

# Lab 01: Introduction to Reinforcement Learning with OpenAI Gym

Junyeong Park

# Reinforcement Learning

## Definition

강화학습은 어떤 환경 안에서 정의된 에이전트가 현재의 상태를 인식하여, 선택 가능한 행동 중 **보상을 최대화**하는 행동 혹은 행동 순서를 선택하는 방법이다.

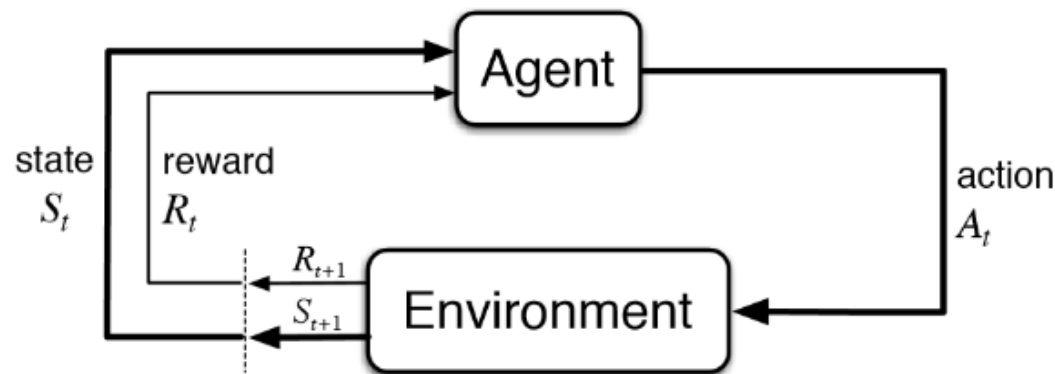
강화학습은 정답이나 잘못된 선택에 대한 정정이 주어지지 않는다.

→ 환경과 에이전트가 상호작용하며 행동에 대한 **보상만 주어진다**.

(예시)

- [AlphaGo](#)
- [AlphaStar](#)

# Reinforcement Learning



- 환경은 에이전트에게 reward와 observation을 제공
- 에이전트는 환경에서 자신의 상태를 인식한 후 행동
- 보상을 통해 에이전트는 어떤 것이 더 좋은 행동인지 알 수 있게 된다.

# FrozenLake-v0

- $4 \times 4$  크기의 맵에서 장애물을 피해 목표에 도달해야 한다.  
S : 시작 지점      F : 얼어 있는 지점 (안전함)  
G : 도착 지점      H : 구멍 (빠지면 게임 오버)
- 도착하면 +1의 보상을, 그 외의 상황엔 0의 보상을 받는다.
- 매 시간 상/하/좌/우로 이동할 수 있다.
- 맵 밖으로 이동할 순 없다.  
(이동해도 게임오버는 안됨)

S	F	F	F
F	H	F	H
F	F	F	H
H	F	F	G

# Markov Decision Process

강화학습 문제를 풀 때 MDP를 정의하여 푼다.

## Definition

Markov Decision Process는  $\langle S, A, P, R, \gamma \rangle$ 의 튜플이다.

- $S$ 는 상태(State)이다.
- $A$ 는 행동(Action)이다.
- $P$ 는 상태 변환 확률(State Transition Probability)이다.
- $R$ 은 보상 함수(Reward Function)이다.
- $\gamma$ 는 감가율(Discount Factor)이다.

# State


에이전트가 관찰 가능한 상태의 집합 :  $\mathcal{S}$

시간  $t$ 에서의 상태  $s_t$ 가 어떤 상태  $s$ 임은 다음과 같이 표현한다.

$$s_t = s$$

## State vs Observation

- State는 관측 불가능한 요소도 포함
- Observation은 상태 중 관측 가능한 요소

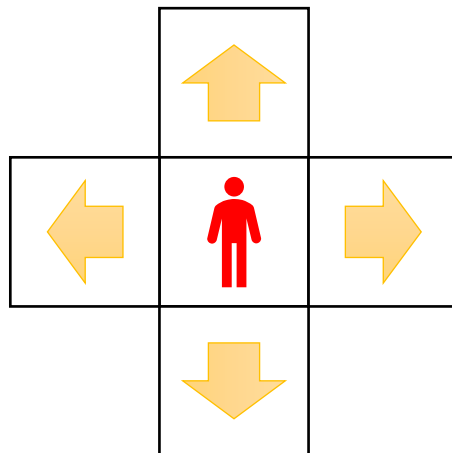
S	F	F	F
	H	F	H
F	F	F	H
H	F	F	G

# Action

에이전트가 어떤 상태에서 할 수 있는 행동의 집합 :  $A$

시간  $t$ 에서 에이전트가 어떤 행동  $a$ 를 함은 다음과 같이 표현한다.

$$A_t = a$$



# Reward Function

에이전트가 학습할 수 있는 **유일한 정보**

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

- 시간  $t$ 에서 상태가  $S_t = s$ 일 때 행동  $A_t = a$ 를 했을 때 받을 보상에 대한 기댓값
- 보상함수에 따라 에이전트의 행동 양상이 달라질 수 있다.



# State Transition Probability

상태의 변화에는 확률적인 요인이 들어간다.

→ 이를 수치적으로 표현한 것이 상태 변환 확률 (각 상태로 변할 확률)

$$P_{s \rightarrow s'}^a = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$$

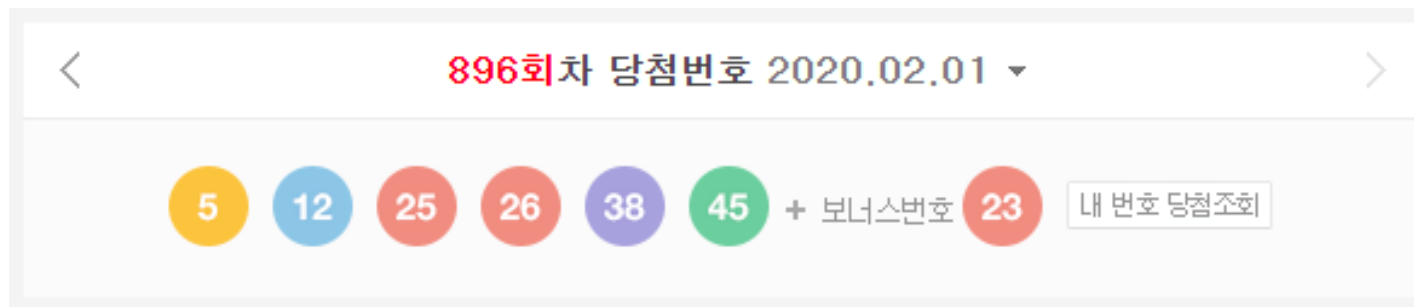
(예시)

- 미로 찾기에서 벽을 뚫을 순 없다. 이때의  $P_{s \rightarrow s'}^a = 0$  이다.
- 어떤 문은  $\frac{1}{3}$ 의 확률로 열릴 때 이때의  $P_{s \rightarrow s'}^a = \frac{1}{3}$  이다.

# Discount Factor

- 에이전트는 항상 현재 시점에서 판단을 내리기 때문에 현재에 가까운 보상일수록 더 큰 가치를 갖는다.
  - 에이전트는 그 보상을 얼마나 시간이 지나서 받는지를 고려해 감가 시켜 **현재의 가치**로 따진다.
- 보상의 크기가 100일 때, 에이전트가 지금 바로 보상을 받을 땐 100 그대로 받아들이지만 현재로부터 일정 시간이 지나서 보상을 받을 경우엔 크기가 100이라 생각하지 않는다.

# Discount Factor



# Discount Factor

A 은행

"당첨금 1억 원을 지금 당장 드리겠습니다."

≠

B 은행

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

# Discount Factor

A 은행

"당첨금 1억 원을 지금 당장 드리겠습니다."

≡

B 은행

"지금 당장 받으면 막쓰다가 탕진할 가능성이 크니 10년 후에 당첨금 1억  
원에 **이자까지** 드리겠습니다."

# Discount Factor

- 감가율은 같은 크기의 보상이 시간이 지날수록 가치가 줄어드는 것을 표현
- $\gamma$ 로 표기하며  $\gamma \in [0, 1]$

현재의 시간  $t$ 로부터 시간  $k$ 가 지난 이후 보상  $R_{t+k}$ 는  $\gamma^{k-1}R_{t+k}$ 가 된다.

(예시)

$\gamma = 0.9$ 라 하고,  $R_{t+1}, R_{t+2}, R_{t+3} = 100$ 이라 하면 각각의 현재 시점의 가치는 다음과 같다.

- $R_{t+1}$ 의 현재 가치:  $R_{t+1} = 100$
- $R_{t+2}$ 의 현재 가치:  $\gamma R_{t+2} = 0.9 \times 100 = 90$
- $R_{t+3}$ 의 현재 가치:  $\gamma^2 R_{t+3} = 0.9^2 \times 100 = 81$

→ 같은 크기의 보상이지만 현재로부터 먼 시점의 보상일수록 가치가 작아진다.

# Policy

모든 상태에서 에이전트가 할 행동

- 상태가 입력으로 들어오면 행동을 출력하는 일종의 함수
- 강화학습 문제는 **최적 정책**을 찾는 것이다.

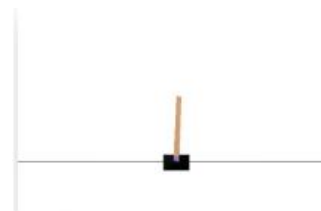
시간  $t$ 에 에이전트가  $S_t = s$ 에 있을 때 가능한 행동 중  $A_t = a$ 를 할 확률은 다음과 같이 표기한다.

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

# OpenAI Gym

- 다양한 강화학습 환경을 제공하는 툴킷
- 게임 환경
  - Atari
  - Classic Control
  - Continuous Control (MuJoCo)
- Robotics
- Algorithm

OpenAI Gym 주소 : <https://gym.openai.com>



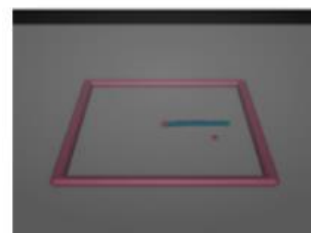
CartPole-v0  
Balance a pole on a cart  
(for a short time).



MountainCar-v0  
Drive up a big hill.



Pendulum-v0  
Swing up a pendulum.



Reacher-v2  
Make a 2D robot reach to a  
randomly located target.



InvertedPendulum-v2  
Balance a pole on a cart.



MsPacman-ram-v0



# Environment

환경은 다음과 같이 만들 수 있다.

## Source Code

```
import gym

env = gym.make('FrozenLake-v0')
```

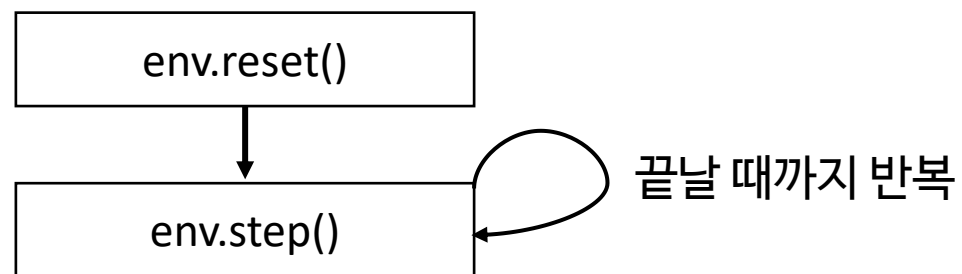
# Environment

환경엔 다음의 멤버가 존재한다.

- `action_space` : 에이전트가 할 수 있는 행동의 집합
- `observation_space` : 환경이 에이전트에게 주는 정보의 집합 (reward 제외)
- `reset()` : 환경을 초기의 상태로 초기화하는 메소드
- `step()` : 에이전트의 행동을 수행하고 에이전트에게 다음 상태, 보상 등을 주는 메소드
- `render()` : 환경을 보여주는 메소드

# Environment

환경과 에이전트의 상호작용은 다음의 흐름대로 진행된다.



# Practice



Google Colab 링크

<http://bitly.kr/BHjazAUhRf>

# Value Function

에이전트는 어떤 행동이 좋을지 어떻게 판단할 수 있을까?

→ 현재 상태에서 앞으로 받을 보상을 고려해야 한다.

## Definition

현재 상태에서 앞으로 받을 보상의 총합을 반환값(Return)이라 하며, 수식은 다음과 같다.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

# Value Function

에이전트는 에피소드가 끝난 후에야 반환값을 알 수 있다.

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

## Definition

어떤 상태  $s$ 에서의 반환값의 기댓값을 가치함수라 하며, 수식은 다음과 같다.

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

# Value Function

- 지금까지 설명한 가치함수는 상태 가치함수(state-value function)
- 에이전트는 가치함수를 통해 다음에 어떤 상태로 가야할 지 판단할 수 있다.
- 어떤 상태로 가면 좋을지 판단한 후에, 그 상태로 가기 위한 행동을 따져본다.

하지만, 상태 가치함수만 가지고 행동을 선택하기는 어렵다.

- 다음 state에 대한 모든 정보가 필요하다.
- 해당 state로 가기 위해선 어떻게 해야 하는지도 알아야 한다.

# Action-Value Function

- 어떤 상태에서 각 행동에 대해 따로 가치함수를 만들 수 있다면 에이전트는 굳이 다음 상태의 가치 함수를 따져보지 않아도 된다.
  - 이처럼 어떤 상태에서 어떤 행동이 얼마나 좋은 지 알려주는 함수를 행동 가치함수라 한다.
- 큐 함수(Q Function)

## Definition

어떤 상태  $s$ 에서 정책  $\pi$ 에 따라 어떤 행동  $a$ 를 했을 때 반환값의 기댓값을 행동 가치함수라 하며, 수식은 다음과 같다.

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



# Action-Value Function

가치함수와 큐 함수는 다음과 같은 관계를 갖는다.

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

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

# Bellman Equation

$$\begin{aligned}v(s) &= \mathbb{E}[G_t \mid S_t = s] \\&= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \cdots \mid S_t = s] \\&= \mathbb{E}[R_{t+1} + \gamma(R_{t+2} + \cdots) \mid S_t = s] \\&= \mathbb{E}[R_{t+1} + \gamma G_{t+1} \mid S_t = s]\end{aligned}$$

위 수식에서 가치함수로 가치함수를 표현하는 다음과 같은 수식을 얻을 수 있다.

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

위 수식을 Bellman Equation이라 한다.

# Bellman Equation

큐함수도 마찬가지로

$$\begin{aligned} q_{\pi}(s, a) &= \mathbb{E}_{\pi}[G_t \mid S_t = s, A_t = a] \\ &= \mathbb{E}_{\pi}[R_{t+1} + \gamma R_{t+2} + \dots \mid S_t = s, A_t = a] \\ &= \mathbb{E}_{\pi}[R_{t+1} + \gamma(R_{t+2} + \dots) \mid S_t = s, A_t = a] \\ &= \mathbb{E}_{\pi}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] \end{aligned}$$

위 수식에서 큐 함수로 큐 함수를 표현하는 다음과 같은 수식을 얻을 수 있다.

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

# Q-Learning

큐 함수가 잘 만들어졌다면 최적의 정책  $\pi_*$ 은 다음과 같이 정의할 수 있다.

$$\pi_*(a \mid s) = \operatorname{argmax}_{a \in A} Q(s, a)$$

최적의 정책을 찾기 위해선 큐 함수를 잘 만들어야 한다.

→ 큐 함수를 업데이트 하는 방법 중 하나를 Q-Learning이라 한다.

# Q-Learning

Q-Learning의 알고리즘은 다음과 같다.

1. 큐 함수  $Q(s, a)$ 를 초기화 한다.
2. 각 에피소드마다 반복한다.
  1. 상태  $s$ 를 초기화 한다.
  2. 에피소드가 끝나기 전까지 에피소드의 각 step마다 반복한다.
    1. 행동  $a$ 를  $Q$ 를 이용해 선택한다. ( $\epsilon$ -greedy 등의 방법 사용)
    2. 행동  $a$ 를 수행하고 보상  $R$ 과 다음 상태  $s'$ 을 얻는다.
    3. 다음 수식에 따라  $Q$ 함수를 업데이트 한다.
$$Q(s, a) \leftarrow Q(s, a) + \alpha[R + \gamma \max_{a'} Q(s', a) - Q(s, a)]$$
    4. 상태  $s$ 를  $s'$ 으로 변경한다.

# Q-Learning

업데이트 수식을 자세히 살펴보자.

$$Q(s, a) \leftarrow Q(s, a) + \alpha[R + \gamma \max_{a'} Q(s', a') - Q(s, a)]$$

- $\alpha[(\text{목표}) - (\text{추정값})]$ 의 관점에서 보면 업데이트의 목표는  $R + \gamma \max Q(s', a')$ 임을 알 수 있다.
- 그런데  $R + \gamma \max Q(s', a')$ 은 큐 함수의 정의와 유사하다.
- 에이전트는 최적의 행동을 하므로  $Q(s', a')$ 을  $\max Q(s', a')$ 으로 한 것이다.
- 업데이트를 거듭할수록 최적의 큐 함수를 얻을 수 있다.

# Practice



Google Colab 링크

<http://bitly.kr/tfbdlbN26>