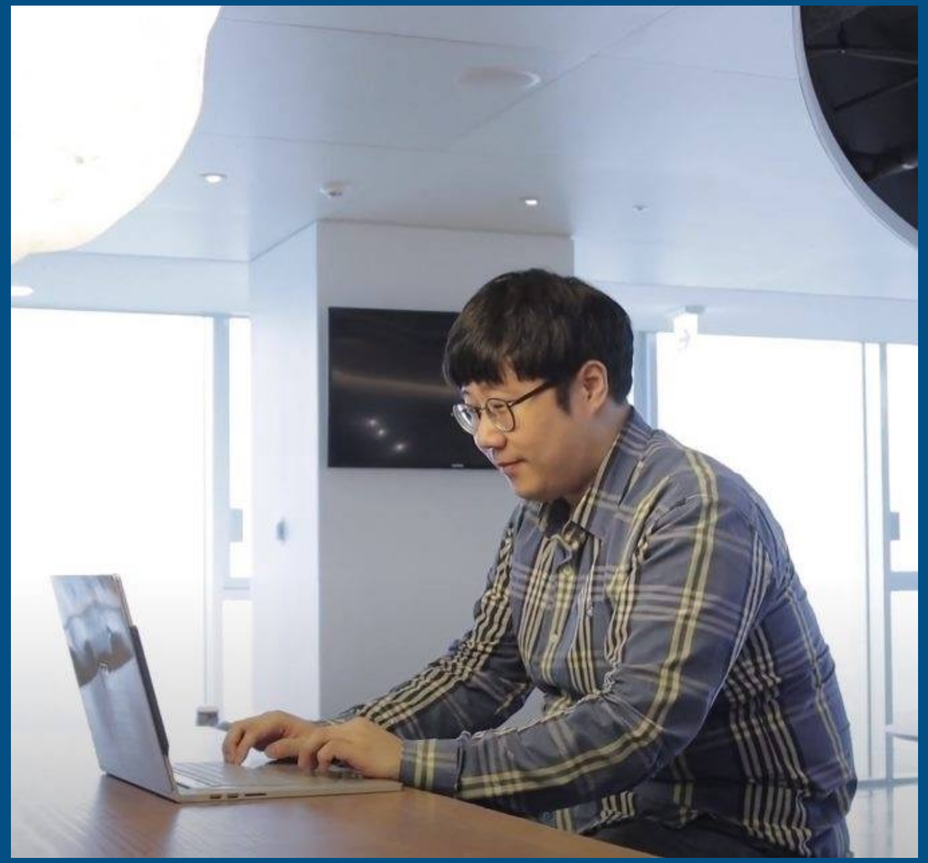
대구소프트웨어마이스터고등학교 특강

Rainbow Is All You Need

Chris Ohk
utilForever@gmail.com

강사 소개

- 옥찬호 (Chris Ohk)
  - 현) Momenti 엔진 엔지니어
  - 전) 넥슨 코리아 게임 프로그래머
  - Microsoft Developer Technologies MVP
  - C++ Korea, Reinforcement Learning KR 관리자
  - IT 전문서 집필 및 번역 다수
    - 게임샐러드로 코드 한 줄 없이 게임 만들기 (2013)
    - 유니티 Shader와 Effect 제작 (2014)
    - 2D 게임 프로그래밍 (2014), 러스트 핵심 노트 (2017)
    - 모던 C++ 입문 (2017), C++ 최적화 (2019)



교재

- 주 교재
  - 파이썬과 케라스로 배우는 강화학습 (위키북스, 2020)
  - Deep Reinforcement Learning Hands-On - Second Edition (Packt, 2020)
- 부 교재
  - Reinforcement Learning, An Introduction - Second Edition (MIT Press, 2018)
  - Reinforcement Learning (O'Reilly Media, 2020)
  - 바닥부터 배우는 강화 학습 (영진닷컴, 2020)

스터디 진행 계획

- Rainbow Is All You Need
  - What is Reinforcement Learning?
  - MDP (Markov Decision Process)
    - State
    - Action
    - Reward Function
    - State Transition Probability
    - Discount Factor
    - Policy
  - Value Function and Q-Function

스터디 진행 계획

- Rainbow Is All You Need
  - Bellman Equation
    - Bellman Expectation Equation
    - Bellman Optimality Equation
  - Dynamic Programming
    - Policy Iteration
    - Value Iteration

스터디 진행 계획

- Rainbow Is All You Need
  - Policy Evaluation
    - Monte-Carlo Prediction
    - Temporal-Difference Prediction
  - SARSA
  - Q-Learning

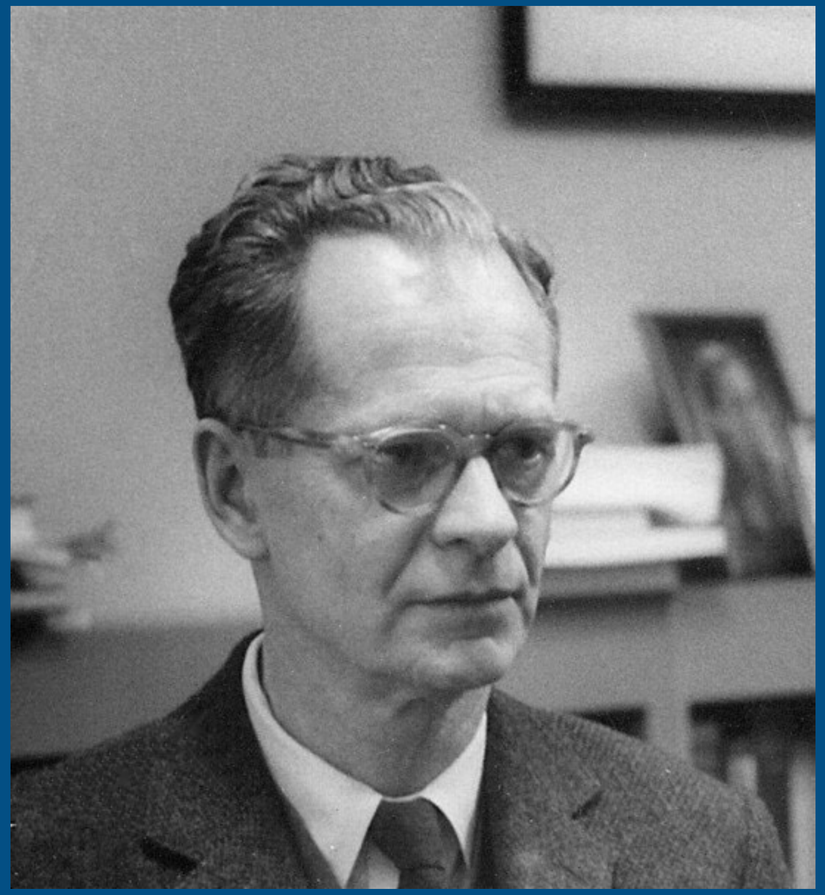
스터디 진행 계획

- Part 2
  - Deep Learning with PyTorch
    - What is PyTorch?
    - PyTorch Tutorial
  - Deep SARSA
  - Policy Gradient
    - Policy-based Reinforcement Learning
    - REINFORCE
  - DQN (Deep Q-Network)

스터디 진행 계획

- Part 3
  - N-step DQN
  - Double DQN
  - Noisy Network
  - Prioritized Experience Replay (PER)
  - Dueling DQN
  - Categorical DQN
  - Rainbow DQN

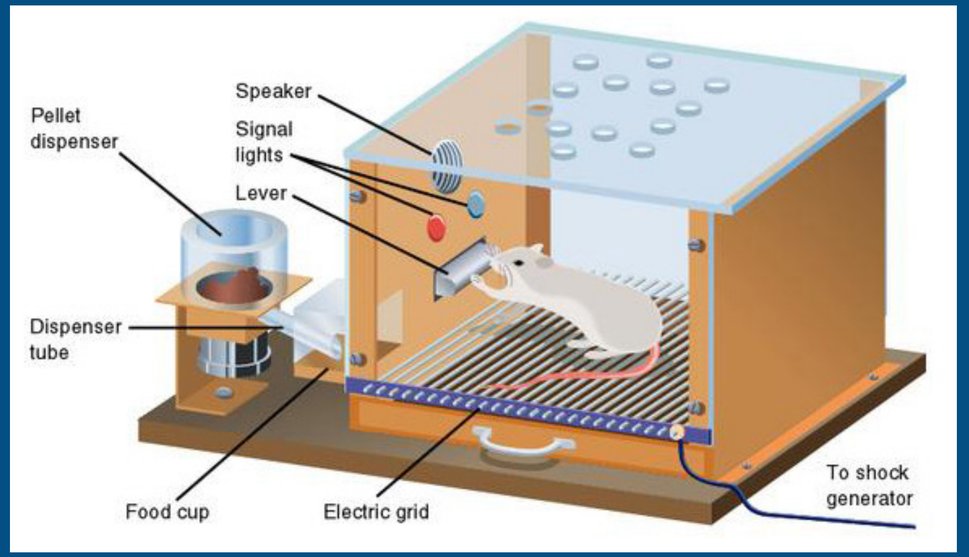
강화학습이란?



B. F. Skinner (1904~1990)

스키너의 강화 연구

1. 굶긴 쥐를 상자에 넣는다.
2. 쥐는 돌아다니다가 우연히 상자 안에 있는 지렛대를 누르게 된다.
3. 지렛대를 누르자 먹이가 나온다.
4. 지렛대를 누르는 행동과 먹이와의 상관관계를 모르는 쥐는 다시 돌아다닌다.
5. 그러다가 우연히 쥐가 다시 지렛대를 누르면 쥐는
   이제 먹이와 지렛대 사이의 관계를 알게 되고
   점점 지렛대를 자주 누르게 된다.
6. 이 과정을 반복하면서 쥐는 지렛대를 누르면
   먹이를 먹을 수 있다는 것을 학습한다.



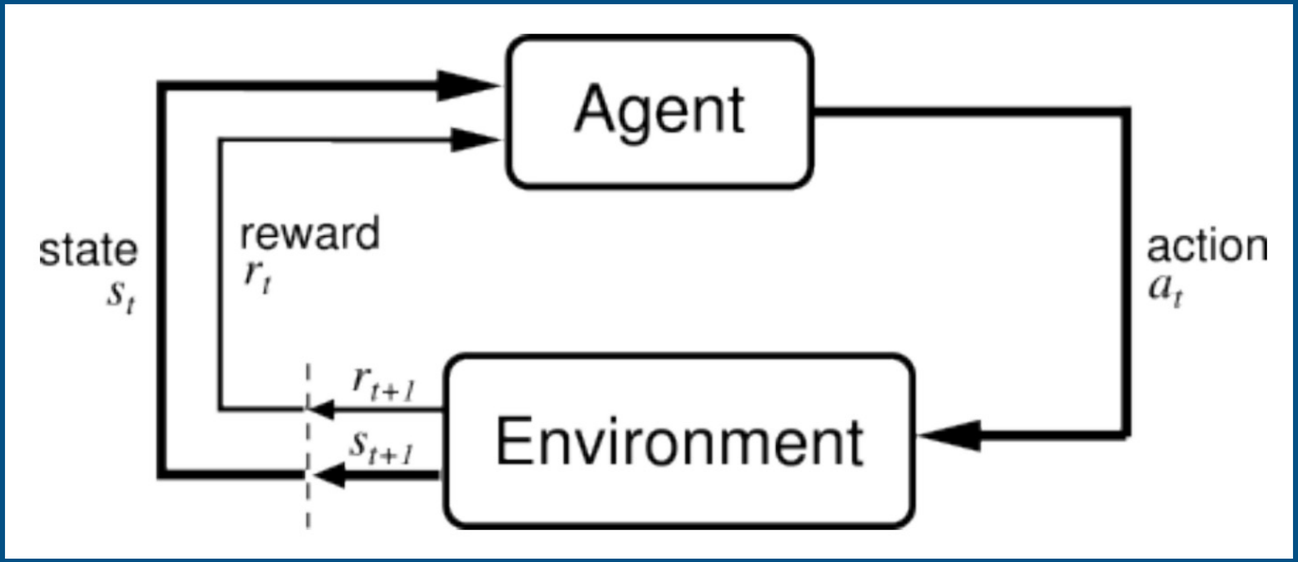
우리 주변에서의 강화

아이가 첫걸음을 떼는 과정도 일종의 강화라고 할 수 있다.

1. 아이는 걷는 것을 배운 적이 없다.
2. 아이는 스스로 이것저것 시도해 보다가 우연히 걷게 된다.
3. 자신이 하는 행동과 걷게 된다는 보상 사이의 상관관계를 모르는 아이는 다시 넘어진다.
4. 시간이 지남에 따라 그 관계를 학습해서 잘 걷게 된다.

강화학습

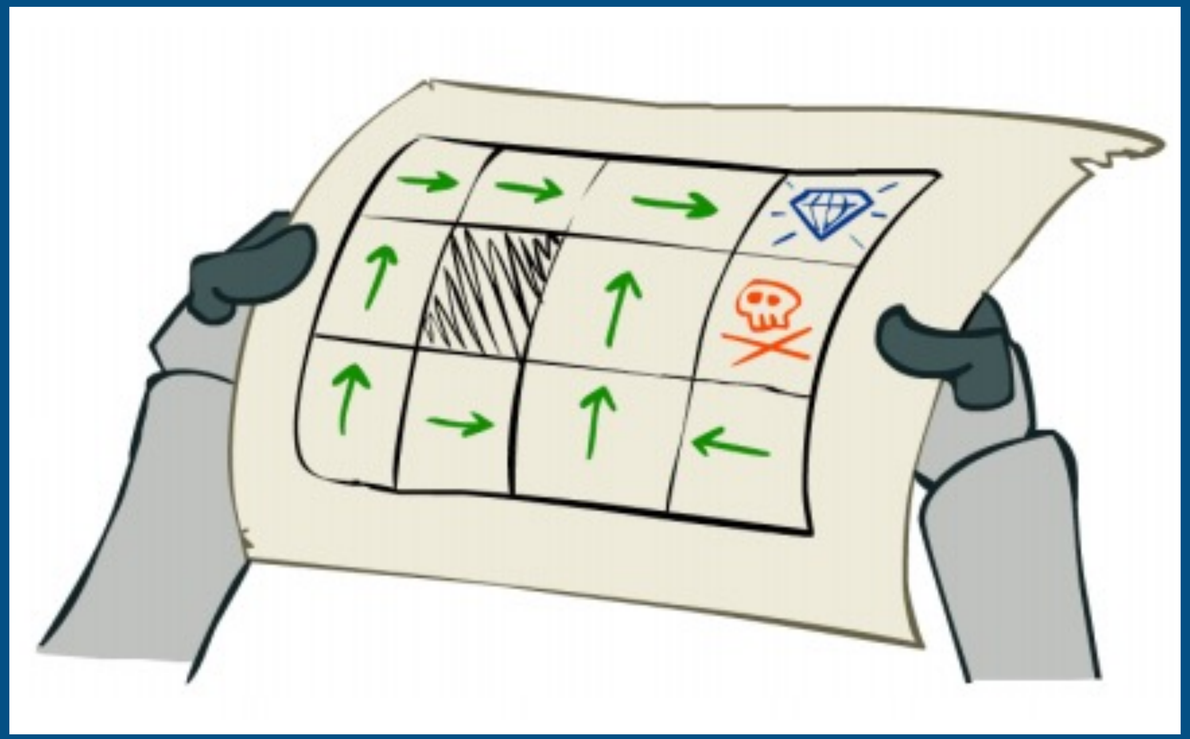
- 에이전트는 사전 지식이 없는 상태에서 학습함
- 에이전트는 자신이 놓인 환경에서 자신의 상태를 인식한 후 행동
- 환경은 에이전트에게 보상을 주고 다음 상태를 알려줌
- 에이전트는 보상을 통해 어떤 행동이 좋은 행동인지 간접적으로 알게 됨



강화학습 문제

결정을 순차적으로 내려야 하는 문제에 강화학습을 적용한다.

이 문제를 풀기 위해서는 문제를 수학적으로 정의해야 한다.



강화학습 문제

수학적으로 정의된 문제는 다음과 같은 구성 요소를 가진다.

1. 상태 (State)
   현재 에이전트의 정보 (정적인 요소 + 동적인 요소)
2. 행동 (Action)
   에이전트가 어떠한 상태에서 취할 수 있는 행동
3. 보상 (Reward)
   에이전트가 학습할 수 있는 유일한 정보, 자신이 했던 행동을 평가할 수 있는 지표
   강화학습의 목표는 시간에 따라 얻는 보상의 합을 최대로 하는 정책을 찾는 것
4. 정책 (Policy)
   순차적 행동 결정 문제에서 구해야 할 답
   모든 상태에 대해 에이전트가 어떤 행동을 해야 하는지 정해놓은 것

MDP

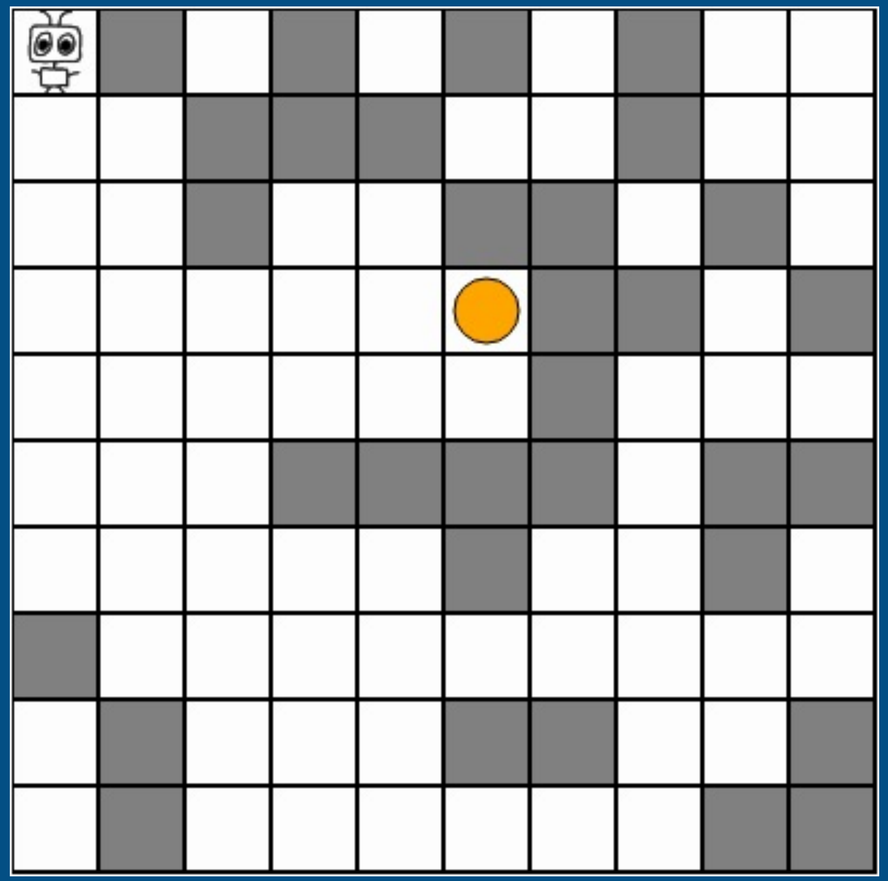
강화 학습은 순차적으로 행동을 계속 결정해야 하는 문제를 푸는 것

→ 이 문제를 수학적으로 표현한 것이 MDP(Markov Decision Process)

- MDP의 구성 요소
  - 상태
  - 행동
  - 보상 함수
  - 상태 변환 확률 (State Transition Probability)
  - 감가율 (Discount Factor)

그리드 월드

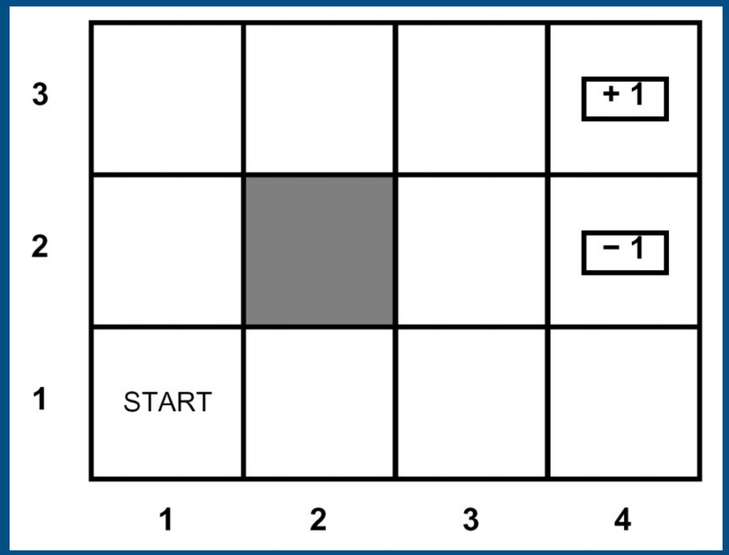
격자로 이뤄진 환경에서 문제를 푸는 각종 예제



상태

에이전트가 관찰 가능한 상태의 집합 : S

- 그리드 월드에서 상태의 개수는 유한
- 그리드 월드에 상태가 5개 있을 경우, 수식으로 표현하면
  $S = \{(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4), (x_5, y_5)\}$
- 그리드 월드에서 상태는 격자 상의 각 위치(좌표)
- 에이전트는 시간에 따라 상태 집합 안에 있는 상태를 탐험한다.
  이 때 시간을 t, 시간 t일 때의 상태를 S_t라고 표현한다.
- 예를 들어, 시간이 t일 때 상태가 $(1, 3)$이라면 $S_t = (1, 3)$



상태

에이전트가 관찰 가능한 상태의 집합 : $\boldsymbol{S}$

- 어떤 t에서의 상태 S_t는 정해진 것이 아니다.
- 때에 따라서 $t = 1$일 때 $S_t = (1, 3)$일 수도 있고 $S_t = (4, 2)$일 수도 있다.

"상태 = 확률 변수(Random Variable)"

상태

$$S_t = s$$

"시간 t에서의 상태 S_t가 어떤 상태 s다."

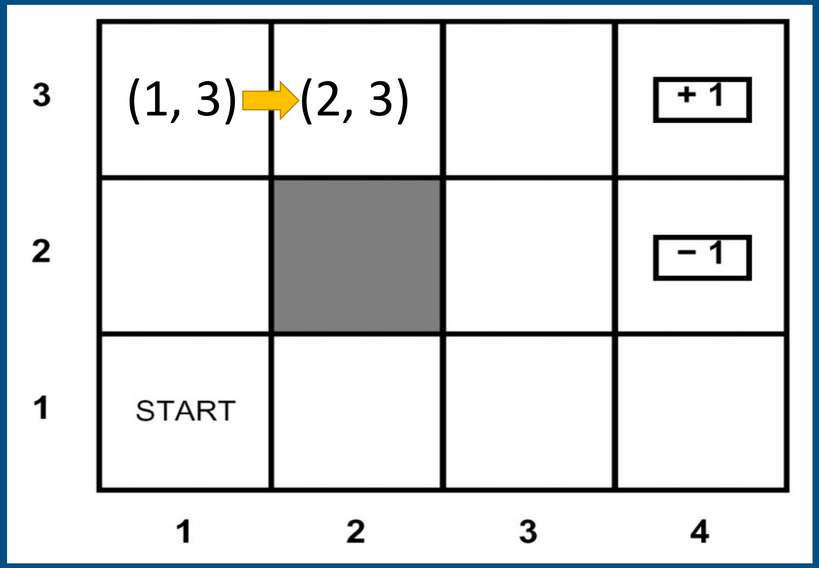
행동

에이전트가 상태 S_t에서 할 수 있는 가능한 행동의 집합 : A

- 보통 에이전트가 할 수 있는 행동은 모든 상태에서 같다.

$$A_t = a$$

- "시간 t에 에이전트가 특정한 행동 a를 했다."
- t라는 시간에 에이전트가 어떤 행동을 할 지는 정해져 있지 않으므로 A_t처럼 대문자로 표현한다.
- 그리드 월드에서 에이전트가 할 수 있는 행동은 $A = \{\text{up}, \text{down}, \text{left}, \text{right}\}$
- 만약 시간 t에서 상태가 (1, 3)이고 $A_t = \text{right}$라면 다음 시간의 상태는 (2, 3)이 된다.



보상 함수

에이전트가 학습할 수 있는 유일한 정보

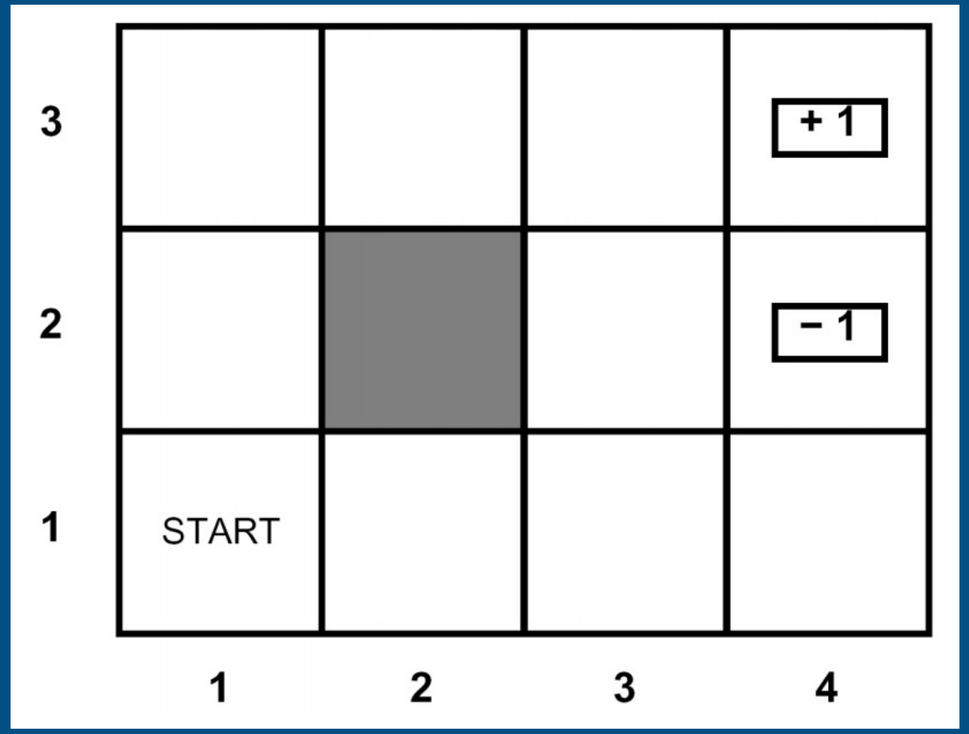
- 보상 함수 (Reward Function)

$$R_s^a = \mathrm{E}[R_{t+1}|S_t = s, A_t = a]$$

- 시간 t일 때 상태가 $S_t = s$이고 그 상태에서 행동이 $A_t = a$를 했을 경우 받을 보상에 대한 기댓값(Expectation) E
- 에이전트가 어떤 상태에서 행동한 시간 : t
  보상을 받는 시간 : $t + 1$
- 이유 : 에이전트가 보상을 알고 있는게 아니라 환경이 알려주기 때문
  에이전트가 상태 s에서 행동 a를 하면 환경은 에이전트가 가게 되는 다음 상태 s'와 에이전트가 받을 보상을 에이전트에게 알려준다. 이 시점이 $t + 1$이다.

보상 함수

에이전트가 학습할 수 있는 유일한 정보



상태 변환 확률

에이전트가 어떤 상태에서 어떤 행동을 취하면 상태가 변한다.

하지만 어떤 이유로 인해 다음 상태로 변하지 못할 수도 있다.

→ 상태의 변화에는 확률적인 요인이 들어간다.

이를 수치적으로 표현한 것이 상태 변환 확률!

$$P_{ss'}^{a} = \mathrm{P}[S_{t+1} = s' | S_t = s, A_t = a]$$

감가율

에이전트는 항상 현재 시점에서 판단을 내리기 때문에
현재에 가까운 보상일수록 더 큰 가치를 갖는다.

보상의 크기가 100일 때, 에이전트가 현재 시각에 보상을 받을 때는
100의 크기 그대로 받아들이지만 현재로부터 일정 시간이 지나서
보상을 받으면 크기가 100이라고 생각하지 않는다.

에이전트는 그 보상을 얼마나 시간이 지나서 받는지를 고려해
감가시켜 현재의 가치로 따진다.

감가율

감가율

A 은행

“당첨금 1억 원을 지금 당장 드리겠습니다.”

vs

B 은행

“지금 당장 받으면 막쓰다가 탕진할 가능성이 크니,
10년 후에 당첨금 1억 원을 드리겠습니다.”

감가율

A 은행
“당첨금 1억 원을 지금 당장 드리겠습니다.”

≠

B 은행
“지금 당장 받으면 막쓰다가 탕진할 가능성이 크니,
10년 후에 당첨금 1억 원을 드리겠습니다.”

감가율

A 은행

“당첨금 1억 원을 지금 당장 드리겠습니다.”

≒

B 은행

“지금 당장 받으면 막쓰다가 탕진할 가능성이 크니,
10년 후에 당첨금 1억 원에 이자까지 드리겠습니다.”

감가율

우리는 이자를 통해 나중에 받을 보상에
추가적인 보상을 더해 현재의 보상과 같게 만든다.

→ 반대로 말하면 같은 보상이면 나중에 받을수록 가치가 줄어든다.
이를 수학적으로 표현한 개념이 “감가율(Discount Factor)”

감가율

감가율 : 시간에 따라서 감가하는 비율

$$\gamma \in [0, 1]$$

현재의 시간 t로부터 시간 k가 지난 후에 받는 보상이 R_{t+k}라면

현재 그 보상의 가치는 $\gamma^{k-1}R_{t+k}$와 같다.

즉, 더 먼 미래에 받을 수록 에이전트가 받는 보상의 크기는 줄어든다.

정책

모든 상태에서 에이전트가 할 행동

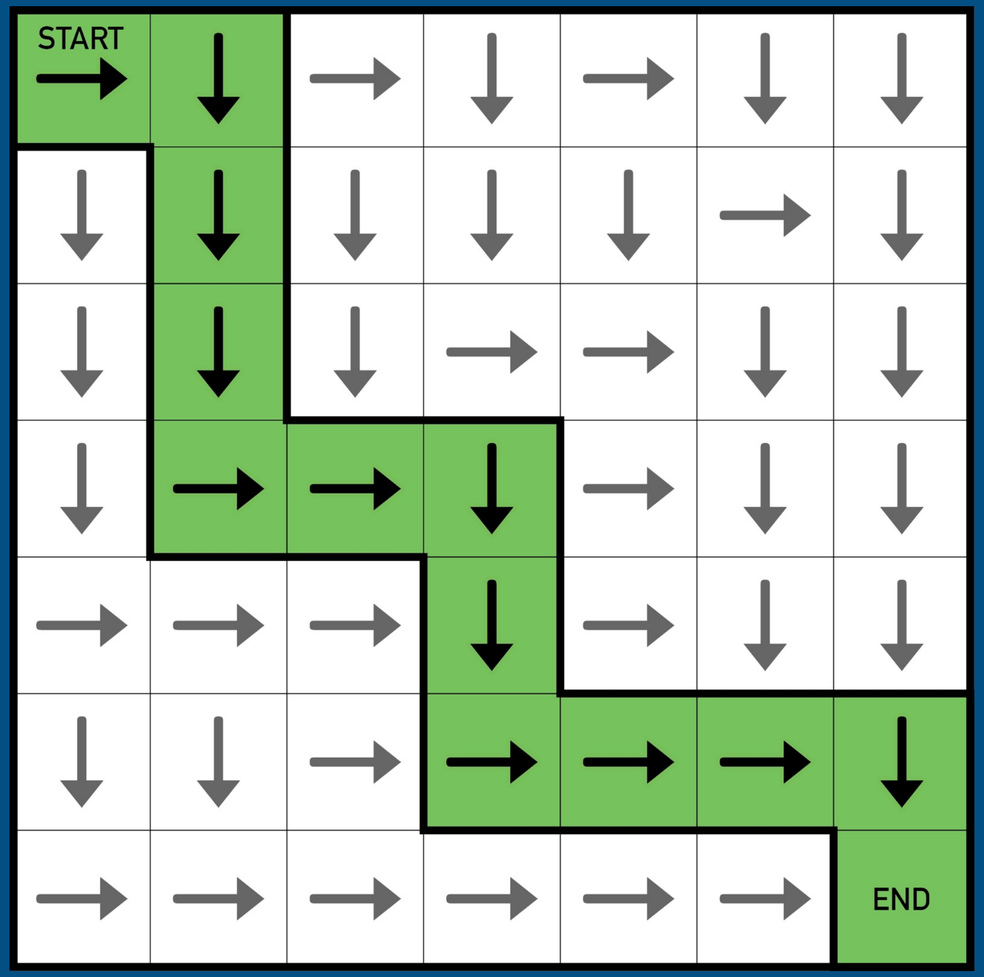
- 상태가 입력으로 들어오면 행동을 출력으로 내보내는 일종의 함수
- 하나의 행동만을 나타낼 수도 있고, 확률적으로 $a_1 = 10\%$, $a_2 = 90\%$로 나타낼 수도 있다.

$$\pi(a|s) = \mathrm{P}[A_t = a|S_t = s]$$

- 시간 t에 에이전트가 $S_t = s$에 있을 때 가능한 행동 중에서 $A_t = a$를 할 확률
- 강화 학습 문제를 통해 알고 싶은 것은 정책이 아닌 “최적 정책”

정책

2021 DGSW
Rainbow Is All You Need



가치함수

우리가 지금까지 한 일 : 문제를 MDP로 정의

→ 에이전트는 MDP를 통해 최적 정책을 찾으면 된다.

하지만 에이전트가 어떻게 최적 정책을 찾을 수 있을까?

가치함수

에이전트 입장에서 어떤 행동을 하는 것이 좋은지를 어떻게 알 수 있을까?

→ 현재 상태에서 앞으로 받을 보상을 고려해서 선택해야 좋은 선택!

하지만 아직 받지 않은 보상들을 어떻게 고려할 수 있을까?

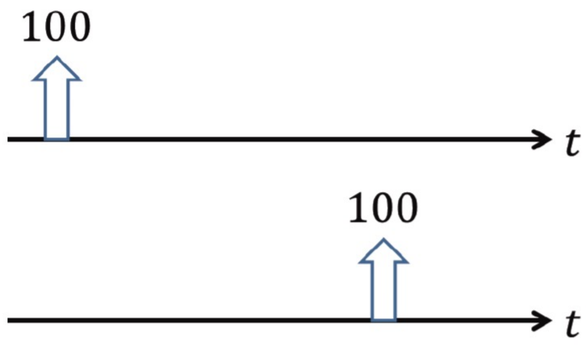
→ 에이전트는 가치함수를 통해 행동을 선택할 수 있다.

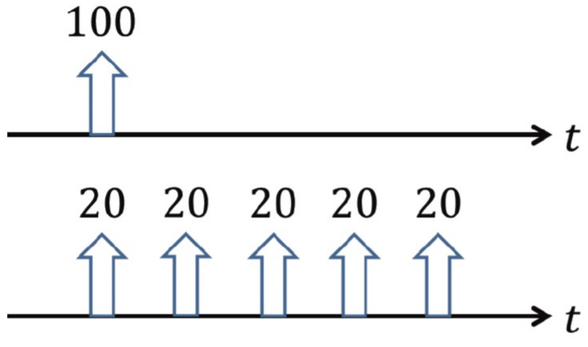
가치함수

현재 시간 t부터 에이전트가 행동을 하면서 받을 보상을 모두 더해보자.

$$R_{t+1} + R_{t+2} + R_{t+3} + R_{t+4} + R_{t+5} + \cdots$$

하지만 이 수식에는 3가지 문제가 있다.



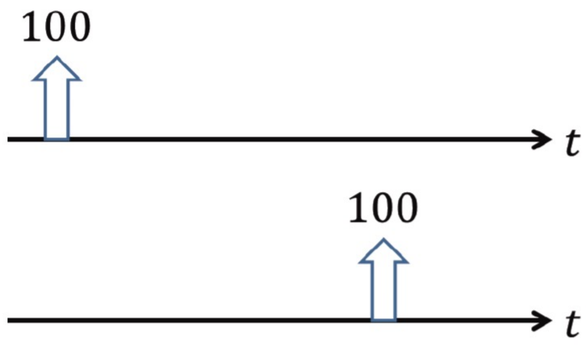


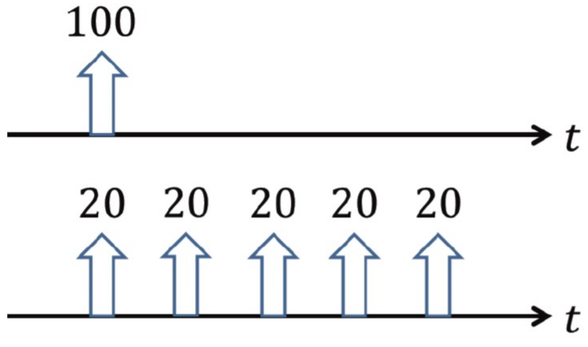
$$0.1 + 0.1 + \cdots = \infty$$

$$1 + 1 + \cdots = \infty$$

가치함수

1. 현재에 받은 보상 100과 미래에 받는 보상 100을 똑같이 취급한다.
2. 보상 100을 1번 받을 때와 보상 20을 5번 받을 때를 구분하지 못한다.
3. 시간이 무한대라면 0.1을 받아도, 1을 받아도 합이 무한대다.





$$0.1 + 0.1 + \cdots = \infty$$
$$1 + 1 + \cdots = \infty$$

가치함수

단순한 보상의 합으로는 판단을 내리기 어려우므로,
좀 더 정확한 판단을 위해 감가율을 고려한다.

$$R_{t+1} + R_{t+2} + R_{t+3} + R_{t+4} + R_{t+5} + \cdots$$

$$R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \gamma^4 R_{t+5} + \cdots$$

가치함수

이 값을 반환값(Return) G_t라고 한다.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$$

예를 들어, 에피소드를 $t = 1$부터 5까지 진행했다면

$$G_1 = R_2 + \gamma R_3 + \gamma^2 R_4 + \gamma^3 R_5 + \gamma^4 R_6$$

$$G_2 = R_3 + \gamma R_4 + \gamma^2 R_5 + \gamma^3 R_6$$

$$G_3 = R_4 + \gamma R_5 + \gamma^2 R_6$$

$$G_4 = R_5 + \gamma R_6$$

$$G_5 = R_6$$

가치함수

에이전트는 에피소드가 끝난 후에야 반환값을 알 수 있다.
하지만 에피소드가 끝날 때까지 기다려야 할까?

때로는 정확한 값을 얻기 위해 끝까지 기다리는 것보다
정확하지 않더라도 현재의 정보를 토대로 행동하는 것이 나을 때가 있다.

→ 가치함수 = 어떠한 상태에 가면 받을 것이라고 예상되는 값

$$v(s) = E[G_t \mid S_t = s]$$

가치함수

$$v(s) = E[G_t \mid S_t = s]$$

$$v(s) = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s]$$

$$v(s) = E[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \cdots) \mid S_t = s]$$

$$v(s) = E[R_{t+1} + \gamma G_{t+1} \mid S_t = s]$$

가치함수

- 반환값으로 나타내는 가치함수

$$v(s) = E[R_{t+1} + \gamma G_{t+1} | S_t = s]$$

- 가치함수로 표현하는 가치함수의 정의

$$v(s) = E[R_{t+1} + \gamma v(S_{t+1}) | S_t = s]$$

여기까지는 가치함수를 정의할 때 정책을 고려하지 않음

가치함수

하지만 정책을 고려하지 않으면 안된다.

- 상태에서 상태로 넘어갈 때 에이전트는 무조건 행동을 해야 하고
  각 상태에서 행동을 하는 것이 에이전트의 정책이기 때문
- 정책에 따라서 계산하는 가치함수는 당연히 달라질 수밖에 없음
- MDP에서의 가치함수는 항상 정책을 고려

가치함수

벨만 기대 방정식(Bellman Expectation Equation)

$$v_\pi(s) = E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s]$$

- 현재 상태의 가치 함수와 다음 상태의 가치함수 사이의 관계를 말해주는 방정식
- 강화학습은 벨만 방정식을 어떻게 풀어나가느냐의 스토리

큐함수

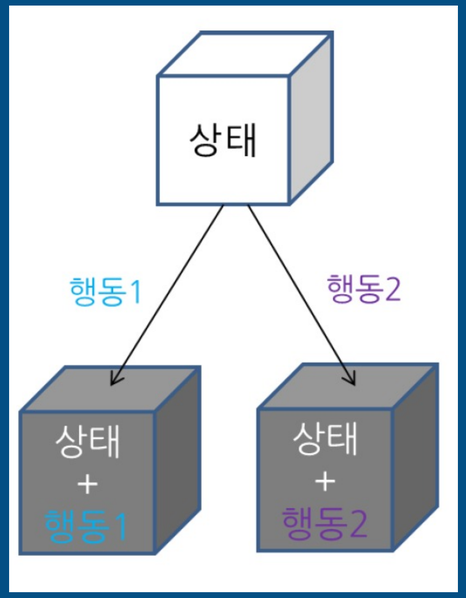
가치함수는 말 그대로 “함수” → 입력/출력이 무엇인지 알아야 한다.

- 지금까지 설명한 가치함수는 상태 가치함수
- 에이전트는 가치함수를 통해 다음에 어떤 상태로 가야할 지 판단할 수 있다.
- 어떤 상태로 가면 좋을지 판단한 후에 그 상태로 가기 위한 행동을 따져볼 것이다.

큐함수

하지만…

- 어떤 상태에서 각 행동에 대해 따로 가치함수를 만들어서 그 정보를 얻어올 수 있다면 에이전트는 굳이 다음 상태의 가치 함수를 따져보지 않아도 어떤 행동을 해야할 지 선택할 수 있다.
- 이처럼 어떤 상태에서 어떤 행동이 얼마나 좋은지 알려주는 함수를 행동 가치함수라고 한다.
  → 큐함수(Q Function)!



큐함수

가치함수와 큐함수 사이의 관계

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

1. 각 행동을 했을 때 앞으로 받을 보상인 큐함수 $q_\pi(s, a)$를 $\pi(a \mid s)$에 곱한다.
2. 모든 행동에 대해 큐함수와 정책을 곱한 값을 더하면 가치함수가 된다.

큐함수는 강화학습에서 중요한 역할을 한다.

- 강화학습에서 에이전트가 행동을 선택하는 기준으로
  가치함수보다는 보통 큐함수를 사용한다.
- 그 이유는 뒤에서 설명할 예정

큐함수

큐함수 또한 벨만 기대 방정식의 형태로 나타낼 수 있다.

$$q_\pi(s,a) = E_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a]$$

- 가치함수의 식과 다른 점은 조건문에 행동이 더 들어간다는 점

벨만 기대 방정식

$$v_\pi(s) = E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s]$$

위 함수는 현재 가치함수 값을 갱신한다.

하지만 갱신하려면 기댓값을 계산해야 하는데 어떻게 계산할까?

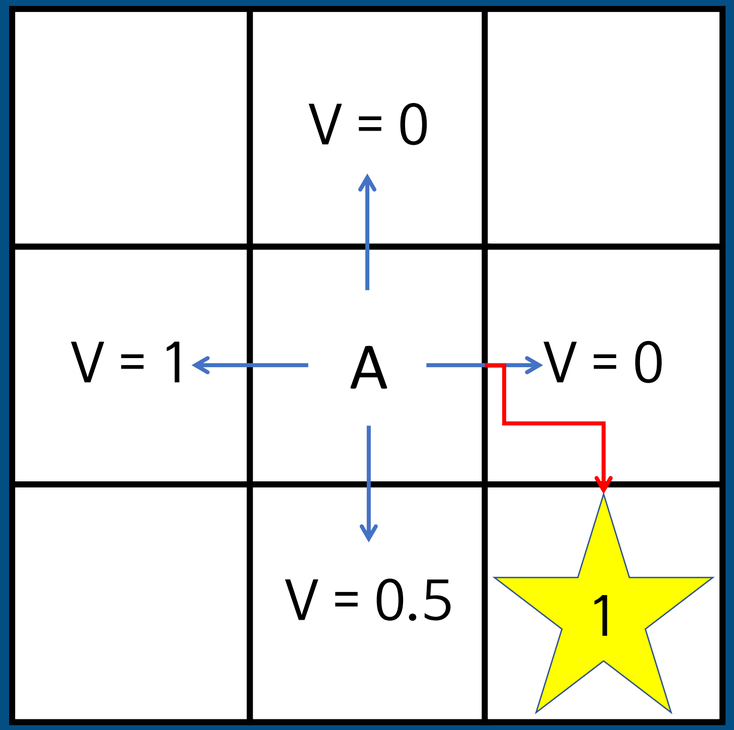
- 기댓값에는 어떤 행동을 할 확률(정책 $\pi(a \mid s)$)과
  그 행동을 했을 때 어떤 상태로 가게 되는 확률(상태 변환 확률 $P^a_{ss'}$)이 포함되어 있다.
- 따라서 정책과 상태 변환 확률을 포함해서 계산하면 된다.

$$v_\pi(s) = \sum_{a \in A} \pi(a \mid s) \left(R_{t+1} + \gamma \sum_{s' \in S} P^a_{ss'} \cdot v_\pi(s') \right)$$

벨만 기대 방정식

상태 변환 확률을 모든 s와 a에 대해 1이라고 가정하자.
그리고 다음 예제를 한 번 생각해보자.

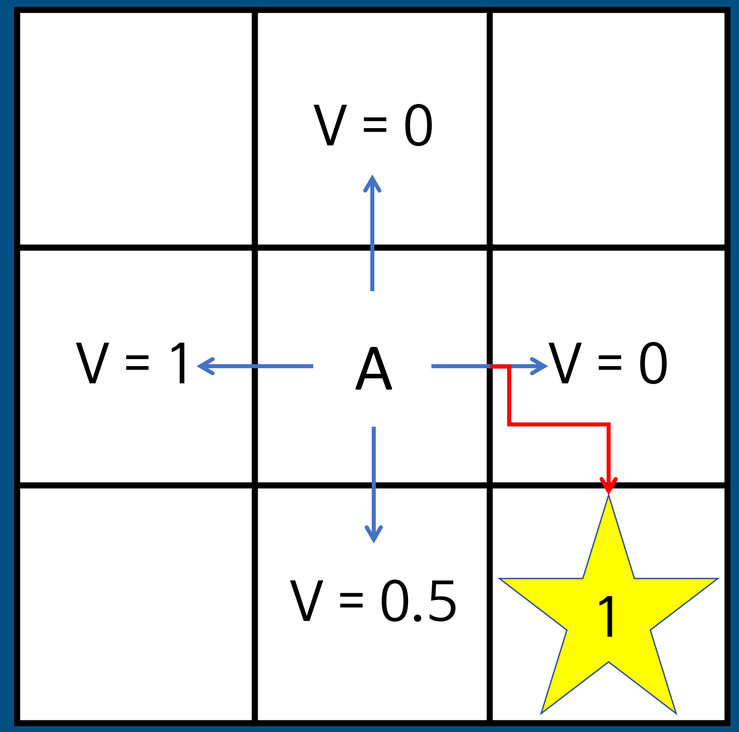
- 행동은 “상, 하, 좌, 우” 4가지
- 에이전트의 초기 정책은 무작위로 각 행동의 선택 확률은 25%
- 현재 에이전트 상태에 저장된 가치함수는 0
  - 왼쪽 상태의 가치함수는 1, 밑쪽 상태의 가치함수는 0.5
    위쪽 상태의 가치함수는 0, 오른쪽 상태의 가치함수는 0
- 감가율은 0.9
- 오른쪽으로 행동을 취할 경우
  노란색 별로 표현된 1의 보상을 받음



벨만 기대 방정식

$$v_\pi(s) = \sum_{a \in A} \pi(a \mid s)\left(R_{t+1} + \gamma v_\pi(s')\right)$$

- 행동 = 상 : $0.25 \times (0 + 0.9 \times 0) = 0$
- 행동 = 하 : $0.25 \times (0 + 0.9 \times 0.5) = 0.1125$
- 행동 = 좌 : $0.25 \times (0 + 0.9 \times 1) = 0.225$
- 행동 = 우 : $0.25 \times (1 + 0.9 \times 0) = 0.25$
- 기댓값 = $0 + 0.1125 + 0.225 + 0.25 = 0.5875$



벨만 최적 방정식

$$v_\pi(s) = E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s]$$

벨만 기대 방정식을 통해 계속 계산을 진행하다 보면

언젠가 식의 왼쪽 항과 오른쪽 항이 동일해지는 순간이 온다.

→ $v_\pi(s)$ 값이 수렴 → 현재 정책 π에 대한 참 가치함수를 구한 것

벨만 최적 방정식

하지만 참 가치함수와 최적 가치함수는 다르다.

- 참 가치함수는 “어떤 정책”을 따라서 움직였을 경우에 받게 되는 보상에 대한 참값
- 가치함수의 정의가 “현재로부터 미래까지 받을 보상의 총합”인데
  이 값이 얼마가 될지에 대한 값
- 하지만 최적의 가치함수는 수많은 정책 중에서 가장 높은 보상을 주는 가치함수다.

$$v_{k+1}(s) = \pi(a \mid s)\left(R_s^a + \gamma v_k(s')\right)$$

- $v_{k+1}(s)$: 현재 정책에 따라 $k+1$번째 계산한 가치함수 (그 중에서 상태 s의 가치함수)
- $k+1$번째의 가치함수는 k번째 가치함수 중에서 주변 상태들 s'을 이용해 구한다.
- 그리고 이 계산은 모든 상태에 대해 동시에 진행한다.

벨만 최적 방정식

강화학습은 MDP로 정의되는 문제에서 최적 정책을 찾는다.
더 좋은 정책은 무엇일까? 어떤 정책이 더 좋은 정책일까?

→ 가치함수(받을 보상들의 합)를 통해 판단할 수 있다.

모든 정책에 대해 가장 큰 가치함수를 주는 정책이 최적 정책

→ 최적 정책을 따라갔을 때 받을 보상의 합이 최적 가치함수

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

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

벨만 최적 방정식

최적 정책은 각 상태 s에서의 최적의 큐함수 중에서
가장 큰 큐함수를 가진 행동을 하는 것

→ 선택 상황에서 판단 기준은 큐함수이며, 최적 정책은 언제나
이 큐함수 중에서 가장 높은 행동을 하나 하는 것

$$\pi_*(s,a) = \begin{cases} 1, & if\ a = \underset{a \in A}{\operatorname{argmax}}\, q_*(s,a) \\ 0, & otherwise \end{cases}$$

벨만 최적 방정식

최적의 가치함수 = 최적의 큐함수 중에서 최대를 선택하는 것

$$v_*(s) = \max_a [q_*(s,a) \mid S_t = s, A_t = a]$$

큐함수를 가치함수로 고쳐보자.

$$v_*(s) = \max_a E[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a]$$

이를 벨만 최적 방정식(Bellman Optimality Equation)이라고 한다.

$$q_*(s,a) = E\left[R_{t+1} + \gamma \max_a q_*(S_{t+1}, a') \mid S_t = s, A_t = a\right]$$

다이내믹 프로그래밍

한번에 풀기 어려운 문제를 여러 개의 작은 문제로 나눠 푸는 것

- 작은 문제들 사이에서 공유하는 계산 결과들을 재사용해 총 계산량을 줄일 수 있음
- 예) 총 4칸인 계단을 한 번에 1칸, 2칸씩 오르는 경우의 수는?

다이내믹 프로그래밍

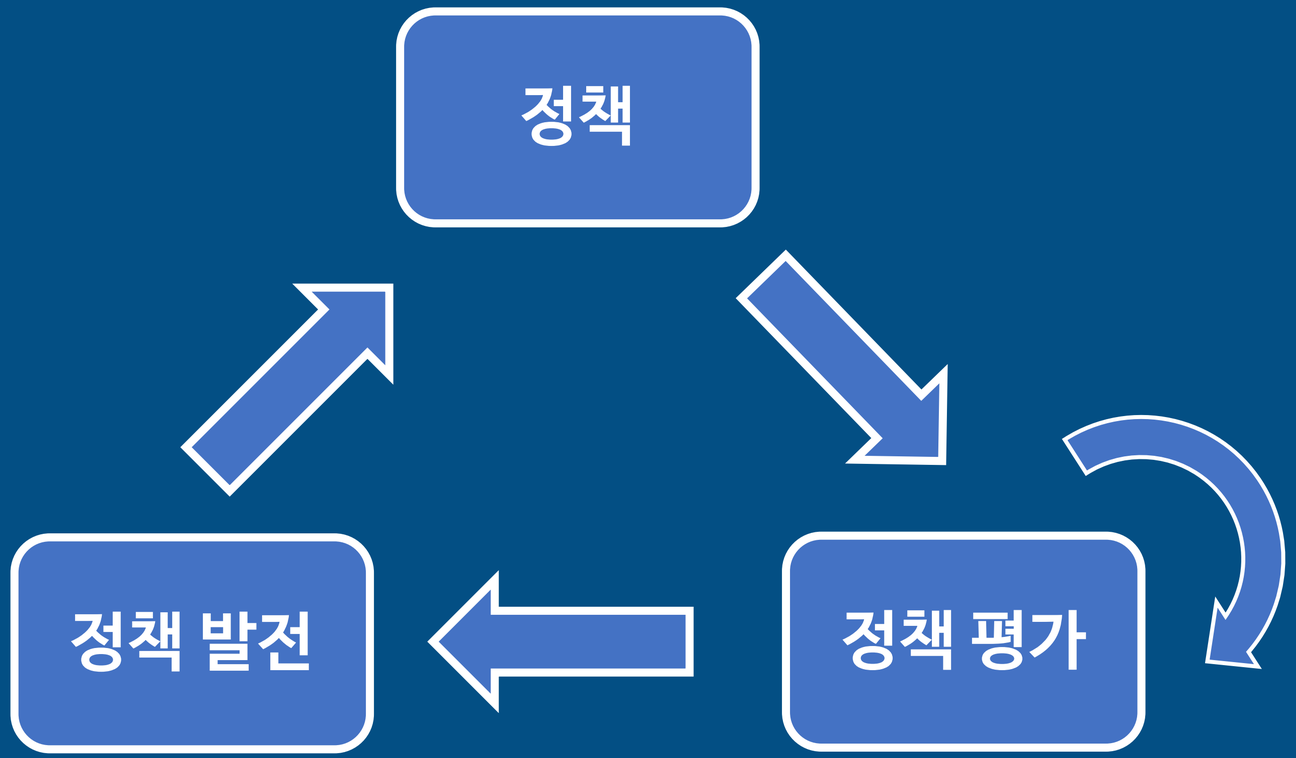
$$v_\pi(s) = E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s]$$

가치함수($v_\pi(s)$)를 기준으로 정책이 얼마나 좋은 지 평가한다.

문제는 정책을 평가하려면 $v_\pi(s)$가 필요한데 이게 뭔 지 모른다.

→ 다이나믹 프로그래밍으로 해결

정책 이터레이션



정책 이터레이션

정책 평가

가치함수 : (지금 받은 보상 + 미래에 받을 보상)의 기대값

가치함수의 정의에 맞게 점화식으로 만들면

$$v_\pi(s) = E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s]$$

정책 평가

기대값 : 사건이 일어날 확률과 사건으로 얻을 이득의 곱의 총합

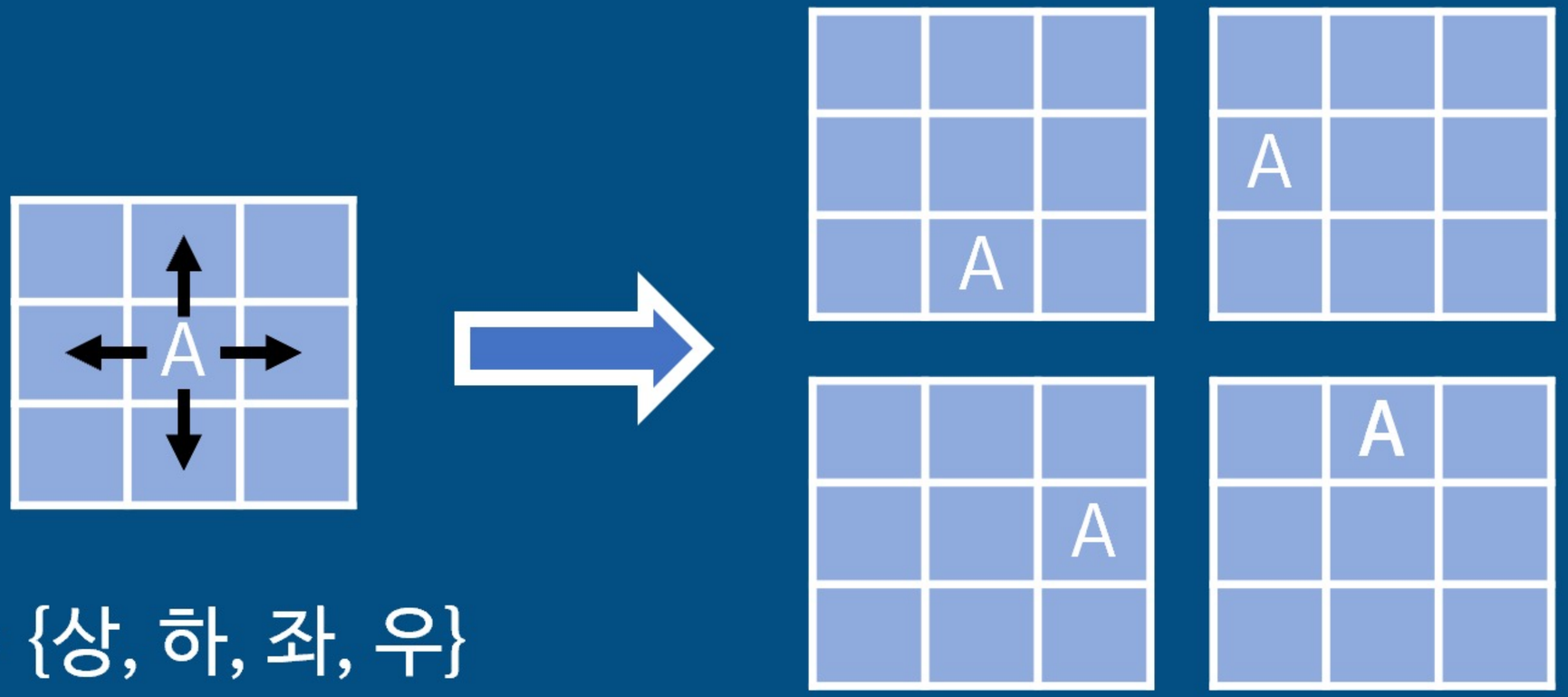
$$v_\pi(s) = E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s]$$

$$v_\pi(s) = \sum_{a \in A} \underbrace{\pi(a|s)}_{\text{확률}} \underbrace{\left(R_{t+1} + \gamma v_\pi(s')\right)}_{\text{이득}}$$

$$v_{k+1}(s) = \sum_{a \in A} \pi(a|s)\left(R_s^a + \gamma v_k(s')\right)$$

정책 평가

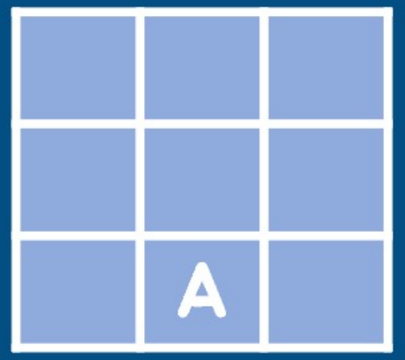
1. 현재 상태 s에서 가능한 행동 A를 통해 다음 상태 s'들을 구한다.



A = {상, 하, 좌, 우}

정책 평가

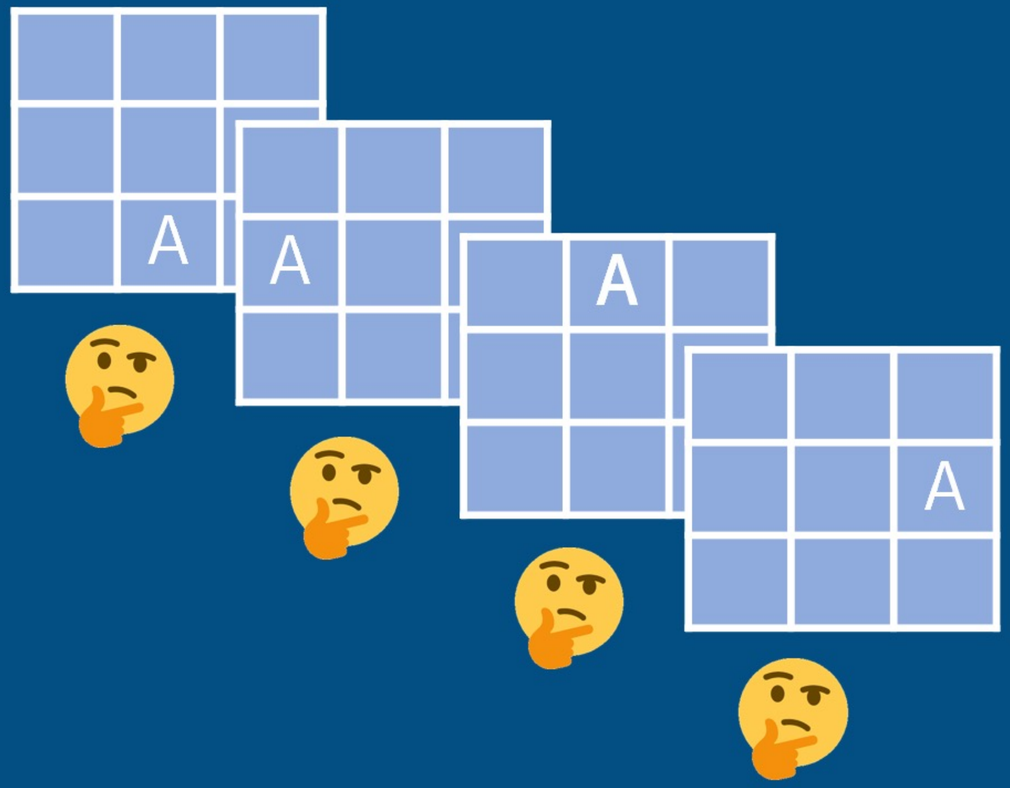
2. k번째(현재) 가치함수(v_k)로,
   다음 상태 s'에 대한 가치($v_k(s')$)를 구한다.
3. 다음 상태에 대한 가치 $v_k(s')$에
   감가율(γ)을 곱하고 그 상태로 가는 행동에
   대한 보상(R_s^a)을 더한다.
4. 위에서 구한 이득에 s에서 s'가 되도록
   행동할 확률, 즉 정책을 곱한다.



$$\pi(a|s)(R_s^a + \gamma v_k(s'))$$

정책 평가

5. 2~4번 과정을 아까 구했던 모든 행동들에 대해 반복하고 더한다.



$$\sum_{a \in A} \pi(a|s)(R_s^a + \gamma v_k(s'))$$

정책 평가

6. 위 과정을 통해 구한 값을 $k+1$번째 가치함수 행렬에 저장한다.
7. 1~6 과정을 모든 $s \in S$에 대해 반복한다.
8. 이 과정을 무한히 반복하면 실제 $v_\pi(s)$에 수렴한다.

$$v_{k+1}(s) = \sum_{a \in A} \pi(a|s)\left(R_s^a + \gamma v_k(s')\right)$$

정책 발전

앞에서 진행했던 정책 평가를 이용해 정책을 발전시킨다.

최초의 정책은 무작위 정책이었다.

정책 평가를 통해 무작위 정책들에 대한 가치를 알았으므로,

큐함수를 이용해 어떤 행동이 좋은 지 알 수 있다.

$$q_\pi(s,a) = E_\pi[R_{t+1} + \underbrace{\gamma v_\pi(S_{t+1})}_{\text{가치}} | S_t = s, A_t = a]$$

정책 발전

$$q_\pi(s,a) = E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a]$$

‘오른쪽 키를 눌렀는데 일정 확률로 왼쪽 이동’과 같은 무작위성이 없다면,
위 큐함수 정의에서 확률이 1이라고 볼 수 있다.

그러므로, 큐함수 정의를 계산 가능한 아래 형태로 바꿀 수 있다.

$$q_\pi(s,a) = R_s^a + \gamma v_\pi(s')$$

정책 발전

현재 상태 s에서 선택 가능한 행동 a들의 $q_\pi(s,a)$를 비교하고,
가장 큰 행동을 선택하도록 새로운 정책을 구하면 된다.

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

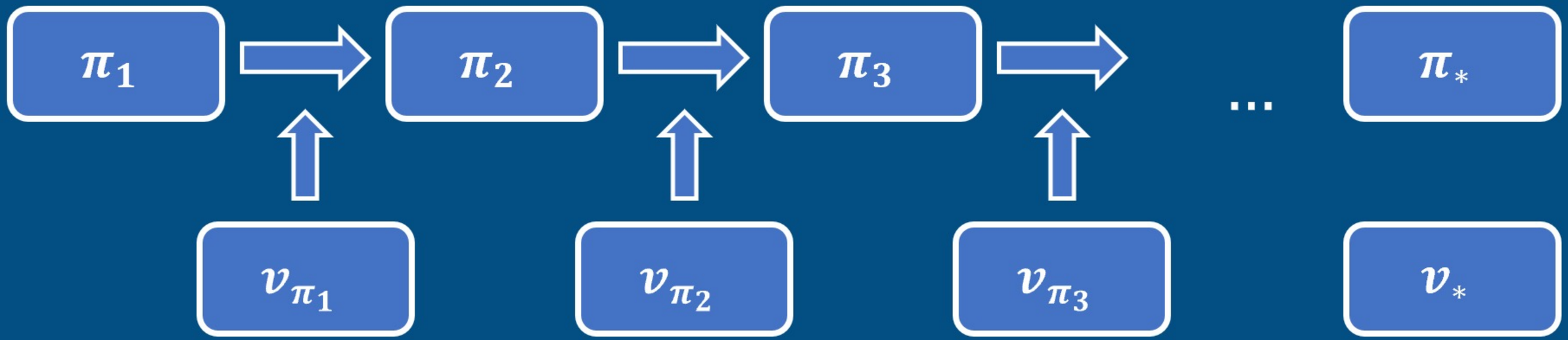
정책 발전

이렇게 근시안적으로 현재에 가장 최적인 해를 구하는 것을
탐욕(Greedy) 알고리즘이라 한다. 그래서 큐함수의 값이
지금 당장 가장 높은 행동을 선택하는 방법을 탐욕 정책 발전이라고 부른다.

이처럼 탐욕 정책 발전을 사용하면 업데이트된 가치함수가
이전보다 무조건 좋거나 같게 되므로, 언젠가는 가치함수가
가장 큰 최적 정책에 수렴한다.

정책 이터레이션?

정책 이터레이션에서는 정책과 가치함수가 명확히 분리된 상태로 발전한다.



정책 이터레이션?

정책이 독립적이므로, 결정적이지 않은 정책이 가능하다.

즉 정책이 확률적으로 여러 행동이 가능한 상태이고,

이를 고려해서 가치함수를 계산하려면 기대값을 구할 필요가 있다.

가치 이터레이션

하지만 현재 정책이 최적이라고 가정하고, 결정적 정책을 적용한다면?

이는 틀린 가정이지만, 이 방법으로 반복적으로 가치함수를 발전시켜

언젠가 최적 정책을 구할 수 있기 때문에 이는 문제가 되지 않는다.

가치 이터레이션

정책 이터레이션에서는 가치함수의 업데이트, 정책의 발전을 모두 다뤘다.

하지만 가치 이터레이션에서는 가치함수의 업데이트만을 다룬다.

그 이유는 가치 이터레이션에서는 가치함수 안에 정책이 내재적으로
포함되어 있어, 가치함수의 업데이트가 정책의 발전을 동반하기 때문이다.

가치 이터레이션

가치 이터레이션은 벨만 최적 방정식을 이용한다.

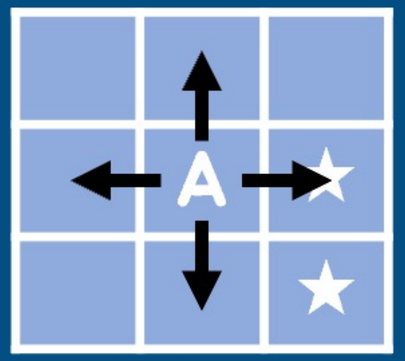
$$v_*(s) = \max_a E[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a]$$

가치 이터레이션에서는 최적 정책이라고 가정했기 때문에 정책 발전이 필요 없고, 가치함수를 업데이트 할 때 정책을 고려할 필요가 없다.

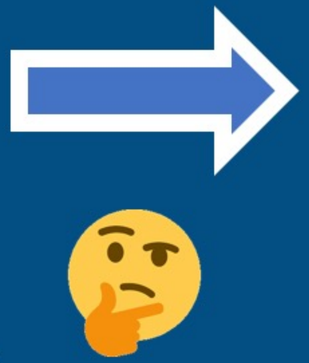
그저 현재 상태에서 얻을 수 있는 이득의 값($R_{t+1} + \gamma v_k(S_{t+1})$) 중 최고의 값으로 업데이트 하면 된다. 이를 가치 이터레이션이라고 한다.

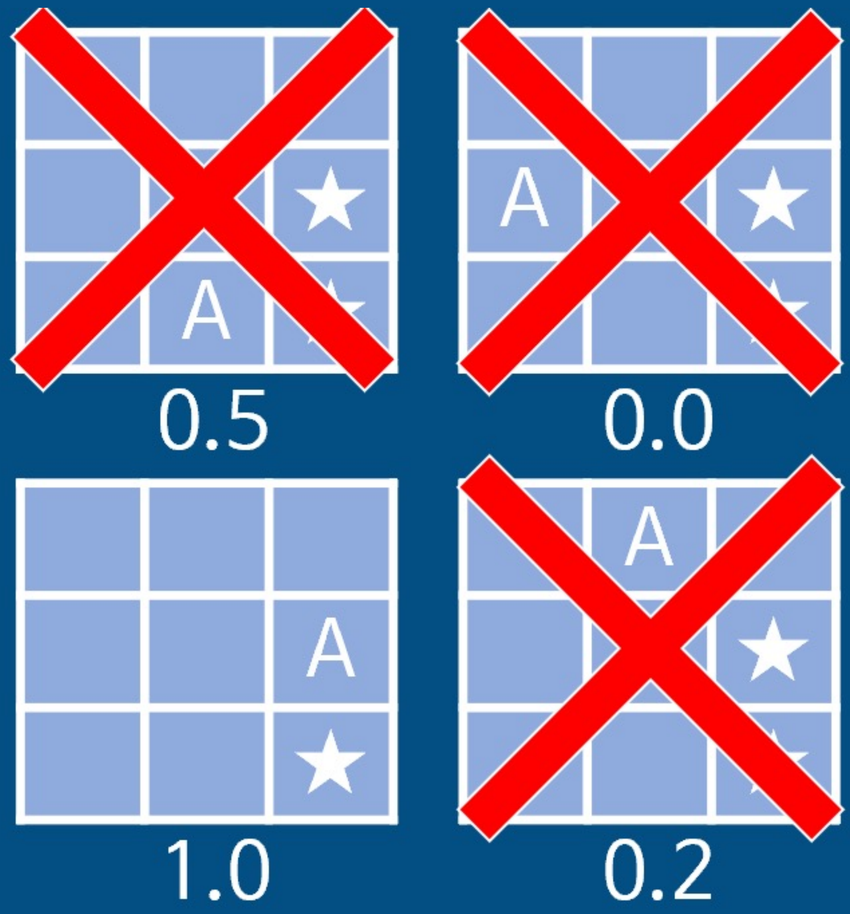
가치 이터레이션

1. 제일 높은 이득을 얻는 행동을 고른다.



A = {상, 하, 좌, 우}





가치 이터레이션

2. 이번에도 무작위성이 없다 가정하면 확률이 1이다.
   그러므로 $v_{k+1}(s) = \max_a(R_{t+1} + \gamma v_k(s'))$로
   가치함수를 업데이트하면 된다.

다이내믹 프로그래밍의 한계

정책 이터레이션, 가치 이터레이션은 벨만 방정식을 이용한 다이내믹 프로그래밍이다.

다이내믹 프로그래밍은 계산을 빠르게 하는 것이지, “학습”을 하는 것은 아니다.

정책 이터레이션, 가치 이터레이션으로도 최적 정책을 찾을 수는 있다. (언젠가는)

그런데 우리가 강화학습을 배우는 이유는?

→ 다이내믹 프로그래밍이 여러 한계를 가지고 있기 때문

다이내믹 프로그래밍의 한계

1. 차원의 저주

설명하면서 다루었던 그리드 월드는 상태가 (x, y)로 표현되는 2차원

크기(N)가 5였으니 $5^2 = 25$, 하지만 3차원? 10차원? … ?

차원의 수가 늘어나면 상태의 수가 지수적으로 증가

→ 계산해야 할 양이 기하급수적으로 늘어난다.

다이내믹 프로그래밍의 한계

2. 환경에 대한 완벽한 정보가 필요

정책 이터레이션과 가치 이터레이션을 다룰 때,
환경의 보상과 상태 변환 확률을 정확히 안다는 가정하에 풀었었다.

하지만 보통은 환경에 대한 정보를 정확히 알 수 없기 때문에,
환경과의 상호작용을 통해 경험을 바탕으로 학습하는 방법론이 등장했다.

→ 그것이 바로 강화학습!

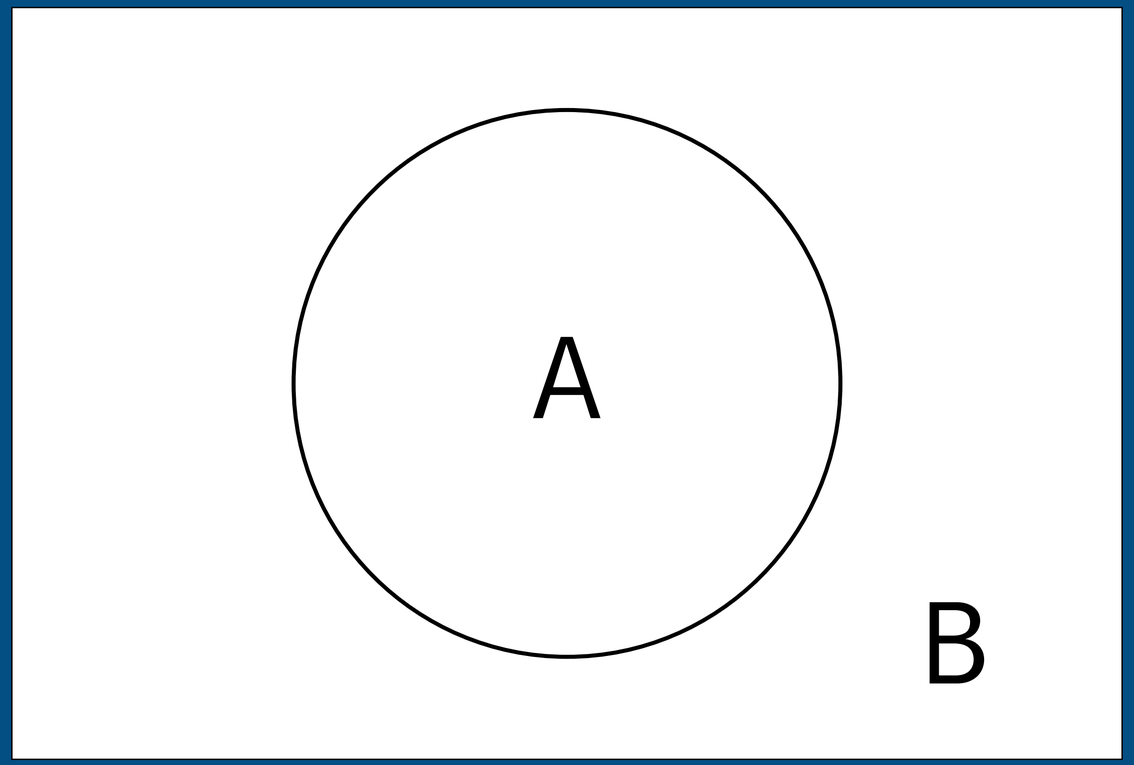
몬테카를로 근사

몬테카를로 : 무작위로 무엇인가를 해본다

근사 : 원래의 값을 추정하는 것

몬테카를로 + 근사 = 무작위로 무엇인가를 해서 원래의 값을 추정한다.

몬테카를로 근사



원의 넓이를 S(A), 직사각형의 넓이를 S(B)라 하면

S(B)와 S(A)/S(B)를 안다면 원의 넓이를 구할 수 있다.

몬테카를로 근사

$\mathrm{S(A)}/\mathrm{S(B)}$를 근사

- 직사각형 위에 무작위로 점을 뿌린다.
- 전체 점 중 원 안에 들어간 점의 개수를 구한다.

무작위로 n개의 점을 뿌렸을 때

$$\frac{\mathrm{S(A)}}{\mathrm{S(B)}} \approx \frac{1}{n}\sum_{i=1}^{n} \mathrm{I}(\mathrm{dot}_i \in \mathrm{A})$$

$n \to \infty$이면 위 두 수식은 같아진다.

몬테카를로 예측

앞의 점 하나하나는 “샘플”
→ 점을 찍는 것은 “샘플링(Sampling)”

강화학습은 샘플링을 통해 참 가치함수의 값을 근사하는 것
→ 에이전트가 환경에서 에피소드를 진행하는 것이 “샘플링”

몬테카를로 예측 : 몬테카를로 근사를 통해 참 가치함수의 값을 근사

몬테카를로 예측

가치함수 : $v_\pi(s) = E_\pi[G_t|S_t = s]$

→ 에피소드를 통해 반환값을 얻으면 참 가치함수의 값을 근사할 수 있다.

여러 에피소드를 통해 근사한 가치함수

$$v_\pi(s) \sim \frac{1}{N(s)} \sum_{i=1}^{N(s)} G_i(s)$$

- $N(s)$: 여러 에피소드 동안 상태 s에 방문한 횟수
- $G_i(s)$: 상태를 방문한 i번째 에피소드에서 s의 반환값

몬테카를로 예측

V_{n+1} : n개의 반환값을 통해 평균을 취한 가치함수

$$V_{n+1} = \frac{1}{n}\sum_{i=1}^{n} G_i = \frac{1}{n}\left(G_n + \sum_{i=1}^{n-1} G_i\right)$$

$$= V_n + \frac{1}{n}(G_n - V_n)$$

몬테카를로 예측

정리하면, 가치함수의 업데이트 식은

$$V(s) \leftarrow V(s) + \alpha\big(G(s) - V(s)\big)$$

위 수식에서

- $G(s) - V(s)$: 오차
- α : 스탭 사이즈, 오차의 얼마를 가지고 업데이트 할 지를 의미

몬테카를로 예측

몬테카를로 예측에서

- 가치함수를 업데이트하며 도달하려는 목표는 반환값
  → 한 번에 목표점으로 가는 것이 아니고 스탭 사이즈를 곱한 만큼만 간다.
- 에이전트는 에피소드 동안 경험했던 모든 상태에 대해 가치함수를 업데이트 한다.
- 가치함수를 업데이트 하기 위해서는 에피소드가 끝날 때까지 기다려야 한다.
  → 에피소드의 끝이 없거나 에피소드의 길이가 긴 경우에는 적합하지 않다.

시간차 예측

- 실시간으로 가치함수를 업데이트
- 다른 형태의 가치함수를 사용

$$v_\pi(s) = E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s]$$

시간차 예측

시간차 예측에서 가치함수의 업데이트

$$V(S_t) \leftarrow V(S_t) + \alpha(R + \gamma V(S_{t+1}) - V(S_t))$$

- 에이전트는 현재의 상태 S_t에서 행동을 하나 선택
- 환경으로 부터 R을 받고 다음 상태 S_{t+1}을 알게 된다.
  → $R + \gamma V(S_{t+1})$을 목표로 가치함수를 업데이트

살사

시간차 제어 = 시간차 예측 + 탐욕 정책

탐욕 정책 : $\pi(s) = \mathrm{argmax}_{a \in A} Q(s, a)$

- 현재 상태의 큐함수를 이용했기 때문에 환경의 모델을 몰라도 된다.
- 업데이트 하는 대상은 큐함수

살사

시간차 제어의 업데이트 수식

$$\mathrm{Q}(\mathrm{S}_t, \mathrm{A}_t) \leftarrow \mathrm{Q}(\mathrm{S}_t, \mathrm{A}_t) + \alpha\left(R_{t+1} + \gamma \mathrm{Q}(\mathrm{S}_{t+1}, \mathrm{A}_{t+1}) - \mathrm{Q}(\mathrm{S}_t, \mathrm{A}_t)\right)$$

여기서 $[\mathrm{S}_t, \mathrm{A}_t, \mathrm{R}_{t+1}, \mathrm{S}_{t+1}, \mathrm{A}_{t+1}]$이 하나의 샘플 → SARSA라고 부름

살사

초기의 에이전트에선 탐욕 정책으로 인해 학습이 잘못될 가능성이 크다.
→ 에이전트에게 충분한 경험을 통해 최적에 가까운 큐함수를 가져야 한다.

대안 : ϵ-탐욕 정책

$$\pi(s) = \begin{cases} a^* = \text{argmax}_{a \in A} Q(s, a), & 1 - \epsilon \\ a \neq a^* & , \epsilon \end{cases}$$

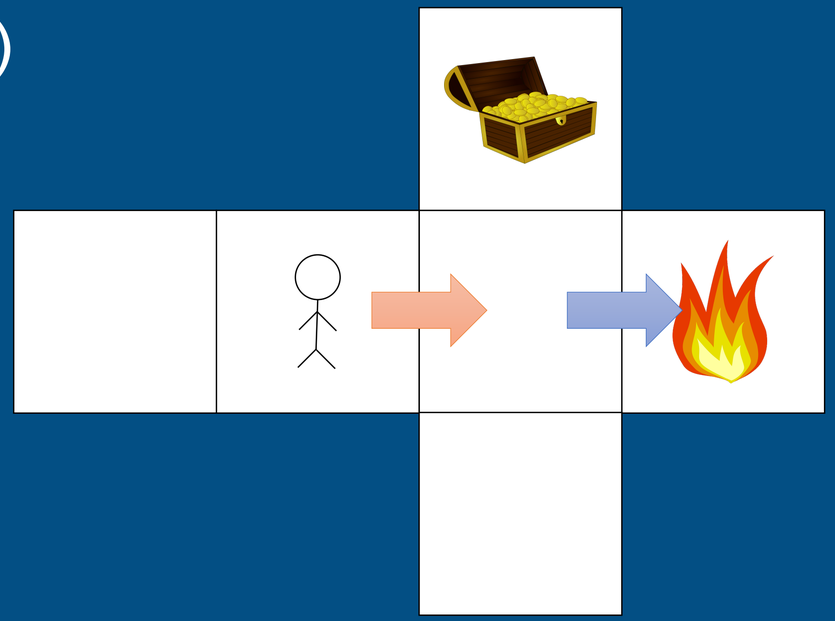
- $1 - \epsilon$의 확률로 현재 상태에서 가장 큰 큐함수의 값을 갖는 행동을 선택 = 탐욕 정책
- ϵ의 확률로 엉뚱한 행동을 선택 = 탐험

살사

살사의 문제점 : 살사는 자신이 행동하는 대로 학습하는 시간차 제어
→ 최적 정책을 학습하지 못하고 잘못된 정책을 학습할 수 있음.

예시

- 에이전트가 현재 상황에서 탐욕 정책에 따라 이동(빨간색)
- 그 이후 탐험을 통해 엉뚱한(파란색) 행동을 함
- 빨간색 행동의 큐함수 값이 낮아지게 됨
  → 잘못된 정책을 학습하게 됨



큐러닝

에이전트가 다음 상태 s'을 알게 되면,
그 상태에서 가장 큰 큐함수를 이용해 업데이트
→ 다음 상태에서 어떤 행동을 했는 지와 상관 없이 업데이트

큐러닝을 통한 큐함수 업데이트

$$\mathrm{Q}(\mathrm{S}_t, \mathrm{A}_t) \leftarrow \mathrm{Q}(\mathrm{S}_t, \mathrm{A}_t) + \alpha\left(R_{t+1} + \gamma \max_{\mathrm{a}'} \mathrm{Q}(S_{t+1}, \mathrm{a}') - \mathrm{Q}(\mathrm{S}_t, \mathrm{A}_t)\right)$$

감사합니다!

스터디 듣느라 고생 많았습니다.