Distributed Deep Reinforcement Learning

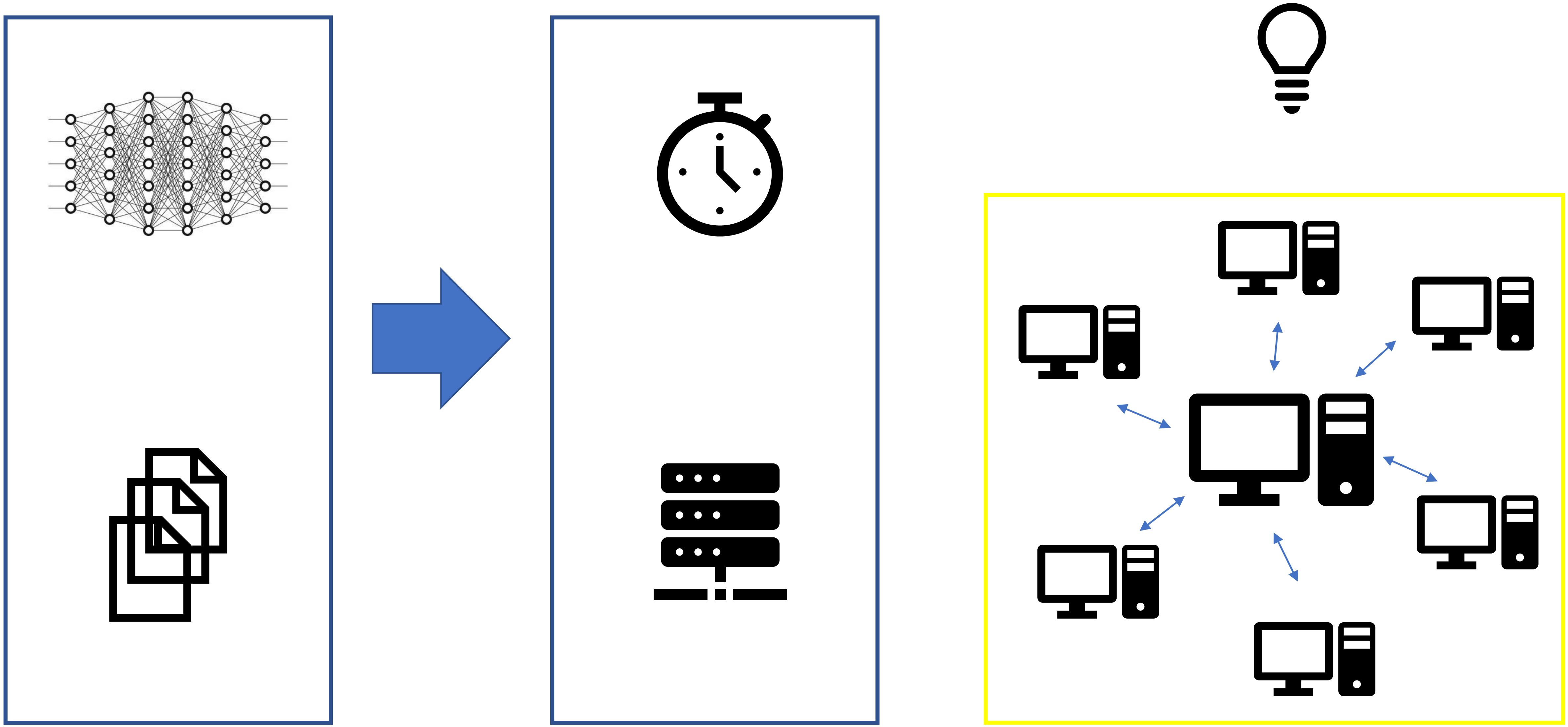
Lucia Liu

Distributed Prioritized Experience Replay

Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, David Silver

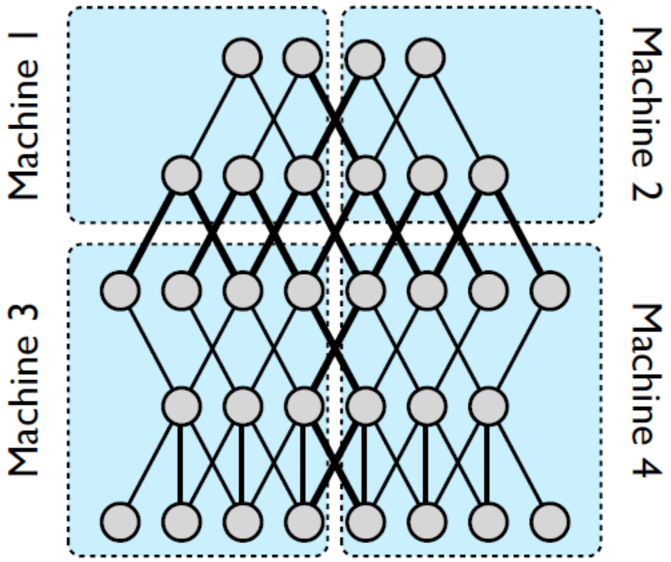
Distributed Distributional Deterministic Policy Gradients

Gabriel Barth-Maron, Matthew W. Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva TB, Alistair Muldal, Nicolas Heess, Timothy Lillicrap



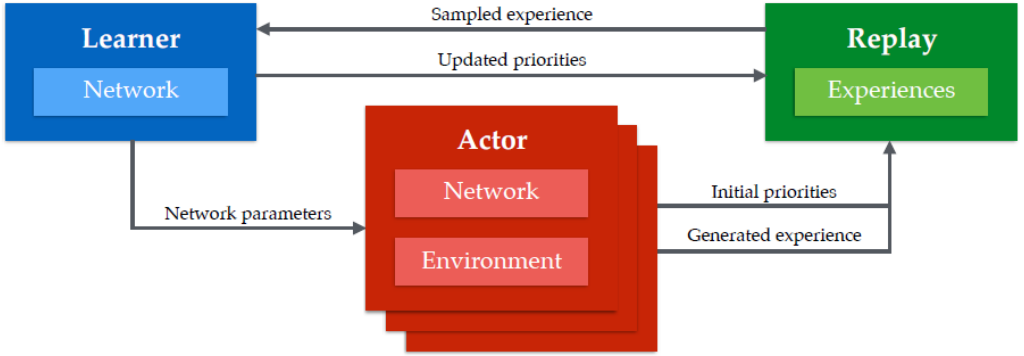
Approaches to Distributed Architecture

- Distribution through **parallelizing computation of gradients**



Dean et al. in NIPS 2012: *Large scale distributed deep networks*.

- Distribution of **generation and selection** of experience data



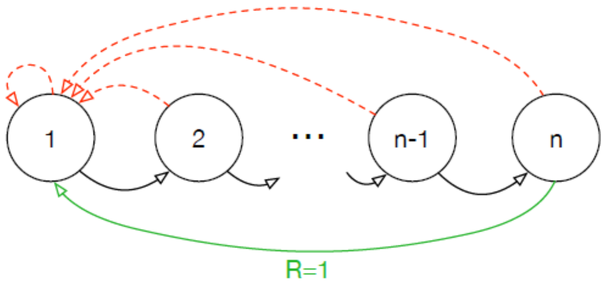
- Deep Q-Network (DQN)
- Deep Deterministic Policy Gradient (DDPG)

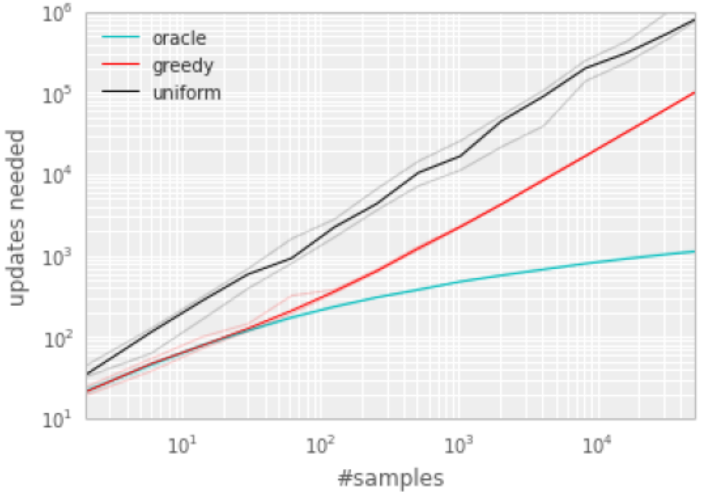
Prioritized Experience Replay

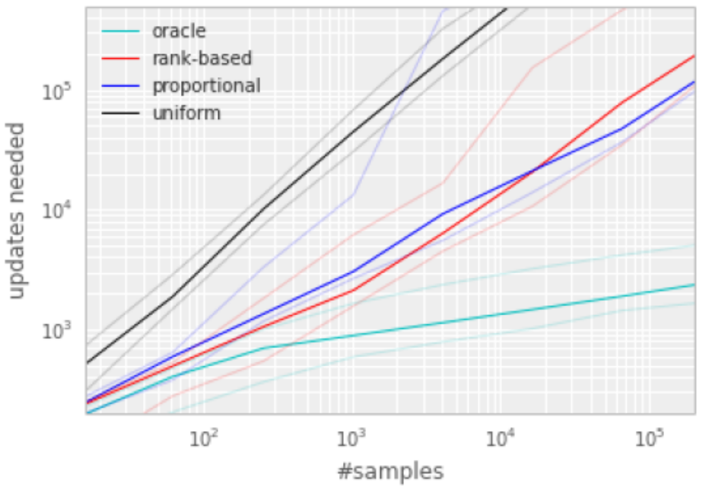
1 **Greedy TD-error prioritization**

2 **Stochastic prioritization**

- Proportional
- Rank-based

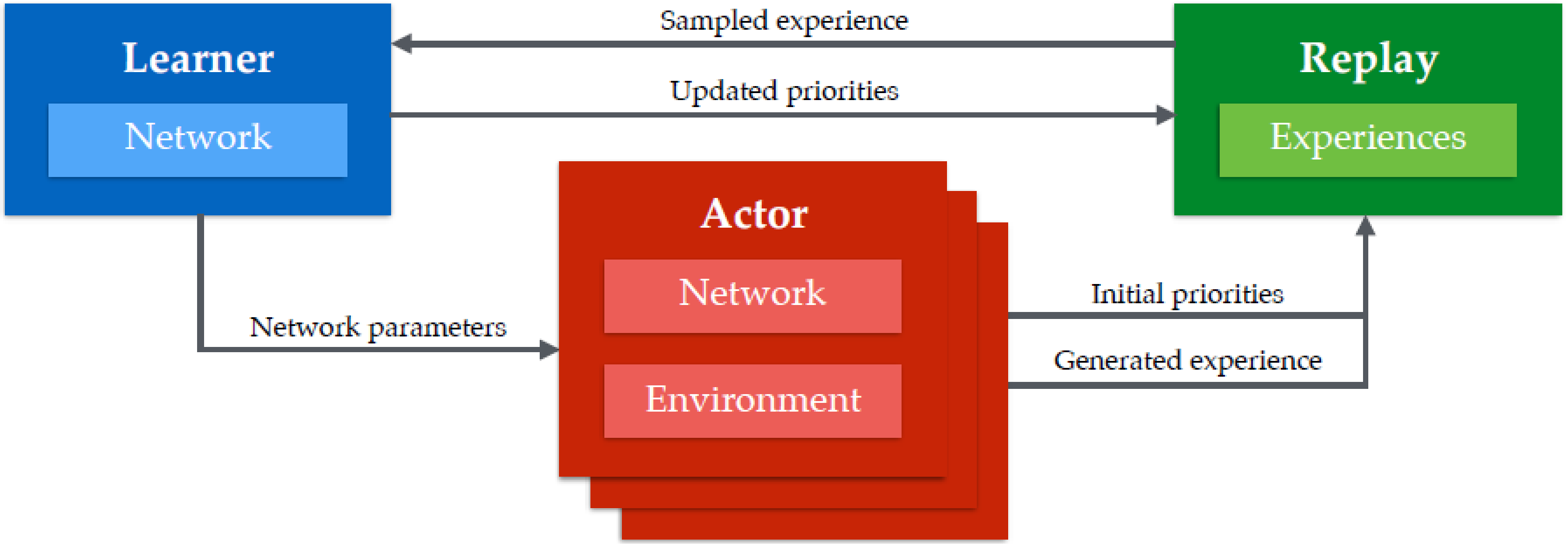






Schaul et al. in ICLR 2016: Prioritized Experience Replay

Distributed Prioritized Experience Replay (*Ape-X*)



Actor

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

Learner

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

Advantages

Shared, centralized replay memory

+

Prioritization

+

Off-Policy

⇒

High priority data discovered by any actor benefits whole system

Ape-X DQN

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

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

- Behavior policy: Different policy for each actor, ε-greedy

Ape-X DPG

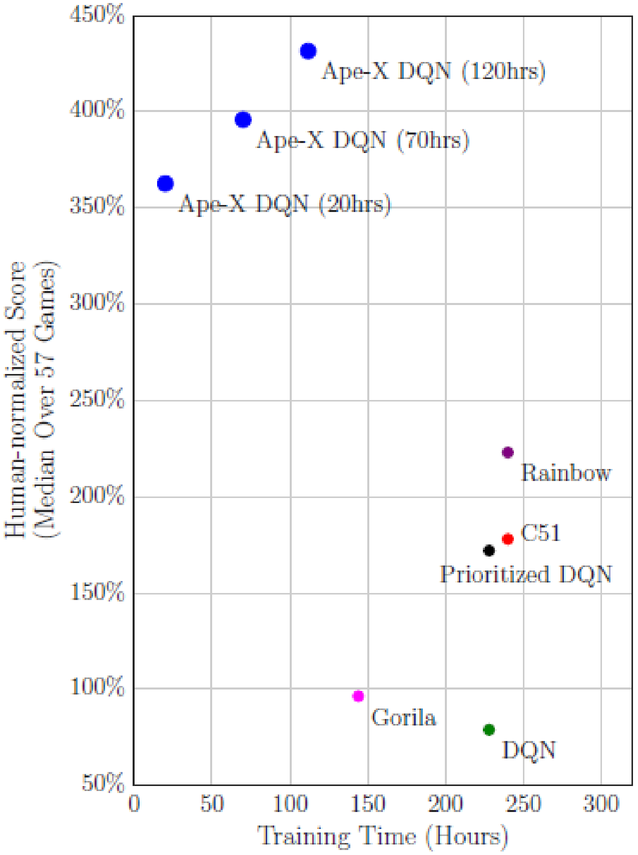
- Actor policy network + Q-network
- Q-network:
  - Action-value estimate q(s,a,ψ)
  - Loss:

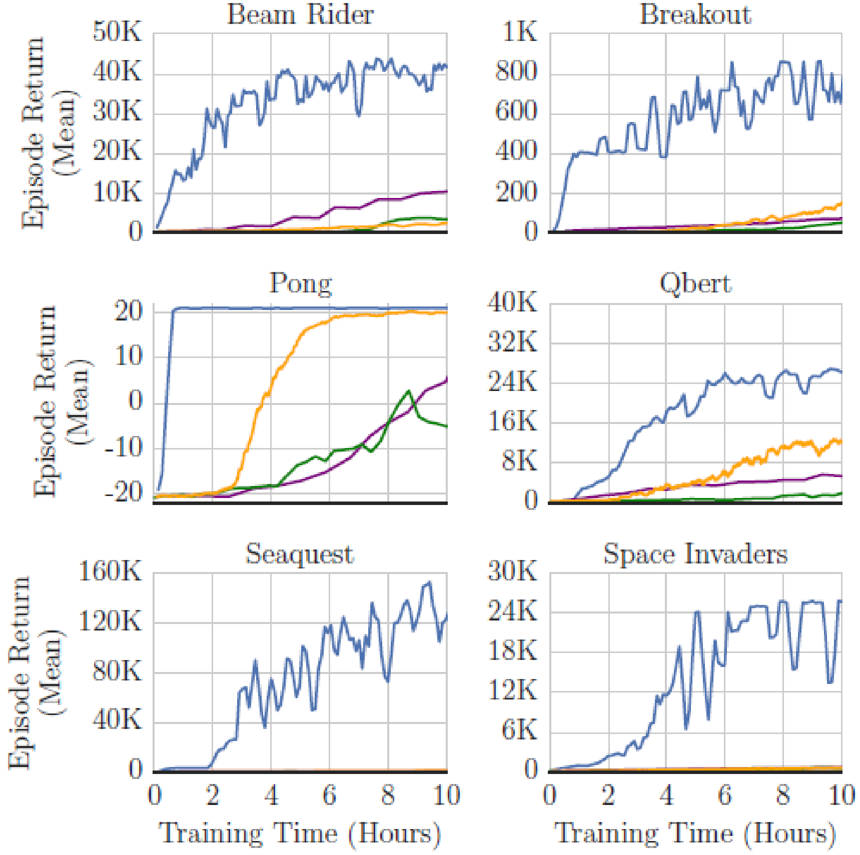
$$l_t(\psi) = \frac{1}{2}(G_t - q(S_t, A_t, \psi))^2$$

- Policy network:
  - Action $A_t = \pi(S_t, \phi)$
  - Gradient $\nabla_\phi q(S_t, \pi(S_t, \phi), \psi)$

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

Results: Atari games



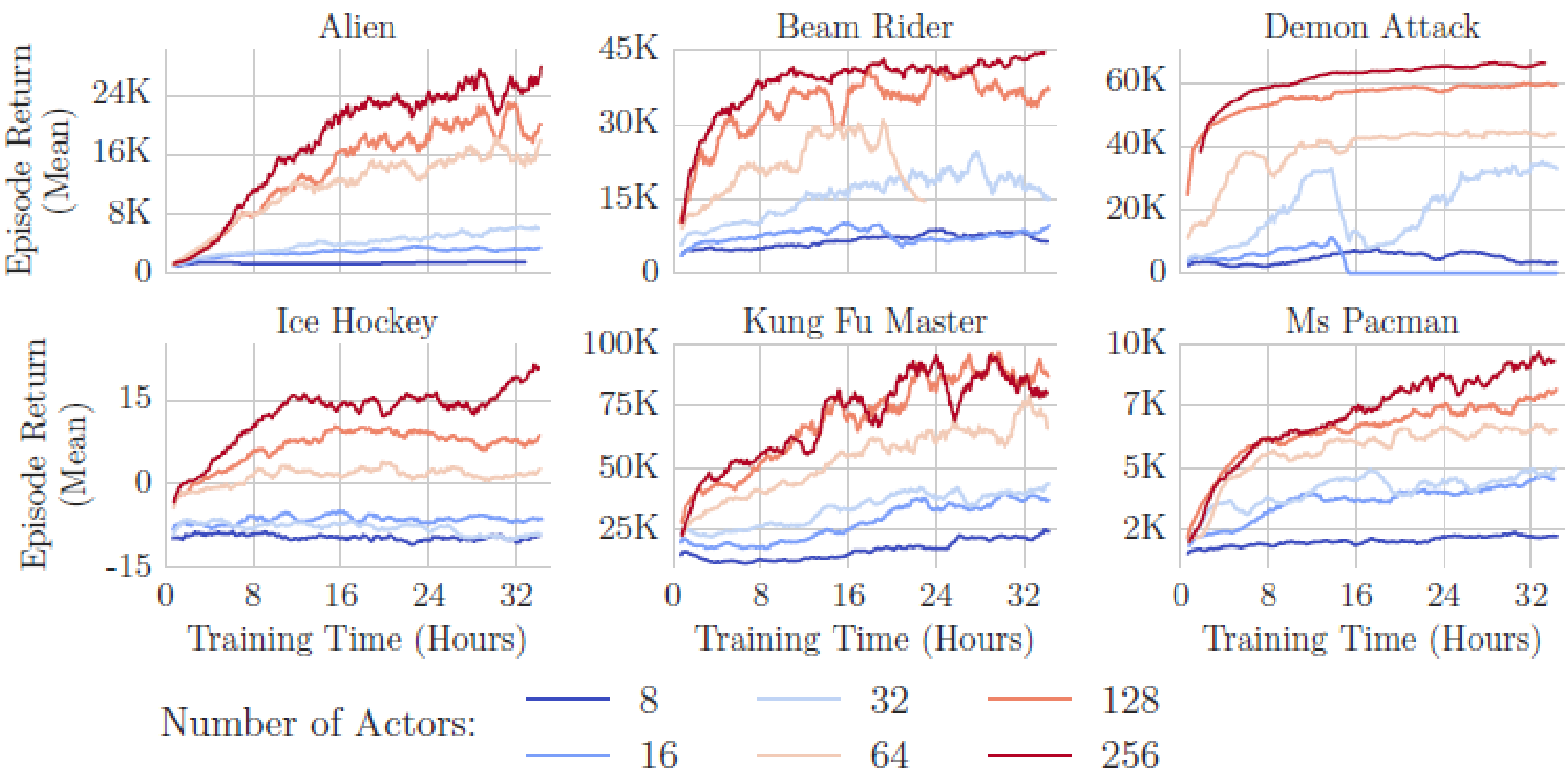


Blue: Ape-X DQN
Orange: A3C
Purple: Rainbow
Green: DQN

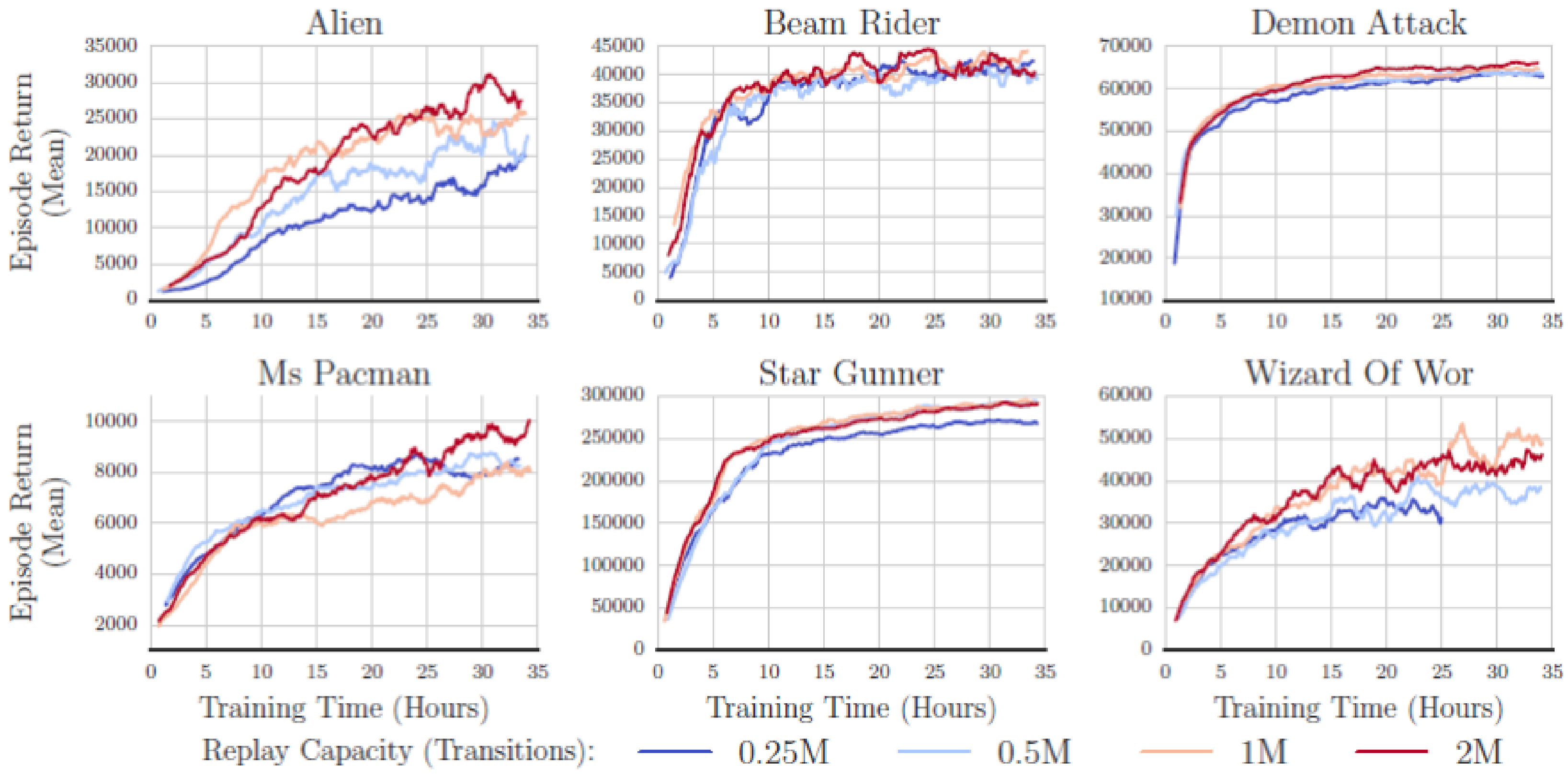
Results: Atari games

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]	$\sim$4 days	—	unknown [b]	96%	78%
UNREAL [d]	—	250M	16 cores	331% [d]	250% [d]

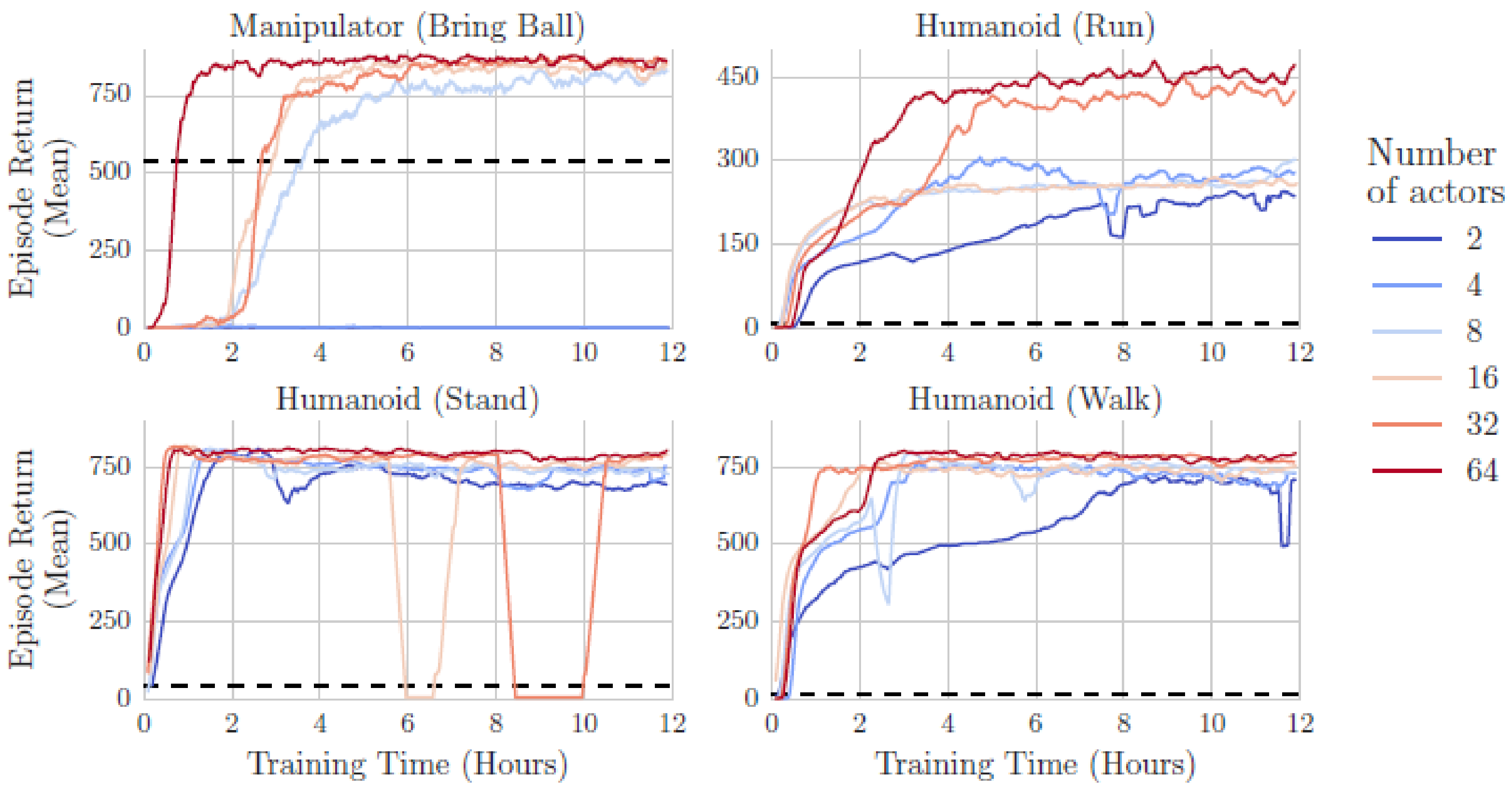
Results: Atari games



Results: Atari games



Results: Continuous Control



Distributed Distributional Deterministic Policy Gradients

Gabriel Barth-Maron, Matthew W. Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva TB, Alistair Muldal, Nicolas Heess, Timothy Lillicrap

Distributed Distributional DDPG (D4PG)

- Based on DDPG algorithm with 4 extensions

Distributional critic update

Distributed parallel actors

N-step returns

Prioritization of experience replay

Distributional critic

- Distributional update with random variable Z_π → $Q_\pi(x,a) = \mathrm{E}[Z_\pi(x,a)]$
- Distributional Bellman operator

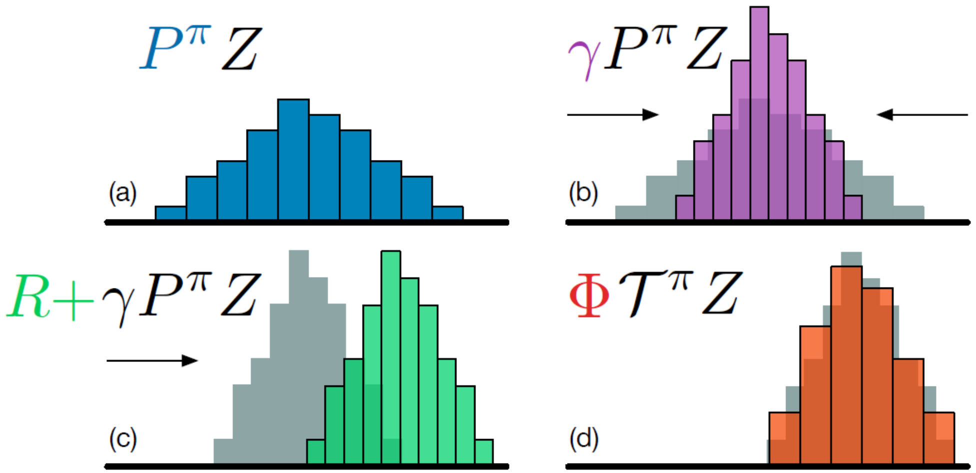
$$(\mathcal{T}_\pi Z)(\mathbf{x},\mathbf{a}) = r(\mathbf{x},\mathbf{a}) + \gamma\mathbb{E}\big[Z(\mathbf{x}', \pi(\mathbf{x}')) \,\big|\, \mathbf{x},\mathbf{a}\big]$$

- Loss

$$L(w) = \mathbb{E}_\rho\Big[d(\mathcal{T}_{\pi_{\theta'}} Z_{w'}(\mathbf{x},\mathbf{a}), Z_w(\mathbf{x},\mathbf{a}))\Big]$$

- Gradient for actor update

$$\nabla_\theta J(\theta) \approx \mathbb{E}_\rho\Big[\nabla_\theta \pi_\theta(\mathbf{x})\, \nabla_{\mathbf{a}} Q_w(\mathbf{x},\mathbf{a})\big|_{\mathbf{a}=\pi_\theta(\mathbf{x})}\Big],$$
$$= \mathbb{E}_\rho\Big[\nabla_\theta \pi_\theta(\mathbf{x})\, \mathbb{E}[\nabla_{\mathbf{a}} Z_w(\mathbf{x},\mathbf{a})]\big|_{\mathbf{a}=\pi_\theta(\mathbf{x})}\Big].$$



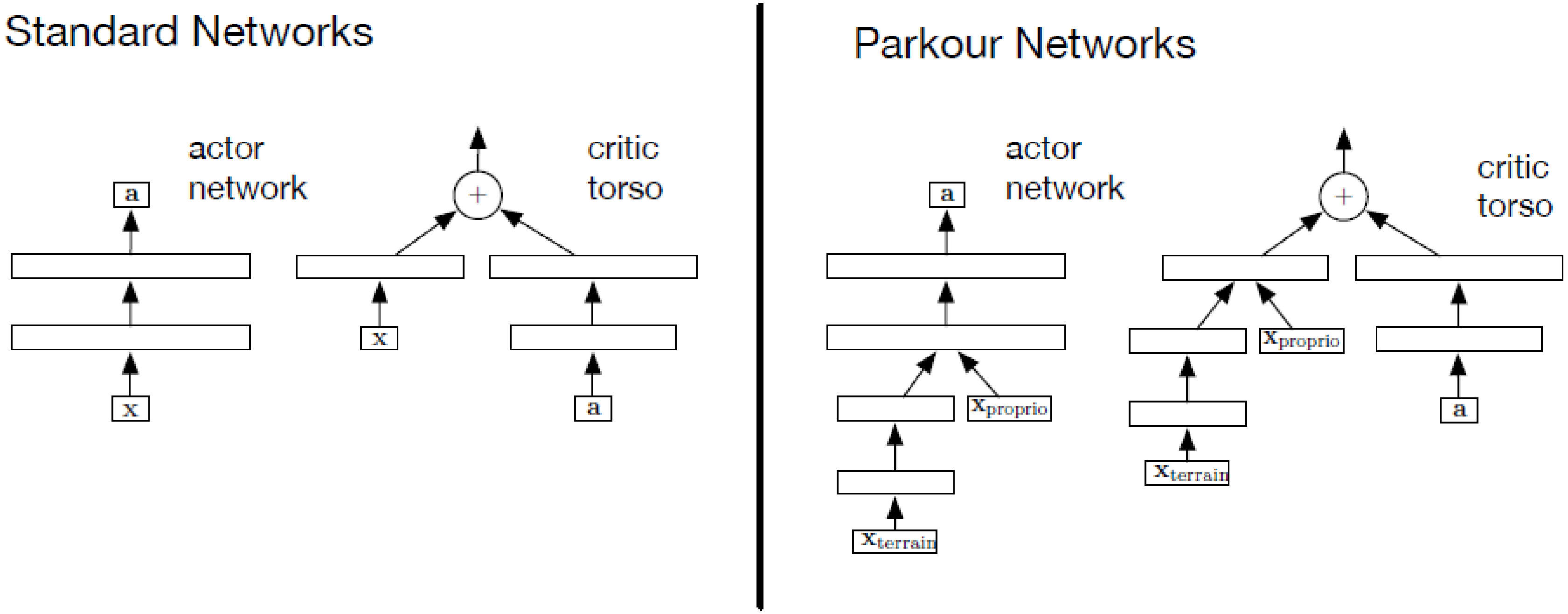
Bellemare et al. ICML 2017. *A distributional perspective on reinforcement learning.*

N-step returns

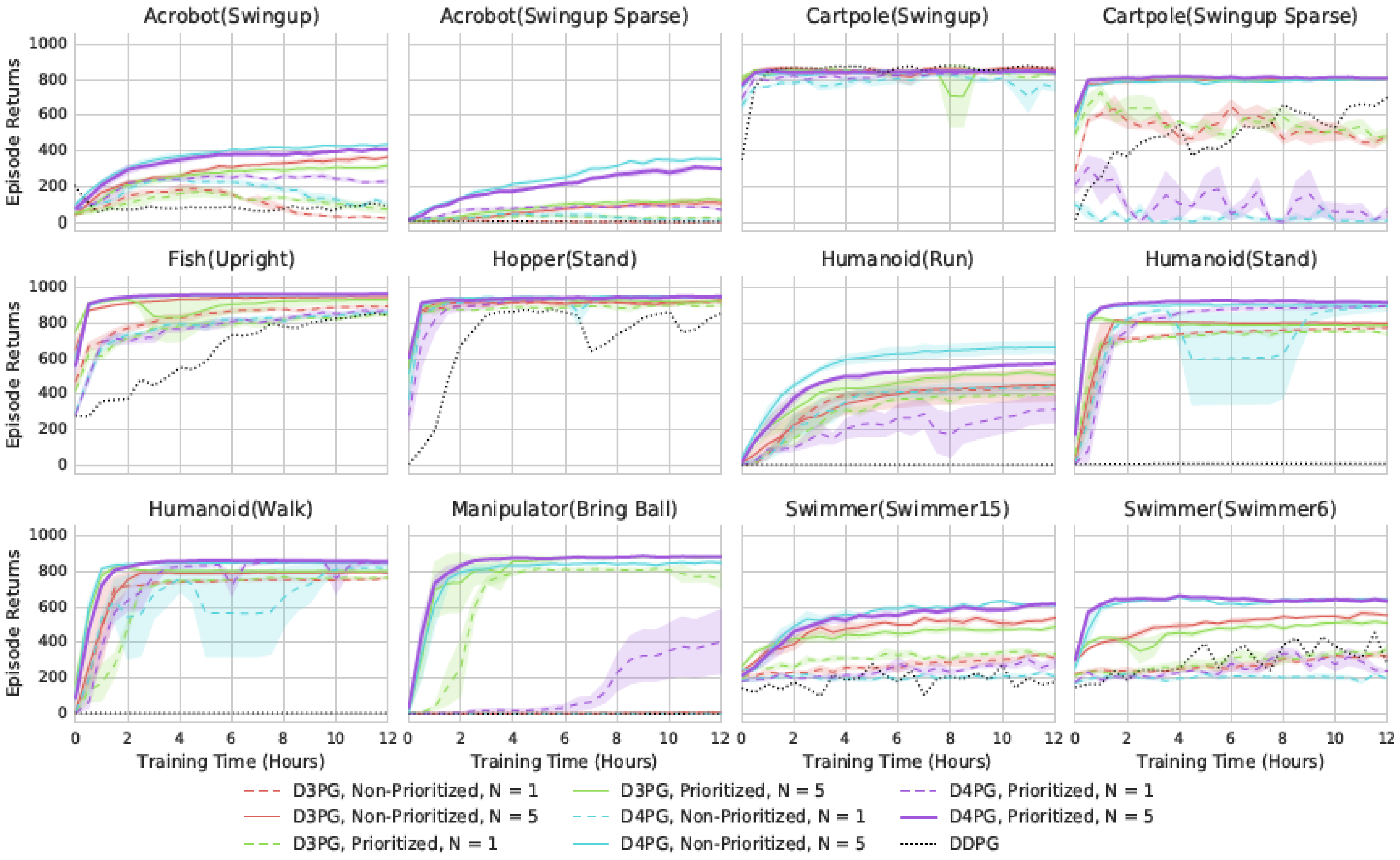
- Replacing Bellman operator with

$$(\mathcal{T}_\pi^N Q)(\mathbf{x}_0, \mathbf{a}_0) = r(\mathbf{x}_0, \mathbf{a}_0) + \mathbb{E}[\sum_{n=1}^{N-1} \gamma^n r(\mathbf{x}_n, \mathbf{a}_n) + \gamma^N Q(\mathbf{x}_N, \pi(\mathbf{x}_N)) \,|\, \mathbf{x}_0, \mathbf{a}_0]$$

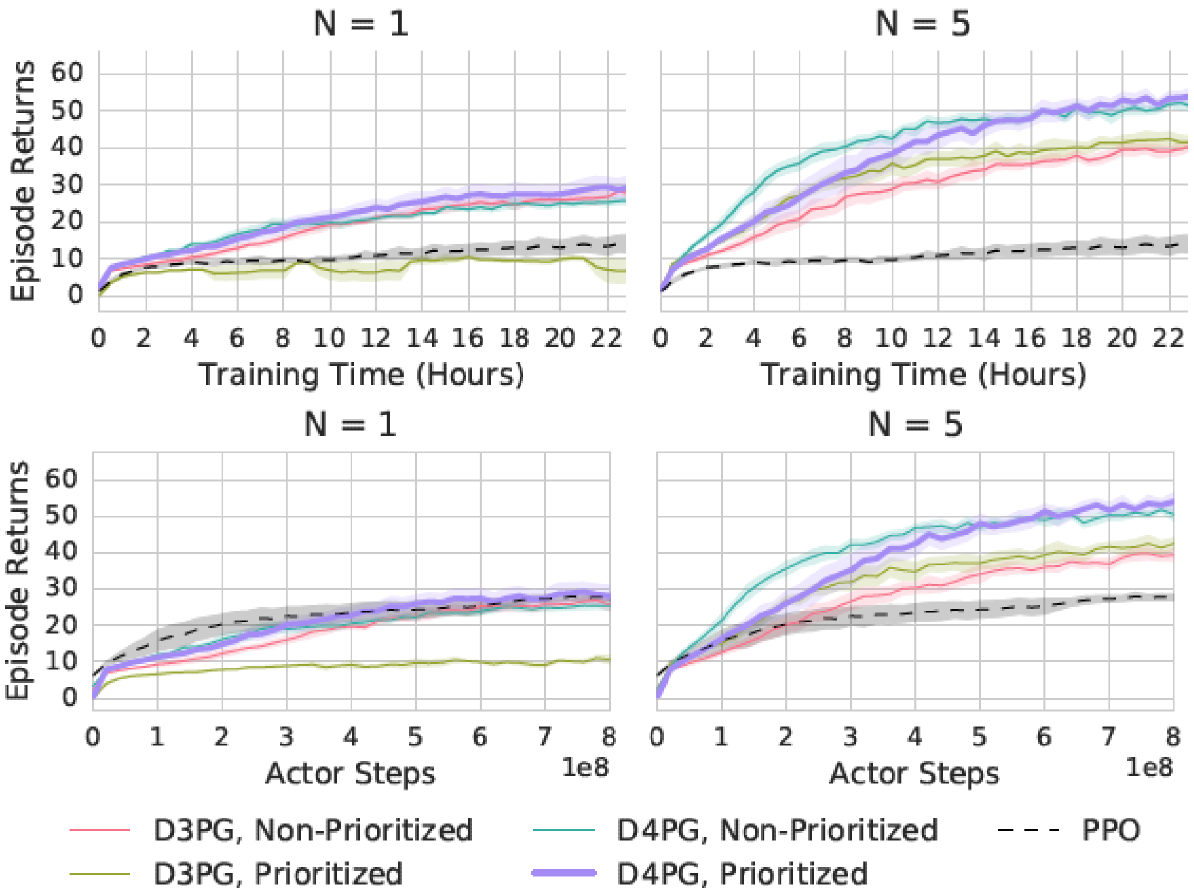
Architecture variants



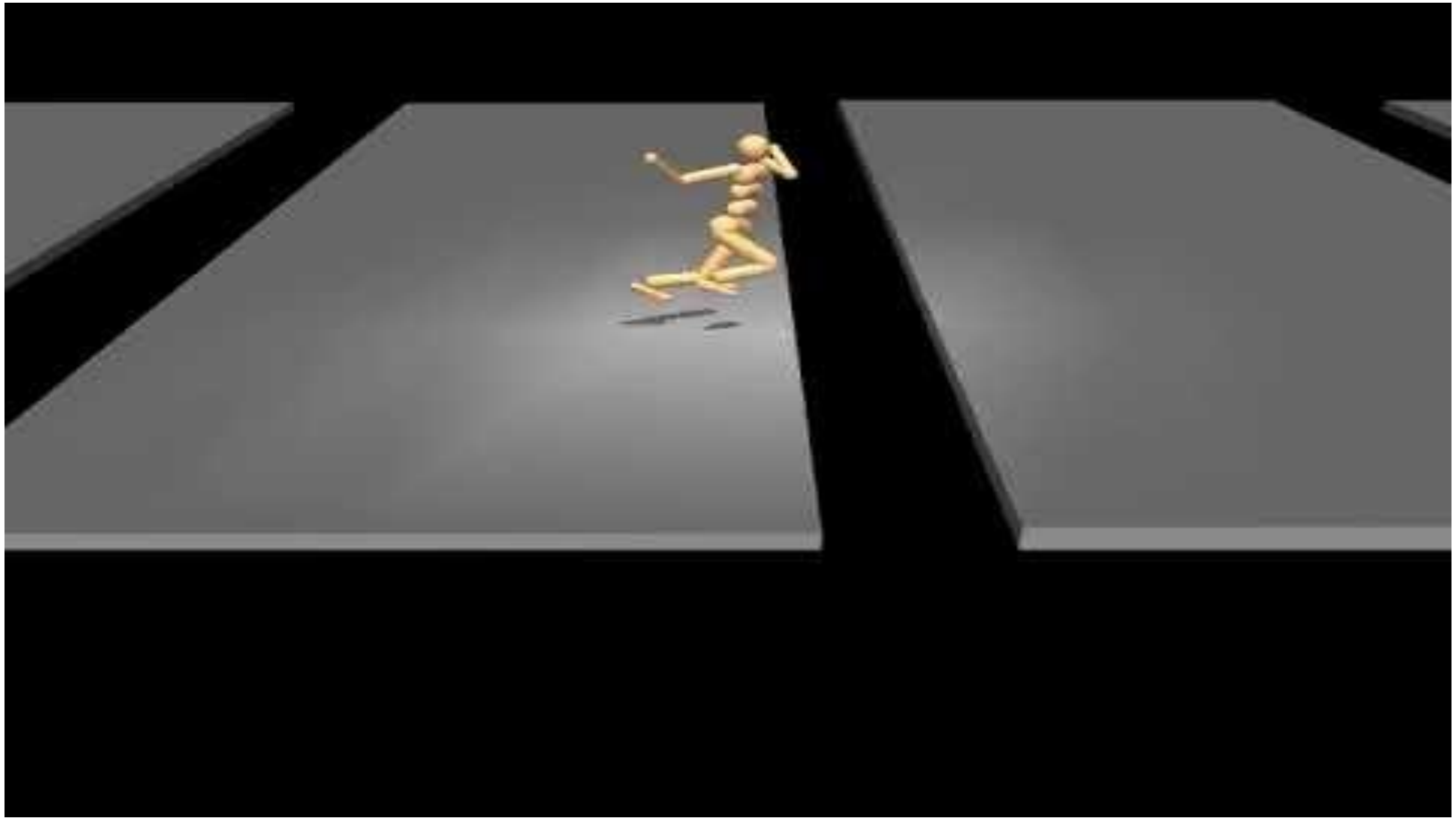
Results: Standard Control



Results: Parkour



Video



Thank you!

References

- *Horgan et al.2018.* Distributed prioritized experience replay.
- Barth-Maron et al. 2018. Distributed distributional deterministic policy gradients.
- Dean et al. 2012. *Large scale distributed deep networks*.
- Schaul et al. 2016. *Prioritized experience replay.*
- Van Hasselt et al. 2016. *Deep reinforcement learning with double Q-learning.*
- Mnih et al. 2016. *Asynchronous methods for deep reinforcement learning.*
- Wang et al. 2016. *Dueling network architectures for deep reinforcement learning.*
- Mnih et al. 2015. *Human-level control through deep reinforcement learning.*
- Bellemare et al. 2017. *A distributional perspective on reinforcement learning.*