

Continuous Control with Deep Reinforcement Learning, TP. Lillicrap et al, 2015

발표자 김봉석

CONTENTS

001 **Deterministic Policy Gradient review**

002 **DDPG review**

003 **Experiment**

Part 1.

Deterministic Policy Gradient review



Deterministic Policy Gradient review

Deterministic Policy Gradient Algorithms

David Silver

DeepMind Technologies, London, UK

Guy Lever

University College London, UK

Nicolas Heess, Thomas Degris, Daan Wierstra, Martin Riedmiller

DeepMind Technologies, London, UK

DAVID@DEEPMIND.COM

GUY.LEVER@UCL.AC.UK

*@DEEPMIND.COM

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

Deep / Deterministic Policy Gradient

Stochastic Policy Gradient review

Policy gradient algorithms are perhaps the most popular class of continuous action reinforcement learning algorithms.

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

(Policy Performance objective)

The basic idea behind these algorithms is **to adjust the parameters θ of the policy in the direction of the performance gradient**

$$\begin{aligned} \nabla_\theta J(\pi_\theta) &= \int_S \rho^\pi(s) \int_{\mathcal{A}} \nabla_\theta \pi_\theta(a|s) Q^\pi(s, a) da ds \\ &= \mathbb{E}_{s \sim \rho^\pi, a \sim \pi_\theta} [\nabla_\theta \log \pi_\theta(a|s) \boxed{Q^\pi(s, a)}] \quad (2) \end{aligned}$$

how to estimate the action-value function $Q^\pi(s, a) \rightarrow G_t$ (Reinforce .. Etc)

Stochastic Actor Critic Algorithm

$$\begin{aligned}\nabla_{\theta} J(\pi_{\theta}) &= \int_S \rho^{\pi}(s) \int_{\mathcal{A}} \nabla_{\theta} \pi_{\theta}(a|s) Q^{\pi}(s, a) da ds \\ &= \mathbb{E}_{s \sim \rho^{\pi}, a \sim \pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(a|s) Q^{\pi}(s, a)] \quad (2)\end{aligned}$$

Critic :

Instead of True $Q(s, a)$, estimated an action-value function $Q_w(s, a)$ is used with parameter vector w

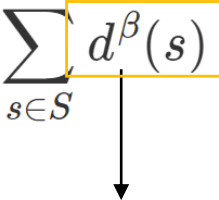
Ex) temporal-difference

Actor:

An actor adjusts the parameters θ of the stochastic policy $\pi_{\theta}(s)$ by stochastic gradient ascent

Off-Policy Actor-Critic

1. Off-policy 방법은 full trajectory가 필요하지 않기 때문에, sample efficiency 측면에서 지난 episode의 sample을 뽑아서 사용 할 수 있다.
2. Sample trajectory가 우리가 구하고자 하는 target policy와 다른 behavior policy로 부터 구할 수 있다. 이를 통해 exploration 측면에서 더 좋을 수 있다.

$$J(\theta) = \sum_{s \in S} d^{\beta}(s) \sum_{a \in A} Q^{\pi}(s, a) \pi_{\theta}(a|s) = \mathbb{E}_{s \sim d^{\beta}} \left[\sum_{a \in A} Q^{\pi}(s, a) \pi_{\theta}(a|s) \right]$$


Behavior policy에 따른 Stationary distribution

Off-Policy Actor-Critic

$$\begin{aligned}\nabla_{\theta} J(\pi_{\theta}) &= \int_{\mathcal{S}} \rho^{\pi}(s) \int_{\mathcal{A}} \nabla_{\theta} \pi_{\theta}(a|s) Q^{\pi}(s, a) da ds \\ &= \mathbb{E}_{s \sim \rho^{\pi}, a \sim \pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(a|s) Q^{\pi}(s, a)] \quad (2)\end{aligned}$$

$$\begin{aligned}\nabla_{\theta} J_{\beta}(\pi_{\theta}) &\approx \int_{\mathcal{S}} \int_{\mathcal{A}} \rho^{\beta}(s) \nabla_{\theta} \pi_{\theta}(a|s) Q^{\pi}(s, a) da ds \quad (4) \\ &= \mathbb{E}_{s \sim \rho^{\beta}, a \sim \beta} \left[\frac{\pi_{\theta}(a|s)}{\beta_{\theta}(a|s)} \nabla_{\theta} \log \pi_{\theta}(a|s) Q^{\pi}(s, a) \right] \quad (5)\end{aligned}$$

Importance weight

Off-policy 에 맞춰 policy gradient를 적용할 경우, 이를 importance weight를 통해 조정 할 수 있고, 이때의 weight는 behavior policy와 target policy 간의 비율로 구할 수 있다.

Deterministic Policy Gradient

Follow a deterministic policy
 $\mu_\theta : S \rightarrow A$

(Deterministic Policy Gradient)

$$\begin{aligned} J(\mu_\theta) &= \int_S \rho^\mu(s) r(s, \mu_\theta(s)) ds \\ &= \mathbb{E}_{s \sim \rho^\mu} [r(s, \mu_\theta(s))] \end{aligned}$$

$$\begin{aligned} \nabla_\theta J(\mu_\theta) &= \int_S \rho^\mu(s) \nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s, a)|_{a=\mu_\theta(s)} ds \\ &= \mathbb{E}_{s \sim \rho^\mu} \left[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s, a)|_{a=\mu_\theta(s)} \right] \quad (9) \end{aligned}$$

Follows stochastic policy
 $\pi_\theta : S \rightarrow P(A)$,

$$\begin{aligned} J(\pi_\theta) &= \int_S \rho^\pi(s) \int_A \pi_\theta(s, a) r(s, a) da ds \\ &= \mathbb{E}_{s \sim \rho^\pi, a \sim \pi_\theta} [r(s, a)] \end{aligned}$$

$$\begin{aligned} \nabla_\theta J(\pi_\theta) &= \int_S \rho^\pi(s) \int_A \nabla_\theta \pi_\theta(a|s) Q^\pi(s, a) da ds \\ &= \mathbb{E}_{s \sim \rho^\pi, a \sim \pi_\theta} [\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s, a)] \quad (2) \end{aligned}$$

the deterministic policy gradient is indeed a special (limiting) case of the stochastic policy gradient.

$$\pi_{\mu_\theta, 0} \equiv \mu_\theta.$$

Off-policy Deterministic Policy Gradient

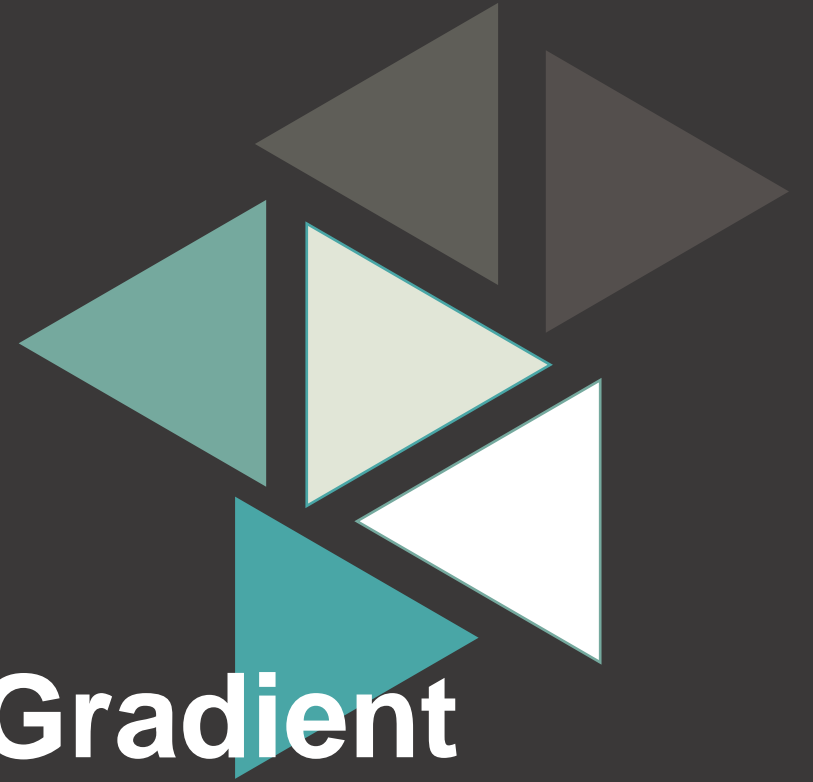
Say, in the off-policy approach, the training trajectories are generated by a stochastic policy $\beta(a|s)$ and thus the state distribution follows the corresponding discounted state density ρ^β

$$J_\beta(\theta) = \int_s \rho^\beta Q^\mu(s, \mu_\theta(s)) ds$$
$$\nabla_\theta J_\beta(\theta) = E_{s \sim \rho^\beta} [\nabla_a Q^\mu(s, a) \nabla_\theta \mu_\theta(s) |_{a=\mu_\theta(s)}]$$

However, because the deterministic policy gradient removes the integral over actions,
we can avoid importance sampling.

Part 2.

Deep Deterministic Policy Gradient (DDPG)



Deep Deterministic Policy Gradient

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

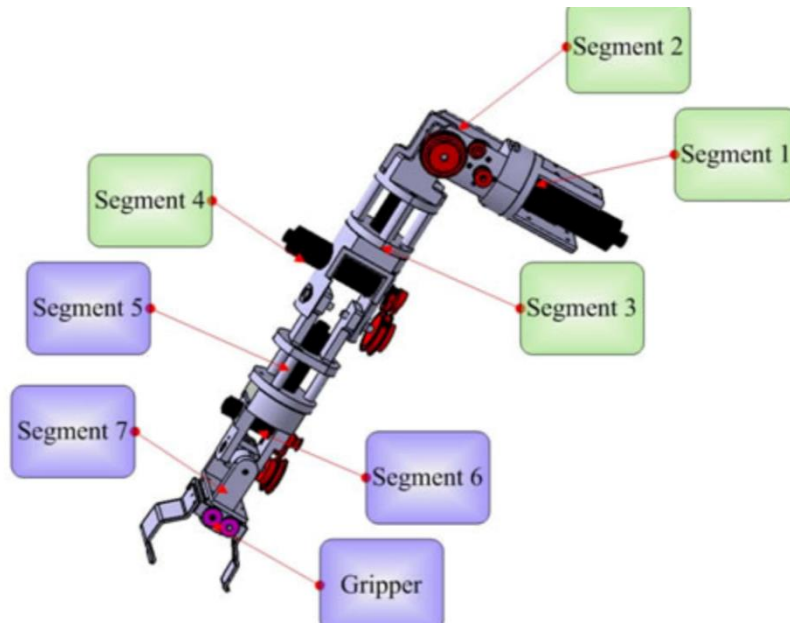
`{countzero, jjhunt, apritzel, heess,
etom, tassa, davidsilver, wierstra} @ google.com`

ABSTRACT

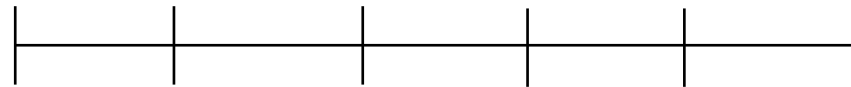
We adapt the ideas underlying the success of Deep Q-Learning to the continuous action domain. We present an actor-critic, model-free algorithm based on the deterministic policy gradient that can operate over continuous action spaces. Using the same learning algorithm, network architecture and hyper-parameters, our algorithm robustly solves more than 20 simulated physics tasks, including classic

Deep Deterministic Policy Gradient

DQN의 성공사례,
연속적인 공간에서 Q-learning을 사용하는 법 ?



DQN을 continuous domain에 적용을 시키는 방법
= action space를 discretization 한다 (action space를 등 간격으로 나눔)



Ex) 7개의 관절을 가진 로봇 팔,
 $3^7 = 2187$ 가지의 action space dimension $a_i \in \{-k, 0, k\}$

너무 작게 행동 공간을 쪼개면 -> 계산/학습 불리

너무 크게 쪼개면 , 정확한 동작이 어려움

Deep Deterministic Policy Gradient

연속적인 공간에서 Q-learning을 사용하는 법 ?

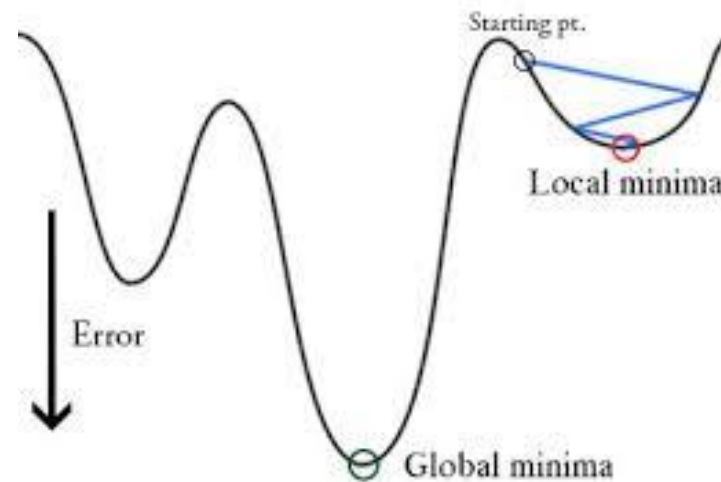
Q-learning target 계산에 필요한
Max $Q(s', a)$ 를 찾는 것은 또 하나의 Optimization 문제

$$Q(s, a) = r(s, a) + \gamma \max_a Q(s', a)$$

찾았다 하더라도 항상 낮은 precision에서 더 좋은 policy 존재

Q-learning 자체가 비효율적이게 됨.

> Actor-critic을 활용해 Policy gradient를 해보자 !



Deep Deterministic Policy Gradient

DQN + Deterministic Policy gradient 아이디어를 사용 + 몇가지 안정화 기법들



+



Target Network
Experience replay (DQN) Q를 평가

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

Deterministic Policy gradient

+ Ornstein-Uhlenback 을 통한 exploration

+ soft target update

+ Batch Normalization

(Model-free, Off-policy, Actor-critic algorithm)

Deep Deterministic Policy Gradient

왜 Deterministic Policy Gradient를 사용하는가?

> 계산상의 이점 때문

(stochastic)

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

(deterministic)

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

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

결정적인 정책을 사용 함으로 써, Expectation 수식 계산이 오직 환경에만 의존함

- off-policy로 학습 할 수 있음을 의미
- Stochastic behavior policy β 로 행동하게 하고, Target을 학습 할 수 있음.
- ddpq에서는, Q function approximator를 DQN과 마찬가지로 신경망을 이용
- Soft target update + replay buffer를 +batch Normalization 사용

Deep Deterministic Policy Gradient

<Soft target update>

Critic Network $Q'(s, a | \theta^{Q'})$

- DQN에서는 일정 주기마다 learned network의 weight를 target network로 직접 복사해서 사용

Actor Network $\mu'(s | \theta^{\mu'})$

- DDPG에서는 마찬가지로 target network를 사용하면서, update를 exponential moving average(지수이동평균) 식으로 대체.

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

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

- learned network를 target network가 천천히 따라가게 함으로써 좀더 stable하게 학습 하도록 함

Deep Deterministic Policy Gradient

< Batch Normalization >

- low dimensional feature vector observation으로 학습 하는 경우, 각 feature의 scale 차이로 인해 network가 효과적으로 학습하지 못 함
 - 이경우 직접 스케일을 조절을 시도 할 수 있지만, 적절한 hyper-parameter와 scale을 동시에 찾기 어려움
- deep-learning technique 중 하나인 batch normalization을 사용해, 문제를 해결함

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots x_m\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{ mini-batch mean}$$
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{ mini-batch variance}$$
$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{ normalize}$$
$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{ scale and shift}$$

Deep Deterministic Policy Gradient

< Ornstein-Uhlenback 을 통한 exploration >

- continuous action space의 경우 효과적인 exploration이 어려움
- 따라서 좀 더 효과적으로 exploration 하기 위해, Ornstein-Uhlenback 과정을 이용함.
- Ornstein-Uhlenback 과정은, 시간적으로 서로 연관된 확률 변수를 생성하는 기법
- 시간적으로 연관성을 가지게 한 랜덤 성을 더 해주어 효과적으로 exploration을 할 수 있다.

$$\mu'(s_t) = \mu(s_t | \theta_t^\mu) + \mathcal{N}$$

↑

Noise Process에 이용

$$dx_t = -\theta x_t dt + \sigma dW_t$$

$$\frac{dx_t}{dt} = -\theta x_t + \sigma \eta(t)$$

$\theta > 0$ $\sigma > 0$ 하이퍼 파라미터, white noise

Deep Deterministic Policy Gradient

<정리>

- Off-policy Actor(deterministic Policy gradient)-critic(Q-network)
- DQN과 마찬가지로, Q를 근사 하기 위해 Neural network 사용
- 효과적으로 학습하기 위해 replay buffer + target network 사용
- 안정성을 위해 soft-update +batch Normalization 사용
- actor의 효과적인 exploration 위해 OU process 이용

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

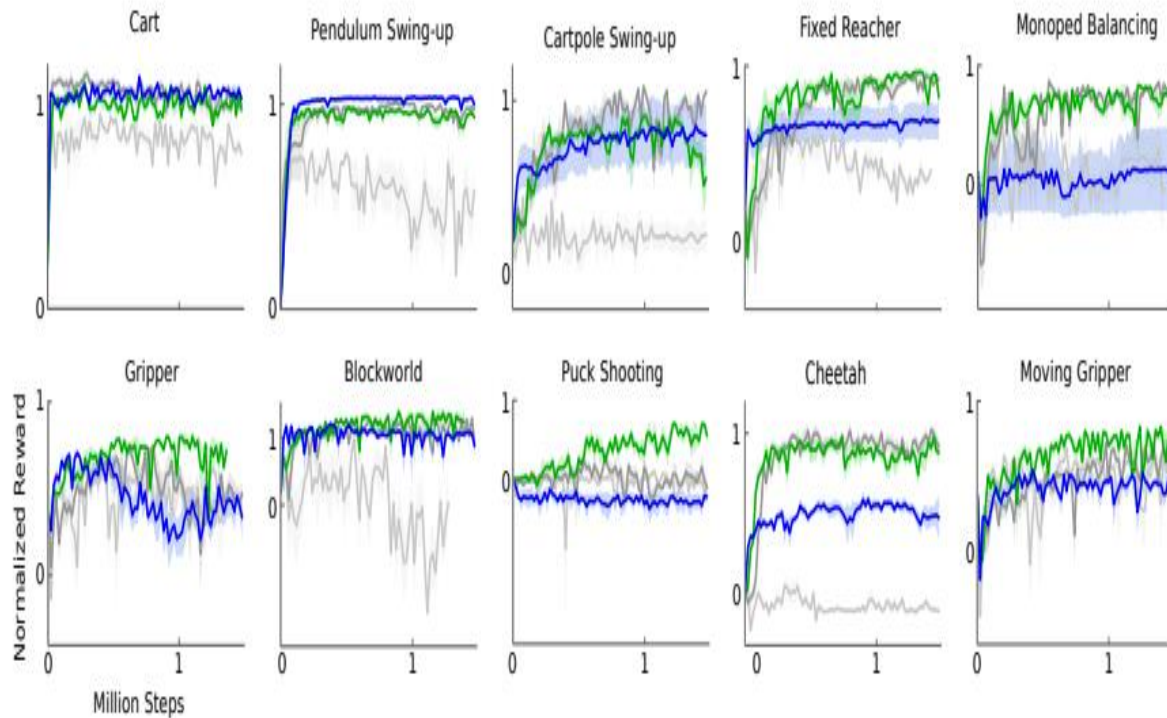
Part 3.

Experiment



Deep Deterministic Policy Gradient

< 실험 결과 >



original DPG에 batch norm_ 만 추가(연한 회색),

target network만 추가(진한 회색)

batch normalization+ target network(초록)

pixel로만 학습 (파랑)

Deep Deterministic Policy Gradient

< 실험 결과 >

environment	$R_{av,lowd}$	$R_{best,lowd}$	$R_{av,pix}$	$R_{best,pix}$	$R_{av,cntrl}$	$R_{best,cntrl}$
blockworld1	1.156	1.511	0.466	1.299	-0.080	1.260
blockworld3da	0.340	0.705	0.889	2.225	-0.139	0.658
canada	0.303	1.735	0.176	0.688	0.125	1.157
canada2d	0.400	0.978	-0.285	0.119	-0.045	0.701
cart	0.938	1.336	1.096	1.258	0.343	1.216
cartpole	0.844	1.115	0.482	1.138	0.244	0.755
cartpoleBalance	0.951	1.000	0.335	0.996	-0.468	0.528
cartpoleParallelDouble	0.549	0.900	0.188	0.323	0.197	0.572
cartpoleSerialDouble	0.272	0.719	0.195	0.642	0.143	0.701
cartpoleSerialTriple	0.736	0.946	0.412	0.427	0.583	0.942
cheetah	0.903	1.206	0.457	0.792	-0.008	0.425
fixedReacher	0.849	1.021	0.693	0.981	0.259	0.927
fixedReacherDouble	0.924	0.996	0.872	0.943	0.290	0.995
fixedReacherSingle	0.954	1.000	0.827	0.995	0.620	0.999
gripper	0.655	0.972	0.406	0.790	0.461	0.816
gripperRandom	0.618	0.937	0.082	0.791	0.557	0.808
hardCheetah	1.311	1.990	1.204	1.431	-0.031	1.411
hopper	0.676	0.936	0.112	0.924	0.078	0.917
hyq	0.416	0.722	0.234	0.672	0.198	0.618
movingGripper	0.474	0.936	0.480	0.644	0.416	0.805
pendulum	0.946	1.021	0.663	1.055	0.099	0.951
reacher	0.720	0.987	0.194	0.878	0.231	0.953
reacher3daFixedTarget	0.585	0.943	0.453	0.922	0.204	0.631
reacher3daRandomTarget	0.467	0.739	0.374	0.735	-0.046	0.158
reacherSingle	0.981	1.102	1.000	1.083	1.010	1.083
walker2d	0.705	1.573	0.944	1.476	0.393	1.397
torcs	-393.385	1840.036	-401.911	1876.284	-911.034	1961.600

Pix , Pixel inputs (DDPG - pix)

lowd, low-dimensional observation (DDPG-lowd)

cntrl, target-network, batch_normalization 추가

평가기준 :

Planning (환경의 모든 정보를 다 아는 알고리즘) 1

Random agent 0 기준 normalize

감사합니다

발표자 김봉석