RL PAPER 3rd

APE-X

Distributed Prioritized Experience Replay

Published as a conference paper at **ICLR 2018**

2020.12.07 Mon

Jungyeon Lee

DISTRIBUTED PRIORITIZED EXPERIENCE REPLAY

Dan Horgan
DeepMind
horgan@google.com

John Quan
DeepMind
johnquan@google.com

David Budden
DeepMind
budden@google.com

Gabriel Barth-Maron
DeepMind
gabrielbm@google.com

Matteo Hessel
DeepMind
mtthss@google.com

Hado van Hasselt
DeepMind
hado@google.com

David Silver
DeepMind
davidsilver@google.com

순서

INTRODUCTION

BACKGROUND

- 3 keywords

OUR CONTRIBUTION

- Structure
- pseudo code
- APE-X DQN
- APE-X DPG

EXPERIMENTS

- Atari
- Continuous control

ANALYSIS

- Actor 수에 대한 분석
- Replay memory에 대한 분석
- Scalability의 다른 요소들에 대한 분석

CONCLUSION

APPENDIX

Abstract

a **distributed** architecture for DRL at scale, that **enables agents to learn effectively from orders of magnitude more data** than previously possible.

- decouple acting from learning:
    - the Actors
        - interact with **their own** instances of the environment by selecting actions according to **a shared neural network**
        - accumulate the resulting experience **in a shared experience replay memory**
    - the Learner
        - replay samples of experience and updates the neural network
- rely on **prioritized experience replay** to focus only on the most significant data generated by the actors.
- substantially improve the state of the art on the Arcade Learning Environment, achieving better final performance in a fraction of the wall-clock training time.

Introduction

이 연구에 대해 빠르게 소개하겠습니다

딥러닝의 트렌드

- 더 파워풀한 모델 + 더 큰 데이터 셋 = 더 좋은 결과
- effective use of greater computational resource 가 중요한 성공 요인

어떻게 scale up 할 것인가

- TF 프레임워크에 의한 scale up 이 아닌,
- **싱글 머신**에서 DRL 자체로 컴퓨팅을 개선 시킬 수 있을까?

이 연구에서는
Data Generation/Selection

- 기존: Gradient들을 병렬 계산해서 파라미터들을 빨리 업데이트

1. **distribute the generation**
2. **selection of experience data**

- 기존 방법들의 대체제
- 2가지 방법을 결합할 수 있지만 (1) 데이터 생산에 초점

실험

- DQN(Deep Q-Networks)
- DDPG(Deep Deterministic Policy Gradient)
- Arcade Learning Env
- Continuous control tasks
- 하이퍼파라미터 튜닝 안하고 SOTA

분석

- how prioritization affects performance as we increase the number of data-generating workers
- replay capacity
- recency of the experience
- use of different data-generating policies for different workers

Background

3가지 키워드로 보는 배경 지식

Distributed SGD

- @ 지도학습 학습시간을 가속하기 위해 흔히 쓰는 방법
- 그라디언트 계산을 병렬화
- 파라미터 업데이트는 동시성/비동시성 둘 다 가능
- 비동시성 파라미터 업데이트+분산 데이터 생산 : 멀티 쓰레드로 성공적이었음
- GA3C/PAAC 등등

Distributed Importance Sampling

- 학습 빠르게 하기
- IS로 variance ▼ bias ▲
- 수렴 속도는 증가
- gradient의 L2 norm에 비례하여 샘플링 → @ 지도학습 성공적

1. Rank samples according to their *latest known loss value*
2. Make the sampling probability *a function of the rank* rather than of the loss itsel

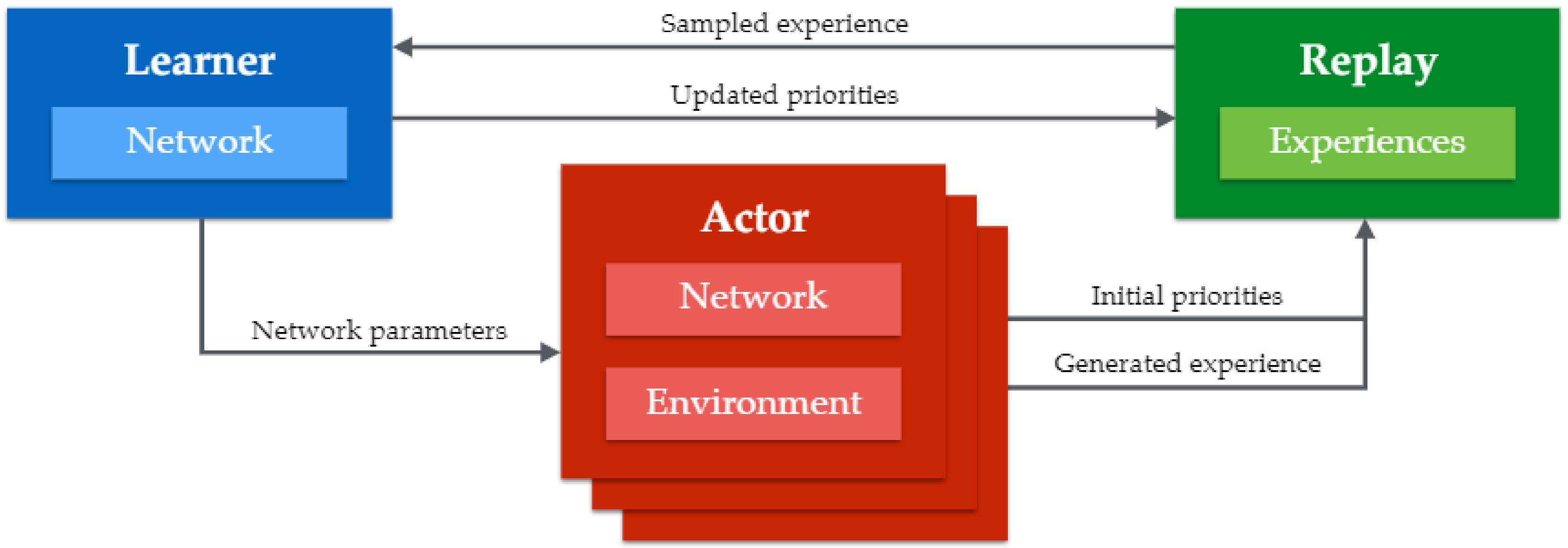
Prioritized Experience Replay

- **Prioritized :** 데이터 효율성 증대 (training NN이 SGD 알고리즘을 사용할 때 효율적)
- **Experience Replay** : 오버피팅을 방지
- 좀 더 *general*한 biased sampling (가장 surprising한 경험을 배우는 데에 초점)
- Rainbow에서 봤듯이 그 효과는 대단!

Distributed PER

EXTEND PER TO THE DISTRIBUTED SETTING

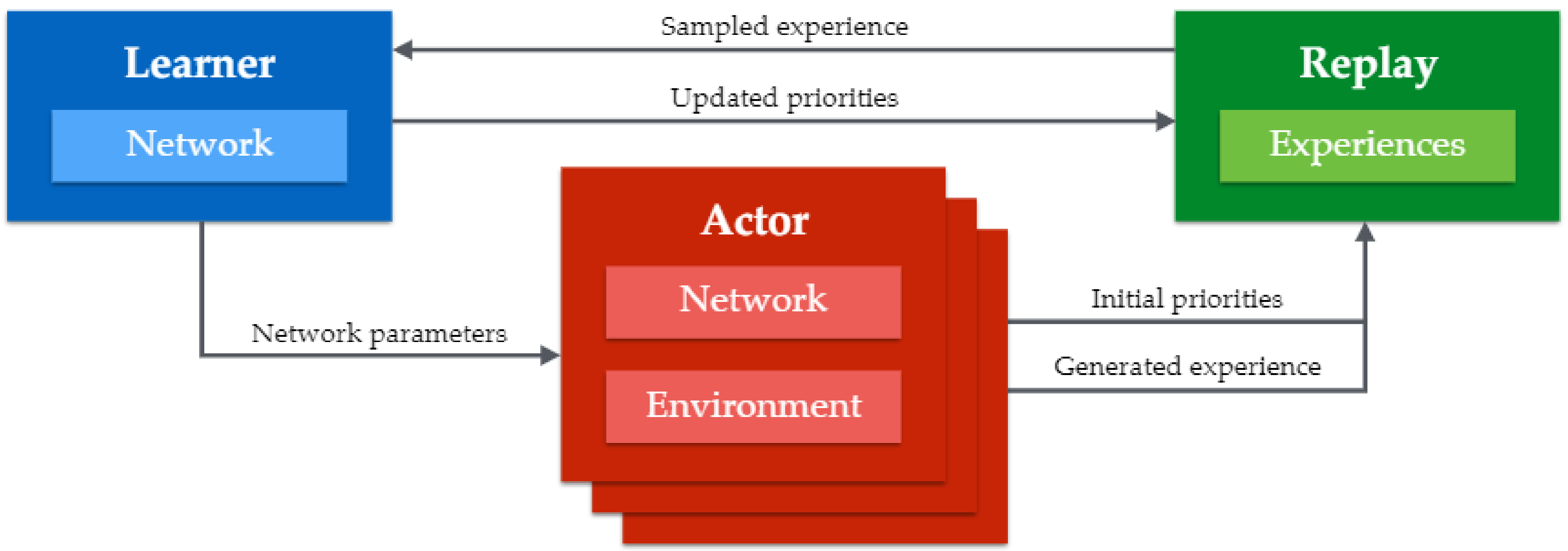
- @ Gorila
  - **acting** :
    - stepping through an environment
    - evaluating a policy implemented as a deep neural network
    - storing the observed data in a replay memory
  - **learning**:
    - sampling batches of data from the memory to update the policy parameters
- **@ APE-X**
  - actor N개 : CPU
  - learner 1개 : GPU



Distributed PER

EXTEND PER TO THE DISTRIBUTED SETTING

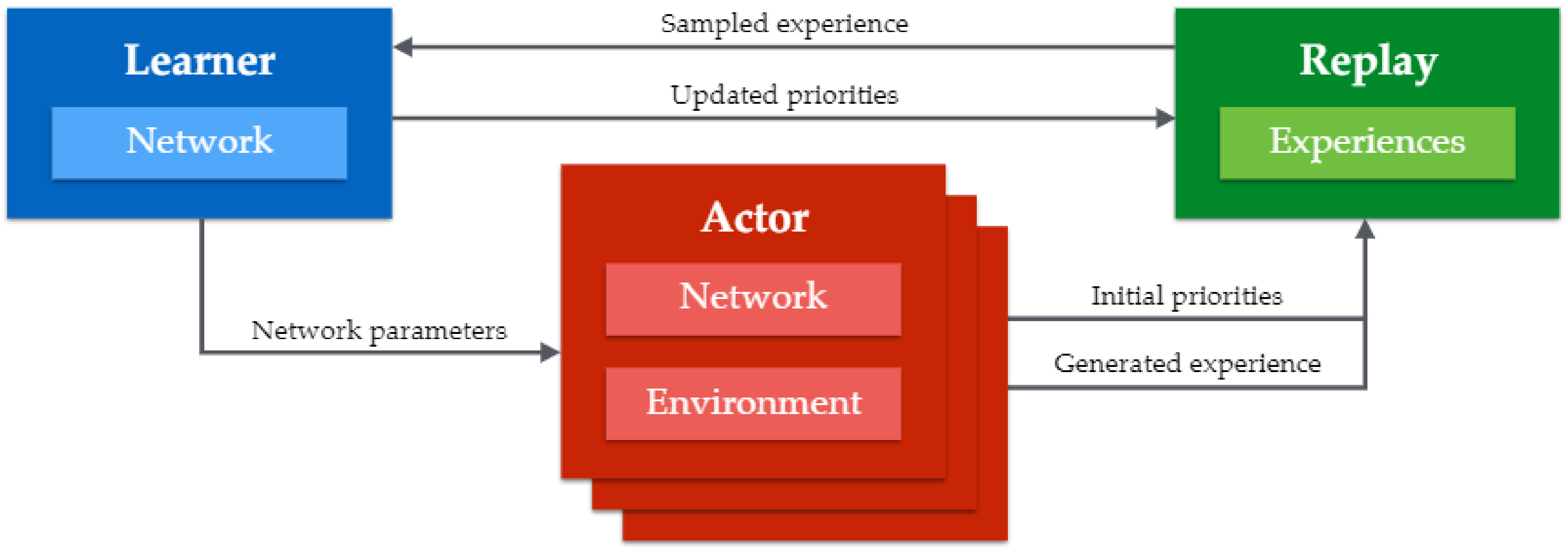
- Shared, Centralized **replay memory**
- **Prioritize** to sample the most useful data more often
    - shared(어떤 actor든지 발견하기만 하면 됨)
- **Priority Definition**
    a. @ Prioritized DQN : sampling된 sample들만 priority를 update / scale up 잘 안됨
    @ **APE-X** : actor들이 replay에 넣을 때 priority를 다시 계산하니, 추가적인 computation없이 큰 문제를 해결

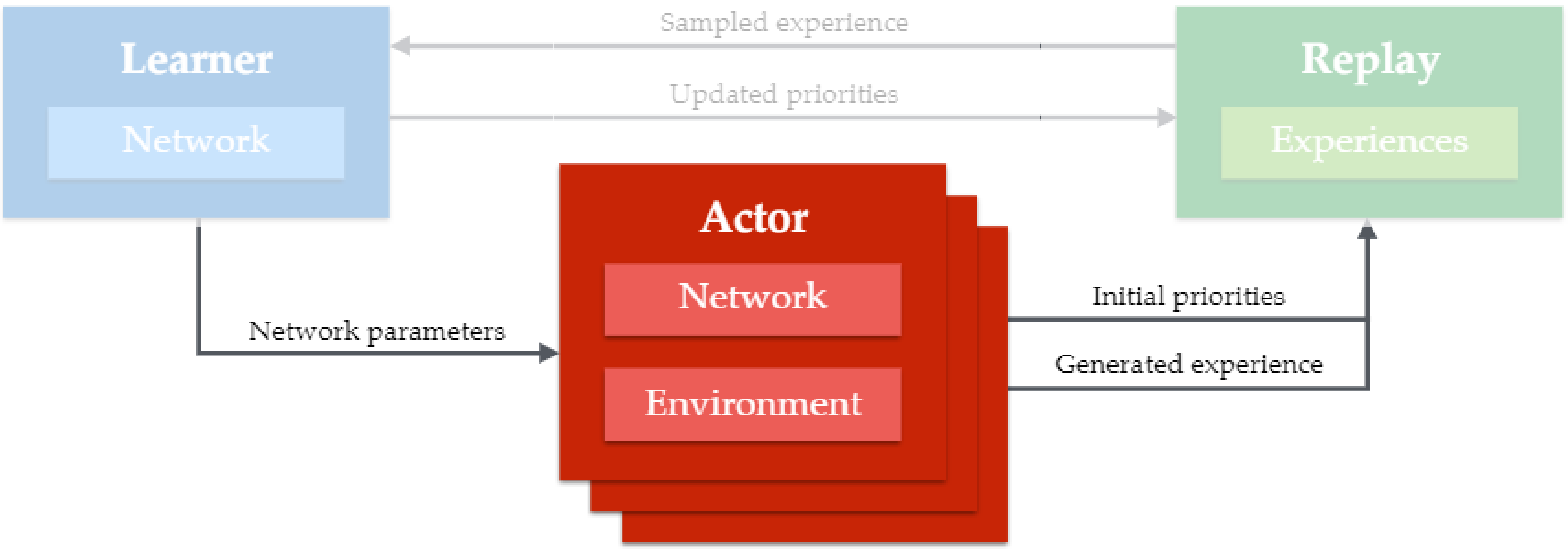


Distributed PER

EXTEND PER TO THE DISTRIBUTED SETTING

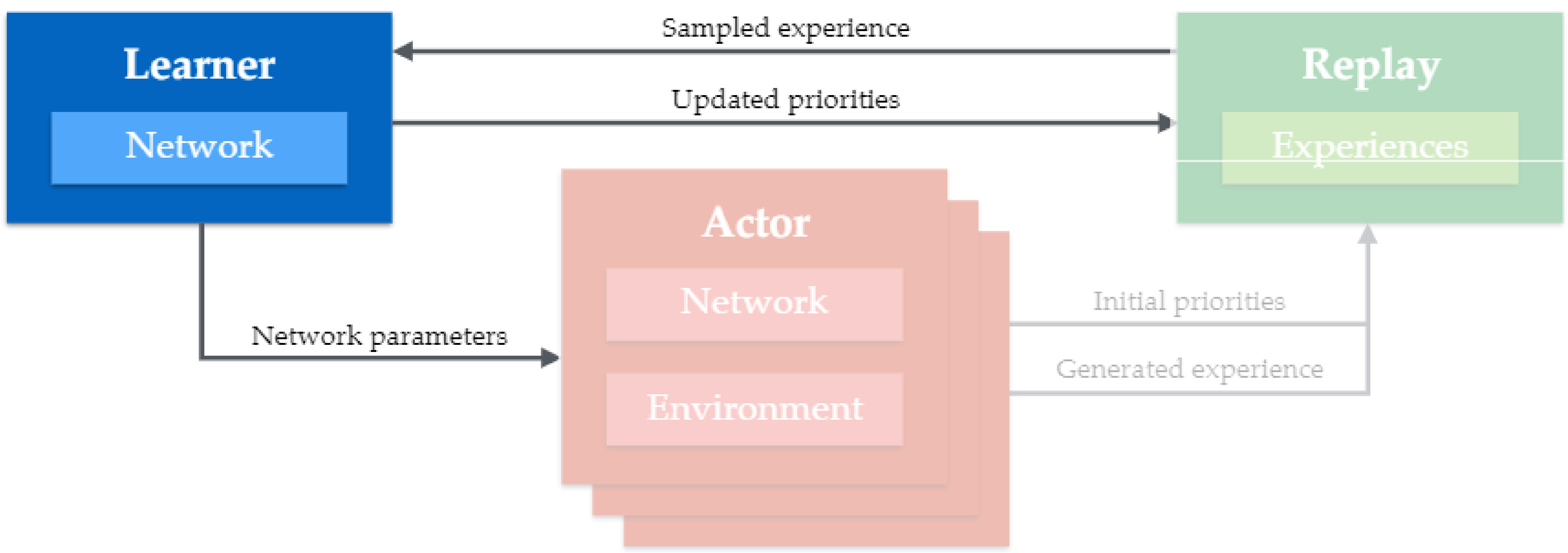
- **Sharing experiences** VS Sharing gradients
  - 경험 데이터의 outdated가 그라디언트보다 늦음
    → off policy 데이터에 대해 robust
  - centralized replay
  - actor와 learner들이 다른 데이터 센터에서 돌아갈 수 있음(퍼포먼스에 제한이 없음)
- **Off policy**
  - 각각의 actor들이 다른 policy로 다양하게 경험
    → exploration에 대해서 진전이 있었음





Algorithm 1 Actor

procedure ACTOR(B, T) ▷ Run agent in environment instance, storing experiences.
  $\theta_0 \leftarrow$ LEARNER.PARAMETERS( ) ▷ Remote call to obtain latest network parameters.
  $s_0 \leftarrow$ ENVIRONMENT.INITIALIZE( ) ▷ Get initial state from environment.
  for $t = 1$ **to** T **do**
    $a_{t-1} \leftarrow \pi_{\theta_{t-1}}(s_{t-1})$ ▷ Select an action using the current policy.
    $(r_t, \gamma_t, s_t) \leftarrow$ ENVIRONMENT.STEP(a_{t-1}) ▷ Apply the action in the environment.
    LOCALBUFFER.ADD(($s_{t-1}, a_{t-1}, r_t, \gamma_t$)) ▷ Add data to local buffer.
    if LOCALBUFFER.SIZE( ) $\geq B$ **then** ▷ In a background thread, periodically send data to replay.
      $\tau \leftarrow$ LOCALBUFFER.GET(B) ▷ Get buffered data (e.g. batch of multi-step transitions).
      $p \leftarrow$ COMPUTEPRIORITIES(τ) ▷ Calculate priorities for experience (e.g. absolute TD error).
      REPLAY.ADD(τ, p) ▷ Remote call to add experience to replay memory.
    end if
    PERIODICALLY($\theta_t \leftarrow$ LEARNER.PARAMETERS()) ▷ Obtain latest network parameters.
  end for
end procedure



Algorithm 2 Learner

procedure LEARNER(T) ▷ Update network using batches sampled from memory.
  $\theta_0 \leftarrow$ INITIALIZENETWORK( )
  for $t = 1$ **to** T **do** ▷ Update the parameters T times.
    $id, \tau \leftarrow$ REPLAY.SAMPLE( ) ▷ Sample a prioritized batch of transitions (in a background thread).
    $l_t \leftarrow$ COMPUTELOSS$(\tau; \theta_t)$ ▷ Apply learning rule; e.g. double Q-learning or DDPG
    $\theta_{t+1} \leftarrow$ UPDATEPARAMETERS$(l_t; \theta_t)$
    $p \leftarrow$ COMPUTEPRIORITIES( ) ▷ Calculate priorities for experience, (e.g. absolute TD error).
    REPLAY.SETPRIORITY(id, p) ▷ Remote call to update priorities.
    PERIODICALLY(REPLAY.REMOVETOFIT()) ▷ Remove old experience from replay memory.
  end for
end procedure

APE-X DQN

ATARI

→ DQN

- **learning algorithm : Double DQN** with **multi-step** bootstrap targets
- **function approximator** : dueling network q(·, ·, θ)

→ batch Loss

$$l_t(\boldsymbol{\theta}) = \frac{1}{2}(G_t - q(S_t, A_t, \boldsymbol{\theta}))^2$$

→ Gain

$$G_t = \underbrace{R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n \overbrace{q(S_{t+n}, \operatorname*{argmax}_a q(S_{t+n}, a, \boldsymbol{\theta}), \boldsymbol{\theta}^-)}^{\text{double-Q bootstrap value}}}_{\text{multi-step return}}$$

target network

APE-X DQN

ATARI

more details

- **Multi-step return with no off-policy correction**
    - adversely affect the value estimation
- Actor executes a **different** policy (off policy)
    - Experiences to be generated from a variety of strategies
    - Behaviour policy affects
        - exploration
        - the quality of function approximation
- Actors use **ε-greedy policies** with different values of ε
    - **Low** ε policies allow *exploring deeper* in the environment
    - **High** ε policies *prevent over-specialization*
    - constant during training

$$\epsilon_i = \epsilon^{1+\frac{i}{N-1}\alpha}$$

$$\epsilon = 0.4, \alpha = 7$$

Ape-X DPG

CONTINUOUS CONTROL TASK

Continuous-action policy gradient system based on DDPG

- smilar to Ape-X DQN
- the policy : φ
- Q-network : ψ
- The two networks are optimized separately, by minimizing different losses on the sampled experience

$$G_t = \underbrace{R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n \overbrace{q(S_{t+n}, \operatorname*{argmax}_a q(S_{t+n}, a, \boldsymbol{\theta}), \boldsymbol{\theta}^-)}^{\text{double-Q bootstrap value}}}_{\text{multi-step return}}$$

$$G_t = \underbrace{R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n q(S_{t+n}, \pi(S_{t+n}, \phi^-), \psi^-)}_{\text{multi-step return}}$$

Ape-X DPG

CONTINUOUS CONTROL TASK

$$G_t = \underbrace{R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n q(S_{t+n}, \pi(S_{t+n}, \phi^-), \psi^-)}_{\text{multi-step return}}$$

more details

- **Q-network**
  - output : an action-value estimate $q(s, a, \psi)$
  - updated using TD learning with a multi-step bootstrap target

→ **Loss**

$$l_t(\psi) = \frac{1}{2}(G_t - q(S_t, A_t, \psi))^2$$

- **Policy network**
  - output : an action $A_t = \pi(S_{t,}, \phi) \in \mathbb{R}^m$
  - updated using policy **gradient ascent** on the estimated Q-value

→ **Gradient**

$$\nabla_\phi q(S_t, \pi(S_t, \phi), \psi)$$

$A_t = \pi(S_{t,}, \phi)$ → input to critic network

Experiments

ATARI

APE-X DQN

- **360** actor machines / 1 CPU core
- 139 FPS each → Total ~50k(= 139X360) FPS = ~12.5K transitions (4개 frame 당 1개 action)
- **Actor** :
  - batch experience data **locally** before sending it to the replay
  - copy Learner's parameters every 400 frames(~2.8 sec)
- **Learner** :
  - 비동기적으로 512 transition을 16개의 batch로 가져옴
  - 1초 당 19 batch update = ~9.7K transitions에 대한 gradient들을 계산
- Observation : PNG codec / Episode length : limited to 50000 frames during training
- Capacity of the shared experience replay memory : 약 2 million transitions (2035050)
- Data is sampled according to **proportional prioritization**
  - a priority exponent of 0.6
  - an importance sampling exponent 0.4

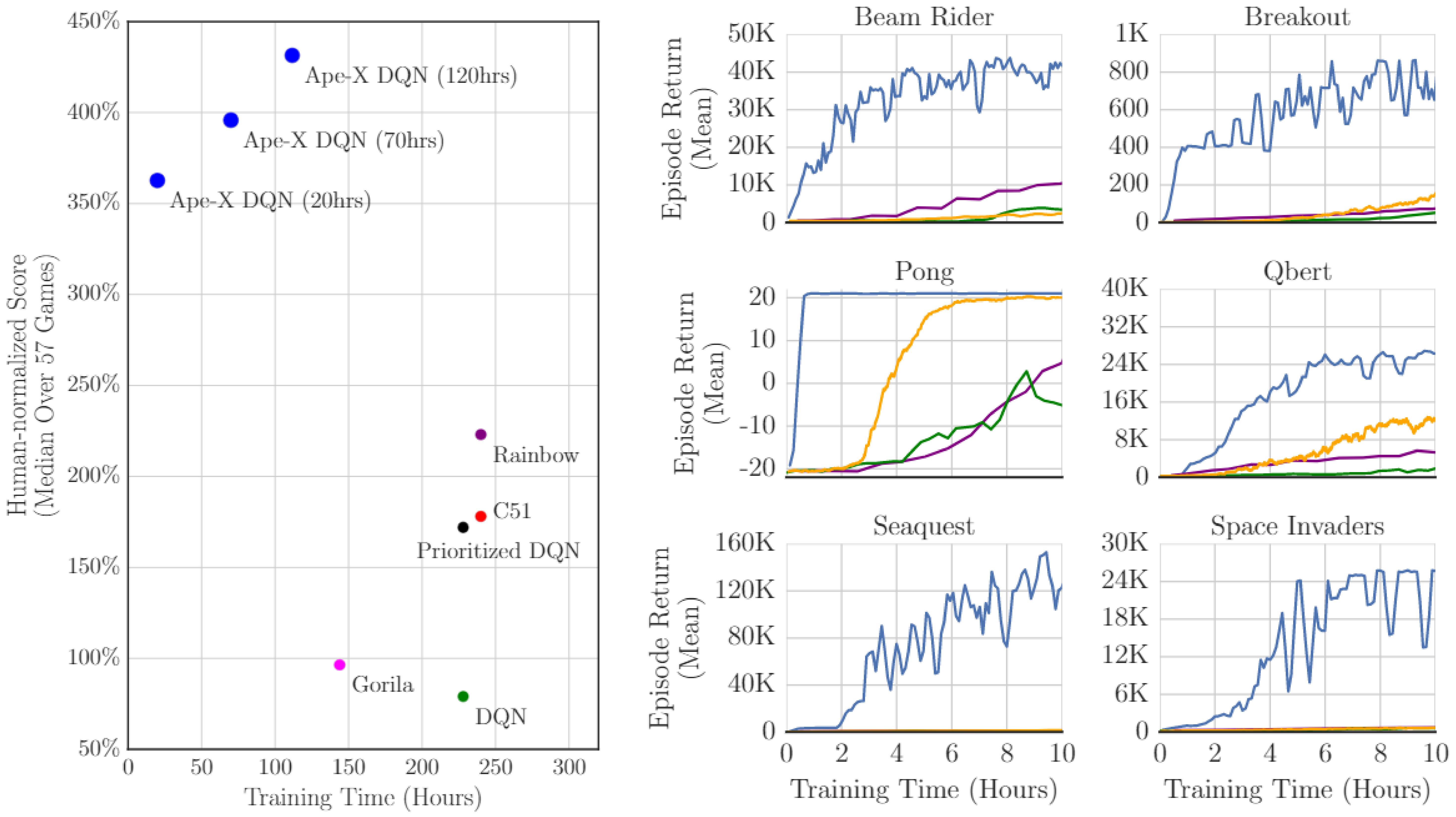


Figure 2: Left: Atari results aggregated across 57 games, evaluated from random no-op starts. Right: Atari training curves for selected games, against baselines. Blue: Ape-X DQN with 360 actors; Orange: A3C; Purple: Rainbow; Green: DQN. See appendix for longer runs over all games.

Experiments

ATARI

Algorithm	Training Time	Environment Frames	Resources (per game)	Median (no-op starts)	Median (human starts)
Ape-X DQN	5 days	22800M	376 cores, 1 GPU [a]	**434%**	**358%**
Rainbow	10 days	200M	1 GPU	223%	153%
Distributional (C51)	10 days	200M	1 GPU	178%	125%
A3C	4 days	—	16 cores	—	117%
Prioritized Dueling	9.5 days	200M	1 GPU	172%	115%
DQN	9.5 days	200M	1 GPU	79%	68%
Gorila DQN [c]	~4 days	—	unknown [b]	96%	78%
UNREAL [d]	—	250M	16 cores	331% [d]	250% [d]

Table 1: Median normalized scores across 57 Atari games. [a] Tesla P100. [b] >100 CPUs, with a mixed number of cores per CPU machine. [c] Only evaluated on 49 games. [d] Hyper-parameters were tuned per game.

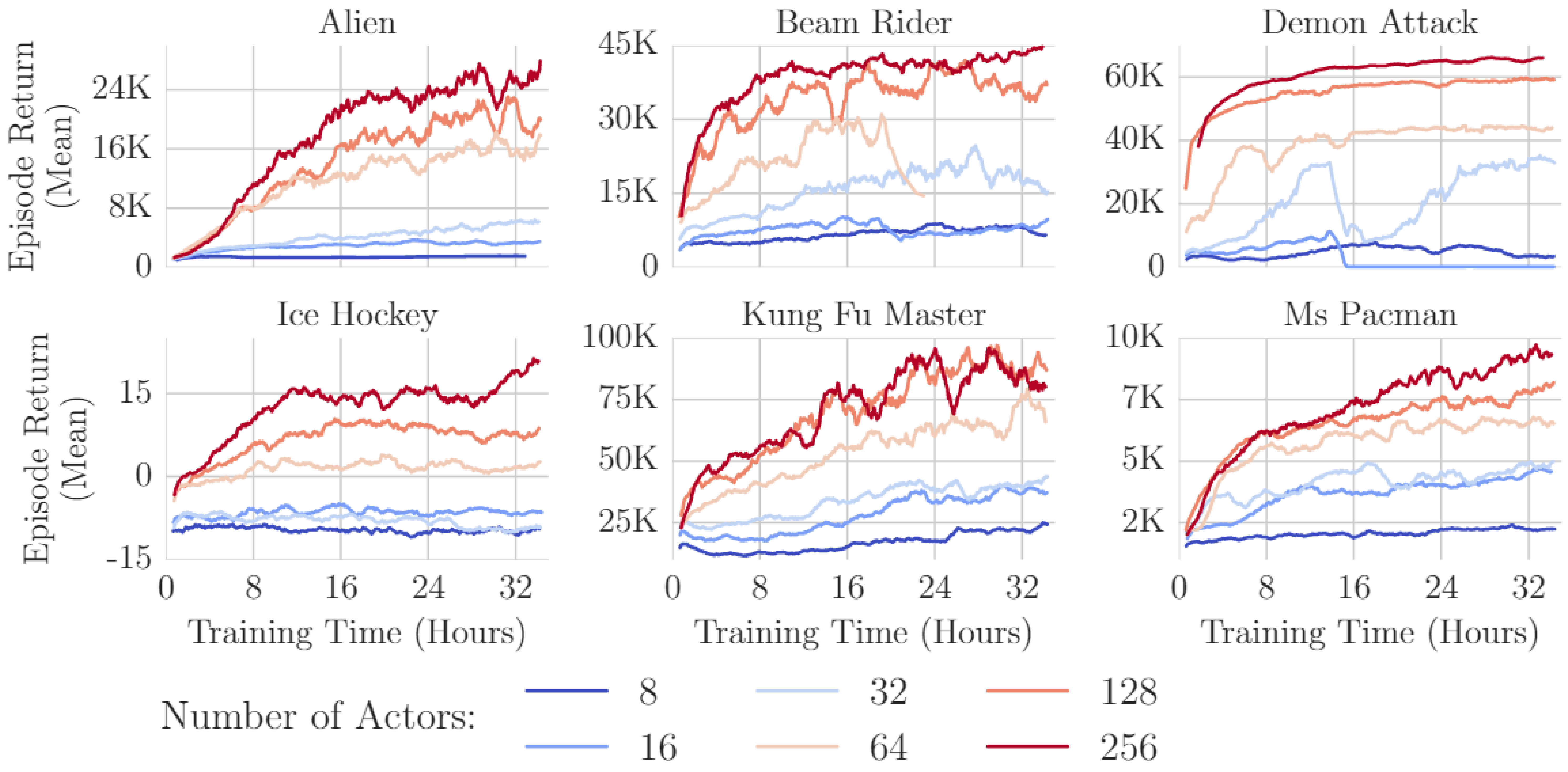


Figure 4: Scaling the number of actors. Performance consistently improves as we scale the number of actors from 8 to 256, note that the number of learning updates performed does not depend on the number of actors.

Experiments

CONTINUOUS CONTROL

APE-X DPG

- 4 tasks
  - manipulator
  - humanoid : Standing / Walking / Running
- observation : features (small FC layers)
- 64 actors
  - ~14K total FPS
  - 86 batches of 256 transitions / sec (~22K transitions /sec)

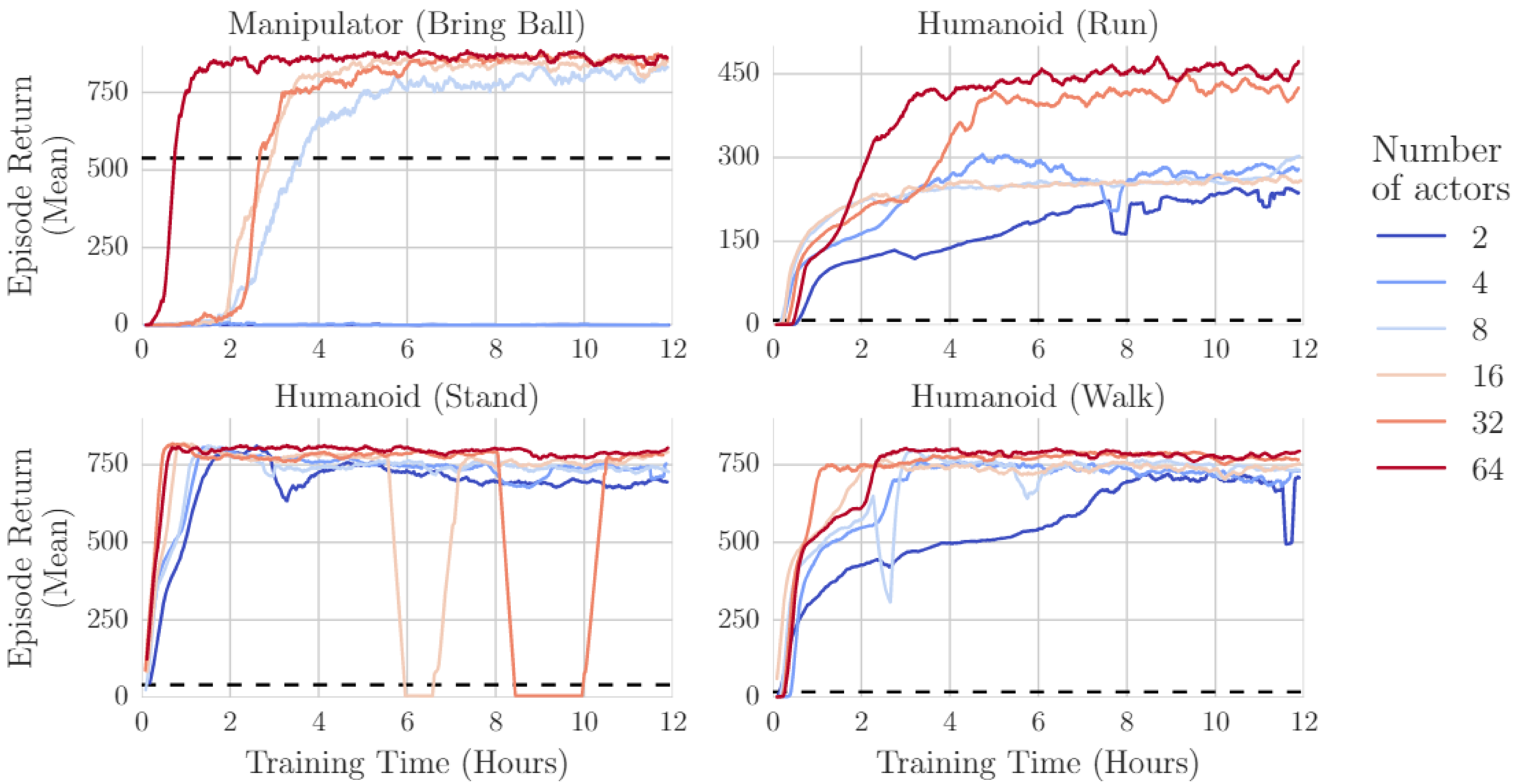


Figure 3: Performance of Ape-X DPG on four continuous control tasks, as a function of wall clock time. Performance improves as we increase the numbers of actors. The black dashed line indicates the maximum performance reached by a standard DDPG baseline over 5 days of training.

Analysis

HOW THE PERFORMANCE SCALES WITH THE NUMBER OF ACTORS

actor수가 많을수록 퍼포먼스가 좋을까?

- 8, 16, 32, 64, 128 and 256) for 35 hours on a subset of 6 Atari games
- shared experience replay memory 크기 고정 (1 million transition)
- 다다익선
- PER 없이도 비교해보았는데 actor가 많을수록 좋다는 결과는 동일했음(appendix)
- 본 연구의 아키텍쳐가 흔한 RL의 실패를 개선 시킬 수 있을 것
    - 어떤 실패?
        - the policy discovered is a **local optimum** in the parameter space, but **not a global one**
    - 왜 개선이 되는가?
        - large number of actors with varying amounts of exploration helps to **discover promising new courses of action**

Analysis

VARYING THE CAPACITY OF THE REPLAY MEMORY

replay memory의 용량이 달라지면 어떨까?

- 256 actors
- a median of ~37K total environment frames per second (약 ~9K transition)
- the contents of the memory is replaced much **faster** than in most DQN-like agents
- a **small benefit** to using a larger replay capacity
  - 왜? the value of keeping some high priority experiences around for longer and replaying them

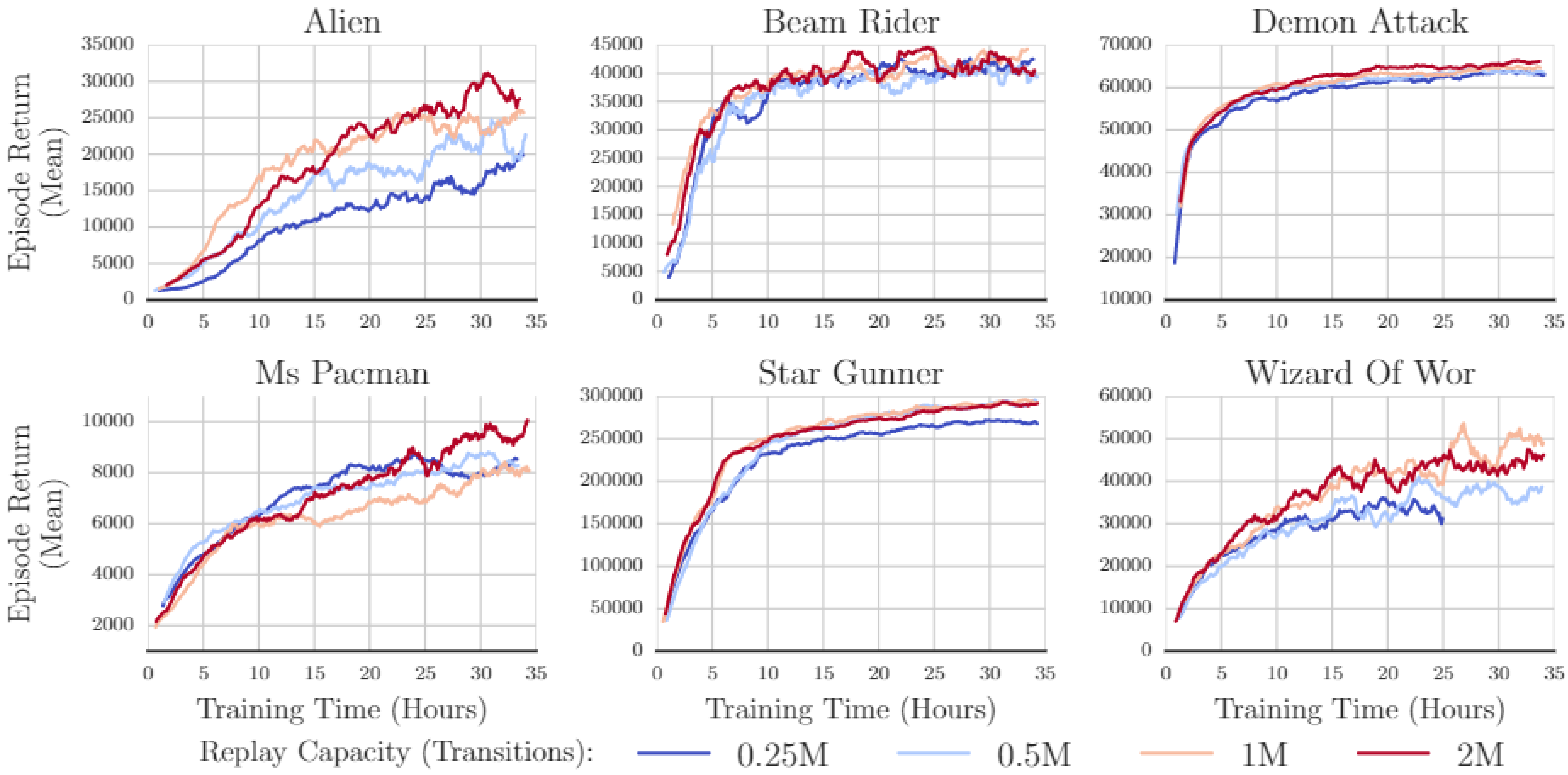


Figure 5: Varying the capacity of the replay. Agents with larger replay memories perform better on most games. Each curve corresponds to a single run, smoothed over 20 points. The curve for Wizard Of Wor with replay size 250K is incomplete because training diverged; we did not observe this with the other replay sizes.

Analysis

DISENTANGLE POTENTIAL EFFECTS OF TWO CONFOUNDING FACTORS IN OUR **SCALABILITY ANALYSIS**

Scalability의 다른 요인들

1. **Recency of the experience data in the replay memory**
2. **Diversity of the data-generating policies**

- appendix
- **neither** factor alone is sufficient to explain the performance we see
- gathering more experience data에 도움을 줄 뿐
- better exploration of the environment and better avoidance of overfitting

Conclusion

- Combined with **any other off-policy** reinforcement learning update
- 병렬적으로 많은 양의 데이터들을 만들어 낼 수 있는 상황에 적합한 체계 (many instances)
- RL의 문제인 large domain에서의 exploration 문제를
    - generating a diverse set of experiences = many actors
    - identifying and learning from the most useful events = per
- simple and direct approaches

Appendix

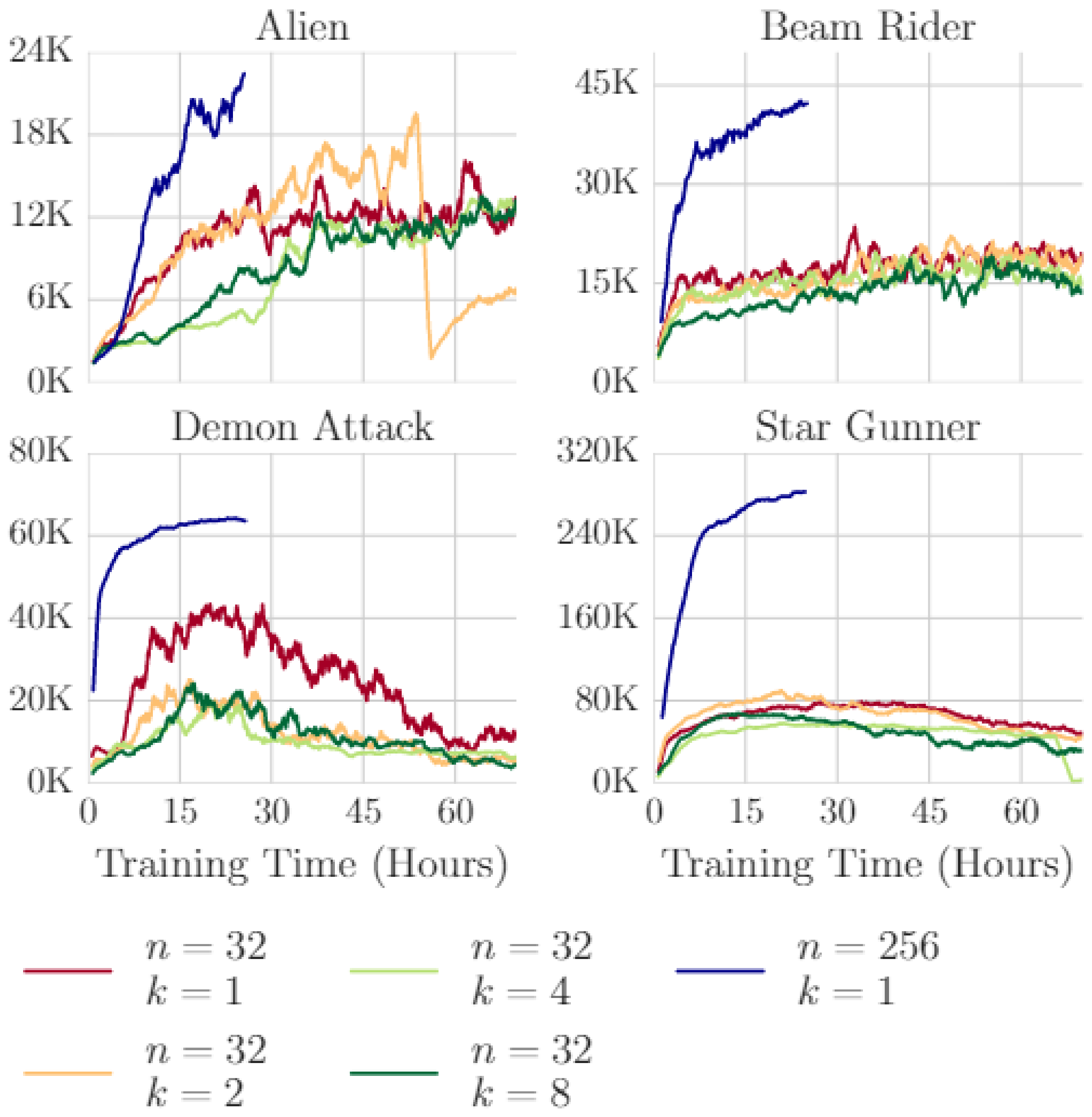


Figure 6: Testing whether improved performance is caused by recency alone: n denotes the number of actors, k the number of times each transition is replicated in the replay. The data in the run with $n = 32$, $k = 8$ is therefore as recent as the data in the run with $n = 256$, $k = 1$, but performance is not as good.

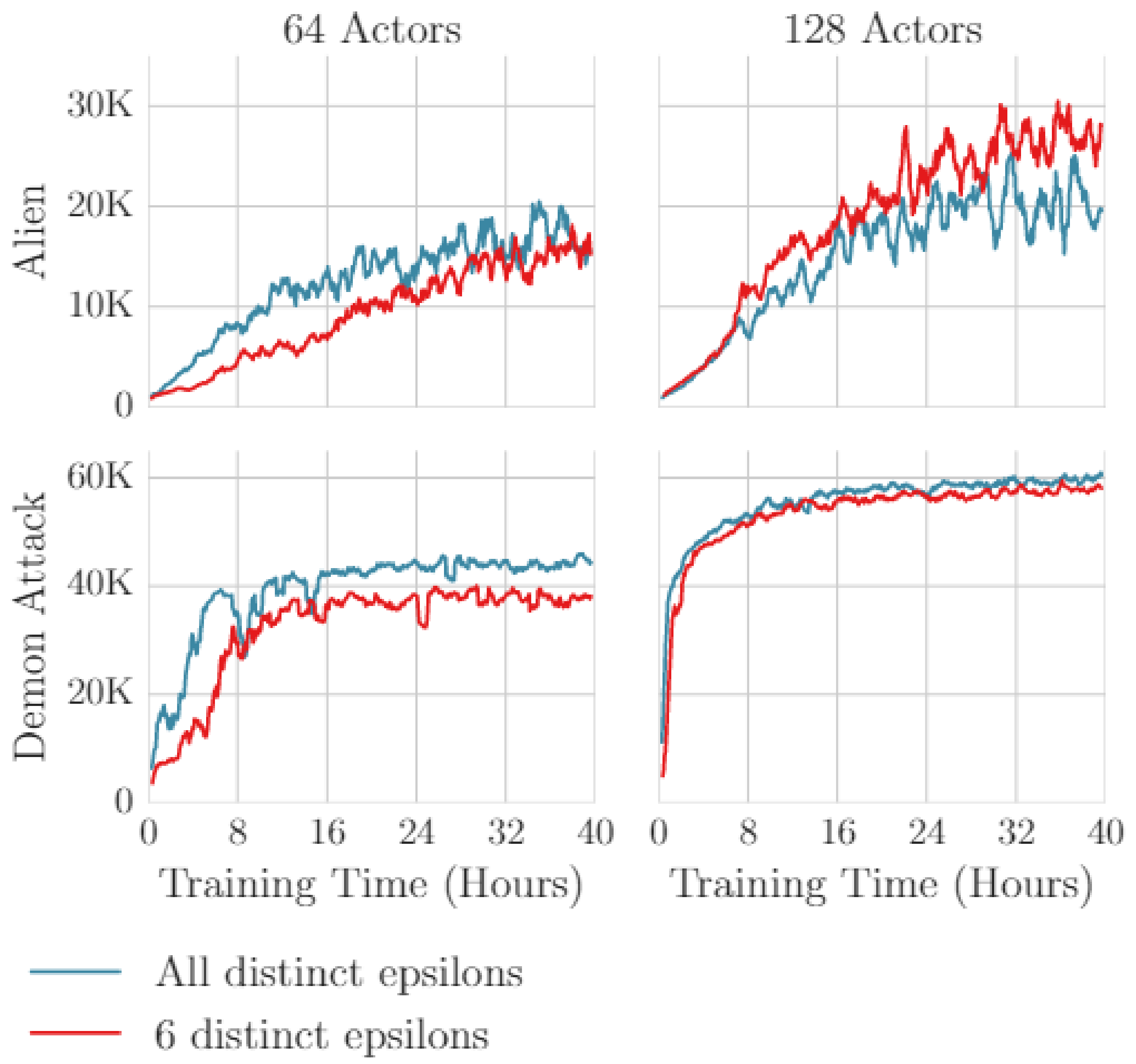


Figure 7: Varying the data-generating policies: Red: fixed set of 6 values for ϵ. Blue: full range of values for ϵ. In both cases, the curve plotted is from a separate actor that does not add data to the replay memory, and which follows an ϵ-greedy policy with $\epsilon = 0.00164$.

https://www.slideshare.net/ssuser163469/distributed-prioritized-experience-replay

https://seolhokim.github.io/deeplearning/2020/04/09/apex-review/

https://www.opentutorials.org/module/3653/22071