2021-2 한양대학교 HAI

강화학습 부트캠프

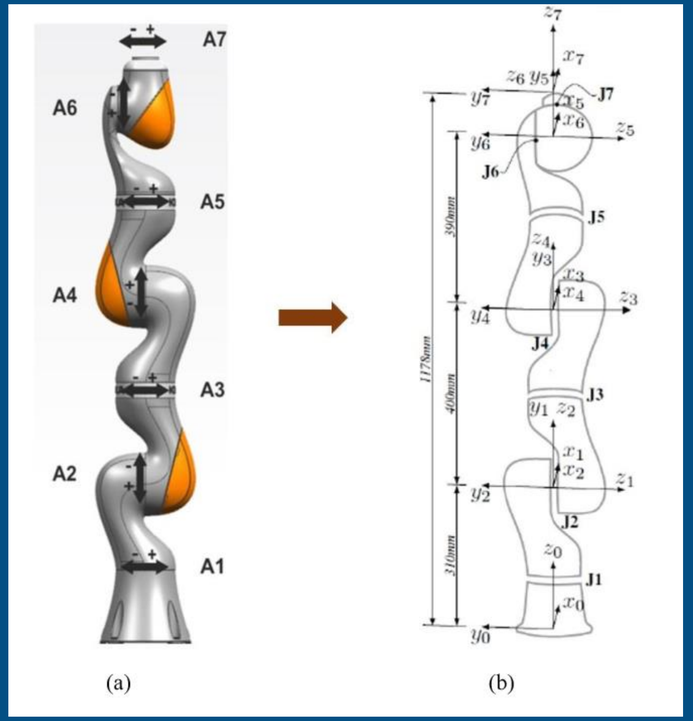
Chris Ohk
utilForever@gmail.com

오늘 배울 내용

- Continuous Action Space
  - Advantage Actor-Critic (A2C)
  - Deep Deterministic Policy Gradient (DDPG)
  - Distributed Distributional DDPG (D4PG)

Continuous Action Space

- DQN으로 풀 수 있는 문제
  - 관찰 공간의 차원이 높다.
  - 행동 공간이 이산(유한) 공간이고 차원이 낮다.
- DQN의 한계
  - 7개의 관절로 구성된 로봇 팔이 있다고 하자.
    로봇 팔은 다양한 각도로 회전할 수 있다.
  - 따라서 이산 공간과 달리 할 수 있는 행동이 무한에 가깝다.
    예를 들어, “13.5도로 회전한다” 또는 “13.512도로 회전한다”는 서로 다른 행동이다.
    → 위와 같은 환경에서는 DQN으로 풀기 힘들다.



The Action Space

- 이산(유한) 공간
  - 값을 하나씩 나열한다. (예 : $\{up, down, left, right\}$)
- 연속(무한) 공간
  - 값을 범위로 표현한다. (예 : $[0 \ldots 1]$)
  - OpenAI Gym에서는 Box를 통해 연속 공간을 나타낸다.
  - 예 : `Box(low=0, high=255, shape=(210, 160, 3))`
    - 3D 텐서에 100,800개의 값들이 있다.
    - 각 값은 0~255의 범위를 갖는다.

Environments

- 연속 행동 공간은 리얼 월드 문제 → 주로 물리 시뮬레이션을 사용한다.
  - 물리 시뮬레이션을 하기 위한 다양한 오픈 소스 라이브러리/프레임워크가 존재한다.
  - 가장 유명한 패키지로 “MuJoCo”(https://www.mujoco.org)가 있다.
    원래는 유료라서 라이선스가 필요했지만, 최근 DeepMind가 인수해 무료가 되었다.
    (소스 코드도 조만간 GitHub에 공개할 예정이다.)
  - 다른 패키지로 “PyBullet”(https://github.com/bulletphysics/bullet3)이 있다.
    예제 코드에서는 이 패키지를 사용해 환경을 구성할 예정이다.

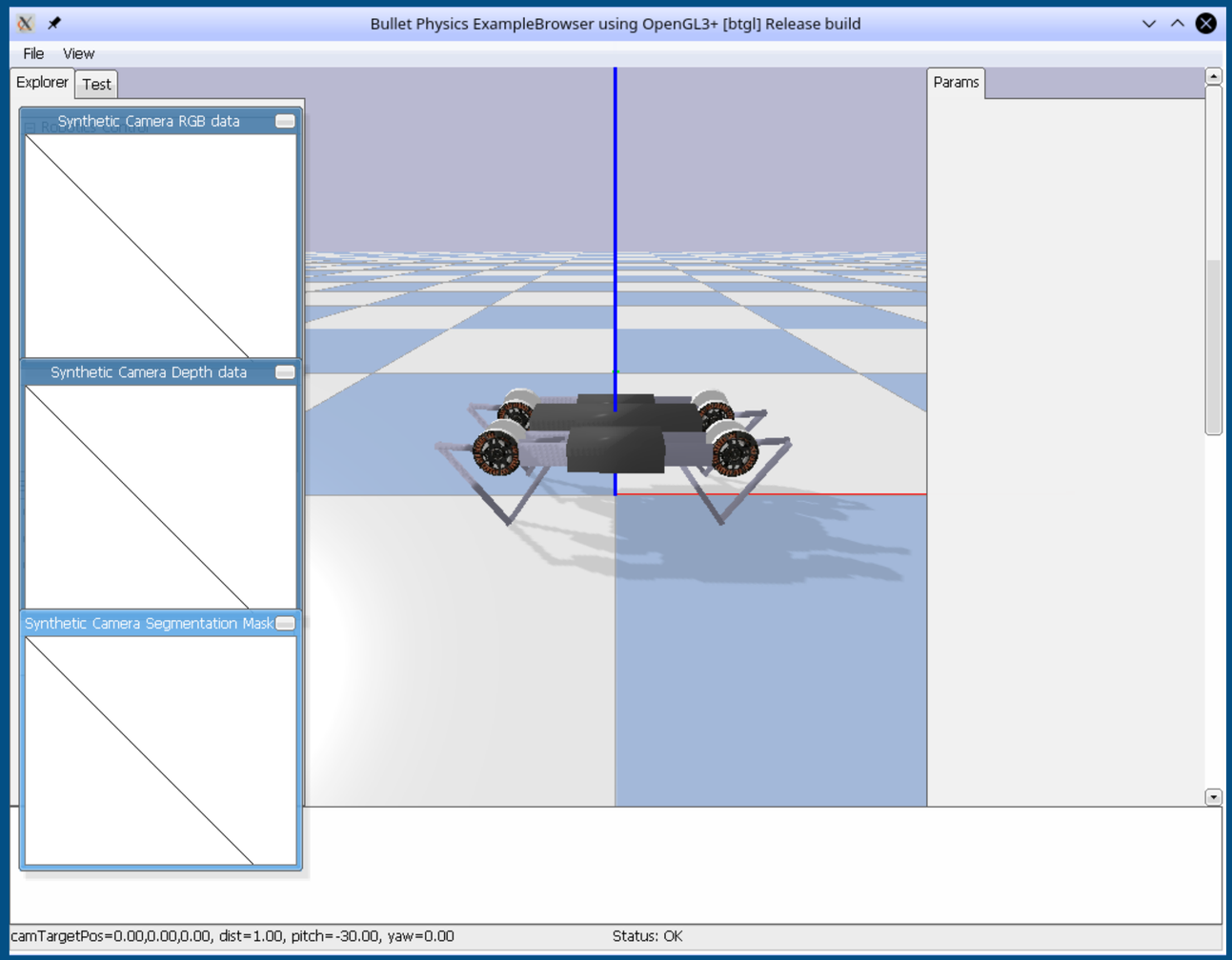
Environments

- MinitaurBulletEnv-v0

```python
import gym
import pybullet_envs

ENV_ID = "MinitaurBulletEnv-v0"
RENDER = True

if __name__ == "__main__":
    spec = gym.envs.registry.spec(ENV_ID)
    spec._kwargs['render'] = RENDER
    env = gym.make(ENV_ID)

    print("Observation space:", env.observation_space)
    print("Action space:", env.action_space)
    print(env)
    print(env.reset())
    input("Press any key to exit\n")
    env.close()
```

Environments

- `MinitaurBulletEnv-v0`



Advantage Actor-Critic (A2C)

- Asynchronous Methods for Deep Reinforcement Learning (Mnih, 2016)
- Dueling DQN에서 다뤘던 Advantage 개념을 Actor-Critic에 사용한다.

$$Q(s_t, a_t) = V(s_t) + A(a_t, s_t)$$

- 기존 Policy Gradient 식

$$\nabla_\theta J(\theta) \approx E_{\pi_\theta}[Q(s_t, a_t)\nabla_\theta \log \pi(a_t|s_t;\theta)]$$

- A2C의 Policy Gradient 식

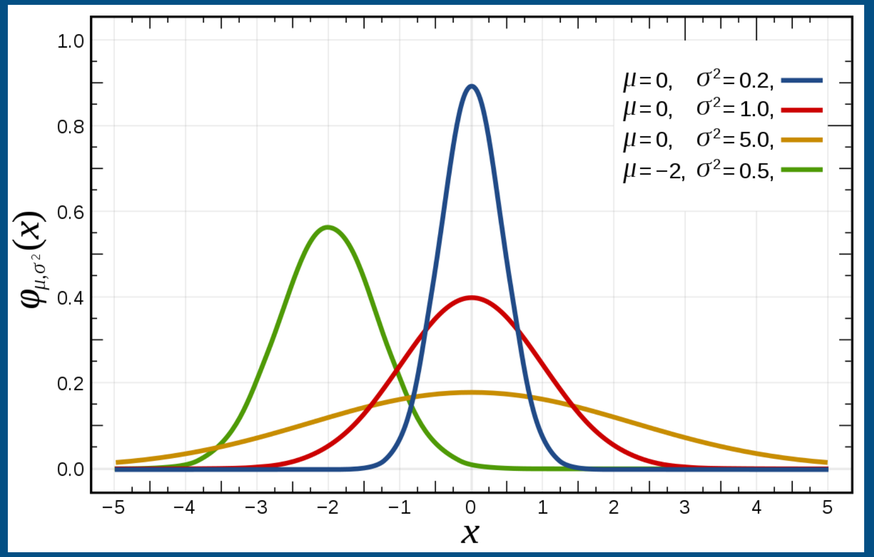
$$\nabla_\theta J(\theta) \approx E_{\pi_\theta}\big[\big(Q(s_t, a_t) - V(s_t)\big)\nabla_\theta \log \pi(a_t|s_t;\theta)\big]$$

$$\approx E_{\pi_\theta}[A(s_t, a_t)\nabla_\theta \log \pi(a_t|s_t;\theta)]$$

Advantage Actor-Critic (A2C)

- 우리가 배웠던 A2C는 행동 공간이 이산(유한) 공간일 경우에 해당한다.
- 만약 행동 공간이 연속(무한) 공간이라면, 분포를 사용해야 한다.
- 가우시안(Gaussian) 분포를 가장 많이 사용한다.

$$f(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$



Advantage Actor-Critic (A2C)

- 가우시안 분포를 바로 사용할 수도 있지만, 수치적 안정성을 높이기 위해 식을 약간 바꾼다.
- 최종 결과

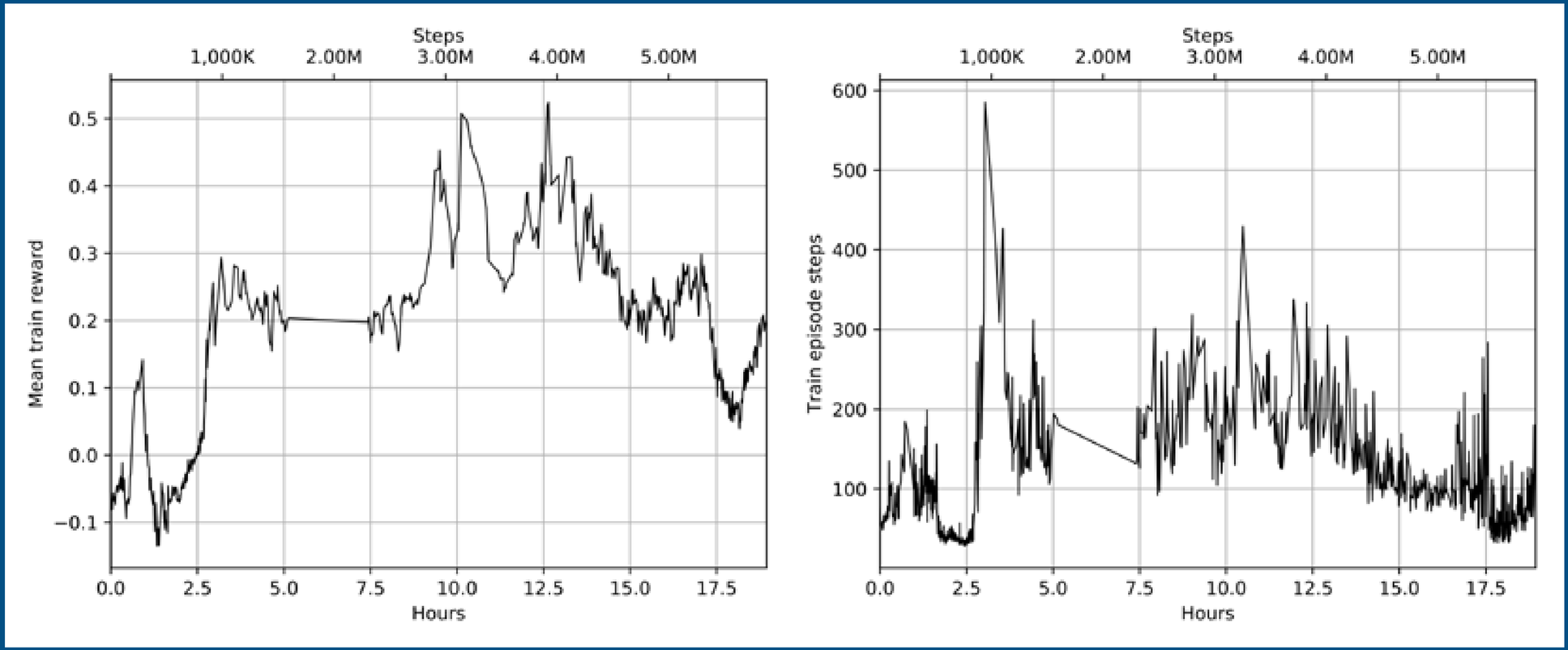
$$\log \pi(a_t|s_t;\theta) = -\frac{(x-\mu)^2}{2\sigma^2} - \log\sqrt{2\pi\sigma^2}$$

- 엔트로피

$$\sqrt{2\pi e\sigma^2}$$

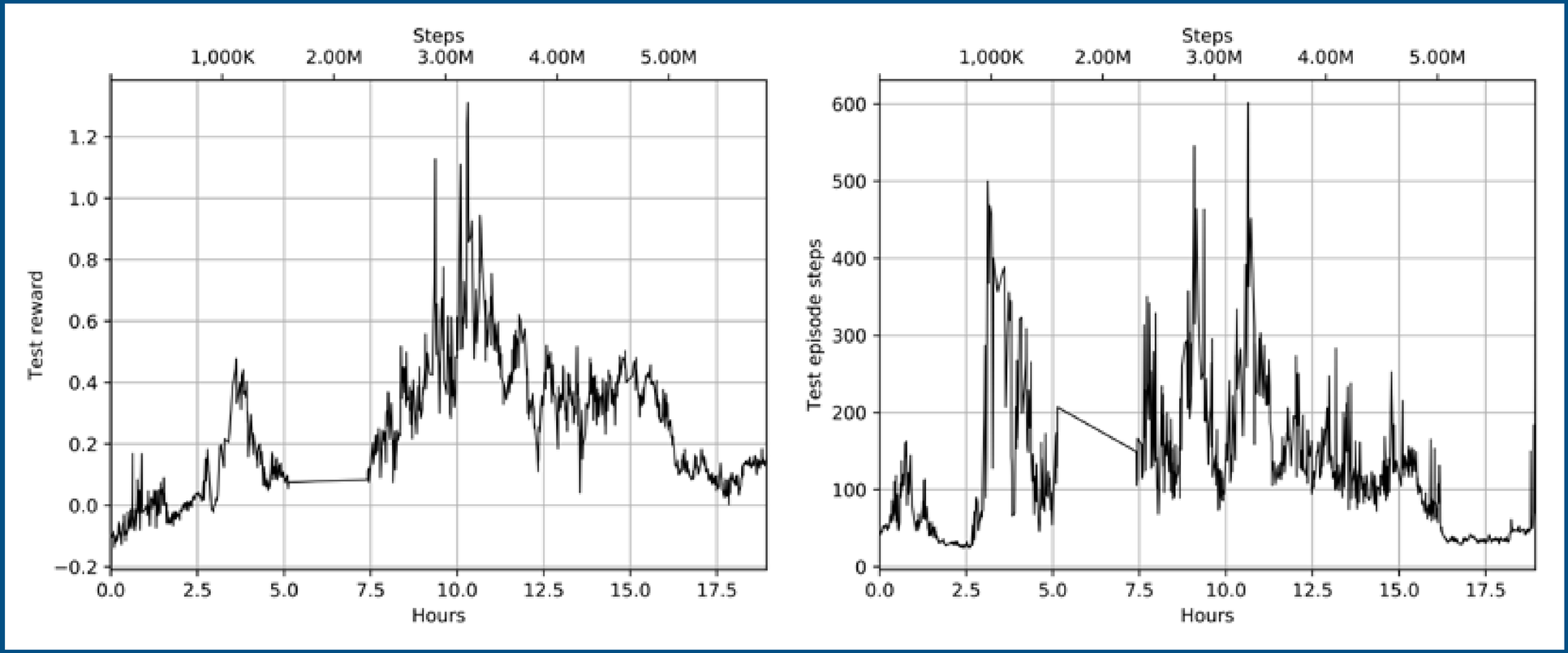
Advantage Actor-Critic (A2C)

- The Reward and Steps for Training Episodes



Advantage Actor-Critic (A2C)

- The Reward and Steps for Test Episodes



Deep Deterministic PG (DDPG)

- Deterministic Policy Gradient Algorithms (Silver, 2014)
- Continuous Control with Deep Reinforcement Learning (Lillicrap, 2015)
- DDPG = DPG(Deterministic PG) + Actor-Critic with Deep Neural Network
- DPG
  - Critic Network (Value) : DQN에 있는 벨만 방정식을 사용

$$y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1})|\theta^Q).$$

  - Actor Network (Policy)

$$\begin{aligned}\nabla_{\theta^\mu} J &\approx \mathbb{E}_{s_t \sim \rho^\beta}\left[\nabla_{\theta^\mu} Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t|\theta^\mu)}\right] \\ &= \mathbb{E}_{s_t \sim \rho^\beta}\left[\nabla_a Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta_\mu} \mu(s|\theta^\mu)|_{s=s_t}\right]\end{aligned}$$

Deep Deterministic PG (DDPG)

- DDPG의 4가지 특징
  - 리플레이 버퍼(Replay Buffer) 사용
  - “Soft” Target Update 사용
  - 배치 정규화(Batch Normalization) 사용
  - 탐험을 위해 행동에 노이즈 추가

Deep Deterministic PG (DDPG)

- 리플레이 버퍼(Replay Buffer) 사용
  - 큐러닝처럼 선형이 아닌 함수 예측법은 수렴이 보장되지 않는다.
  - 그러나 차원이 큰 상태 공간을 학습하고 일반화하려면 선형이 아닌 함수 예측법이 반드시 필요하다.
  - NFQCA (2011)는 DPG와 같이 Actor와 Critic을 갱신하지만, 신경망을 사용한다는 점이 다르다.
  - NFQCA는 수렴의 안정성을 위해 배치 학습(Batch Learning)을 사용한다.
  - 배치 학습이란 모든 데이터 셋을 한 번에 학습하기 때문에 크기가 큰 신경망에서 다루기 힘들다.
    (왜냐하면 학습이 오래 걸리기 때문이다.)
  - 이를 해결하기 위해 미니 배치를 사용한 NFQCA가 있지만, 갱신마다 정책을 초기화하지 않는 단점이 있다.
  - DDPG는 DQN에서 사용했던 리플레이 버퍼를 사용해 온라인 배치 갱신을 가능하게 한다.

Deep Deterministic PG (DDPG)

- “Soft” 타겟 업데이트 사용
  - DQN은 많은 환경에서 학습이 불안정하다. (즉, 수렴이 잘 안된다.)
  - 따라서 일정 주기마다 Q-Network의 가중치를 타겟 신경망으로 직접 복사해서 사용한다.
  - DDPG에서는 지수 이동 평균(Exponential Moving Average) 식으로 대체한다.
  - 타겟 신경망의 갱신 속도는 매개 변수 τ로 조절한다. ($\tau \ll 1$)

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

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

Deep Deterministic PG (DDPG)

- 배치 정규화(Batch Normalization) 사용
  - 표현 범위가 서로 다른 특징들을 상태로 사용할 때에 신경망은 일반화하는 데 어려움을 겪는다.
  - 원래 이 문제를 해결하려면 직접 표현 범위를 조정해줘야 했었다.
  - DDPG에서는 각 층의 입력을 Unit Gaussian이 되도록 강제하는 배치 정규화를 사용해 해결한다.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Deep Deterministic PG (DDPG)

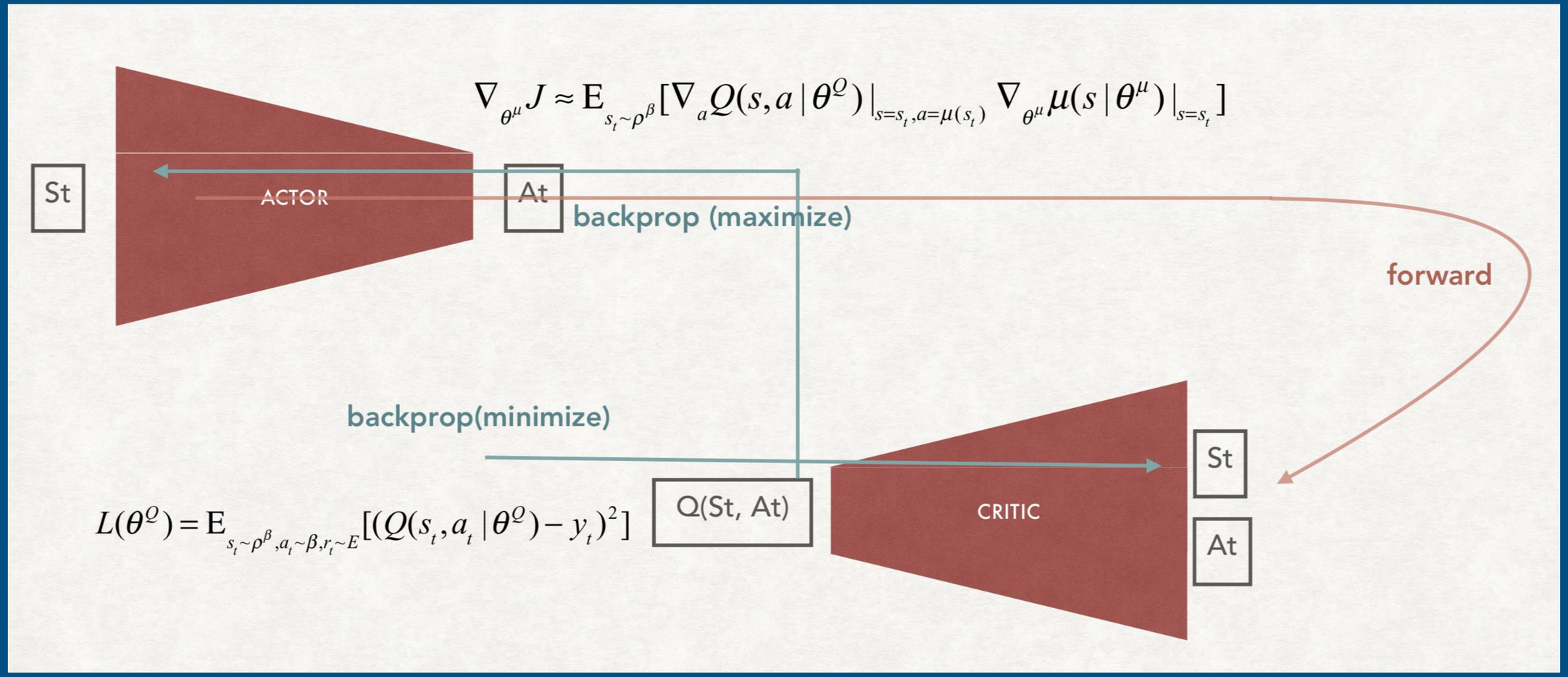
- 탐험을 위해 행동에 노이즈 추가
  - 연속 행동 공간에서 딥러닝 학습의 주요 과제는 탐험(Exploration)이다.
  - DDPG에서는 탐험을 위해 출력으로 나온 행동에 노이즈를 추가한다.
  - Ornstein Uhlenbeck(OU) Process

$$dx_t = \theta(\mu - x_t)dt + \sigma dW_t$$

    - OU Process는 평균으로 회귀하는 랜덤 프로세스다.
    - θ는 얼마나 빨리 평균으로 회귀할 지를 나타내는 매개 변수이며 μ는 평균을 의미한다.
    - σ는 프로세스의 변동성을 의미하며 W_t는 Wiener Process를 의미한다.
    - 시간적으로 연관성이 있는 노이즈 프로세스를 사용한다.
      (물리적인 제어처럼 관성이 있는 환경에서 학습시킬 때 보다 효과적이기 때문)

Deep Deterministic PG (DDPG)

- DDPG Diagram



Deep Deterministic PG (DDPG)

- DDPG Pesudocode

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:

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

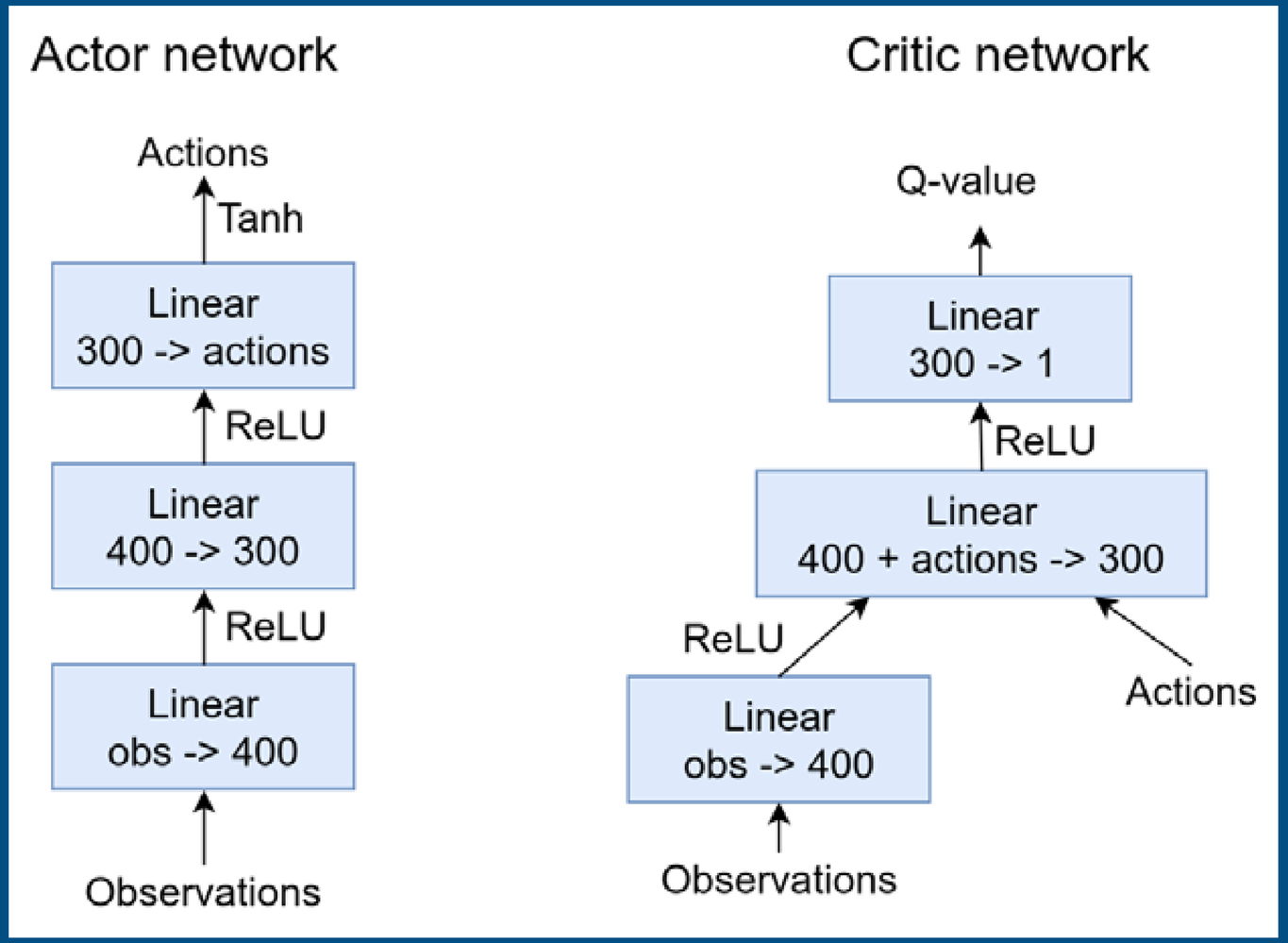
    Update the target networks:

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

  end for
end for

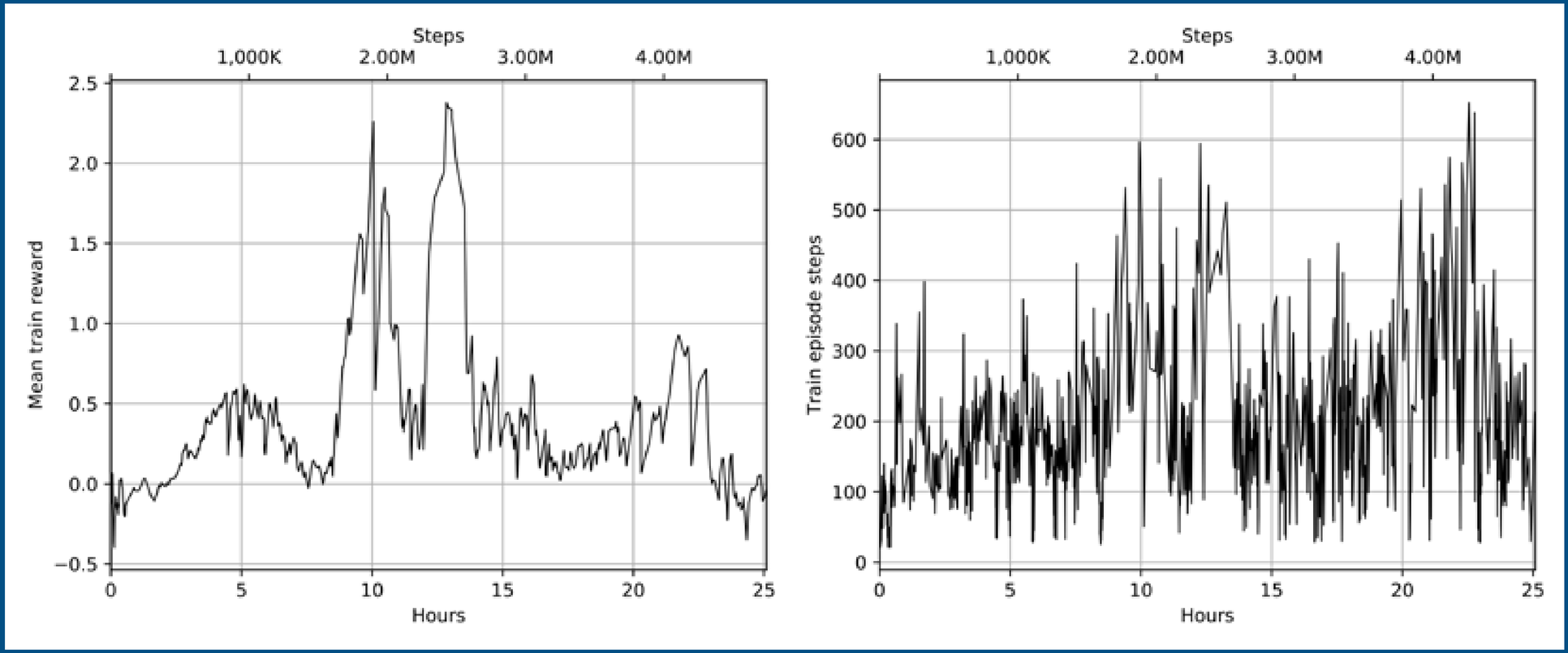
Deep Deterministic PG (DDPG)

- Actor & Critic Network



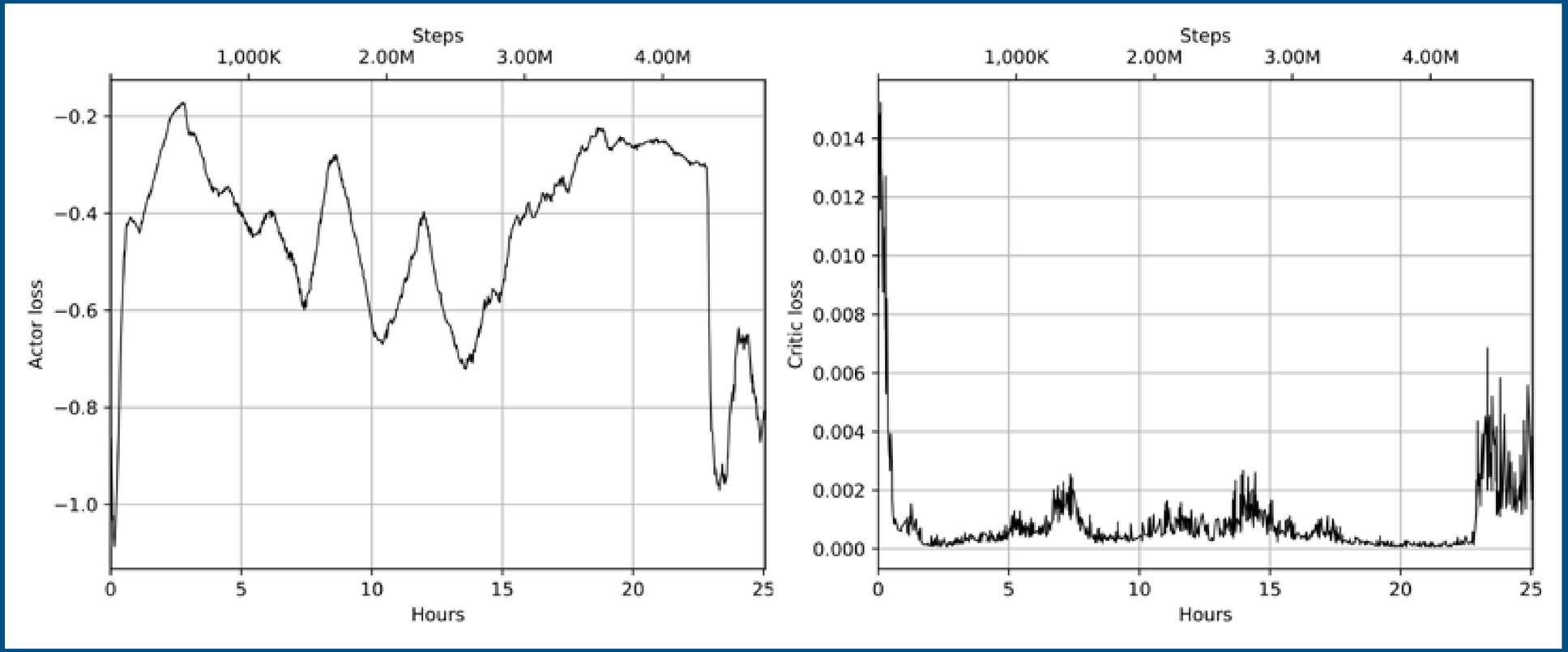
Deep Deterministic PG (DDPG)

- DDPG Training Reward and Episode Steps



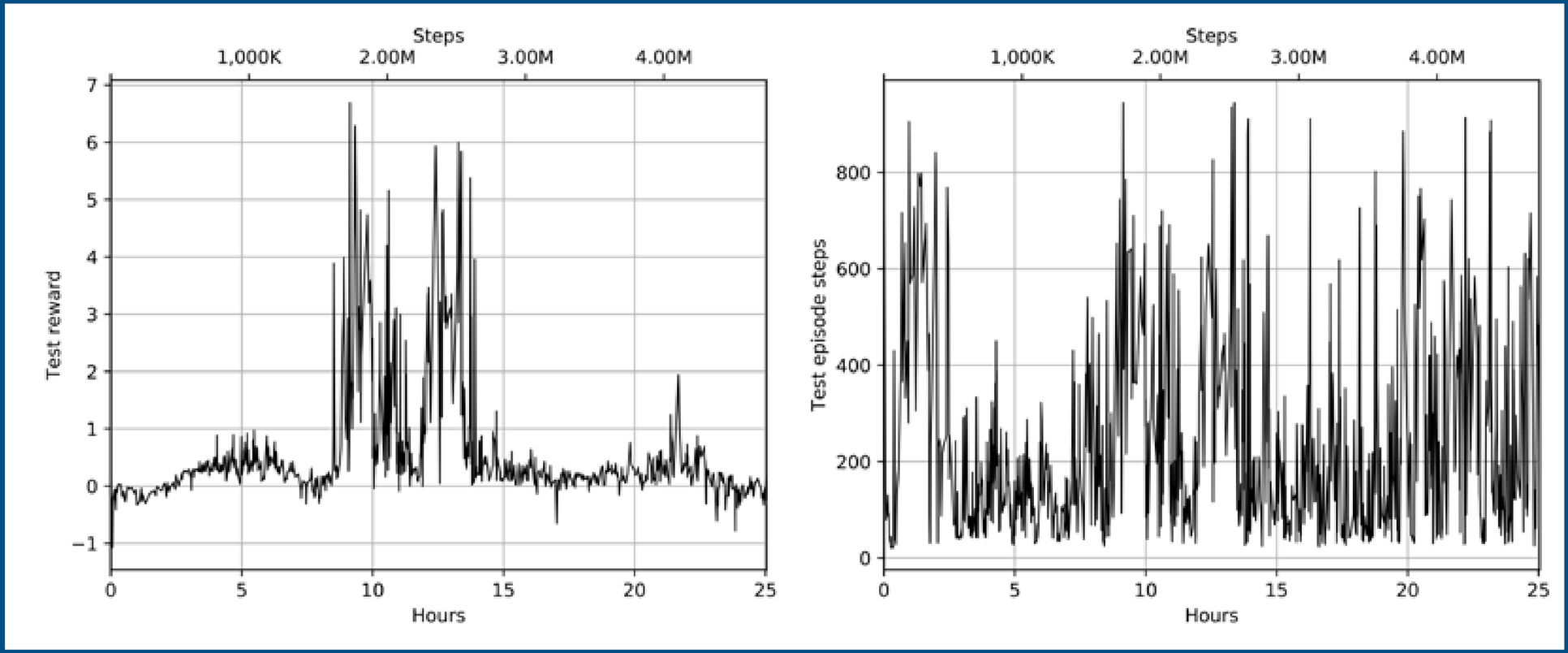
Deep Deterministic PG (DDPG)

- Actor Loss and Critic Loss during the DDPG Training



Deep Deterministic PG (DDPG)

- Test Reward and Test Steps



Distribut{ed|ional} DDPG (D4PG)

- Distributed Distributional Deterministic Policy Gradients (Barth-Maron and Hoffman, 2018)
- D4PG = DDPG + @(Distributional)
- D4PG의 특징
  - Distributional Critic
  - N-step Returns
  - Multiple Distributed Parallel Actors
  - Prioritized Experienced Replay (PER)

Distribut{ed|ional} DDPG (D4PG)

- Distributional Critic
  - Critic Network는 예상 Q 값을 하나의 확률 변수로 예측하는데,
    ω를 통해 매개 변수화되는 분포 Z_ω를 갖는다. 이를 통해 $Q_\omega(s,a) = \mathbb{E}[Z_\omega(x,a)]$를 갖는다.
  - 이 때 분포 매개 변수를 학습하는데 있어 Loss는 두 분포 사이의 거리를 수치화시킨 어떤 값을 최소화하는 것이 될텐데, 논문에서는 이를 Distributional TD Error라고 표현하고 있다.

$$L(\omega) = \mathbb{E}_\rho \left[d\left(\mathcal{T}_{\mu_\theta}, Z_{\omega'}(x,a), Z_\omega(x,a) \right) \right]$$

  - 이를 이용해서 DPG에서 도출된 폴리시 그레디언트 업데이트 식을 다음과 같이 바꿀 수 있다.

$$\begin{aligned} \nabla_\theta J(\theta) &\approx \mathbb{E}_\rho \Big[\nabla_\theta \pi_\theta(\mathbf{x})\, \nabla_{\mathbf{a}} Q_w(\mathbf{x},\mathbf{a})\big|_{\mathbf{a}=\pi_\theta(\mathbf{x})} \Big], \\ &= \mathbb{E}_\rho \Big[\nabla_\theta \pi_\theta(\mathbf{x})\, \mathbb{E}[\nabla_{\mathbf{a}} Z_w(\mathbf{x},\mathbf{a})]\big|_{\mathbf{a}=\pi_\theta(\mathbf{x})} \Big]. \end{aligned}$$

Distribut{ed|ional} DDPG (D4PG)

- N-step Returns
  - TD-error를 계산할 때, 이후 Step에서 얻어질 보상을 반영하기 위해서 1-step이 아닌 N-step Target Error를 구한다. 그래서 이에 따른 새로운 TD Error는 다음과 같이 정의된다.

$$(\mathcal{T}_\pi^N Q)(x_0, a_0) = r(x_0, a_0) + \mathbb{E}\left[\sum_{n=1}^{N-1} \gamma^n r(x_n, a_n) + \gamma^N Q\big(x_N, \pi(x_N)\big) | x_0, a_0\right]$$

Distribut{ed|ional} DDPG (D4PG)

- Multiple Distributed Parallel Actors
  - D4PG는 K의 Actor를 운용하면서 Experience를 병렬로 수집하고 같은 리플레이 버퍼에 데이터를 넣는다.
- Prioritized Experienced Replay (PER)
  - 크기가 R인 리플레이 버퍼에서 균등하지 않은 확률 p_i로 샘플링을 수행한다.
    이 방법을 통해서 샘플 i는 $(Rp_i)^{-1}$의 확률로 선택되게 되고,
    이로 인해 Importance Weight는 $(Rp_i)^{-1}$가 된다.

Distribut{ed|ional} DDPG (D4PG)

- D4PG Pesudocode

Algorithm 1 D4PG

Input: batch size M, trajectory length N, number of actors K, replay size R, exploration constant ϵ, initial learning rates α_0 and β_0

Initialize network weights (θ, w) at random
Initialize target weights $(\theta', w') \leftarrow (\theta, w)$
Launch K actors and replicate network weights (θ, w) to each actor
for $t = 1, \ldots, T$ **do**
  Sample M transitions $(\mathbf{x}_{i:i+N}, \mathbf{a}_{i:i+N-1}, r_{i:i+N-1})$ of length N from replay with priority p_i
  Construct the target distributions $Y_i = \sum_{n=0}^{N-1} \gamma^n r_{i+n} + \gamma^N Z_{w'}(\mathbf{x}_{i+N}, \pi_{\theta'}(\mathbf{x}_{i+N}))$
  Compute the actor and critic updates

$$\delta_w = \frac{1}{M} \sum_i \nabla_w (R p_i)^{-1} d(Y_i, Z_w(\mathbf{x}_i, \mathbf{a}_i))$$

$$\delta_\theta = \frac{1}{M} \sum_i \nabla_\theta \pi_\theta(\mathbf{x}_i)\, \mathbb{E}[\nabla_{\mathbf{a}} Z_w(\mathbf{x}_i, \mathbf{a})]\big|_{\mathbf{a}=\pi_\theta(\mathbf{x}_i)}$$

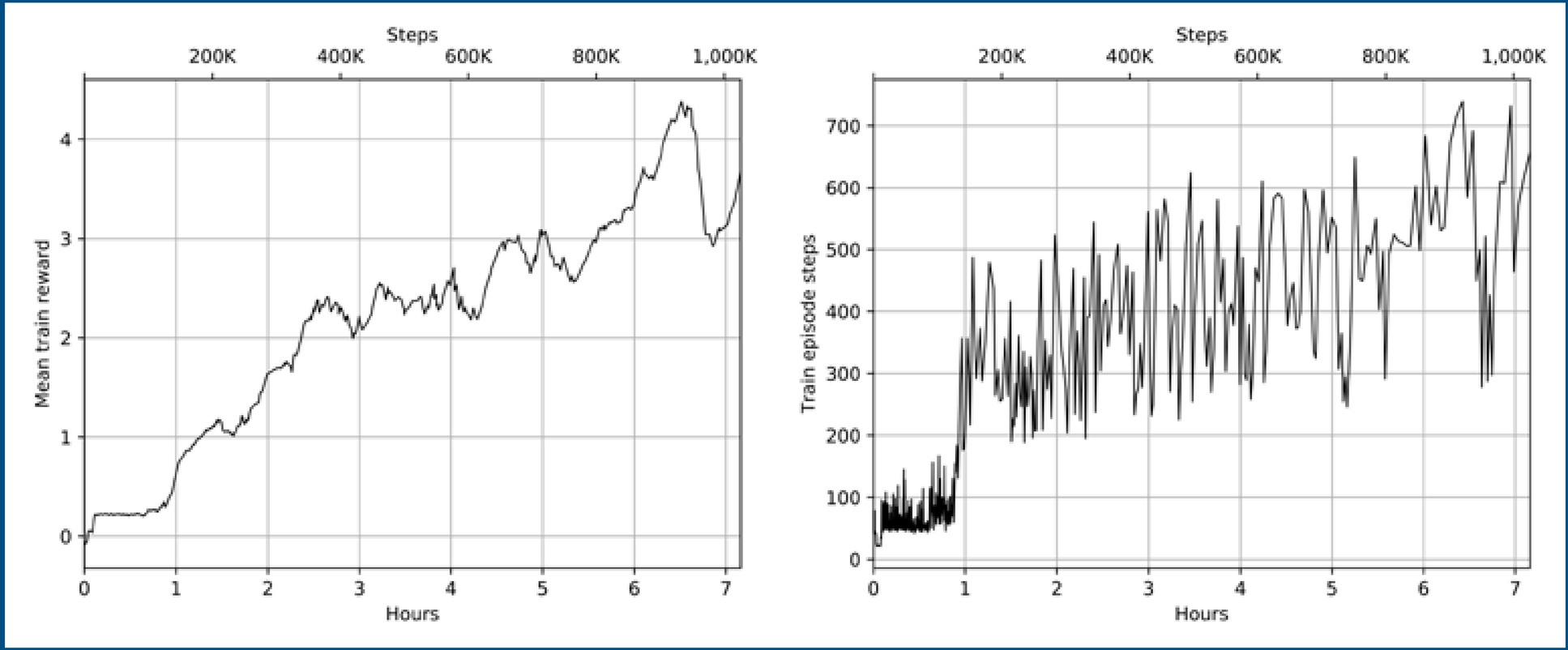
  Update network parameters $\theta \leftarrow \theta + \alpha_t\, \delta_\theta$, $w \leftarrow w + \beta_t\, \delta_w$
  If $t = 0 \mod t_{\text{target}}$, update the target networks $(\theta', w') \leftarrow (\theta, w)$
  If $t = 0 \mod t_{\text{actors}}$, replicate network weights to the actors
end for
return policy parameters θ

Actor

repeat
  Sample action $\mathbf{a} = \pi_\theta(\mathbf{x}) + \epsilon \mathcal{N}(0, 1)$
  Execute action $\mathbf{a}$, observe reward r and state $\mathbf{x}'$
  Store $(\mathbf{x}, \mathbf{a}, r, \mathbf{x}')$ in replay
until learner finishes

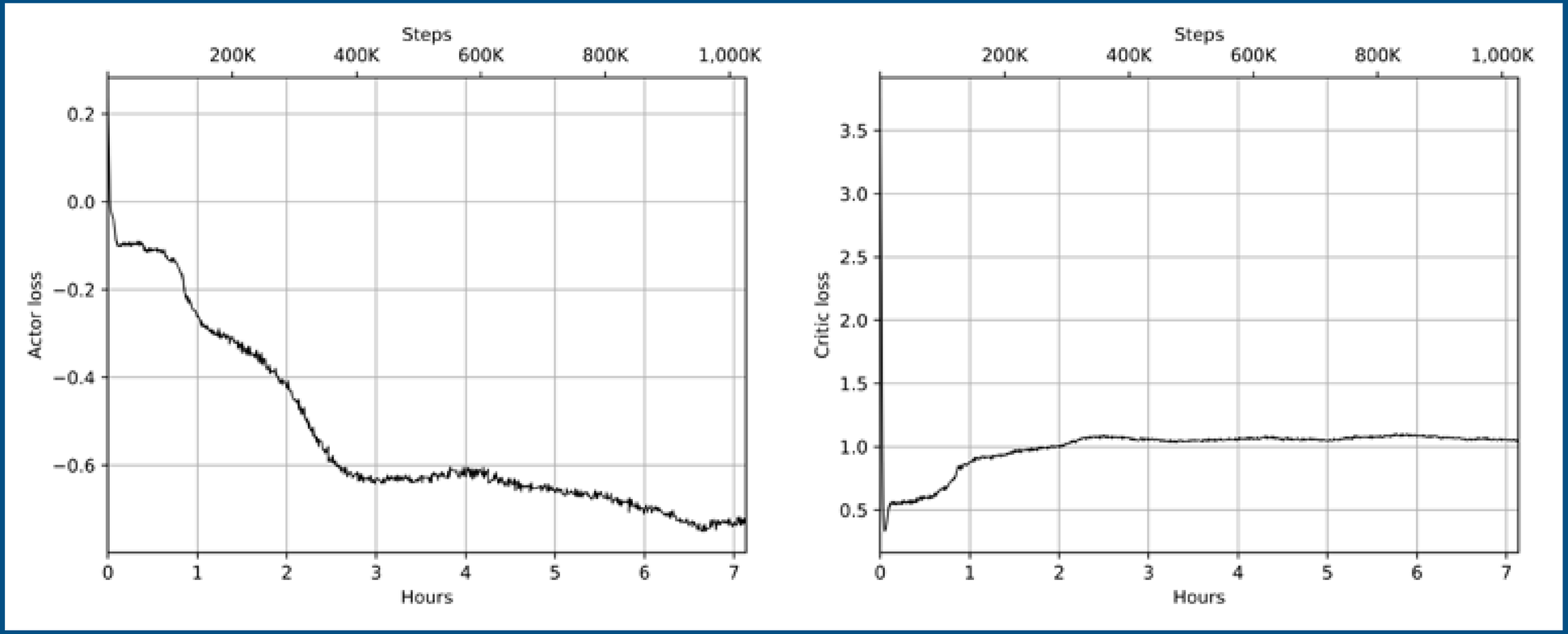
Distribut{ed|ional} DDPG (D4PG)

- D4PG Training Reward and Episode Steps



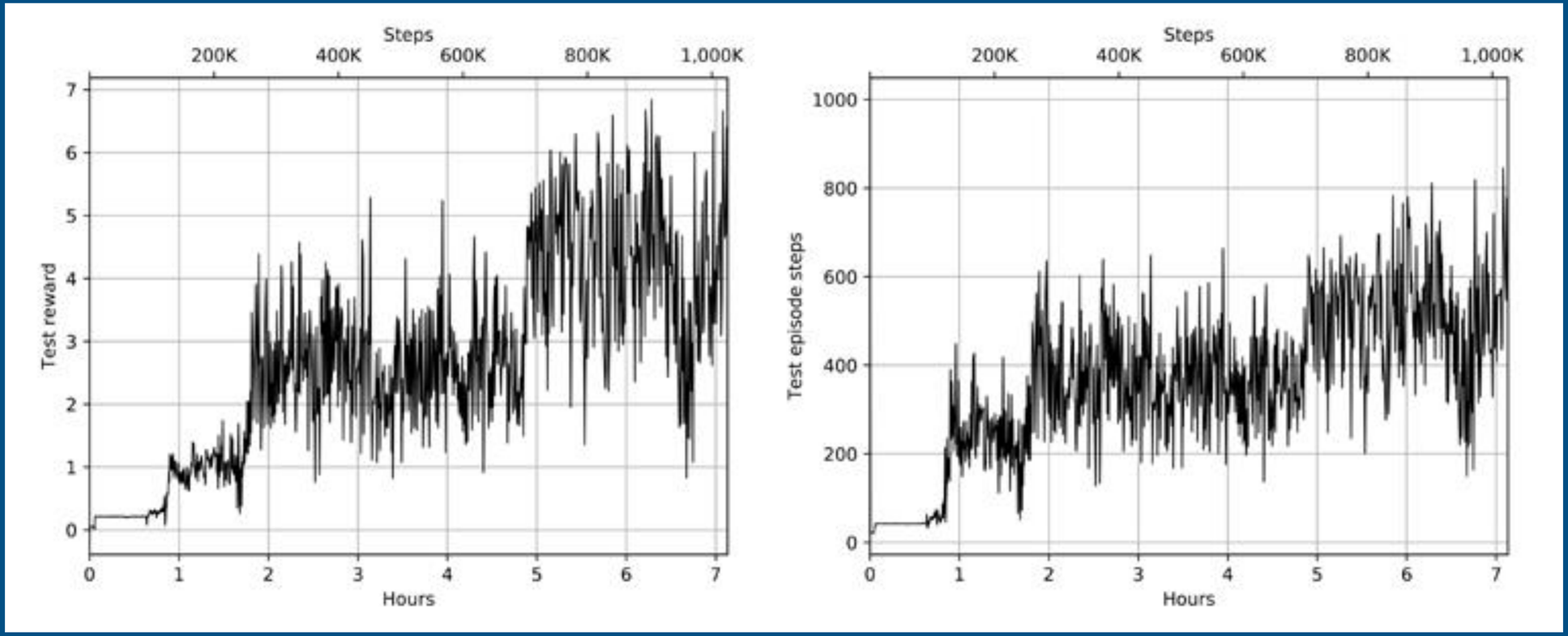
Distribut{ed|ional} DDPG (D4PG)

- Actor Loss and Critic Loss during the D4PG Training



Distribut{ed|ional} DDPG (D4PG)

- Test Reward and Test Steps



감사합니다!

스터디 듣느라 고생 많았습니다.