Project 3

Instructor: Kyuree AHN

TA: Kanghoon Lee

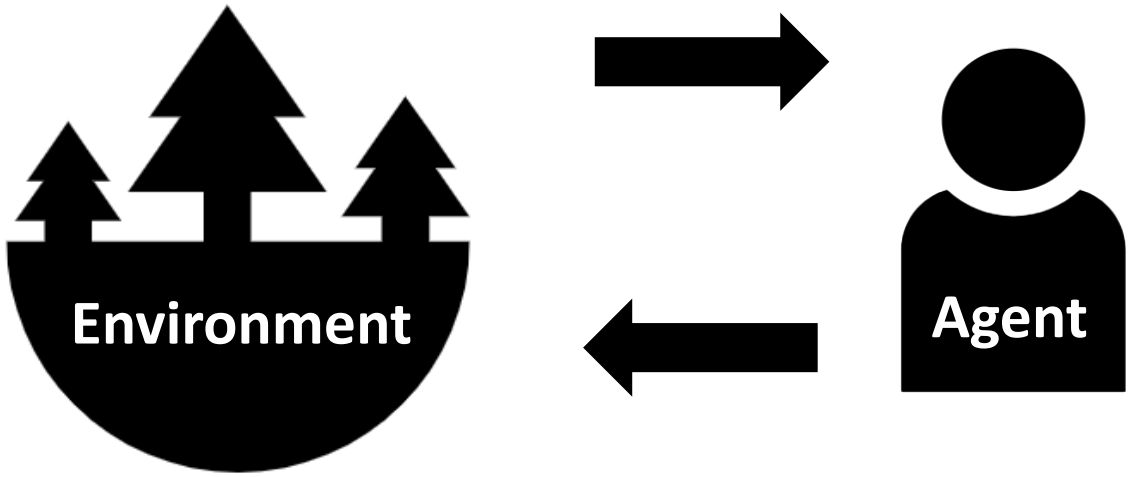
Before beginning…

Survey

1. 강화학습이 무엇인지 알고 있다
2. Q-function이 무엇인지 알고 있다
3. Q learning 과 Actor Critic 의 차이를 알고 있다
4. Q-learning을 코드로 짤 수 있을 것 같다

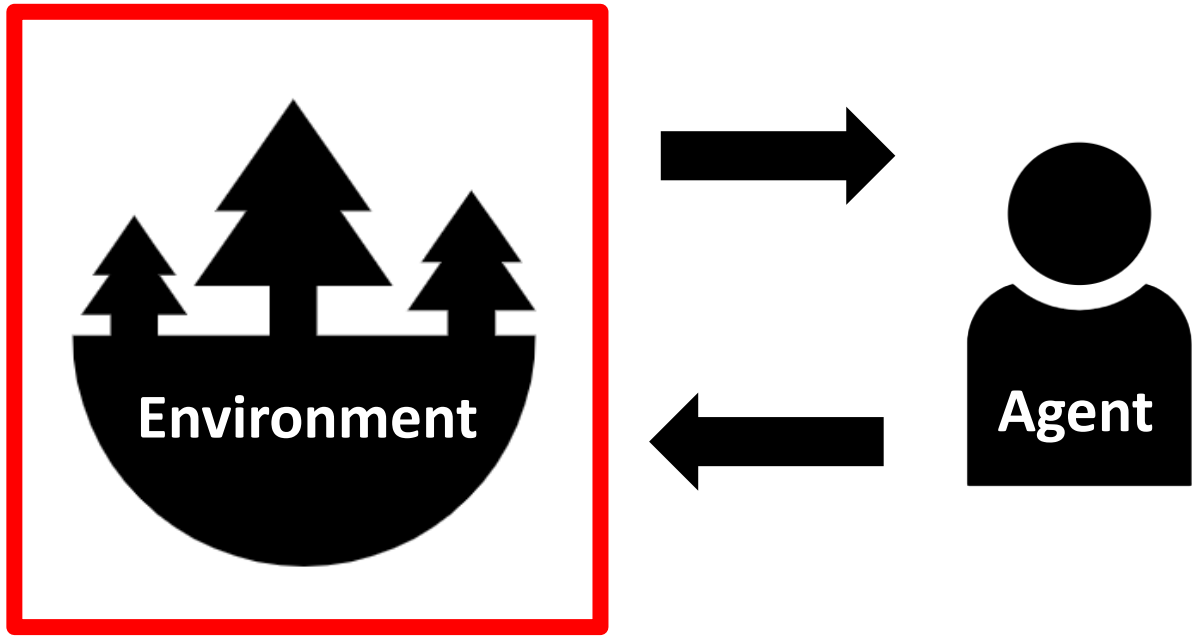
0. Recap: Reinforcement Learning

Reinforcement learning at a glance



Agent 가 환경 (**Environment**)과 **상호작용**하며 정책 (**policy**) 를 배워 나가는 것

Reinforcement learning at a glance

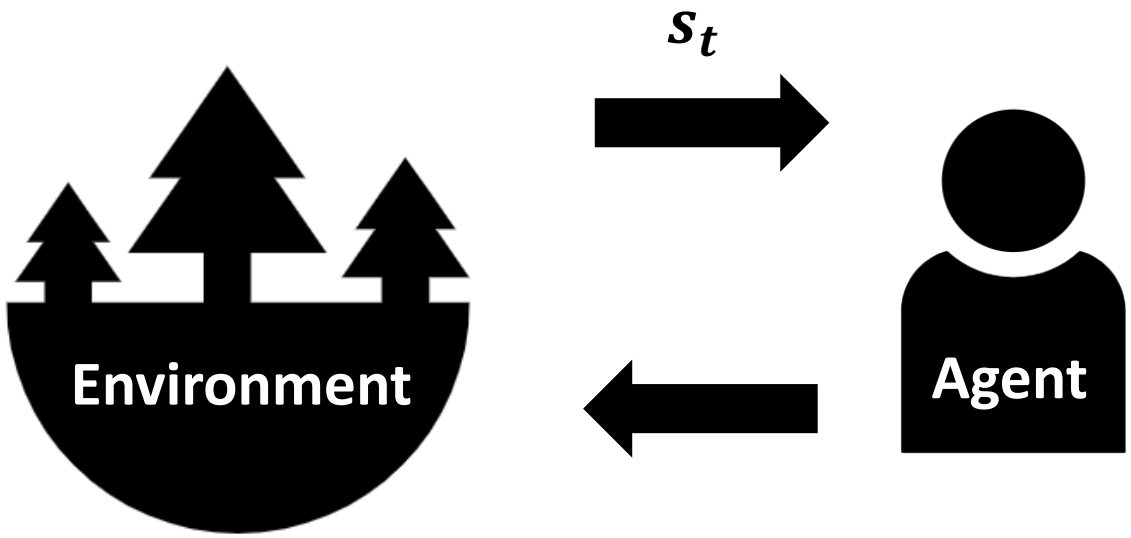


환경 (**Environment**)

- 에이전트가 행동을 했을 때 (내재된)규칙에 따라 '무언가' 를 제공해주는 역할

Ex) 바둑 게임, 시뮬레이터, 미로 찾기 …

Reinforcement learning at a glance



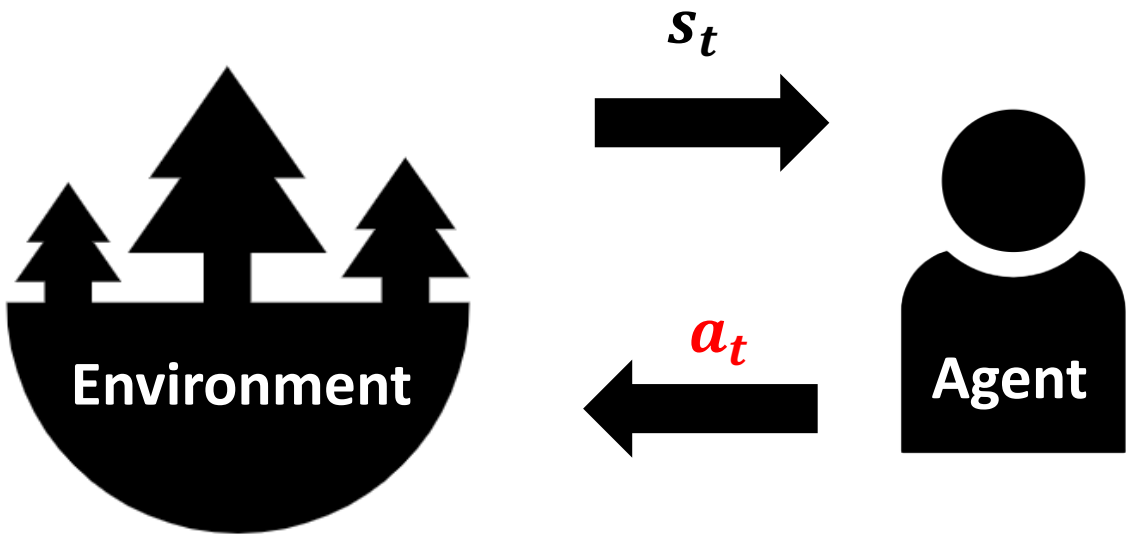
상태 (**State**)

- 현재 시점 (t) 에서 환경이 어떤 상태인지를 표현하는 값, s_t 로 표기

Ex) 바둑판에 현재 바둑돌이 어떻게 놓여 있는가? 미로에서 내 위치가 어디인가?

: 환경은 agent에게 s_t 를 제공.

Reinforcement learning at a glance



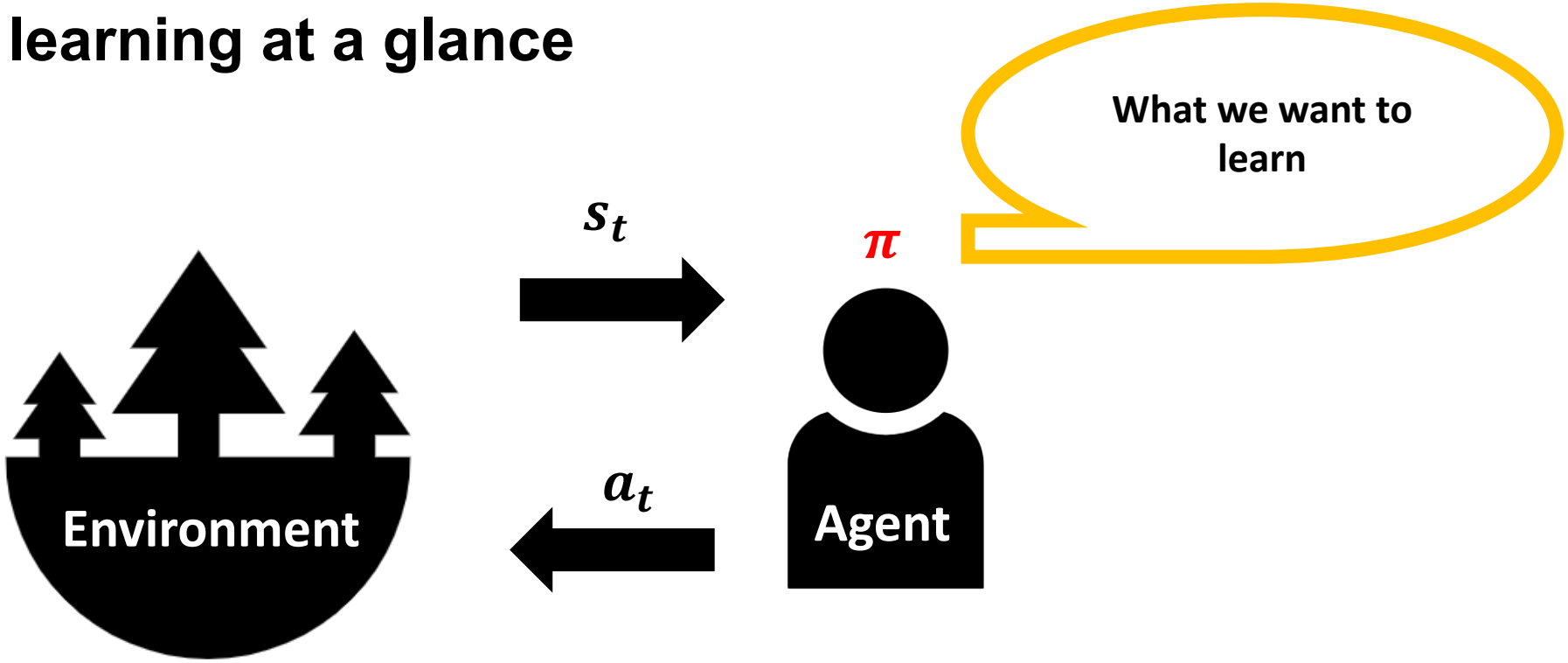
행동 (**Action**)

- 에이전트가 특정 시간 t 에 상태 s_t 를 보고 취하는 행동, a_t 로 표기

Ex) 바둑돌을 (2,2) 위치에 놓는다, 미로에서 ↑ 방향으로 간다

그런데, action은 어떻게 정하지?

Reinforcement learning at a glance



정책 (**Policy**)

- 에이전트가 특정 상태에서 행동을 할 규칙이며, s_t 에서 a_t 를 취할 확률분포로 표현됨

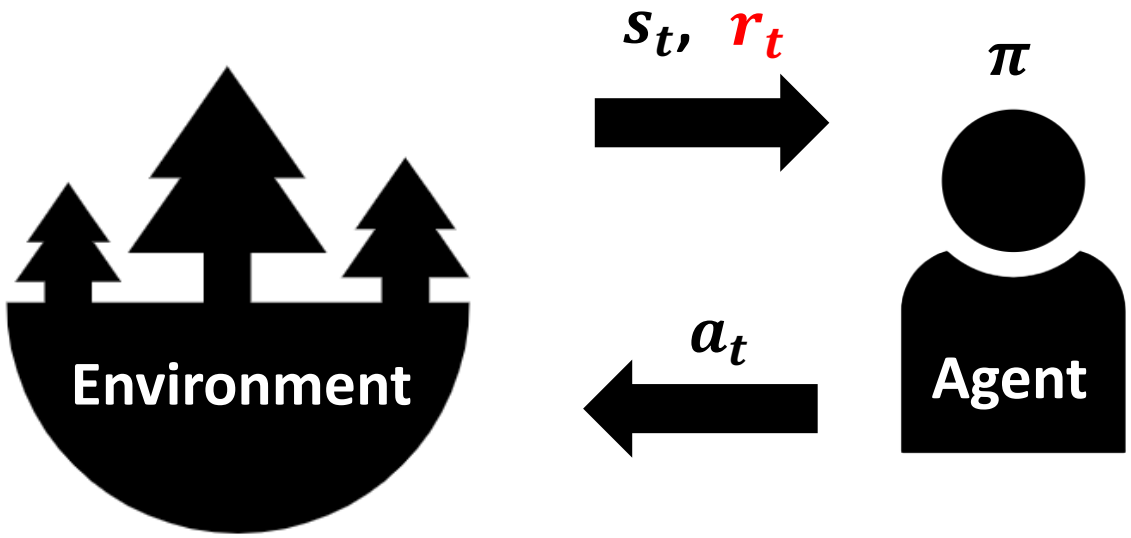
$$a_t \sim \pi(\cdot \mid s_t)$$

  - **최적 정책 (optimal policy) 는 π^* 로 표기**

Ex) 미로의 현재 위치에서 상/하/좌/우로 각각 $[0.1, 0.4, 0.3, 0.2]$ 의 확률로 가자!

그런데, 에이전트는 정책을 어떻게 결정하지?

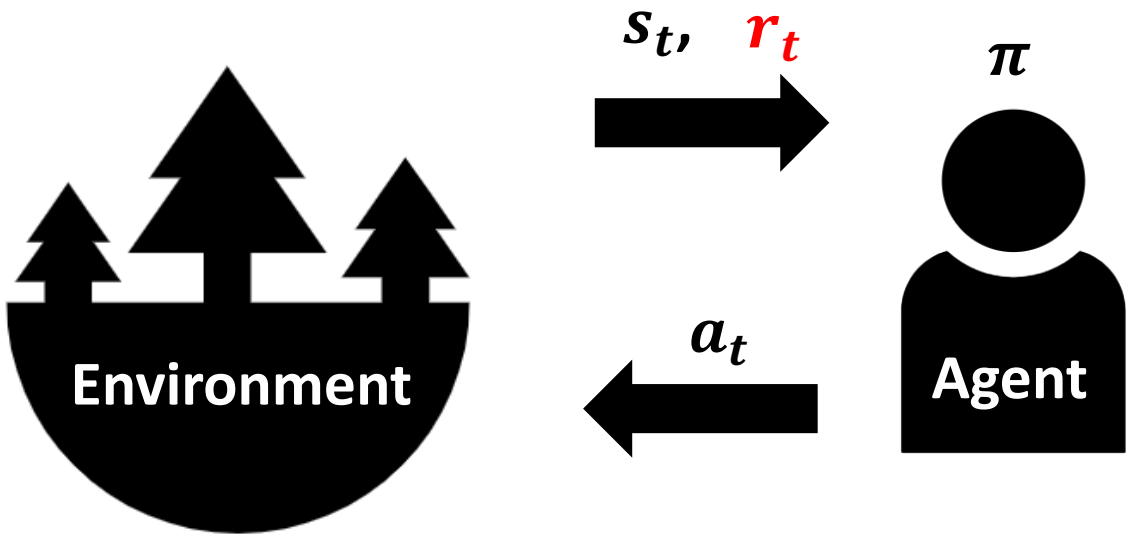
Reinforcement learning at a glance



보상 (**Reward**)

- 에이전트가 어떤 행동을 취했을 때 따라오는 **즉각적인** 이득 (혹은 비용)을 표현한 값.
- 에이전트는 reward를 통해 policy를 업데이트함

Reinforcement learning at a glance



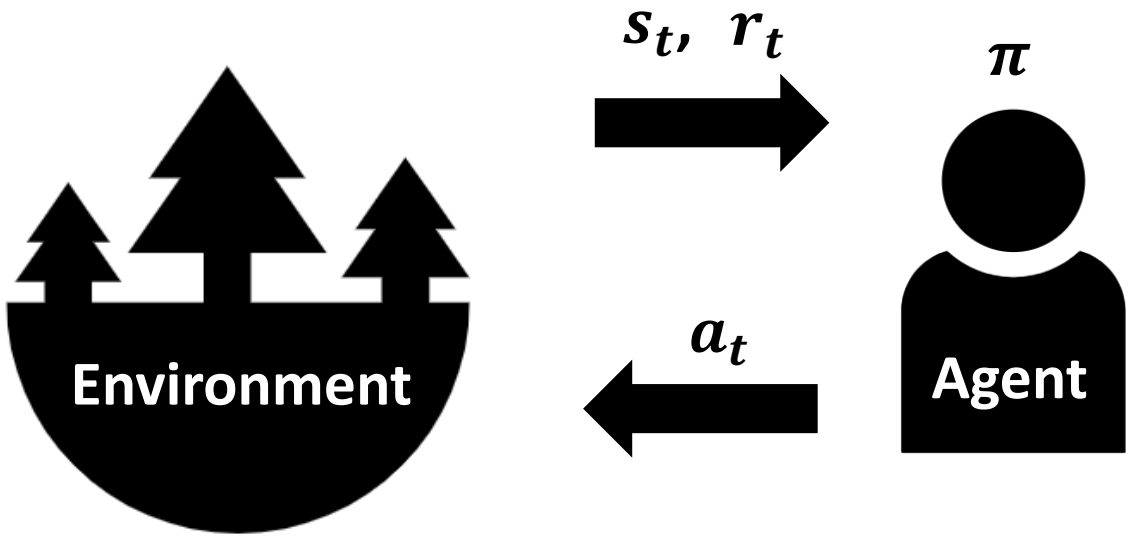
보상 (**Reward**)

- Reward 는 s_t, a_t 그리고 s_{t+1} 에 의해 결정됨. 즉

$$r_t = r(s_t, a_t, s_{t+1})$$

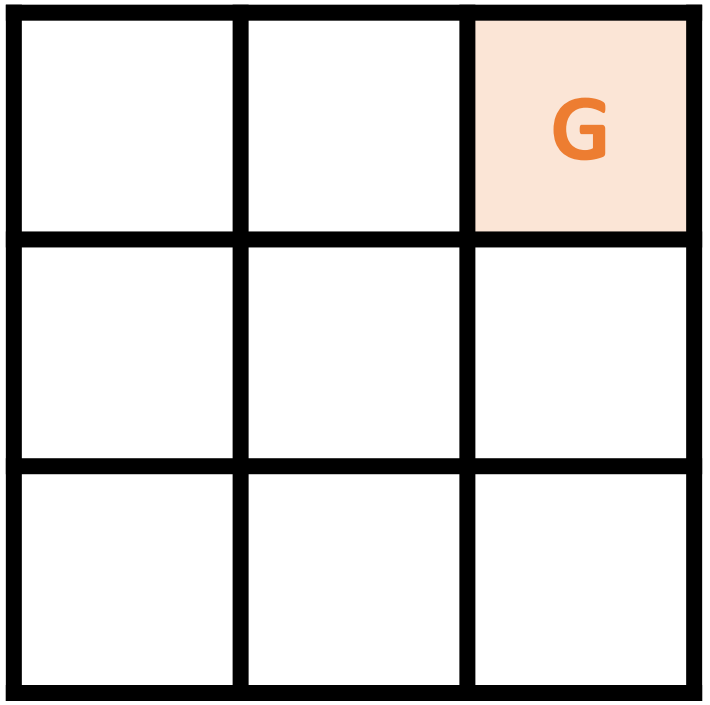
? 예제를 통해 이해해 봅시다!

In summary, 강화학습이란



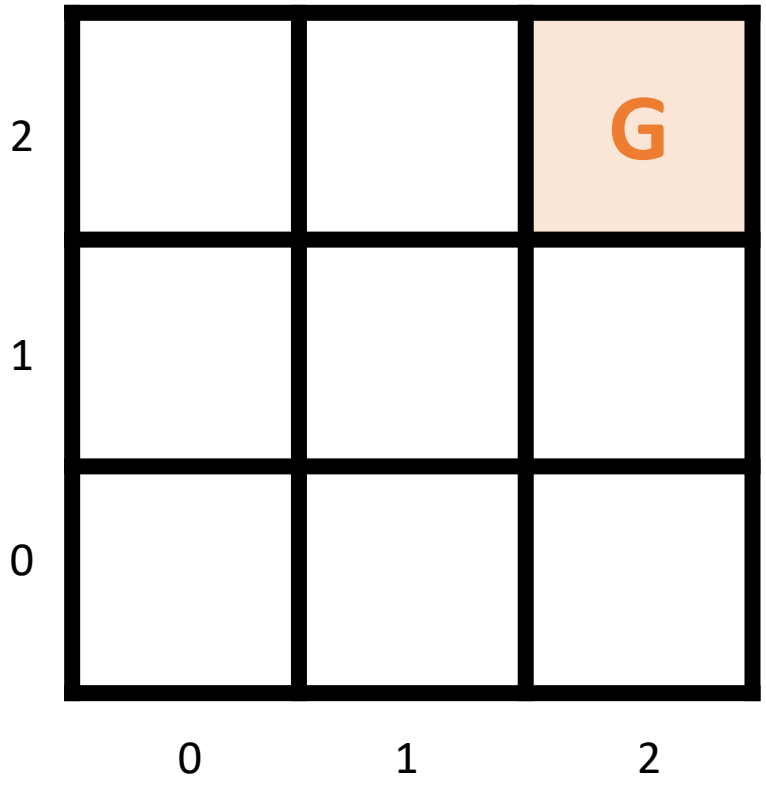
Agent 가 특정 **state** 에서 **action**을 취했을 때 받는 **reward**를 통해 **policy**를 배우는 과정

Example: 다음 미로찾기 문제에서 state, action, reward는 어떻게 정의될까요?



Goal에 도달할 시 +100을 받으며,
벽에 부딪힐 시 -10점을 받음

Example: 다음 미로찾기 문제에서 state, action, reward는 어떻게 정의될까요?



State: 각 셀의 위치

$$s_t \in \{(0,0), (0,1), \dots, (2,2)\}$$

Action: 상/하/좌/우

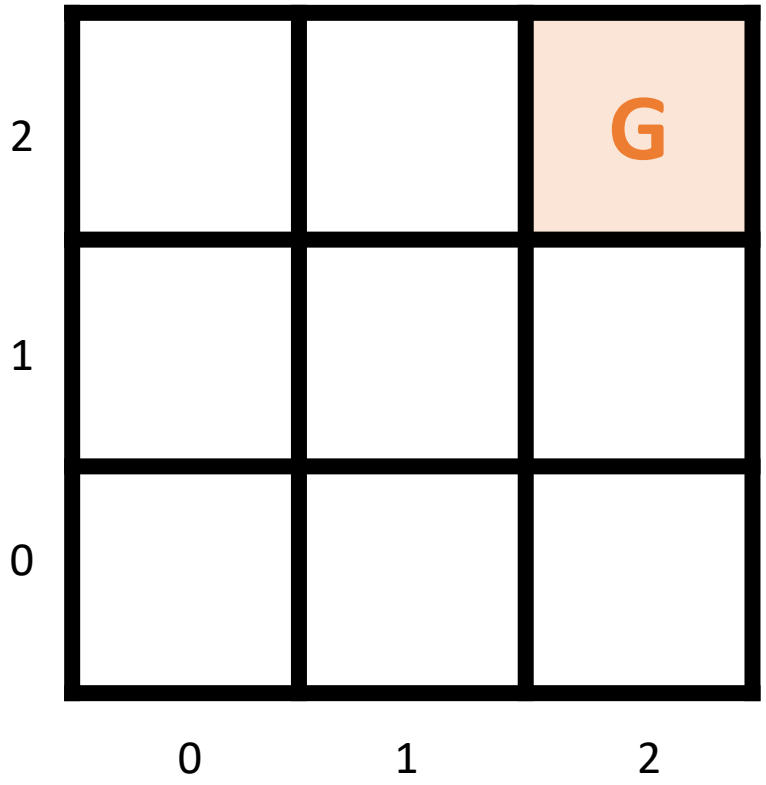
$$a_t \in \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$$

Reward:

!

$$r(s_t, a_t, s_{t+1}) = \begin{cases} 100 & if\ s_{t+1} = (2,2) \\ -10 & if\ s_t = s_{t+1} \\ 0 & otherwise \end{cases}$$

Example: 다음 미로찾기 문제에서 state, action, reward는 어떻게 정의될까요?



State: 각 셀의 위치

$$s_t \in \{(0,0), (0,1), \dots, (2,2)\}$$

Action: 상/하/좌/우

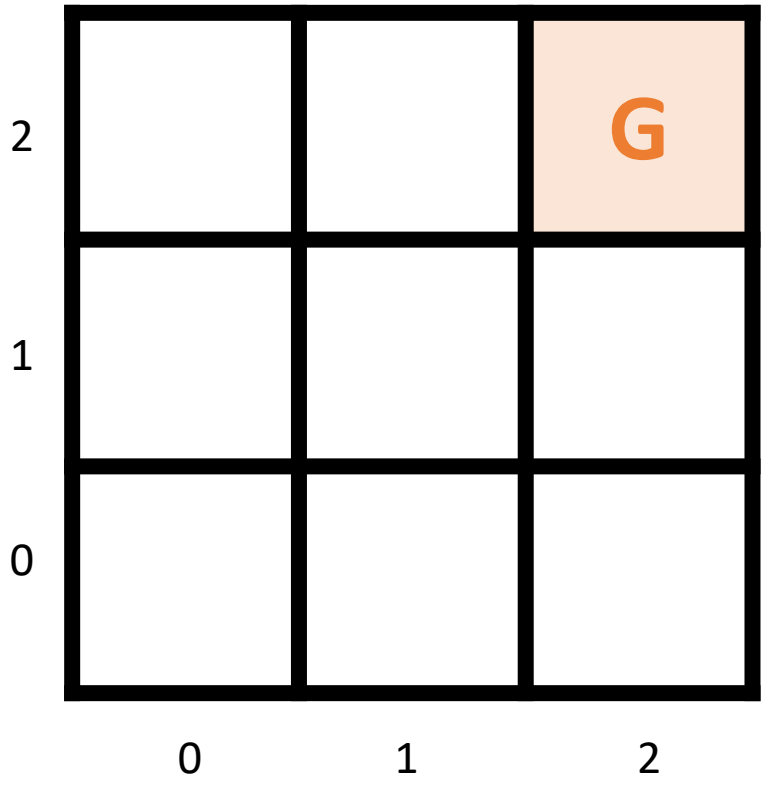
$$a_t \in \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$$

Reward:

$$r(s_t, a_t, s_{t+1}) = \begin{cases} 100 & if\ s_{t+1} = (2,2) \\ -10 & if\ s_t = s_{t+1} \\ 0 & otherwise \end{cases}$$

그러면 $s_t = (2,2)$ 일 때는 어떻게 되나요?

Example: 다음 미로찾기 문제에서 state, action, reward는 어떻게 정의될까요?



State: 각 셀의 위치

$$s_t \in \{(0,0), (0,1), \dots, (2,2)\}$$

Action: 상/하/좌/우

$$a_t \in \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$$

Reward:

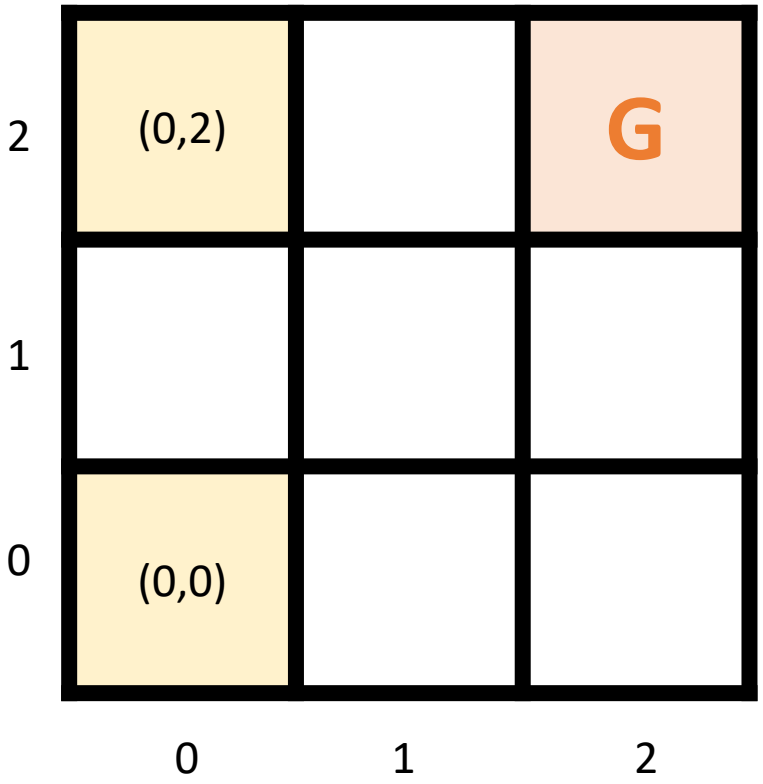
$$r(s_t, a_t, s_{t+1}) = \begin{cases} 100 & if\ s_{t+1} = (2,2) \\ -10 & if\ s_t = s_{t+1} \\ 0 & otherwise \end{cases}$$

그러면 $s_t = (2,2)$ 일 때는 어떻게 되나요?

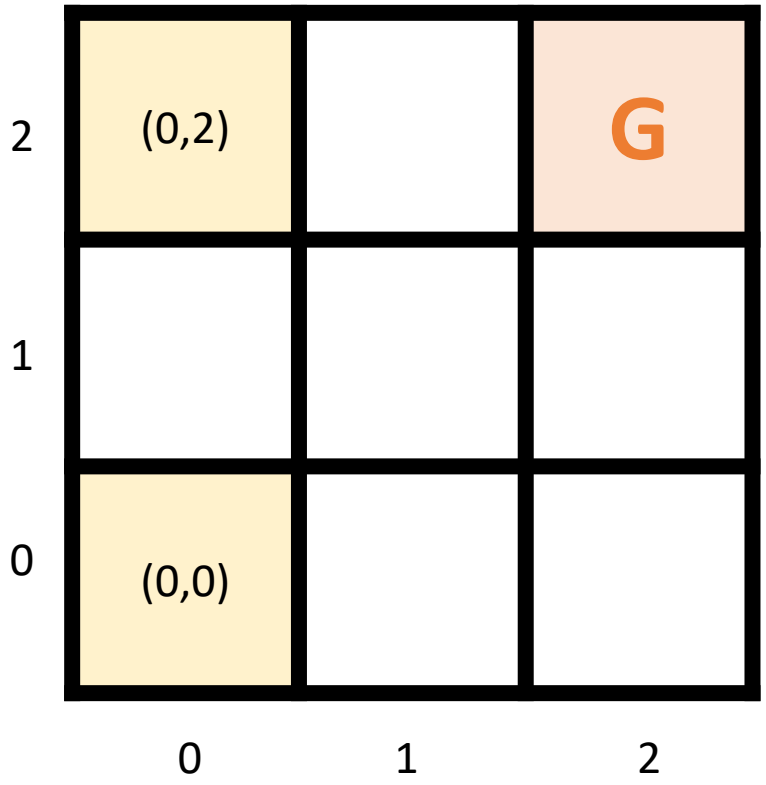
➔ Terminal state라 부릅니다! (episodic case)

Q-function

그렇다면, $s_t = (0, 0)$ 일 때와 $s_t = (0, 2)$ 일 때의 가치는 똑같을까요?



Q-function



그렇다면, $s_t = (0,0)$ 일 때와 $s_t = (0,2)$ 일 때의 가치는 똑같을까요?

Value function (가치판단 함수)

Or, ∞

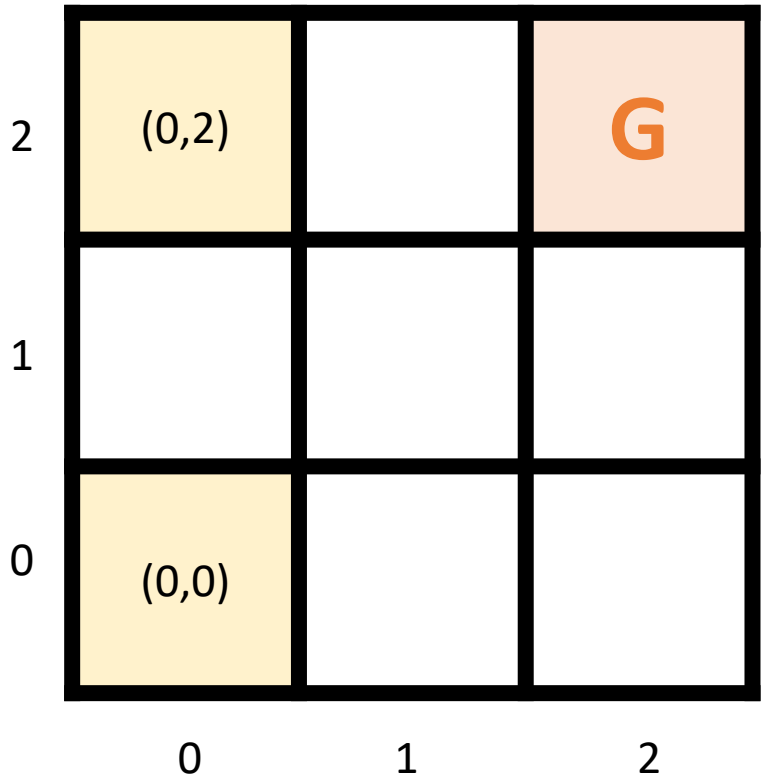
$$V^{\pi}(s_t) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{T-t} \gamma^k r_{t+k} \mid s_t\right]$$

$(0,2)$에서는 $(0,0)$ 에서보다 더 **빠르게** 미래의 reward를 높일 수 있으므로,

$$V\big(s_t = (0,2)\big) > V\big(s_t = (0,0)\big)$$

Q-function

그렇다면, $s_t = (0,0)$ 일 때와 $s_t = (0,2)$ 일 때의 가치는 똑같을까요?



Value function (가치평가 함수)

Or, ∞

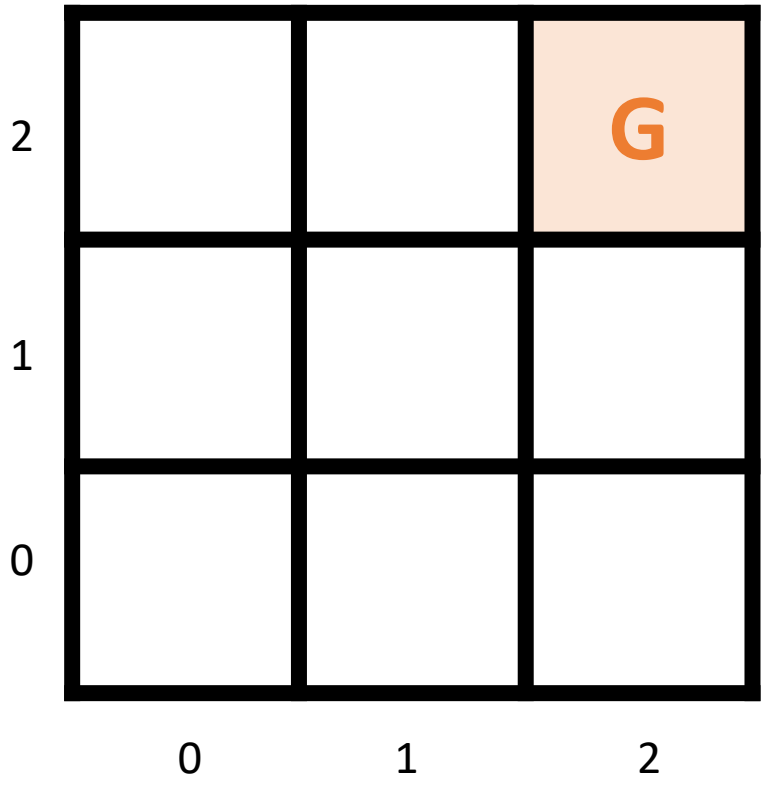
$$V^{\pi}(s_t) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{T-t} \gamma^k r_{t+k} \mid s_t\right]$$

(0,2)에서는 (
높일 수 있으므

미래에 받을 reward에 대한 감가율, discount rate
$\gamma \in (0,1)$

$$V\left(s_t = (0,2)\right) > V\left(s_t = (0,0)\right)$$

Q-function



행동에 대한 가치도 평가할 수 있나요? 예!

Q-function (state-action value, 행동가치평가 함수)

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{T-t} \gamma^k r_{t+k} \mid s_t, a_t\right]$$

$(2,1)$에서는 위로 가면 reward=100을 받고 끝이므로,

$$Q\big((0,2), \uparrow\big) = 100$$

$(1,1)$ 에서는 상/우 action이 좌/하 action보다 좋으므로,

$$Q\big((1,1), \uparrow\big) = Q\big((1,1), \rightarrow\big)$$
$$> Q\big((1,1), \leftarrow\big) = Q\big((1,1), \downarrow\big)$$

Q-learning

- Q function을 갱신하며 배우는 기법
- 초기 $Q(s_t, a_t)$ 는 임의의 값을 가짐 (모든 s_t, a_t에 대해)

$$Q(s_t, a_t) \leftarrow (1-\alpha) \cdot Q(s_t, a_t) + \alpha \cdot \left(r_t + \gamma \cdot \max_a Q(s_{t+1}, a) \right)$$

Q-learning

- 갱신될 Q 값은 다음과 같이 업데이트된다.

$$Q(s_t, a_t) \leftarrow (1-\alpha) \cdot Q(s_t, a_t) + \alpha \cdot \left(r_t + \gamma \cdot \max_a Q(s_{t+1}, a)\right)$$

과거의 Q값과

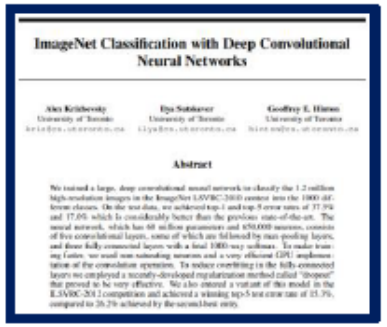
Reward로부터 받은 새로운 정보

를 가중해서 더하자 (weighted sum)

α: learning rate, 새로운 정보에 얼마나 가중치를 더해서 학습할지 결정.

DQN

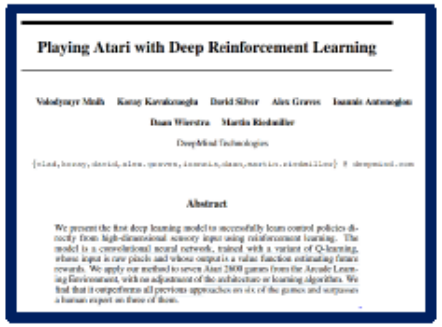
Alexnet (NIPS 2012)


ImageNet Classification with Deep Convolutional Neural Networks

Abstract

Dec 3, 2012

DQN (Arxiv version)


Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih Koray Kavukcuoglu David Silver Alex Graves Ioannis Antonoglou

Daan Wierstra Martin Riedmiller

DeepMind Technologies

Abstract

Dec 19, 2013

DQN (Nature version)


LETTER

doi:10.1038/nature14236

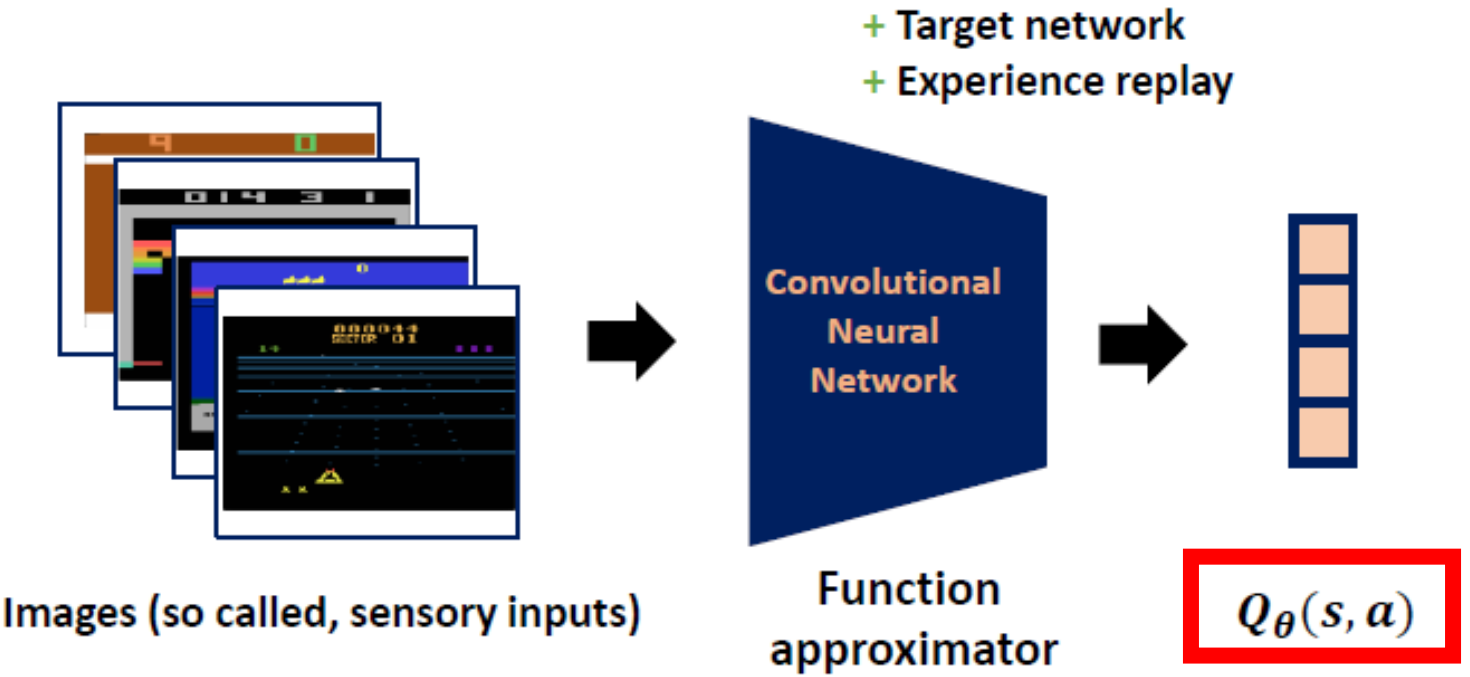
Human-level control through deep reinforcement learning

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]

Feb 26, 2015

- DQN is the "first" deep learning model to successfully learn control policies "directly" from high-dimensional sensory input using reinforcement learning.
- *Not current (2020) SOTA*, but DQN papers proposed **important tricks** for stabilizing the training of deep neural network based RL methods.

DQN



- Overcoming "curse of dimensionality" with function approximators.
- Tricks for stabilizing training procedures:
  - Experience replay *
  - Target network **

* Long-Ji Lin. Reinforcement learning for robots using neural networks. Technical report, DTIC Document, 1993
** Firstly proposed from the 2nd DQN paper.
*** Game images are reproduced from the DQN paper <Playing Atari with Deep RL>

Function approximation

Q learning

$$Q(s_t, a_t) \leftarrow (1-\alpha) \cdot Q(s_t, a_t) + \alpha \cdot \left(r_t + \gamma \cdot \max_a Q(s_{t+1}, a)\right)$$

Q learning with function approximation

Q function을 근사하는 network의 parameter θ

$$Q_\theta(s_t, a_t) \leftarrow (1-\alpha) \cdot Q_\theta(s_t, a_t) + \alpha \cdot \left(r_t + \gamma \cdot \max_a Q_\theta(s_{t+1}, a)\right)$$

Target network

Q learning with function approximation

$$Q_\theta(s_t, a_t) \leftarrow (1-\alpha) \cdot Q_\theta(s_t, a_t) + \alpha \cdot \left(r_t + \gamma \cdot \max_a Q_\theta(s_{t+1}, a)\right)$$

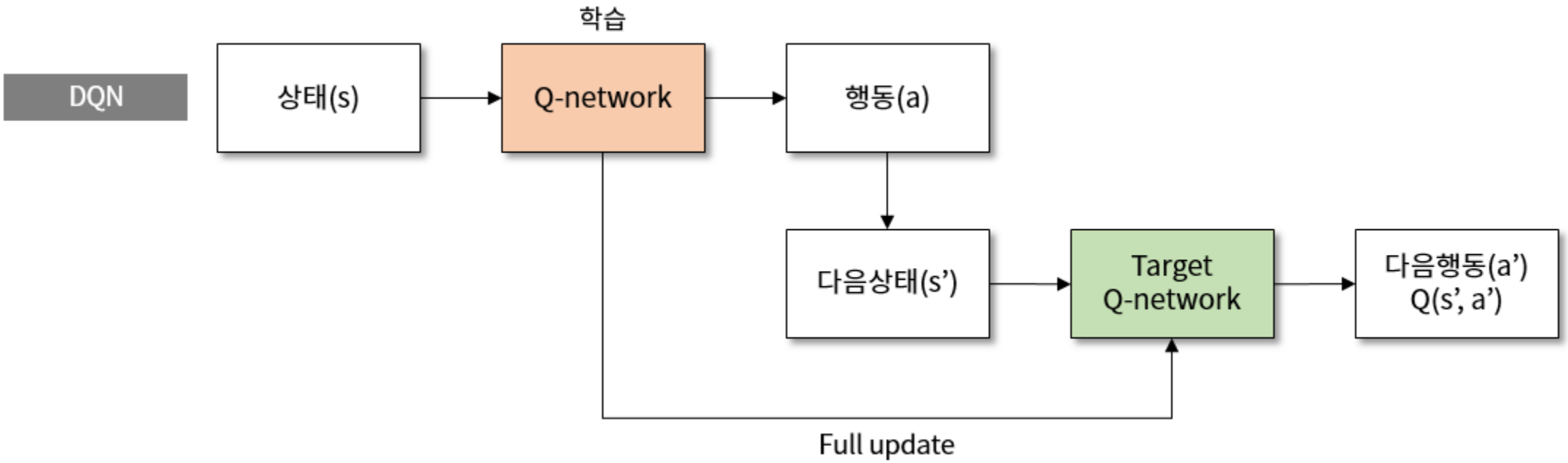
Q learning with function approximation and target network

$$Q_\theta(s_t, a_t) \leftarrow (1-\alpha) \cdot Q_\theta(s_t, a_t) + \alpha \cdot \left(r_t + \gamma \cdot \max_a Q_{\bar{\theta}}(s_{t+1}, a)\right)$$

$Q_{\bar{\theta}}$: target network

- 기존의 Q-network와 구조가 완전히 동일하고 parameter만 다른 $(\theta \neq \bar{\theta})$ 별도의 네트워크

Target network



Target network를 쓰는 이유:

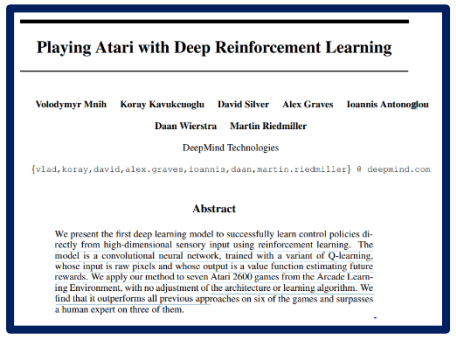
학습할 때마다 Q-network 의 가중치는 변하게 되는데, 목표도 같이 변하기 때문에 일정한 값으로 수렴하는 데에 어려움이 있을 있기 때문

https://greentec.github.io/reinforcement-learning-third/

1. Recap: DDPG

DDPG : Deep Deterministic Policy Gradient

DQN



Playing Atari with Deep Reinforcement Learning

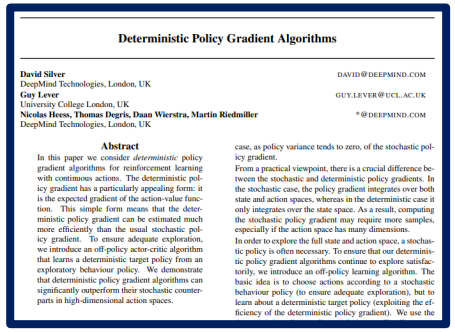
Volodymyr Mnih Koray Kavukcuoglu David Silver Alex Graves Ioannis Antonoglou
Daan Wierstra Martin Riedmiller
DeepMind Technologies
{vlad,koray,david,alex.graves,ioannis,daan,martin.riedmiller} @ deepmind.com

Abstract

We present the first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning. The model is a convolutional neural network, trained with a variant of Q-learning, whose input is raw pixels and whose output is a value function estimating future rewards. We apply our method to seven Atari 2600 games from the Arcade Learning Environment, with no adjustment of the architecture or learning algorithm. We find that it outperforms all previous approaches on six of the games and surpasses a human expert on three of them.

Dec 19, 2013

Deterministic Policy Gradient



Deterministic Policy Gradient Algorithms

David Silver DAVID@DEEPMIND.COM
DeepMind Technologies, London, UK
Guy Lever GUY.LEVER@UCL.AC.UK
University College London, UK
Nicolas Heess, Thomas Degris, Daan Wierstra, Martin Riedmiller *@DEEPMIND.COM
DeepMind Technologies, London, UK

Abstract

In this paper we consider *deterministic* policy gradient algorithms for reinforcement learning with continuous actions. The deterministic policy gradient has a particularly appealing form: it is the expected gradient of the action-value function. This simple form means that the deterministic policy gradient can be estimated much more efficiently than the usual stochastic policy gradient. To ensure adequate exploration, we introduce an off-policy actor-critic algorithm that learns a deterministic target policy from an exploratory behaviour policy. We demonstrate that deterministic policy gradient algorithms can significantly outperform their stochastic counterparts in high-dimensional action spaces.

case, as policy variance tends to zero, of the stochastic policy gradient.
From a practical viewpoint, there is a crucial difference between the stochastic and deterministic policy gradients. In the stochastic case, the policy gradient integrates over both state and action spaces, whereas in the deterministic case it only integrates over the state space. As a result, computing the stochastic policy gradient may require more samples, especially if the action space has many dimensions.
In order to explore the full state and action space, a stochastic policy is often necessary. To ensure that our deterministic policy gradient algorithms continue to explore satisfactorily, we introduce an off-policy learning algorithm. The basic idea is to choose actions according to a stochastic behaviour policy (to ensure adequate exploration), but to learn about a deterministic target policy (exploiting the efficiency of the deterministic policy gradient). We use the

ICML 2014

DDPG

Published as a conference paper at ICLR 2016

CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING

Timothy P. Lillicrap; Jonathan J. Hunt; Alexander Pritzel, Nicolas Heess,
Tom Erez, Yuval Tassa, David Silver & Daan Wierstra
Google Deepmind
London, UK
{countzero, jjhunt, apritzel, heess,
etom, tassa, davidsilver, wierstra} @ google.com

ABSTRACT

We adapt the ideas underlying the success of Deep Q-Learning to the continuous action domain. We present an actor-critic, model-free algorithm based on the deterministic policy gradient that can operate over continuous action spaces. Using the same learning algorithm, network architecture and hyper-parameters, our algorithm robustly solves more than 20 simulated physics tasks, including classic problems such as cartpole swing-up, dexterous manipulation, legged locomotion and car driving. Our algorithm is able to find policies whose performance is competitive with those found by a planning algorithm with full access to the dynamics of the domain and its derivatives. We further demonstrate that for many of the tasks the algorithm can learn policies "end-to-end": directly from raw pixel inputs.

ICLR 2016

- DDPG는 "**Deterministic Policy Gradient**" 의 심층학습 버전.
- DQN에서 사용된 심층강화학습을 안정화 하는 기법들이 사용.

DDPG : Motivation

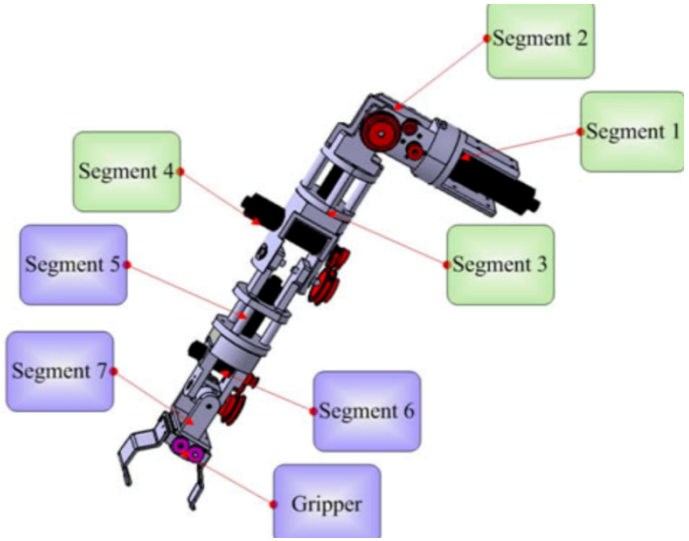
- DQN 방법으로는 **discrete**한 action만 다룰 수 있다.

- DQN으로 Continuous한 action을 다루려면, **discretizing (**행동 이산화**)** 를 해야한다

  - 오른쪽과 같은 7개의 관절을 가진 로봇 팔을 제어한다고 생각해보자.
  - 각 segment당 움직임을 $a_i = \{-k, 0, +k\}$ 로 discretize 한다고 가정하면,
  - Action space : 3 X 3 X … X 3 = 3^7 = 2187 가지

  ➢ Continuous action을 완벽하게 표현할 수 없고,

  ➢ Action space가 exponential하게 늘어남



DDPG : Idea & Contribution

- What we want
  - 연속적인 정책 함수 : $\pi(a|s) \in [a_{min}, a_{max}]$

- Idea
  - 네트워크의 output을 바로 action으로 사용하는 deterministic한 방법을 사용하자.
  - DPG (Deterministic Policy Gradient)에 기반.
  - **Actor-Critic** 을 활용.

- Contribution
  - High-dimensional + Continuous 한 action space에서 학습 가능

DDPG : Background

- actions $a_t \in \mathbb{R}^N$, action space $A = \mathbb{R}^N$

- policy $\mu : S \rightarrow A$
  - policy의 output을 직접 action으로 사용

- discounted future reward : $R_t = \sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i)$
  - Discount factor : $\gamma \in [0,1]$

- 목표 : **expected return을 maximize하는 policy**를 학습하는 것

Critic의 Loss function

- Action value function

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{s_t \sim E, a_t \sim \pi}[R_t | s_t, a_t]$$

- Bellman equation

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}[r(s_t, a_t) + \gamma \mathbb{E}_{a_{t+1} \sim \pi}[Q^{\pi}(s_{t+1}, a_{t+1})]]$$

- Bellman equation 에 deterministic policy(μ)를 적용 : Expectation term을 빠져나오게 된다.

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}[r(s_t, a_t) + \gamma Q^{\mu}(s_{t+1}, a_{t+1})]$$

- Loss function

$$L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i))^2$$
$$y_i = r_i + \gamma Q(s_{i+1}, \mu(s_{i+1}))$$

i 는 batch index

Actor의 Loss function : from DPG(Deterministic Policy Gradient)

- Policy π 를 따랐을 때의 state distribution $\rho^{\pi}(s')$ 은 다음과 같이 계산 가능

$$\rho^{\pi}(s') \stackrel{\text{def}}{=} \int_{s \in \mathcal{S}} \sum_{t=1}^{\infty} \gamma^{t-1} p_1(s) p(s \to s', t, \pi) \; ds$$

$p(s \to s', t, \pi)$: 정책 π를 고려할 때, 현재 상태 s 에서 t 시점 이후에 상태 s' 로 이동할 확률

- Improper state distribution:
  - 임의의 state s 에서, state s'에 도달할 확률.
  - 하지만, 먼 미래의 도착 확률을 감가해서 고려한다.

$$\begin{aligned} J(\pi_\theta) &= \int_{s \in \mathcal{S}} \rho^{\pi_\theta}(s) \int_{a \in \mathcal{A}} \pi_\theta(s, a) r(s, a) \, da \, ds \\ &= \mathbb{E}_{s \sim \rho^{\pi_\theta}, a \sim \pi_\theta}[r(s, a)] \end{aligned}$$

Actor의 Loss function : from DPG(Deterministic Policy Gradient)

Stochastic policy의 cost function $J(\boldsymbol{\pi_\theta})$:

$$J(\boldsymbol{\pi_\theta}) = \int_{s\in\mathcal{S}} \rho^{\pi_\theta}(s) \int_{a\in\mathcal{A}} \pi_\theta(s,a) r(s,a)\, da\, ds$$
$$= \mathbb{E}_{s\sim\rho^{\pi_\theta}, a\sim\pi_\theta}[r(s,a)]$$

Deterministic policy의 cost function $J(\boldsymbol{\mu_\theta})$:

$$J(\boldsymbol{\mu_\theta}) = \int_{s\in\mathcal{S}} \rho^{\mu_\theta}(s)\, r\big(s, \mu_\theta(s)\big)\, ds$$
$$= \mathbb{E}_{s\sim\rho^{\pi_\theta}}\big[r\big(s, \mu_\theta(s)\big)\big]$$

Actor의 Gradient

- $J(\boldsymbol{\mu_\theta})$ 의 gradient:

$$\nabla_\theta J(\mu_\theta) = \int_{s \in \mathcal{S}} \rho^{\mu_\theta}(s)\, \nabla_\theta \mu_\theta(s)\, \nabla_a Q(s,a)\Big|_{a=\mu_\theta(s)}\, ds$$
$$= \mathbb{E}_{s\sim\rho^{\pi_\theta}}\left[\nabla_\theta \mu_\theta(s)\, \nabla_a Q(s,a)\Big|_{a=\mu_\theta(s)}\right]$$

이 알고리즘을 구현하기 위해서 필요한 것?:

1) θ에 대한 gradient를 계산하기 쉬운 결정적 정책 함수 $\mu_\theta(s)$ → **Neural Network?**
2) a에 대한 gradient를 계산하기 쉬운 $Q(s,a)$.

Challenges

- 학습이 불안정함
  - DQN처럼 target network를 도입하여 학습을 안정화 시킴.
  - $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$, with $\tau \ll 1$

- Exploration이 힘듦

  ➢ Deterministic policy를 사용함으로써, off-policy로 학습하는 것이 가능해짐!

  - **Ornstein-Uhlenbeck 과정 (OU process)**를 이용한 exploration

    : 시간적으로 연관된 noise를 주어서 exploration을 진행

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

2. Toy example with DDPG

Pendulum-v0 (OpenAI gym)

Goal: Pendulum 이 위로 잘 서 있도록 control하는 것

State

$$s_t = [\cos(\theta), \sin(\theta), \dot{\theta}]$$

- $\theta \in [-\pi, \pi]$: 단위 축으로부터의 각도

Action

- $a_t \in [-2, 2]$
- Joint에 가하는 힘의 크기

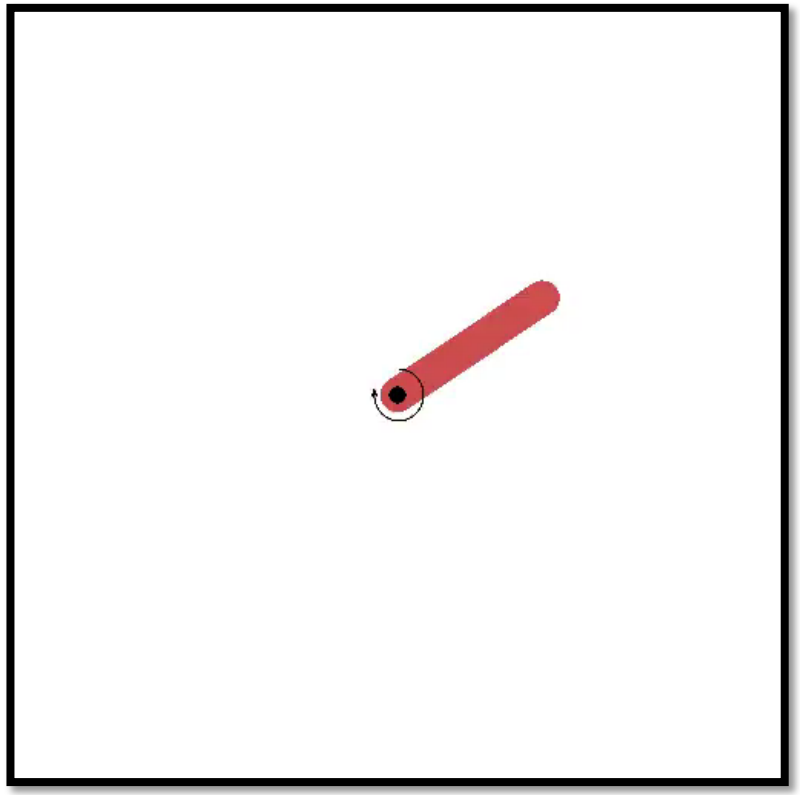
Reward

$$r_t = -(\theta^2 + 0.1 \times \dot{\theta}^2 + 0.001 \times a_t^2)$$

- $r_t \in [-16.273, 0]$
- $\theta = -\pi$ or π 일 때 minimum, $\theta = 0$ 일 때 maximum

Terminal

Terminal 조건은 없으며, time step>200 일 때 종료



https://gym.openai.com/envs/Pendulum-v0/

Jupyter Notebook 실습코드

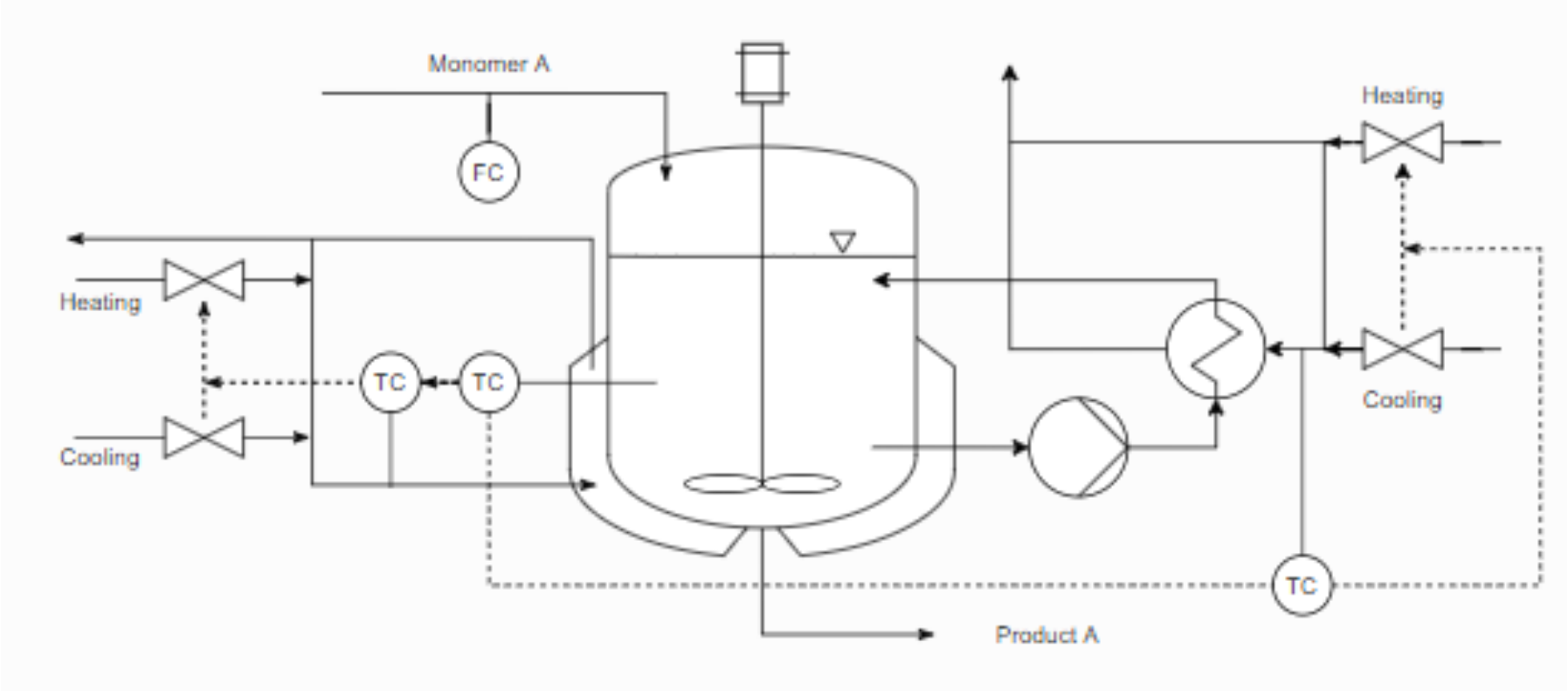
DDPG_tutorial.ipynb

1. Gym-Pendulum 환경 불러오기
2. Agent 만들 준비하기
   - Actor Network
   - Critic Network
   - Target Network (Actor, Critic)
   - Replay Memory
   - OU process
3. Agent 만들기
4. Agent 학습 및 테스트

3. Project Description

Industrial polymerization reactor

from *do-mpc :* https://www.do-mpc.com/en/latest/example_gallery/industrial_poly.html



- 고분자 반응로 모델: monomer 와 water을 이용해 product를 생성

Project Description

- Objective:

$$maximize_{a_1,\ldots,a_T} \sum_{t=1}^{T-1} R(s_t, a_t, s_{t+1})$$
$$s.t. \quad s_{t+1} = f(s_t, a_t)$$

- State variable s_t:

State	Init. cond.	Min.	Max.	Unit
m_{W}	10,000	0	inf.	kg
m_{A}	853	0	inf.	kg
m_{P}	26.5	0	inf.	kg
T_{R}	90.0	$T_{\mathrm{set}} - 2.0$	$T_{\mathrm{set}} + 2.0$	°C
T_{S}	90.0	0	100	°C
T_{M}	90.0	0	100	°C
T_{EK}	35.0	0	100	°C
T_{AWT}	35.0	0	100	°C
T_{adiab}	104.897	0	109	°C
$m_{\mathrm{F}}^{\mathrm{acc}}$	0	0	30,000	kg

Objective는 m_p를 가장 빨리 높이는 것 ~ 20680 까지!

Safety variable

Project Description

- Control variable a_t:

Control	Min.	Max.	Unit
$\dot{m}_{\mathrm{F}}$	0	30,000	$\mathrm{kg\,h^{=-1}}$
$T_{\mathrm{M}}^{\mathrm{IN}}$	60	100	°C
$T_{\mathrm{AWT}}^{\mathrm{IN}}$	60	100	°C

Action variable의 boundary를 잘 체크할 것!

Project Description

- Reward
  - Stage reward
    - $r(s_t, a_t, s_{t+1}) = s_{t+1}(m_p) - s_t(m_p)$

    : m_p 를 maximize 하는 것이 objective 이므로, 한 step 간의 m_p 의 gap 을 reward로 줌

  - Terminal reward

$s_T = Terminal$ if

Condition	Terminal reward	
$s_T(m_p) > 20680\ (kg)$ or $T_{adiab} > 109\ (℃)$	$r(s_{T-1}, a_{T-1}, s_T) = 100$	Well done ☺
	$r(s_{T-1}, a_{T-1}, s_T) = -100$	Bad done ☹

Jupyter Notebook 실습코드

main.ipynb

1. Gym-Poly reactor 환경 불러오기
2. State / action variable plot

Task

1. 주어진 환경에서 DDPG agent로 학습한 결과

2. MPC control / model-based 강화학습과 비교