Deep Deterministic Policy Gradient (DDPG)

Timothy et al. (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

What is DDPG?

DDPG = DPG + DQN

- **Deterministic**
- **Continuous action space**
- **Stable**

Background of DPG

Main reason : To compensate for the shortcomings of the DQN algorithm

1. DQN can solves problems with high-dimensional observation spaces. But, it can only handle discrete and low-dimensional action spaces.

2. In real world, the most of problem are have continuous (real valued) and high dimensional action spaces.

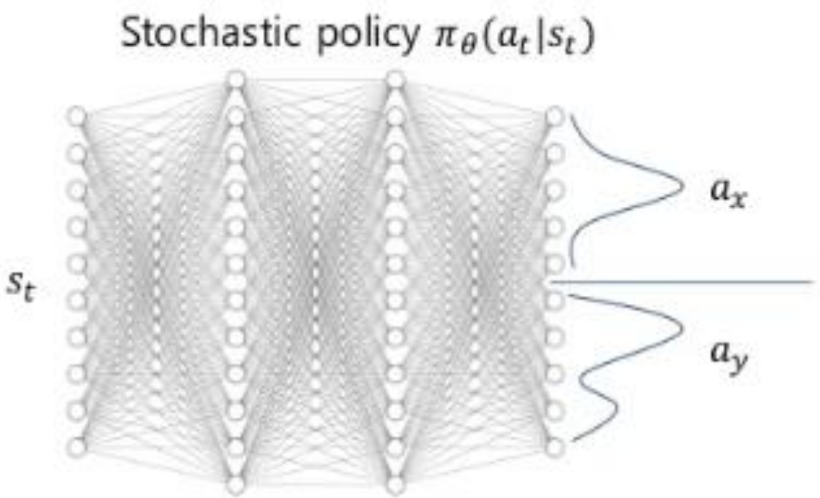
3. DQN shows that powerful performance, but it is hard to applied in real world problem

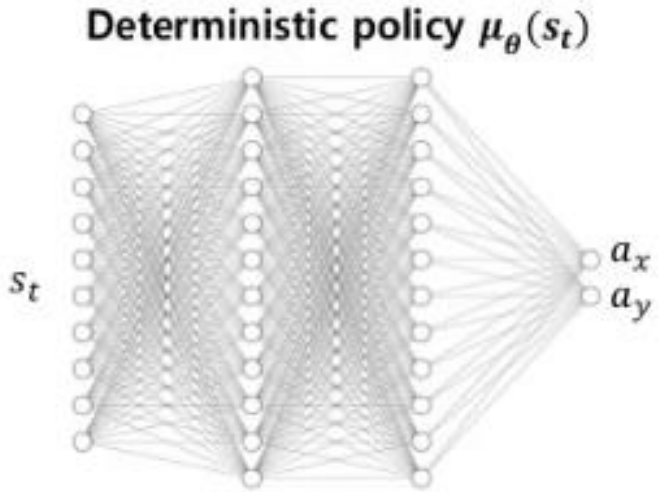
→ Make the new algorithm deterministic & continuous action space

→ DPG Algorithm (Silver et al., 2014)

Background of DPG

- **Deterministic policy gradient (DPG)** models the actor policy as a deterministic policy: $a_t = \mu_\theta(s_t)$





Background of DPG

- **Deterministic policy gradient (DPG)** models the actor policy as a deterministic policy: $a_t = \mu_\theta(s_t)$

trajectory distribution

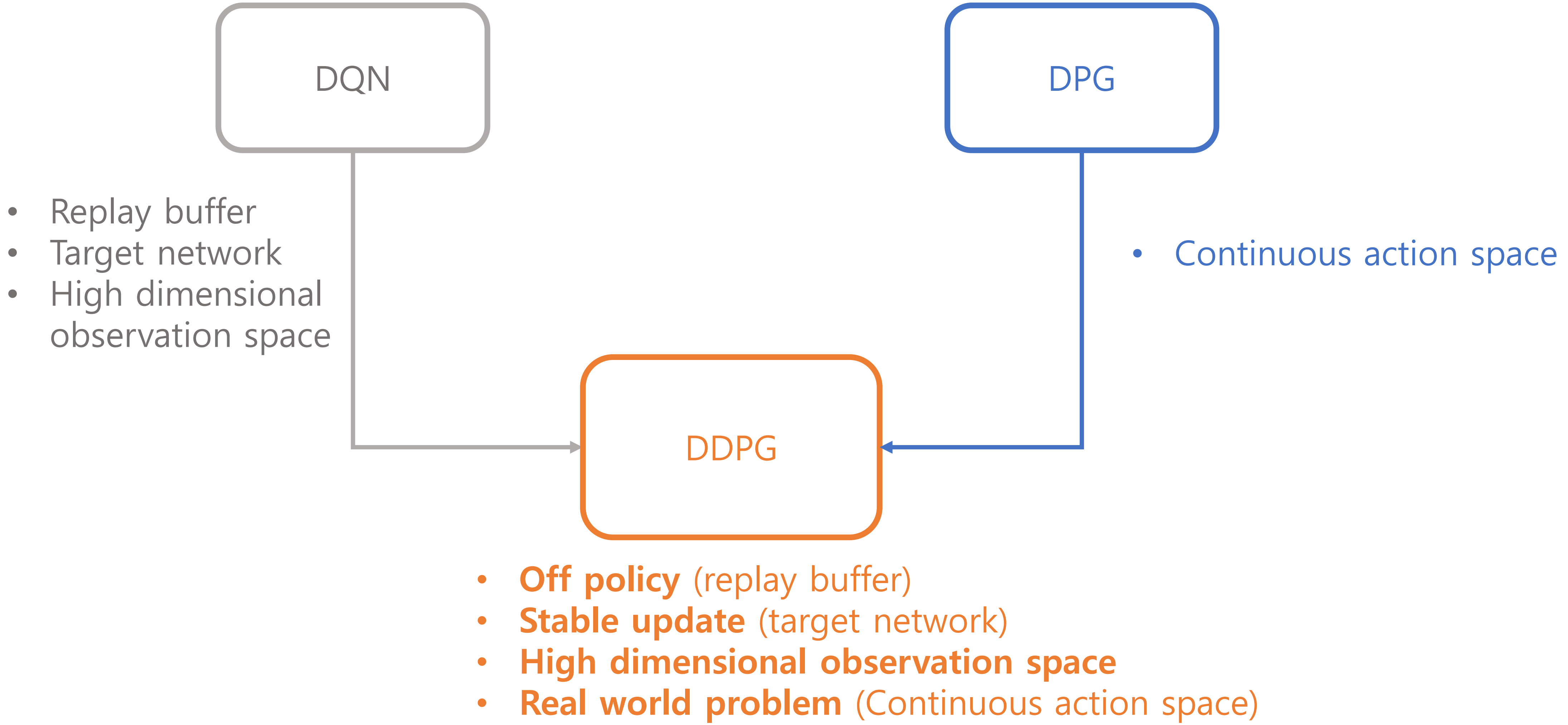
$$\underbrace{p_{\theta(s_1,a_1,\cdots,s_T,a_T)}}_{\tau} = p(s_1)\prod_{t=1}^{T}\pi_\theta(a_t|s_t)p(s_{t+1}|s_t,a_t) \Rightarrow \underbrace{p_{\theta(s_1,s_2,s_3\cdots,s_T)}}_{\tau} = p(s_1)\prod_{t=1}^{T}p(s_{t+1}|s_t,\mu_\theta(s_t))$$

objective

$$J(\theta) = E_{s,a\sim p_\theta(\tau)}[Q_\phi(s_t,a_t)] \Rightarrow J(\theta) = E_{s\sim p_\theta(\tau)}[Q(s,\mu_\theta(s))]$$

1. Sample (s_t, a_t, r_t, s_{t+1}) from $\mu_\theta(s)$ i times
2. Update $Q_\phi(s_t,a_t)$ to samples → **loss:** $L = r_t + \gamma Q_\phi(s_{t+1},\mu_\theta(s_{t+1})) - Q_\phi(s_t,a_t)$
3. $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \mu_\theta(s_t)\nabla_\phi Q_\phi(s_t,a_t)|_{a_t=\mu_\theta(s_t)}$
4. $\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$

DDPG Algorithm



DDPG Algorithm

1. Ornstein-Uhlenback (ou-noise)

$$a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$$

- Add noise for **exploration**

2. Soft update

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

- Slowing the network being trained to follow the target network
- It makes training to **stable**

3. Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

- If learning is carried out with the **observed low-dimensional feature vector**, the network suffers from learning difficulties due to the **difference in the scale of the features**.

- Put the samples in one **minibatch** and **normalize** for all dimensions.

DDPG Algorithm

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

Initialize components

for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration

Ou-noise

  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}

Transition

    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

Store the transition & sampling the minibatch from replay buffer

    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

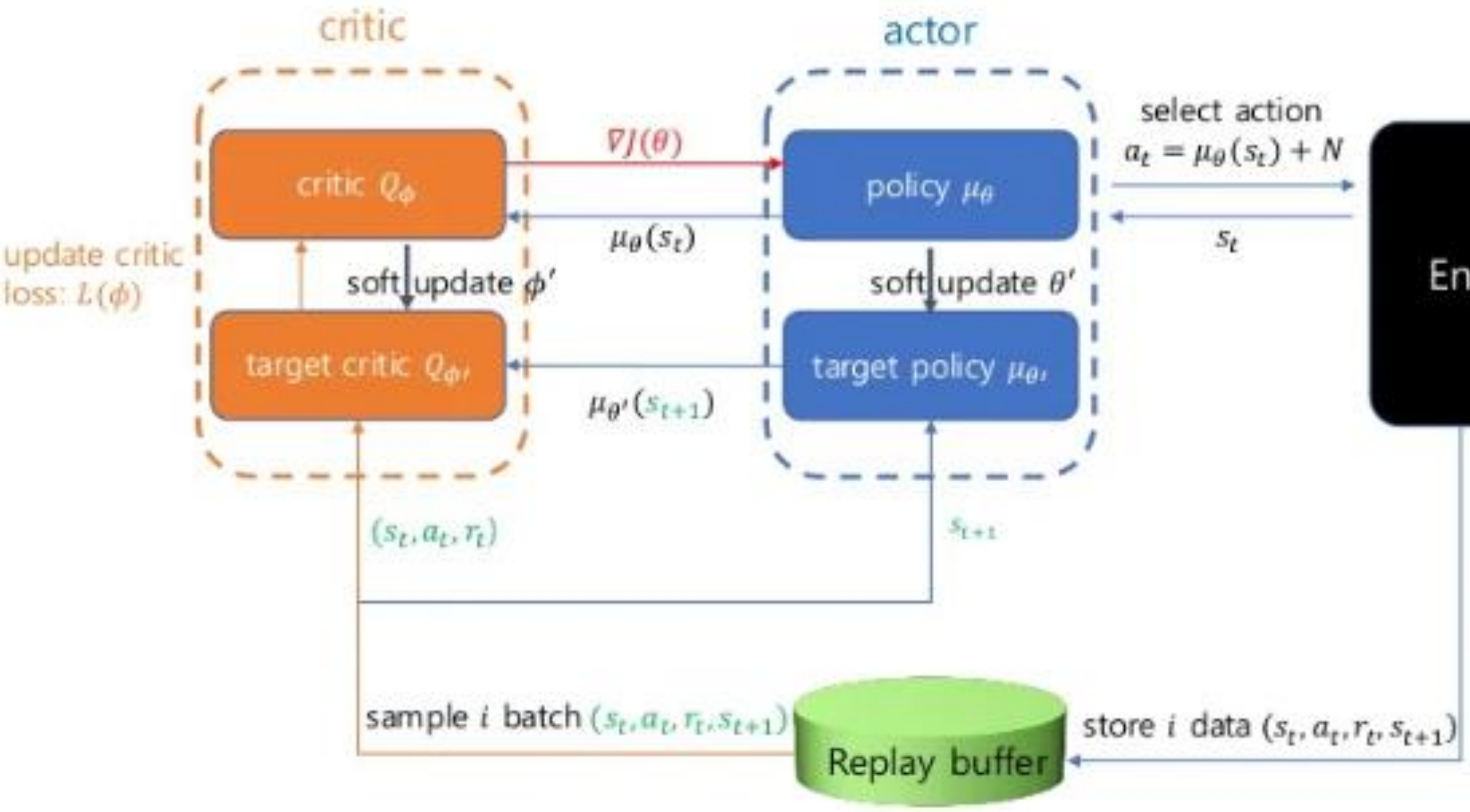
$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$

- **DDPG's target value : Calculate target value through target critic network**
- **(input : next state & actor networks' output(action)**)

- **actor network update**: update the actor's policy using the **policy gradient algorithm**
- critic-network updates the network **parameters** (weights), while actor-net updates the actor's **policy**

Soft update

DDPG Algorithm



DDPG Algorithm

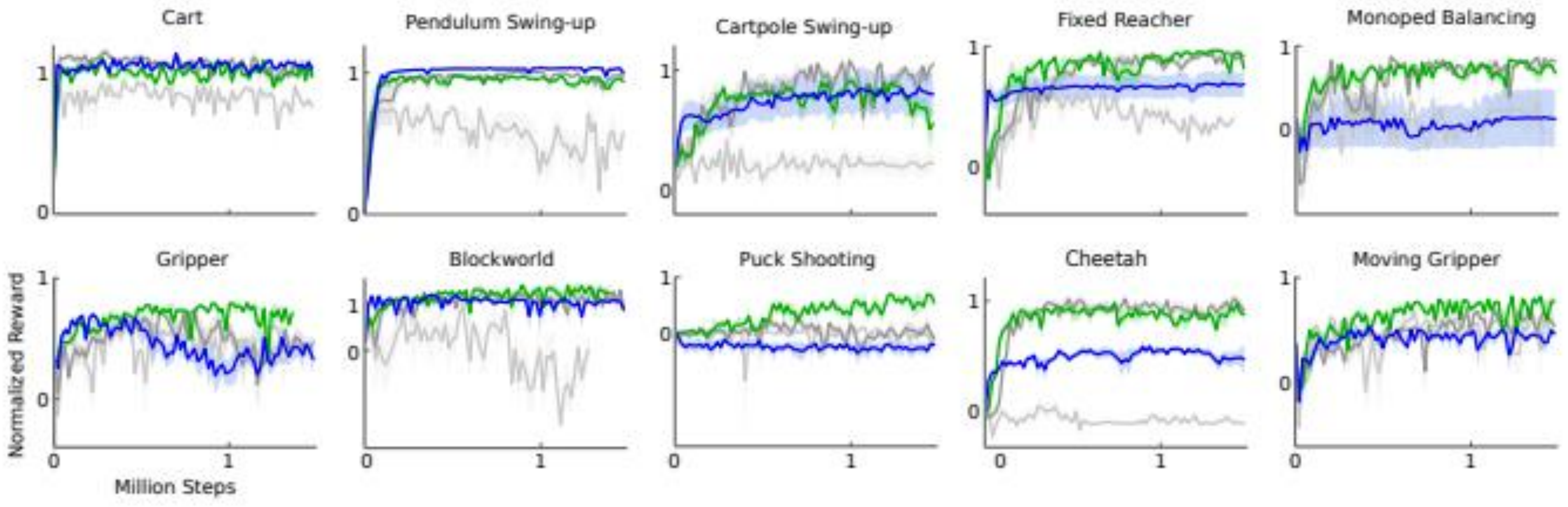


Figure 2: Performance curves for a selection of domains using variants of DPG: original DPG algorithm (minibatch NFQCA) with batch normalization (light grey), with target network (dark grey), with target networks and batch normalization (green), with target networks from pixel-only inputs (blue). Target networks are crucial.