6] def policy\_improv(iter, grid\_size = 7):  
 post\_value\_t = np.full([grid\_size, grid\_size], -1.0)

3초 오전 2:28에 완료됨

```
Untitled1.ipynb - Colaboratory
colab.research.google.com/drive/16lLZdzNQJ5BY-QWRc9p0NzPmTqWmKcQL#scrollTo=H...

+ 코드 + 텍스트
[4] def getRewardTable(grid_size = 7):
    reward_t = np.full([grid_size, grid_size], -1)
    reward_t[0][2] = -100
    reward_t[1][2] = -100
    reward_t[grid_size - 1][2] = -100
    reward_t[grid_size - 1][3] = -100
    reward_t[3][4] = -100
    reward_t[3][5] = -100
    reward_t[grid_size - 1][grid_size - 1] = 0

    return reward_t

[5] def policy_eval(iter, grid_size = 7):
    post_value_t = np.full([grid_size, grid_size], -1.0)
    reward_t = getRewardTable()

    for i in range(iter):
        temp = np.full([grid_size, grid_size], 0.0)
        for j in range(grid_size):
            for k in range(grid_size):
                if j == k and j == grid_size - 1:
                    temp[j][k] = 0
                else:
                    for l in range(4):
                        x, y = getState([j, k], l)
                        temp[j][k] += (reward_t[j][k] + post_value_t[x][y]) * 0.25
        post_value_t = temp
    return post_value_t

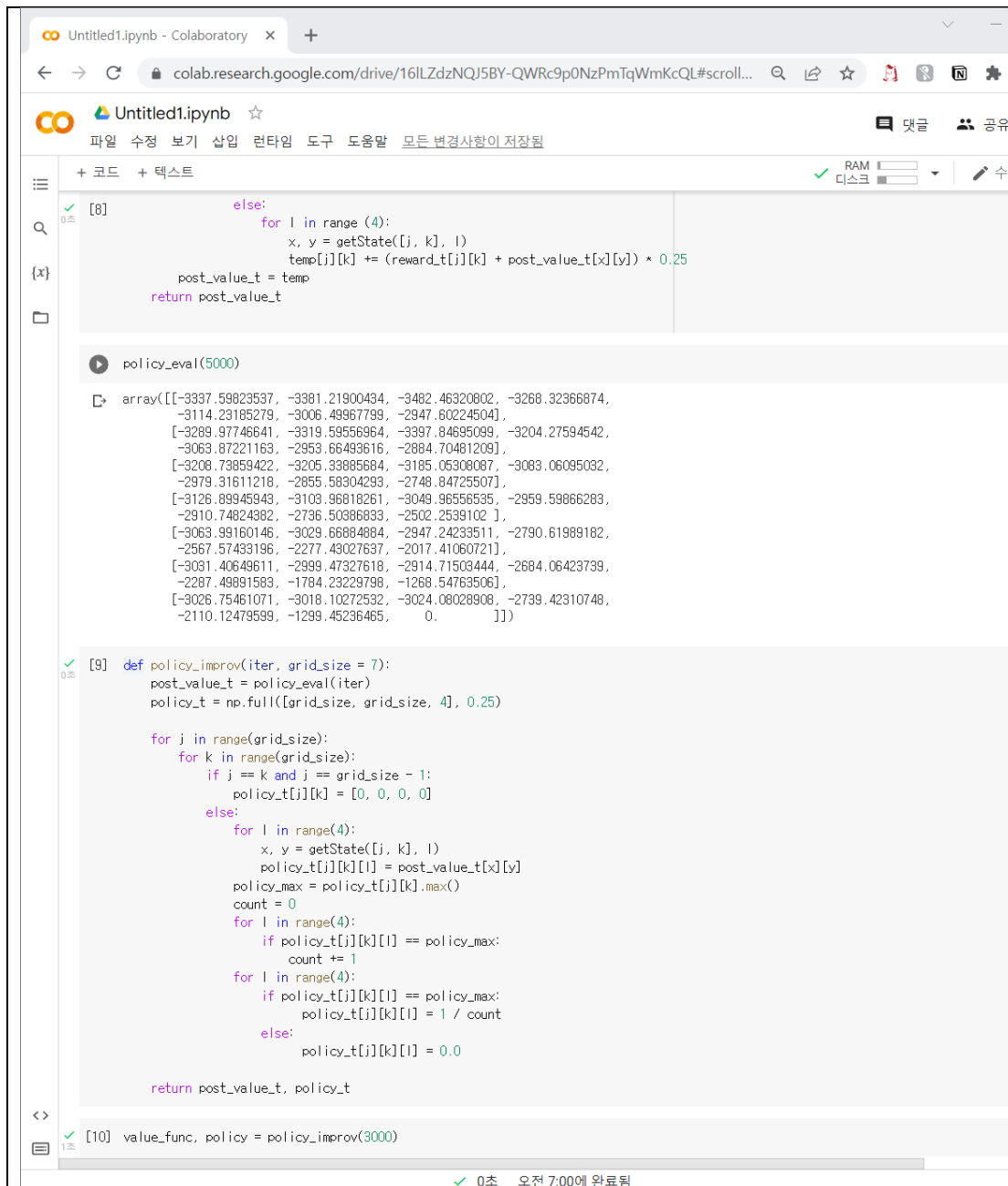
policy_eval(5000)

array([[ -3337.59823537, -3381.21900434, -3482.46320802, -3268.32366874,
        -3114.23185279, -3006.49967799, -2947.60224504],
       [-3289.97746641, -3319.59556964, -3397.84695099, -3204.27594542,
        -3063.87221163, -2953.66493616, -2884.70481209],
       [-3208.73859422, -3205.33885684, -3185.05308087, -3083.06095032,
        -2979.31611218, -2855.58304293, -2748.84725507],
       [-3126.89945943, -3103.96818261, -3049.96556535, -2959.59866283,
        -2910.74824382, -2736.50386833, -2502.2539102 ],
       [-3063.99160146, -3029.66884884, -2947.24233511, -2790.61989182,
        -2567.57433196, -2277.43027637, -2017.41060721],
       [-3031.40649611, -2999.47327618, -2914.71503444, -2684.06423739,
        -2287.49891583, -1784.23229798, -1268.54763506],
       [-3026.75461071, -3018.10272532, -3024.08028908, -2739.42310748,
        -2110.12479599, -1299.45236465,  0.          ]])

[6] def policy_improv(iter, grid_size = 7):
    post_value_t = np.full([grid_size, grid_size], -1.0)
```

위의 결과는 값이 약 3000 이상으로 커지게 되었을 때 결과가 거의 변하지 않는다는 것을 보인다. 즉 이는 policy evaluation을 통해 구한 true value function임을 알 수 있다.

## b. Policy Improvement



```
[8] else:
    for l in range(4):
        x, y = getState([j, k], l)
        temp[j][k] += (reward_t[j][k] + post_value_t[x][y]) * 0.25
    post_value_t = temp
    return post_value_t

policy_eval(5000)

array([[ -3337.59823537, -3381.21900434, -3482.46320802, -3268.32366874,
        -3114.23185279, -3006.49967799, -2947.60224504],
       [-3289.97746641, -3319.59556964, -3397.84695099, -3204.27594542,
        -3063.87221163, -2953.66493616, -2884.70481209],
       [-3208.73859422, -3205.33885684, -3185.05308087, -3083.06095092,
        -2979.31611218, -2855.58304293, -2748.84725507],
       [-3126.89945943, -3108.96818261, -3049.96556535, -2959.59866283,
        -2910.74824382, -2736.50986833, -2502.2539102 ],
       [-3063.99160146, -3029.66884884, -2947.24233511, -2790.61989182,
        -2567.57433196, -2277.43027637, -2017.41060721],
       [-3031.40649611, -2999.47327618, -2914.71503444, -2684.06423739,
        -2287.49891583, -1784.23229798, -1268.54763506],
       [-3026.75461071, -3018.10272532, -3024.08028908, -2739.42310748,
        -2110.12479599, -1299.45236465,  0.          ]])

[9] def policy_improv(iter, grid_size = 7):
    post_value_t = policy_eval(iter)
    policy_t = np.full([grid_size, grid_size, 4], 0.25)

    for j in range(grid_size):
        for k in range(grid_size):
            if j == k and j == grid_size - 1:
                policy_t[j][k] = [0, 0, 0, 0]
            else:
                for l in range(4):
                    x, y = getState([j, k], l)
                    policy_t[j][k][l] = post_value_t[x][y]
                policy_max = policy_t[j][k].max()
                count = 0
                for l in range(4):
                    if policy_t[j][k][l] == policy_max:
                        count += 1
                for l in range(4):
                    if policy_t[j][k][l] == policy_max:
                        policy_t[j][k][l] = 1 / count
                else:
                    policy_t[j][k][l] = 0.0

    return post_value_t, policy_t

[10] value_func, policy = policy_improv(3000)
```

다음 `policy_improv()` 함수는 `policy_evaluation`을 통해 구한 true function을 이용하여 `policy_improvement`를 구하는 함수다. 각 state에 대하여 도착지의 경우 policy를 0으로 설정하며, 도착지가 아닌 경우 max값의 수를 확인하여 이를 확률로 표기하는 greedy한 방법을 확인한다.





```
Untitled1.ipynb - Colaboratory
colab.research.google.com/drive/16LZdzNQJ5BY-QWRc9p0NzPmTqWmKcQL#scroll...

+ 코드 + 텍스트
[24] value_func
array([[ -3. ,  -27.75, -151.5 ,  -27.75,  -3. ,  -3. ,  -3. ],
       [ -3. ,  -27.75, -126.75,  -27.75,  -3. ,  -3. ,  -3. ],
       [ -3. ,   -3. ,  -27.75,   -3. ,  -27.75,  -27.75,  -3. ],
       [ -3. ,   -3. ,   -3. ,  -27.75, -126.75, -126.75, -27.75],
       [ -3. ,   -3. ,   -3. ,   -3. ,  -27.75,  -27.75,  -3. ],
       [ -3. ,   -3. ,  -27.75,  -27.75,  -3. ,   -3. ,  -2.5 ],
       [ -3. ,  -27.75, -151.5 , -151.5 ,  -27.75,  -2.5 ,   0. ]])

[25] for i in range(7):
      for j in range(7):
          print("U : {}, D : {}, L : {}, R : {}".format(policy[i][j][0], policy[i][j][1], policy[i][j][2], policy[i][j][3]), end = '\n')
          print("\n", end = '')

U : 0.3333333333333333, D : 0.3333333333333333, L : 0.3333333333333333, R : 0.0], [U : 0.0, D : 0.0, L : 1.0, R : 0.0], [U : 0.0
U : 0.3333333333333333, D : 0.3333333333333333, L : 0.3333333333333333, R : 0.0], [U : 0.0, D : 0.5, L : 0.5, R : 0.0], [U : 0.0
U : 0.25, D : 0.25, L : 0.25, R : 0.25], [U : 0.0, D : 0.5, L : 0.5, R : 0.0], [U : 0.0, D : 0.3333333333333333, L : 0.333333333
U : 0.25, D : 0.25, L : 0.25, R : 0.25], [U : 0.25, D : 0.25, L : 0.25, R : 0.25], [U : 0.0, D : 0.5, L : 0.5, R : 0.0], [U : 0.
U : 0.25, D : 0.25, L : 0.25, R : 0.25], [U : 0.25, D : 0.25, L : 0.25, R : 0.25], [U : 0.3333333333333333, D : 0.0, L : 0.33333
U : 0.25, D : 0.25, L : 0.25, R : 0.25], [U : 0.5, D : 0.0, L : 0.5, R : 0.0], [U : 0.5, D : 0.0, L : 0.5, R : 0.0], [U : 0.5, D
U : 0.3333333333333333, D : 0.3333333333333333, L : 0.3333333333333333, R : 0.0], [U : 0.5, D : 0.0, L : 0.5, R : 0.0], [U : 0.5

print("action : ")
for i in range(7):
    for j in range(7):
        print("[", end = '')
        if(policy[i][j][0] == policy[i][j].max() and [i, j] != [6, 6]):
            print("U, ", end = '')
        else:
            print("X, ", end = '')
        if(policy[i][j][1] == policy[i][j].max() and [i, j] != [6, 6]):
            print("D, ", end = '')
        else:
            print("X, ", end = '')
        if(policy[i][j][2] == policy[i][j].max() and [i, j] != [6, 6]):
            print("L, ", end = '')
        else:
            print("X, ", end = '')
        if(policy[i][j][3] == policy[i][j].max() and [i, j] != [6, 6]):
            print("R, ", end = '')
        else:
            print("X, ", end = '')
        print("]", end = '')

action :
[U, D, L, X], [X, X, L, X], [X, X, L, R], [X, X, X, R], [U, D, X, R], [U, D, L, R], [U, D, L, R],
[U, D, L, X], [X, D, L, X], [X, D, L, R], [X, D, X, R], [U, X, L, R], [U, X, L, R], [U, D, L, R],
[U, D, L, R], [X, D, L, X], [X, D, L, R], [U, D, L, R], [U, X, L, X], [U, X, X, R], [U, X, X, R],
[U, D, L, R], [U, D, L, R], [X, D, L, X], [U, D, L, X], [U, D, L, X], [U, D, X, R], [U, D, X, X],
[U, D, L, R], [U, D, L, R], [U, X, L, R], [X, X, L, X], [X, D, L, X], [X, D, X, R], [X, D, X, X],
[U, D, L, R], [U, X, L, X], [U, X, L, X], [U, X, X, R], [X, X, X, R], [X, D, X, R], [X, D, X, X],
[U, D, L, X], [U, X, L, X], [U, X, L, X], [U, X, X, R], [X, X, X, R], [X, X, X, R], [X, X, X, X],
```

다음 사진은  $k = 2$ 일 때를 나타낸다. 이 때 (0, 1) 위치를 보았을 때 왼쪽의 값이 가장 크기 때문에 greedy에서 왼쪽으로 이동하는 확률만 남았으며, 이는 action에도 반영된 것을 확인할 수 있다.

다음 결과는  $k = 3$ 일 때로 반복이 진행됨의 따라 policy가 optimize되고 있는 과정을 눈으로 확인할 수 있다.





```

    print("\n", end = '')
    else:
        print("X, ", end = '')
        if(action[i][j][2] == action[i][j].max() and [i, j] != [6, 6]):
            print("L, ", end = '')
        else:
            print("X, ", end = '')
            if(action[i][j][3] == action[i][j].max() and [i, j] != [6, 6]):
                print("R, ", end = '')
            else:
                print("X, ", end = '')
        print("\n", end = '')

    [X, D, X, X], [X, D, X, X], [X, X, X, R], [X, X, X, R], [X, X, X, R], [X, X, X, R], [X, D, X, X],
    [X, D, X, X], [X, D, X, X], [X, D, X, X], [X, X, X, R], [X, X, X, R], [X, D, X, X], [X, D, X, X],
    [X, D, X, X], [X, D, X, X], [X, D, X, X], [X, D, X, X], [X, X, X, R], [X, D, X, X], [X, D, X, X],
    [X, D, X, X], [X, D, X, X], [X, D, X, X], [X, D, X, X], [X, D, X, X], [X, D, X, X], [X, D, X, X],
    [X, X, X, R], [X, X, X, R], [X, X, X, R], [X, X, X, R], [X, X, X, R], [X, X, X, R], [X, D, X, X],
    [X, X, X, R], [X, X, X, R], [X, X, X, R], [X, X, X, R], [X, X, X, R], [X, X, X, R], [X, D, X, X],
    [X, X, X, R], [U, X, X, X], [X, X, X, R], [X, X, X, R], [X, X, X, R], [X, X, X, R], [X, X, X, X],

[120] def value_iter(iter, grid_size = 7):
    post_value_t = np.full([grid_size, grid_size], -1.0)
    action_t = np.full([grid_size, grid_size, 4], 1)
    action_t[grid_size - 1][grid_size - 1] = [0, 0, 0, 0]
    reward_t = getRewardTable()

    for i in range(iter):
        temp = np.full([grid_size, grid_size], 0.0)
        for j in range(grid_size):
            for k in range(grid_size):
                if j == k and j == grid_size - 1:
                    temp[j][k] = 0.0
                    action_t[j][k] = [0.0, 0.0, 0.0, 0.0]
                else:
                    for l in range(4):
                        x, y = getState([j, k], l)
                        action_t[j][k][l] = reward_t[x][y] + post_value_t[x][y]
                    temp[j][k] = action_t[j][k].max()
                    for l in range(4):
                        if(action_t[j][k][l] == temp[j][k]):
                            action_t[j][k][l] = 1
                        else:
                            action_t[j][k][l] = 0
                    post_value_t = temp
    return post_value_t, action_t

[121] value_func, action = value_iter(0)

```

Value\_iter() 함수는 value iteration을 구현한 함수다. 이전의 함수와 마찬가지로 post\_value\_t와 action\_t, reward\_t를 초기화한 후 i횟수만큼 반복을 수행한다. 각 state에 대하여 종료지점일 경우 temp와 action을 0으로 설정하며, 이외의 경우에 대해서는 action\_t에 다음 state의 기댓값을 저장한다. 이후 이 중 가장 큰 기대값만을 temp에 저장한 후 temp와 같은 action\_t 값을 1로, 이외의 경우를 0으로 설정한다.









```

[149] value_func, action = value_iter(13)

[150] value_func
array([[ -12.,  -11., -109.,  -8.,  -8.,  -7.,  -6.],
       [ -11., -10., -108.,  -8.,  -7.,  -6.,  -5.],
       [ -10.,  -9.,  -8.,  -7.,  -6.,  -5.,  -4.],
       [  -9.,  -8.,  -7.,  -6., -104., -103.,  -3.],
       [  -8.,  -7.,  -6.,  -5.,  -4.,  -3.,  -2.],
       [  -7.,  -6.,  -5.,  -4.,  -3.,  -2.,  -1.],
       [  -8.,  -7., -105., -102.,  -2.,  -1.,   0.]])

[151] for i in range(7):
    for j in range(7):
        print("U : {}, D : {}, L : {}, R : {}".format(round(action[i][j][0], 3), round(action[i][j][1], 3), round(action[i][j][2], 3), round(action[i][j][3], 3)), end = '\n')

[152] for i in range(7):
    for j in range(7):
        print("U, ", end = '')
        if(action[i][j][0] == 1 and [i, j] != [6, 6]):
            print("D, ", end = '')
        else:
            print("X, ", end = '')
        if(action[i][j][1] == 1 and [i, j] != [6, 6]):
            print("L, ", end = '')
        else:
            print("X, ", end = '')
        if(action[i][j][2] == 1 and [i, j] != [6, 6]):
            print("R, ", end = '')
        else:
            print("X, ", end = '')
        print("\n", end = '')

[153] [X, D, X, R], [X, D, X, X], [X, X, X, R], [X, D, X, R], [X, D, X, R], [X, D, X, R], [X, D, X, X],
[X, D, X, R], [X, D, X, X], [X, D, X, R], [X, D, X, R], [X, D, X, R], [X, D, X, R], [X, D, X, X],
[X, D, X, R], [X, D, X, R], [X, D, X, R], [X, D, X, R], [X, X, X, R], [X, X, X, R], [X, D, X, X],
[X, D, X, R], [X, D, X, R], [X, D, X, R], [X, D, X, X], [X, D, X, X], [X, D, X, R], [X, D, X, X],
[X, D, X, R], [X, D, X, R], [X, D, X, R], [X, D, X, R], [X, D, X, R], [X, D, X, R], [X, D, X, X],
[X, X, X, R], [X, X, X, R], [X, X, X, R], [X, X, X, R], [X, X, X, R], [X, D, X, R], [X, D, X, X],
[U, X, X, R], [U, X, X, X], [U, X, X, X], [X, X, X, R], [X, X, X, R], [X, X, X, R], [X, X, X, X]]

```

다음은 k=13으로 하였을 때의 결과로 value 값이 수렴한 case다. 해당 반복 이상으로 확인하더라도 같은 결과가 나온다. Value를 확인하였을 때 함정을 제외하고 도착지에 가까울수록 value가 작아지며 이에 따라 방향도 value가 작은 방향으로 결정되었다. 해당 방식에서는 이전과 달리 방향이 2개정도로 결정되는 경우도 많이 보이는 모습을 보였다. 또한 start에서 이동을 추적할 시 도착지로 가는 것을 확인하였다. 이를 통해 해당 알고리즘이 제대로 구현되었음을 확인하였다.

## 4. Consideration

해당 과제를 통해 Dynamic Programming을 사용하여 강화학습을 구현할 수 있다는 점을 알 수 있었으며, 이전의 value 1 step만을 저장하면 된다는 사실을 알았다. 특히 Markov Decision Process와 Bellman Equation의 식에 대해 이해도를 높일 수 있었다. 그리고 Policy

Evaluation을 통해 직접 value가 어떻게 변하는지 눈으로 확인할 수 있었으며, Policy improvement와 Value iteration을 직접 구현하여 비교함으로써, 두 모델의 세부적인 차이를 이해할 수 있었다. 마지막으로 이를 통해 시험 준비를 같이 할 수 있었던 과제였다.