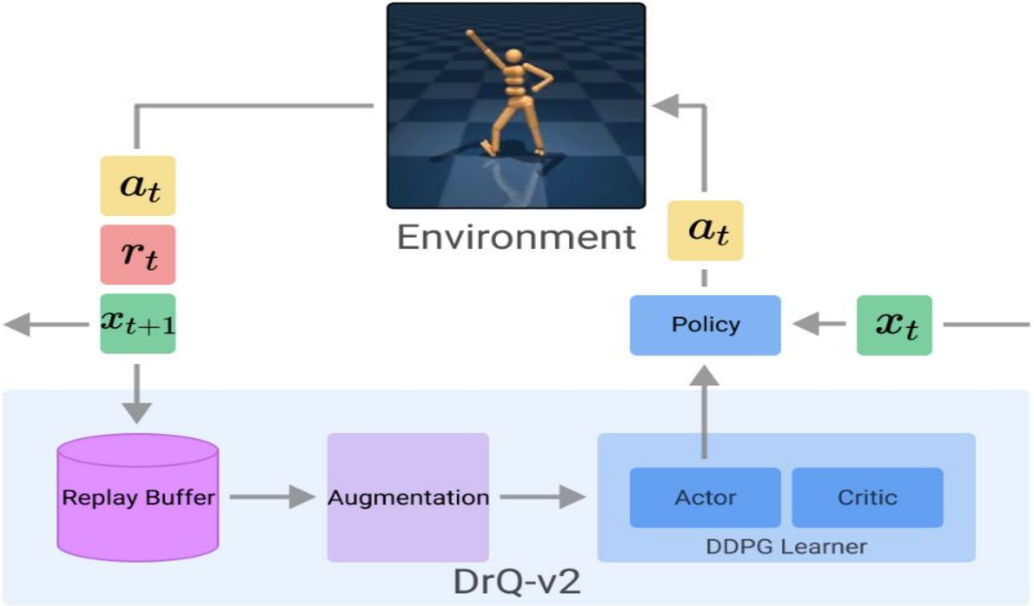



# Mastering Visual Continuous Control: Improved Data-Augmented Reinforcement Learning

(facebook, 2021.7)

Hard game playing
Using pixels
On 1 GPU

# Introduction

1. 고차원의 image observation으로 sample-efficient한 continuous control이 지난 3년간 RL의 도전과제였음
2. auto-encoder, variational inference, contrastive learning, self-prediction, data augmentations의 방법론이 있었음. 그러나 3가지 한계점 존재
   a. 어려운 task(ex.quadruped, humanoid)는 못 품
   b. 어마어마한 계산량
   c. 시스템 성능에 영향을 주는 디자인 설계의 모호함
3. DrQ-v2는 data augmentation을 이용한 model-free 알고리즘으로 어려운 visual control 문제를 해결
4. better sample efficiency and performance than DreamerV2

# Introduction

DrQ-v1에서 변경사항

1. Base RL algorithm SAC => DDPG(allow multi-step return)
2. Adding bilinear interpolation to the random shift image augmentation
3. better hyp-params including a larger capacity of the replay buffer

# Background

## image-based control as infinite-horizon MDP

a. pixel 정보만으로 system’s underlying state를 표현하기 부족=> 3 stack frame

$$(\mathcal{X}, \mathcal{A}, P, R, \gamma, d_0)$$

X : state space (a three-stack of image observations)
A : action space
P : the transition function. X * A -> X’
R : X * A -> [0,1]. reward function
r : [0,1). discount factor
d0 : distribution of initial state x0

# Method

1. Image Augmentation : random shift $\mathrm{aug}(\boldsymbol{x})$

2. Image Encoder : augmented image → conv encoder→ low-dim vector $\boldsymbol{h} = f_\xi(\mathrm{aug}(\boldsymbol{x}))$

same encoder

3. Actor-Critic Algorithm : DDPG as backbone, clipped double Q-learning

$$\boldsymbol{a}_t = \pi_\phi(\boldsymbol{h}_t) + \epsilon, \text{ and } \epsilon \sim \mathrm{clip}(\mathcal{N}(0, \sigma^2), -c, c)$$

$$\mathcal{L}_{\theta_k,\xi}(\mathcal{D}) = \mathbb{E}_{\tau\sim\mathcal{D}}\big[(Q_{\theta_k}(\boldsymbol{h}_t, \boldsymbol{a}_t) - y)^2\big] \quad \forall k \in \{1, 2\},$$

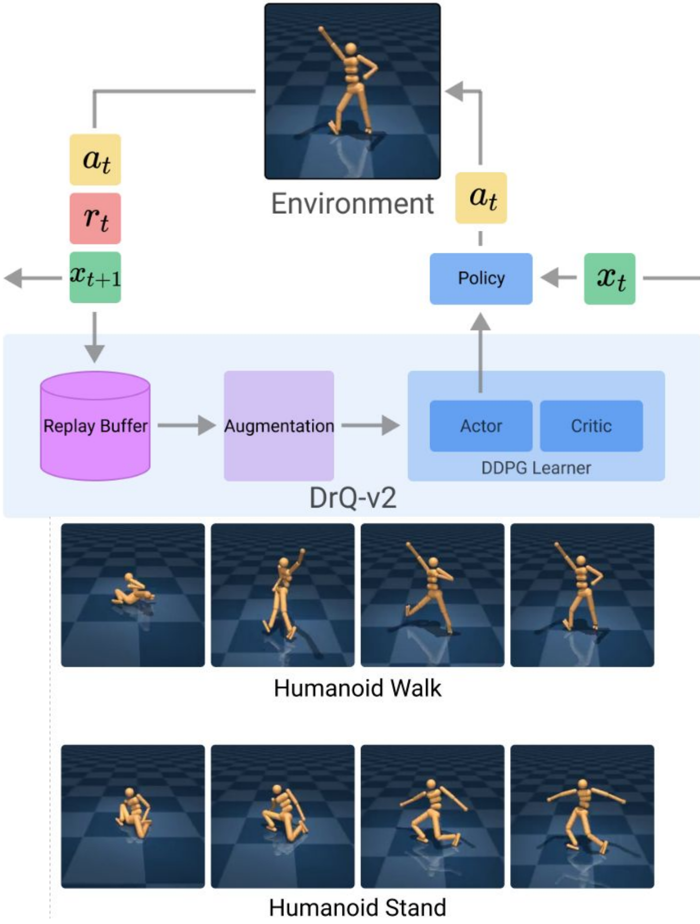
$$y = \sum_{i=0}^{n-1} \gamma^i r_{t+i} + \gamma^n \min_{k=1,2} Q_{\bar{\theta}_k}(\boldsymbol{h}_{t+n}, \boldsymbol{a}_{t+n}), \quad \boldsymbol{h}_{t+n} = f_\xi(\mathrm{aug}(\boldsymbol{x}_{t+n}))$$

4. Scheduled Exploration Noise $\sigma(t) = \sigma_{\text{init}} + (1 - \min(\frac{t}{T}, 1))(\sigma_{\text{final}} - \sigma_{\text{init}})$

5. Key Hyp-Params Change : 10 x larger replay buffer. 256 minibatch, learning rate

**Algorithm 1** DrQ-v2: Improved data-augmented RL.

**Inputs:**
$f_\xi, \pi_\phi, Q_{\theta_1}, Q_{\theta_2}$: parametric networks for encoder, policy, and Q-functions respectively.
aug: random shifts image augmentation.
$\sigma(t)$: scheduled standard deviation for the exploration noise defined in Equation (3).
$T$, $B$, $\alpha$, $\tau$, $c$: training steps, mini-batch size, learning rate, target update rate, clip value.
**Training routine:**
**for** each timestep $t = 1..T$ **do**
    $\sigma_t \leftarrow \sigma(t)$ ▷ Compute stddev for the exploration noise
    $\boldsymbol{a}_t \leftarrow \pi_\phi(f_\xi(\boldsymbol{x}_t)) + \epsilon$ and $\epsilon \sim \mathcal{N}(0, \sigma_t^2)$ ▷ Add noise to the deterministic action
    $\boldsymbol{x}_{t+1} \sim P(\cdot|\boldsymbol{x}_t, \boldsymbol{a}_t)$ ▷ Run transition function for one step
    $\mathcal{D} \leftarrow \mathcal{D} \cup (\boldsymbol{x}_t, \boldsymbol{a}_t, R(\boldsymbol{x}_t, \boldsymbol{a}_t), \boldsymbol{x}_{t+1})$ ▷ Add a transition to the replay buffer
    UPDATECRITIC($\mathcal{D}, \sigma_t$)
    UPDATEACTOR($\mathcal{D}, \sigma_t$)
**end for**
**procedure** UPDATECRITIC($\mathcal{D}, \sigma$)
    $\{(\boldsymbol{x}_t, \boldsymbol{a}_t, r_{t:t+n-1}, \boldsymbol{x}_{t+n})\} \sim \mathcal{D}$ ▷ Sample a mini batch of $B$ transitions
    $\boldsymbol{h}_t, \boldsymbol{h}_{t+n} \leftarrow f_\xi(\text{aug}(\boldsymbol{x}_t)), f_\xi(\text{aug}(\boldsymbol{x}_{t+n}))$ ▷ Apply data augmentation and encode
    $\boldsymbol{a}_{t+n} \leftarrow \pi_\phi(\boldsymbol{h}_{t+n}) + \epsilon$ and $\epsilon \sim \text{clip}(\mathcal{N}(0, \sigma^2))$ ▷ Sample action
    Compute $\mathcal{L}_{\theta_1,\xi}$ and $\mathcal{L}_{\theta_2,\xi}$ using Equation (1) ▷ Compute critic losses
    $\xi \leftarrow \xi - \alpha \nabla_\xi (\mathcal{L}_{\theta_1,\xi} + \mathcal{L}_{\theta_2,\xi})$ ▷ Update encoder weights
    $\theta_k \leftarrow \theta_k - \alpha \nabla_{\theta_k} \mathcal{L}_{\theta_k,\xi} \quad \forall k \in \{1, 2\}$ ▷ Update critic weights
    $\bar{\theta}_k \leftarrow (1-\tau)\bar{\theta}_k + \tau\theta_k \quad \forall k \in \{1, 2\}$ ▷ Update critic target weights
**end procedure**
**procedure** UPDATEACTOR($\mathcal{D}, \sigma$)
    $\{(\boldsymbol{x}_t)\} \sim \mathcal{D}$ ▷ Sample a mini batch of $B$ observations
    $\boldsymbol{h}_t \leftarrow f_\xi(\text{aug}(\boldsymbol{x}_t))$ ▷ Apply data augmentation and encode
    $\boldsymbol{a}_t \leftarrow \pi_\phi(\boldsymbol{h}_t) + \epsilon$ and $\epsilon \sim \text{clip}(\mathcal{N}(0, \sigma^2))$ ▷ Sample action
    Compute $\mathcal{L}_\phi$ using Equation (2) ▷ Compute actor loss
    $\phi \leftarrow \phi - \alpha \nabla_\phi \mathcal{L}_\phi$ ▷ Update actor's weights only
**end procedure**



# Implementation Details

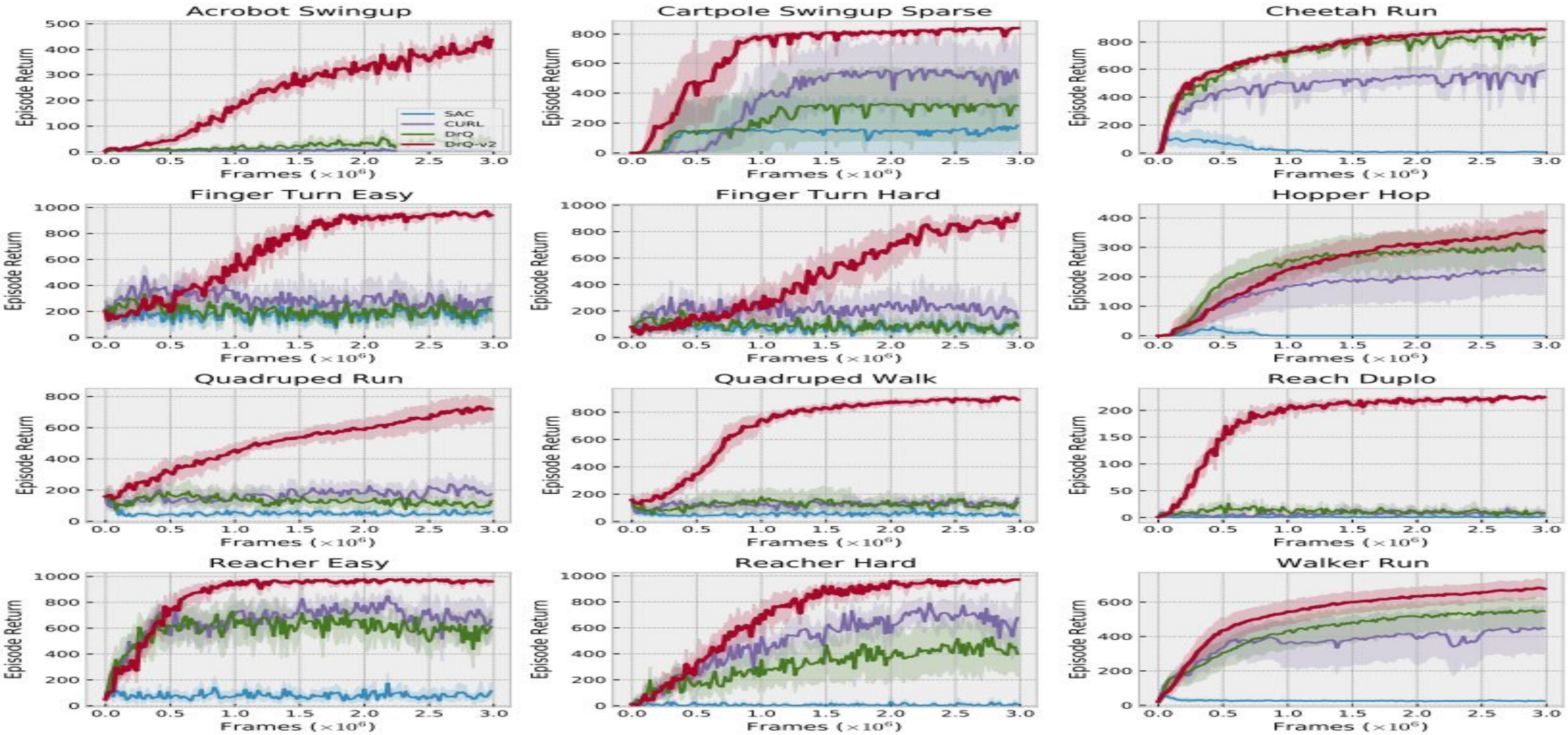
1. Faster Image Augmentation
   a. DrQv1에서는 kornia.augmentation.RandomCrop을 썼는데, GPU ←→ CPU전환과정이 있어서 bottle neck이 생겨 Pytorch의 grid_sample을 썼다. + bilinear interpolation
2. Faster Replay Buffer : 10 x storage capacity and faster

```
class ReplayBufferStorage:...

class ReplayBuffer(IterableDataset):...
```

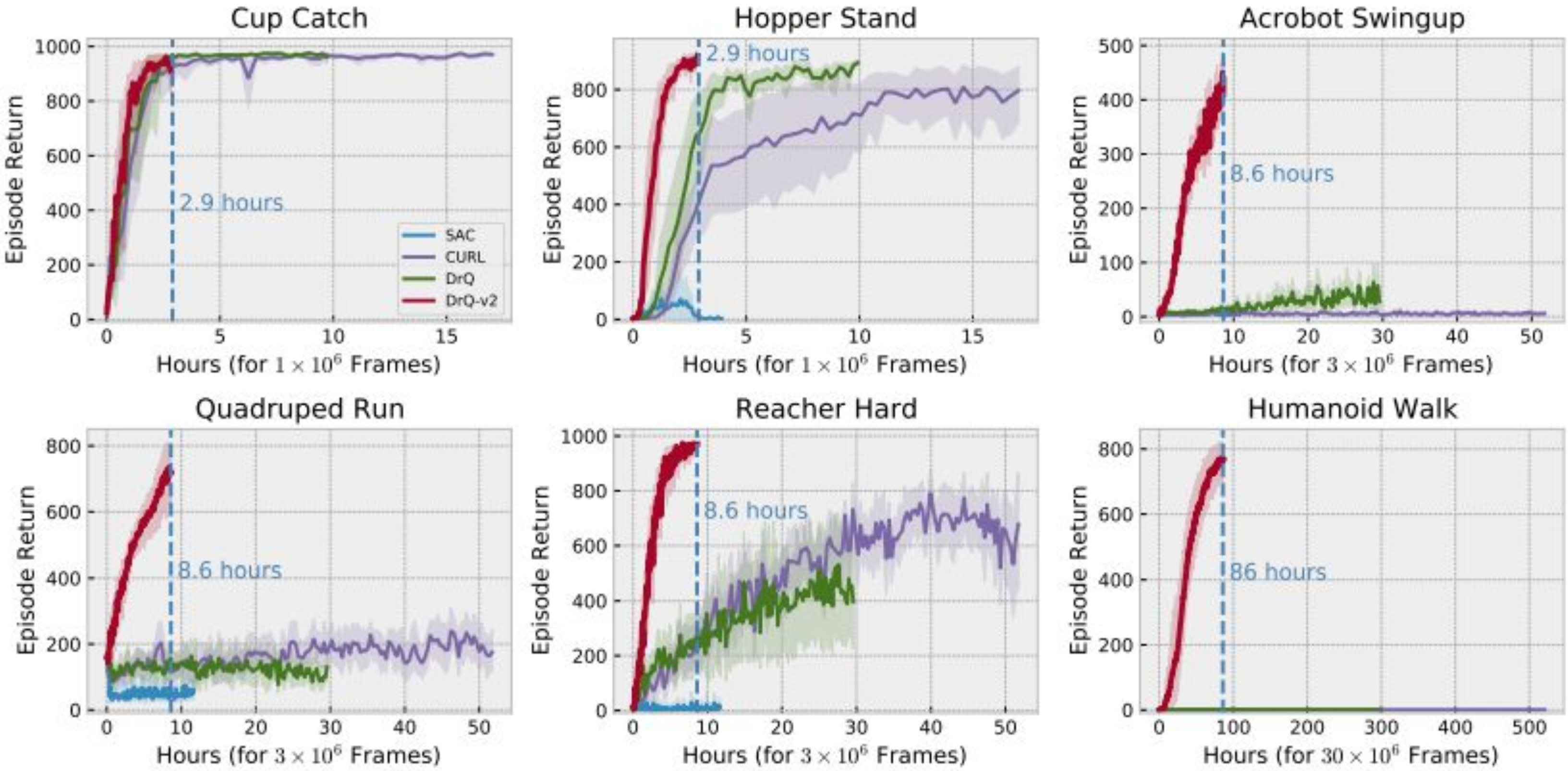
```
def replay_iter(self):
    if self._replay_iter is None:
        self._replay_iter = iter(self.replay_loader)
    return self._replay_iter
```

```
self.replay_storage = ReplayBufferStorage(data_specs,
                                          self.work_dir / 'buffer')

self.replay_loader = make_replay_loader(
    self.work_dir / 'buffer', self.cfg.replay_buffer_size,
    self.cfg.batch_size, self.cfg.replay_buffer_num_workers,
    self.cfg.save_snapshot, self.cfg.nstep, self.cfg.discount)
```

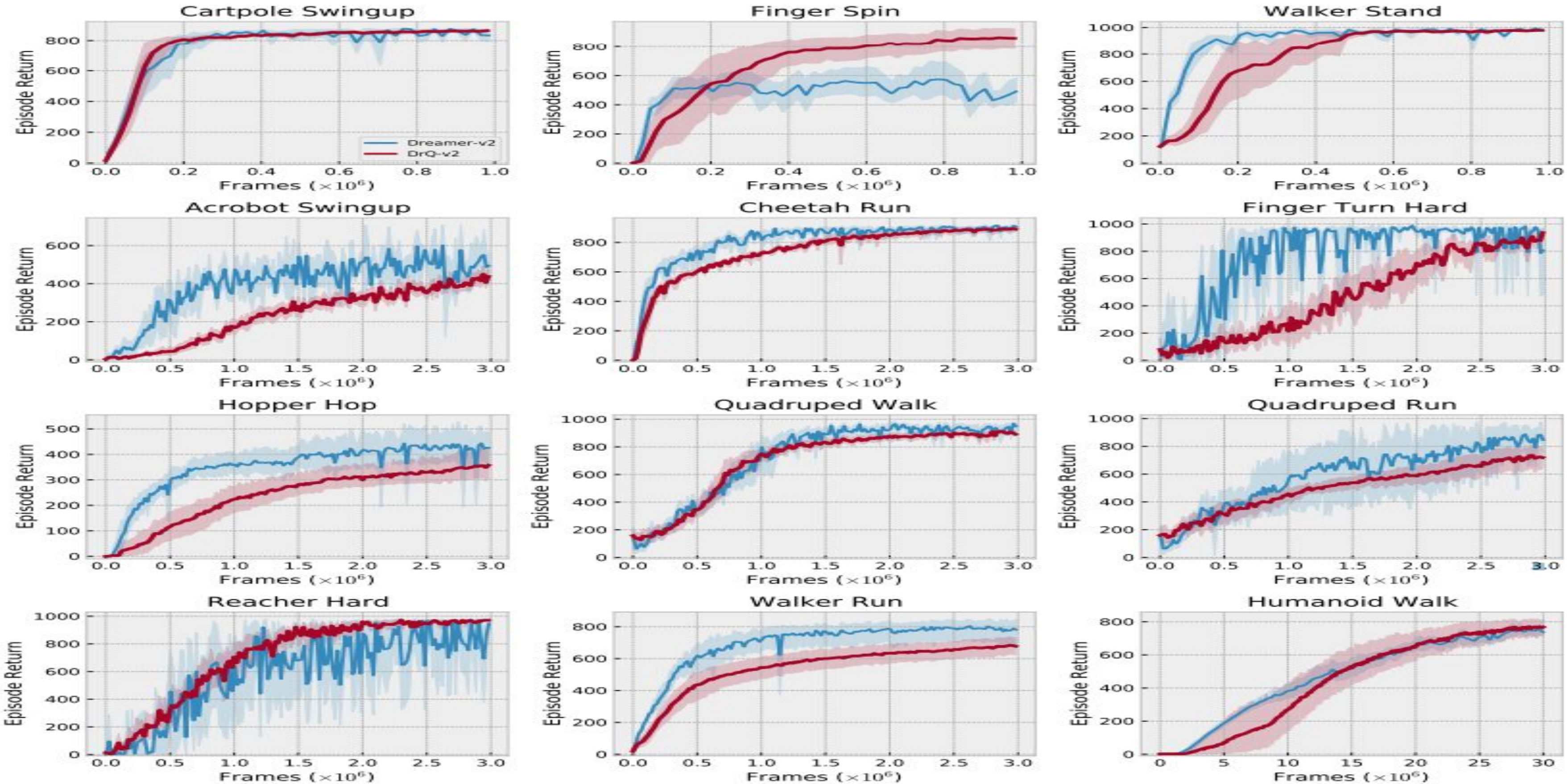
# Comparison to Model-Free Methods



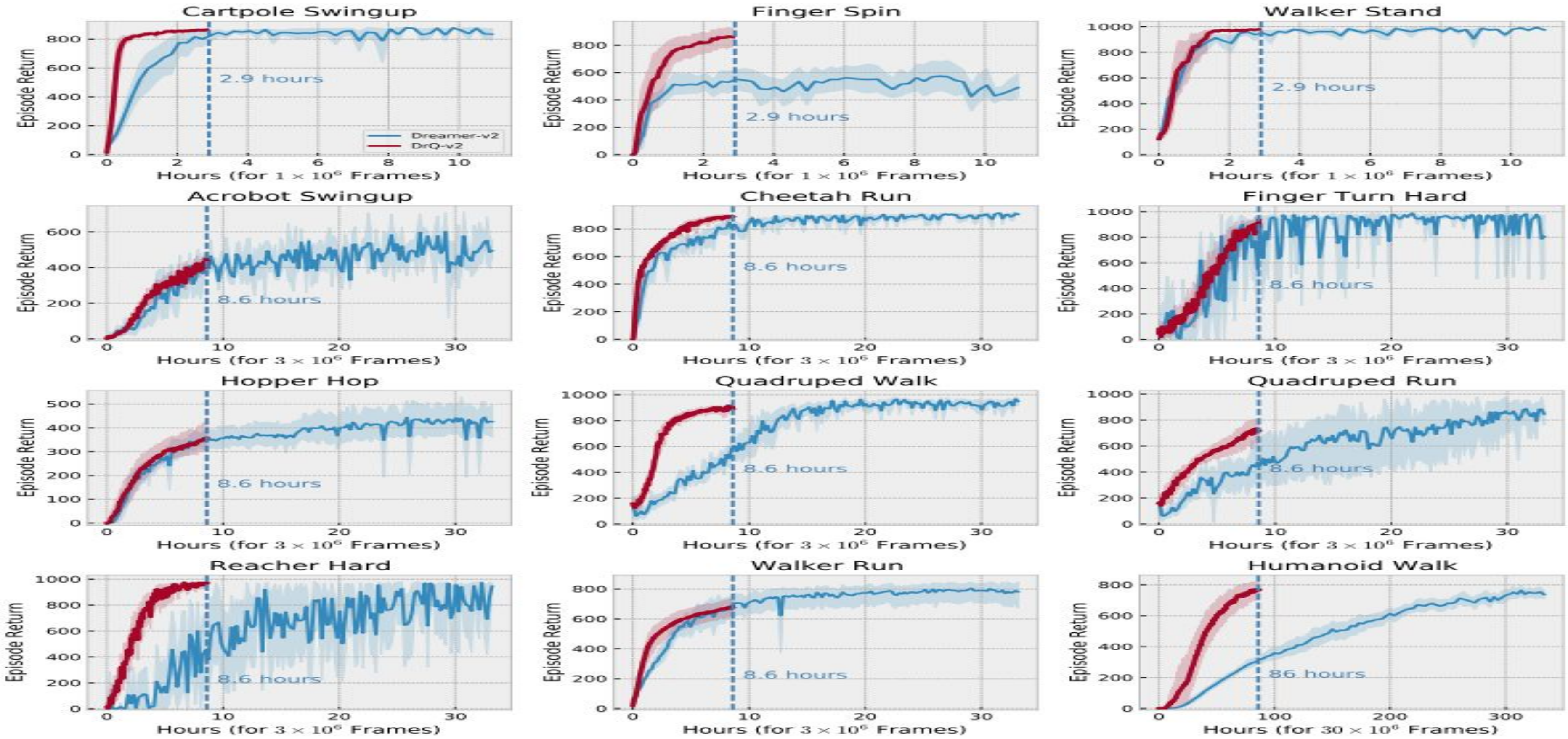
# Comparison to Model-Free Methods



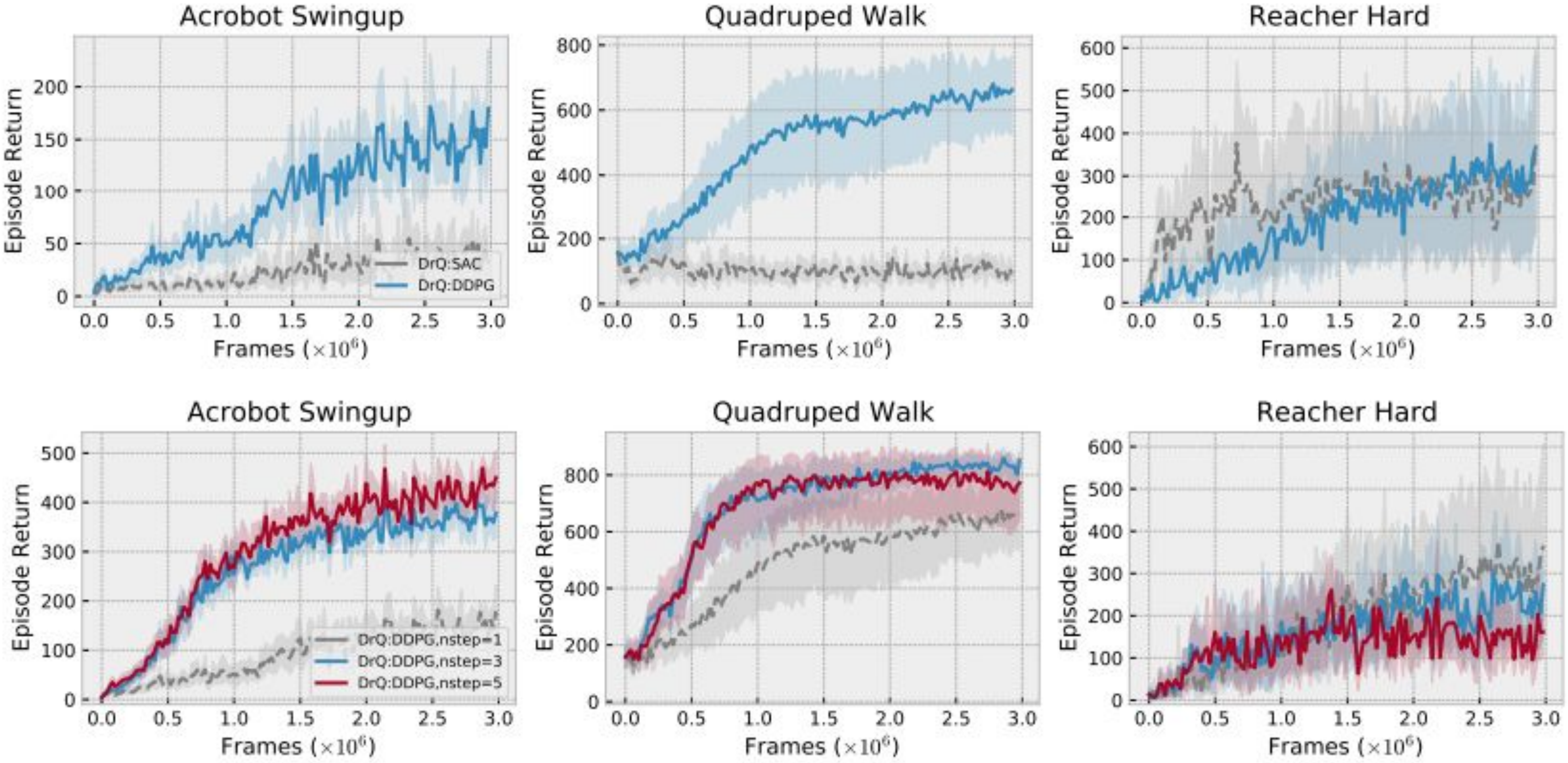
# Comparison to Model-Based Methods



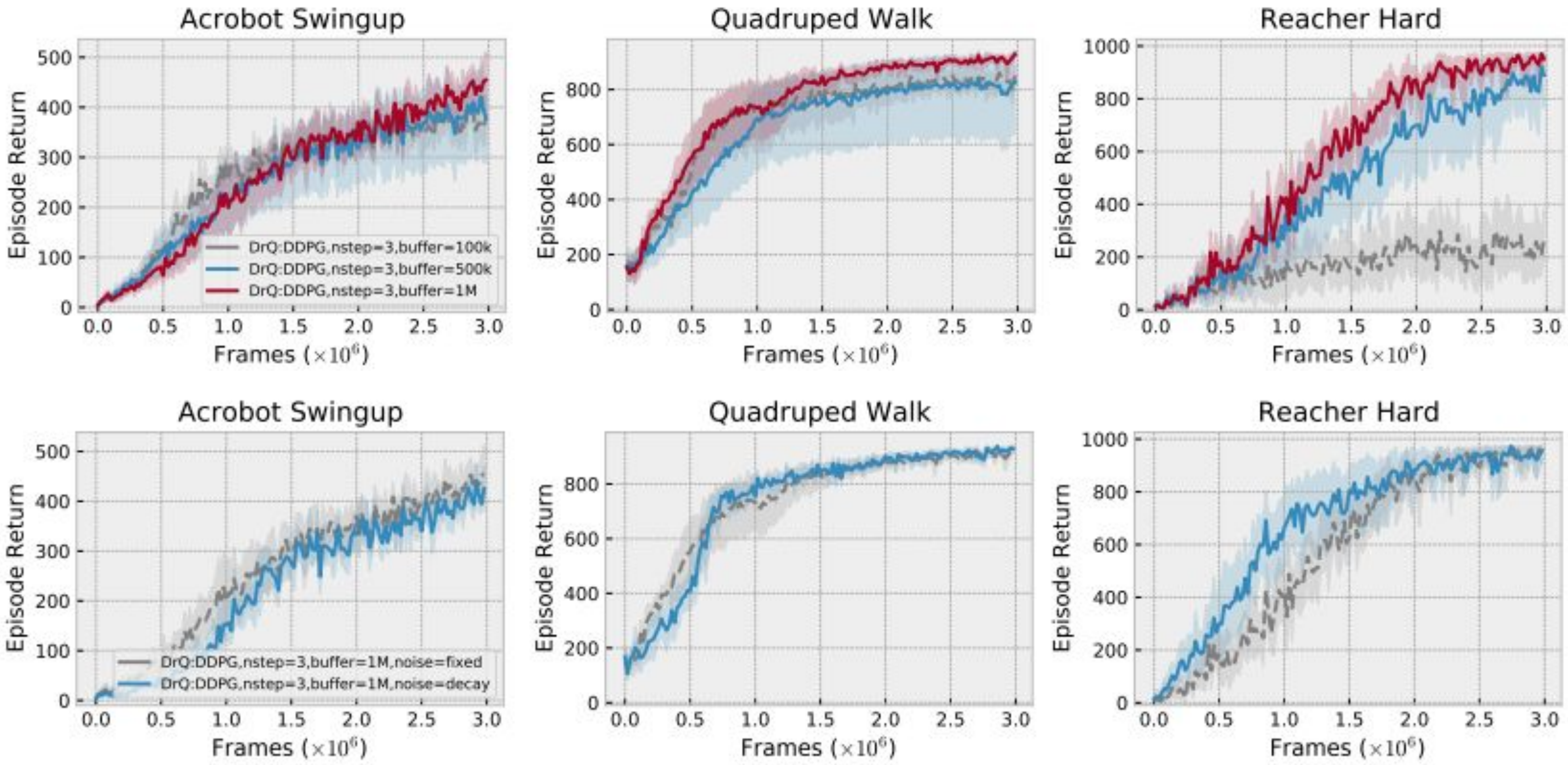
# Comparison to Model-Based Methods



# Ablation(SAC-DDPG, N-step)



# Ablation(Buffer size, noise exploration)



# Conclusion

- simple model-free actor-critic RL algorithm for image-based continuous control – DrQ-v2
- significantly better computational footprint and masters tasks from DMC directly from pixels
- efficient PyTorch implementation

https:// github.com/facebookresearch/drqv2.

# 나도 한번 돌려보자~

But, Window10이 또... https://github.com/facebookresearch/drqv2/issues/12

**denisyarats** commented yesterday Contributor

Hi,

I personally haven't tested on Window 10 and don't have access to it, so not sure I can help. But you should seek out help here https://github.com/deepmind/, since it is a dm_control related issue, not drqv2.

dm_control에서 rendering하는 platform을 'egl'을 쓰는데, 경로를 찾지 못한다고...

```
  File "C:\Users\KANG\anaconda3\envs\drqv2\lib\site-packages\OpenGL\platform\egl.py", line 73, in EGL
    raise ImportError("Unable to load EGL library", *err.args)
ImportError: ('Unable to load EGL library', "Could not find module 'EGL' (or one of its dependencies). Try using the full path with constructor syntax.", 'EGL', None)
```

# Q&A

감사합니다