

A3C Paper Review

[쉽게 읽는 강화학습 논문 4화]

판요랩 – 노승은, 전민영

2019.02.24

Asynchronous Methods for Deep Reinforcement Learning

Volodymyr Mnih¹
Adrià Puigdomènech Badia¹
Mehdi Mirza^{1,2}
Alex Graves¹
Tim Harley¹
Timothy P. Lillicrap¹
David Silver¹
Koray Kavukcuoglu¹

VMNIH@GOOGLE.COM
ADRIAP@GOOGLE.COM
MIRZAMOM@IRO.UMONTREAL.CA
GRAVESA@GOOGLE.COM
THARLEY@GOOGLE.COM
COUNTZERO@GOOGLE.COM
DAVIDSILVER@GOOGLE.COM
KORAYK@GOOGLE.COM

¹ Google DeepMind

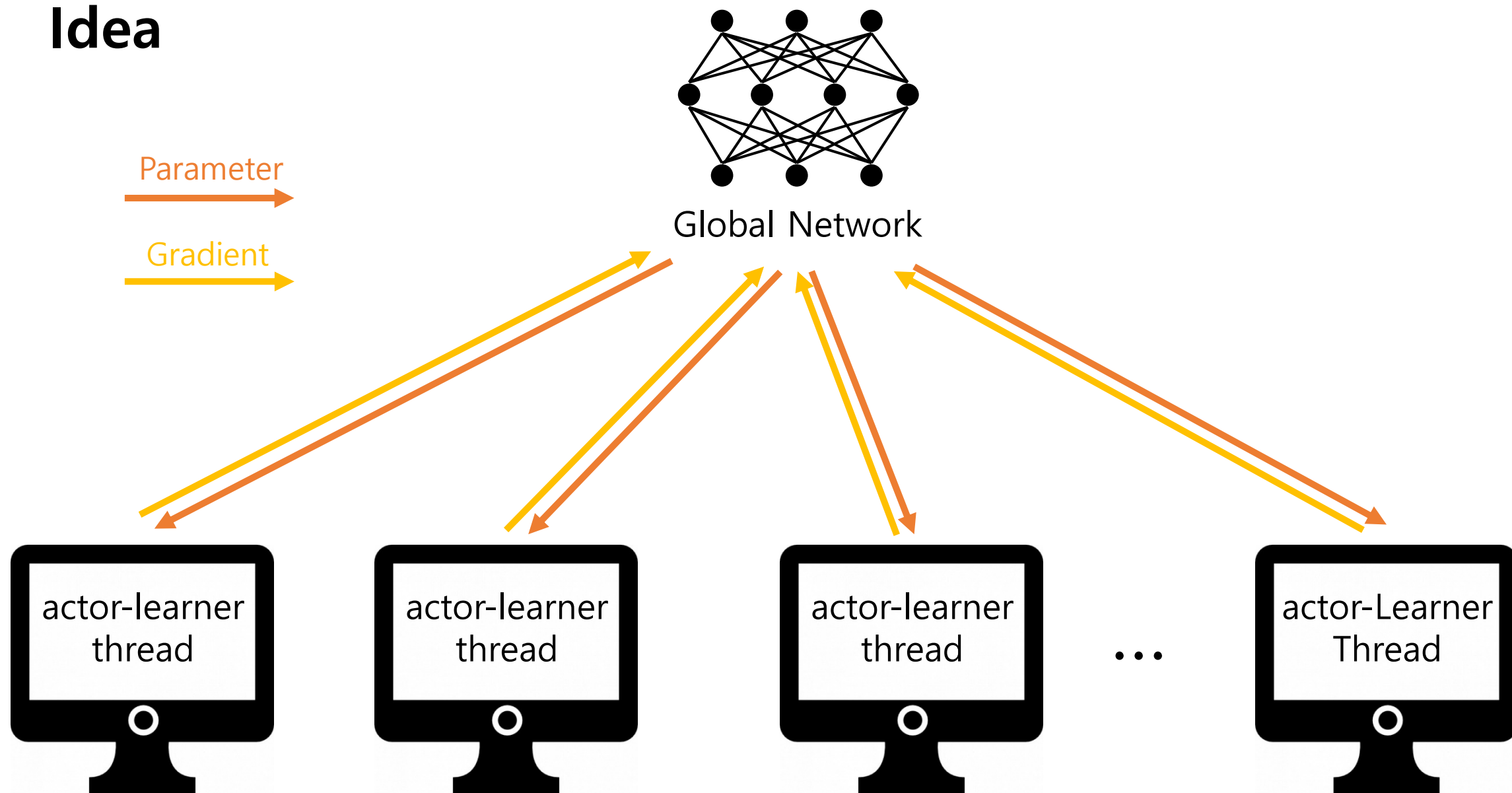
² Montreal Institute for Learning Algorithms (MILA), University of Montreal

Abstract

We propose a conceptually simple and lightweight framework for deep reinforcement learning that uses asynchronous gradient descent for optimization of deep neural network controllers. We present asynchronous variants of

line RL updates are strongly correlated. By storing the agent's data in an experience replay memory, the data can be batched (Riedmiller, 2005; Schulman et al., 2015a) or randomly sampled (Mnih et al., 2013; 2015; Van Hasselt et al., 2015) from different time-steps. Aggregating over memory in this way reduces non-stationarity and decorrelates updates, but at the same time limits the methods to

Idea



Contribution

1. 다양한 RL 알고리즘의 **scale-up** 방법 제시!
 - Off/on policy, Value/Policy based ... 모두에서 stable
 - Experience Replay 대신 병렬적 actor가 decorrelation 가능케 해줌.
 - GPU 대신 CPU Thread 사용
 - 심지어 Super Linear...
2. SOTA 갱신! (State-Of-The-Art)

논문에서 다루는 4가지 알고리즘

One-step
Q-learning

N-step
Q-learning

One-step
SARSA

Advantage-
Actor Critic

Q-learning 복습

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim U(D)} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right)^2 \right]$$

$$\nabla_{\theta_i} L(\theta_i) = \mathbb{E}_{s,a,r,s'} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right) \nabla_{\theta_i} Q(s, a; \theta_i) \right]$$

Experience Replay

Target Network

Asynchronous one-step Q-learning

Algorithm 1 Asynchronous one-step Q-learning - pseudocode for each actor-learner thread.

// Assume global shared θ , θ^- , and counter $T = 0$.

Initialize thread step counter $t \leftarrow 0$

Initialize target network weights $\theta^- \leftarrow \theta$

Initialize network gradients $d\theta \leftarrow 0$

Get initial state s

repeat

Take action a with ϵ -greedy policy based on $Q(s, a; \theta)$

Receive new state s' and reward r

$$y = \begin{cases} r & \text{for terminal } s' \\ r + \gamma \max_{a'} Q(s', a'; \theta^-) & \text{for non-terminal } s' \end{cases}$$

Accumulate gradients wrt θ : $d\theta \leftarrow d\theta + \frac{\partial(y - Q(s, a; \theta))^2}{\partial \theta}$

$s = s'$

$T \leftarrow T + 1$ and $t \leftarrow t + 1$

if $T \bmod I_{target} == 0$ **then**

Update the target network $\theta^- \leftarrow \theta$

end if

if $t \bmod I_{AsyncUpdate} == 0$ or s is terminal **then**

Perform asynchronous update of θ using $d\theta$.

Clear gradients $d\theta \leftarrow 0$.

end if

until $T > T_{max}$

요기서 Epsilon은 각 actor마다 다르게 설정할 수도 있음!

Asynchronous one-step SARSA

Algorithm 1 Asynchronous one-step Q-learning - pseudocode for each actor-learner thread.

// Assume global shared θ , θ^- , and counter $T = 0$.

Initialize thread step counter $t \leftarrow 0$

Initialize target network weights $\theta^- \leftarrow \theta$

Initialize network gradients $d\theta \leftarrow 0$

Get initial state s

repeat

Take action a with ϵ -greedy policy based on $Q(s, a; \theta)$

Receive new state s' and reward r

$$y = \begin{cases} r & \text{for terminal } s' \\ r + \gamma \max_{a'} Q(s', a'; \theta^-) & \text{for non-terminal } s' \end{cases}$$

Accumulate gradients wrt θ : $d\theta \leftarrow d\theta + \frac{\partial(y - Q(s, a; \theta))^2}{\partial \theta}$

$s = s'$

$T \leftarrow T + 1$ and $t \leftarrow t + 1$

if $T \bmod I_{target} == 0$ **then**

Update the target network $\theta^- \leftarrow \theta$

end if

if $t \bmod I_{AsyncUpdate} == 0$ or s is terminal **then**

Perform asynchronous update of θ using $d\theta$.

Clear gradients $d\theta \leftarrow 0$.

end if

until $T > T_{max}$

Q의 target value 만

$$r + \gamma Q(s', a'; \theta')$$

로 바꾸면 됨!

Policy Gradient 복습 1

Lecture 7: Policy Gradient

└ Actor-Critic Policy Gradient

└ Snake example

Summary of Policy Gradient Algorithms

- The **policy gradient** has many equivalent forms

$$\begin{aligned}\nabla_{\theta} J(\theta) &= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) \quad v_t] && \text{REINFORCE} \\ &= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) \quad Q^w(s, a)] && \text{Q Actor-Critic} \\ &= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) \quad A^w(s, a)] && \text{Advantage Actor-Critic} \\ &= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) \quad \delta] && \text{TD Actor-Critic} \\ &= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) \quad \delta e] && \text{TD}(\lambda) \text{ Actor-Critic} \\ G_{\theta}^{-1} \nabla_{\theta} J(\theta) &= w && \text{Natural Actor-Critic}\end{aligned}$$

요놈을 쓸 거임

- Each leads a stochastic gradient ascent algorithm
- Critic uses **policy evaluation** (e.g. MC or TD learning) to estimate $Q^{\pi}(s, a)$, $A^{\pi}(s, a)$ or $V^{\pi}(s)$

Policy Gradient 복습 2

Lecture 7: Policy Gradient

└ Actor-Critic Policy Gradient

└ Advantage Function Critic

Estimating the Advantage Function (2)

- For the true value function $V^{\pi_{\theta}}(s)$, the TD error $\delta^{\pi_{\theta}}$

$$\delta^{\pi_{\theta}} = r + \gamma V^{\pi_{\theta}}(s') - V^{\pi_{\theta}}(s)$$

- is an unbiased estimate of the advantage function

$$\begin{aligned}\mathbb{E}_{\pi_{\theta}} [\delta^{\pi_{\theta}} | s, a] &= \mathbb{E}_{\pi_{\theta}} [r + \gamma V^{\pi_{\theta}}(s') | s, a] - V^{\pi_{\theta}}(s) \\ &= Q^{\pi_{\theta}}(s, a) - V^{\pi_{\theta}}(s) \\ &= A^{\pi_{\theta}}(s, a)\end{aligned}$$

- So we can use the TD error to compute the policy gradient

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) \delta^{\pi_{\theta}}]$$

- In practice we can use an approximate TD error

$$\delta_v = r + \gamma V_v(s') - V_v(s)$$

- This approach only requires one set of critic parameters v

Policy Gradient 복습 3

Theorem

*For any differentiable policy $\pi_\theta(s, a)$,
for any of the policy objective functions $J = J_1, J_{avR}$, or $\frac{1}{1-\gamma} J_{avV}$,
the policy gradient is*

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} [\nabla_\theta \log \pi_\theta(s, a) Q^{\pi_\theta}(s, a)]$$

- A good baseline is the state value function $B(s) = V^{\pi_\theta}(s)$
- So we can rewrite the policy gradient using the **advantage function** $A^{\pi_\theta}(s, a)$

$$A^{\pi_\theta}(s, a) = Q^{\pi_\theta}(s, a) - V^{\pi_\theta}(s)$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} [\nabla_\theta \log \pi_\theta(s, a) A^{\pi_\theta}(s, a)]$$

Asynchronous advantage actor-critic (A3C)

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// Assume global shared parameter vectors θ and θ_v and global shared counter $T = 0$

// Assume thread-specific parameter vectors θ' and θ'_v

Initialize thread step counter $t \leftarrow 1$

repeat

Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.

Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$

$t_{start} = t$

Get state s_t

repeat

Perform a_t according to policy $\pi(a_t|s_t; \theta')$

Receive reward r_t and new state s_{t+1}

$t \leftarrow t + 1$

$T \leftarrow T + 1$

until terminal s_t **or** $t - t_{start} == t_{max}$

$R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t // \text{ Bootstrap from last state} \end{cases}$

for $i \in \{t - 1, \dots, t_{start}\}$ **do**

$R \leftarrow r_i + \gamma R$

Accumulate gradients wrt θ' : $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i; \theta')(R - V(s_i; \theta'_v))$

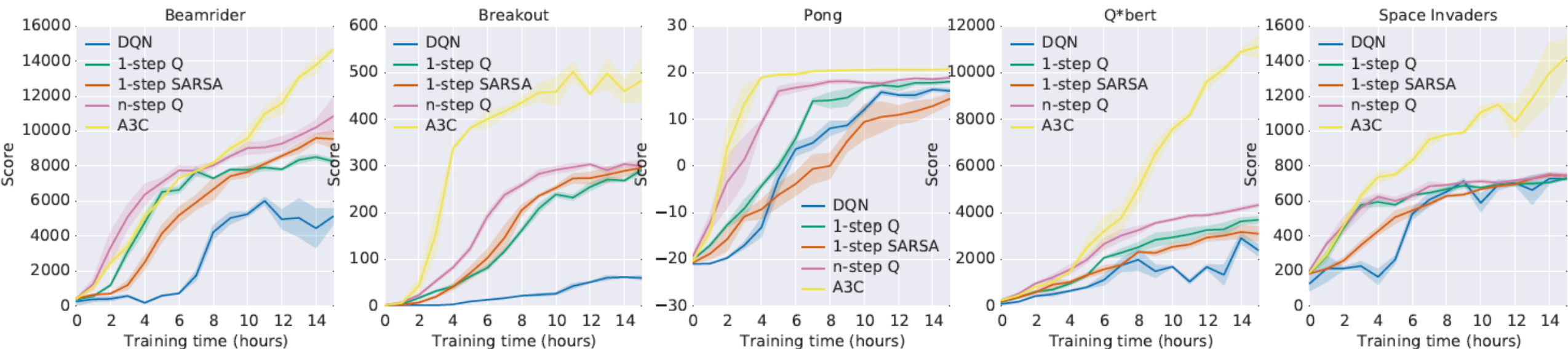
Accumulate gradients wrt θ'_v : $d\theta_v \leftarrow d\theta_v + \partial (R - V(s_i; \theta'_v))^2 / \partial \theta'_v$

end for

Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.

until $T > T_{max}$

SOTA – Asynchronous vs. DQN

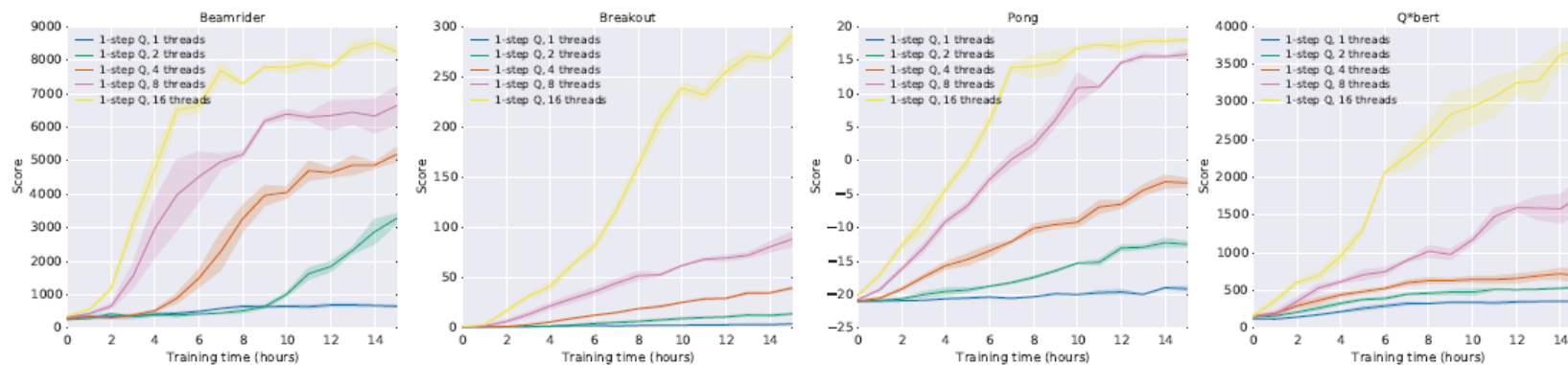


SOTA – A3C vs. Others

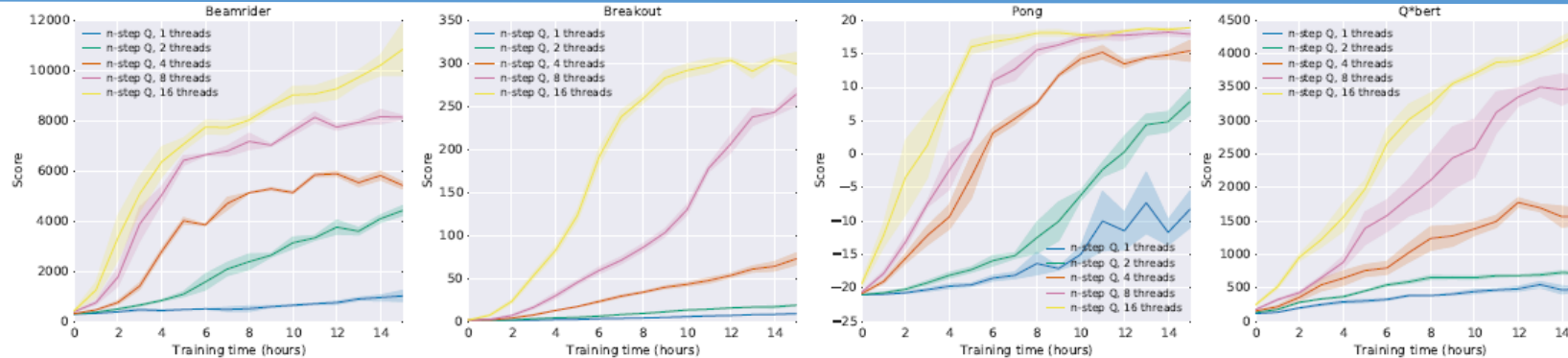
Method	Training Time	Mean	Median
DQN	8 days on GPU	121.9%	47.5%
Gorila	4 days, 100 machines	215.2%	71.3%
D-DQN	8 days on GPU	332.9%	110.9%
Dueling D-DQN	8 days on GPU	343.8%	117.1%
Prioritized DQN	8 days on GPU	463.6%	127.6%
A3C, FF	1 day on CPU	344.1%	68.2%
A3C, FF	4 days on CPU	496.8%	116.6%
A3C, LSTM	4 days on CPU	623.0%	112.6%

시간에 따른 성능비교 (알고리즘별, thread 개수 별)

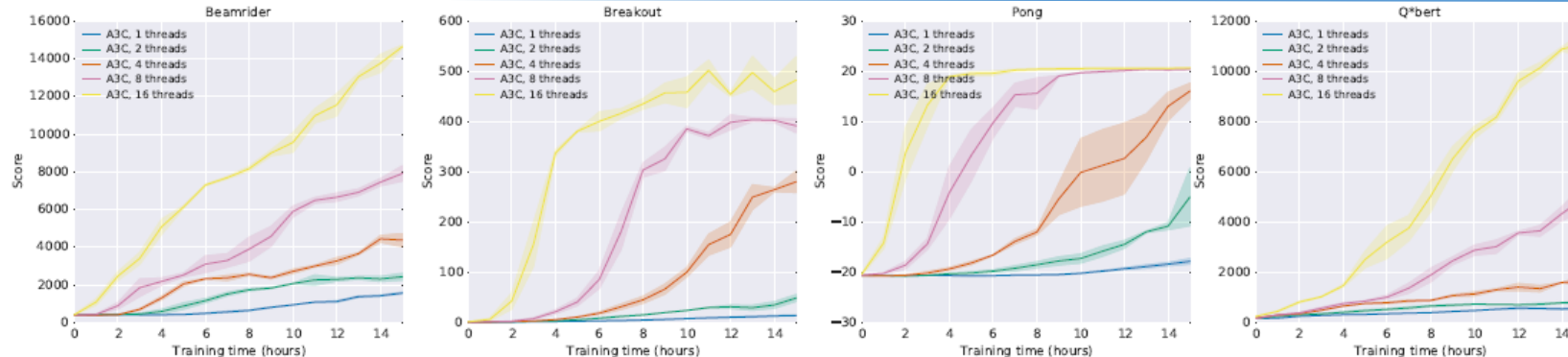
1-step Q



n-step Q



A3C

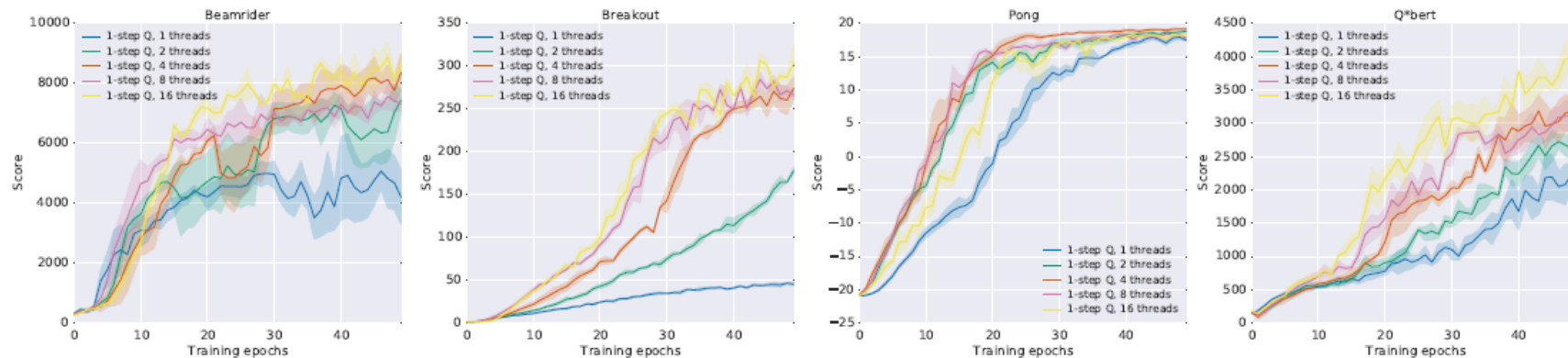


Scalability

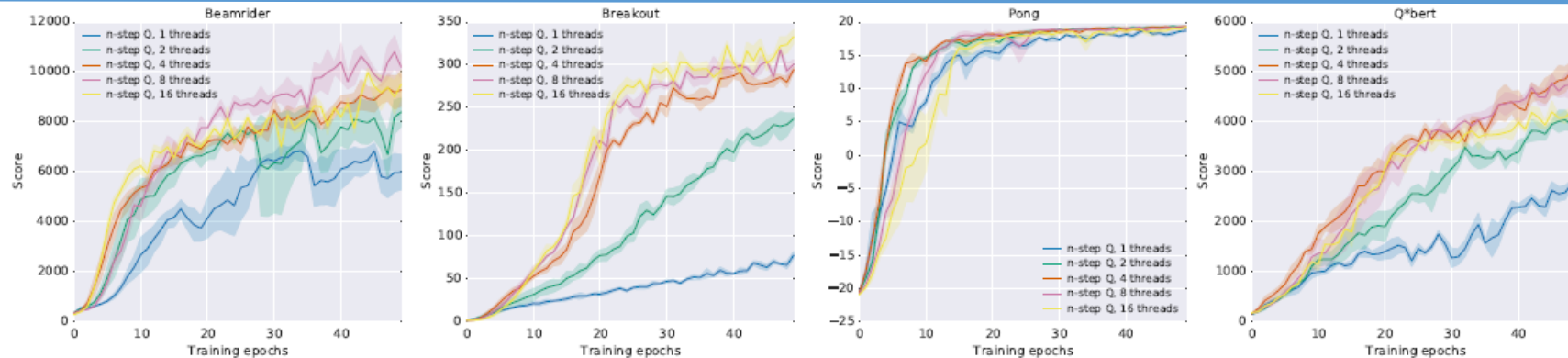
	Number of threads				
Method	1	2	4	8	16
1-step Q	1.0	3.0	6.3	13.3	24.1
1-step SARSA	1.0	2.8	5.9	13.1	22.1
n-step Q	1.0	2.7	5.9	10.7	17.2
A3C	1.0	2.1	3.7	6.9	12.5

Data Efficiency 비교 (알고리즘별, thread 개수 별)

1-step Q



n-step Q



A3C

