Continuous Control With Deep Reinforcement Learning, Lillicrap et al, 2015.

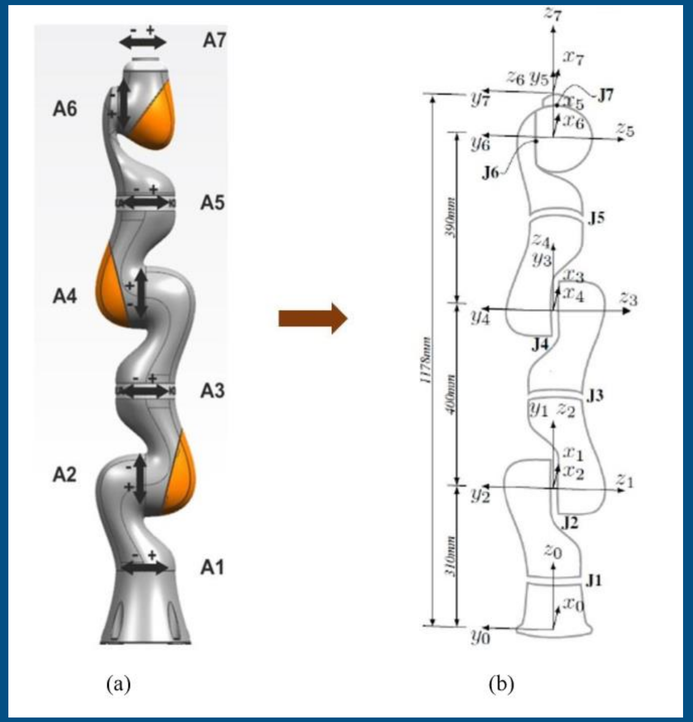
옥찬호
utilForever@gmail.com

Introduction

Continuous Control With Deep Reinforcement Learning, Lillicrap et al, 2015.

- Deep Q Network solve problems…
  - High-dimensional observation spaces
  - Discrete and low-dimensional action spaces
- Limitation of DQN
  - For example, a 7 degree of freedom system with the coarsest discretization $a_i \in \{-k, 0, k\}$ for each joint leads to an action space with dimensionality: $3^7 = 2187$.

→ How to solve high-dimensional action spaces problem?



Introduction

Continuous Control With Deep Reinforcement Learning,
Lillicrap et al, 2015.

- Use Deterministic Policy Gradient (DPG) algorithm!
  - However, a naive application of this actor-critic method with neural function approximators is unstable for challenging problems.
    → How?
- Combine actor-critic approach with insights from Deep Q Network (DQN)
  - Use replay buffer to minimize correlations between samples.
  - Use target Q network to give consistent targets.
  - Use batch normalization.

Background

Continuous Control With Deep Reinforcement Learning, Lillicrap et al, 2015.

- The action-value function is used in many reinforcement learning algorithms. It describes the expected return after taking an action a_t in state s_t and thereafter following policy π:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{r_{i \geq t}, s_{i>t} \sim E, a_{i>t} \sim \pi} \left[R_t | s_t, a_t \right]$$

- Action-value function using Bellman-equation

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

- If target policy is deterministic,

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

Background

Continuous Control With Deep Reinforcement Learning, Lillicrap et al, 2015.

- Q-learning, a commonly used off-policy algorithm, uses the greedy policy $\mu(s) = \text{argmax}_a Q(s, a)$. We consider function approximators parameterized by θ^Q, which we optimize by minimizing the loss:

$$L(\theta^Q) = \mathbb{E}_{s_t \sim \rho^\beta, a_t \sim \beta, r_t \sim E} \left[\left(Q(s_t, a_t | \theta^Q) - y_t \right)^2 \right]$$

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

Algorithm

Continuous Control With Deep Reinforcement Learning, Lillicrap et al, 2015.

- It is not possible to straightforwardly apply Q-learning to continuous action space.
  - In continuous spaces finding the greedy policy requires an optimization of a_t at every timestep.

$$\mu(s) = \text{argmax}_a Q(s, a)$$

  - This optimization is too slow to be practical with large, unconstrained function approximators and nontrivial action spaces.
- Instead, here we used an actor-critic approach based on the DPG algorithm.

Algorithm

Continuous Control With Deep Reinforcement Learning, Lillicrap et al, 2015.

- The DPG algorithm maintains a parameterized actor function $\mu(s|\theta^\mu)$ which specifies the current policy by deterministically mapping states to a specific action.
  - Critic network (Value): Using the Bellman equation as in DQN

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

  - Actor network (Policy):

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

Replay buffer

Continuous Control With Deep Reinforcement Learning, Lillicrap et al, 2015.

- As with Q learning, introducing non-linear function approximators means that **convergence is no longer guaranteed**. However, such approximators appear essential in order to learn and generalize on large state spaces.
- NFQCA uses the same update rules as DPG but with neural network function approximators, uses **batch learning for stability**, which is intractable for large networks. A minibatch version of NFQCA which **does not reset the policy at each update**, as would be required to scale to large networks.
- Our contribution here is to provide modifications to DPG, inspired by the success of DQN, which allow it to use neural network function approximators to learn in large state and action spaces **online**. We refer to our algorithm as Deep DPG (DDPG, Algorithm 1).

Replay buffer

Continuous Control With Deep Reinforcement Learning, Lillicrap et al, 2015.

- As in DQN, we used a replay buffer to address these issues. The replay buffer is a finite sized cache $\mathcal{R}$. Transitions were sampled from the environment according to the exploration policy and the tuple (s_t, a_t, r_t, s_{t+1}) was store in the replay buffer. When the replay buffer was full the oldest samples were discarded. At each timestep the actor and critic are updated by sampling minibatch uniformly from the buffer. Because DDPG is an off-policy algorithm, the replay buffer can be large, allowing the algorithm to benefit from learning across a set of uncorrelated transitions.

Soft target update

Continuous Control With Deep Reinforcement Learning, Lillicrap et al, 2015.

- **Directly implementing Q learning (equation 4) with neural networks proved to be unstable in many environments**. Since the network $Q(s, a|\theta^Q)$ being updated is also used in calculating the target value (equation 5), the Q update is prone to divergence.
- Our solution is similar to the target network used in (Mnih et al., 2013) but **modified for actor-critic and using "soft" target updates**, rather than directly copying the weights.

Soft target update

Continuous Control With Deep Reinforcement Learning, Lillicrap et al, 2015.

- We create a copy of the actor and critic networks, $Q'(s, a|\theta^{Q'})$ and $\mu'(s|\theta^{\mu'})$ respectively, that are used for calculating the target values. **The weights of these target networks are then updated by having them slowly track the learned networks: $\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$ with $\tau \ll 1$**. This means that the target values are constrained to change slowly, greatly improving the stability of learning.

Batch normalization

Continuous Control With Deep Reinforcement Learning, Lillicrap et al, 2015.

- When learning from low dimensional feature vector observations, **the different components of the observation may have different physical units** (for example, positions versus velocities) and **the ranges may vary across environments**. This can make it difficult for the network to learn effectively and may make **it difficult to find hyper-parameters which generalize across environments with different scales of state values**.

Batch normalization

Continuous Control With Deep Reinforcement Learning, Lillicrap et al, 2015.

- One approach to this problem is to manually scale the features so they are in similar ranges across environments and units. We address this issue by adapting a recent technique from deep learning called **batch normalization**. This technique normalizes each dimension across the samples in a minibatch to have unit mean and variance.

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

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

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

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

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

Noise process

Continuous Control With Deep Reinforcement Learning, Lillicrap et al, 2015.

- A major challenge of learning in continuous action spaces is exploration. An advantage of off-policies algorithms such as DDPG is that we can treat the problem of exploration independently from the learning algorithm.
- We constructed an exploration policy μ' by adding noise sampled from a noise process $\mathcal{N}$ to our actor policy $\mu'(s_t) = \mu\left(s_t \middle| \theta_t^{\mu}\right) + \mathcal{N}$, where $\mathcal{N}$ can be chosen to suit the environment.
- As detailed in the supplementary materials we used an Ornstein-Uhlenbeck process (Uhlenbeck & Ornstein, 1930) to generate temporally correlated exploration for exploration efficiency in physical control problems with inertia (similar use of autocorrelated noise was introduced in (Wawrzynski, 2015)).

Noise process

Continuous Control With Deep Reinforcement Learning, Lillicrap et al, 2015.

- Ornstein – Uhlenbeck process

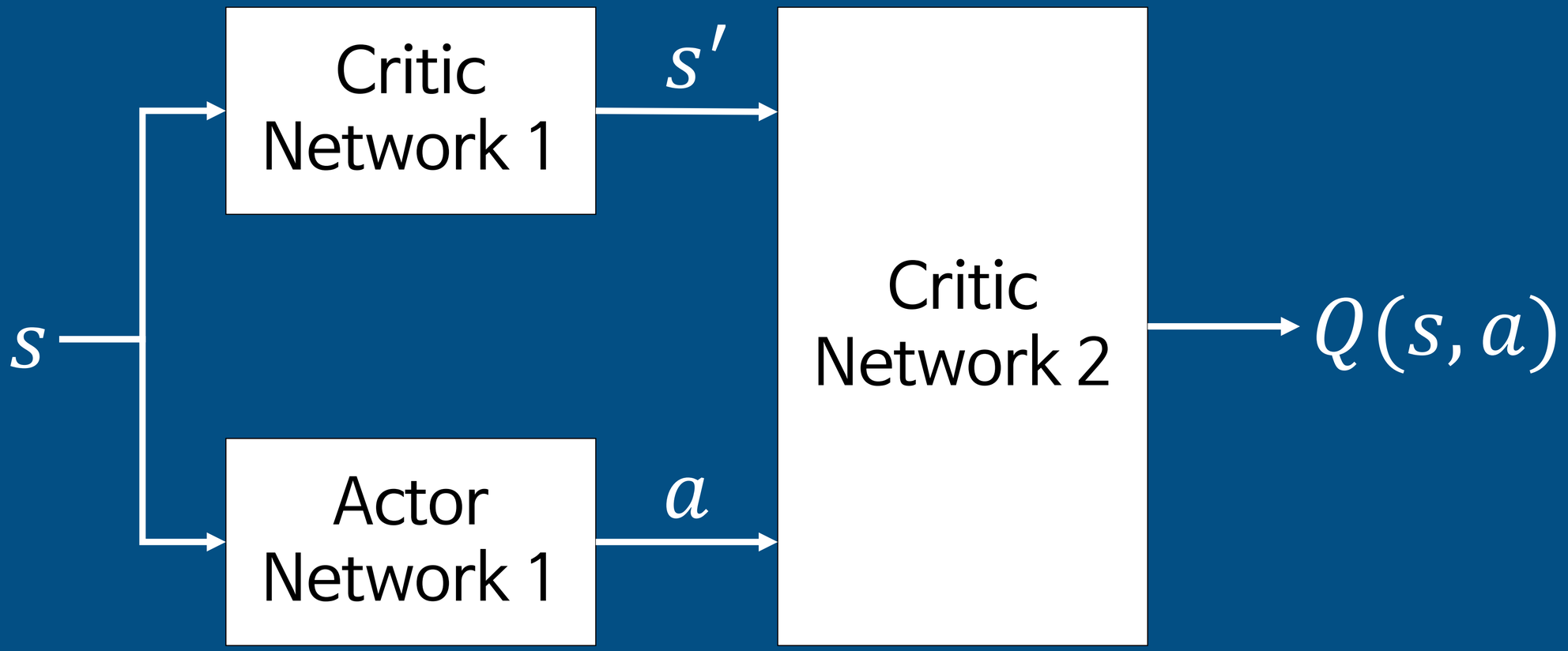
$$dx_t = -\theta\, x_t\, dt + \sigma\, dW_t$$

  - It is a stochastic process that returns to the mean. θ means a parameter that indicates how quickly this process returns to the mean and μ means the mean. σ means variability of processes and W_t means Wiener process.
  - Therefore, it is temporarily correlated with previous noise.
  - The reason for using the temporally correlated noise process is that it is more effective when learning in an inertial environment such as physical control.

Algorithm

Continuous Control With Deep Reinforcement Learning, Lillicrap et al, 2015.

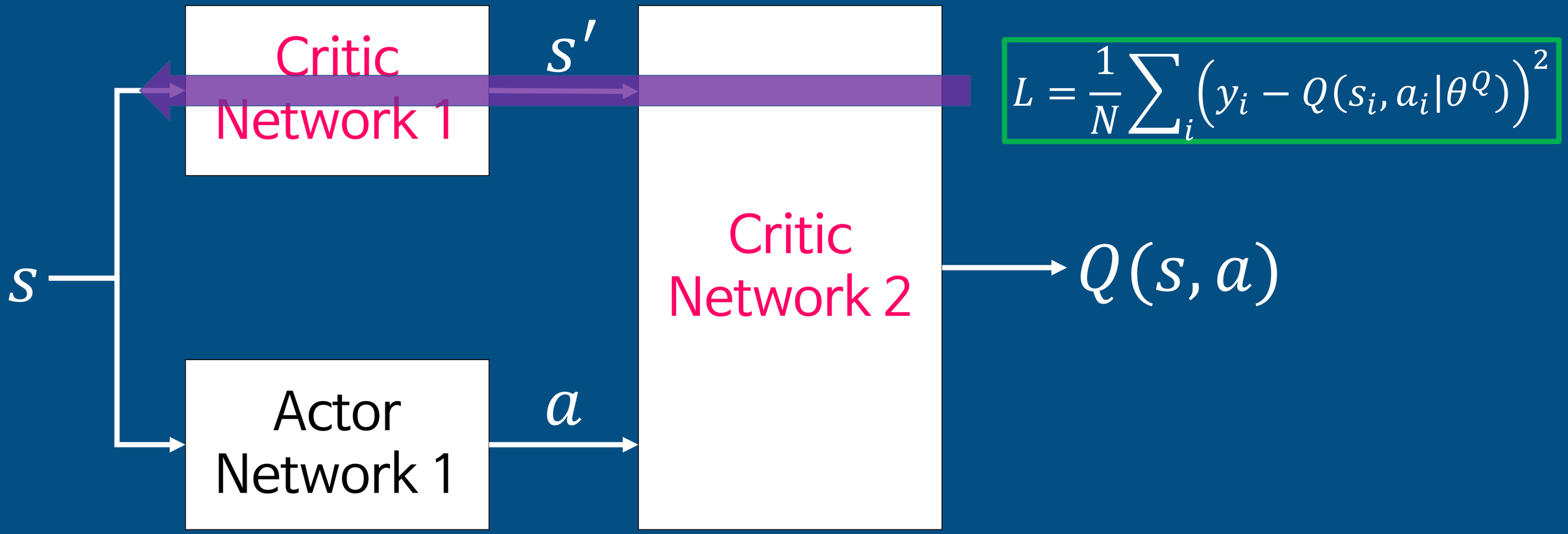
- Network structure



Algorithm

Continuous Control With Deep Reinforcement Learning, Lillicrap et al, 2015.

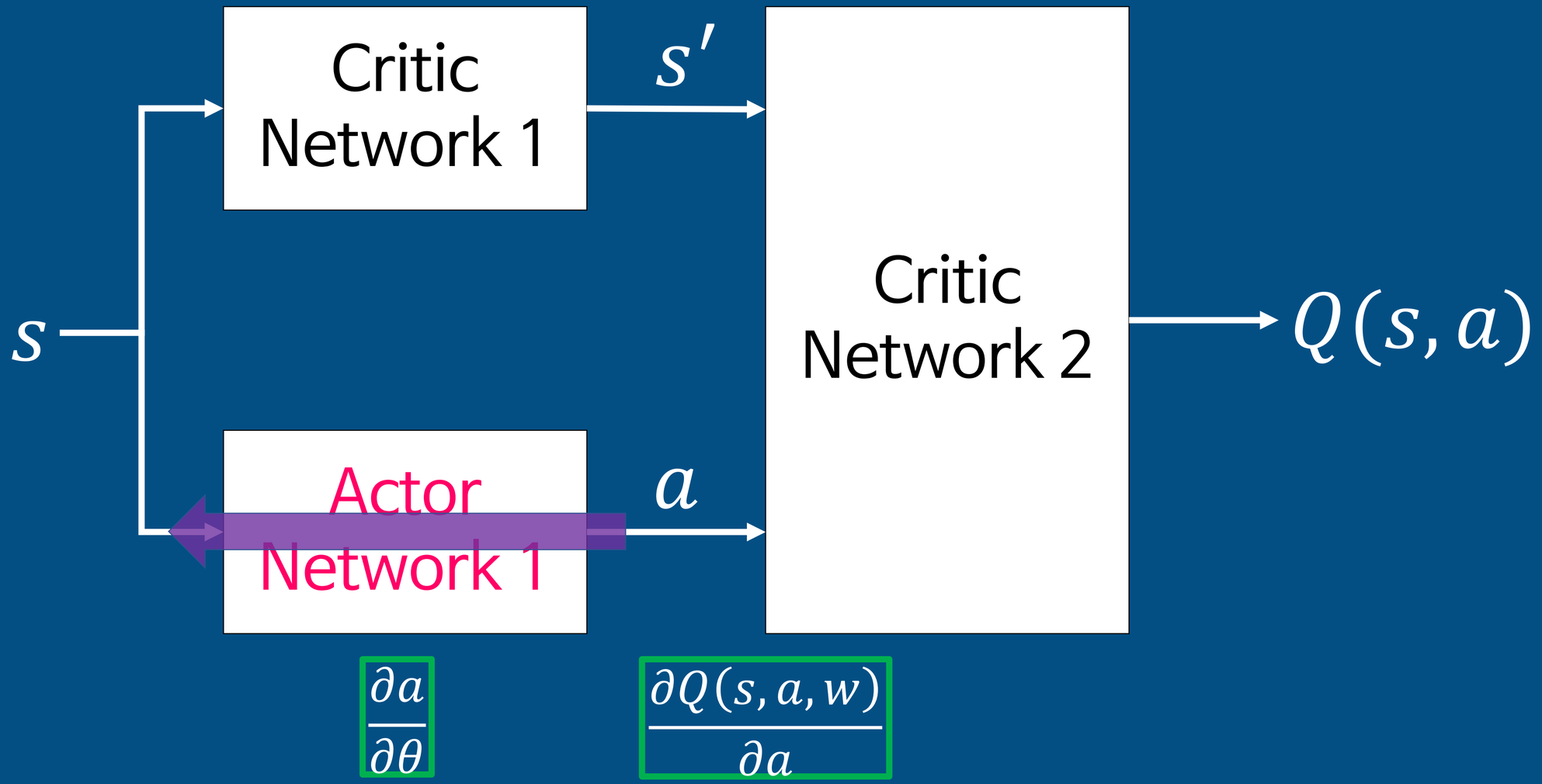
- Critic network



Algorithm

Continuous Control With Deep Reinforcement Learning, Lillicrap et al, 2015.

- Critic network



Algorithm

Continuous Control With Deep Reinforcement Learning, Lillicrap et al, 2015.

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:

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

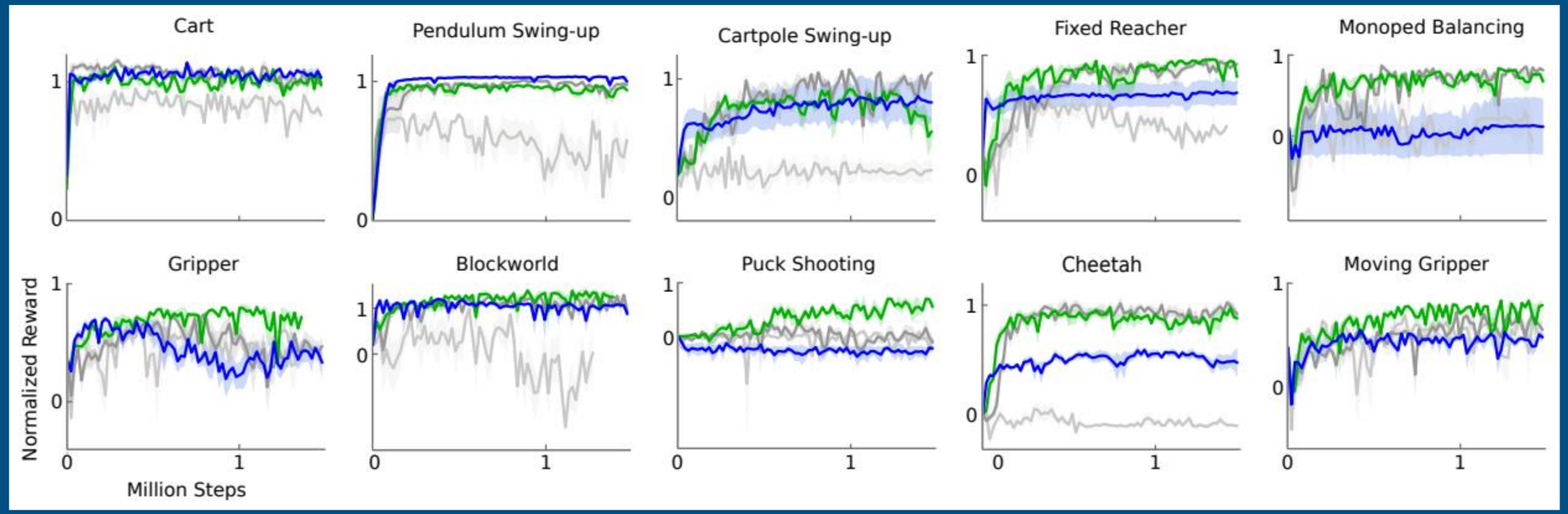
    Update the target networks:

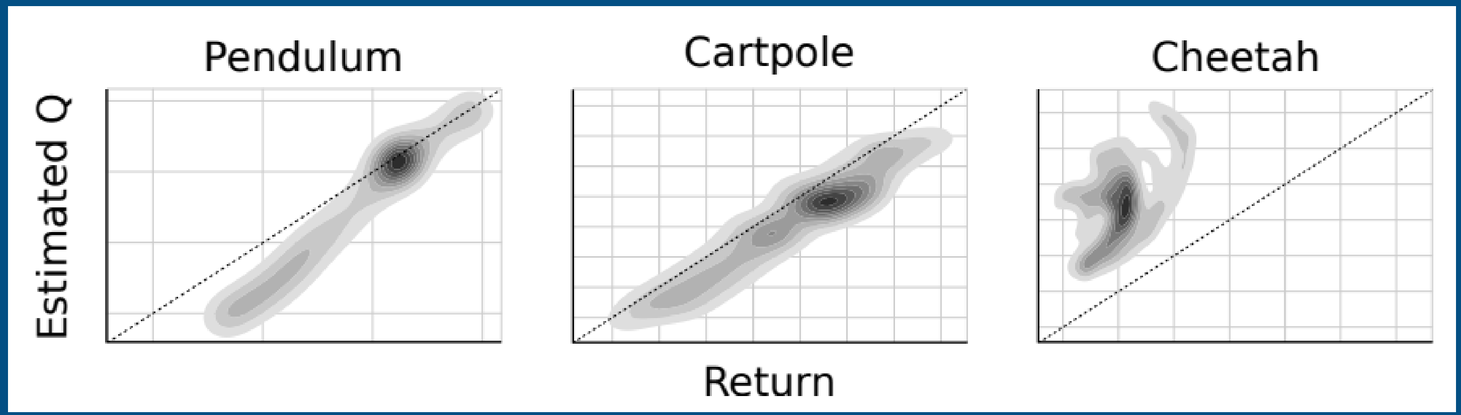
$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Results

Continuous Control With Deep Reinforcement Learning, Lillicrap et al, 2015.





Results

Continuous Control With Deep Reinforcement Learning, Lillicrap et al, 2015.

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

Conclusion

Continuous Control With Deep Reinforcement Learning, Lillicrap et al, 2015.

- The work combines insights from recent advances in deep learning and reinforcement learning, resulting in an algorithm that robustly solves challenging problems across a variety of domains with continuous action spaces, even when using raw pixels for observations.
- As with most reinforcement learning algorithms, the use of non-linear function approximators nullifies any convergence guarantees; however, our experimental results demonstrate that stable learning without the need for any modifications between environments.

Conclusion

Continuous Control With Deep Reinforcement Learning, Lillicrap et al, 2015.

- Interestingly, all of our experiments used substantially fewer steps of experience than was used by DQN learning to find solutions in the Atari domain.
- Most notably, as with most model-free reinforcement approaches, DDPG requires a large number of training episodes to find solutions. However, we believe that a robust model-free approach may be an important component of larger systems which may attack these limitations.

References

Continuous Control With Deep Reinforcement Learning, Lillicrap et al, 2015.

- https://nervanasystems.github.io/coach/components/agents/policy_optimization/ddpg.html
- https://youtu.be/h2WSVBAC1t4
- https://reinforcement-learning-kr.github.io/2018/06/27/2_dpg/
- https://reinforcement-learning-kr.github.io/2018/06/26/3_ddpg/

Thank you!