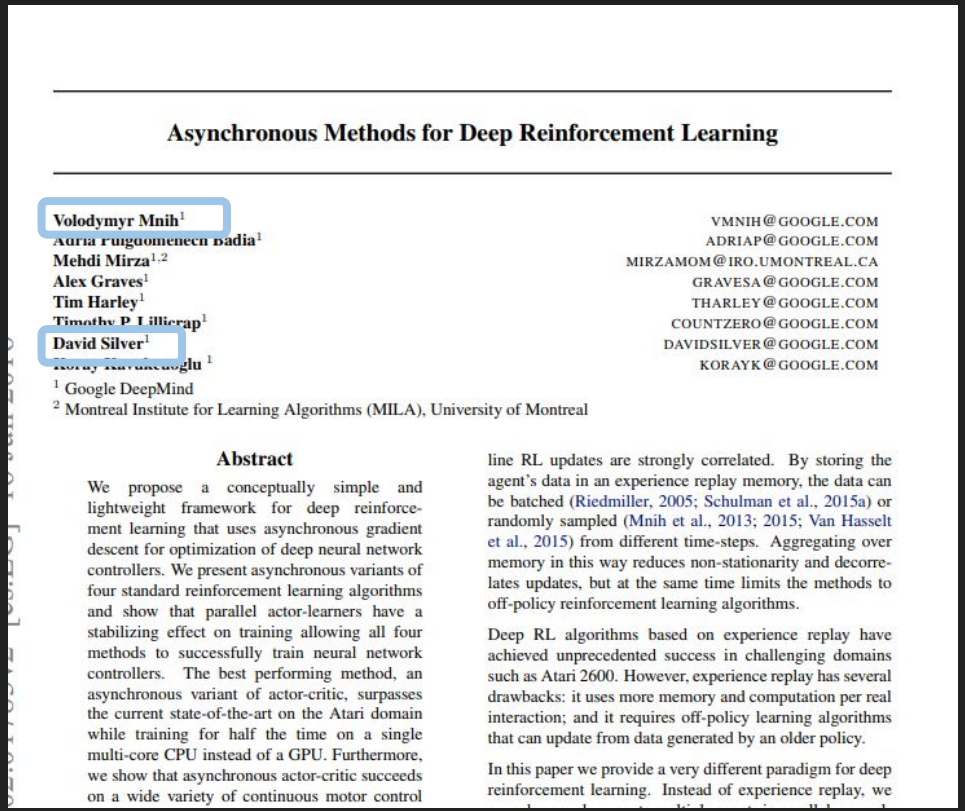
A3C

A3C



Asynchronous Methods for Deep Reinforcement Learning

Volodymyr Mnih[1] VMNIH@GOOGLE.COM
Adrià Puigdomènech Badia[1] ADRIAP@GOOGLE.COM
Mehdi Mirza[1,2] MIRZAMOM@IRO.UMONTREAL.CA
Alex Graves[1] GRAVESA@GOOGLE.COM
Tim Harley[1] THARLEY@GOOGLE.COM
Timothy P. Lillicrap[1] COUNTZERO@GOOGLE.COM
David Silver[1] DAVIDSILVER@GOOGLE.COM
Koray Kavukcuoglu[1] 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 four standard reinforcement learning algorithms and show that parallel actor-learners have a stabilizing effect on training allowing all four methods to successfully train neural network controllers. The best performing method, an asynchronous variant of actor-critic, surpasses the current state-of-the-art on the Atari domain while training for half the time on a single multi-core CPU instead of a GPU. Furthermore, we show that asynchronous actor-critic succeeds on a wide variety of continuous motor control

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 off-policy reinforcement learning algorithms.

Deep RL algorithms based on experience replay have achieved unprecedented success in challenging domains such as Atari 2600. However, experience replay has several drawbacks: it uses more memory and computation per real interaction; and it requires off-policy learning algorithms that can update from data generated by an older policy.

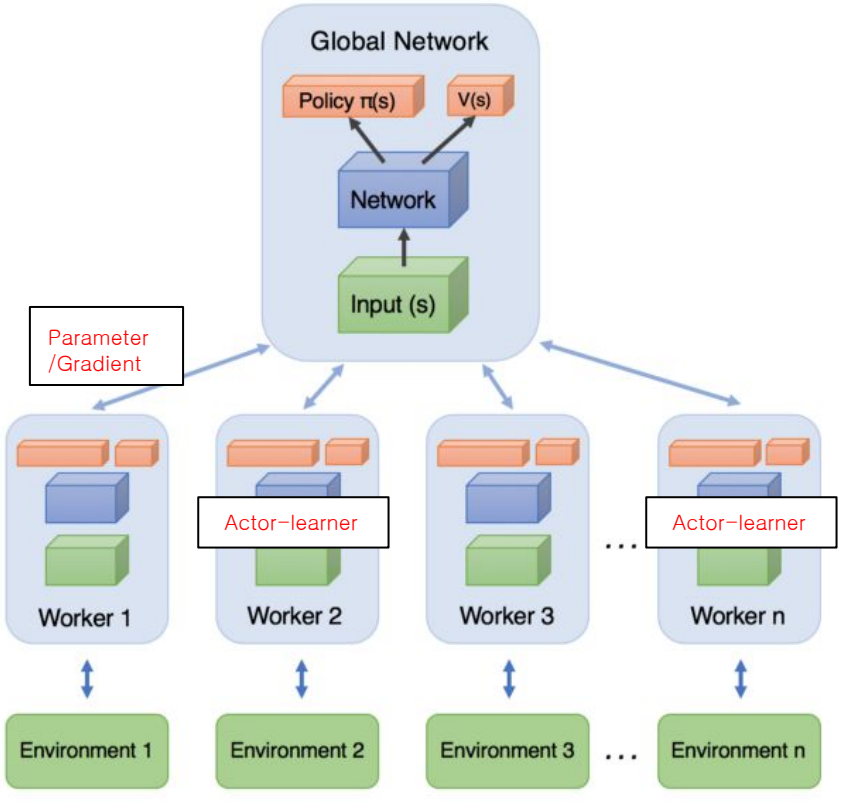
In this paper we provide a very different paradigm for deep reinforcement learning. Instead of experience replay, we

Asynchronous Methods for Deep Reinforcement Learning
https://arxiv.org › cs ▾
by V Mnih - 2016 - Cited by 359 - Related articles
Feb 4, 2016 - Asynchronous Methods for Deep Reinforcement Learning. ... The best performing method, an asynchronous variant of actor-critic, surpasses the current state-of-the-art on the Atari domain while training for half the time on a single multi-core CPU instead of a GPU.
You've visited this page 3 times. Last visit: 7/11/19

Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih **Koray Kavukcuoglu** **David Silver** **Alex Graves** **Ioannis Antonoglou**
Daan Wierstra **Martin Riedmiller**
DeepMind Technologies

A3C 개요



- Worker : Actor-Learner Thread
- Actor-learner : 경험을 쌓고, 학습하는것
- 그 전까지는 한개의 Actor-Learner

- A3C(Asynchronous Methods 4 DRL)

- 경험을 모으는것과 Gradient를 update하는 것이 비동기적
- Gradient를 구하는 알고리즘은 다양하게 적용 가능(여기서는 4가지 방법을 제안)

- 예) step를 5번 마다 업데이트
  . 1번 worker가 경험을 5번 모음
  . 5개의 경험으로 gradient 계산 (Network의 변화량)
  . 중앙 Global Network 에 보냄
  . 각각의 thread들이 비동기적으로 업데이트 함
  . Global을 다시 각각의 Woker에 보냄 (Asynchronous하게)

A3C 개요

1. Scale up 방법을 제시
   a. 모든 종류의 RL 알고리즘 에 확장가능
      (Off/On/Value/Policy Based, Actor-Critic 등)
   b. DQN의 Experience Replay 대신 병렬적 Actor가 decorrelation 해줌
   c. GPU대신 CPU thread
   d. Super linear
   e. 그리고 SOTA 됨(A3C로)

- Experience Replay : RL은 Data가 순서대로 되어 있어 Correlation 심해 학습이 힘듬. 그래서 경험을 버퍼에 쌓아놓고 랜덤하게 가져와서 Data Correlation을 해소하는 E.R을 제안(DQN)

4가지 Method

어떤 알고리즘도 scaleup 할수 있으므로 4개를 이야기함

Asynchronous one-step Q-learning: Pseudocode for our variant of Q-learning, which we call Asynchronous one-step Q-learning, is shown in Algorithm 1. Each thread interacts with its own copy of the environment and at each step computes a gradient of the Q-learning loss. We use

DQN

Asynchronous n-step Q-learning: Pseudocode for our variant of multi-step Q-learning is shown in Supplementary Algorithm S2. The algorithm is somewhat unusual because it operates in the forward view by explicitly computing n-step returns, as opposed to the more common backward view used by techniques like eligibility traces (Sutton & Barto, 1998). We found that using the forward view is easier when training neural networks with momentum-based methods and backpropagation through time. In order to compute a single update, the algorithm first selects actions using its exploration policy for up to t_{max} steps or until a

Asynchronous one-step Sarsa: The asynchronous one-step Sarsa algorithm is the same as asynchronous one-step Q-learning as given in Algorithm 1 except that it uses a different target value for $Q(s, a)$. The target value used by one-step Sarsa is $r + \gamma Q(s', a'; \theta^-)$ where a' is the action taken in state s' (Rummery & Niranjan, 1994; Sutton & Barto, 1998). We again use a target network and updates accumulated over multiple timesteps to stabilize learning.

Sarsa (DQN on policy version)

Asynchronous advantage actor-critic: The algorithm, which we call asynchronous advantage actor-critic (A3C), maintains a policy $\pi(a_t|s_t; \theta)$ and an estimate of the value function $V(s_t; \theta_v)$. Like our variant of n-step Q-learning, our variant of actor-critic also operates in the forward view and uses the same mix of n-step returns to update both the policy and the value-function. The policy and the value function are updated after every t_{max} actions or when a terminal state is reached. The update performed by the algorithm can be seen as $\nabla_{\theta'} \log \pi(a_t|s_t; \theta')A(s_t, a_t; \theta, \theta_v)$ where $A(s_t, a_t; \theta, \theta_v)$ is an estimate of the advantage function given by $\sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k}; \theta_v) - V(s_t; \theta_v)$, where k can vary from state to state and is upper-bounded

Actor Critic

Asynchronous one step Q-Learning

- Optimal Q-values should obey Bellman equation

$$Q^*(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q(s',a')^* \mid s,a\right]$$

- Treat right-hand side $r + \gamma \max_{a'} Q(s',a',\mathbf{w})$ as a target
- Minimise MSE loss by stochastic gradient descent

$$I = \left(r + \gamma \max_{a} Q(s',a',\mathbf{w}) - Q(s,a,\mathbf{w})\right)^2$$

Loss

Q Value Fuction

- Q-value function gives expected total reward
  - from state s and action a
  - under policy π
  - with discount factor γ

$$Q^{\pi}(s,a) = \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s,a\right]$$

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 $T \mod I_{target} == 0$ **then**
    Update the target network $\theta^- \leftarrow \theta$
  end if
  if $t \mod I_{AsyncUpdate} == 0$ or s is terminal **then**
    Perform asynchronous update of θ using $d\theta$.
    Clear gradients $d\theta \leftarrow 0$.
  end if
until $T > T_{max}$

- t : global network update용 count
- θ^- : Target network
- $d\theta$: gradient
- repeat : 각각의 actor-learnet에서 실행
- e-greedy policy : exploit exploration 특정 기준에 따라 랜덤? policy 따라 e-greedy에 따라 행동함
- state와 reward를 받는다
- y를 계산. 끝이면 r, 아니면, reward, 디스카운트팩터,맥스Q
- 방금있었던 한 step에 대한 gradient계산를 계속 더함(축적)
- T는 target update용, t는 Global용
- Target을 global로 보냄
- Global network는 $d\theta$ 빼줌

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 $T \mod I_{target} == 0$ **then**
    Update the target network $\theta^- \leftarrow \theta$
  end if
  if $t \mod I_{AsyncUpdate} == 0$ or s is terminal **then**
    Perform asynchronous update of θ using $d\theta$.
    Clear gradients $d\theta \leftarrow 0$.
  end if
until $T > T_{max}$

Asynchronous one-step Sarsa: The asynchronous one-step Sarsa algorithm is the same as asynchronous one-step Q-learning as given in Algorithm 1 except that it uses a different target value for $Q(s, a)$. The target value used by one-step Sarsa is $r + \gamma Q(s', a'; \theta^-)$ where a' is the action taken in state s' (Rummery & Niranjan, 1994; Sutton & Barto, 1998). We again use a target network and updates accumulated over multiple timesteps to stabilize learning.

- Max Q를 감마Q로 바꿈
- Max Q는 bellman Equiation을 사용한것
- 그냥 Policy에서 선택한 다음 스텝의 Action이 A'이고, 그 Q value를 update함
- 내 policy가 어떤 행동을 하는지에 쓰이니깐 on Policy임(Sarsa)

Asynchronous Advanced Actor Critic

Reducing Variance Using a Baseline

- We subtract a baseline function $B(s)$ from the policy gradient
- This can reduce variance, without changing expectation

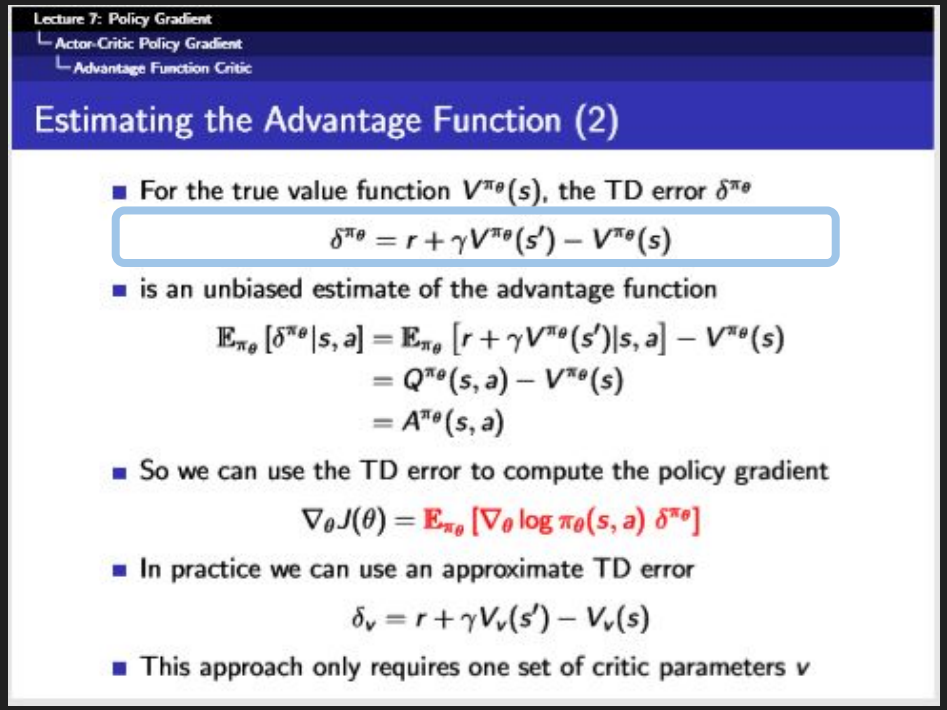
$$\begin{aligned}\mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)B(s)] &= \sum_{s\in\mathcal{S}} d^{\pi_\theta}(s) \sum_a \nabla_\theta \pi_\theta(s,a)B(s) \\ &= \sum_{s\in\mathcal{S}} d^{\pi_\theta} B(s) \nabla_\theta \sum_{a\in\mathcal{A}} \pi_\theta(s,a) \\ &= 0\end{aligned}$$

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

$$\begin{aligned}A^{\pi_\theta}(s,a) &= Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s) \\ \nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\ A^{\pi_\theta}(s,a)]\end{aligned}$$

- Q-V는 Variance 가 줄어듬
- 그것을 Advantage라 한다

Asynchronous Advantage Actor Critic



- Advantage : Q-V
- Q, V, Policy랑 3개를 훈련해야함
- TD Target이 Advanced랑 같다
- 델타가 Value 펑션의 TD Error다
- Value function 하나만 만들면 이게 Advantage랑 같다

Asynchronous Advantage Actor Critic

Summary of Policy Gradient Algorithms

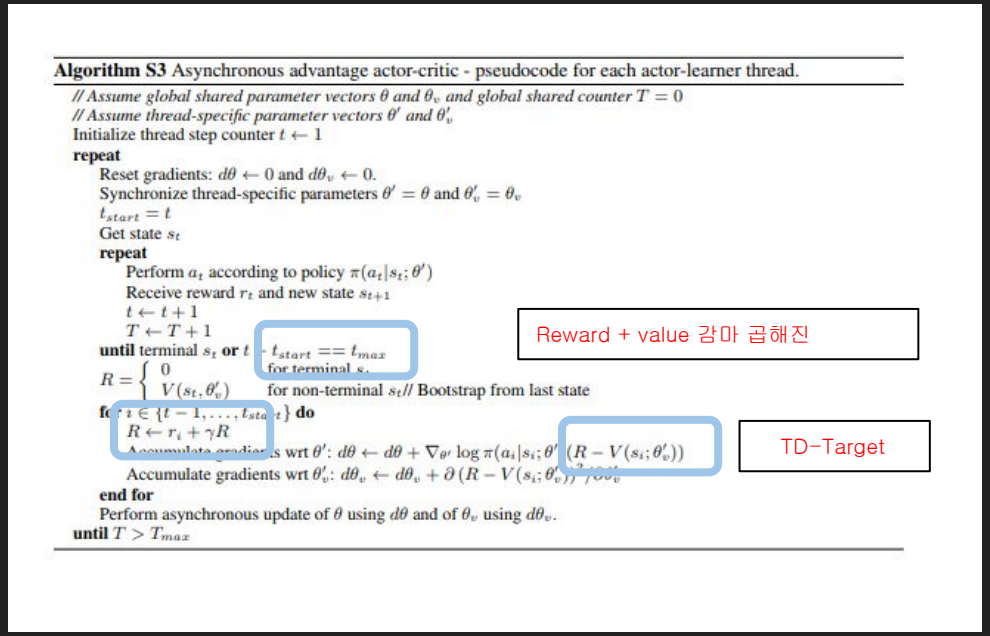
- The policy gradient has many equivalent forms

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

- 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 $V^\pi(s)$

- A3c에서 TD-actor-Critic을 사용

Asynchronous Advantage Actor Critic



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

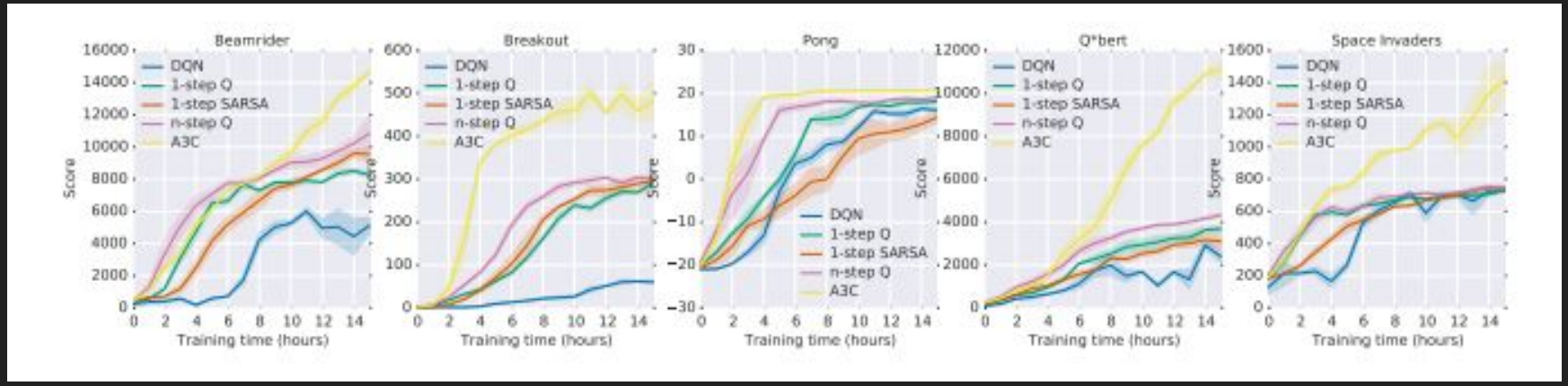
// Assume global shared parameter vectors θ and θ_v and global shared counter $T = 0$
// Assume thread-specific parameter vectors θ' and θ'_v
Initialize thread step counter $t \leftarrow 1$
repeat
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
  Synchronize thread-specific parameters $\theta' = \theta$ 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 $i \in \{t-1,\ldots,t_{start}\}$ **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 $T > T_{max}$

Reward + value 감마 곱해진

TD-Target

- A, A, A
- Actor 1개의 Network가 훈련필요
- Critic 1개의 Network가 훈련필요,
- Global도 2개가 있음
- 예를 들어 5번동안 state와 reward를 받는다.
- t – t start == tmax(예를 들어 5)
- R에다가 V를 넣는다(Bootstrap 자른다(?))
-
- R에서 거꾸로 대입.(TD target)
- R-V : advantage
- 이것을 TD-Target방법으로 함.
- r t-1 + 감마(value펑션)
- r t-2 + 감마(r t-1 + 감마(v))
-
- R에 계속 다른 다른값이 들어가는 형태
- 2개의 network의 gradient 값 계산(5 step동안 축적)
- Global network update

Asynchronous Advantage Actor Critic



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%

Table 1. Mean and median human-normalized scores on 57 Atari games using the human starts evaluation metric. Supplementary Table SS3 shows the raw scores for all games.

	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

Super linear