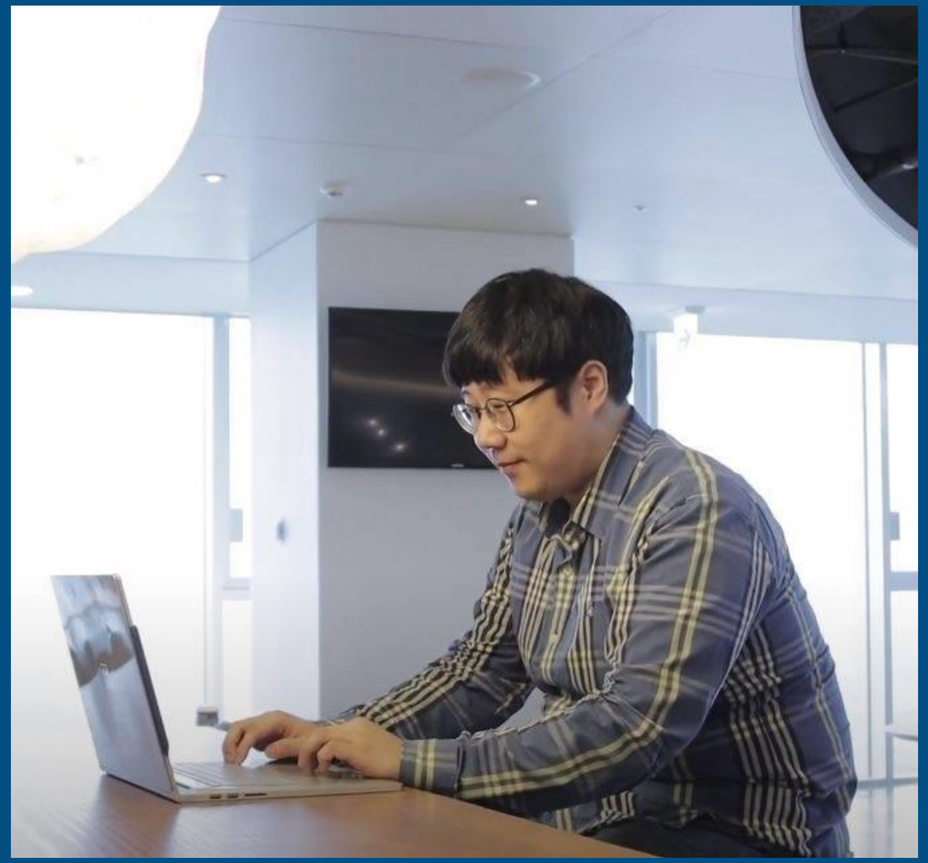
2021-2 한양대학교 HAI

강화학습 부트캠프

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
  - 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)
  - 파이썬과 케라스로 배우는 강화학습 (위키북스, 2020)
  - 바닥부터 배우는 강화 학습 (영진닷컴, 2020)

스터디 진행 계획

- Week 1 (9/8)
  - Review about the Basic Knowledge of Reinforcement Learning
    - MDP (Markov Decision Process)
    - The Bellman Equation
    - SARSA & Q-Learning
    - Policy Gradient
    - Deep Q-Network (DQN)
    - Actor-Critic

스터디 진행 계획

- Week 2 (9/13)
  - DQN Extensions #1
    - N-step DQN
    - Double DQN
    - Noisy Network
    - Prioritized Experience Replay (PER)
- Week 3 (9/27)
  - DQN Extensions #2
    - Dueling DQN
    - Categorical DQN
    - Rainbow

스터디 진행 계획

- Week 4 (10/4)
  - Actor-Critic Extensions
    - Advantage Actor-Critic (A2C)
    - Asynchronous Advantage Actor-Critic (A3C)
      - Data Parallelism
      - Gradient Parallelism
- Week 5 (11/1)
  - Example #1: Stock Trading

스터디 진행 계획

- Week 6 (11/8)
  - Continuous Action Space
    - Advantage Actor-Critic (A2C)
    - Deep Deterministic Policy Gradient (DDPG)
    - Distributed Distributional DDPG (D4PG)
- Week 7 (11/15)
  - Trust Regions #1
    - Trust Region Policy Optimization (TRPO)
    - Proximal Policy Optimization (PPO)

스터디 진행 계획

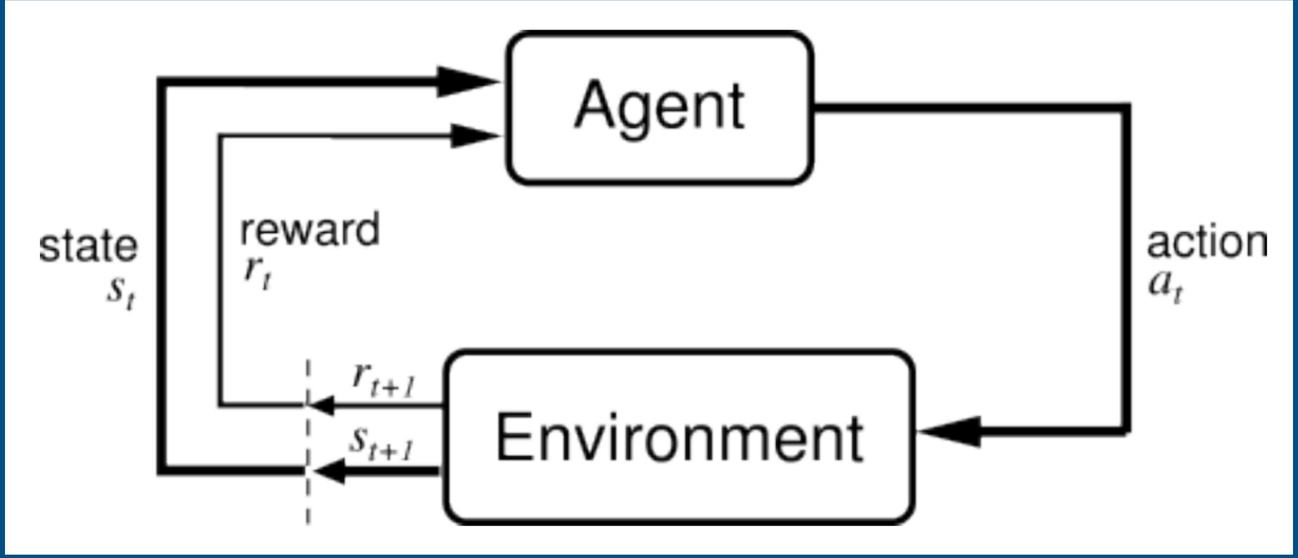
- Week 8 (11/22)
  - Trust Regions #2
    - Averaged Stochastic K-Timestep Trust Region (ACKTR)
    - Soft Actor-Critic (SAC)
- Week 9 (1/3)
  - Example #2: Robotics
- Week 10 (1/10)
  - Imagination
    - Interpretable Inference for Autonomous Agents (I2A)

스터디 진행 계획

- Week 11 (1/17)
  - Multi-Agent RL
    - Multi-Agent Deep Deterministic Policy Gradient (MADDPG)
    - Multi-Agent Twin Delayed Deep Deterministic Policy Gradient (MATD3)
    - Multi-Agent Proximal Policy Optimization (MAPPO)
    - Multi-Agent Soft Actor-Critic (MASAC)
    - Monotonic Value Function Factorization for Deep Multi-Agent Reinforcement Learning (QMIX)
- Week 12 (1/24)
  - Example #3: Rubik's Cube

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



MDP

강화 학습은 순차적으로 행동을 계속 결정해야 하는 문제를 푸는 것

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

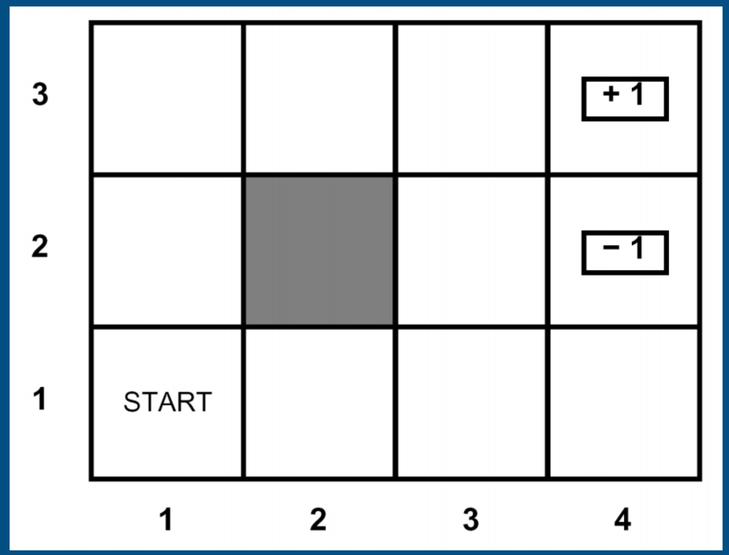
- MDP의 구성 요소
  - 상태
  - 행동
  - 보상 함수
  - 상태 변환 확률 (State Transition Probability)
  - 감가율 (Discount Factor)

상태

에이전트가 관찰 가능한 상태의 집합 : S

- 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 $S_t = (1, 3)$
- 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)”



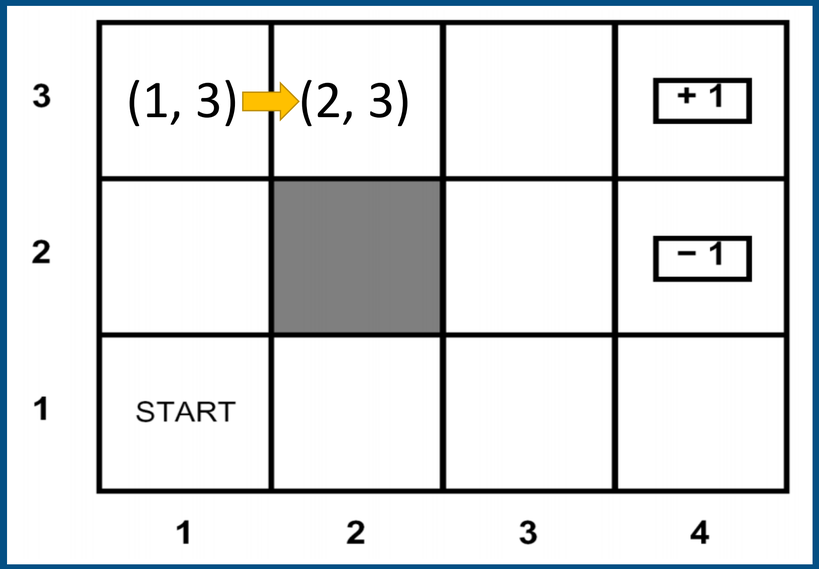
행동

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.
- 오른쪽 환경에서 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.

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

시간에 따라서 감가하는 비율

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

- 에이전트는 항상 현재 시점에서 판단을 내리기 때문에
  현재에 가까운 보상일수록 더 큰 가치를 갖는다.
  - 보상의 크기가 100일 때, 에이전트가 현재에 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.
- 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”

가치함수

현재 상태에서 미래에 받을 것이라 기대하는 보상의 합

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

- 반환값 (Return) : 에이전트가 실제로 환경을 탐험하며 받은 보상의 합 (감가율 적용)

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

- 반환값으로 나타내는 가치함수

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

- 가치함수로 표현하는 가치함수의 정의

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

큐함수

어떤 상태에서 어떤 행동이 얼마나 좋은지 알려주는 함수 (행동 가치함수)

- 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.

벨만 기대 방정식

현재 상태의 가치함수와 다음 상태의 가치함수 사이의 관계를 나타낸 식

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

- 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 최적 방정식

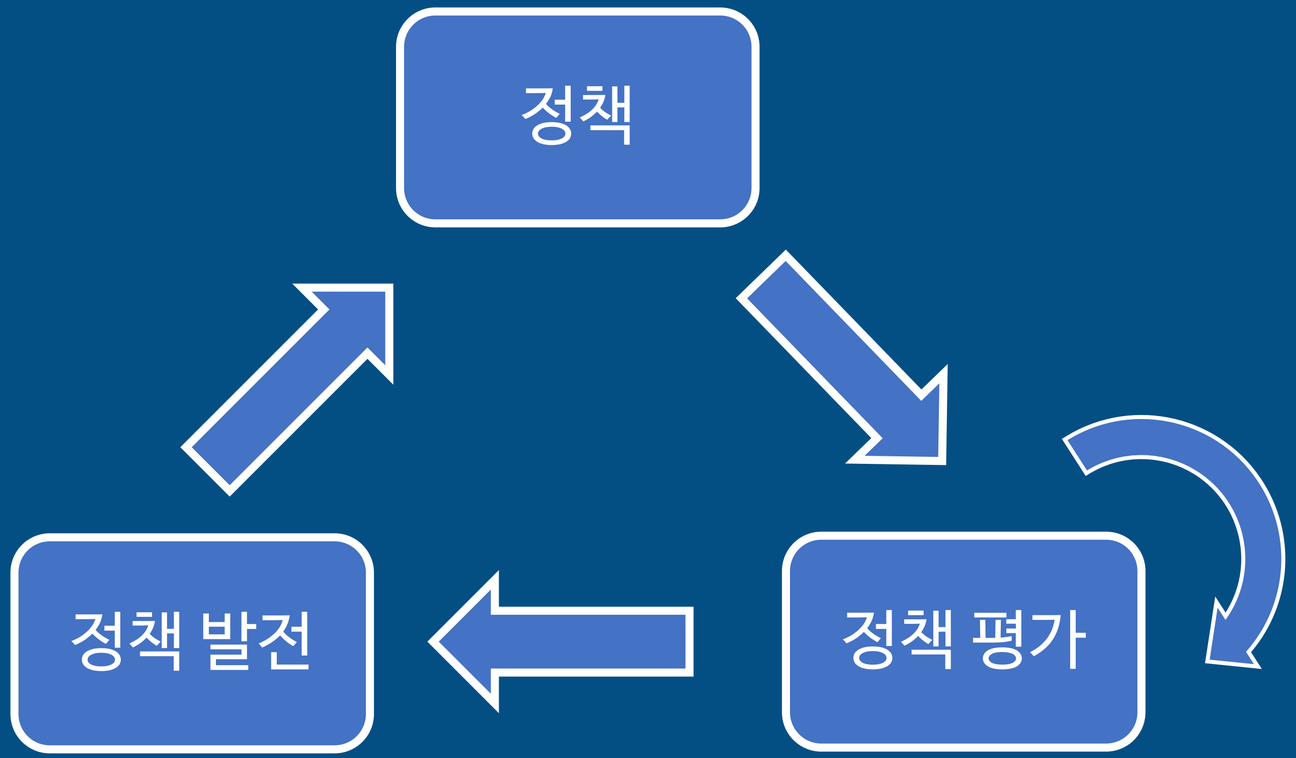
현재 상태의 최적 가치함수와 다음 상태의 최적 가치함수 사이의 관계를 나타낸 식

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

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

- 참 가치함수 : “어떤 정책”을 따라서 움직였을 경우에 받게 되는 보상에 대한 참값
- 최적 가치함수 : 수많은 정책 중에서 가장 높은 보상을 주는 가치함수

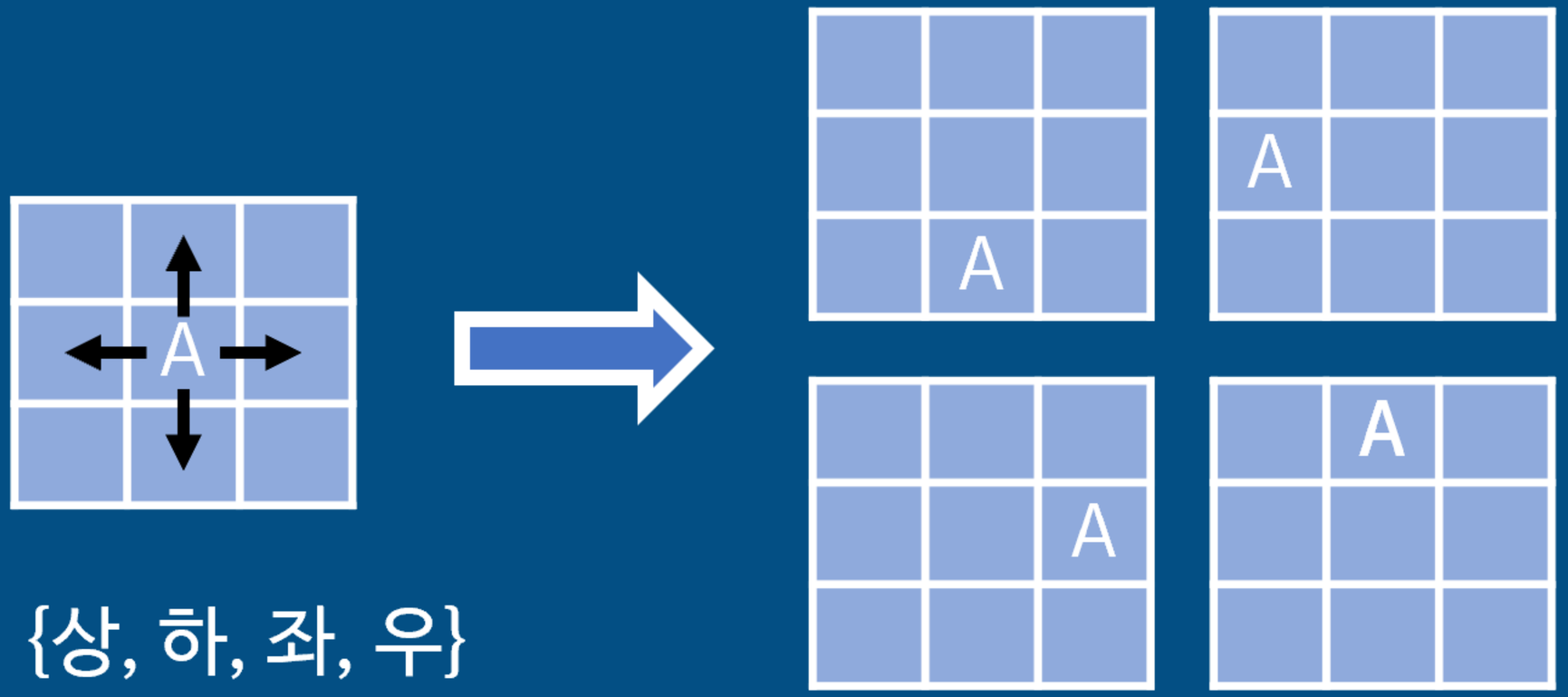
정책 이터레이션



정책 이터레이션

정책 평가

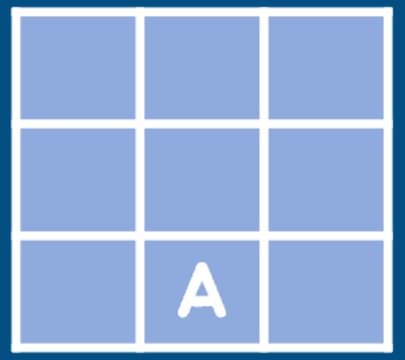
1. 현재 상태 s에서 가능한 행동 A를 통해 다음 상태 s'들을 구한다.



A = {상, 하, 좌, 우}

정책 평가

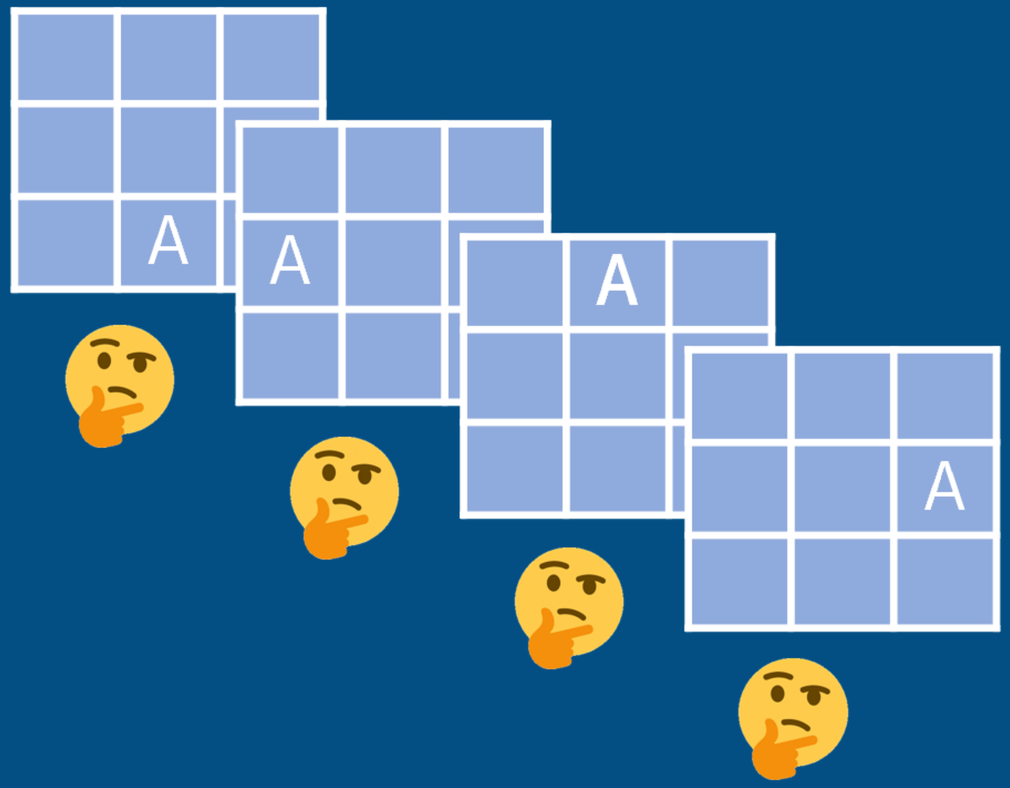
2. (현재) k번째 가치함수(v_k)로,
   다음 상태 s'에 대한 가치($v_k(s')$)를 구한다.
3. 다음 상태에 대한 가치 $v_k(s')$에
   감가율(γ)을 곱하고 그 상태로 가는 행동에
   대한 보상(R_s^a)을 더한다.
4. 위에서 구한 이득에 s에서 s'가 되도록
   행동할 확률, 즉 정책을 곱한다.



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

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.

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.

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

- 상태 변환 확률이 1이라면, 계산 가능한 아래 형태로 바꿀 수 있다.

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

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

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

가치 이터레이션

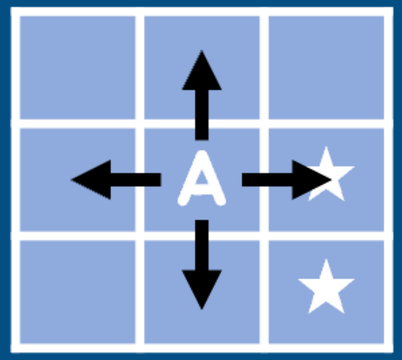
- 정책 이터레이션에서는 가치함수의 업데이트, 정책의 발전을 모두 다뤘다.
  하지만 가치 이터레이션에서는 가치함수의 업데이트만을 다룬다.
- 그 이유는 가치 이터레이션에서는 가치함수 안에 정책이 내재적으로 포함되어 있어,
  가치함수의 업데이트가 정책의 발전을 동반하기 때문이다.
- 가치 이터레이션은 벨만 최적 방정식을 이용한다.

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

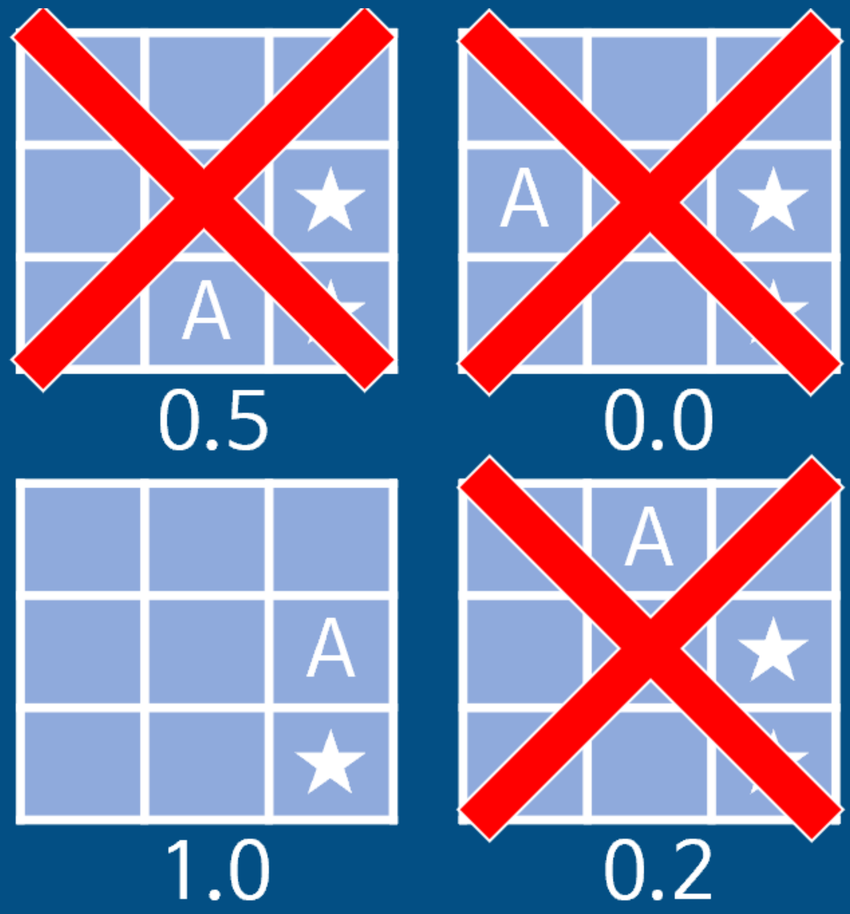
- 가치 이터레이션에서는 최적 정책이라고 가정했기 때문에 정책 발전이 필요 없고,
  가치함수를 업데이트 할 때 정책을 고려할 필요가 없다.

가치 이터레이션

1. 제일 높은 이득을 얻는 행동을 고른다.



A = {상, 하, 좌, 우}



가치 이터레이션

2. 무작위성이 없다 가정하면 확률은 1이다. 그러므로

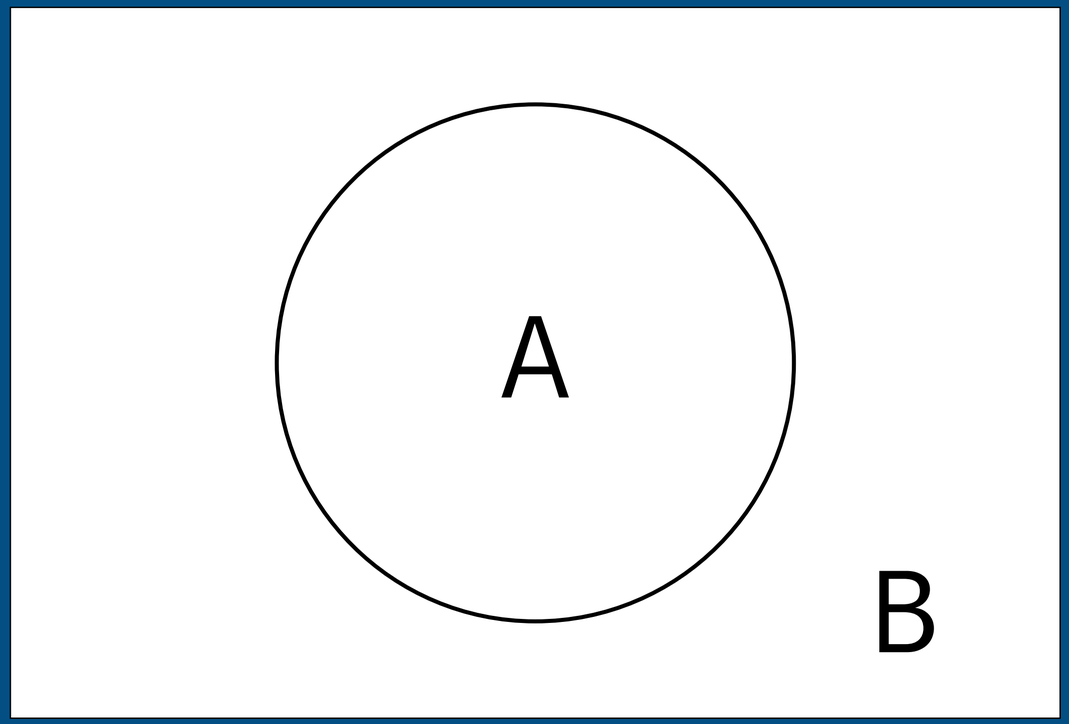
$v_{k+1}(s) = \max_a(R_{t+1} + \gamma v_k(s'))$로 가치함수를 업데이트하면 된다.

몬테카를로 근사

무작위로 무엇인가를 해서 원래의 값을 추정한다.

- 몬테카를로 : 무작위로 무엇인가를 해본다.
  근사 : 원래의 값을 추정한다.
- 원의 넓이 S(A), 직사각형의 넓이 S(B)일 때

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



몬테카를로 예측

몬테카를로 근사를 통해 참 가치함수의 값을 근사하는 것

- 강화학습은 샘플링을 통해 참 가치함수의 값을 근사한다.
  → 에이전트가 환경에서 에피소드를 진행하는 것이 “샘플링”
- 가치함수 : $\mathrm{v}_\pi(\mathrm{s}) = \mathrm{E}_\pi[\mathrm{G}_t|\mathrm{S}_t = \mathrm{s}]$
- 여러 에피소드를 통해 근사한 가치 함수

$$\mathrm{v}_\pi(\mathrm{s}) \sim \frac{1}{\mathrm{N}(\mathrm{s})} \sum_{i=1}^{\mathrm{N}(\mathrm{s})} \mathrm{G}_i(\mathrm{s})$$

- $\mathrm{N}(\mathrm{s})$: 여러 에피소드 동안 상태 s에 방문한 횟수
- $\mathrm{G}_i(\mathrm{s})$: 상태를 방문한 i번째 에피소드에서 s의 반환값

몬테카를로 예측

- V_{n+1} : n개의 반환값을 통해 평균을 취한 가치함수

$$\mathrm{V}_{n+1} = \frac{1}{n}\sum_{i=1}^{n} \mathrm{G}_i = \frac{1}{n}\left(\mathrm{G}_n + \sum_{i=1}^{n-1} \mathrm{G}_i \right)$$

$$= \mathrm{V}_n + \frac{1}{n}(\mathrm{G}_n - \mathrm{V}_n)$$

- 가치함수의 업데이트 식

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

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

시간차 예측

실시간으로 가치함수를 업데이트하는 방법

- 다른 형태의 가치함수를 사용

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

- 시간차 예측에서 가치함수의 업데이트 식

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

  - 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 : $\pi(\mathrm{s}) = \mathrm{argmax}_{\mathrm{a}\in\mathrm{A}}\mathrm{Q}(\mathrm{s},\mathrm{a})$
  - 현재 상태의 큐함수를 이용했기 때문에 환경의 모델을 몰라도 된다. (업데이트 하는 대상은 큐함수)
- 시간차 제어에서 큐함수의 업데이트 식

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

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

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

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

큐러닝

살사의 한계를 극복하기 위해 오프폴리시 시간차 제어를 사용한 방법

- 온폴리시(On-Policy) : 행동하는 정책과 학습하는 정책이 같음
  오프폴리시(Off-Policy) : 행동하는 정책과 학습하는 정책이 다름
- 에이전트가 다음 상태 s'을 알게 되면, 그 상태에서 가장 큰 큐함수를 이용해 업데이트한다.
  → 다음 상태에서 어떤 행동을 했는 지와 상관 없이 업데이트
- 큐러닝에서 큐함수의 업데이트 식

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

딥살사

살사 + 심층 신경망

- 큐함수를 심층 신경망으로 근사한다.
- 살사의 큐함수 업데이트 식

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right)$$

- 정답에 해당하는 식은 $R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$이고, 예측에 해당하는 식은 $Q(S_t, A_t)$이다.
- 오차 함수 : MSE 사용

$$\text{MSE} = (\text{정답} - \text{예측})^2$$

$$= \left(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right)^2$$

폴리시 그레디언트

- 강화학습 알고리즘의 종류
  - 가치 기반 강화학습 (Value-based Reinforcement Learning)
  - 정책 기반 강화학습 (Policy-based Reinforcement Learning)
- 정책 기반 강화학습에서는 상태에 따라 바로 행동을 선택한다.
  가치함수를 토대로 행동을 선택하지 않고, 대신 정책을 직접적으로 근사한다.
- 이때 활성화 함수로 Softmax를 사용한다.
  (가장 최적의 행동을 선택하는 분류 문제로 생각할 수 있다.)

$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

폴리시 그레디언트

- 폴리시 그레디언트의 업데이트 식

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\theta) \approx \theta_t + \alpha [\nabla_\theta \log \pi_\theta(a|s)\, q_\pi(s,a)]$$

- 에이전트에 가치함수나 큐함수가 없기 때문에 $q_\pi(s, a)$를 구할 수 없다는 문제가 있다.
- 목표함수의 미분값 $\nabla_\theta J(\theta)$를 잘 근사하는 게 중요하다. 가장 고전적인 방법으로는 큐함수를 반환값 G_t로 대체하는 방법이 있는데, 이를 REINFORCE 알고리즘이라고 한다.
- REINFORCE 알고리즘의 업데이트 식

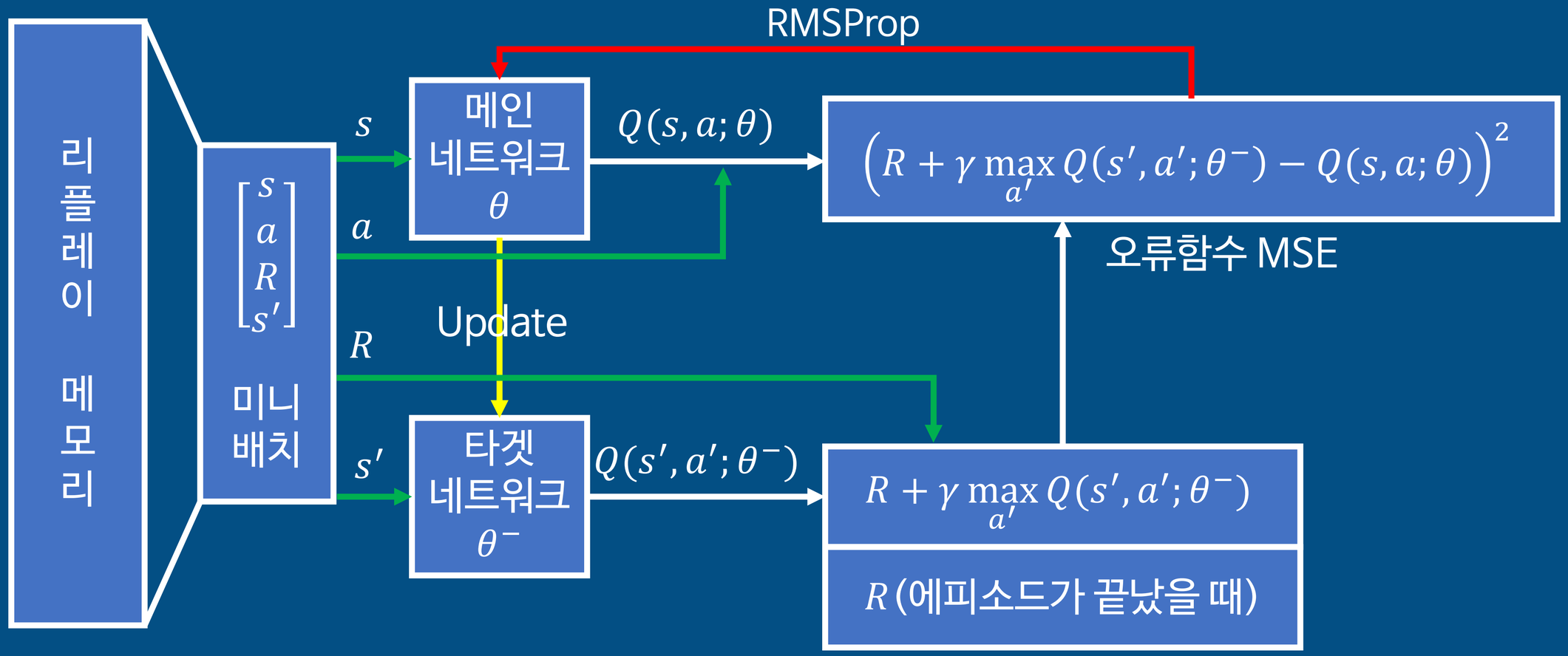
$$\theta_{t+1} \approx \theta_t + \alpha [\nabla_\theta \log \pi_\theta(a|s) G_t]$$

- 오차 함수 : 크로스 엔트로피 사용

$$\log \pi_\theta(a|s) G_t$$

DQN

큐러닝 + 심층 신경망



DQN

- DQN의 특징
  - 오프폴리시(Off-Policy)
  - 리플레이 메모리 + 미니배치
  - 타겟 신경망
- 온폴리시 : 학습하는 정책과 행동을 고르는 정책이 항상 같아야 한다.
  따라서 정책을 업데이트하면 과거의 경험들을 학습에 이용 불가능해 비효율적이다.
- 오프폴리시 : 학습하는 정책과 행동을 고르는 정책이 달라도 된다.
  따라서 과거에 경험한 에피소드들도 학습에 계속해서 이용 가능하다.

DQN

- 리플레이 메모리 + 미니배치
  - 환경에서 받은 $[s, a, R, s']$을 저장한다.
  - 받은 s'를 새로운 s로 에이전트에게 전달해 에피소드를 계속 진행시킨다.
  - 리플레이 메모리에 저장되어 있는 $[s, a, R, s']$ 집합에서 일부를 무작위로 샘플링한 뒤 에이전트를 학습시킨다.
  - 리플레이 메모리 크기 이상으로 데이터가 추가되면 오래된 순서대로 지워준다.
- 타겟 신경망
  - 기존 오류 함수는 신경망이 스스로 목표를 만들어 내기 때문에 신경망이 업데이트될 때
    목표가 되는 정답 부분이 계속 변하고, 그 결과 학습이 굉장히 불안하게 이루어진다.

$$\text{MSE} = (\text{정답} - \text{예측})^2 = \left(R_{t+1} + \gamma \max_{a'} Q(s', a'; \theta) - Q(s, a; \theta)\right)^2$$

  - 따라서 θ^- 를 매개변수로 갖는 타겟 신경망을 추가한다.

$$\text{MSE} = (\text{정답} - \text{예측})^2 = \left(R_{t+1} + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\right)^2$$

  - 타겟 신경망은 일정 시간동안 그대로 유지되며 정답을 만들어내다가, 에피소드가 끝날 때마다 업데이트된다.

Actor-Critic

REINFORCE의 장점과 DQN의 장점을 조합해
행동을 취하는 동작과 정책을 비판하는 동작을 교대로 수행하는 알고리즘

- 액터(Actor) : 직접적인 보상에 기반해 정책을 결정한다.
- 크리틱(Critic) : 환경 상태의 가치 추정에 비해 우리의 정책이 얼마나 좋은지 알려준다.
- 정책 반복의 구조를 사용해 학습한다.

$$\pi_0 \rightarrow v_{\pi_0} \rightarrow \pi_1 \rightarrow v_{\pi_1} \rightarrow \pi_2 \rightarrow \cdots \rightarrow \pi_* \rightarrow v_*$$

  - 정책 평가 : 가치 신경망을 이용해 정책을 평가
  - 정책 발전 : 정책 신경망의 업데이트
- 오류 함수

$$L_{\text{actor}} = \big(R + \gamma v(s') - v(s)\big) \log \pi_\theta(a|s)$$

$$L_{\text{critic}} = \big(R + \gamma v(s') - v(s)\big)^2$$

감사합니다!

스터디 듣느라 고생 많았습니다.