Reinforce Learning

Deep Deterministic Policy Gradient (DDPG)

소프트웨어 꼰대 강의

노기섭 교수

(kafa46@hongik.ac.kr)

Contents

- **Goal:**

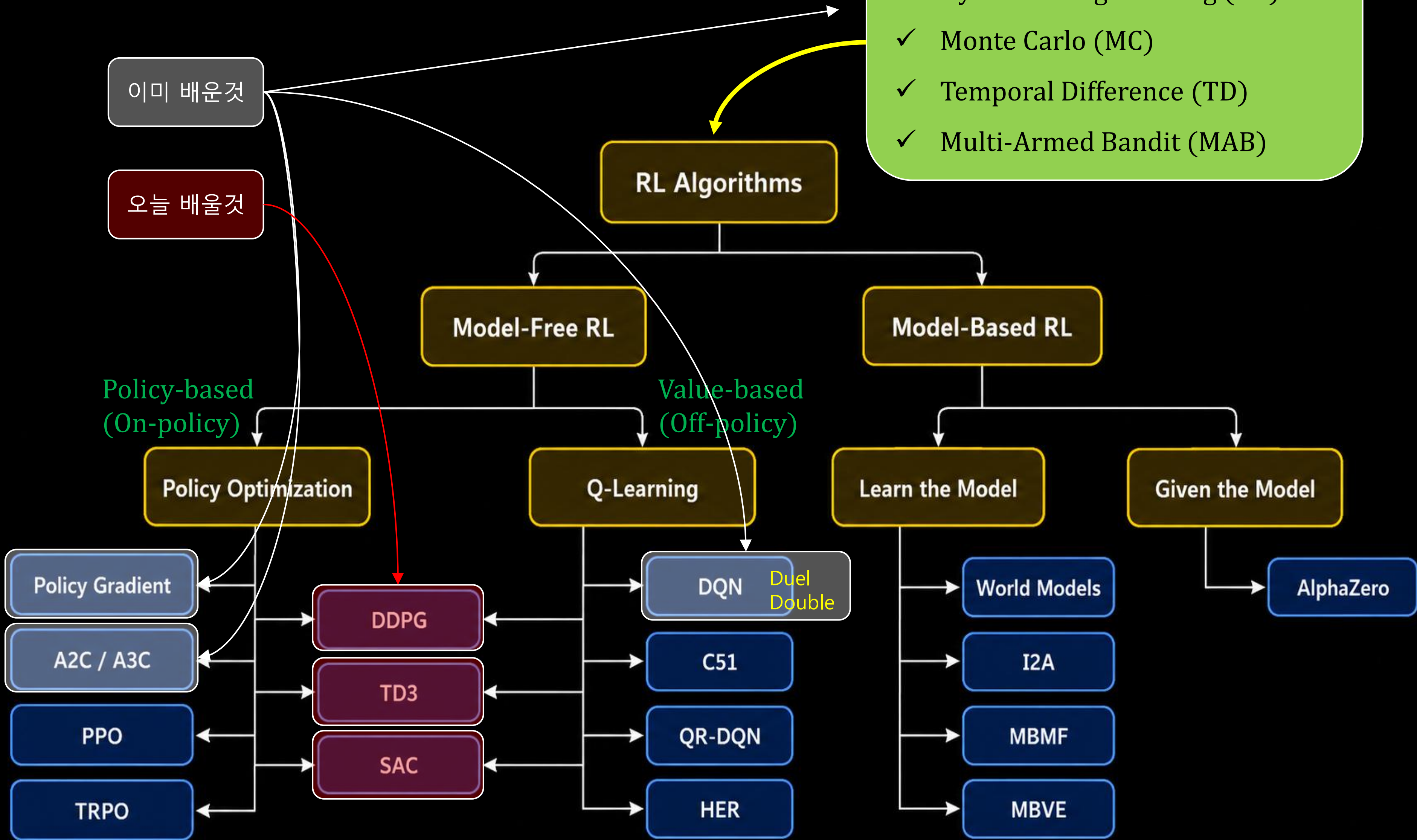
  Understand how reinforcement learning handles continuous action spaces and learn the core ideas behind the Deep Deterministic Policy Gradient (DDPG) algorithm.

- **Recap: Actor-Critic Algorithm**

- **You will learn:**
  - Why Q-learning fails for continuous actions
  - Deterministic Policy Gradient (DPG)
  - DDPG Network Architecture
  - Training Tricks: Replay Buffer & Target Networks
  - From DDPG to Modern Continuous RL (TD3 / SAC)

Recap: Actor-Critic Algorithm

A Taxonomy of RL Algorithms

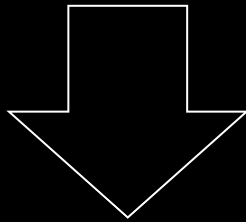


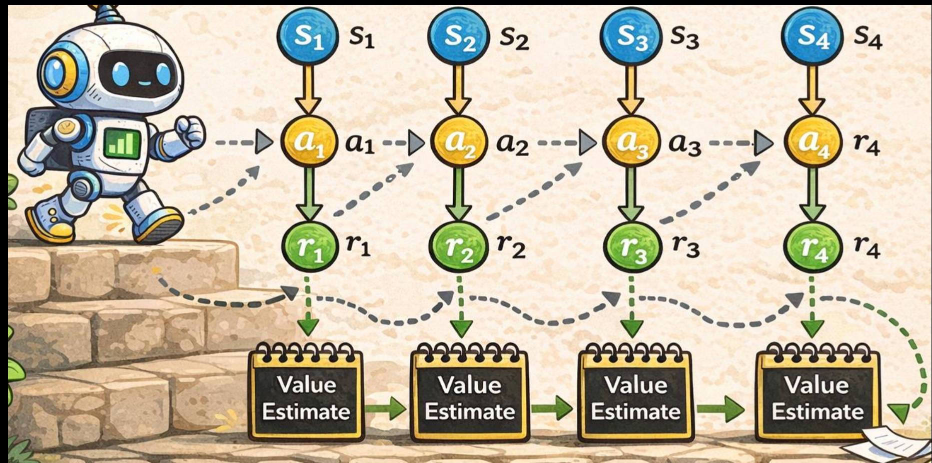
Recap: Actor-Critic Algorithm

Motivation for Actor-Critic

REINFORCE updates policy using **Monte Carlo return**

- ✓ Requires full trajectory before update
- ✓ High variance
- ✓ Slow learning





Actor-Critic solves this problem by learning step-by-step using bootstrapping.

Actor–Critic Architecture

Critic provides **learning signals at every step**.

Key Idea: Use two neural networks

Actor

- ✓ Policy network
- ✓ Selects action

$$a \sim \pi(a \mid s)$$

Critic

- ✓ Value network
- ✓ Evaluates state or action quality

$$V(s) \text{ or } Q(s,a)$$

Actor update:

$$\nabla_\theta J(\theta) = \mathbb{E}[\nabla_\theta \log \pi(a|s) A(s,a)]$$

$\pi(a|s)$: 현재 상태 s에서 행동 a를 선택할 확률

How much better an action is than average

Critic update: using TD learning

$$\delta_t = r + \gamma V(s_{t+1}) - V(s_t) \quad \text{TD error}$$

Advantage: $A(s,a) = Q(s,a) - V(s)$

- ✓ Lower variance than REINFORCE
- ✓ Step-by-step learning via TD
- ✓ Foundation of many modern RL algorithms

Critic NN update:

$$L(\theta_v) = \frac{1}{2}\delta_t^2$$

$$\theta_v \leftarrow \theta_v - \alpha \nabla_{\theta_v} L(\theta_v)$$

Continuous Control Motivation

Discrete action example	Continuous action example
• Move Left • Move Right • Move Up • Move Down	• Steering angle $\in$ [-30°, 30°] • Motor torque $\in$ [-1, 1] • Joint velocity $\in \mathbb{R}$

Many real-world control problems require continuous actions

Examples:

- Robot arm manipulation
- Autonomous driving (steering angle)
- Drone flight control
- Industrial motor control

Action values can take infinitely many values (possibilities).

Deep Deterministic Policy Gradient (DDPG)

Problem with Q-learning

Q-learning requires

$$\max_{a} Q(s, a)$$

When actions are continuous

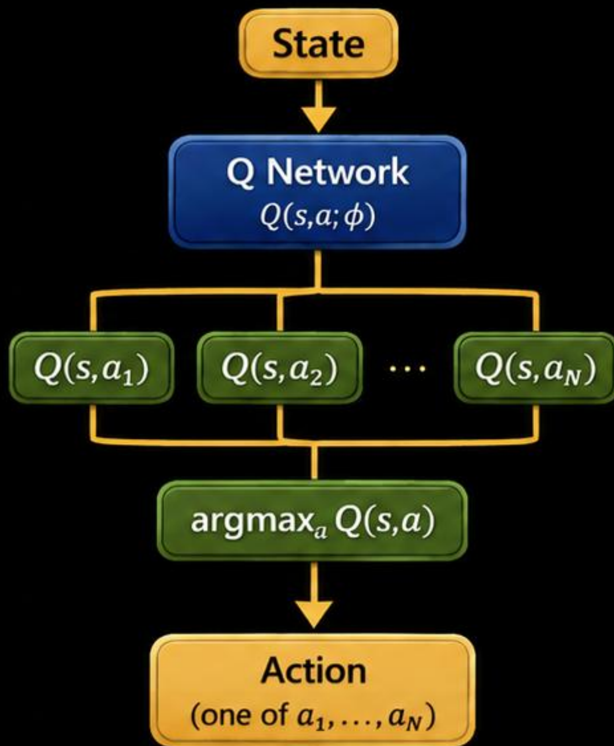
- Infinite number of actions
- Impossible to evaluate all actions

⇨ Therefore max operation becomes intractable.

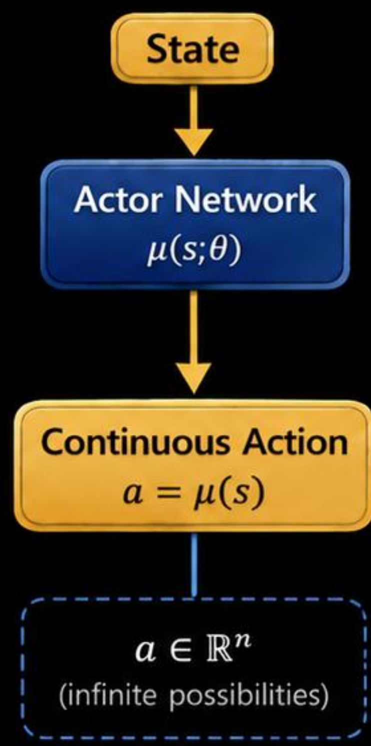
How can we learn a policy

when the action space is continuous?

Deterministic Policy Gradient (DPG): Key Idea



finite action search possible
(exhaustive evaluation feasible)



exhaustive search impossible
(infinite action space)

Advantages:

- **No need to search over actions**
  Actor outputs action directly.
- **Naturally handles continuous action spaces**
  Works in $\mathbb{R}^n$ action spaces.
- **Foundation of DDPG**
  DDPG uses a deterministic actor policy.

같은 상태라면
항상 같은 출력으로
고정하자!

Key Idea: Instead of searching for the best action, **learn a deterministic policy** that maps states to actions directly: $a = \mu(s)$

```
actor = NeuralNetwork()
action = actor(state)
```

Deterministic Policy Gradient (DPG): Formulation

Searching for the best action	New idea
Q-learning requires $\max_a Q(s,a)$ which is **intractable in continuous action spaces**.	Learn a function that **directly outputs the best action $\boldsymbol{a} = \boldsymbol{\mu}(\boldsymbol{s})$** where $\mu(s)$ is the **deterministic policy.** Policy directly maps: $state \rightarrow action$

Objective is to Maximize:

$$J(\theta) = \mathbb{E}\left[Q\left(s, \mu_\theta(s)\right)\right]$$

Policy parameter update

Policy param update:

critic gradient　　actor parameter gradient

$$\nabla_\theta J(\theta) = \mathbb{E}[\nabla_a Q(s,a) \nabla_\theta \mu_\theta(s)]_{a=\mu_\theta(s)}$$

Action a가 변할 때 $Q(s,a)$가 얼마나 변하는지에 대한 값

(Q값의 action 방향 gradient → critic이 action 방향 제시)

"action을 어디로 움직이면 Q가 증가하는가?"

"그 방향으로 action이 움직이게 하려면 파라미터를 어떻게 바꿔야 하나?"

하지만 Actor-Critic에서는 action을 actor가 생성

$$a = \mu_\theta(s)$$

Gradient를 계산할 때, 현재 **actor**가 선택한 **action** 위치에서 gradient를 계산

$$\nabla_a Q(s,a)\Big|_{a=\mu_\theta(s)}$$

"a를 $\mu_\theta(s)$로 대입해서 계산한다"

From Deterministic Policy Gradient to DDPG

Deterministic Policy Gradient

Policy directly outputs an action $a = \mu_\theta(s)$

Policy is optimized using

$$\nabla_\theta J(\theta) = \mathbb{E}[\nabla_a Q(s,a) \nabla_\theta \mu_\theta(s)]_{a=\mu_\theta(s)}$$

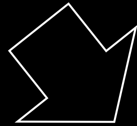
Problem: Using neural networks introduces **training instability**

- Correlated samples
- Continuously changing target values
- Unstable critic updates
- DDPG adopts stabilization techniques from **DQN**

From Deterministic Policy Gradient to DDPG

Combine deterministic policy gradient with:

- Actor-Critic architecture
- Experience Replay
- Target Networks



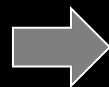
- ✓ Deterministic Policy Gradient (**DPG**)

  +

- ✓ Deep Neural Networks (**DNN**)

  +

- ✓ Stabilization Techniques (**Buffer**)



Deep Deterministic Policy Gradient (**DDPG**)

Key Components of DDPG

DDPG = Actor-Critic + DQN Stabilization for Continuous Control

Replay Buffer

- Store past transitions
- Sample mini-batches randomly (approx. i.i.d.)
- Break correlation between samples
- Improves learning stability

Target Networks

- Separate target actor and critic
- Slowly updated networks
  (soft update, see next slide for details)
  $\theta_{target} \leftarrow \tau\theta_{online} + (1 - \tau)\theta_{target}$
- Prevents unstable learning updates

Off-policy Learning

- Learn from previously collected data
- Improves sample efficiency

DDPG Training Pipeline

1. Interaction w/ Environment

- Agent observes current state s_t
- Actor NN outputs action
  $$a_t = \mu(s_t) + N_t$$
- Execute action in environment
- Receive r_t, s_{t+1}

2. Store Experience

- Transition stored in replay buffer
  $$(s_t, a_t, r_t, s_{t+1})$$
- Replay buffer enables off-policy learning

3. Critic Update

- Sample mini-batch from replay buffer
- Compute target Q value
  $$y = r + \gamma Q'\big(s', \mu'(s')\big)$$
- Update critic by minimizing
  $$L = \frac{1}{N}\sum_i \big(Q_\phi(s_i, a_i) - y_i\big)^2$$

4. Actor Update

- Policy gradient
  $$\nabla_\theta J = \mathbb{E}[\nabla_a Q(s,a) \nabla_\theta \mu(s)]_{a=\mu(s)}$$

5. Target Net Update

- Soft update: $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$

  A.K.A., Polyak Averaging

  https://www.cs.cornell.edu/courses/cs4787/2021sp/notebooks/Slides10.html#Polyak-Averaging:-Intuition

DDPG Architecture

DDPG uses an actor–critic architecture with experience replay and target networks to learn stable policies in continuous action spaces.

Online Networks (trained every step)

Target Networks (updated slowly)

Agent

Actor (online) $\mu_\theta(s)$

Critic (online) $Q_\phi(s, a)$

Environment

state s_t

action a_t (+ exploration noise)

reward r_t, next state s_{t+1}

Replay Buffer $\mathcal{D}$

store (s_t, a_t, r_t, s_{t+1})

sample mini-batch

state s

Actor (online) $\mu_\theta(s)$

Critic (online) $Q_\phi(s, a)$

Soft Update (Polyak averaging)

$\theta' \leftarrow \tau\,\theta + (1-\tau)\theta'$

Soft Update (Polyak averaging)

$\phi' \leftarrow \tau\,\phi + (1-\tau)\phi'$

Actor (target) $\mu_{\theta'}(s)$

Critic (target) $Q_{\phi'}(s, a)$

$a' = \mu_{\theta'}(s')$

$Q_{\phi'}(s', a')$

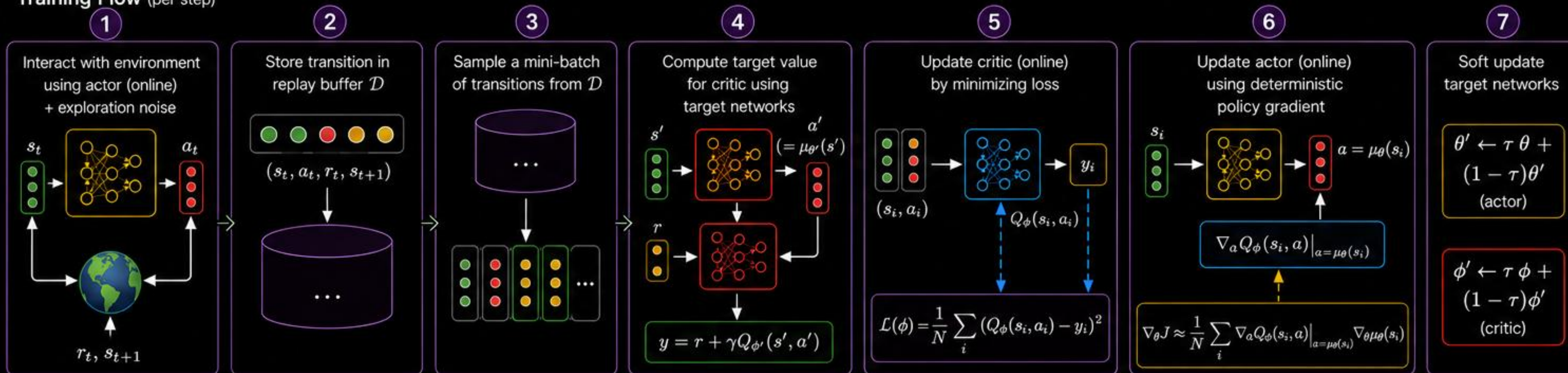
state s'

action a'

Target value for critic update

$y = r + \gamma Q_{\phi'}(s', a')$

where $a' = \mu_{\theta'}(s')$



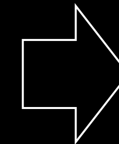
Exploration in DDPG

Problem:

DDPG uses a **deterministic policy**

$$a = \mu(s)$$

The same state always produces the same action.



No Exploration!

Solution:

Add **exploration noise** to the action.

$$a_t = \mu(s_t) + N_t$$

N_t: a noise process

Ornstein–Uhlenbeck (OU) Noise (originally used for exploration)

$$dX_t = \theta(\mu - X_t)dt + \delta dW_t$$

- Temporally correlated noise
- Smooth changes in actions

Modern Practice (Gaussian Noise)

$$a_t = \mu(s_t) + \epsilon$$

- Simple as well as works well

Limitations of DDPG

Core Problem of DDPG

Although DDPG works well, it suffers from several instability issues.

---

1. Overestimation Bias

- Critic estimates $Q(s,a)$ may become overestimated.
- Reason: $\max Q(s,a)$
  → greedy policy tends to select overestimated values
- Result: policy learns incorrect actions

---

2. Function Approximation Error (NN errors)

- Both Actor and Critic use neural networks
- Small errors in Q-value estimation can propagate.
- Actor may exploit incorrect Q estimates.

---

3. Highly Sensitive Training

Learning can become unstable due to:

- actor updates too frequently
- critic estimation noise
- correlated updates

Twin Delayed Deep Deterministic Policy Gradient (TD3)

Twin Delayed Deep Deterministic Policy Gradient (TD3)

Motivations:
- ✓ DDPG often suffers from **overestimation bias** and **training instability**.
- ✓ Errors in the critic can cause the actor to learn **incorrect actions**.
- ✓ TD3 was proposed to address these issues.

Twin Critics:
- ✓ Two Q-networks are used $Q_1(s,a),\ Q_2(s,a)$
- ✓ Target value uses the **minimum** $y = r + \gamma \min\big(Q_1'(s',a'), Q_2'(s',a')\big)$
- ✓ This reduces **overestimation bias**.

Delayed Policy Update: Actor is updated less frequently than critic

```
Example) critic update: every step
         actor update: every 2 steps
```

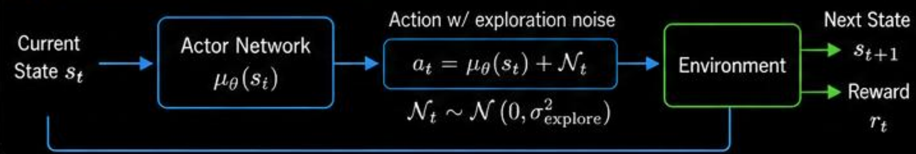
Actor is updated by using Q_1 : $\nabla_\theta J = E[\nabla_a Q_1(s,a) \nabla_\theta \mu(s)]_{a=\mu(s)}$

Target Policy Smoothing: Noise is added to target actions

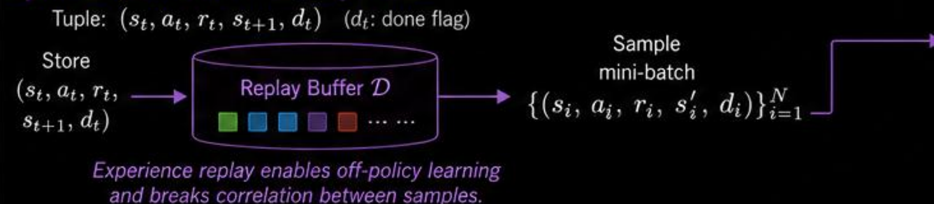
$$a' = \text{clip}(\mu'(s') + \epsilon, a_{min}, a_{max})$$

TD3 Architecture

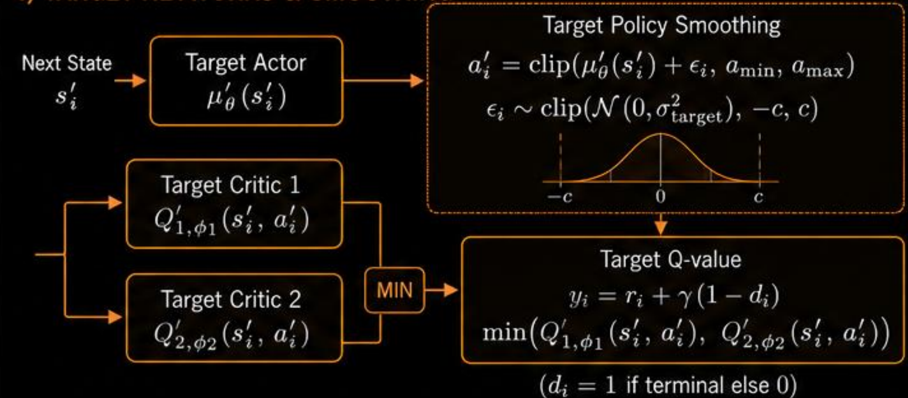
1) ENVIRONMENT INTERACTION



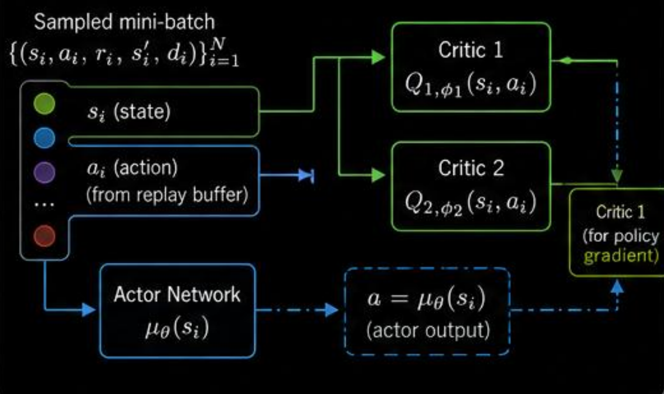
2) REPLAY BUFFER: Store Experience



4) TARGET NETWORKS & SMOOTHING



3) SAMPLE MINIBATCH & TWIN CRITICS (Online)



LOSS FUNCTIONS

Critic Loss (update every step)

$$L_{\text{crit}}(\phi_1, \phi_2) = \frac{1}{N}\sum_{i=1}^{N}\Big[(Q_{1,\phi1}(s_i, a_i) - y_i)^2 + (Q_{2,\phi2}(s_i, a_i) - y_i)^2\Big]$$

Actor Loss (update every d steps)

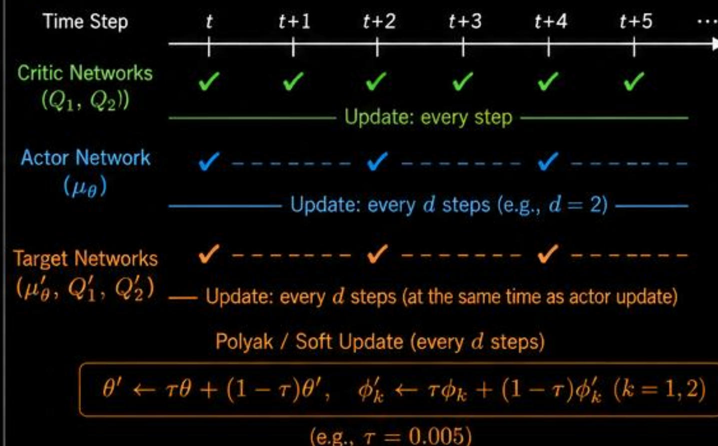
$$L_{\text{act}}(\theta) = -\frac{1}{N}\sum_{i=1}^{N} Q_{1,\phi1}(s_i, \mu_\theta(s_i))$$

(maximize $Q_1(s, \mu(s))$)

Policy Gradient

$$\nabla_\theta J(\theta) = \mathrm{E}\Big[\nabla_a Q_{1,\phi1}(s, a)\nabla_\theta \mu_\theta(s)\Big]_{a=\mu_\theta(s)}$$

5) DELAYED UPDATES TIMELINE



***TD3 Key Ideas** (How TD3 Solves DDPG Instability)*

1. Twin Critics + Min Target
   → reduce overestimation bias
2. Target Policy Smoothing
   → prevent exploiting Q errors (increase target robustness)
3. Delayed Policy Updates
   → improve training stability (critic becomes more reliable before actor moves)

DDPG vs TD3

DDPG vs TD3

REPLAY BUFFER $\mathcal{D}$
(s, a, r, s', d) sampled as minibatch
(d: done flag)

Critic data flow (forward)
Policy gradient flow (actor update only)
Target computation flow
Replay buffer sampling flow

DDPG

State s_i

Actor $\mu_\theta(s_i)$

Action (for critic) a_i (from buffer)

(s_i, a_i)

Critic $Q_\phi(s_i, a_i)$

Policy gradient (through Q)

Next state s'_i

Target Actor $\mu_{\theta'}(s'_i)$

Target action $a'_i = \mu_{\theta'}(s'_i)$

Target Critic $Q_{\phi'}(s'_i, a'_i)$

Target value:
$$y_i = r_i + \gamma(1 - d_i)Q_{\phi'}(s'_i, a'_i)$$

Critic Loss (update every step)
$$\mathcal{L}_{crit}(\phi) = \frac{1}{N}\sum_{i=1}^{N}\left[Q_\phi(s_i, a_i) - y_i\right]^2$$

Actor Loss (update every step)
$$\mathcal{L}_{act}(\theta) = -\frac{1}{N}\sum_{i=1}^{N}Q_\phi(s_i, \mu_\theta(s_i))$$
(maximize Q for policy update)

TD3

State s_i

Actor $\mu_\theta(s_i)$

Action (for critics) a_i (from buffer)

(s_i, a_i)

Critic 1 $Q_{1,\phi_1}(s_i, a_i)$

Critic 2 $Q_{2,\phi_2}(s_i, a_i)$

Policy gradient (only through Q_1)

Next state s'_i

Target Actor $\mu_{\theta'}(s'_i)$

Noisy target action $a'_i = \mathrm{clip}(\mu_{\theta'}(s'_i) + \epsilon_i, a_{min}, a_{max})$

Target Critic 1 $Q'_{1,\phi'_1}(s'_i, a'_i)$

Target Critic 2 $Q'_{2,\phi'_2}(s'_i, a'_i)$

MIN

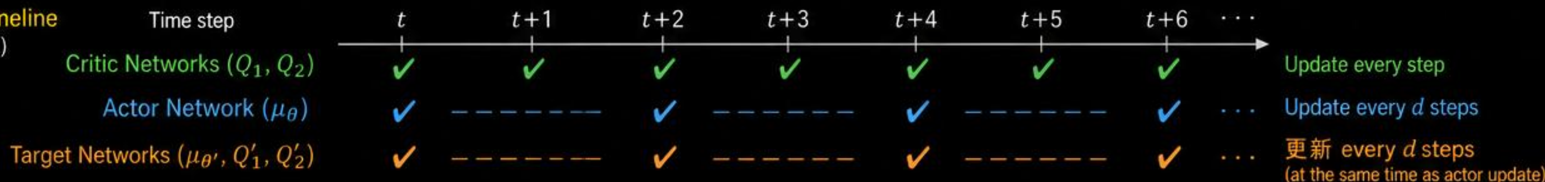
Target Policy Smoothing
Add clipped noise to target action
$$\epsilon_i \sim \mathrm{clip}(\mathcal{N}(0, \sigma^2), -c, c)$$
$$a'_i = \mathrm{clip}(\mu_{\theta'}(s'_i) + \epsilon_i, a_{min}, a_{max})$$

Target value:
$$y_i = r_i + \gamma(1 - d_i)\min(Q'_{1,\phi'_1}(s'_i, a'_i), Q'_{2,\phi'_2}(s'_i, a'_i))$$

Critic Loss (update every step)
$$\mathcal{L}_{crit}(\phi_1, \phi_2) = \frac{1}{N}\sum_{i=1}^{N}\left[(Q_{1,\phi_1}(s_i, a_i) - y_i)^2 + (Q_{2,\phi_2}(s_i, a_i) - y_i)^2\right]$$

Actor Loss (update every d steps)
$$\mathcal{L}_{act}(\theta) = -\frac{1}{N}\sum_{i=1}^{N}Q_{1,\phi_1}(s_i, \mu_\theta(s_i))$$
(maximize Q_1 for policy update)

Policy Gradient (only every d steps)
$$\nabla_\theta J(\theta) = \frac{1}{N}\sum_{i=1}^{N}\nabla_a Q_{1,\phi_1}(s_i, a)\nabla_\theta\mu_\theta(s_i)\Big|_{a=\mu_\theta(s_i)}$$



Key Takeaways: How TD3 Solves DDPG Instability

1. Twin Critics
Use two critics and take the minimum of target values to reduce overestimation bias.

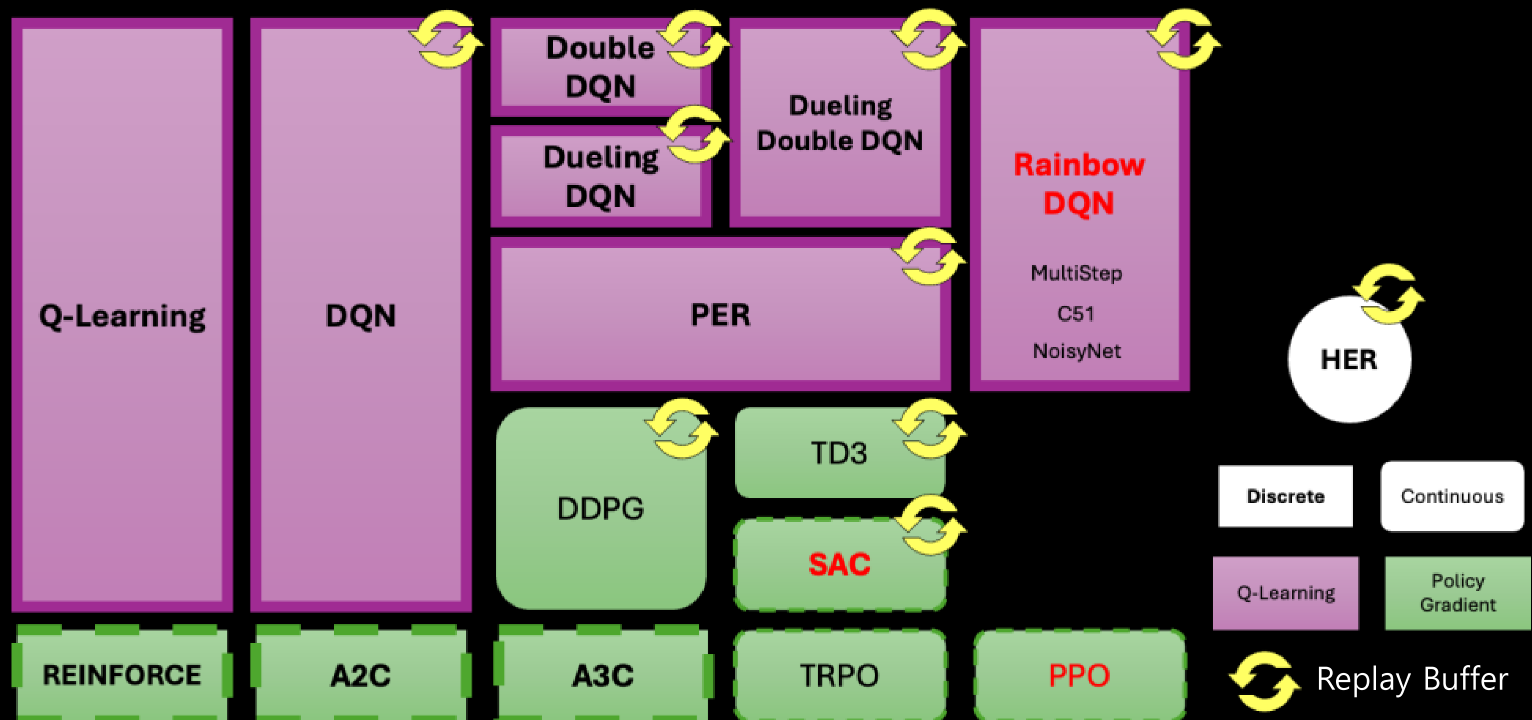
2. Target Policy Smoothing
Add clipped noise to target actions to prevent exploiting Q estimation errors and make targets smoother.

3. Delayed Policy Updates
Update actor (and target networks) less frequently than critics for more stable learning.

Soft Actor-Critic (SAC)

Actor-Critic Evolution

Algorithm	Characteristics
DDPG	deterministic policy
TD3	Stabilization of DDPG
SAC	stochastic policy + entropy



https://blog.naver.com/yhsyhs0602/223759706615

Limitations of Deterministic Policy Methods

Deterministic Policy Algorithms:
DDPG and TD3 use deterministic policies.

$$a = \mu(s)$$

The policy always outputs a single action for each state.

Exploration via Noise:
Exploration is typically achieved by adding noise:

$$a = \mu(s) + \epsilon, \text{where } \epsilon \sim \mathcal{N}(0, \sigma)$$

Limitations: Noise-based exploration has several weaknesses

- ✓ Exploration becomes inefficient in high-dimensional action spaces
- ✓ Policies may converge prematurely to suboptimal behaviors
- ✓ Exploration is not guided by the learning objective

Soft Actor-Critic (SAC)

Maximum Entropy Reinforcement Learning

$$J = \mathbb{E}[R + \alpha H(\pi)]$$

Maximum Entropy Reinforcement Learning

Standard Reinforcement Learning Objective

Traditional reinforcement learning maximizes the expected return.

$$J(\pi) = \mathbb{E}\left[\sum_t r\,(s_t, a_t)\right]$$

The objective focuses **only on maximizing reward**.

reward만 maximize하지 말고
entropy도 maximize하자
((randomness도 유지)

Maximum Entropy Objective

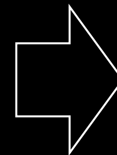
Maximum entropy reinforcement learning adds an entropy term.

$$J(\pi) = \mathbb{E}\left[\sum_t \Big(r(s_t, a_t) + \alpha H\big(\pi(\cdot \mid s_t)\big)\Big)\right]$$

, where $H\big(\pi(\cdot \mid s)\big) = -\mathbb{E}_{a\sim\pi}[\pi(a|s)\log \pi\,(a \mid s)]$

Intuition: encourages **diverse and exploratory behaviors**.

- Maximize reward
- Maintain high policy entropy



SAC optimizes the **maximum entropy objective**, leading to **stable learning and improved exploration**.

Soft Value Functions in SAC

Soft Q-function:

measures the expected return including the entropy bonus.

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^{t} \left(r(s_t, a_t) + \alpha H\left(\pi(\cdot \mid s_t)\right)\right)\right]$$

This value function considers **both reward and entropy**.

Soft State Value Function:

The soft state value function is defined as

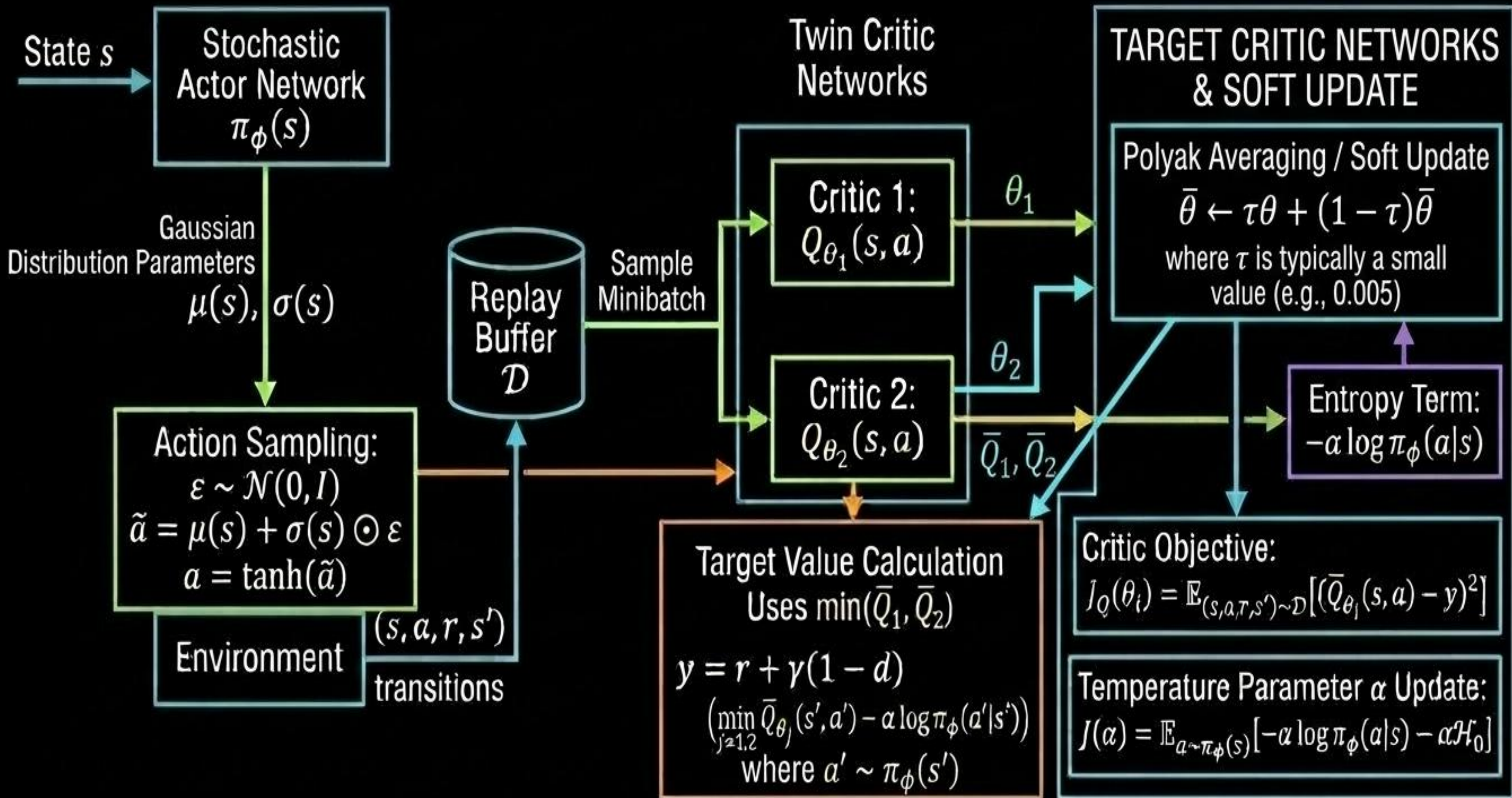
$$V^{\pi}(s) = \mathbb{E}_{a\sim\pi}[Q^{\pi}(s,a) - \alpha \log \pi\,(a \mid s)]$$

The entropy term encourages **stochastic policies**.

$V^{\pi}(s)$ incorporate entropy regularization.

- better exploration
- more stable policy learning

SAC Architecture



Evolution of Continuous Control RL

DDPG

- Deterministic Policy
- Single Critic Network
- Off-Policy Learning

$Q(s,a)$

Actor → Critic

$Q(s,a)$

$a = \mu(s)$

TD3

- Twin Critic Networks
- Delayed Policy Update
- Reduced Overestimation

Critic | Critic

min Q-value

Actor

Del policy

SAC

- Stochastic Policy
- Entropy Maximization
- MaxEnt RL
- Twin Critic Networks

Stochastic policy

$\mu, \sigma)$

Entropy Term $H(\pi)$

Critic | Critic

e.g. $\alpha * H(\pi(\cdot|s))$

Design Principles of Modern RL

Covered these technologies so far!

Function Approximation

Neural networks approximate value and policy functions

Off-policy Learning

Replay buffers improve sample efficiency

Variance Reduction

Advantage estimation reduces gradient variance

Entropy Regularization

Encourages exploration and prevents premature convergence

We'll explore

Clipped / Constrained Updates

Improve training stability (TRPO, PPO)

수고하셨습니다 ..^^..