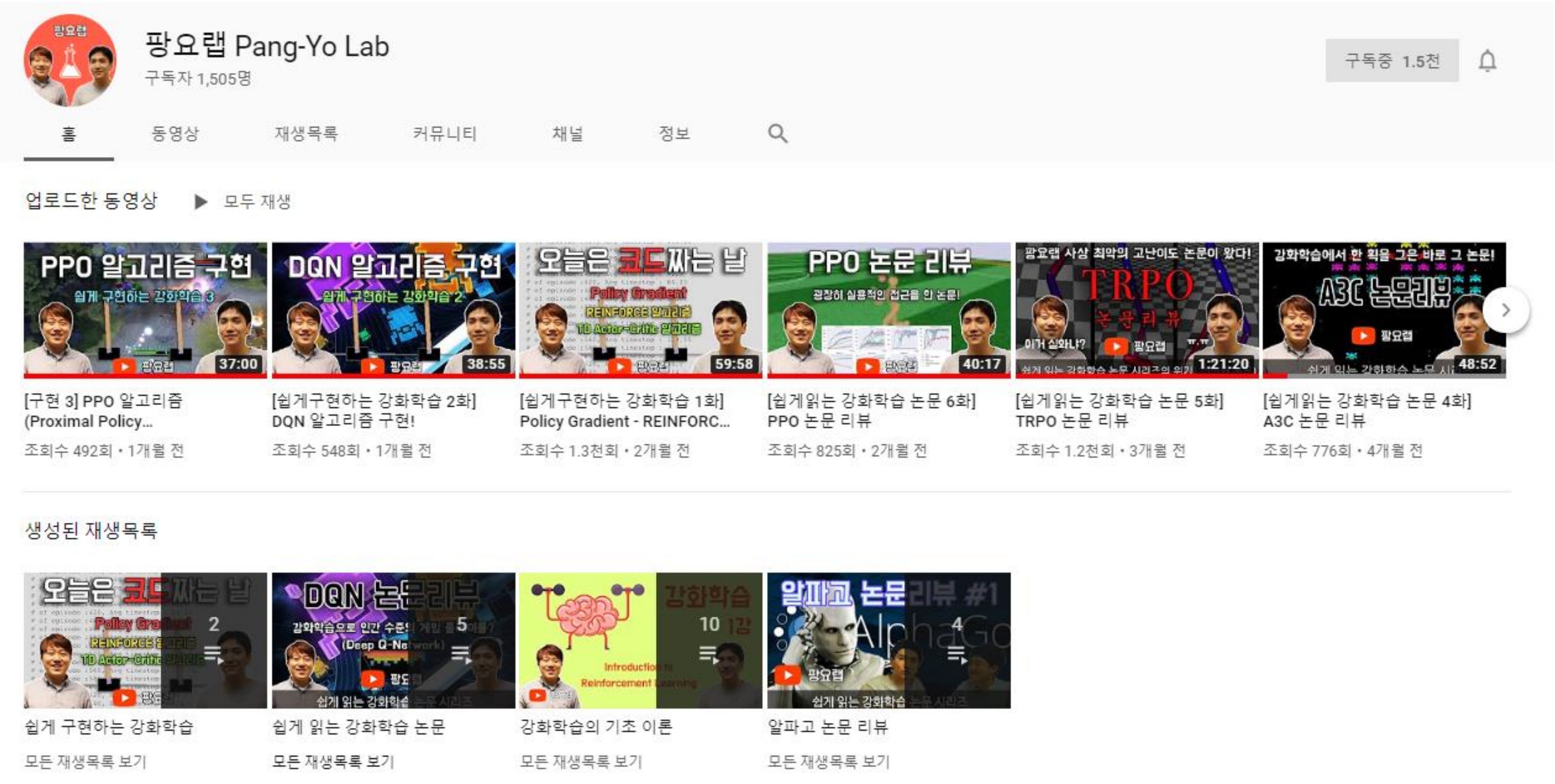
강화학습 입문하기

노승은
엔씨소프트 Game AI 랩
2019.07.12

Slides from David Silver

https://www.youtube.com/watch?v=_KL5iU_nXEs&feature=youtu.be


팡요랩 Pang-Yo Lab
구독자 1,505명
구독중 1.5천
홈
동영상
재생목록
커뮤니티
채널
정보
업로드한 동영상
모두 재생
PPO 알고리즘 구현
쉽게 구현하는 강화학습 3
37:00
[구현 3] PPO 알고리즘 (Proximal Policy...
조회수 492회 • 1개월 전
DQN 알고리즘 구현
쉽게 구현하는 강화학습 2
38:55
[쉽게구현하는 강화학습 2화] DQN 알고리즘 구현!
조회수 548회 • 1개월 전
오늘은 코드짜는 날
Policy Gradient
REINFORCE 알고리즘
TD Actor-Critic 알고리즘
59:58
[쉽게구현하는 강화학습 1화] Policy Gradient - REINFORC...
조회수 1.3천회 • 2개월 전
PPO 논문 리뷰
40:17
[쉽게읽는 강화학습 논문 6화] PPO 논문 리뷰
조회수 825회 • 2개월 전
팡요랩 사상 최악의 고난이도 논문이 왔다!
TRPO
논문리뷰
1:21:20
[쉽게읽는 강화학습 논문 5화] TRPO 논문 리뷰
조회수 1.2천회 • 3개월 전
A3C 논문리뷰
48:52
[쉽게읽는 강화학습 논문 4화] A3C 논문 리뷰
조회수 776회 • 4개월 전
생성된 재생목록
쉽게 구현하는 강화학습
모든 재생목록 보기
DQN 논문리뷰
쉽게 읽는 강화학습 논문
모든 재생목록 보기
Introduction to Reinforcement Learning
강화학습의 기초 이론
모든 재생목록 보기
알파고 논문리뷰 #1
AlphaGo
알파고 논문 리뷰
모든 재생목록 보기

1. 강화 학습 인트로

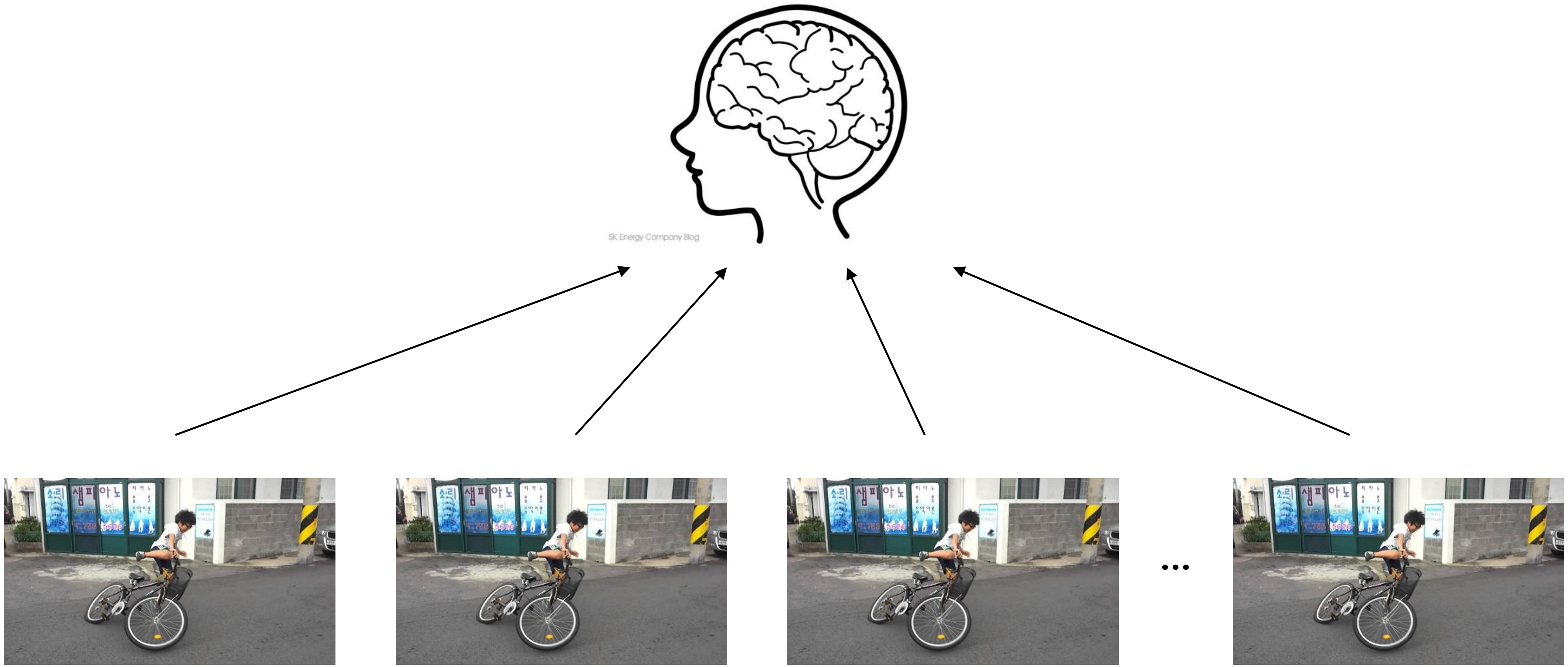
(1) 지도 학습과 강화 학습
(2) 순차적 의사 결정 문제
(3) 보상

자전거 배우기

지도를 통한 학습

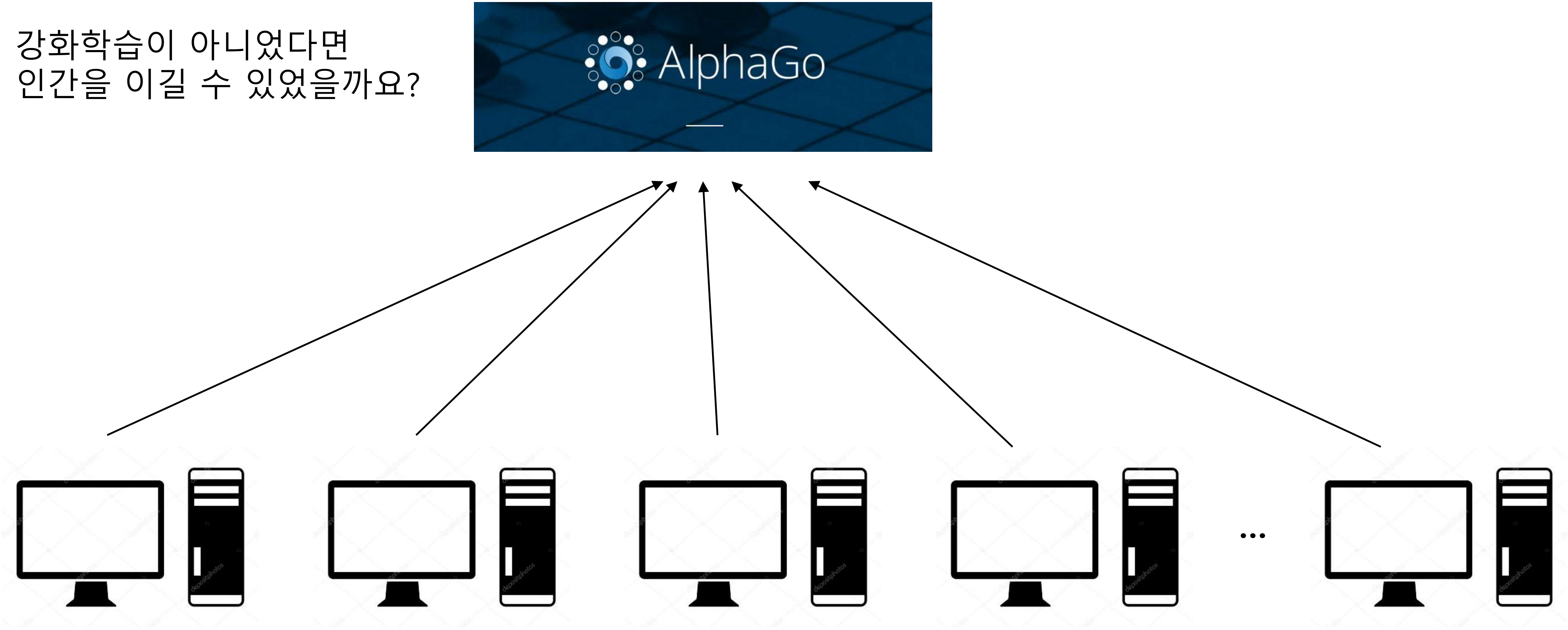
Trial & Error를 통한 학습

경험을 여러명이 나눠서 쌓고, 각 경험에서의 지혜를 한 데로 모을 수 있다면?



알파고

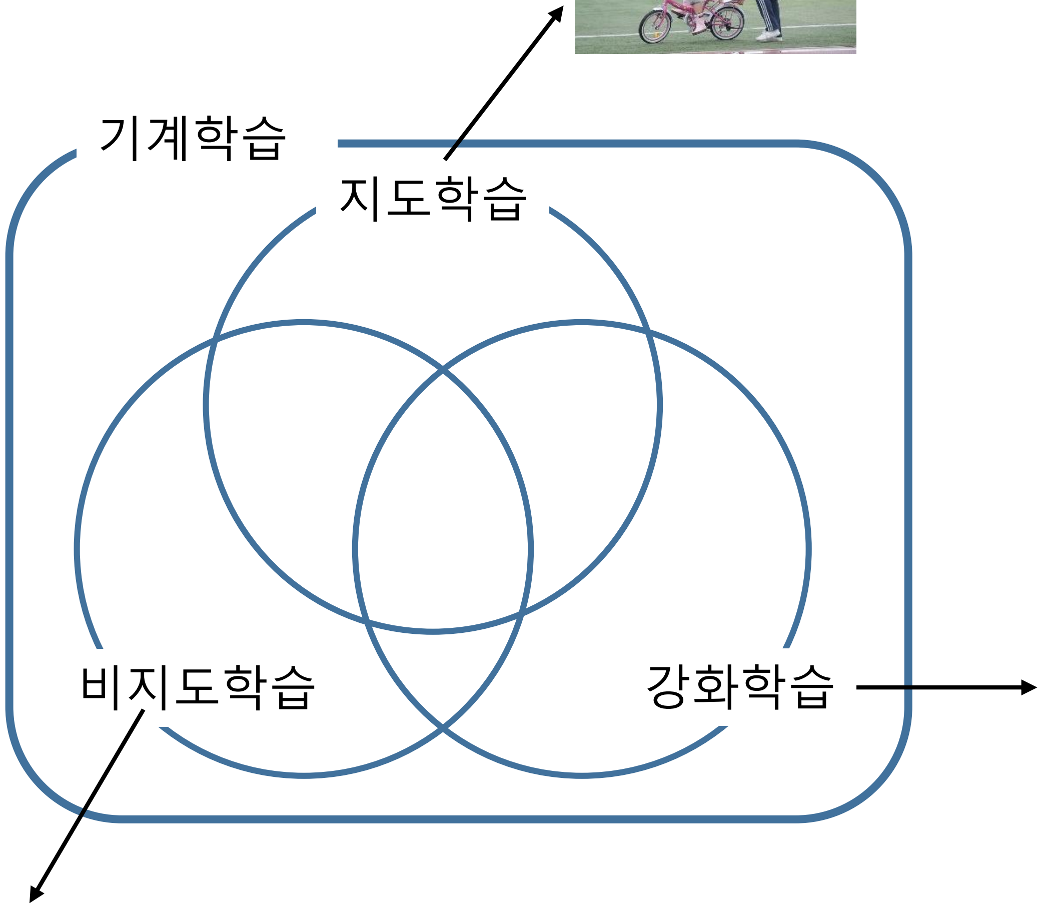
강화학습이 아니었다면
인간을 이길 수 있었을까요?



강화 학습의 매력

Self-Learning의 힘

기계 학습의 분류



강화 학습이란

쉬운 버전

"시행 착오를 통해 보상이 좋았던 행동은 더 하고,
보상이 적었던 행동은 덜 하며 발전하는 과정"

정확한 버전

"순차적 의사 결정 문제에서 누적 보상을 최적화 하기 위해
시행 착오를 통해 행동을 교정하며 학습하는 과정"

순차적 의사결정 문제

샤워를 하는 과정

① 옷을 벗는다.

② 샤워를 한다.

③ 물기를 닦는다.

④ 옷을 입는다.

- 옷을 입고, 샤워를 하고, 물기를 닦고, 옷을 벗는다 => 벌거 벗은 채로 끝남.
- 물기를 닦고, 샤워를 하고, 옷을 벗고, 옷을 입는다 => 물기를 닦을 이유가…?
- 옷을 벗고, 샤워를 하고, 옷을 입고, 물기를 닦는다 => 젖은 채로 옷을 입으면 안 됨

순차적 의사결정 문제의 예시

보상의 특징

어떻게 X
얼마나 O

Scalar

스칼라

희소하고 지연된 보상

Reward Hypothesis

강화학습은 Reward Hypothesis 에 기반

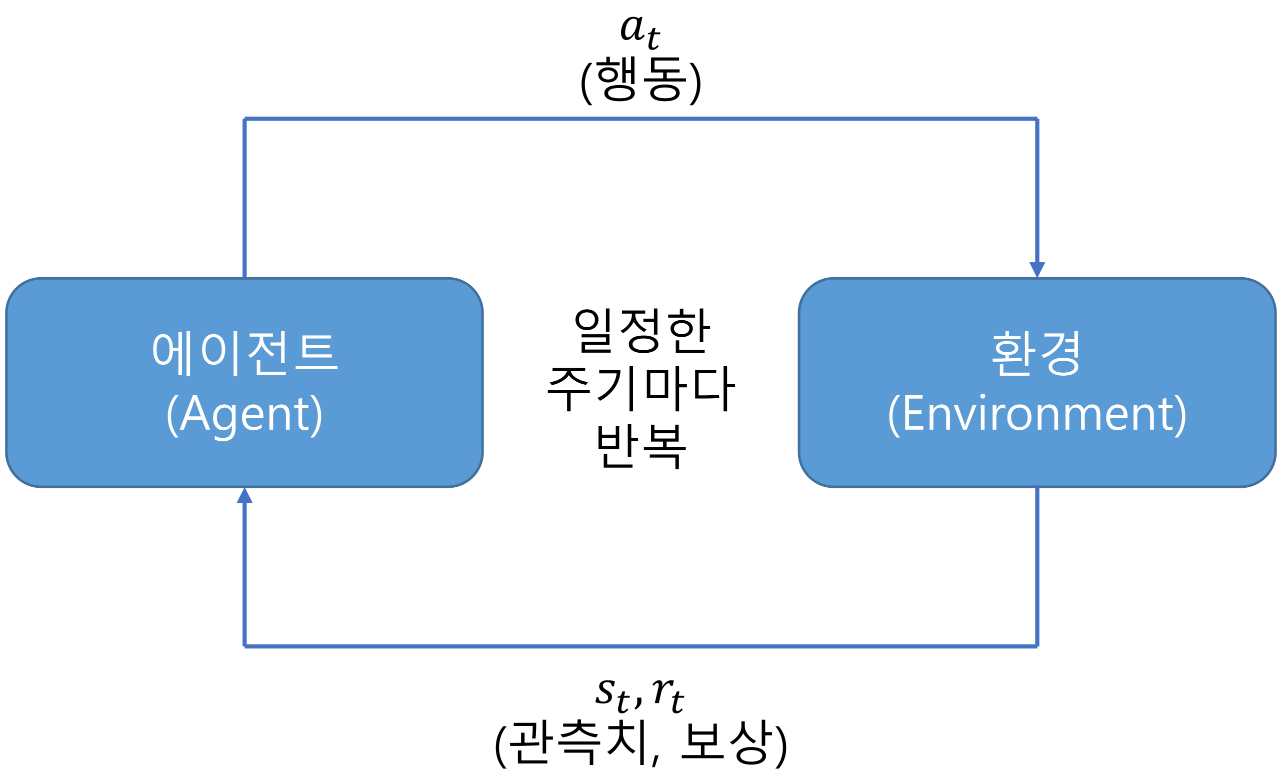
Definition (Reward Hypothesis)

All goals can be described by the maximisation of expected cumulative reward

Reward 설계의 예시

- 바둑을 잘 둔다?
- 운전을 잘 한다?
- 로봇을 걷게 한다?
- 스타크래프트를 잘 한다?
- …

에이전트와 환경



- 에이전트
  - 환경으로부터 현재 시점 t에서의 환경에 대한 정보 s_t와 보상 r_t 를 받음
  - s_t를 바탕으로 어떤 행동을 해야 할지 결정.
  - 결정된 행동 a_t를 환경으로 보냄.

- 환경
  - 에이전트로부터 받은 행동 a_t를 통해서 상태 변화를 일으킴.
  - 그 결과 상태는 s_t -> s_{t+1}로 바뀜.
  - 에이전트에게 줄 보상 r_{t+1}도 함께 계산
  - s_{t+1}과 r_{t+1}을 에이전트에게 전달.

RL Agent의 카테고리

Value Based
(가치 기반)

Policy Based
(정책 기반)

Actor Critic

오늘 배울것!

Exploration vs. Exploitation

- Exploration : 정보를 더 모으고자 모험적 행동을 해보는 것
- Exploitation : 아는 것을 바탕으로 최선을 다 하는 것

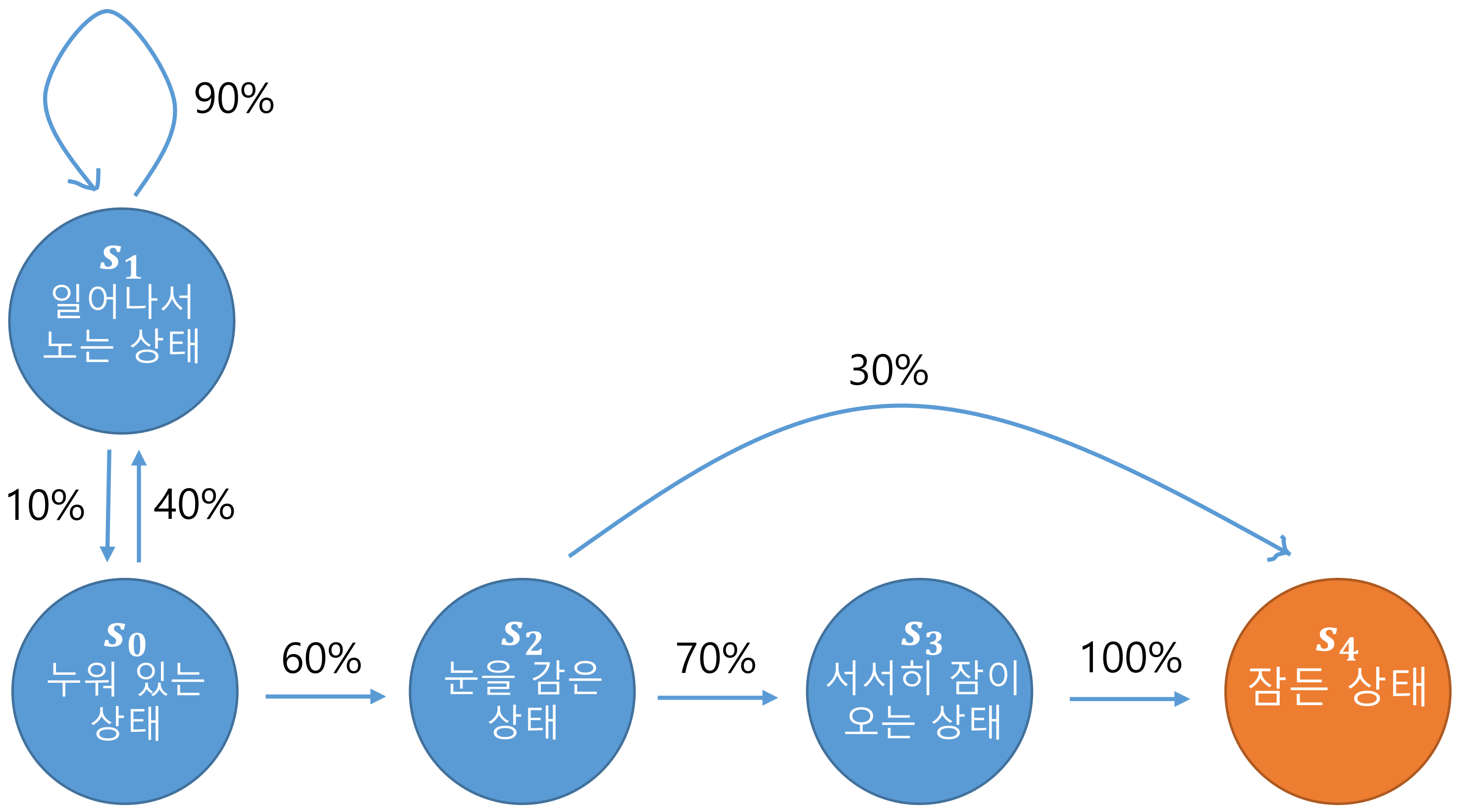
일상 속 Exploration vs. Exploitation

- 저녁 약속을 위한 식당 고르기
- 연애(?)
- 온라인 광고
- 게임 플레이

2. Markov Decision Process

(1) Markov Process
(2) Markov Reward Process
(3) Markov Decision Process

(1) Markov Process



아이 재우기 MP

Markov Process 정의

$$\mathbf{MP} \equiv (\mathbf{S}, \mathbf{P})$$

- 상태의 집합 S
  - 가능한 상태 들을 모두 모아놓은 집합
  - 아이 재우기 예시의 경우에는 이 집합의 원소가 5개

    $S = \{s_0, s_1, s_2, s_3, s_4\}$

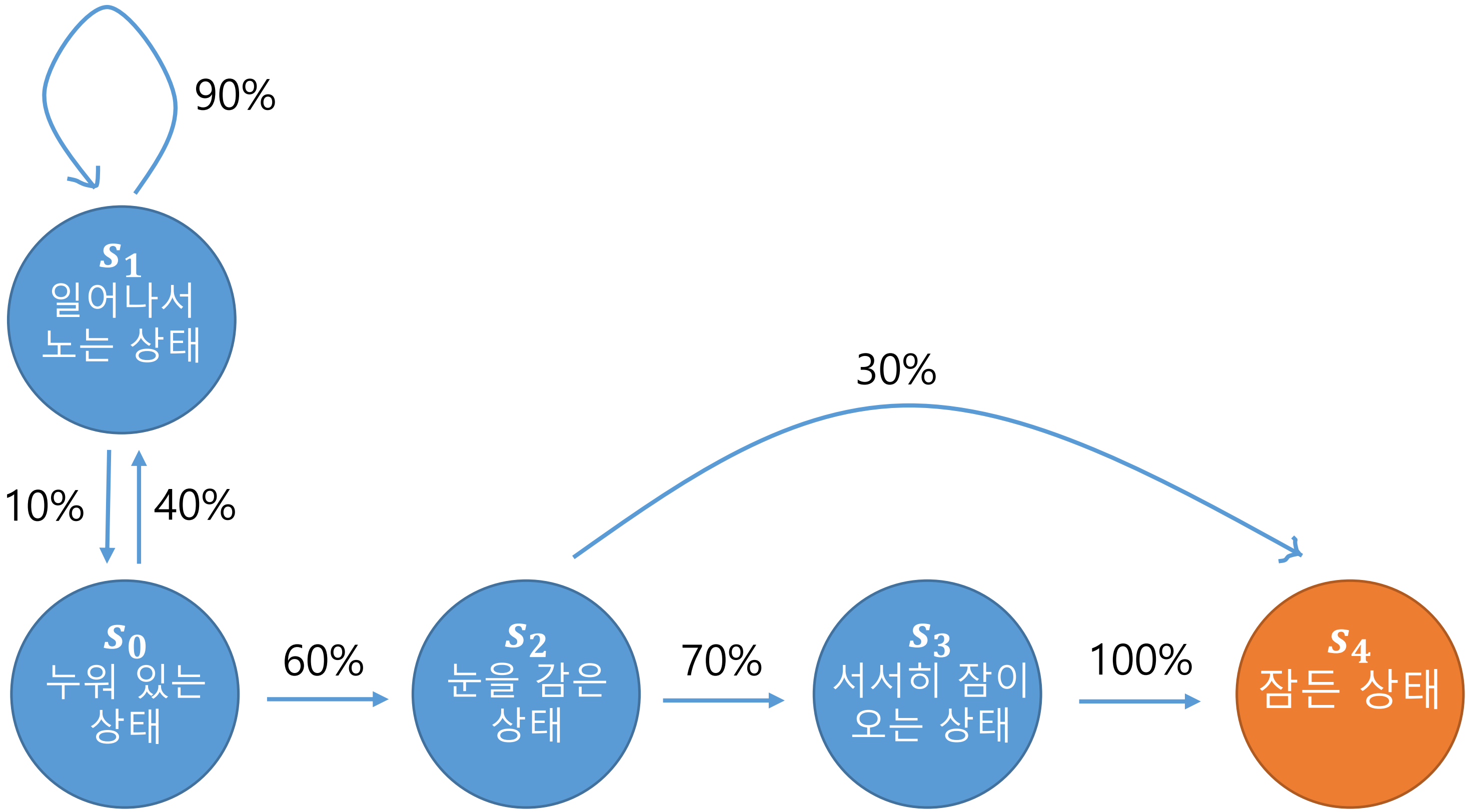
- 전이 확률 행렬 P

$$P_{ss'}$$

1. 상태 s에서

2. 다음 상태가 s′이 될 확률

전이 확률 행렬

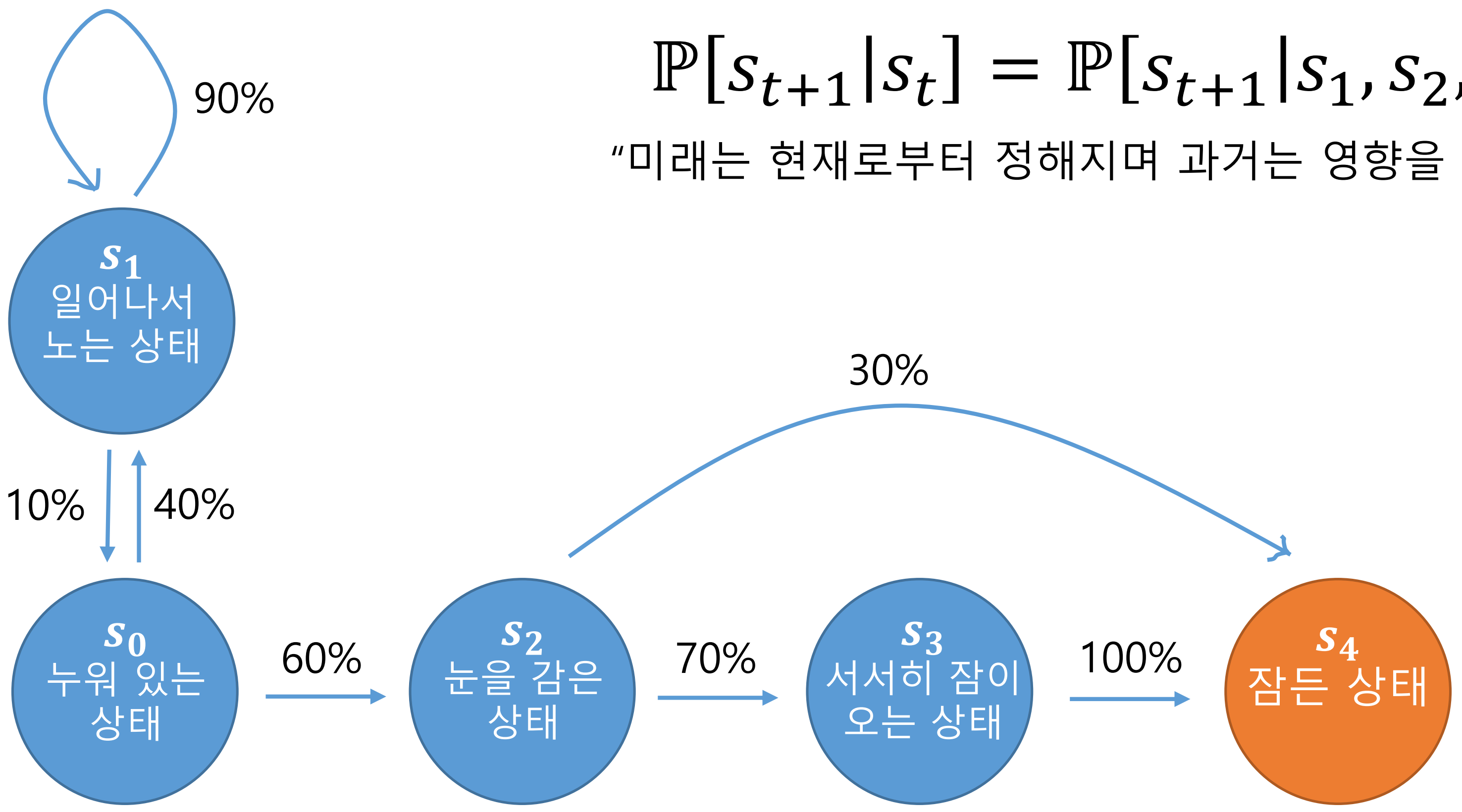


$$\begin{array}{cc} & \begin{array}{ccccc} s_0 & s_1 & s_2 & s_3 & s_4 \end{array} \\ \begin{array}{c} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{array} & \begin{pmatrix} & 0.4 & 0.6 & & \\ 0.1 & 0.9 & & & \\ & & & 0.7 & 0.3 \\ & & & & 1.0 \\ & & & & 1.0 \end{pmatrix} \end{array}$$

마르코프 성질

$$\mathbb{P}[s_{t+1}|s_t] = \mathbb{P}[s_{t+1}|s_1, s_2, \dots, s_t]$$

"미래는 현재로부터 정해지며 과거는 영향을 주지 못한다"

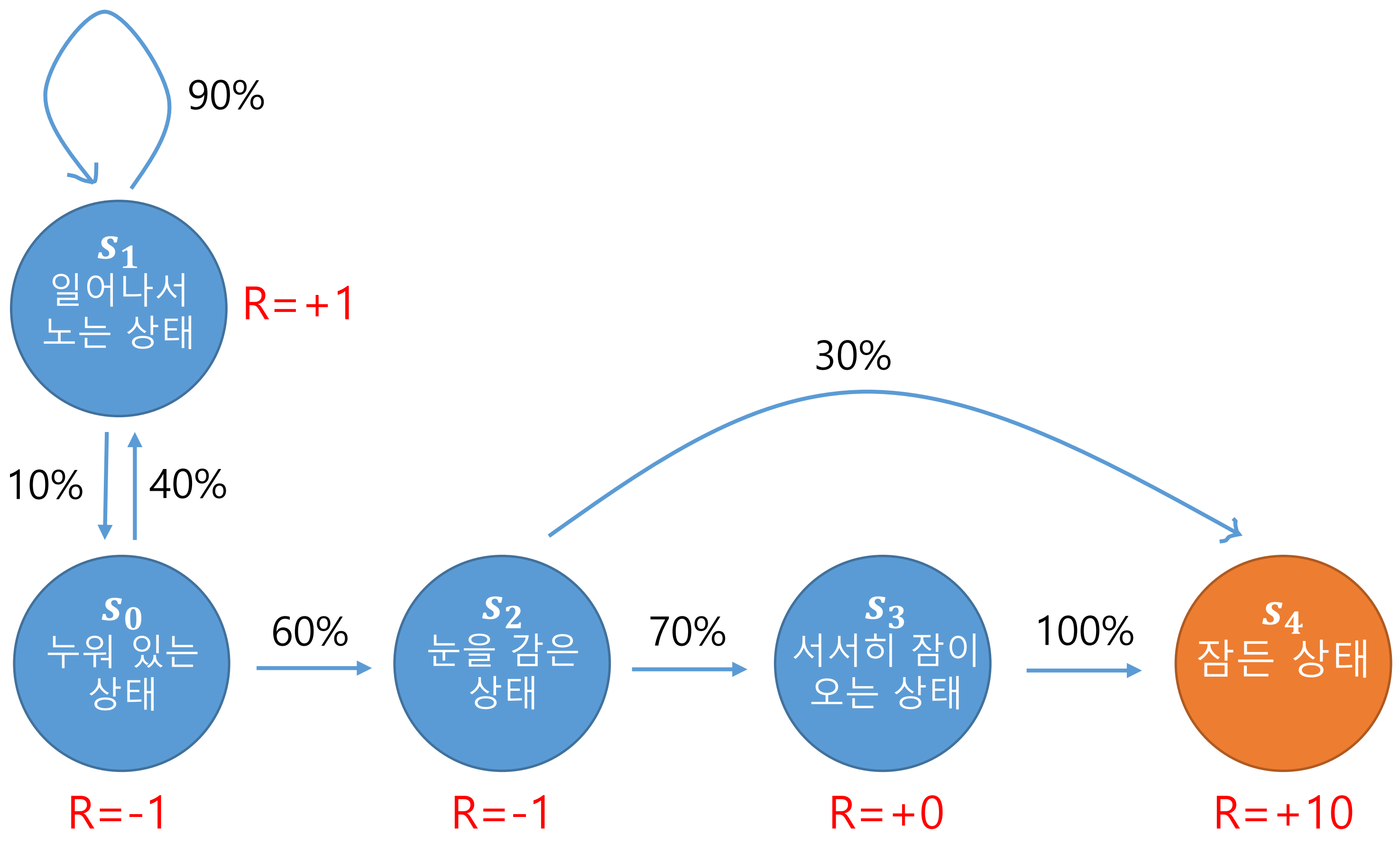


Markov State vs. Non-Markov State

체스판의 상태

운전자의 상태
(시각 정보만 활용)

(2) Markov Reward Process



MP에 리워드가 추가됨.

MRP - 정의

$$\mathbf{MRP} \;\; \equiv (\mathbf{S}, \mathbf{P}, \mathbf{R}, \boldsymbol{\gamma})$$

- 상태의 집합 S
- 전이 확률 행렬 P
- 보상 함수 R
  - 어떤 상태 s에 도착했을 때 받게 되는 보상을 의미
  - $\mathbf{R} = \mathbb{E}[\boldsymbol{R_t} | \boldsymbol{S_t} = \boldsymbol{s}]$
- 감쇠 인자 γ
  - 0에서 1사이의 숫자
  - 보상의 값에 곱하여 미래의 보상을 작게 만드는 역할

리턴

$$\boldsymbol{R_1 + R_2 + R_3 + R_4 + \cdots}$$

$$\boldsymbol{R_t + R_{t+1} + R_{t+2} + R_{t+3} \ldots}$$

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

γ는 왜 필요한가?

- 수학적 편리성
- 사람의 선호 반영
- 미래에 대한 불확실성 반영
- 실제 시스템이 그러한 경우가 있음
  - 이자

상태 평가하기

- 보상과 관련이 있을 것.
- 미래 > 과거
- 보상이 매번 같은가...?

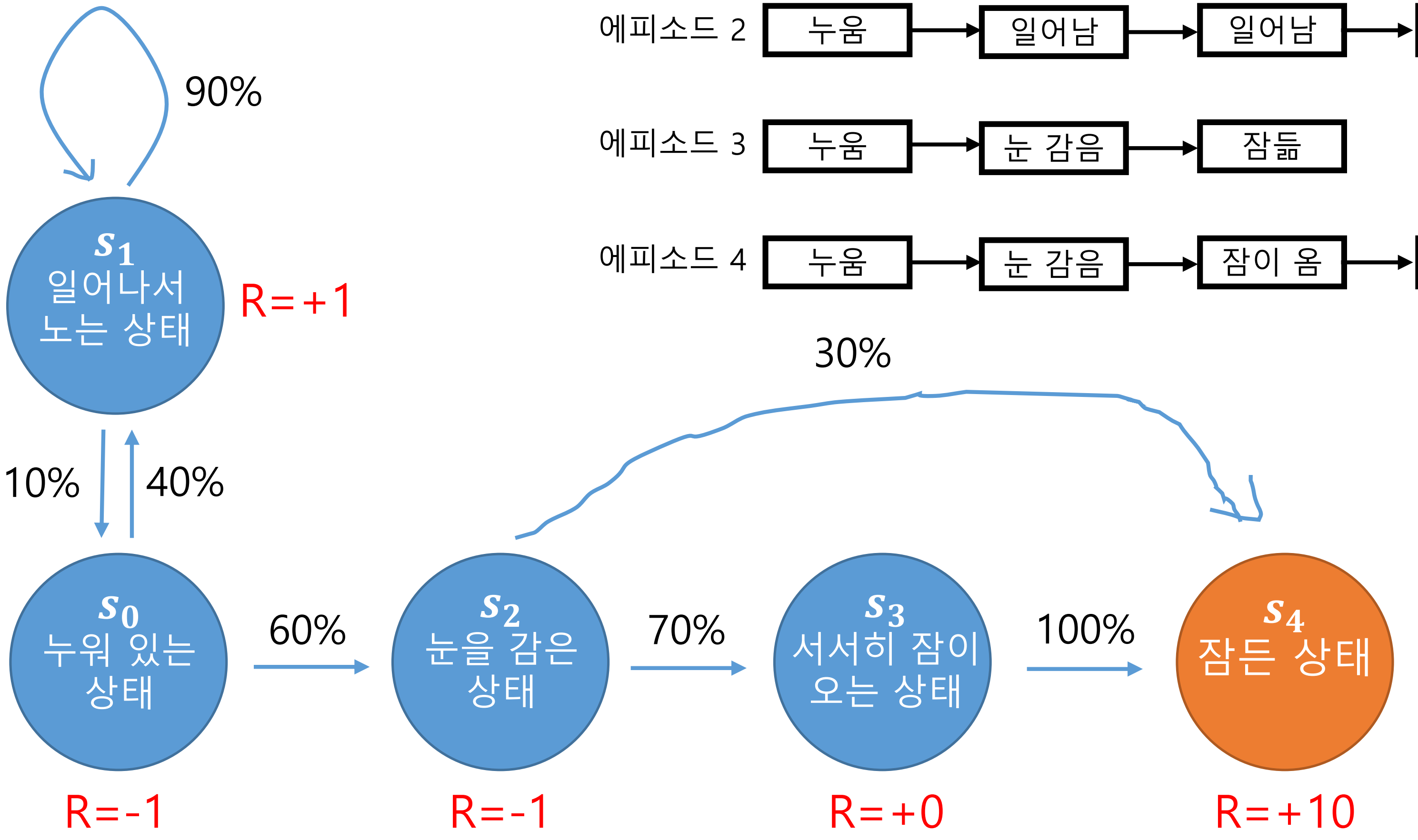
에피소드와 샘플링

에피소드 1 누움 → 일어남 → 누움 → 눈 감음 → 잠이 옴 → 잠듦

에피소드 2 누움 → 일어남 → 일어남 → 누움 → 눈 감음 → 잠이 옴 → 잠듦

에피소드 3 누움 → 눈 감음 → 잠듦

에피소드 4 누움 → 눈 감음 → 잠이 옴 → 잠듦



리턴(return)

$\boldsymbol{\gamma = 0.9}$

에피소드 1 누움 → 일어남 → 누움 → 눈 감음 → 잠이 옴 → 잠듦

$-1 \quad +1 * 0.9 \quad -1 * 0.9^2 \quad -1 * 0.9^3 \quad +0 * 0.9^4 \quad +10 * 0.9^5 \quad = 4.3$

에피소드 2 누움 → 일어남 → 일어남 → 누움 → 눈 감음 → 잠이 옴 → 잠듦

$-1 \quad +1 * 0.9 \quad +1 * 0.9^2 \quad -1 * 0.9^3 \quad -1 * 0.9^4 \quad +0 * 0.9^5 \quad +10 * 0.9^6 \quad = 4.6$

에피소드 3 누움 → 눈 감음 → 잠듦

$-1 \quad -1 * 0.9 \quad +10 * 0.9^2 \quad = 6.2$

에피소드 4 누움 → 눈 감음 → 잠이 옴 → 잠듦

$-1 \quad -1 * 0.9 \quad +0 * 0.9^2 \quad +10 * 0.9^3 \quad = 5.4$

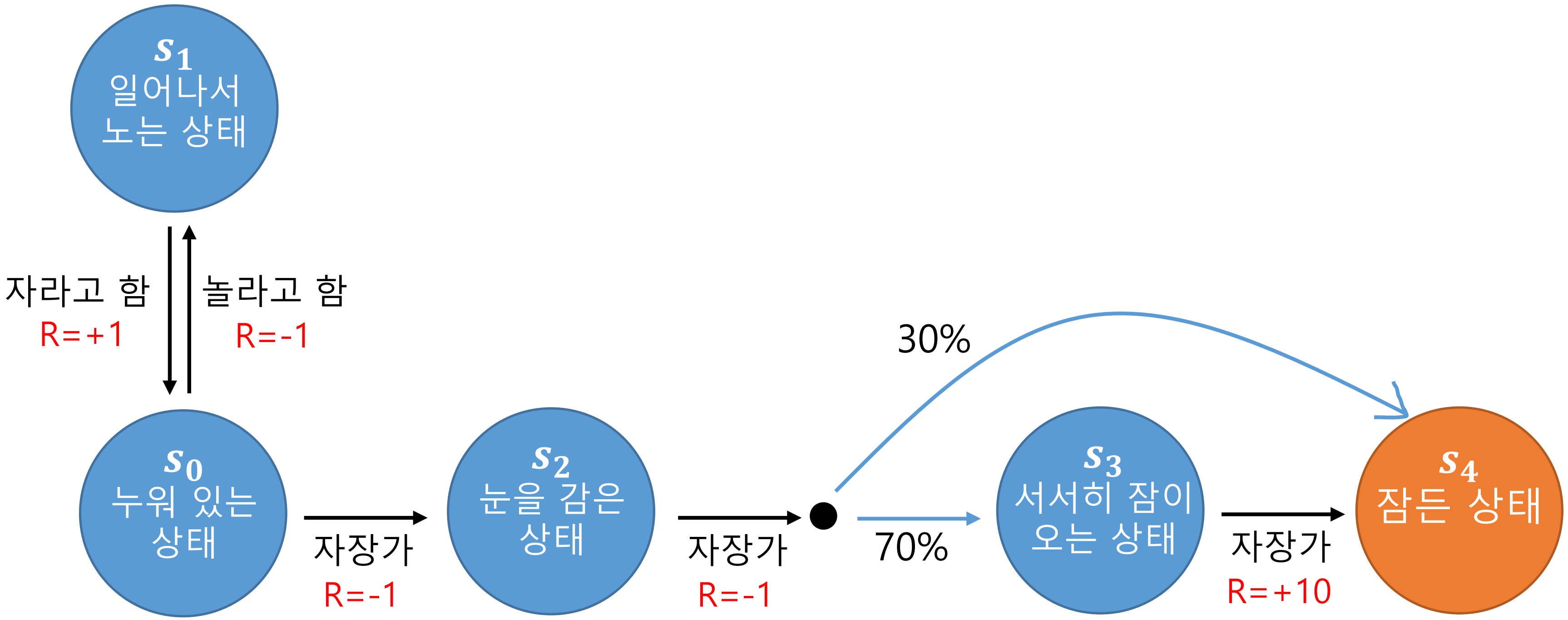
MRP의 가치 함수

가치 함수!

$$\mathbf{v(s)} = \mathbb{E}[\boldsymbol{G_t} | \boldsymbol{S_t} = \boldsymbol{s}]$$

상태 s로부터 시작하여 에피소드가 끝날 때 까지 얻는 리턴(감쇠된 누적 보상)의 기댓값

(3) Markov Decision Process



어머니 출동!
행동하는 주체가 추가됨.

MDP - 정의

$$\mathbf{MDP} \equiv (\mathbf{S}, \mathbf{A}, \mathbf{P}, \mathbf{R}, \boldsymbol{\gamma})$$

- 상태의 집합 S
- 전이 확률 행렬 P

2. 액션 a를 선택 했을때

$$P^{a}_{ss'}$$

1. 상태 s에서

3. 다음 상태가 s′이 될 확률

$$P^{자장가}_{s_2,s_3} = 0.7,\ P^{자장가}_{s_2,s_4} = 0.3$$

- 액션의 집합 S
- 보상 함수 R

$$R^{a}_{s} = \mathbb{E}\,[R_{t+1} | S_t = s, A_t = a]$$

- 감쇠 인자 γ

Policy, Value

정의 1.

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

상태 s에서 액션 a를 선택할 확률

정의 2.

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$$

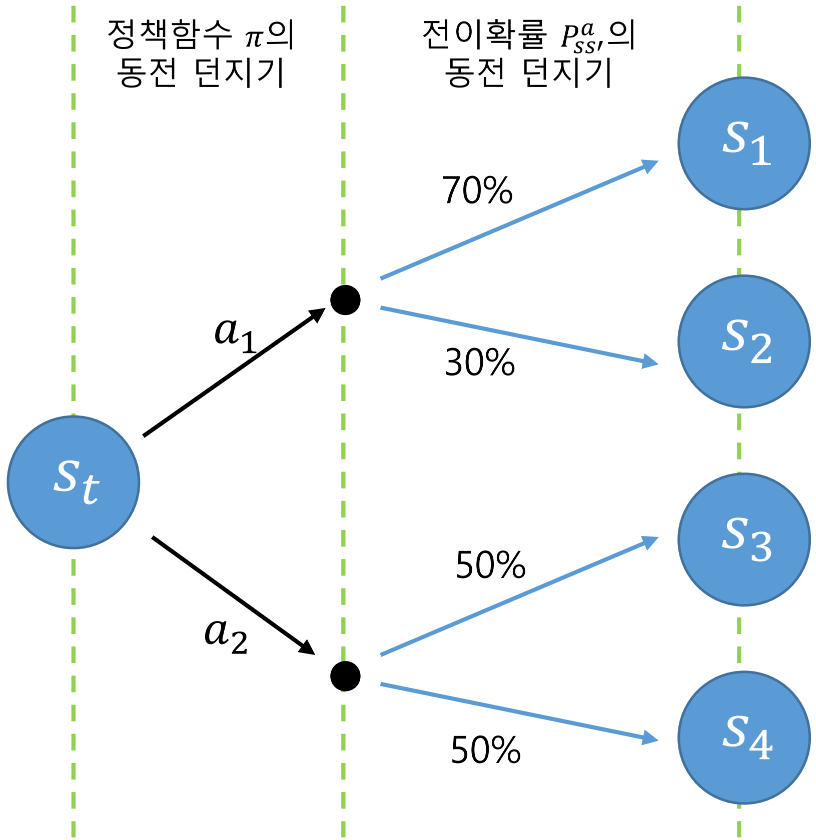
상태 s부터 π를 따라서 움직일 때 리턴의 기댓값

정의 3.

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$$

s에서 a를 선택하고, 그 이후에는 π를 따라서 움직일 때 얻는
리턴의 기댓값

상태 전이를 위한 두 번의 동전 던지기



벨만 기대 방정식

- 가치 함수는 두 파트로 나눠 생각할 수 있다.
  - 즉각적인 보상 R_{t+1}
  - 다음 상태의 가치 $\gamma v_{\pi}(s_{t+1})$

기억하세요

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

기억하세요

- 액션-가치 함수도 마찬가지로 생각 가능.

$$
q_{\pi}(s, a) = \mathbb{E}_{\pi}\left[R_{t+1} + \gamma q_{\pi}(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\right]
$$

OX 퀴즈!

① $v_\pi(s_t) \;=\; r_{t+1} + \gamma v_\pi(s_{t+1})$ 가 성립한다. X

② $v_\pi(s_t) \;=\; \mathbb{E}_\pi[r_{t+1} + \gamma r_{t+1} + \gamma^2 v_\pi(s_{t+2})]$ 가 성립한다. O

벨만 최적 방정식

- 최적의 가치 함수

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

- q_*를 알게 되는 순간 우리는 optimal하게 행동할 수 있다

$$\pi_* = \operatorname*{argmax}_{a} q_*(s,a)$$

벨만 최적 방정식

- 최적의 가치는 모든 액션에 대해 max를 취하는 것이다.

$$q_*(s,a) = r_{t+1} + \gamma \max_{a_{t+1}} r_{t+2} + \gamma^2 \max_{a_{t+2}} r_{t+3} + \cdots$$

$$= r_{t+1} + \gamma \max_{a_{t+1}} q_*(s_{t+1}, a_{t+1})$$

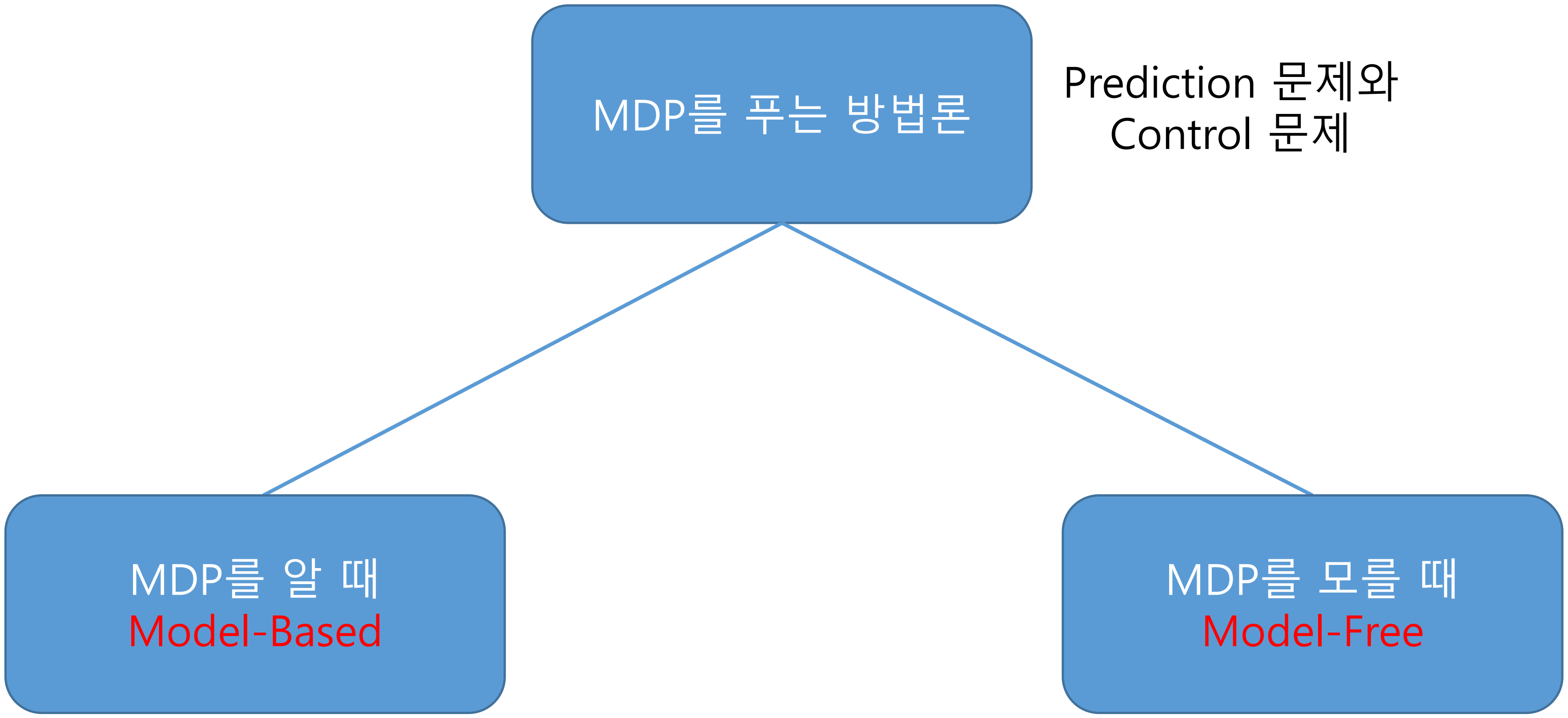
- 벨만 최적 방정식

$$q_*(s,a) = \mathbb{E}_{s'}[\mathrm{r} + \gamma \max_{a'} q_*(s', a')]$$

3. 가치 함수 학습하기

(1) Monte Carlo Learning
(2) TD Learning

Model-Free Prediction



Prediction 문제와
Control 문제

오늘 배울 것!

가치 평가하기 문제

술에 취해 움직이는 사람

출발	s_1	s_2	s_3
s_4	s_5	s_6	s_7
s_8	s_9	s_{10}	s_{11}
s_{12}	s_{13}	s_{14}	종료

보상 : 매 스텝 마다 -1

정책 : 4방향 uniform 랜덤

각 상태에서 종료까지 평균 몇 걸음이 필요할까요?

몬테 카를로 방법론

- MC는 경험으로부터 직접 배우는 방법론
- MC는 model-free 방법론
  - MDP의 상태 전이나 보상 함수에 관한 지식이 전혀 필요 없음
- MC는 **완전한** 에피소드로부터 배움
  - 에피소드가 끝나야 배울 수 있다는 뜻
- MC는 세상에서 가장 간단한 아이디어를 쓴다
  - 가치 = 평균 리턴
- 참고 : MC는 에피소드 단위로 끝나는 MDP에서만 사용할 수 있다.
  - 안 끝나는 MDP에서는 사용 불가.

몬테 카를로 방법론

- 목표 : 정책 π를 이용해 얻은 에피소드들로 부터 가치 함수 v_π 학습하기

$$S_{1,} A_{1,} R_{2,} \ldots, S_{k,} \sim \pi$$

- 리턴이 누적된 보상의 합임을 기억

$$\boldsymbol{G_t} = \boldsymbol{R_t} + \boldsymbol{\gamma} \boldsymbol{R_{t+1}} + \boldsymbol{\gamma}^2 \boldsymbol{R_{t+2}} + \cdots + \boldsymbol{\gamma}^{\mathrm{T}-1} \boldsymbol{R_T}$$

- 가치 함수는 리턴의 기댓값임을 기억

$$\boldsymbol{v_\pi(s)} = \mathbb{E}_\pi \left[\boldsymbol{G_t} | \boldsymbol{S_t} = \boldsymbol{s}\right]$$

- Monte-Carlo policy evaluation은 기댓값 대신에 실제 리턴의 평균을 사용

Every-Visit 몬테 카를로 가치 평가법

- 상태 s의 가치를 평가하기 위해서
- 에피소드 안에서 상태 s를 방문할 때 마다
- 카운터를 증가시키고, $N(s) \leftarrow N(s) + 1$
- 총 리턴 값도 증가시키고, $S(s) \leftarrow S(s) + G_t$
- 가치는 그 평균으로 계산함, $V(s) = S(s)/N(s)$
- 대수의 법칙에 의해 $N(s) \rightarrow \infty$ 이면 $V(s) \rightarrow v_{\pi(s)}$

조금씩 업데이트 하는 버전

$$V(S_t) \leftarrow (1-\alpha) * \underbrace{V(S_t)}_{\text{원래 값 90\%랑}} + \alpha * \underbrace{G_t}_{\text{새로운 값 10\%를 섞음.}}$$

α가 0.1이라면 : 원래 값 90%랑 새로운 값 10%를 섞음.

코드 – 환경

```python
class GridWorld():
    def __init__(self):
        self.x=0
        self.y=0
        self.history = []

    def move_right(self):
        self.y += 1
        if self.y > 3:
            self.y = 3

    def move_left(self):
        self.y -= 1
        if self.y < 0:
            self.y = 0
```

```python
def move_up(self):
    self.x -= 1
    if self.x < 0:
        self.x = 0

def move_down(self):
    self.x += 1
    if self.x > 3:
        self.x = 3
```

https://colab.research.google.com/drive/1XR9NER3FWiruHjgH_zZI37SwRmFmyX3r

코드 – 환경

```python
def move_random(self):
    coin = random.randint(0,3)
    if coin==0:
        self.move_right()
    elif coin==1:
        self.move_left()
    elif coin==2:
        self.move_up()
    else:
        self.move_down()

    self.history.append((self.x, self.y))

def move_random_until_end(self):
    while True:
        self.move_random()

        if self.x == 3 and self.y == 3:
            history = self.history
            self.initialize()
            return history

def initialize(self):
    self.x = 0
    self.y = 0
    self.history = []
```

코드 – MC 학습

```python
data = [[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]
env = GridWorld()

for k in range(10000):
  history = env.move_random_until_end()
  cum_reward = 0

  for position in history[::-1]:
    x, y = position
    data[x][y] = 0.999*data[x][y] + 0.001*cum_reward
    cum_reward -= 1

for row in data:
  print(row)
```

α =0.001

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=10

-2.7	-2.4	-1.2	-1.1
-2.4	-1.7	-1.2	-0.7
-2.1	-1.7	-1.5	-0.4
-2.6	-1.6	-1.1	0.0

k=100

-28.7	-24.0	-16.1	-13.8
-24.2	-18.4	-12.6	-10.9
-19.5	-14.3	-8.3	-5.1
-18.3	-12.1	-4.9	0.0

k=1000

-52.1	-51.0	-47.7	-42.9
-50.7	-47.9	-43.0	-38.7
-49.8	-43.9	-34.2	-22.7
-47.6	-38.6	-23.4	0.0

k=10000

-58.3	-57.0	-56.7	-56.2
-56.4	-55.3	-50.2	-45.9
-52.9	-50.6	-42.7	-30.6
-53.6	-46.0	-31.5	0.0

$\times\infty$

k=∞

-59.4	-57.4	-54.3	-51.7
-57.4	-54.6	-49.7	-45.1
-54.3	-49.7	-40.9	-30
-51.7	-45.1	-30	0.0

(2) Temporal-Difference 학습

- TD 방법론은 경험으로부터 직접 학습한다
- TD는 model-free 방법론. MDP에 대한 정보를 필요로 하지 않는다.
- TD는 에피소드가 끝나지 않아도 학습할 수 있다.
- TD는 추측을 추측으로 업데이트 하는 방법론이다.

Key Idea
모레에 비가 오는지 알고싶어?
오늘 추측하는 것 보다는 내일 추측하는게 더 정확하겠지!

MC 와 TD

MC

$$V(S_t) \leftarrow (1-\alpha) * \underbrace{V(S_t)}_{\text{원래 값 90\%랑}} + \alpha * \underbrace{G_t}_{\text{새로운 값 10\%를 섞음.}}$$

α가 0.1이라면 : 원래 값 90%랑 새로운 값 10%를 섞음.

TD

$$V(S_t) \leftarrow (1-\alpha) * \underbrace{V(S_t)}_{\text{원래 값 90\%랑}} + \alpha * \underbrace{(R_{t+1} + \gamma V(S_{t+1}))}_{\text{새로운 값 10\%를 섞음.}}$$

α가 0.1이라면 : 원래 값 90%랑 새로운 값 10%를 섞음.
TD Target이라고 부름.

코드 – TD 학습

```
data = [[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]
env = GridWorld()

for k in range(10):
  history = env.move_random_until_end()

  for i in range(len(history)-1):
    x, y = history[i]
    next_x, next_y = history[i+1]
    data[x][y] = 0.99*data[x][y] + 0.01*(-1+data[next_x][next_y])


for row in data:
  print(row)
```

https://colab.research.google.com/drive/1MSFCuJ-OjslWVHnK1Lzcwmp-4dB0wedH

α =0.01

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=10

-1.0	-0.8	-0.6	-0.4
-0.7	-0.7	-0.6	-0.4
-0.5	-0.4	-0.3	-0.1
-0.4	-0.2	-0.2	0.0

k=100

-5.0	-4.6	-3.7	-3.2
-4.6	-4.0	-3.2	-2.7
-4.0	-3.4	-2.7	-1.7
-3.6	-3.1	-1.9	0.0

k=1000

-32.2	-30.8	-28.6	-27.0
-31.0	-28.9	-25.9	-22.3
-28.3	-25.7	-20.6	-13.9
-26.7	-23.3	-14.6	0.0

k=10000

-59.2	-57.4	-53.6	-50.1
-57.2	-54.3	-48.7	-42.7
-54.7	-49.7	-38.6	-28.0
-52.3	-45.5	-29.3	0.0

$\times \infty$

k=∞

-59.4	-57.4	-54.3	-51.7
-57.4	-54.6	-49.7	-45.1
-54.3	-49.7	-40.9	-30
-51.7	-45.1	-30	0.0

각 방법론의 특징

- TD 는 최종 결과를 알기 전에 학습할 수 있다.
  - TD는 매 스텝마다 온라인으로 학습할 수 있음.
  - 반면 MC는 에피소드가 끝나서 리턴을 알게 될 때 까지 기다려야 함.

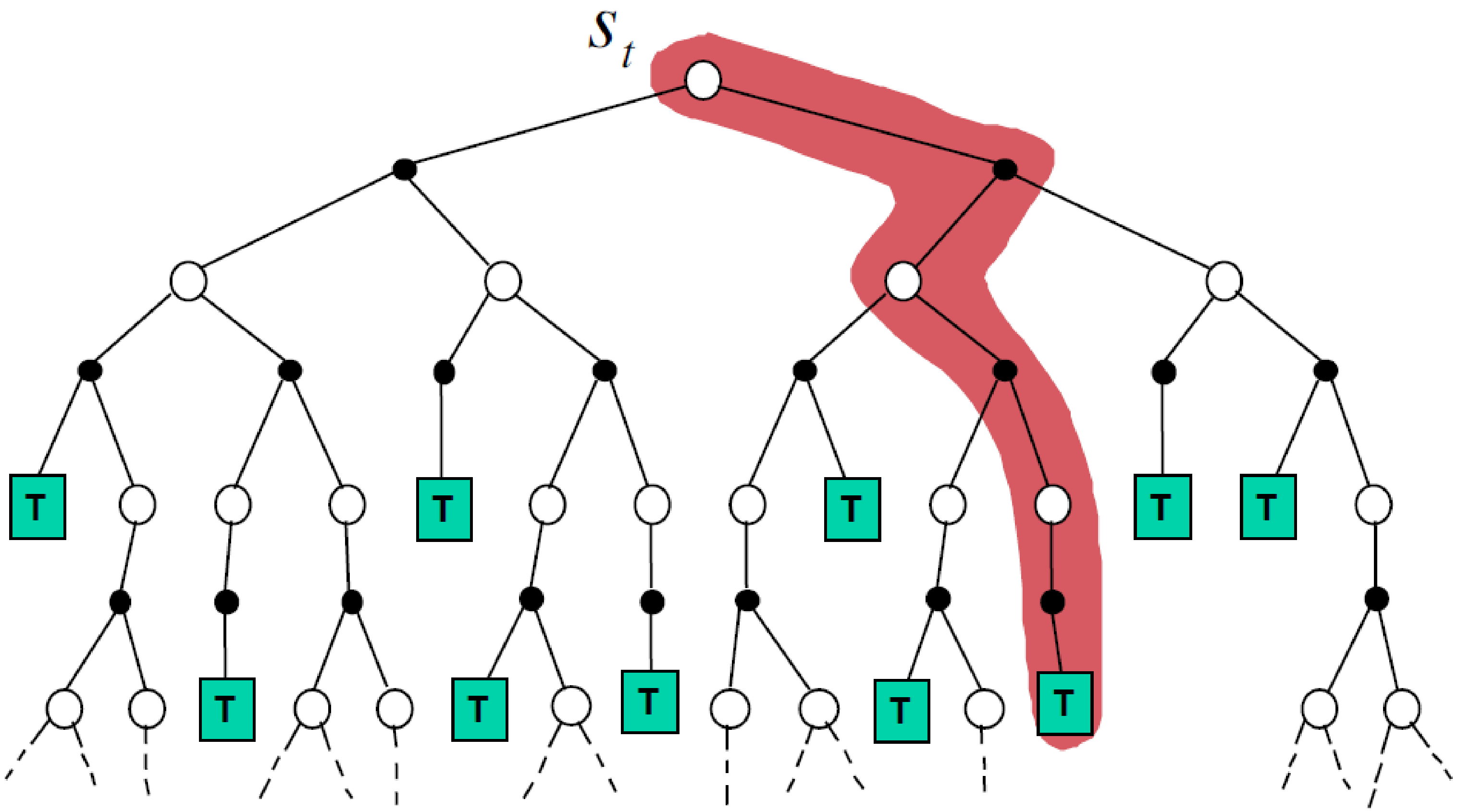
- MC : $v_\pi(s_t) = \mathbb{E}[G_t]$
  TD : $v_\pi(s_t) = \mathbb{E}[r_{t+1} + \gamma v_\pi(s_{t+1})]$

- Bias
  - 리턴 G_t는 가치 함수 $v_\pi(S_t)$의 unbiased estimate
  - $R_{t+1} + \gamma v_\pi(S_{t+1})$ 도 unbiased
  - 하지만! $R_{t+1} + \gamma V(S_{t+1})$ 은 biased

- Variance
  - TD 타겟은 리턴보다 variance가 훨씬 작음.
  - 리턴은 수많은 액션, 트랜지션, 보상과 관련이 되지만 TD 타겟은 한 개의 액션, 트랜지션, 보상과 관련이 있기 때문.

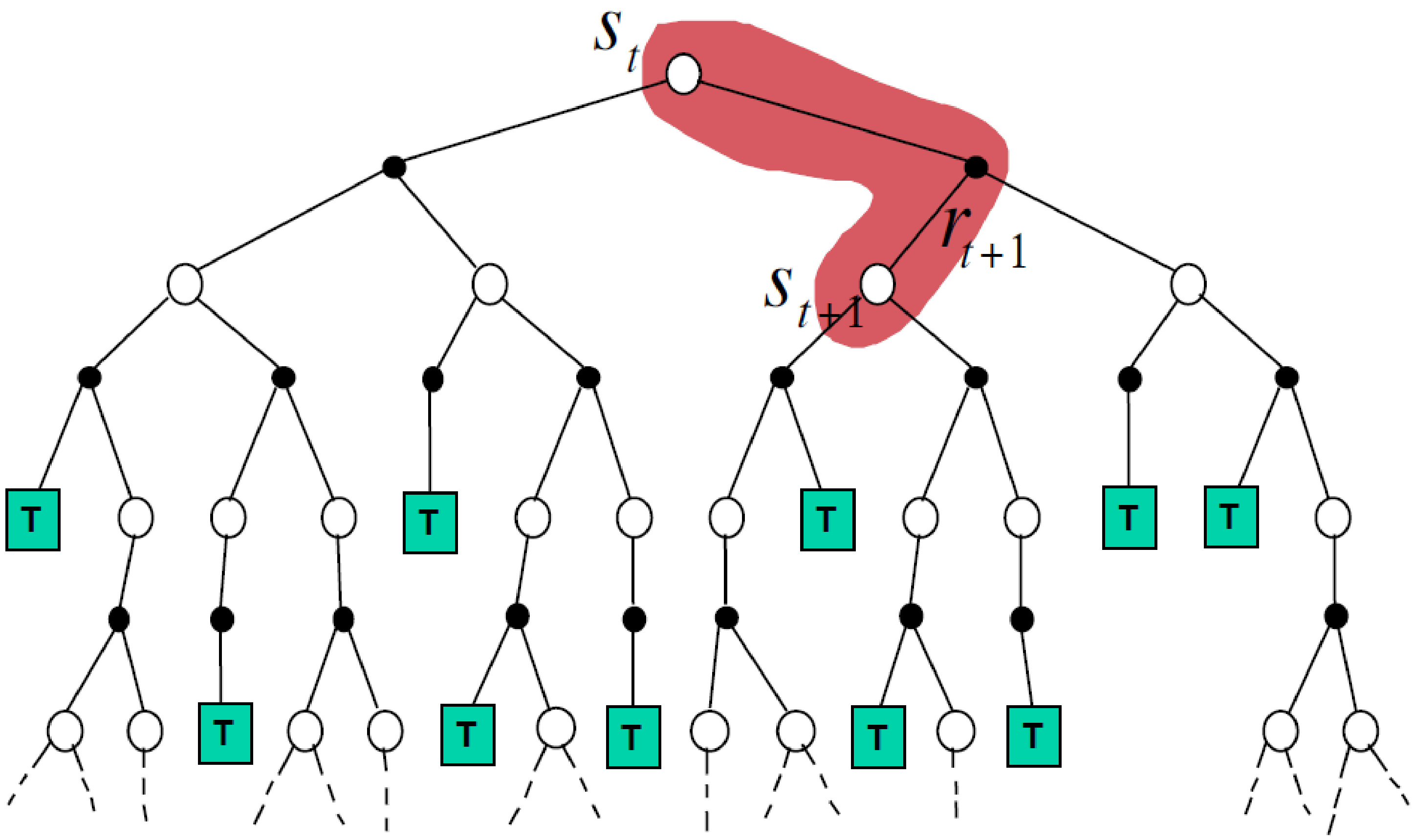
MC의 back up

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t - V(S_t)\right)$$



TD의 back up

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

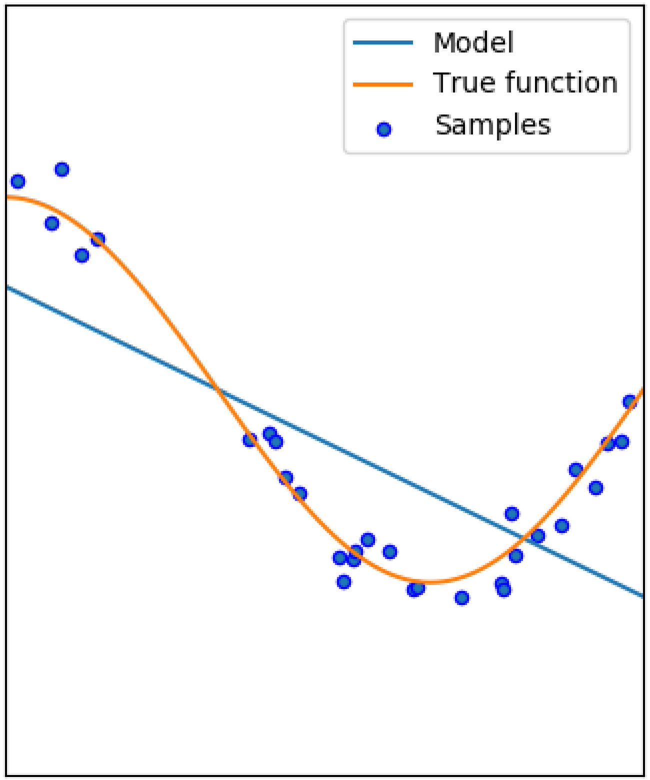


4. DQN 이론

(1) 인공 신경망과 경사 하강법
(2) Deep Q-Learning

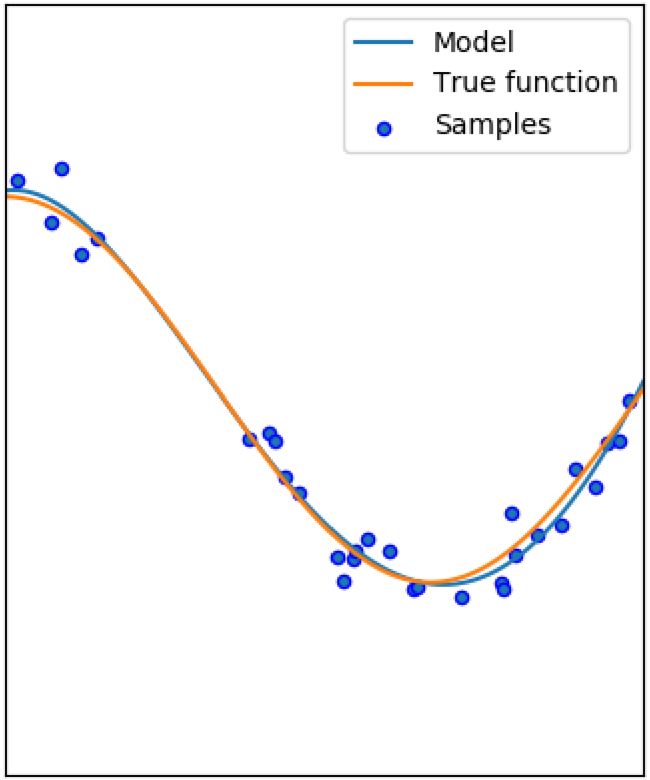
데이터와 모델링

1차 모델



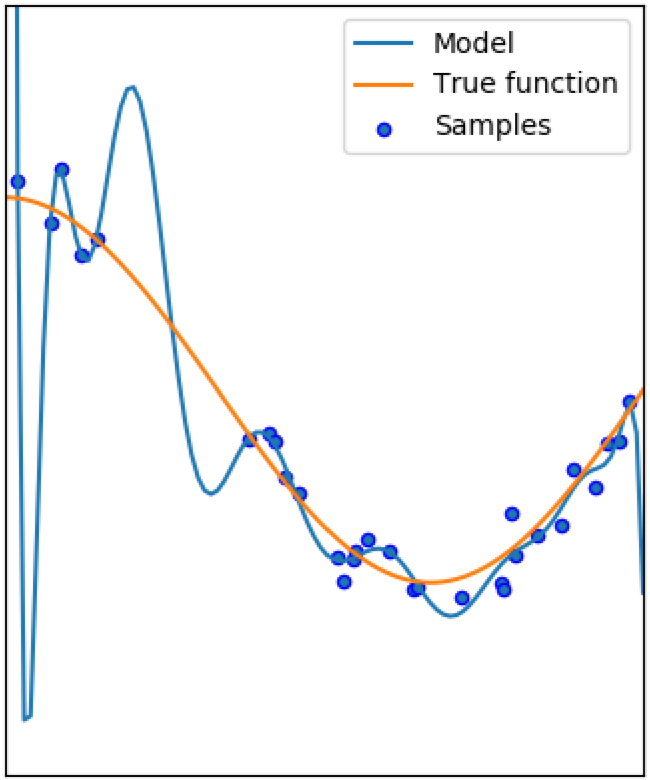
$$y = a_1 x + a_0$$

4차 모델



$$y = a_4 x^4 + a_3 x^3 + \cdots + a_1 x + a_0$$

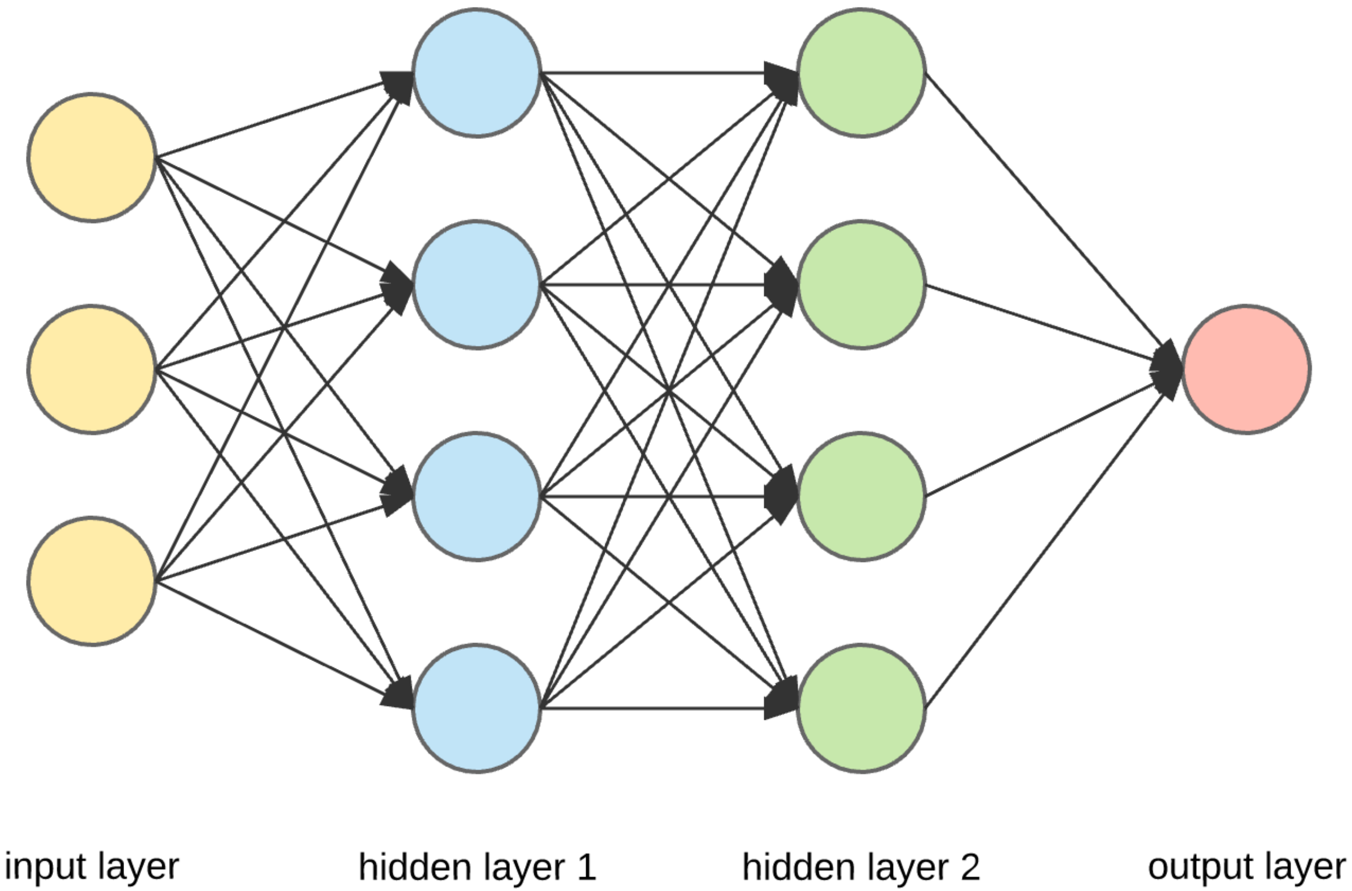
15차 모델



$$y = a_{15} x^{15} + a_{14} x^{14} + \cdots + a_1 x + a_0$$

https://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html

Neural Network



여기 아주 유연한 함수가 하나 있다.

https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6

Neural Network

- 학습 과정 : 뉴럴 넷을 수정해 주는 과정
- 의도한 값 보다 뉴럴넷 아웃풋이 작으면 -> 아웃풋이 커지도록
  의도한 값 보다 뉴럴넷 아웃풋이 크면 -> 아웃풋이 작아지도록
- 어떻게?
  - Gradient Descent(경사 하강법)!
- 결국 뉴럴넷에 정답을 새겨 넣는 것. 마치 테이블과 비슷함.
- 대신 좀 더 generalization이 잘 될 뿐.

그러면 정답은 무엇...?

다음 두 아이디어의 혼합!

- TD 러닝

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

- 벨만 최적 방정식

$$q_*(s,a) = \mathbb{E}_{s'}\,[\mathrm{r} + \gamma \max_{a'} q_*(s',a')]$$

Pseudo Code

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        Choose A from S using policy derived from Q (e.g., ε-greedy)
        Take action A, observe R, S'
        $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
        $S \leftarrow S'$;
    until S is terminal

LETTER

doi:10.1038/nature14236

Human-level control through deep reinforcement learning

Volodymyr Mnih[1]*, Koray Kavukcuoglu[1]*, David Silver[1]*, Andrei A. Rusu[1], Joel Veness[1], Marc G. Bellemare[1], Alex Graves[1], Martin Riedmiller[1], Andreas K. Fidjeland[1], Georg Ostrovski[1], Stig Petersen[1], Charles Beattie[1], Amir Sadik[1], Ioannis Antonoglou[1], Helen King[1], Dharshan Kumaran[1], Daan Wierstra[1], Shane Legg[1] & Demis Hassabis[1]

The theory of reinforcement learning provides a normative account[1], deeply rooted in psychological[2] and neuroscientific[3] perspectives on animal behaviour, of how agents may optimize their control of an environment. To use reinforcement learning successfully in situations approaching real-world complexity, however, agents are confronted with a difficult task: they must derive efficient representations of the environment from high-dimensional sensory inputs, and use these to generalize past experience to new situations. Remarkably, humans and other animals seem to solve this problem through a harmonious combination of reinforcement learning and hierarchical sensory processing systems[4,5], the former evidenced by a wealth of neural data revealing notable parallels between the phasic signals emitted by dopa-

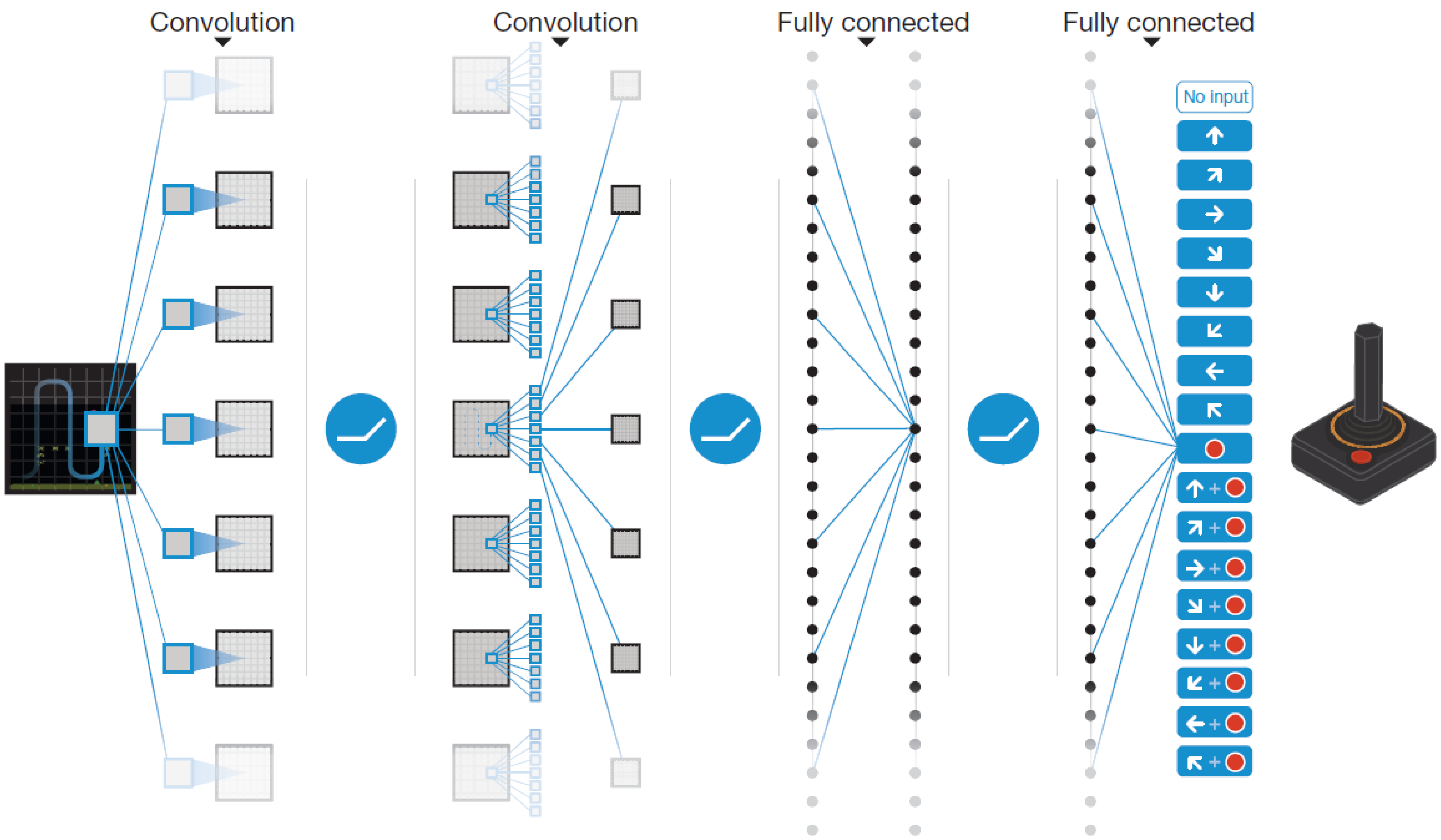
agent is to select actions in a fashion that maximizes cumulative future reward. More formally, we use a deep convolutional neural network to approximate the optimal action-value function

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \middle| s_t = s,\ a_t = a,\ \pi\right],$$

which is the maximum sum of rewards r_t discounted by γ at each time-step t, achievable by a behaviour policy $\pi = P(a|s)$, after making an observation (s) and taking an action (a) (see Methods)[19].

Reinforcement learning is known to be unstable or even to diverge when a nonlinear function approximator such as a neural network is used to represent the action-value (also known as Q) function[20]. This instability has several causes: the correlations present in the sequence

아타리 게임에서의 agent

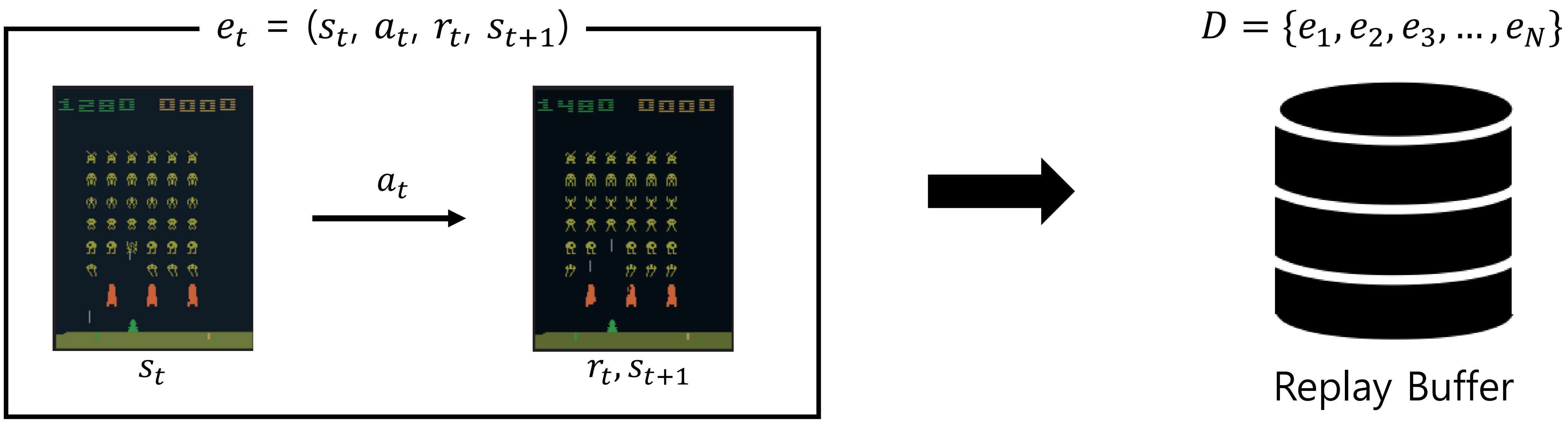


뉴럴 넷의 Loss Function

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim \mathrm{U}(D)} \left[\left(\underline{r + \gamma \max_{a'} Q(s', a'; \theta_i^-)} - Q(s, a; \theta_i) \right)^2 \right]$$

$$q_*(s,a) = \mathbb{E}_{s'} [\mathrm{r} + \gamma \max_{a'} q_*(s',a')]$$

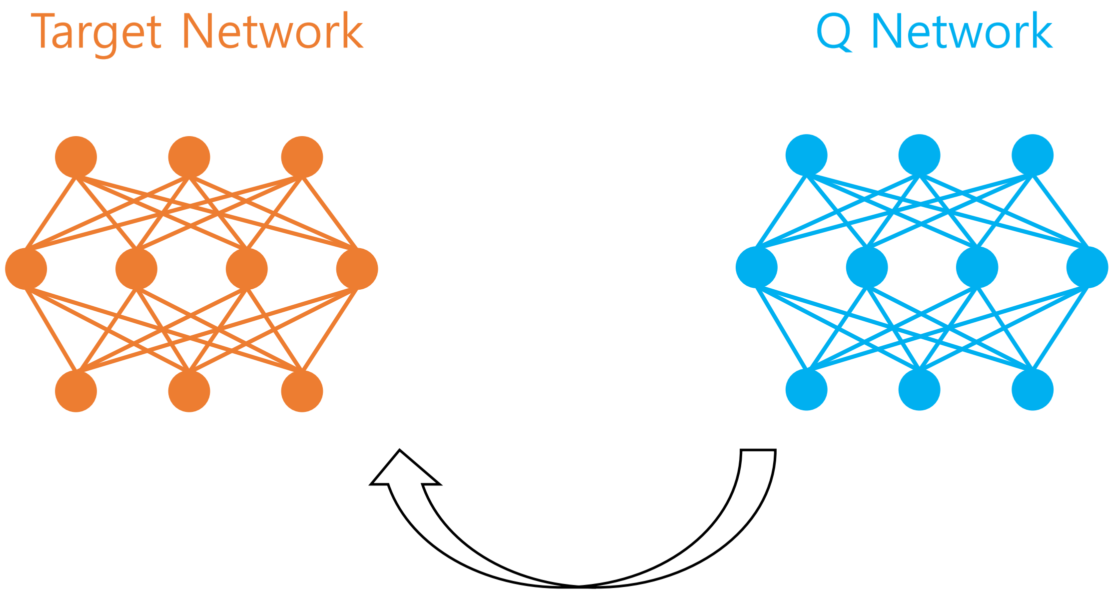
리플레이 버퍼



1. 시뮬레이션때 매 틱마다 생성되는 Transition 튜플 (s_t, a_t, r_t, s_{t+1})을 Replay Buffer 에 저장
2. Replay Buffer는 자동으로 가장 최신의 5만개의 튜플을 갖고 있다.
3. 학습 시에는 이 5만개 중에서 임의로 32 개를 뽑아서, minibatch를 구성하여 학습.

타겟 네트워크

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim \mathrm{U}(D)}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i)\right)^2\right]$$

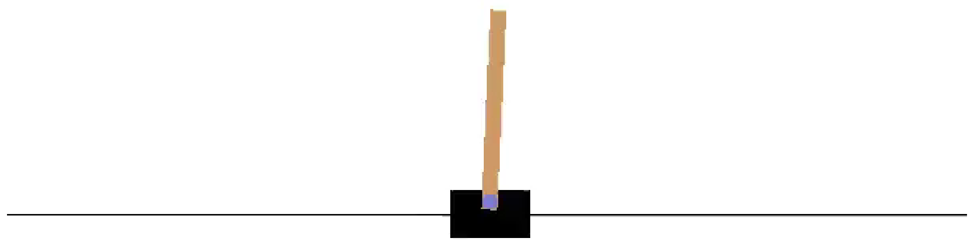


일정 주기마다 update

$$\theta_i^- = \theta_i$$

5. DQN 실습

문제 - CartPole



- 카트를 잘 밀어서 균형을 잡는 문제
- 카트를 왼쪽이나 오른쪽으로 밀 수 있음
- 매 스텝마다 +1의 보상을 받음
- 막대가 수직으로부터 15도 이상 기울어지거나 화면 끝으로 나가면 종료

Import & Hyperparameter setting

```
import gym
import collections
import random

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

#Hyperparameters
learning_rate = 0.0005
gamma         = 0.98
buffer_limit  = 50000
batch_size    = 32
```

- Collections library는 replay buffer에서 쓰일 deque 를 import 하기 위함.

https://colab.research.google.com/drive/1EHwPQOF-GWYPHMD54YAchbveMLLOVpPk

Replay Buffer

```
class ReplayBuffer():
    def __init__(self):
        self.buffer = collections.deque(maxlen=buffer_limit)

    def put(self, transition):
        self.buffer.append(transition)

    def sample(self, n):
        mini_batch = random.sample(self.buffer, n)
        s_lst, a_lst, r_lst, s_prime_lst, done_mask_lst = [], [], [], [], []

        for transition in mini_batch:
            s, a, r, s_prime, done_mask = transition
            s_lst.append(s)
            a_lst.append([a])
            r_lst.append([r])
            s_prime_lst.append(s_prime)
            done_mask_lst.append([done_mask])

        return torch.tensor(s_lst, dtype=torch.float), torch.tensor(a_lst), \
               torch.tensor(r_lst), torch.tensor(s_prime_lst, dtype=torch.float), \
               torch.tensor(done_mask_lst)

    def size(self):
        return len(self.buffer)
```

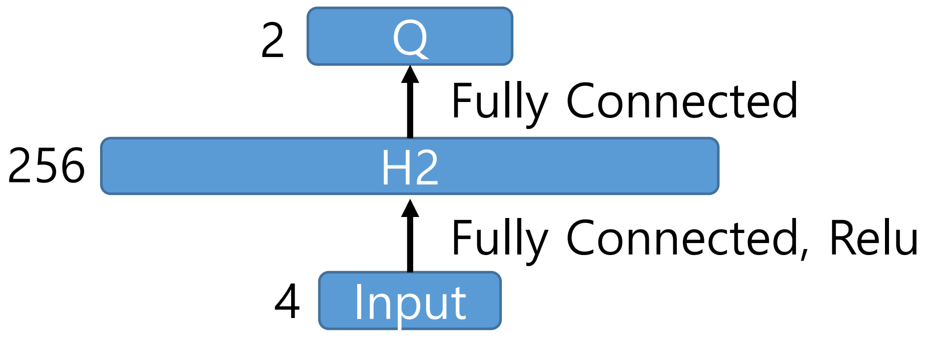
Buffer의 최대 크기

Q Network

```python
class Qnet(nn.Module):
    def __init__(self):
        super(Qnet, self).__init__()
        self.fc1 = nn.Linear(4, 256)
        self.fc2 = nn.Linear(256, 2)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

    def sample_action(self, obs, epsilon):
        out = self.forward(obs)
        coin = random.random()
        if coin < epsilon:
            return random.randint(0,1)
        else :
            return out.argmax().item()
```



Main

```python
for n_epi in range(10000):
    epsilon = max(0.01, 0.08 - 0.01*(n_epi/200)) #Linear annealing from 8% to 1%
    s = env.reset()

    for t in range(600):
        a = q.sample_action(torch.from_numpy(s).float(), epsilon)
        s_prime, r, done, info = env.step(a)
        done_mask = 0.0 if done else 1.0
        memory.put((s,a,r/100.0,s_prime, done_mask))
        s = s_prime

        score += r
        if done:
            break

    if memory.size()>2000:
        train(q, q_target, memory, optimizer)
```

Train

```python
def train(q, q_target, memory, optimizer):
    for i in range(10):
        s,a,r,s_prime,done_mask = memory.sample(batch_size)

        q_out = q(s)
        q_a = q_out.gather(1,a)
        max_q_prime = q(s_prime).max(1)[0].unsqueeze(1)
        target = r + gamma * max_q_prime * done_mask
        loss = F.smooth_l1_loss(q_a, target)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

Shape : [32,2]

취한 action의 q값만 골라냄.
Shape : [32,1]

$$r + \gamma \max_{a'} Q(s', a'; \theta_i^-)$$

학습 결과

```
# of episode :20, Avg timestep : 9.1, buffer size : 202, epsilon : 7.9%
# of episode :40, Avg timestep : 8.6, buffer size : 393, epsilon : 7.8%
# of episode :60, Avg timestep : 8.7, buffer size : 587, epsilon : 7.7%
# of episode :80, Avg timestep : 8.6, buffer size : 779, epsilon : 7.6%
# of episode :100, Avg timestep : 8.7, buffer size : 973, epsilon : 7.5%
# of episode :120, Avg timestep : 8.6, buffer size : 1165, epsilon : 7.4%
# of episode :140, Avg timestep : 8.8, buffer size : 1360, epsilon : 7.3%
# of episode :160, Avg timestep : 9.0, buffer size : 1560, epsilon : 7.2%
# of episode :180, Avg timestep : 8.7, buffer size : 1754, epsilon : 7.1%
# of episode :200, Avg timestep : 8.7, buffer size : 1947, epsilon : 7.0%
# of episode :220, Avg timestep : 10.6, buffer size : 2179, epsilon : 6.9%
# of episode :240, Avg timestep : 14.6, buffer size : 2491, epsilon : 6.8%
# of episode :260, Avg timestep : 10.9, buffer size : 2729, epsilon : 6.7%
# of episode :280, Avg timestep : 9.9, buffer size : 2947, epsilon : 6.6%
# of episode :300, Avg timestep : 17.4, buffer size : 3316, epsilon : 6.5%
# of episode :320, Avg timestep : 109.5, buffer size : 5525, epsilon : 6.4%
# of episode :340, Avg timestep : 121.2, buffer size : 7970, epsilon : 6.3%
# of episode :360, Avg timestep : 213.4, buffer size : 12259, epsilon : 6.2%
# of episode :380, Avg timestep : 179.5, buffer size : 15869, epsilon : 6.1%
# of episode :400, Avg timestep : 114.1, buffer size : 18171, epsilon : 6.0%
# of episode :420, Avg timestep : 101.2, buffer size : 20215, epsilon : 5.9%
# of episode :440, Avg timestep : 139.5, buffer size : 23025, epsilon : 5.8%
# of episode :460, Avg timestep : 163.5, buffer size : 26315, epsilon : 5.7%
# of episode :480, Avg timestep : 199.3, buffer size : 30322, epsilon : 5.6%
# of episode :500, Avg timestep : 268.6, buffer size : 35715, epsilon : 5.5%
# of episode :520, Avg timestep : 260.0, buffer size : 40935, epsilon : 5.4%
# of episode :540, Avg timestep : 207.1, buffer size : 45096, epsilon : 5.3%
# of episode :560, Avg timestep : 234.8, buffer size : 49812, epsilon : 5.2%
# of episode :580, Avg timestep : 209.8, buffer size : 50000, epsilon : 5.1%
```

기타 코드

https://github.com/seungeunrho/minimalRL

감사합니다

seung_eun07@naver.com