AI 응용 12차시 강화학습 정리 by 송주훈

목차

<1. 강화학습이란>

강화학습: 지도 학습 / 비지도 학습과 다르게 시행착오와 reward를 통해 목표를 찾아가는 기계학습. 가장 유명한 예시로는 알파고가 있다. 보상을 최대화하는 쪽으로 학습하는 방식이라고 생각하면 된다.

<2. 강화학습 용어 정리>

state (s)

-현재의 상황을 나타냄

agent

-행동을 하는 주체 혹은 알고리즘

action (a)

- 에이전트가 선택할 수 있는 의사결정의 단위

reward (r)

- 에이전트가 행동을 취한 후 환경에서 받는 피드백 (점수). 즉각적임

penalty

- 행동이 잘못되었을 때 주는 처벌 (중요한 용어는 아님)

environment

- 에이전트와 상호 작용하는 시스템. 다양한 방식으로 에이전트의 행동에 반응

observation

- 에이전트가 환경에서 가져오는 정보. state의 부분집합 혹은 같은걸로 본다고 하심.

transition probability (P)

- 특정 행동을 취했을 때 다른 상태로 전이될 확률

return(gain) (강의노트에는 G로 나와있음)

- 총 얻는 reward의 합

policy (π)

- agent가 현재 상태에 대해 선택하는 행동의 전략

deterministic policy

- 결정적으로 활동을 고르는 전략 (확률적이 아니고 특정 상태에서 특정 행동을 정함)

stochastic policy

- 확률적으로 활동을 고르는 전략

value function

- 특정 정책에 대해 해당 상태에서 시작하여 기대되는 총 보상의 기대값

state value function (v)

- 각 상태 s에서 특정 정책을 따라 행동했을 때 얻는 보상의 기대값

action value function (q)

- 상태 s에서 특정 행동 s를 취했을 때 그 이후에 정책 π에 의해 얻을 총 보상의 기대값

<3. Markov 마르코프>

마르코프 상태

- 어떤 시스템이 있을 때 현재 시스템이 어떤 상태에 있는 지 나타내는 것. 현재 상태만으로 앞으로 일어날 상황이 결정되는건 아니지만 현재 상태가 중요한 것임. 현재 상태만으로 미래의 상황을 예측.

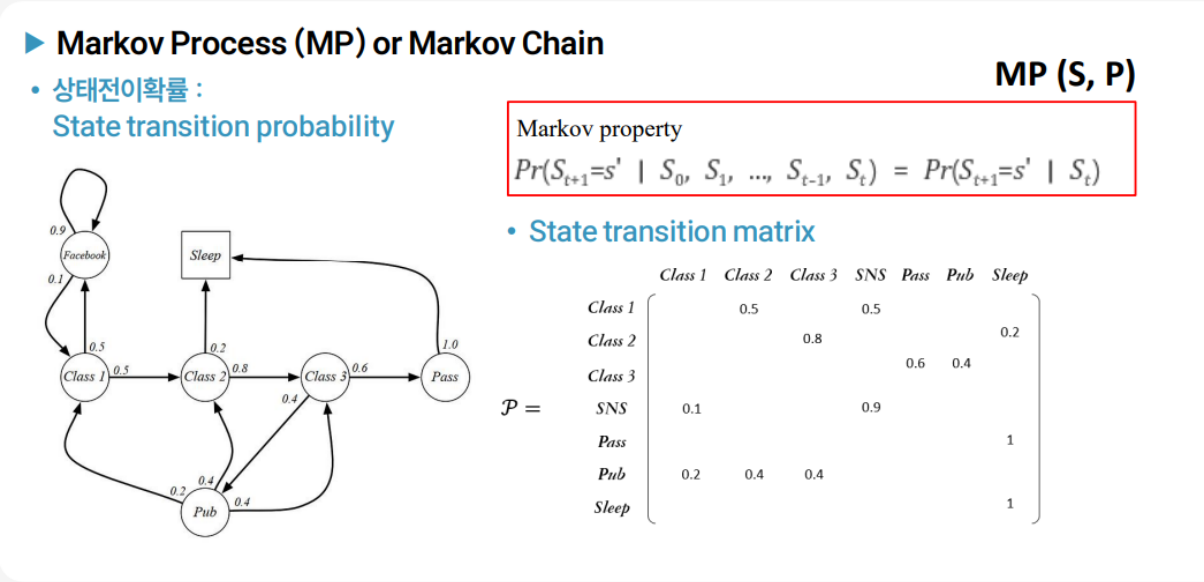
간단한 예시)

1.일반 초등학교 -> 일반 중학교 -> 일반 고등학교 -> 서울대 졸업

2.명문 초등학교 -> 영재 중학교 -> 과학 고등학교 -> 탈선으로 대학안가고 25세 양아치

누가 취업을 잘할까? 물론 과고에서 엄청난 커리어를 쌓았을 수는 있지만 마르코프 상태에선 현재 상태만 가지고 판단하므로 1번이 취업을 더 잘할 것.

마르코프 프로세스



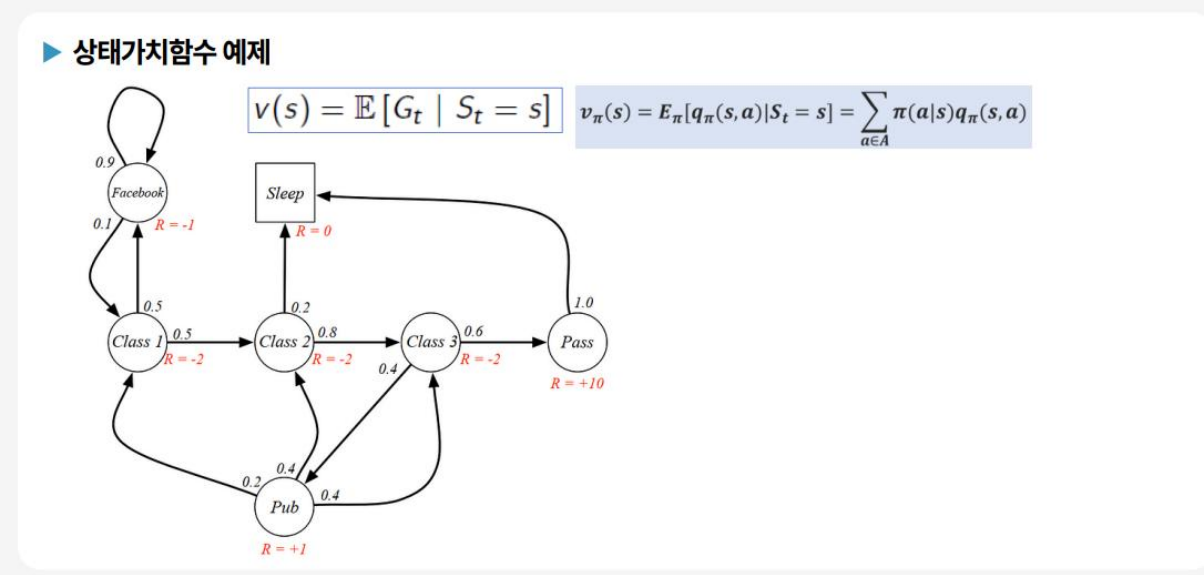
State transition matrix를 통해 현재 상태에서 다른 상태로 전이 확률이 나타남

간단한 예시)

침대, 책상만 있는 고시원에서 생활한다고 상상. 침대->책상, 책상->침대만 반복되고 이 과정에서 현재 상태를 알면 다음 상태가 어떻게 될 지 예측 가능. 이 때 과거의 경로는 중요하지 않음

마르코프 디시전 프로세스

마르코프 프로세스 + 결정적인 행동. 각 상태에서 어떤 행동을 취할 지 결정할 수 있는 과정이 있음. 이에 따른 보상을 추가한 형태라고 생각하면 됨.



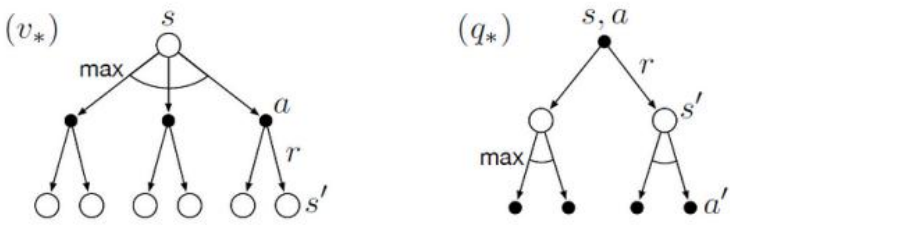
여기에는 전과 다르게 행동을 고르는 정책과 reward가 추가된 형태임을 확인 가능

<4. 벨만 방정식>

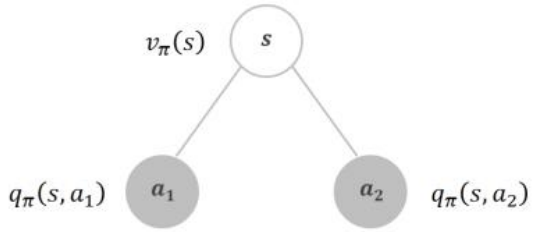
벨만 방정식을 쓰는 이유를 알아야 함

➔ 식을 state value fuction 아니면 action value function으로만 정리하여 계산하기 편하려고

➔ 재귀적으로 계산이 가능해짐



Backup diagrams for v_* and q_*



$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s, a)$$

v 는 q와 정책의 곱들의 합으로 이루어져있음 -> 이걸 v만으로 정리가 가능!

$$\begin{aligned} v(s) &= \mathbb{E}\left[G_t \mid S_t = s\right] \\ &= \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s\right] \\ &= \mathbb{E}\left[R_{t+1} + \gamma\left(R_{t+2} + \gamma R_{t+3} + \ldots\right) \mid S_t = s\right] \\ &= \mathbb{E}\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \\ &= \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s\right] \end{aligned}$$

▶ **상태가치 함수 (State value function) 와 policy**

$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \quad \text{MC method} \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \quad \text{TD method} \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)\Big[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']\Big] \\
&= \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a)\Big[r + \gamma v_\pi(s')\Big], \quad \text{for all } s \in \mathcal{S},
\end{aligned}
$$

$$
Q_\pi(s, a) = q_\pi(s, a) = \sum_{s', r} p(s', r|s, a)[r + \gamma V_\pi(s')]
$$

벨만 최적 방정식

$$
v_*(s) = \max_\pi v_\pi(s)
$$

$$
q_*(s, a) = \max_\pi q_\pi(s, a)
$$

벨만 최적화 방정식을 통해 최적의 정책을 찾는 것이 학습의 목표 (물론 정책 없이 하는 강화학습도 있는데 12차시 수업에서는 정책 기반으로 설명하기 때문에 일단을 이렇게 생각합시다)

<5. 다이나믹 프로그래밍>

정책을 최적화하는데에 다이나믹 프로그래밍을 씁니다

정책 반복 vs 가치 반복이 있는데 수업했던 정책 반복까지 설명하겠습니다

- Represent the problem as an MDP
- $< S, A, P, R, \gamma >$
  - $S = \{1,2, \dots, 14, T\}$
  - $A = \{n, e, s, w\}$
  - $P^a_{ss'}$
    - $P^n_{11} = 1.0,\ P^e_{12} = 1.0,\ P^s_{15} = 1.0,\ P^w_{17} = 1.0$
    - ...
  - $R^a_{ss'} = -1$
  - γ=1
  - $\pi(A_t = a \mid S_t = s) = 0.25$ s.t. $s \in S, a \in A$

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

해당 문제에서 회색 칸으로 가면 Goal이라고 생각해봅시다. 현재 정책은 상하좌우를 각각 같은 확률로 선택하는 것입니다. k번 반복하며 업데이트를 다음과 같이 합니다.

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

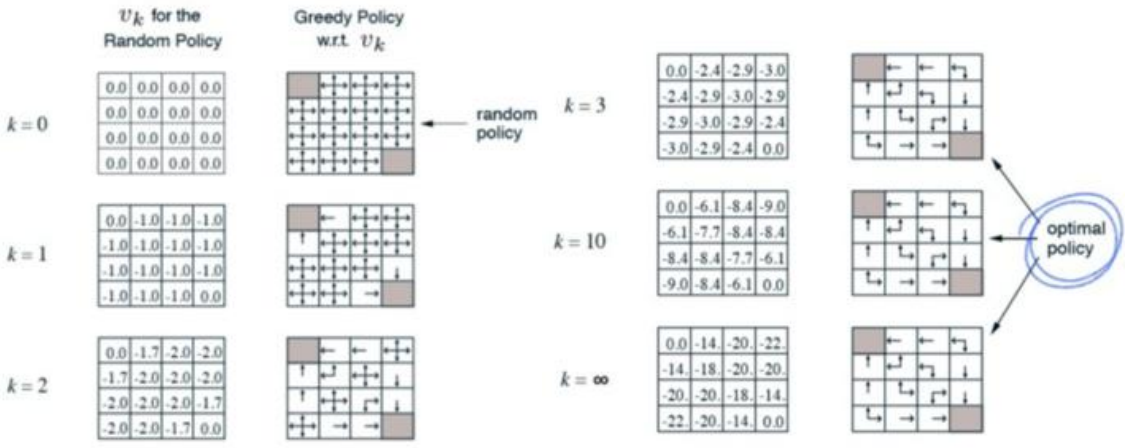
0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

14번만에 goal에 간다는 의미

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

이를 통해 각 반복 수마다 각 칸(상태)에서 v를 계산 및 확인할 수 있습니다.



얻은 v를 통해 greedy하게 정책을 선택하면 최적의 방법으로 목표 지점에 갈 수 있습니다.

$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

$$\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, q_\pi(s, a)$$

greedy

이처럼 반복하여 최선의 정책을 찾는 것입니다.

Frozen Lake 코드를 통해 확인해보겠습니다.

가치 평가 (Policy evaluation)

```
## Policy evaluation
## deterministic world : is_slippery =  False
env = gym.make("FrozenLake-v1", desc = None,
               map_name="4x4", is_slippery = False)

num_states = env.observation_space.n # 16
num_actions = env.action_space.n
transitions = env.unwrapped.P # (probability_1, next_state_1, reward_1, is_terminal_1)
print("num_states = ", num_states)
print("num_actions = ", num_actions)
print("="*50)
print('transitions = \n' )
transitions
```

[10] ✓ 0.0s

```
num_states =  16
num_actions =  4
==================================================
transitions =


{0: {0: [(1.0, 0, 0.0, False)],
  1: [(1.0, 4, 0.0, False)],
  2: [(1.0, 1, 0.0, False)],
  3: [(1.0, 0, 0.0, False)]},
 1: {0: [(1.0, 0, 0.0, False)],
  1: [(1.0, 5, 0.0, True)],
  2: [(1.0, 2, 0.0, False)],
  3: [(1.0, 1, 0.0, False)]},
```

각각의 상태, 행동의 수를 확인하고 transition 확률을 확인합니다.

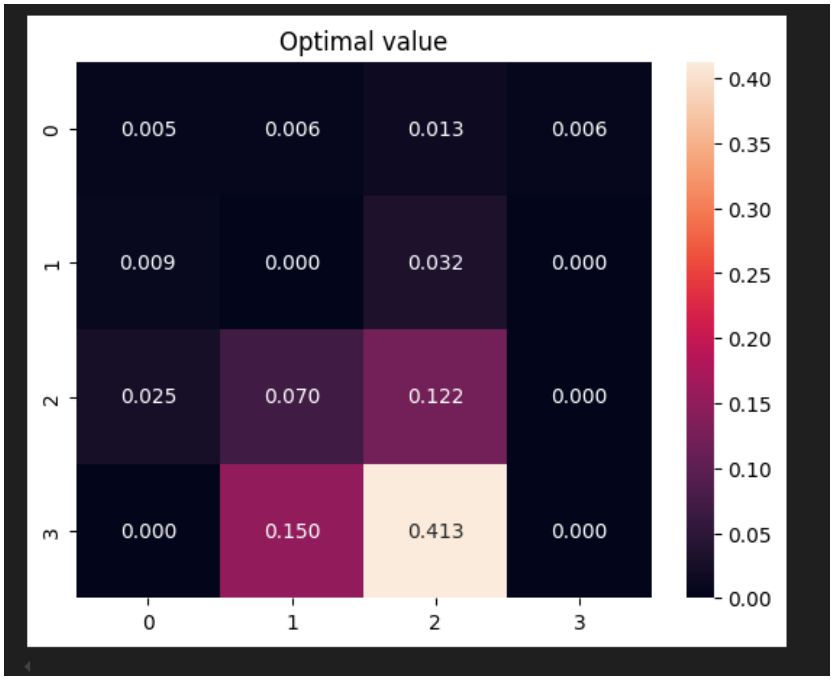
```
V = np.zeros(num_states)
pi = np.ones([num_states, num_actions])*0.25
```

현재 모든 칸의 v는 0, 각 행동을 할 확률이 1/4인 정책입니다.

```
gamma = 0.95
theta = 1e-3
count = 0

while True:
    delta = 0
    count += 1
    for s in range(num_states):
        old_value = V[s]
        new_value = 0

        for a, prob_action in enumerate(pi[s]): # [(0, 0.25), (1, 0.25), (2, 0.25), (3, 0.25)]
            for prob_environ, s_, reward, terminated in transitions[s][a]:
                new_value += prob_action*prob_environ*(reward + gamma*V[s_])
        V[s] = new_value

        delta = max(delta, np.abs(old_value - V[s]))
        # print(f"V({count}) = ", V)
        # time.sleep(0.5)

    if delta <= theta: # 수렴조건
        break
```

모든 상태 s에 대해서 v를 업데이트 합니다. 또한 해당 상태에서 할 수 있는 모든 행동을 하고 이에 따른 value를 구하고 업데이트 해줍니다.

prob_environ같은 경우엔 1인데, 다른 강화학습 문제에선 1이 아닐 수도 있어요 (예시: is_slippery = True일 때)



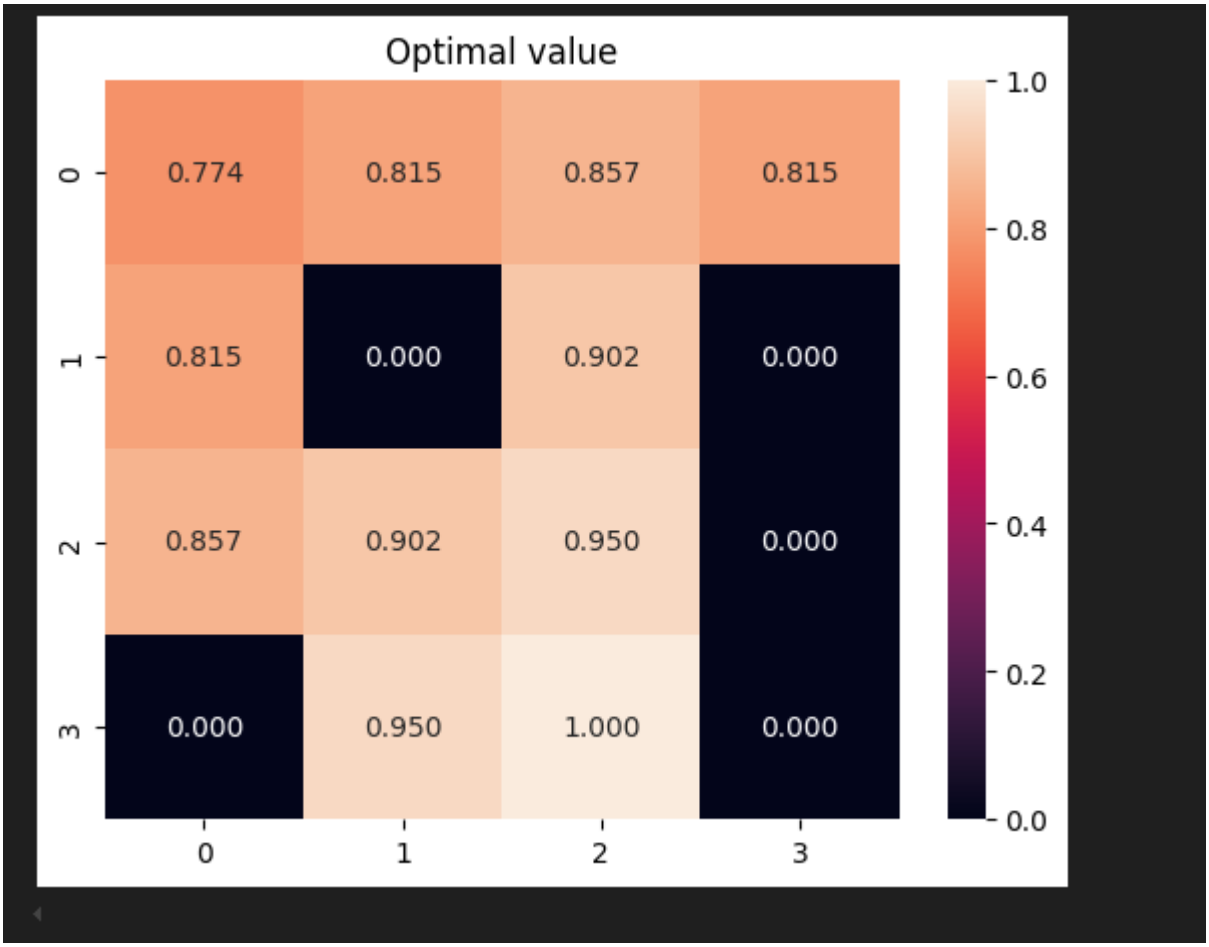
이를 통해 각 칸에서 optimal value 값들을 구합니다. 다시 얘기하자면 value값은 그 칸에서 얻을 수 있는 Return의 기대값입니다. 해당 칸(상태)에서 Return이 큰 값 쪽으로 이동을 하게 된다면 최적의 행동을 찾는겁니다.

여기에 정책을 업데이트까지 해준다면 다음과 같은 코드 및 결과가 나옵니다

```python
V = np.zeros(num_states)
pi = np.ones([num_states, num_actions])*0.25

gamma = 0.95
theta = 1e-5 #0.00001
policy_converge = False

count = 0
while not policy_converge:
    count += 1
    # V(s) evaluation converge
    while True:
        delta = 0
        for s in range(num_states):
            old_value  = V[s]
            new_value = 0

            for a, prob_action in enumerate(pi[s]):

                for prob_environ, s_, reward, terminated_ in transitions[s][a]:
                    new_value += prob_action*prob_environ*(reward + gamma*V[s_])
            V[s] = new_value

            delta = max(delta, np.abs(old_value - V[s]))

        if delta < theta:
            break
```

```python
    ## pi(a|s) update,
    old_pi = np.copy(pi) ## 주소값
    # old_pi = pi ## 주소값

    for s in range(num_states):

        new_action_values = np.zeros(num_actions) # []

        for a in range(num_actions):
            for prob_environ, s_, reward, _ in transitions[s][a]:
                new_action_values[a] += prob_environ*(reward + gamma* V[s_])

        new_action = np.argmax(new_action_values) # 2

        pi[s] = np.eye(num_actions)[new_action]

    print("iteration = {}".format(count))
    # print(pi)
    # time.sleep(0.5)

    if (old_pi == pi).all():
        print("converge = True")
        policy_converge = True
```

```python
## Value
df = pd.DataFrame(V.reshape(4, 4))
print("Optimal State value = \n", df)
sns.heatmap(df, annot=True, fmt = ".3f")
plt.title("Optimal value")
plt.show()

## Policy
# print("Optimal policy = \n", pi)
print()
print("Optimal Action = \n", np.argmax(pi, axis = 1).reshape(4, 4))
```



이외에도 Value Iteration을 통한 optimal policy 찾기도 있으나 내일 수업 듣고 모르겠으면 연락 주세용 최대한 설명드리겠습니다!