동적 프로그래밍

사람	다훈 정
주차	2주차

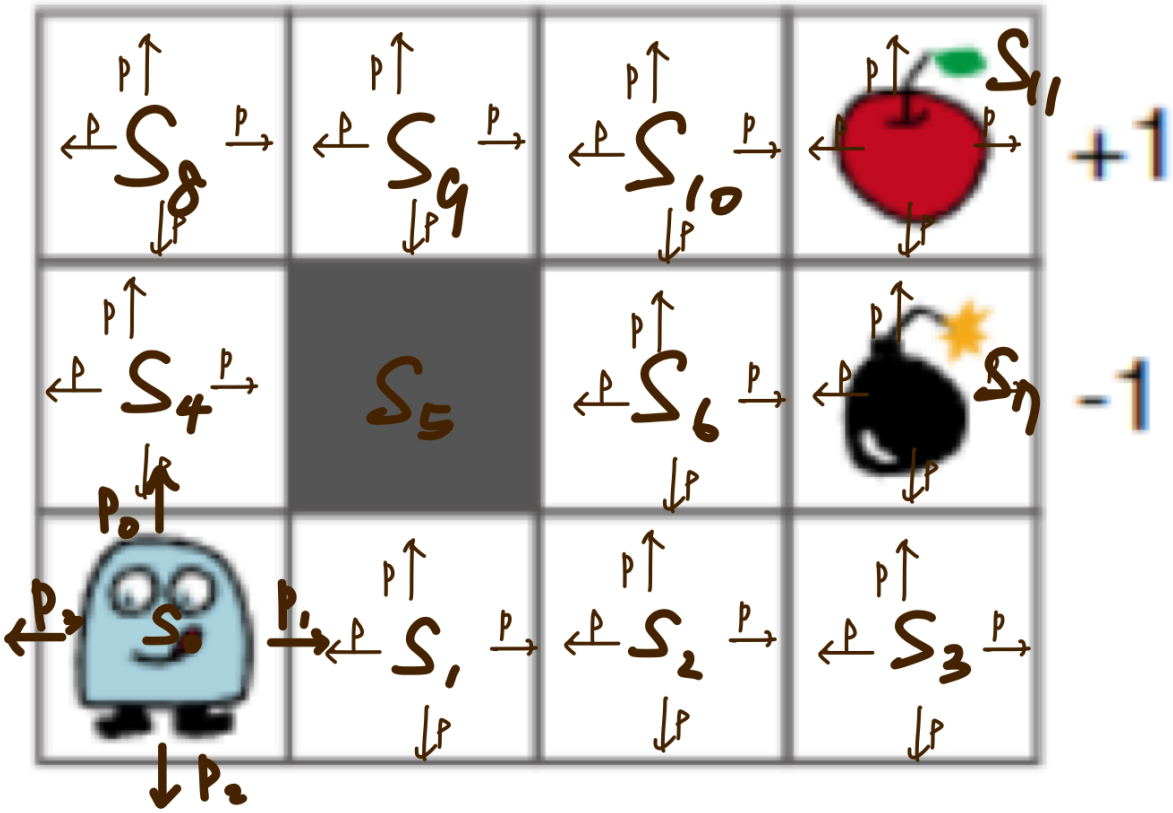
🗄️ 스터디 자료

큰 흐름 잡기

강화학습은 discrete time 에서 stochastic 하게 agent를 control하는 문제이다.

1. Agent는 Policy에 따라 행동을 결정한다.
2. Agent의 행동에 따라 상태가 전이된다.
3. 전이된 상태에서의 Reward를 Agent에게 준다.
4. Agent는 Reward에 따라 자신의 Policy를 수정한다.
   -> Reward가 높았던 행동은 확률을 높여 다음 번엔 더 많이 하도록 기록 / 낮은 경우는 반대로
   -> 직접 반복적으로 수행하며 학습된다 / " 시행착오를 통해 배운다" 라고 표현한다.

1. Agent는 Policy에 따라 행동을 결정한다.



- Policy(정책)는 env(환경) 내에서 전체적으로 정의된 것이다.
- 'Policy'의 용어 사용
  1. Policy(정책)는 env(환경) 내에서 전체적으로 정의된 것이다.(일반적인 정의)
  2. '상태 s에서의 Policy'라는 표현은 '환경 내의 모든 transmission probability를 정의한 Policy'가 정의내린, s에서의 'transmission probability'라고 이해하면 된다.

2. Agent의 행동에 따라 상태가 전이된다.

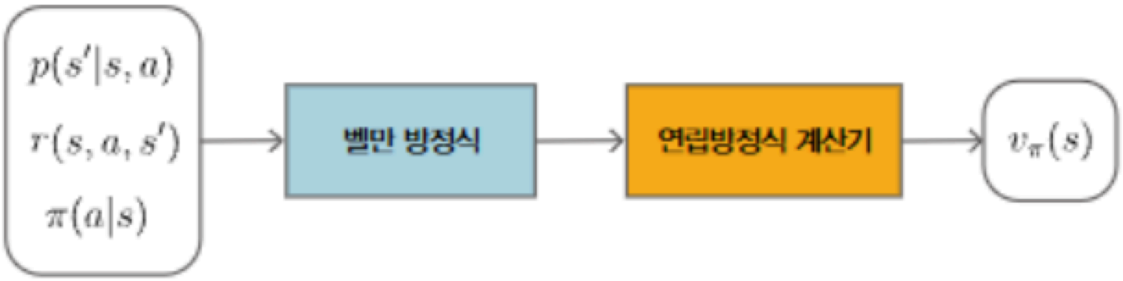
3. 전이된 상태에서의 Reward를 Agent에게 준다.

- 강화학습에서 보상의 구조는 일반적으로 학습 환경에 의해 미리 정의.
- 에이전트는 이 보상을 최대화하기 위해 최적의 전략을 학습하게 된다.

4. Agent는 Reward에 따라 자신의 Policy를 수정한다.

- Q. Policy는 어떻게 수정하는가?
- A. Value function(상태 가치 함수 (V(s))와 행동 가치 함수 (Q(s, a))로 나뉜다.)는 정책이 좋고 나쁨을 판단하는 기준이 되므로 가치 함수의 추정치를 이용해 현재 정책보다 더 나은 정책을 찾을 수 있다.(정책 개선)

- Q. Value function은 어떻게 구하는가?
- A. 벨만 방정식을 이용하면 연립방정식을 얻을 수 있고, 그 연립방정식을 풀 수 있다면 가치 함수를 구할 수 있다.



하지만 상황이 조금만 복잡해지면 계산량이 감당할 수 없는 정도로 많아진다. 그래서 등장한 것이 동적 프로그래밍(Dynamic programming, DP)이다.

4.1 DP와 정책 평가

정책 평가란?

💡 주어진 Policy에 대해 가치 함수를 구하고 싶다.

(그래야 Policy를 평가할 수 있고, 이후 평가를 기반으로 개선할 수 있으니까.)

정책 제어란?

> 💡 평가한 Policy를 개선해서 언젠가 최적 정책을 만들어내고싶다.

(강화 학습의 궁극적인 목표)

- 정책 평가 없이 정책 제어를 하기는 매우 어려우니 정책 평가부터 공부한다.

4.1.1 동적 프로그래밍 기초

가치 함수의 정의 → 벨만 방정식 → 갱신식

가치 함수

$$v_\pi(s) = \mathbb{E}_\pi[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots | S_t = s]$$

- 식에 무한대가 포함되어있다.
  - 계산 못함

벨만 방정식

$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots = R_t + \gamma G_{t+1}$$

- 위 식을 이용해서 벨만 방정식을 유도했음.

$$v_\pi(s) = \sum_{a,s'} \pi(a \mid s)\, p(s' \mid s,a)\{r(s,a,s') + \gamma v_\pi(s')\}$$

- 무한대를 없애버림.
  - 계산 가능
- 식을 잘 보면, 현재 상태(s)의 가치 함수와 다음 상태(s')의 가치 함수의 관계를 나타내고 있음을 알 수 있다.
  - 이 특징을 살려 이 식(벨만 방정식)을 갱신식으로 변형한다.

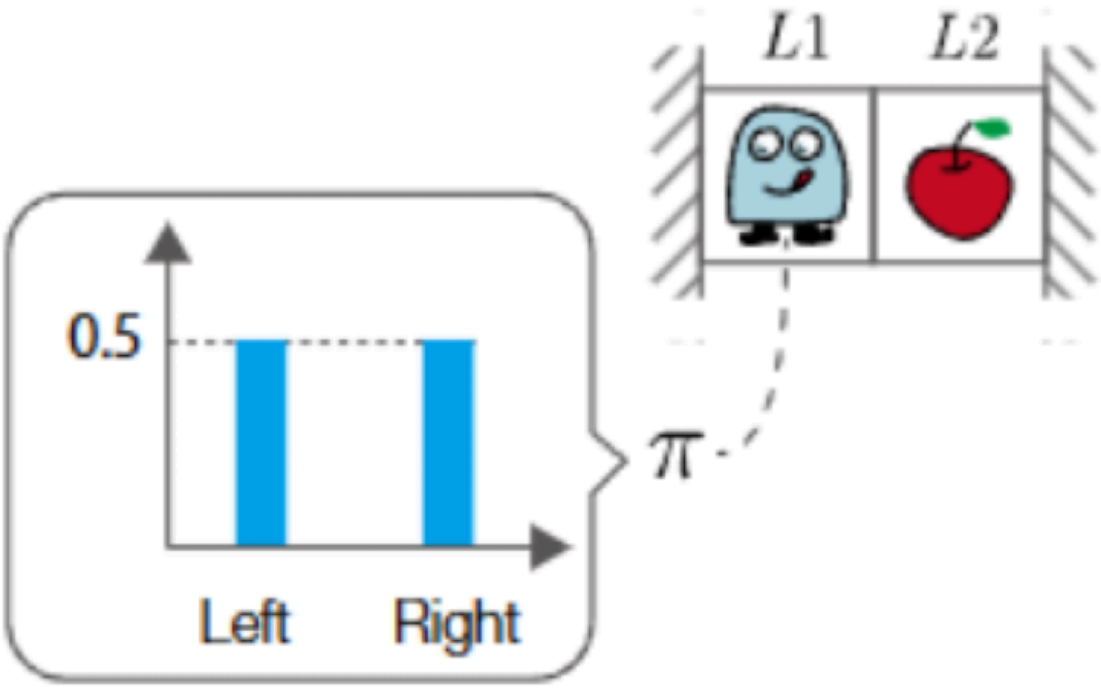
갱신식

$$V_{k+1}(s) = \sum_{a,s'} \pi(a \mid s)\, p(s' \mid s,a)\{r(s,a,s') + \gamma V_k(s')\}$$

- 갱신식 사용법
  1. 초기 가치함수는 맘대로 설정(책에서는 보통 0으로 세팅)
  2. 초기 가치 함수의 추정치 부터 다음 가치 함수의 추정치, 그 다음 가치 함수의 추정치, 그 이후를 연쇄적으로 유도
  3. 그렇게 반복하다보면 최종 목표인 $V_\pi(s)$에 수렴하게 된다.
     - $V_\pi(s)$는 정책 π에 대한 실제 가치 함수 $v_\pi(s)$와 같은 가치 함수 추정치이다.

이것을 **반복적 정책 평가**(iterative policy evaluation)라고 한다.

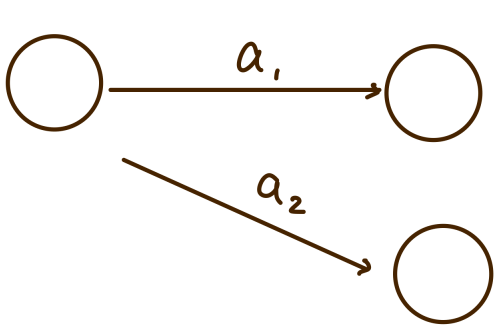
4.1.2 반복적 정책 평가

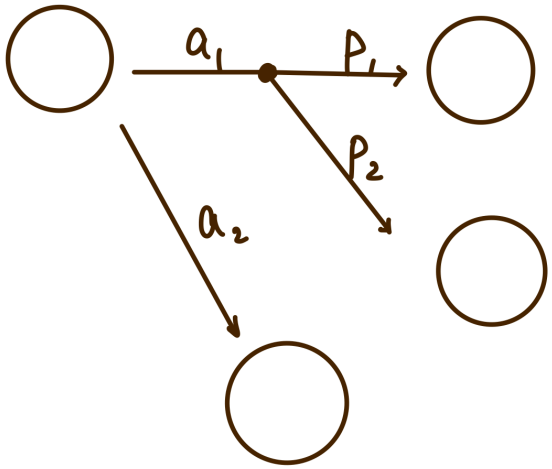


- 에이전트는 무작위 정책 π에 따라 행동한다.
- 이 문제에서 상태 전이는 결정적이다.

! 헷갈리지 말기 !

- stochastic / deterministic Policy
- stochastic / deterministic State Transmission





이 문제에서의 갱신식

$$s' = f(s, a) \text{ 일 때}$$

$$V_{k+1}(s) = \sum_{a} \pi(a|s) \{r(s, a, s') + \gamma V_k(s')\}$$

1. 초기 가치 함수를 맘대로 설정

$$V_0(L1) = 0$$

$$V_0(L2) = 0$$

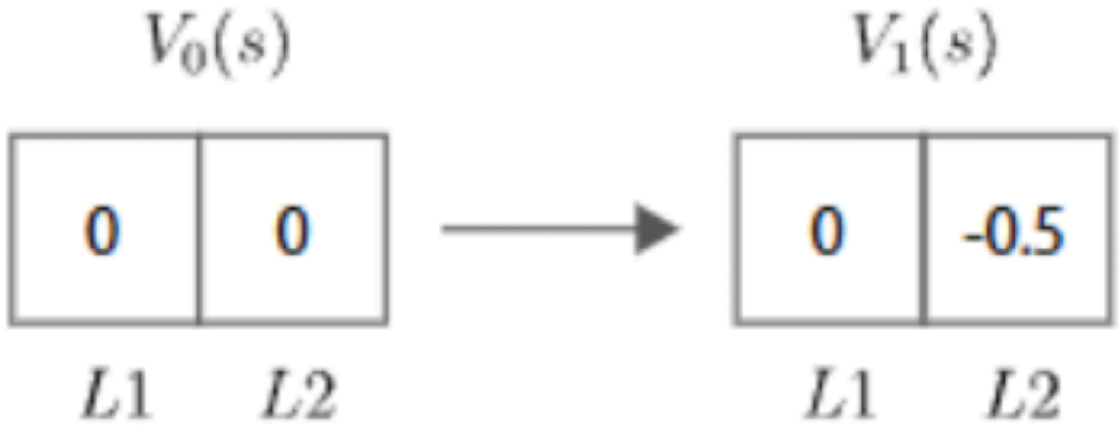
- 쉬운 계산을 위해 0으로 설정해보자.

2. 초기 가치 함수의 추정치 부터 다음 가치 함수의 추정치, 그 다음 가치 함수의 추정치, 그 이후를 연쇄적으로 유도

$$
\begin{aligned}
V_1(L1) &= 0.5\{-1+0.9V_0(L1)\}+0.5\{1+0.9V_0(L2)\} \\
&= 0.5(-1+0.9\cdot 0)+0.5(1+0.9\cdot 0) \\
&= 0
\end{aligned}
$$

$$
\begin{aligned}
V_1(L2) &= 0.5\{0+0.9V_0(L1)\}+0.5\{-1+0.9V_0(L2)\} \\
&= 0.5(0+0.9\cdot 0)+0.5(-1+0.9\cdot 0) \\
&= -0.5
\end{aligned}
$$

그림 4-5 가치 함수 1차 갱신



구현

```
V = {'L1': 0.0, 'L2': 0.0}
new_V = V.copy()  # V의 복사본

for _ in range(100):
    new_V['L1'] = 0.5 * (-1 + 0.9 * V['L1']) + 0.5 * (1 + 0.9 * V['L2'])
    new_V['L2'] = 0.5 * (0 + 0.9 * V['L1']) + 0.5 * (-1 + 0.9 * V['L2'])
    V = new_V.copy()
    print(V)
```

3. 그렇게 반복하다보면 최종 목표인 $V_\pi(s)$에 수렴하게 된다.

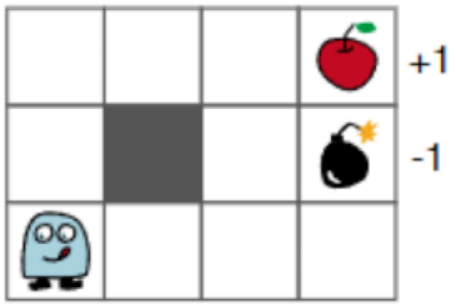
```
V = {'L1': 0.0, 'L2': 0.0}
new_V = V.copy()

cnt = 0  # 갱신 횟수 기록
while True:
    new_V['L1'] = 0.5 * (-1 + 0.9 * V['L1']) + 0.5 * (1 + 0.9 * V['L2'])
    new_V['L2'] = 0.5 * (0 + 0.9 * V['L1']) + 0.5 * (-1 + 0.9 * V['L2'])

     # 갱신된 양의 최댓값
    delta = abs(new_V['L1'] - V['L1'])
    delta = max(delta, abs(new_V['L2'] - V['L2']))
    V = new_V.copy()

    cnt += 1
    if delta < 0.0001:  # 임계값 = 0.0001
        print(V)
        print('갱신 횟수:', cnt)
        break
```

4.2 더 큰 문제를 향해



'3×4 그리드 월드'의 문제 설정은 다음과 같습니다.

- 에이전트는 상하좌우 네 방향으로 이동할 수 있다.
- [그림 4-8]에서 회색 칸은 벽을 뜻하며 벽 안으로는 들어갈 수 없다.
- 그리드 바깥도 벽으로 둘러싸여 더 이상 나아갈 수 없다.
- 벽에 부딪히면 보상은 0이다.
- 사과는 보상 +1, 폭탄은 보상 −1, 그 외의 보상은 0이다.
- 환경의 상태 전이는 고유하다(결정적). 즉, 에이전트가 오른쪽으로 이동하는 행동을 선택하면 (벽만 없다면) 반드시 오른쪽으로 이동한다.
- 이번 문제는 일회성 과제로서, 사과를 얻으면 종료한다.

- 직접 해보길

4.3 정책 반복법

- 정책 반복법은 '정책 평가'와 '정책 개선'을 반복하며 최적 Policy를 찾는 것이다.
- 지금까지 '정책 평가'는 공부했으니, '정책 개선'을 공부해보자.

4.3.1 정책 개선

복습부터 시작하겠습니다. 3.5.2절에서 설명한 바와 같이 최적 정책 μ_*는 다음 식으로 표현됩니다.

$$\mu_*(s) = \underset{a}{\operatorname{argmax}}\, q_*(s,a) \quad \text{[식 4.4]}$$

$$= \underset{a}{\operatorname{argmax}} \sum_{s'} p(s' \mid s,a)\{r(s,a,s') + \gamma v_*(s')\} \quad \text{[식 4.5]}$$

- 식 4.4를 보자.
- State s에서 최적의 Policy a를 찾는 과정이다.
  - s에서 a를 할 때, a′를 할 때, a′′를 할 때,... 중 가장 큰 q function 값을 갖게 하는 Action을 찾는다.
  - 참고: q function이란?(복습)

$$q_\pi(s,a) = E_\pi[G_t | S_t = s, A_t = a]$$

- 단순히 s에서 할 수 있는 행동 중 가장 큰 q value를 갖는 Action을 찾으면 된다.
  - 이를 탐욕 정책(greedy policy)이라 부른다.

개선은 어떻게 이뤄지나?

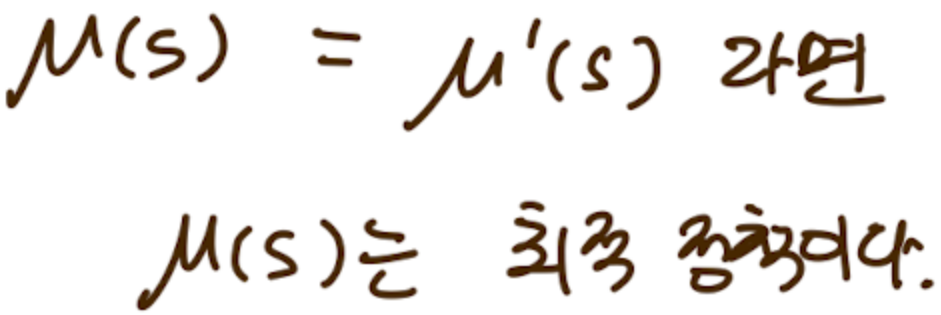
$$\mu'(s) = \underset{a}{\operatorname{argmax}}\, q_\mu(s,a) \quad \text{[식 4.6]}$$

$$= \underset{a}{\operatorname{argmax}} \sum_{s'} p(s' \mid s,a)\{r(s,a,s') + \gamma v_\mu(s')\} \quad \text{[식 4.7]}$$

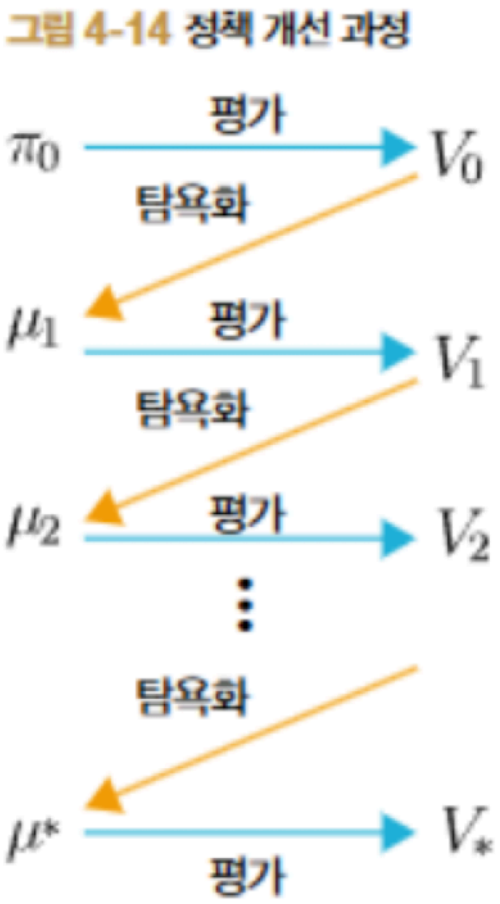
- 식 4.4에서 최적 정책(μ_*)의 자리에 임의의 정책(μ)를 넣어 식을 변형해보자.

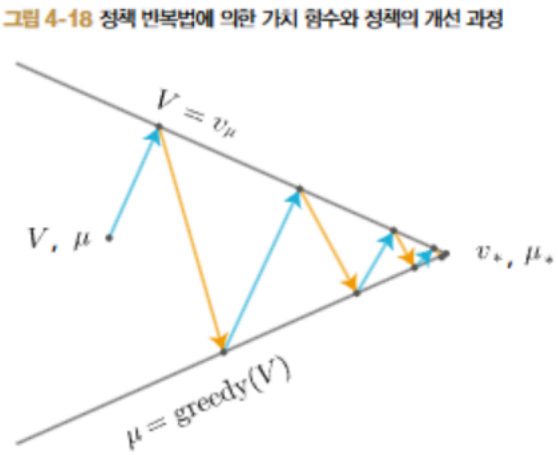
- 그럼 이전 정책(μ)에 의한 가치 함수로 개선된 정책(μ')을 얻을 수 있다.
- 이 때 정책은 무조건 개선된다.(정책 개선 정리, policy improvement theorom)

언제까지 개선해야 최적 정책인가?



4.3.2 평가와 개선 반복

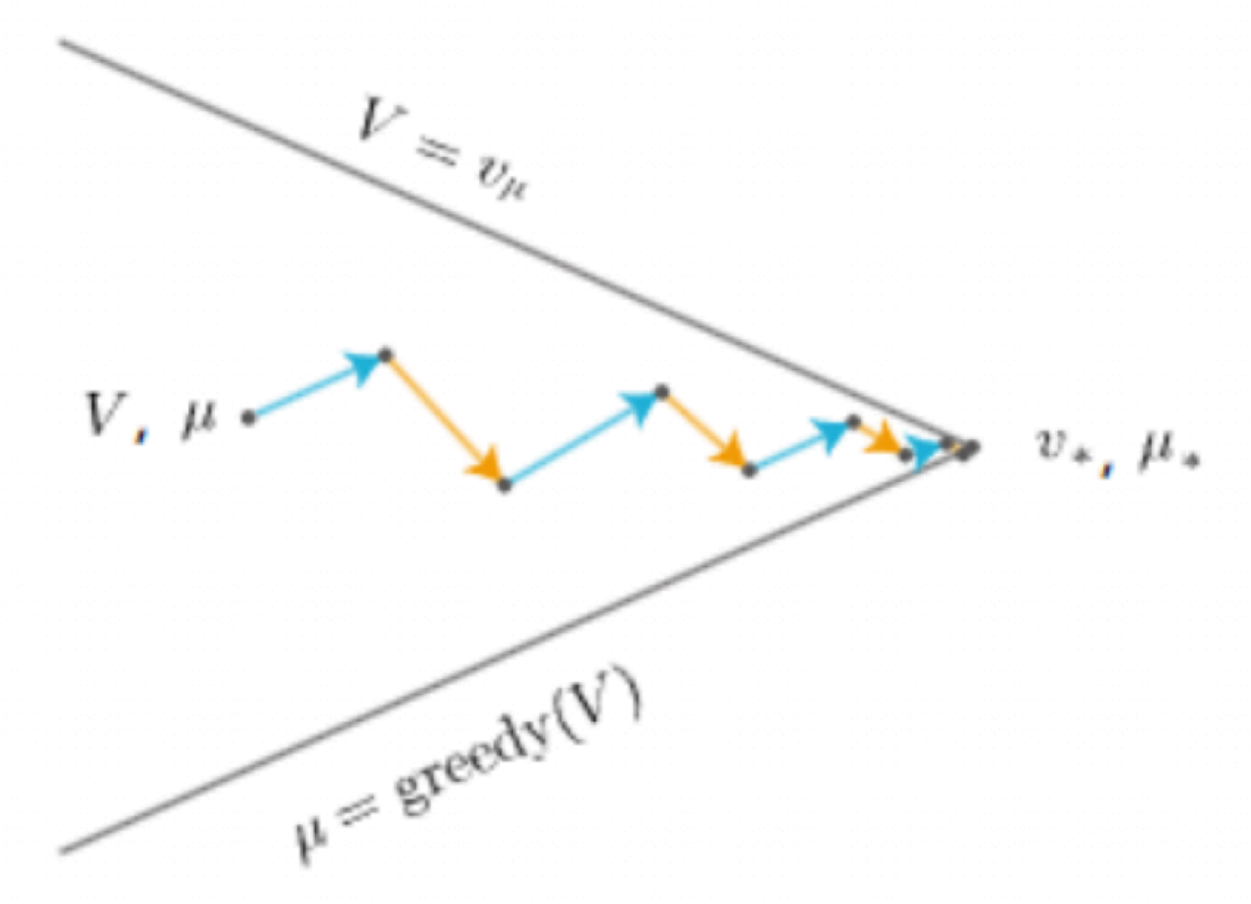




1. 정책이 μ일 때 모든 S에 대한 가치 함수 v_s를 갱신식을 통해 찾아냄
2. $q_\mu(s,a)$를 구할 수 있으므로 정책 개선 식을 통해 정책 μ보다 개선된 μ'를 찾고 찾아 $\mu = \mu$'가 성립할 때까지 반복
3. 1번과 2번을 반복

4.5 가치 반복법

- 정책 반복법에서는 평가와 개선을 각각 '최대한'으로 하고 번갈아 수행한다.
- 평가와 개선을 조금 덜 하면 아래와 같은 그림이 되고, 이를 일반화한 정책 반복(generalized policy iteration)이라고 한다.



- 일반화한 정책 반복은 그냥 정책 반복보다 빠르게 최적 정책에 도달할 것이다.
- 그러니 평가와 개선을 '최소한'으로 하고 번갈아 수행하면 어떨까? 이것을 '가치 반복법 (value iteration)'의 아이디어이다.

❓ Q&A

🚀 구선주	내용

🚀 **김민성**	가치 반복법이 정책 반복법보다 보통 성능이 더 좋은가?
🚀 **이호영**	가치 반복법에서 평가와 개선을 최대로 안한다는게 무슨 의미인지?
🚀 **정지욱**	내용
🚀 **최예림**	내용