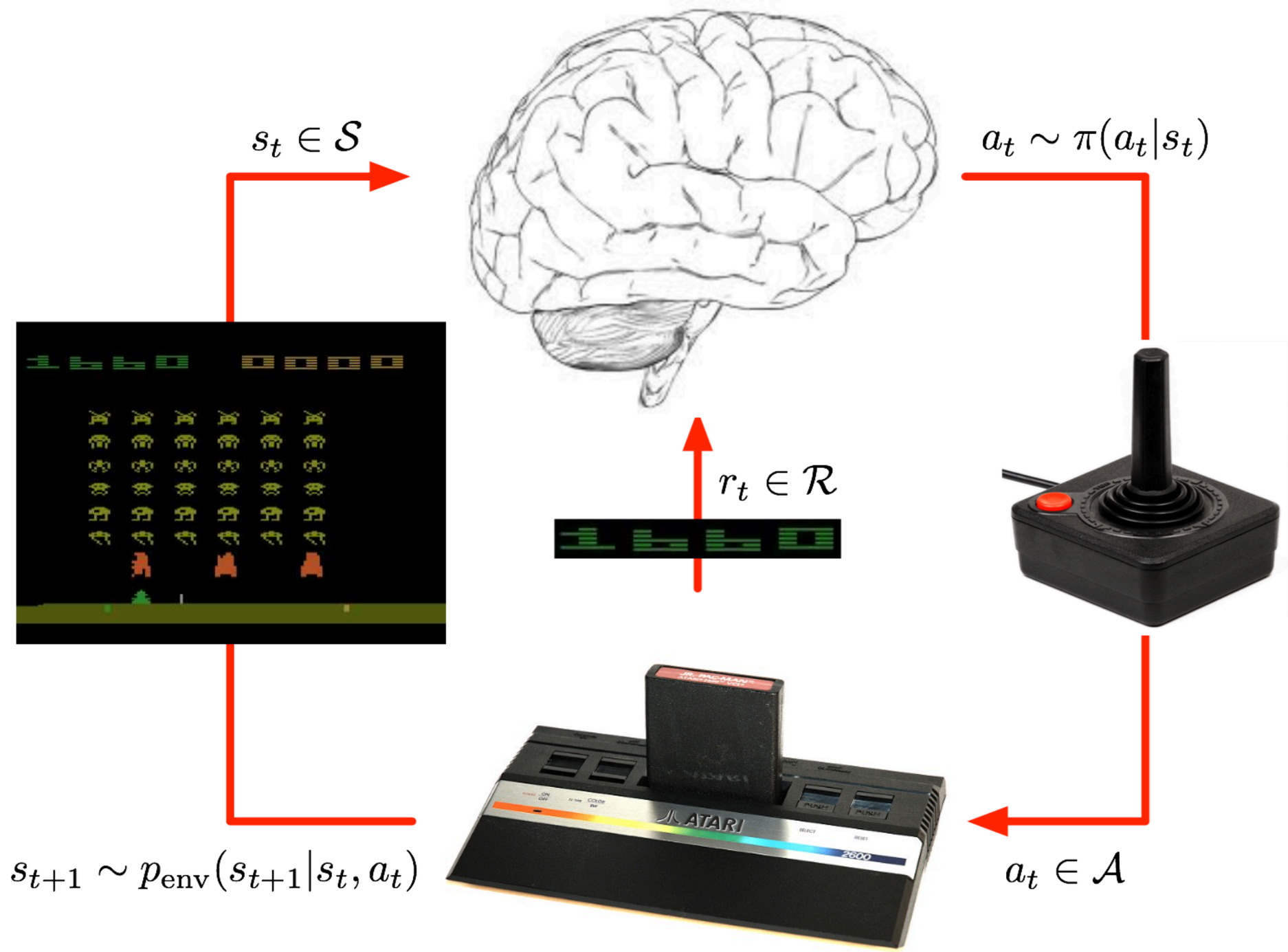
Distributional Policy Gradients

Alexander Novikov

Reinforcement Learning



Q-values

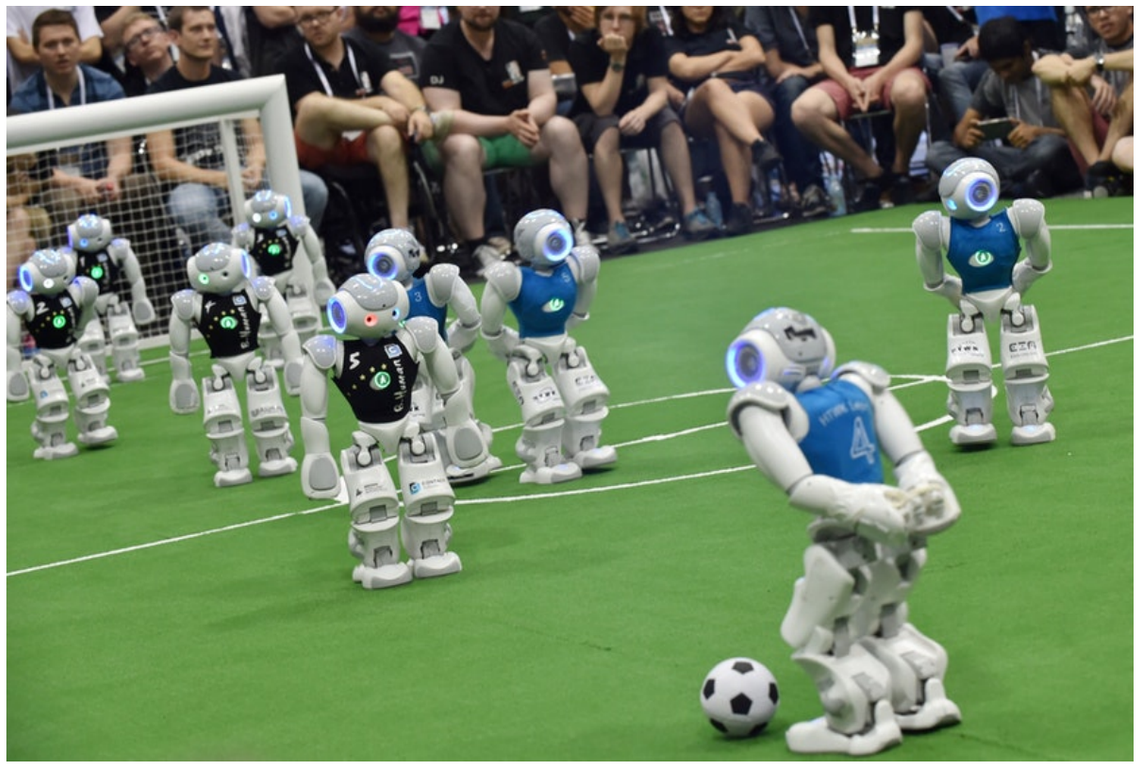
Q-function: future reward if doing action a

$$Q^*(s_t, a_t) = \max_{\pi} \mathbb{E}_{s_{t+1}, a_{t+1}, s_{t+2}, \ldots | \pi, p, s_t, a_t} \left[\mathbb{E}_{r_t | s_t} r_t + \mathbb{E}_{r_{t+1} | s_{t+1}} \gamma r_{t+1} + \ldots \right]$$

Q-values

Q-function: future reward if doing action a

$$Q^*(s_t, a_t) = \max_{\pi} \mathbb{E}_{s_{t+1}, a_{t+1}, s_{t+2}, \ldots | \pi, p, s_t, a_t} \left[\mathbb{E}_{r_t | s_t} r_t + \mathbb{E}_{r_{t+1} | s_{t+1}} \gamma r_{t+1} + \ldots \right]$$



$$Q^*(s_t, a_t = a^0) = -3 \qquad Q^*(s_t, a_t = a^1) = 0.1$$

Q-values

Q-function: future reward if doing action a

$$Q^*(s_t, a_t) = \max_{\pi} \mathbb{E}_{s_{t+1}, a_{t+1}, s_{t+2}, \ldots | \pi, p, s_t, a_t} \left[\mathbb{E}_{r_t | s_t} r_t + \mathbb{E}_{r_{t+1} | s_{t+1}} \gamma r_{t+1} + \ldots \right]$$

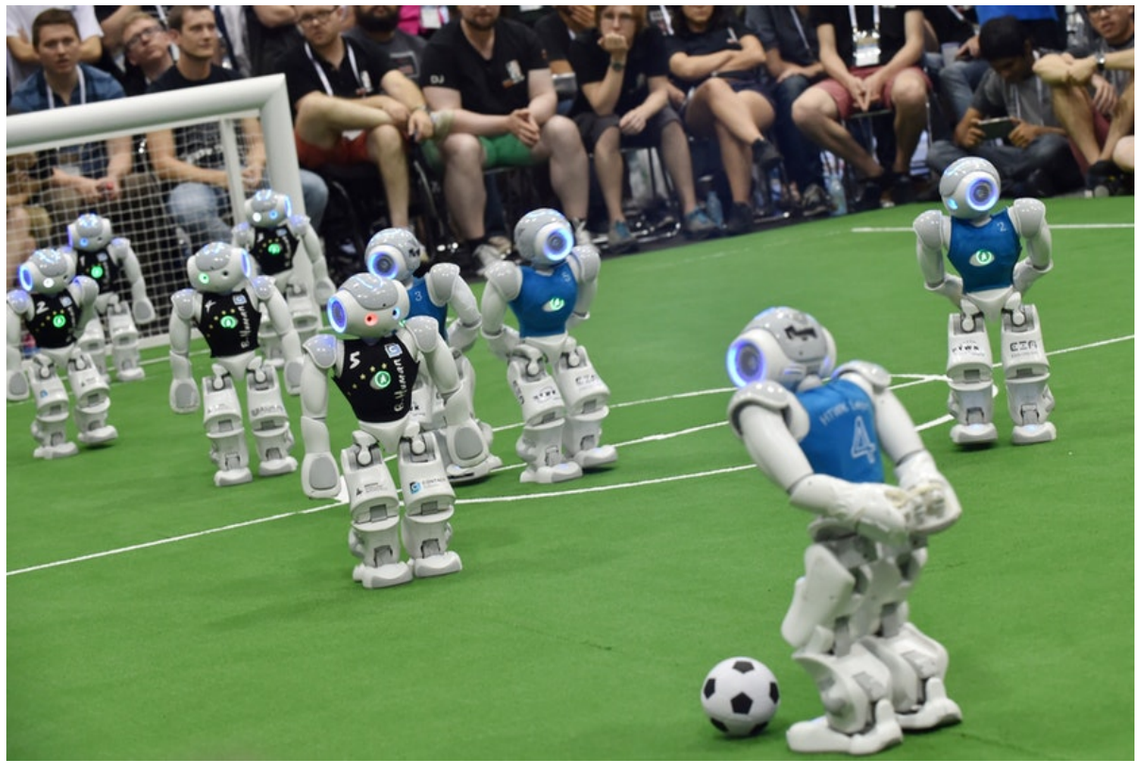
Dynamic programming

$$Q^*(s_t, a_t) = \mathbb{E}_{r_t | s_t} r_t + \gamma \mathbb{E}_{s_{t+1}} \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})$$

Training

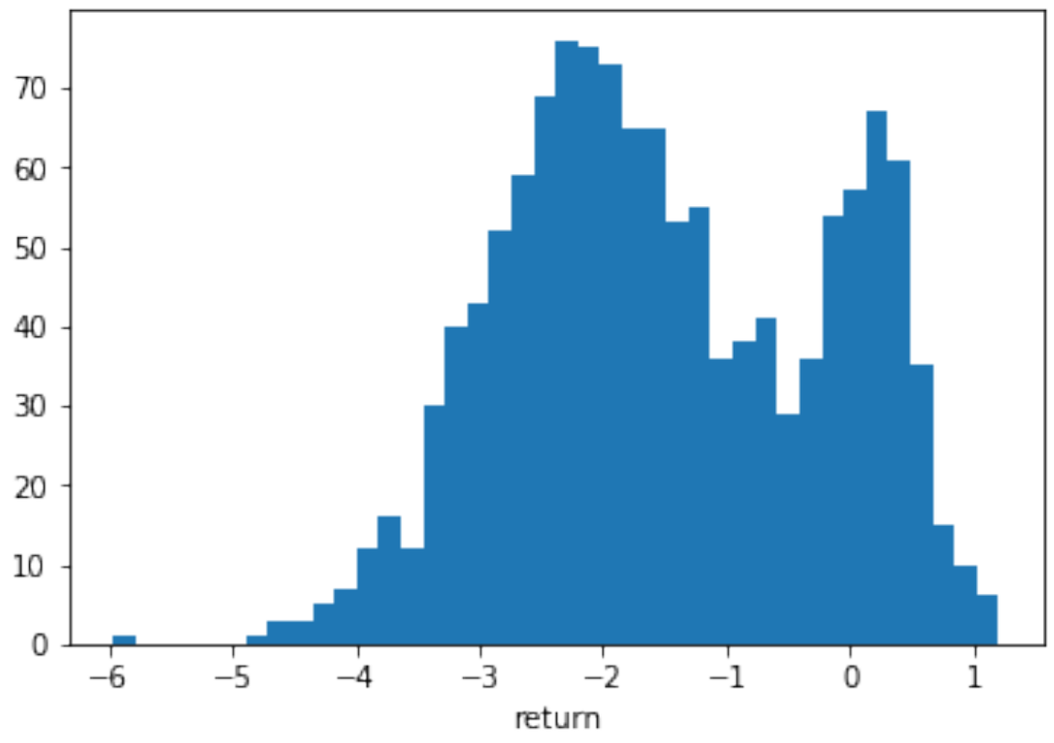
$$\min_{\theta} \mathbb{E}_{s_t, a_t, r_t, s_{t+1}} \left[r + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}, \theta) - Q(s_t, a_t, \theta) \right]^2$$

Distributional perspective



$$Z^*(s_t, a_t = a^1) =$$

$$Q^*(s_t, a_t = a^1) = \mathbb{E}Z^*(s_t, a_t = a^1) = -1$$



Training

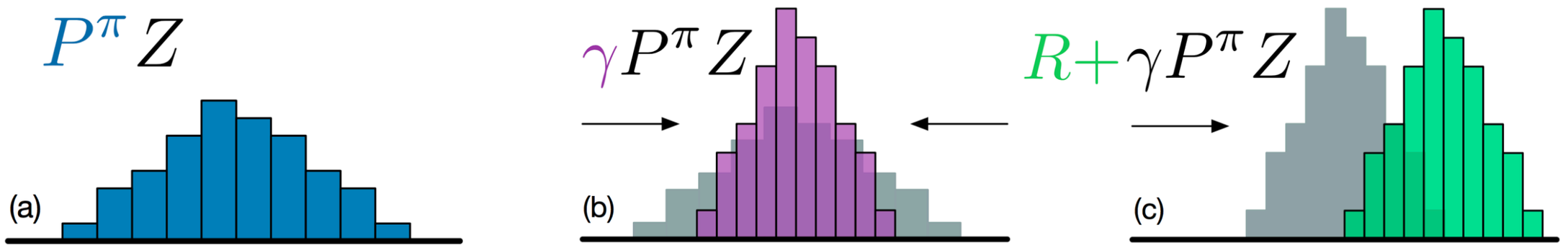
Usual

$$\min_{\theta} \mathbb{E}_{s_t, a_t, r_t, s_{t+1}} \left[r + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}, \theta) - Q(s_t, a_t, \theta) \right]^2$$

Distributional

$$\min_{\theta} \mathbb{E}_{s_t, a_t, r_t, s_{t+1}} \mathcal{KL}\left(Z(s_t, a_t, \theta) \,\|\, r + \gamma Z(s_{t+1}, a^*), \theta) \right)$$

$$a^* = \arg\max_{a} \mathbb{E} Z(s, a, \theta)$$



D4PG

- Same idea, but for continuous control. Q-updates like in distributional DQN, other parts use return expectation

$$\min_{\theta} \mathbb{E}_{s_t, a_t, r_t, s_{t+1}} \mathcal{KL}\left(Z(s_t, a_t, \theta) \,\|\, r + \gamma Z(s_{t+1}, \pi(s_{t+1}), \theta)\right)$$

D4PG

- Same idea, but for continuous control. Q-updates like in distributional DQN, other parts use return expectation

$$\min_{\theta} \mathbb{E}_{s_t, a_t, r_t, s_{t+1}} \mathcal{KL}\left(Z(s_t, a_t, \theta) \,\|\, r + \gamma Z(s_{t+1}, \pi(s_{t+1}), \theta)\right)$$

$$\begin{aligned}
\nabla_\theta J(\theta) &\approx \mathbb{E}\left[\nabla_\theta \pi_\theta(s) \nabla_a Q(s, a, \theta)|_{a=\pi(s)}\right] \\
&= \mathbb{E}\left[\nabla_\theta \pi_\theta(s) \nabla_a (\mathbb{E} Z(s, a, \theta))|_{a=\pi(s)}\right]
\end{aligned}$$

D4PG

- Same idea, but for continuous control. Q-updates like in distributional DQN, other parts use return expectation

- N step returns (not rigor!)

$$\sum_{n=0}^{N-1} \gamma^n r_{t+n} + \gamma^N Z\big(s_{t+N}, \pi\big(s_{t+N}\big)\big)$$

instead of

$$r_t + \gamma Z\big(s_{t+1}, \pi\big(s_{t+1}\big)\big)$$

D4PG

- Same idea, but for continuous control. Q-updates like in distributional DQN, other parts use return expectation

- N step returns

- Prioritized experience replay

- Distributed (on a cluster of servers)

Some results

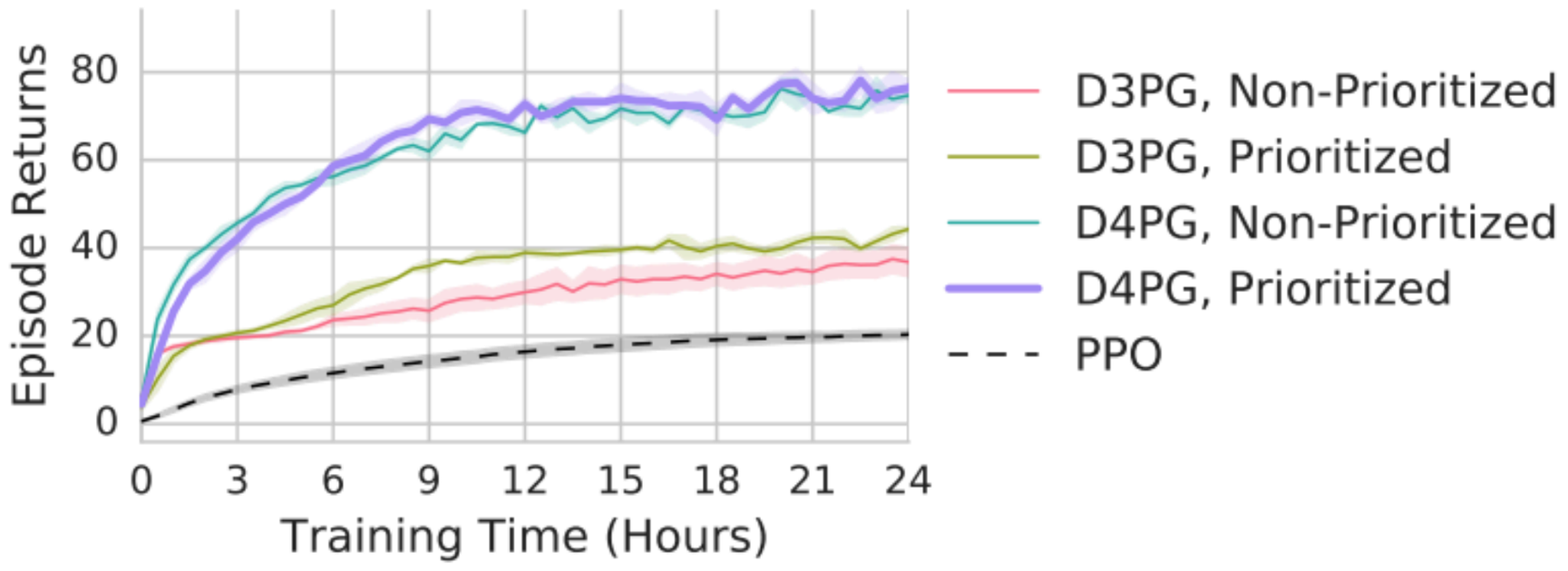


Figure 6: Experimental results for the three-dimensional (humanoid) parkour domain.

Summary

- Applying distributional perspective idea to continuous control
- A few tricks to make it work
- Good results and minor code modifications (except for cluster version)
- Less sensitive to hyperparams