PAAC, DDPG, TD3, SAC

Milan Straka

April 08, 2024

EUROPEAN UNION
European Structural and Investment Fund
Operational Programme Research,
Development and Education

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Parallel Advantage Actor Critic

An alternative to independent workers is to train in a synchronous and centralized way by having the workers to only generate episodes. Such approach was described in 2017 as **parallel advantage actor-critic** (PAAC) by [Clemente et al., https://arxiv.org/abs/1705.04862](https://arxiv.org/abs/1705.04862).

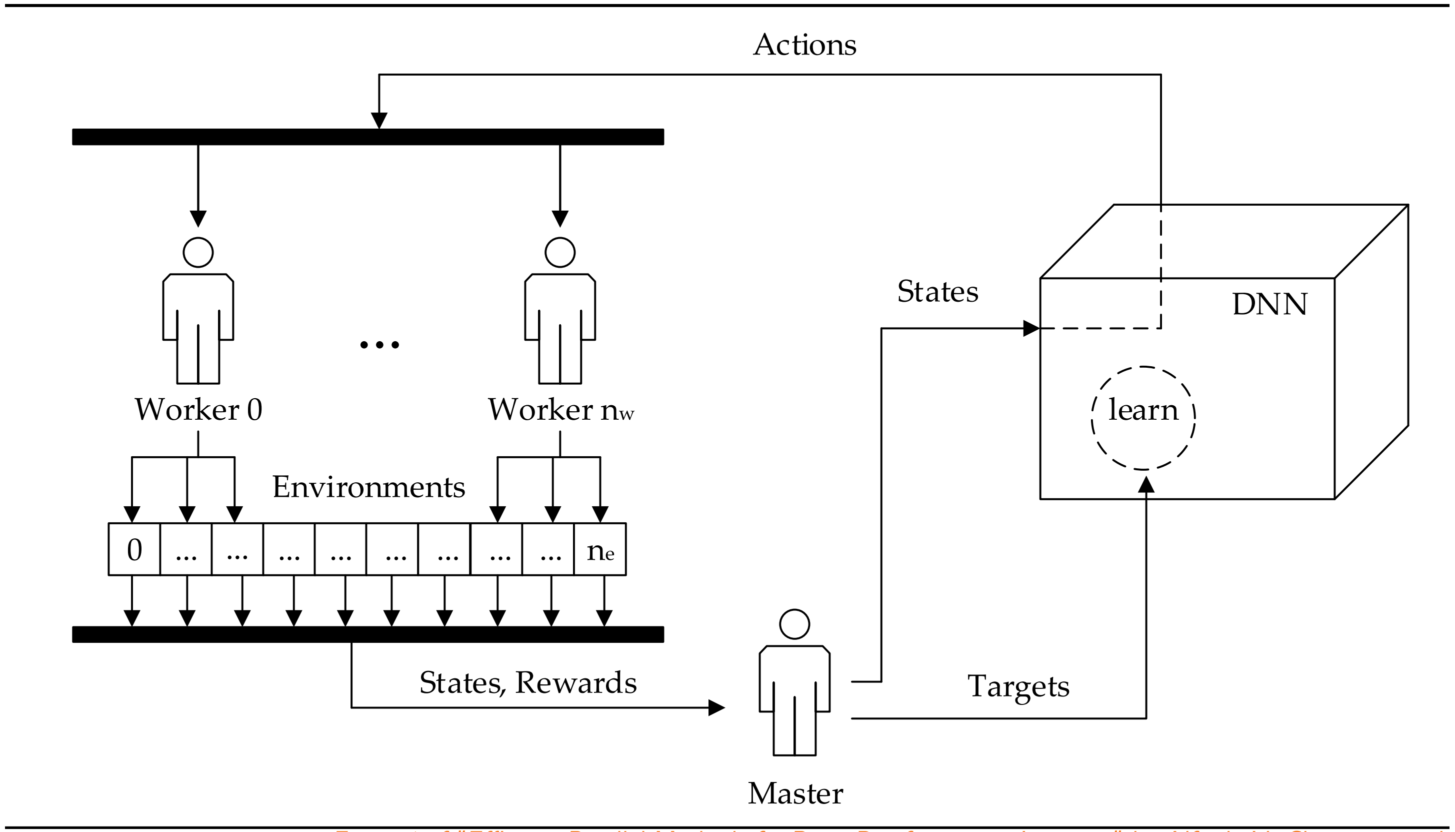


Figure 1 of "Efficient Parallel Methods for Deep Reinforcement Learning" by Alfredo V. Clemente et al.

Parallel Advantage Actor Critic

Algorithm 1 Parallel advantage actor-critic

```
 1: Initialize timestep counter $N = 0$ and network weights $\theta$, $\theta_v$
 2: Instantiate set $\boldsymbol{e}$ of $n_e$ environments
 3: repeat
 4:     for $t = 1$ to $t_{max}$ do
 5:         Sample $\boldsymbol{a}_t$ from $\pi(\boldsymbol{a}_t|\boldsymbol{s}_t;\theta)$
 6:         Calculate $\boldsymbol{v}_t$ from $V(\boldsymbol{s}_t;\theta_v)$
 7:         parallel for $i = 1$ to $n_e$ do
 8:             Perform action $a_{t,i}$ in environment $e_i$
 9:             Observe new state $s_{t+1,i}$ and reward $r_{t+1,i}$
10:         end parallel for
11:     end for
12:     $\boldsymbol{R}_{t_{\max}+1} = \begin{cases} 0 & \text{for terminal } \boldsymbol{s}_t \\ V(s_{t_{\max}+1};\theta) & \text{for non-terminal } \boldsymbol{s}_t \end{cases}$
13:     for $t = t_{\max}$ down to 1 do
14:         $\boldsymbol{R}_t = \boldsymbol{r}_t + \gamma \boldsymbol{R}_{t+1}$
15:     end for
16:     $d\theta = \frac{1}{n_e \cdot t_{max}} \sum_{i=1}^{n_e} \sum_{t=1}^{t_{max}} (R_{t,i} - v_{t,i}) \nabla_\theta - \log \pi(a_{t,i}|s_{t,i};\theta) - \beta \nabla_\theta H(\pi(s_{e,t};\theta))$
17:     $d\theta_v = \frac{1}{n_e \cdot t_{max}} \sum_{i=1}^{n_e} \sum_{t=1}^{t_{max}} \nabla_{\theta_v} (R_{t,i} - V(s_{t,i};\theta_v))^2$
18:     Update $\theta$ using $d\theta$ and $\theta_v$ using $d\theta_v$.
19:     $N \leftarrow N + n_e \cdot t_{\max}$
20: until $N \geq N_{max}$
```

Modification of Algorithm 1 of "Efficient Parallel Methods for Deep Reinforcement Learning" by Alfredo V. Clemente et al.

Parallel Advantage Actor Critic

Game	Gorila	A3C FF	GA3C	PAAC arch$_{\text{nips}}$	PAAC arch$_{\text{nature}}$
Amidar	1189.70	263.9	218	701.8	1348.3
Centipede	8432.30	3755.8	7386	5747.32	7368.1
Beam Rider	3302.9	22707.9	N/A	4062.0	6844.0
Boxing	94.9	59.8	92	99.6	99.8
Breakout	402.2	681.9	N/A	470.1	565.3
Ms. Pacman	3233.50	653.7	1978	2194.7	1976.0
Name This Game	6182.16	10476.1	5643	9743.7	14068.0
Pong	18.3	5.6	18	20.6	20.9
Qbert	10815.6	15148.8	14966.0	16561.7	17249.2
Seaquest	13169.06	2355.4	1706	1754.0	1755.3
Space Invaders	1883.4	15730.5	N/A	1077.3	1427.8
Up n Down	12561.58	74705.7	8623	88105.3	100523.3
Training	4d CPU cluster	4d CPU	1d GPU	12h GPU	15h GPU

Table 1 of "Efficient Parallel Methods for Deep Reinforcement Learning" by Alfredo V. Clemente et al.

The authors use 8 workers, $n_e = 32$ parallel environments, 5-step returns, $\gamma = 0.99$, $\varepsilon = 0.1$, $\beta = 0.01$, and a learning rate of $\alpha = 0.0007 \cdot n_e = 0.0224$.

The $\mathrm{arch}_{\mathrm{nips}}$ is from A3C: 16 filters 8×8 stride 4, 32 filters 4×4 stride 2, a dense layer with 256 units. The $\mathrm{arch}_{\mathrm{nature}}$ is from DQN: 32 filters 8×8 stride 4, 64 filters 4×4 stride 2, 64 filters 3×3 stride 1 and 512-unit fully connected layer. All nonlinearities are ReLU.

Parallel Advantage Actor Critic

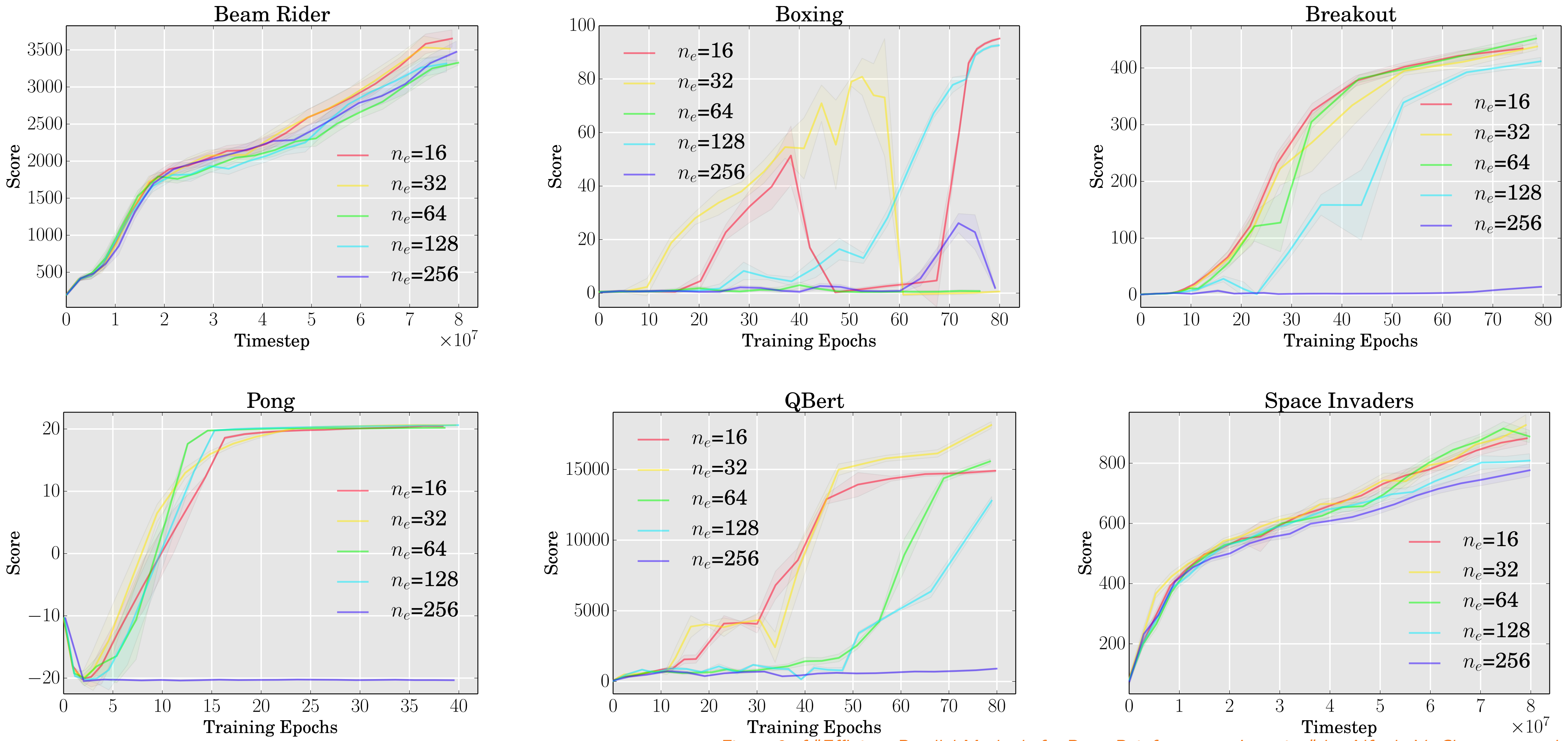


Figure 3 of "Efficient Parallel Methods for Deep Reinforcement Learning" by Alfredo V. Clemente et al.

Parallel Advantage Actor Critic

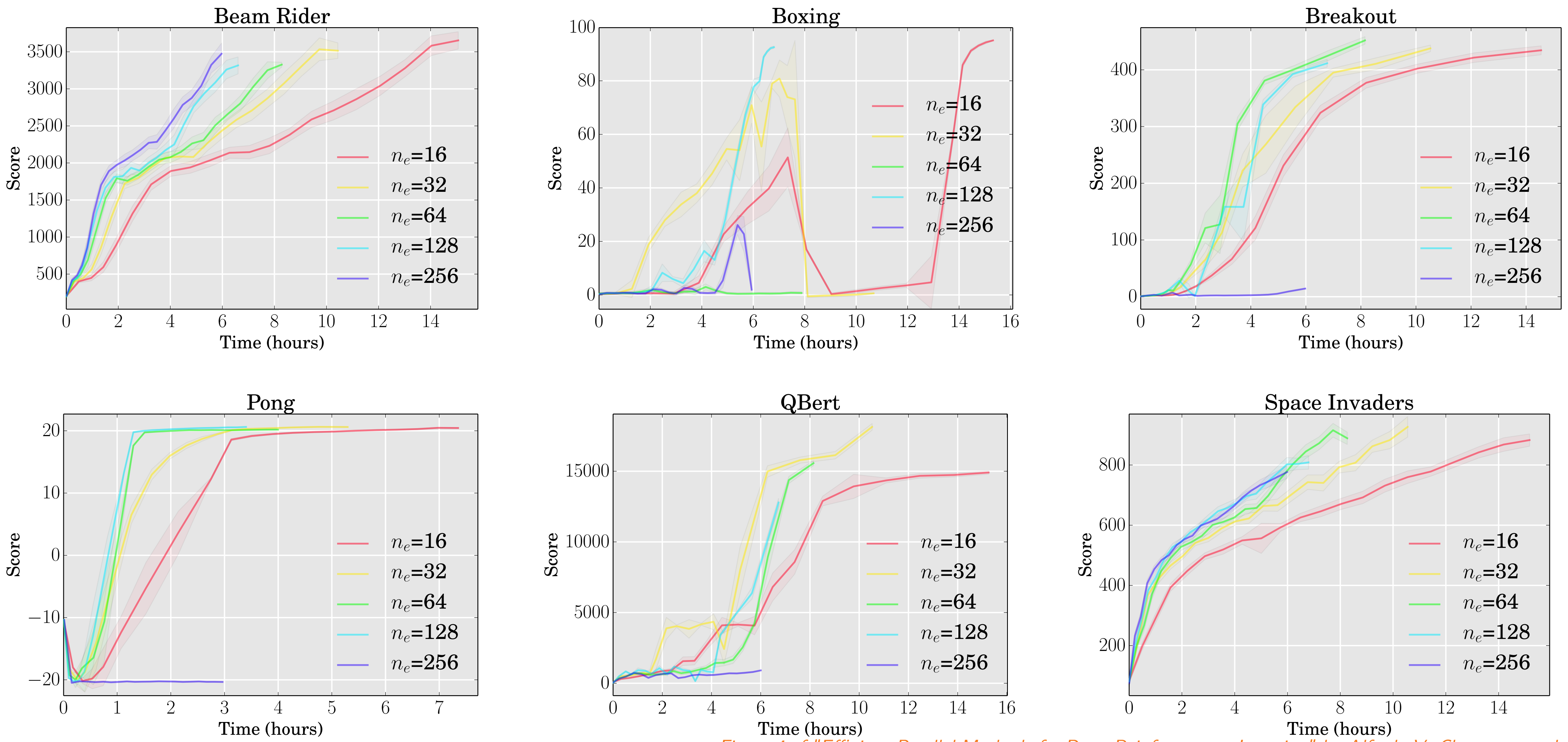


Figure 4 of "Efficient Parallel Methods for Deep Reinforcement Learning" by Alfredo V. Clemente et al.

Parallel Advantage Actor Critic

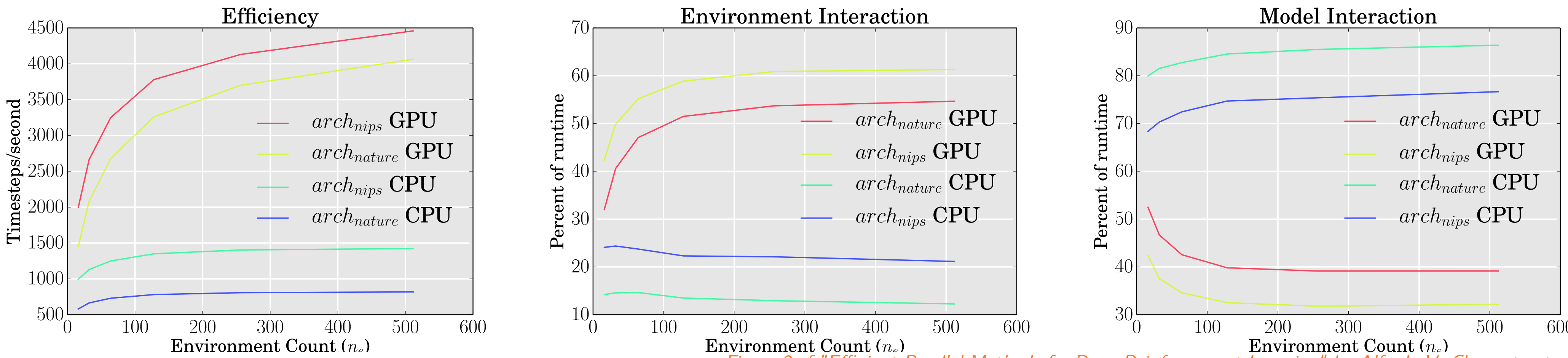


Figure 2 of "Efficient Parallel Methods for Deep Reinforcement Learning" by Alfredo V. Clemente et al.

Continuous Action Space

Until now, the actions were discrete. However, many environments naturally accept actions from continuous space. We now consider actions which come from range $[a, b]$ for $a, b \in \mathbb{R}$, or more generally from a Cartesian product of several such ranges:

$$\prod_i [a_i, b_i].$$

A simple way how to parametrize the action distribution is to choose them from the normal distribution.

Given mean μ and variance σ^2, probability density function of $\mathcal{N}(\mu, \sigma^2)$ is

$$p(x) \stackrel{\text{def}}{=} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

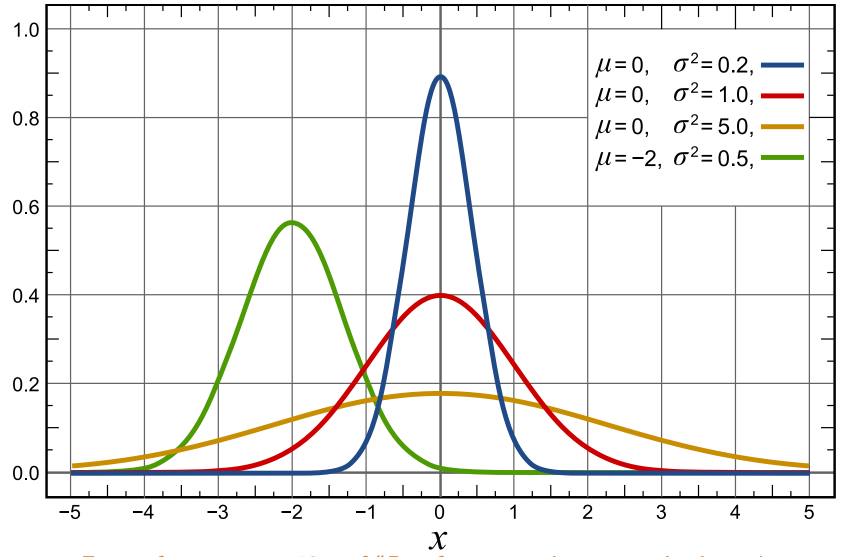


Figure from section 13.7 of "Reinforcement Learning: An Introduction, Second Edition".

Continuous Action Space in Gradient Methods

Utilizing continuous action spaces in gradient-based methods is straightforward. Instead of the $\operatorname{softmax}$ distribution, we suitably parametrize the action value, usually using the normal distribution.

Considering only one real-valued action, we therefore have

$$\pi(a|s;\boldsymbol\theta) \stackrel{\text{def}}{=} P\Big(a \sim \mathcal{N}\big(\mu(s;\boldsymbol\theta), \sigma(s;\boldsymbol\theta)^2\big)\Big),$$

where $\mu(s;\boldsymbol\theta)$ and $\sigma(s;\boldsymbol\theta)$ are function approximation of mean and standard deviation of the action distribution.

The mean and standard deviation are usually computed from the shared representation, with

- the mean being computed as a common regression (i.e., one output neuron without activation);
- the standard deviation (which must be positive) being computed again as a single neuron, but with either $\exp$ or $\operatorname{softplus}$, where $\operatorname{softplus}(x) \stackrel{\text{def}}{=} \log(1+e^x)$.

Continuous Action Space in Gradient Methods

During training, we compute $\mu(s;\boldsymbol\theta)$ and $\sigma(s;\boldsymbol\theta)$ and then sample the action value (clipping it to $[a, b]$ if required). To compute the loss, we utilize the probability density function of the normal distribution (and usually also add the entropy penalty).

```
mus = torch.nn.Linear(..., actions)(hidden_layer)
sds = torch.nn.Linear(..., actions)(hidden_layer)
sds = torch.exp(sds)   # or sds = torch.nn.softplus(sds)

action_dist = torch.distributions.Normal(mus, sds)

# Loss computed as - log π(a|s) * returns - entropy_regularization
loss = - action_dist.log_prob(actions) * returns \
       - args.entropy_regularization * action_dist.entropy()
```

Continuous Action Space

When the action consists of several real values, i.e., action is a suitable subregion of $\mathbb{R}^n$ for $n > 1$, we can:

- either use multivariate Gaussian distribution;
- or factorize the probability into a product of univariate normal distributions.
  - This is the most commonly used approach; we then consider an action to be composed of several independent *action components*.

Modeling the action distribution using a single normal distribution might be insufficient, in which case a mixture of normal distributions is usually used.

Deterministic Policy Gradient Theorem

Combining continuous actions and Deep Q Networks is not straightforward. In order to do so, we need a different variant of the policy gradient theorem.

Recall that in policy gradient theorem,

$$\nabla_{\boldsymbol\theta} J(\boldsymbol\theta) \propto \mathbb{E}_{s\sim\mu}\Big[\sum\nolimits_{a\in\mathcal{A}} q_\pi(s, a)\nabla_{\boldsymbol\theta}\pi(a|s;\boldsymbol\theta)\Big].$$

Deterministic Policy Gradient Theorem

Assume that the policy $\pi(s;\boldsymbol\theta)$ is deterministic and computes an action $a\in\mathbb{R}$. Further, assume the reward $r(s, a)$ is actually a deterministic function of the given state-action pair. Then, under several assumptions about continuousness, the following holds:

$$\nabla_{\boldsymbol\theta} J(\boldsymbol\theta) \propto \mathbb{E}_{s\sim\mu}\Big[\nabla_{\boldsymbol\theta}\pi(s;\boldsymbol\theta)\nabla_a q_\pi(s, a)\big|_{a=\pi(s;\boldsymbol\theta)}\Big].$$

The theorem was first proven in the paper Deterministic Policy Gradient Algorithms by David Silver et al in 2014.

Deterministic Policy Gradient Theorem – Proof

The proof is very similar to the original (stochastic) policy gradient theorem.

However, we will be exchanging derivatives and integrals, for which we need several assumptions:

- we assume that $h(s), p(s'|s,a), \nabla_a p(s'|s,a), r(s,a), \nabla_a r(s,a), \pi(s;\boldsymbol\theta), \nabla_{\boldsymbol\theta}\pi(s;\boldsymbol\theta)$ are continuous in all parameters and variables;
- we further assume that $h(s), p(s'|s,a), \nabla_a p(s'|s,a), r(s,a), \nabla_a r(s,a)$ are bounded.

Details (which assumptions are required and when) can be found in Appendix B of the paper *Deterministic Policy Gradient Algorithms: Supplementary Material* by David Silver et al.

Deterministic Policy Gradient Theorem – Proof

$$\begin{aligned}
\nabla_{\boldsymbol\theta} v_\pi(s) &= \nabla_{\boldsymbol\theta} q_\pi(s, \pi(s;\boldsymbol\theta)) \\
&= \nabla_{\boldsymbol\theta} \Big(r\big(s, \pi(s;\boldsymbol\theta)\big) + \int_{s'} \gamma p\big(s' | s, \pi(s;\boldsymbol\theta)\big) \big(v_\pi(s')\big) \,\mathrm{d}s' \Big) \\
&= \nabla_{\boldsymbol\theta} \pi(s;\boldsymbol\theta) \nabla_a r(s,a) \big|_{a=\pi(s;\boldsymbol\theta)} + \nabla_{\boldsymbol\theta} \int_{s'} \gamma p\big(s' | s, \pi(s;\boldsymbol\theta)\big) v_\pi(s') \,\mathrm{d}s' \\
&= \nabla_{\boldsymbol\theta} \pi(s;\boldsymbol\theta) \nabla_a \Big(r(s,a) + \int_{s'} \gamma p\big(s' | s, a\big) v_\pi(s') \,\mathrm{d}s' \Big) \Big|_{a=\pi(s;\boldsymbol\theta)} \\
&\qquad + \int_{s'} \gamma p\big(s' | s, \pi(s;\boldsymbol\theta)\big) \nabla_{\boldsymbol\theta} v_\pi(s') \,\mathrm{d}s' \\
&= \nabla_{\boldsymbol\theta} \pi(s;\boldsymbol\theta) \nabla_a q_\pi(s,a) \big|_{a=\pi(s;\boldsymbol\theta)} + \int_{s'} \gamma p\big(s' | s, \pi(s;\boldsymbol\theta)\big) \nabla_{\boldsymbol\theta} v_\pi(s') \,\mathrm{d}s'
\end{aligned}$$

We finish the proof as in the gradient theorem by continually expanding $\nabla_{\boldsymbol\theta} v_\pi(s')$, getting $\nabla_{\boldsymbol\theta} v_\pi(s) = \int_{s'} \sum_{k=0}^{\infty} \gamma^k P(s \rightarrow s' \text{ in } k \text{ steps } | \pi) \big[\nabla_{\boldsymbol\theta} \pi(s';\boldsymbol\theta) \nabla_a q_\pi(s',a) \big|_{a=\pi(s';\boldsymbol\theta)}\big] \,\mathrm{d}s'$ and then $\nabla_{\boldsymbol\theta} J(\boldsymbol\theta) = \mathbb{E}_{s \sim h} \nabla_{\boldsymbol\theta} v_\pi(s) \propto \mathbb{E}_{s \sim \mu} \big[\nabla_{\boldsymbol\theta} \pi(s;\boldsymbol\theta) \nabla_a q_\pi(s,a) \big|_{a=\pi(s;\boldsymbol\theta)}\big]$.

Deep Deterministic Policy Gradients

Note that the formulation of deterministic policy gradient theorem allows an off-policy algorithm, because the loss functions no longer depends on actions (similarly to how expected Sarsa is also an off-policy algorithm).

We therefore train function approximation for both $\pi(s;\boldsymbol\theta)$ and $q(s,a;\boldsymbol\theta)$, training $q(s,a;\boldsymbol\theta)$ using a deterministic variant of the Bellman equation:

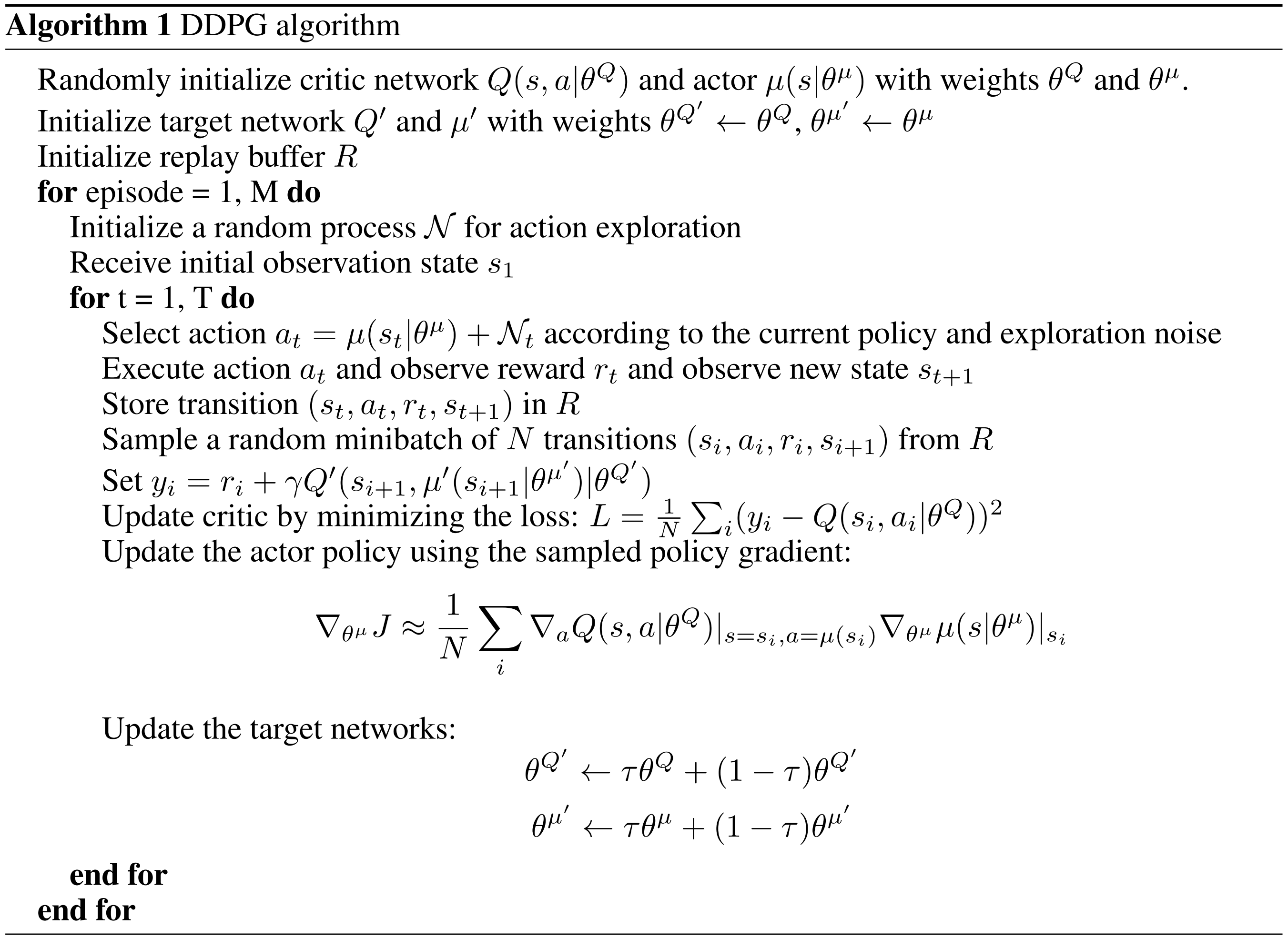
$$q(S_t, A_t; \boldsymbol\theta) = \mathbb{E}_{S_{t+1}}\big[r(S_t, A_t) + \gamma q(S_{t+1}, \pi(S_{t+1}; \boldsymbol\theta))\big]$$

and $\pi(s;\boldsymbol\theta)$ according to the deterministic policy gradient theorem.

The algorithm was first described in the paper Continuous Control with Deep Reinforcement Learning by Timothy P. Lillicrap et al. (2015).

The authors utilize a replay buffer, a target network (updated by exponential moving average with $\tau = 0.001$), batch normalization for CNNs, and perform exploration by adding a Ornstein-Uhlenbeck noise to the predicted actions. Training is performed by Adam with learning rates of 1e-4 and 1e-3 for the policy and critic network, respectively.

Deep Deterministic Policy Gradients



Algorithm 1 DDPG algorithm

```
Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
Initialize target network $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer $R$
for episode = 1, M do
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial observation state $s_1$
    for t = 1, T do
        Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
        Execute action $a_t$ and observe reward $r_t$ and observe new state $s_{t+1}$
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
        Sample a random minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $R$
        Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
        Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
        Update the actor policy using the sampled policy gradient:
            $$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$
        Update the target networks:
            $$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
            $$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$
    end for
end for
```

Algorithm 1 of "Continuous Control with Deep Reinforcement Learning" by Timothy P. Lillicrap et al.

Deep Deterministic Policy Gradients

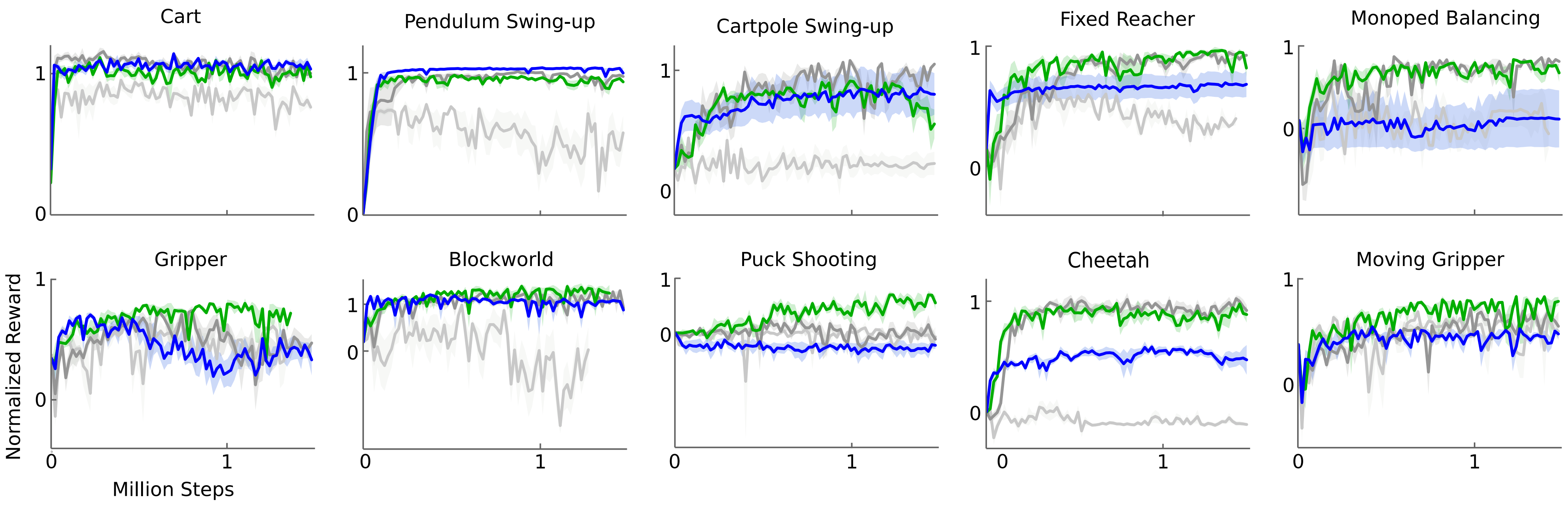
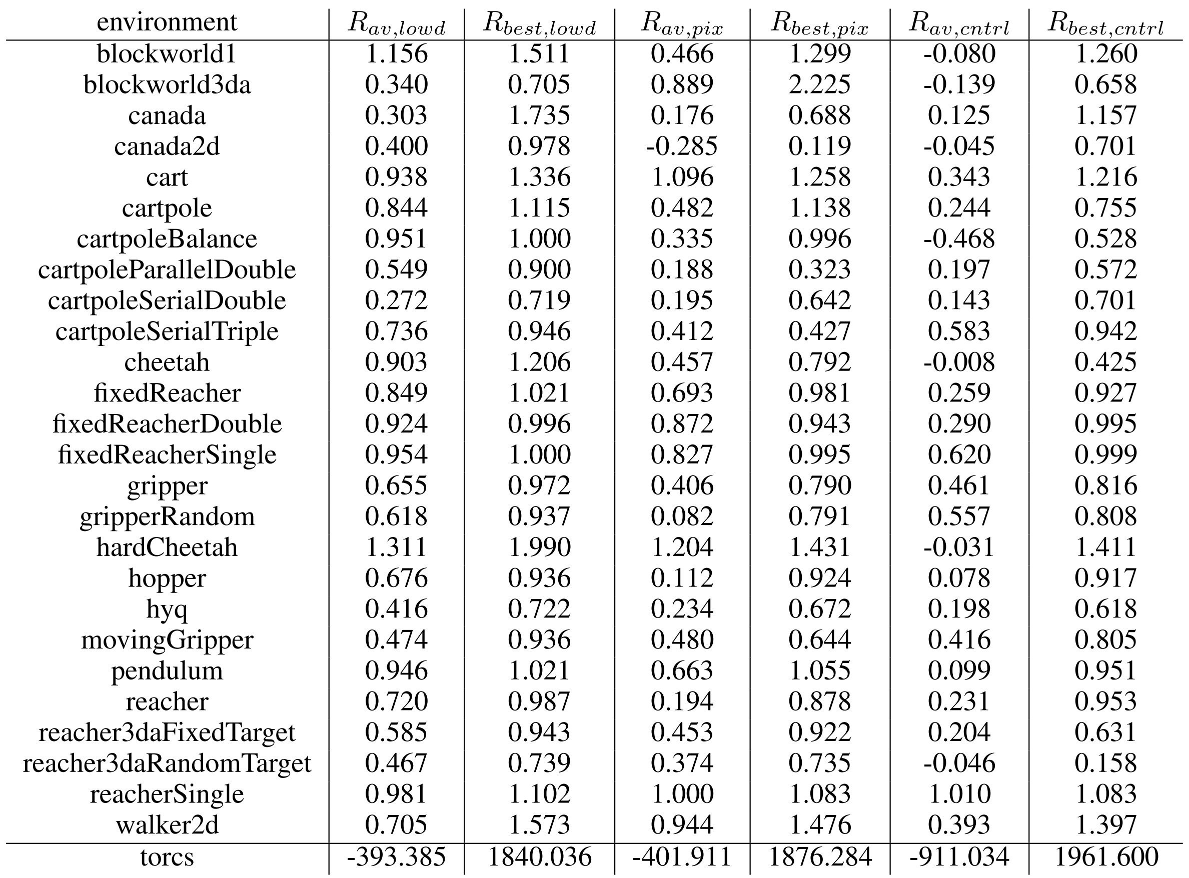


Figure 2: Performance curves for a selection of domains using variants of DPG: original DPG algorithm (minibatch NFQCA) with batch normalization (light grey), with target network (dark grey), with target networks and batch normalization (green), with target networks from pixel-only inputs (blue). Target networks are crucial.

Figure 3 of "Continuous Control with Deep Reinforcement Learning" by Timothy P. Lillicrap et al.

Deep Deterministic Policy Gradients

Results using low-dimensional (*lowd*) version of the environment, pixel representation (*pix*) and DPG reference (*cntrl*).



environment	$R_{av,lowd}$	$R_{best,lowd}$	$R_{av,pix}$	$R_{best,pix}$	$R_{av,cntrl}$	$R_{best,cntrl}$
blockworld1	1.156	1.511	0.466	1.299	-0.080	1.260
blockworld3da	0.340	0.705	0.889	2.225	-0.139	0.658
canada	0.303	1.735	0.176	0.688	0.125	1.157
canada2d	0.400	0.978	-0.285	0.119	-0.045	0.701
cart	0.938	1.336	1.096	1.258	0.343	1.216
cartpole	0.844	1.115	0.482	1.138	0.244	0.755
cartpoleBalance	0.951	1.000	0.335	0.996	-0.468	0.528
cartpoleParallelDouble	0.549	0.900	0.188	0.323	0.197	0.572
cartpoleSerialDouble	0.272	0.719	0.195	0.642	0.143	0.701
cartpoleSerialTriple	0.736	0.946	0.412	0.427	0.583	0.942
cheetah	0.903	1.206	0.457	0.792	-0.008	0.425
fixedReacher	0.849	1.021	0.693	0.981	0.259	0.927
fixedReacherDouble	0.924	0.996	0.872	0.943	0.290	0.995
fixedReacherSingle	0.954	1.000	0.827	0.995	0.620	0.999
gripper	0.655	0.972	0.406	0.790	0.461	0.816
gripperRandom	0.618	0.937	0.082	0.791	0.557	0.808
hardCheetah	1.311	1.990	1.204	1.431	-0.031	1.411
hopper	0.676	0.936	0.112	0.924	0.078	0.917
hyq	0.416	0.722	0.234	0.672	0.198	0.618
movingGripper	0.474	0.936	0.480	0.644	0.416	0.805
pendulum	0.946	1.021	0.663	1.055	0.099	0.951
reacher	0.720	0.987	0.194	0.878	0.231	0.953
reacher3daFixedTarget	0.585	0.943	0.453	0.922	0.204	0.631
reacher3daRandomTarget	0.467	0.739	0.374	0.735	-0.046	0.158
reacherSingle	0.981	1.102	1.000	1.083	1.010	1.083
walker2d	0.705	1.573	0.944	1.476	0.393	1.397
torcs	-393.385	1840.036	-401.911	1876.284	-911.034	1961.600

Table 1 of "Continuous Control with Deep Reinforcement Learning" by Timothy P. Lillicrap et al.

Ornstein-Uhlenbeck Exploration

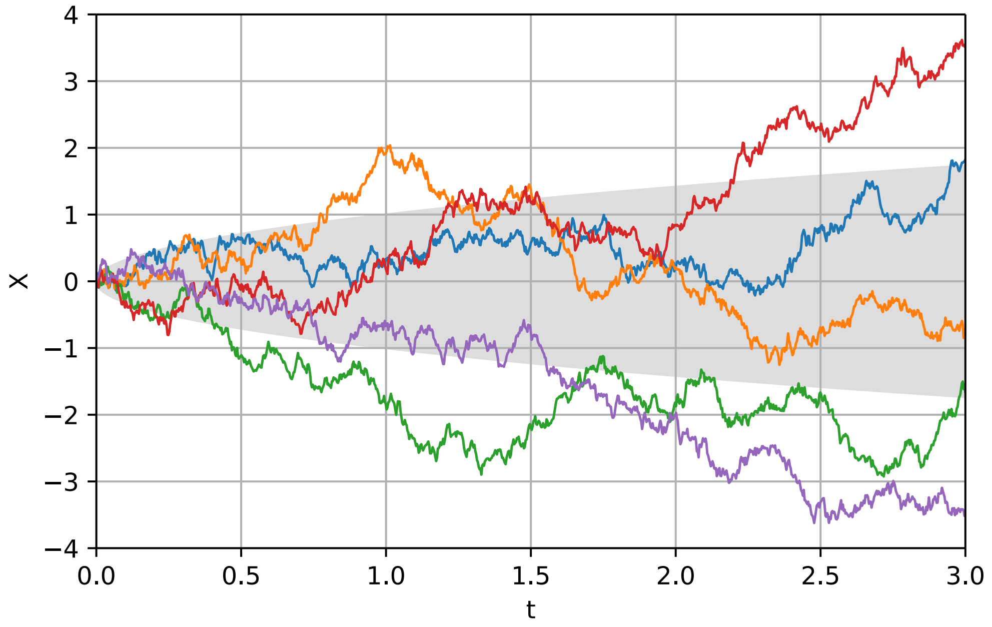
While the exploration policy could just use Gaussian noise, the authors claim that temporarily-correlated noise is more effective for physical control problems with inertia.

They therefore generate noise using Ornstein-Uhlenbeck process, by computing

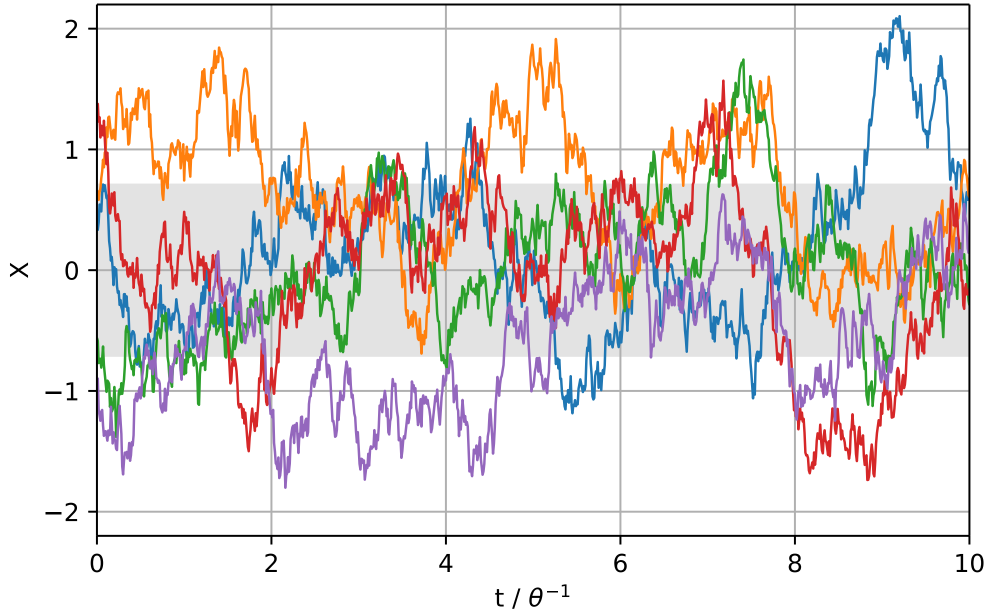
$$n_t \leftarrow n_{t-1} + \theta \cdot (\mu - n_{t-1}) + \varepsilon \sim \mathcal{N}(0, \sigma^2),$$

utilizing hyperparameter values $\theta = 0.15$ and $\sigma = 0.2$.

Ornstein-Uhlenbeck Exploration Visualization



https://upload.wikimedia.org/wikipedia/commons/7/79/Wiener-process-5traces.svg



https://upload.wikimedia.org/wikipedia/commons/6/60/Ornstein-Uhlenbeck-5traces.svg

- On the left, there is a continuous *Wiener process* (a "brownian path"), corresponding to $\theta = 0$ and $\sigma = 1$.
- On the right, there is Ornstein-Uhlenbeck process example with $\theta = \sigma = 1$ and $\mu = 0$.

The gray are corresponds to the standard deviation of x (n in our notation).

MuJoCo

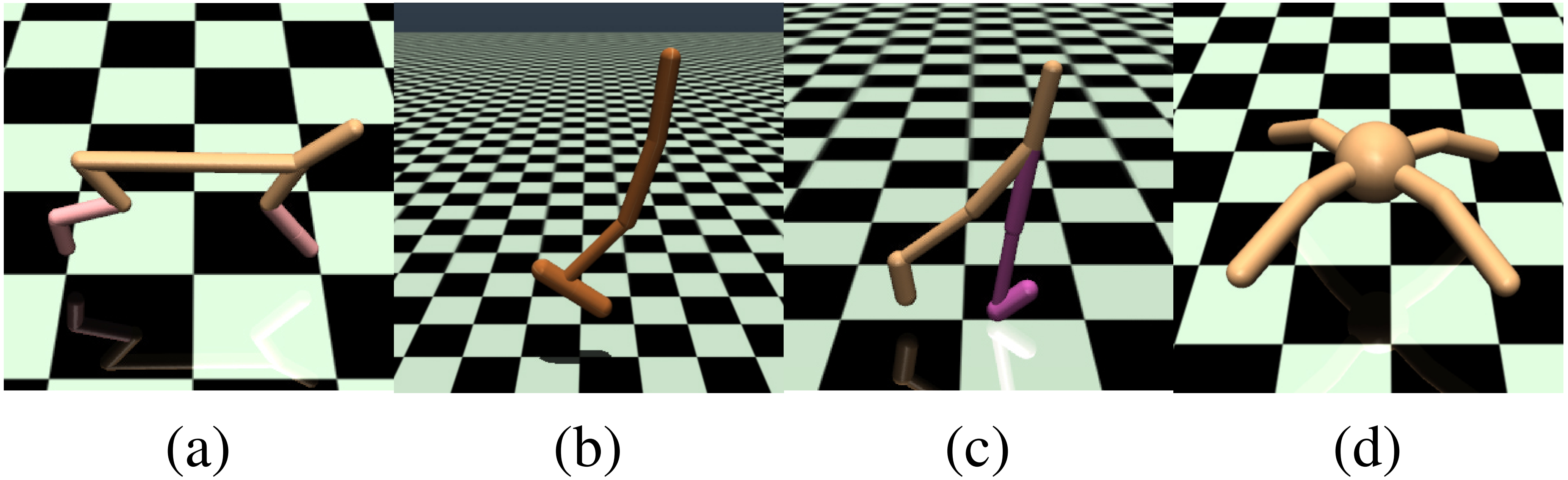


Figure 4. Example MuJoCo environments (a) HalfCheetah-v1, (b) Hopper-v1, (c) Walker2d-v1, (d) Ant-v1.

Figure 4 of "Addressing Function Approximation Error in Actor-Critic Methods" by Scott Fujimoto et al.

Twin Delayed Deep Deterministic Policy Gradient

The paper Addressing Function Approximation Error in Actor-Critic Methods by Scott Fujimoto et al. from February 2018 proposes improvements to DDPG which

- decrease maximization bias by training two critics and choosing the minimum of their predictions;
- introduce several variance-lowering optimizations:
  - delayed policy updates;
  - target policy smoothing.

The TD3 algorithm has been together with SAC one of the best algorithms for off-policy continuous-actions RL training (as of 2022).

TD3 – Maximization Bias

Similarly to Q-learning, the DDPG algorithm suffers from maximization bias. In Q-learning, the maximization bias was caused by the explicit $\max$ operator. For DDPG methods, it can be caused by the gradient descent itself. Let $\boldsymbol\theta_{approx}$ be the parameters maximizing the $q_{\boldsymbol\theta}$ and let $\boldsymbol\theta_{true}$ be the hypothetical parameters which maximise true q_π, and let π_{approx} and π_{true} denote the corresponding policies.

Because the gradient direction is a local maximizer, for sufficiently small $\alpha < \varepsilon_1$ we have

$$\mathbb{E}\big[q_{\boldsymbol\theta}(s, \pi_{approx})\big] \geq \mathbb{E}\big[q_{\boldsymbol\theta}(s, \pi_{true})\big].$$

However, for real q_π and for sufficiently small $\alpha < \varepsilon_2$, it holds that

$$\mathbb{E}\big[q_\pi(s, \pi_{true})\big] \geq \mathbb{E}\big[q_\pi(s, \pi_{approx})\big].$$

Therefore, if $\mathbb{E}\big[q_{\boldsymbol\theta}(s, \pi_{true})\big] \geq \mathbb{E}\big[q_\pi(s, \pi_{true})\big]$, for $\alpha < \min(\varepsilon_1, \varepsilon_2)$

$$\mathbb{E}\big[q_{\boldsymbol\theta}(s, \pi_{approx})\big] \geq \mathbb{E}\big[q_\pi(s, \pi_{approx})\big].$$

TD3 – Maximization Bias

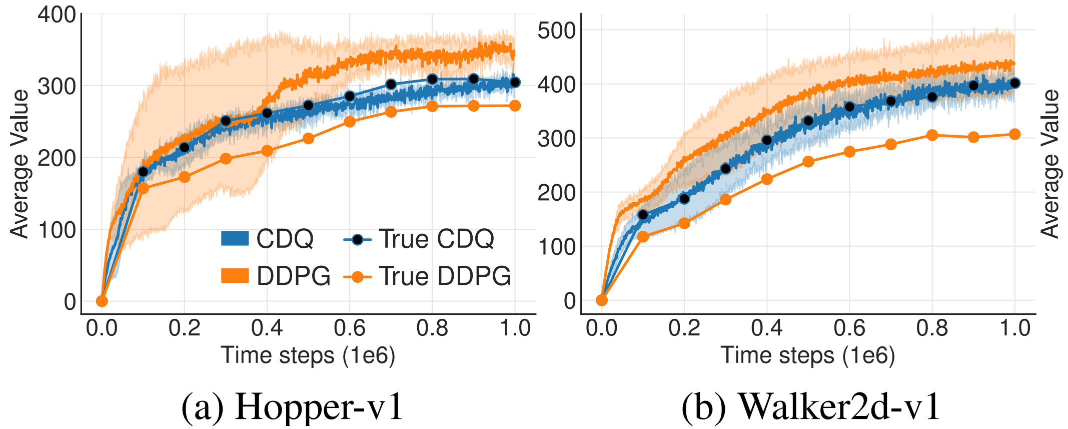


Figure 1 of "Addressing Function Approximation Error in Actor-Critic Methods" by Scott Fujimoto et al.

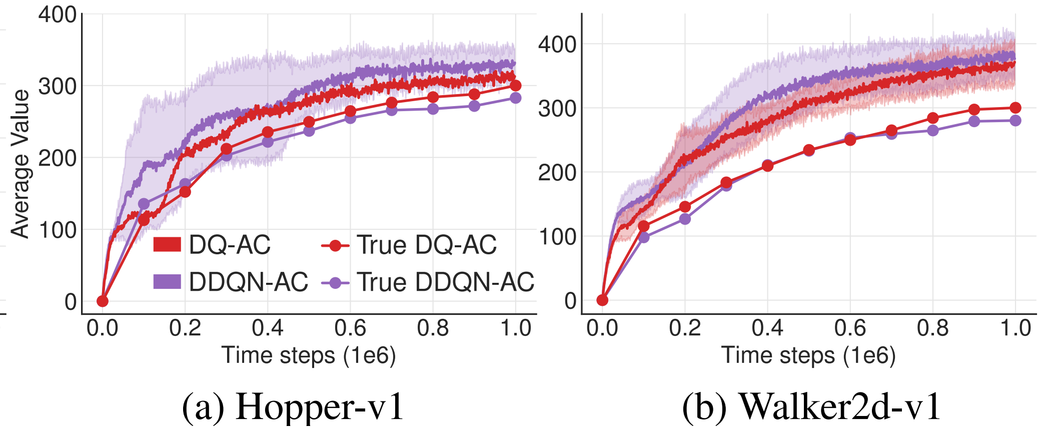


Figure 2 of "Addressing Function Approximation Error in Actor-Critic Methods" by Scott Fujimoto et al.

Analogously to Double DQN we could compute the learning targets using the current policy and the target critic, i.e., $r + \gamma q_{\boldsymbol\theta'}(s', \pi_{\boldsymbol\varphi(s')})$ (instead of using the target policy and the target critic as in DDPG), obtaining DDQN-AC algorithm. However, the authors found out that the policy changes too slowly and the target and current networks are too similar.

Using the original Double Q-learning, two pairs of actors and critics could be used, with the learning targets computed by the opposite critic, i.e., $r + \gamma q_{\boldsymbol\theta_2}(s', \pi_{\boldsymbol\varphi_1}(s'))$ for updating $q_{\boldsymbol\theta_1}$. The resulting DQ-AC algorithm is slightly better, but still suffering from overestimation.

TD3 – Algorithm

The authors instead suggest to employ two critics and one actor. The actor is trained using one of the critics, and both critics are trained using the same target computed using the *minimum* value of both critics as

$$r + \gamma \min_{i=1,2} q_{\boldsymbol\theta'_i}(s', \pi_{\boldsymbol\varphi'}(s')).$$

The resulting algorithm is called CDQ – Clipped Double Q-learning.

Furthermore, the authors suggest two additional improvements for variance reduction.

- For obtaining higher quality target values, the authors propose to train the critics more often. Therefore, critics are updated each step, but the actor and the target networks are updated only every d-th step ($d = 2$ is used in the paper).
- To explicitly model that similar actions should lead to similar results, a small random noise is added to the performed actions when computing the target value:

$$r + \gamma \min_{i=1,2} q_{\boldsymbol\theta'_i}(s', \pi_{\boldsymbol\varphi'}(s') + \varepsilon) \quad \text{for} \quad \varepsilon \sim \operatorname{clip}(\mathcal{N}(0, \sigma), -c, c).$$

TD3 – Algorithm

Algorithm 1 TD3

```
Initialize critic networks $Q_{\theta_1}, Q_{\theta_2}$, and actor network $\pi_\phi$
with random parameters $\theta_1, \theta_2, \phi$
Initialize target networks $\theta'_1 \leftarrow \theta_1, \theta'_2 \leftarrow \theta_2, \phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for $t = 1$ to $T$ do
    Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$,
    $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward $r$ and new state $s'$
    Store transition tuple $(s, a, r, s')$ in $\mathcal{B}$

    Sample mini-batch of $N$ transitions $(s, a, r, s')$ from $\mathcal{B}$
    $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
    $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
    Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
    if $t$ mod $d$ then
        Update $\phi$ by the deterministic policy gradient:
        $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
        Update target networks:
        $\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$
        $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
    end if
end for
```

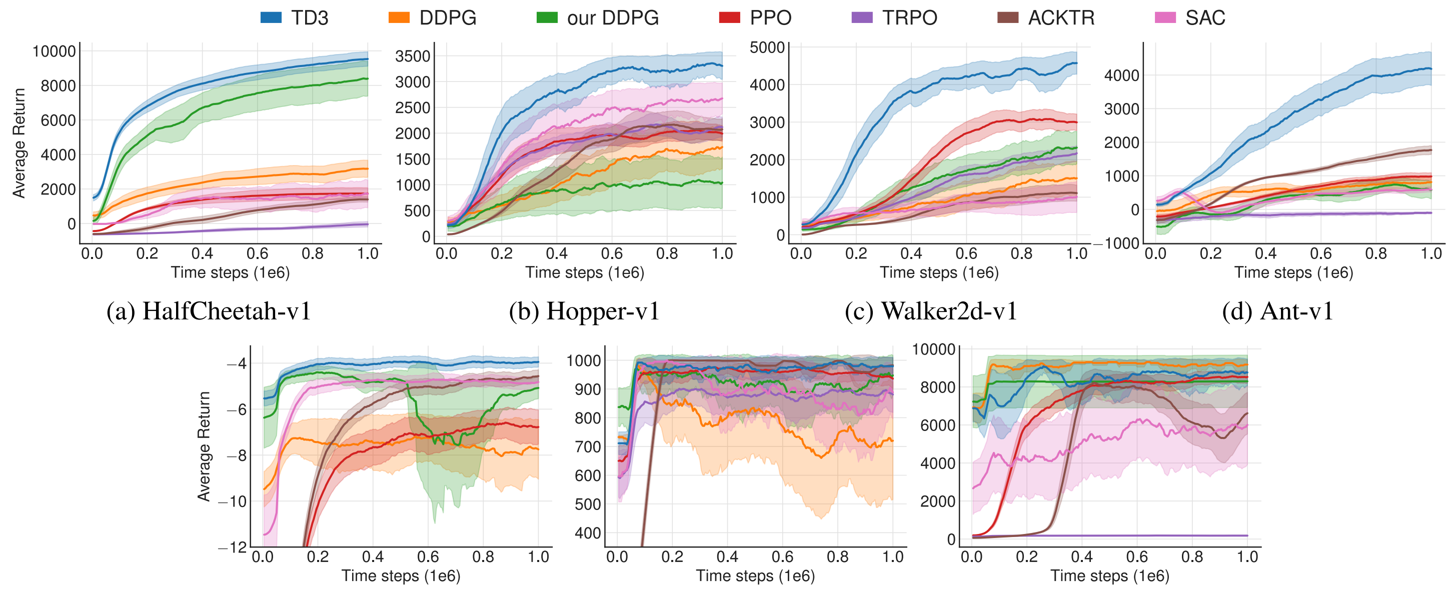
Algorithm 1 of "Addressing Function Approximation Error in Actor-Critic Methods" by Scott Fujimoto et al.

TD3 – Algorithm

Hyper-parameter	**Ours**	**DDPG**
Critic Learning Rate	10^{-3}	10^{-3}
Critic Regularization	None	$10^{-2} \cdot \|\|\theta\|\|^2$
Actor Learning Rate	10^{-3}	10^{-4}
Actor Regularization	None	None
Optimizer	Adam	Adam
Target Update Rate (τ)	$5 \cdot 10^{-3}$	10^{-3}
Batch Size	100	64
Iterations per time step	1	1
Discount Factor	0.99	0.99
Reward Scaling	1.0	1.0
Normalized Observations	False	True
Gradient Clipping	False	False
Exploration Policy	$\mathcal{N}(0, 0.1)$	OU, $\theta = 0.15, \mu = 0, \sigma = 0.2$

Table 3 of "Addressing Function Approximation Error in Actor-Critic Methods" by Scott Fujimoto et al.

TD3 – Results



(a) HalfCheetah-v1 (b) Hopper-v1 (c) Walker2d-v1 (d) Ant-v1

(e) Reacher-v1 (f) InvertedPendulum-v1 (g) InvertedDoublePendulum-v1

Figure 5 of "Addressing Function Approximation Error in Actor-Critic Methods" by Scott Fujimoto et al.

Environment	**TD3**	**DDPG**	**Our DDPG**	**PPO**	**TRPO**	**ACKTR**	**SAC**
HalfCheetah	**9636.95 ± 859.065**	3305.60	8577.29	1795.43	-15.57	1450.46	2347.19
Hopper	**3564.07 ± 114.74**	2020.46	1860.02	2164.70	2471.30	2428.39	2996.66
Walker2d	**4682.82 ± 539.64**	1843.85	3098.11	3317.69	2321.47	1216.70	1283.67
Ant	**4372.44 ± 1000.33**	1005.30	888.77	1083.20	-75.85	1821.94	655.35
Reacher	**-3.60 ± 0.56**	-6.51	**-4.01**	-6.18	-111.43	-4.26	-4.44
InvPendulum	**1000.00 ± 0.00**	**1000.00**	**1000.00**	**1000.00**	985.40	**1000.00**	**1000.00**
InvDoublePendulum	**9337.47 ± 14.96**	**9355.52**	8369.95	8977.94	205.85	9081.92	8487.15

Table 1 of "Addressing Function Approximation Error in Actor-Critic Methods" by Scott Fujimoto et al.

TD3 – Ablations

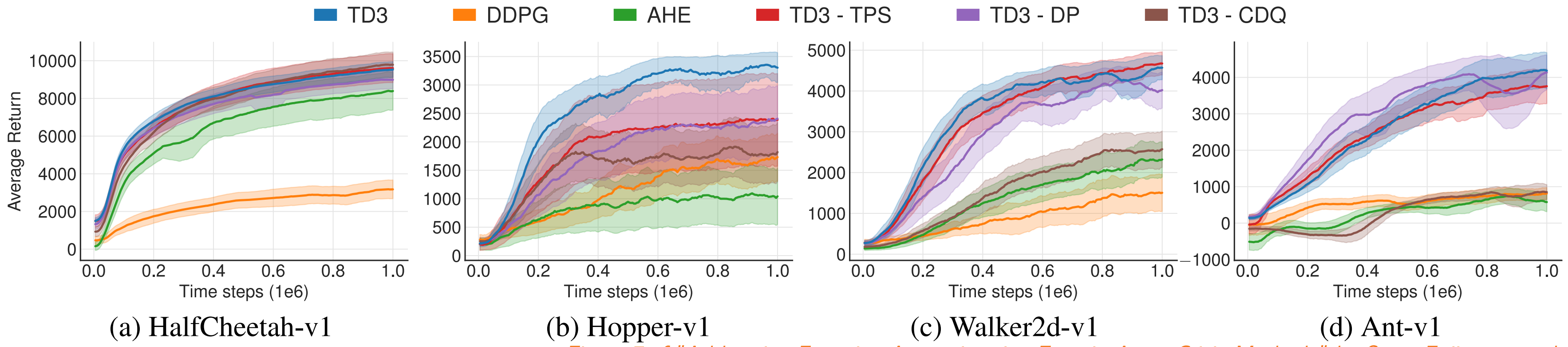


Figure 7 of "Addressing Function Approximation Error in Actor-Critic Methods" by Scott Fujimoto et al.

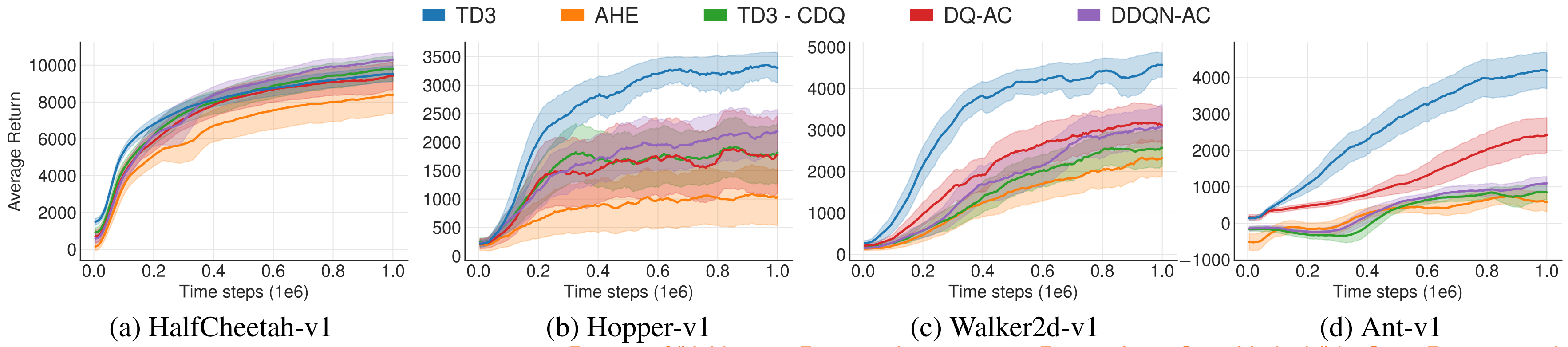


Figure 8 of "Addressing Function Approximation Error in Actor-Critic Methods" by Scott Fujimoto et al.

The AHE is the authors' reimplementation of DDPG using updated architecture, hyperparameters, and exploration. TPS is Target Policy Smoothing, DP is Delayed Policy update, and CDQ is Clipped Double Q-learning.

TD3 – Ablations

Method	**HCheetah**	**Hopper**	**Walker2d**	**Ant**
TD3	9532.99	**3304.75**	**4565.24**	**4185.06**
DDPG	3162.50	1731.94	1520.90	816.35
AHE	8401.02	1061.77	2362.13	564.07
AHE + DP	7588.64	1465.11	2459.53	896.13
AHE + TPS	9023.40	907.56	2961.36	872.17
AHE + CDQ	6470.20	1134.14	3979.21	3818.71
TD3 - DP	9590.65	2407.42	**4695.50**	3754.26
TD3 - TPS	8987.69	2392.59	4033.67	**4155.24**
TD3 - CDQ	9792.80	1837.32	2579.39	849.75
DQ-AC	9433.87	1773.71	3100.45	2445.97
DDQN-AC	**10306.90**	2155.75	3116.81	1092.18

Table 2 of "Addressing Function Approximation Error in Actor-Critic Methods" by Scott Fujimoto et al.

Soft Actor Critic

The paper *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor* by Tuomas Haarnoja et al. from Jan 2018 introduces a different off-policy algorithm for continuous action space.

It was followed by a continuation paper *Soft Actor-Critic Algorithms and Applications* in Dec 2018.

The general idea is to introduce entropy directly in the value function we want to maximize, instead of just ad-hoc adding the entropy penalty. Such an approach is an instance of *regularized policy optimization*.

Soft Actor Critic Objective

Until now, our goal was to optimize

$$\mathbb{E}_\pi \big[G_0\big].$$

Assume the rewards are deterministic and that μ_π is on-policy distribution of a policy π.

In the soft actor-critic, the authors instead propose to optimize the maximum entropy objective

$$\begin{aligned}\pi_* &= \arg\max_\pi \mathbb{E}_{s\sim\mu_\pi} \Big[\mathbb{E}_{a\sim\pi(s)} \big[r(s,a)\big] + \alpha H(\pi(\cdot|s))\Big] \\ &= \arg\max_\pi \mathbb{E}_{s\sim\mu_\pi, a\sim\pi(s)} \big[r(s,a) - \alpha \log \pi(a|s)\big].\end{aligned}$$

Note that the value of α is dependent on the magnitude of returns and that for a fixed policy, the entropy penalty can be "hidden" in the reward.

Soft Actor Critic Objective

To maximize the regularized objective, we define the following augmented reward:

$$r_\pi(s, a) \stackrel{\text{def}}{=} r(s, a) + \mathbb{E}_{s' \sim p(s,a)} \big[\alpha H(\pi(\cdot|s'))\big].$$

From now on, we consider **soft action-value** function corresponding to this augmented reward.

Soft Policy Evaluation

Our goal is now to derive **soft policy iteration**, an analogue of policy iteration algorithm.

We start by considering soft policy evaluation. Let a modified Bellman backup operator $\mathcal{T}_\pi$ be defined as

$$\mathcal{T}_\pi q(s, a) \stackrel{\text{def}}{=} r(s, a) + \gamma \mathbb{E}_{s' \sim p(s,a)} \big[v(s')\big],$$

where the **soft (state-)value** function $v(s)$ is defined as

$$v(s) = \mathbb{E}_{a \sim \pi} \big[q(s, a)\big] + \alpha H(\pi(\cdot|s)) = \mathbb{E}_{a \sim \pi} \big[q(s, a) - \alpha \log \pi(a|s)\big].$$

This modified Bellman backup operator corresponds to the usual one for the augmented rewards $r_\pi(s, a)$, and therefore the repeated application $\mathcal{T}_\pi^k q$ converges to q_π according to the original proof.

Soft Policy Improvement

While the soft policy evaluation was a straightforward modification of the original policy evaluation, the soft policy improvement is quite different.

Assume we have a policy π, its action-value function q_π from the soft policy evaluation, and we want to improve the policy. Furthermore, we should select the improved policy from a family of parametrized distributions Π.

We define the improved policy π' as

$$\pi'(\cdot|s) \stackrel{\text{def}}{=} \operatorname*{arg\,min}_{\bar\pi\in\Pi} J_\pi(\bar\pi) \stackrel{\text{def}}{=} \operatorname*{arg\,min}_{\bar\pi\in\Pi} D_{\text{KL}}\left(\bar\pi(\cdot|s) \middle\| \frac{\exp\left(\frac{1}{\alpha}q_\pi(s,\cdot)\right)}{z_\pi(s)}\right),$$

where $z_\pi(s)$ is the partition function (i.e., normalization factor such that the right-hand side is a distribution), which does not depend on the new policy and thus can be ignored.

Soft Policy Improvement

We now prove that $q_{\pi'}(s, a) \geq q_\pi(s, a)$ for any state s and action a.

We start by noting that $J_\pi(\pi') \leq J_\pi(\pi)$, because we can always choose π as the improved policy. Therefore,

$$\mathbb{E}_{a\sim\pi'}\big[\alpha \log \pi'(a|s) - q_\pi(s, a) + \alpha \log z_\pi(s)\big] \leq \mathbb{E}_{a\sim\pi}\big[\alpha \log \pi(a|s) - q_\pi(s, a) + \alpha \log z_\pi(s)\big],$$

which results in

$$\mathbb{E}_{a\sim\pi'}\big[q_\pi(s, a) - \alpha \log \pi'(a|s)\big] \geq v_\pi(s).$$

We now finish the proof analogously to the original one:

$$\begin{aligned} q_\pi(s, a) &= r(s, a) + \gamma \mathbb{E}_{s'}[v_\pi(s')] \\ &\leq r(s, a) + \gamma \mathbb{E}_{s'}[\mathbb{E}_{a'\sim\pi'}[q_\pi(s', a') - \alpha \log \pi'(a'|s')] \\ &\dots \\ &\leq q_{\pi'}(s, a). \end{aligned}$$

Soft Policy Iteration

The soft policy iteration algorithm alternates between the soft policy evaluation and soft policy improvement steps.

The repeated application of these two steps produce better and better policies. In other words, we get a monotonically increasing sequence of soft action-value functions.

If the soft action-value function is bounded (the paper assumes a bounded reward and a finite number of actions to bound the entropy), the repeated application converges to some q_*, from which we get a π_* using the soft policy improvement step. (It is not clear to me why the algorithm should converge in finite time, but we can make the rest of the slide conditional on “if the algorithm converges”).

It remains to show that the π_* is indeed the optimal policy fulfilling $q_{\pi_*}(s, a) \geq q_\pi(s, a)$.

However, this follows from the fact that at convergence, $J_{\pi_*}(\pi_*) \leq J_{\pi_*}(\pi)$, and following the same reasoning as in the proof of the soft policy improvement, we obtain the required $q_{\pi_*}(s, a) \geq q_\pi(s, a)$.

Soft Policy Improvement Derivation

The following derivation is not in the original paper, but it is my understanding of how the softmax of the action-value function arises. For simplicity, we assume finite number of actions.

Assuming we have a policy π and its action-value function q_π, we usually improve the policy using

$$\begin{aligned}
\nu(\cdot|s) &= \arg\max_\nu \mathbb{E}_{a\sim\nu(\cdot|s)}\big[q_\pi(s,a)\big] \\
&= \arg\max_\nu \sum\nolimits_a q_\pi(s,a)\nu(a|s) \\
&= \arg\max_\nu \boldsymbol{q}_\pi(s,\cdot)^T\boldsymbol{\nu}(\cdot|s),
\end{aligned}$$

which results in a greedy improvement with the form of

$$\nu(s) = \arg\max\nolimits_a q_\pi(s,a).$$

Soft Policy Improvement Derivation

Now consider instead the regularized objective

$$\begin{aligned}\nu(\cdot|s) &= \arg\max_\nu \Big(\mathbb{E}_{a\sim\nu(\cdot|s)}\big[q_\pi(s,a)\big] + \alpha H(\nu(\cdot|s))\Big) \\ &= \arg\max_\nu \Big(\mathbb{E}_{a\sim\nu}\big[q_\pi(s,a) - \alpha\log\nu(a|s)\big]\Big)\end{aligned}$$

To maximize it for a given s, we form a Lagrangian

$$\mathcal{L} = \Big(\sum\nolimits_a \nu(a|s)\big(q_\pi(s,a) - \alpha\log\nu(a|s)\big)\Big) - \lambda\Big(1 - \sum\nolimits_a \nu(a|s)\Big).$$

The derivative with respect to $\nu(a|s)$ is

$$\frac{\partial\mathcal{L}}{\partial\nu(a|s)} = q_\pi(s,a) - \alpha\log\nu(a|s) - \alpha + \lambda.$$

Setting it to zero, we get $\alpha\log\nu(a|s) = q_\pi(s,a) + \lambda - \alpha$, resulting in $\nu(a|s) \propto e^{\frac{1}{\alpha}q_\pi(s,a)}$.

Soft Actor Critic

Our soft actor critic will be an off-policy algorithm with continuous action space. The model consist of two critics $q_{\boldsymbol\theta_1}$ and $q_{\boldsymbol\theta_2}$, two target critics $q_{\bar{\boldsymbol\theta}_1}$ and $q_{\bar{\boldsymbol\theta}_2}$ and finally a single actor $\pi_{\boldsymbol\varphi}$.

The authors state that

- with a single critic, all the described experiments still converge;
- they adopted the two critics from the TD3 paper;
- using two critics "significantly speed up training".

Soft Actor Critic – Critic Training

To train the critic, we use the modified Bellman backup operator, resulting in the loss

$$J_q(\boldsymbol\theta_i) = \mathbb{E}_{s\sim\mu_\pi, a\sim\pi_\varphi(s)}\Big[\Big(q_{\boldsymbol\theta_i}(s,a) - \big(r(s,a) + \gamma\mathbb{E}_{s'\sim p(s,a)}[v_{\min}(s')]\big)\Big)^2\Big],$$

where

$$v_{\min}(s) = \mathbb{E}_{a\sim\pi_\varphi(s)}\Big[\min_i\big(q_{\bar{\boldsymbol\theta}_i}(s,a)\big) - \alpha\log\pi_{\boldsymbol\varphi}(a|s)\Big].$$

The target critics are updated using exponential moving averages with momentum τ.

Soft Actor Critic – Actor Training

The actor is updated by directly minimizing the KL divergence, resulting in the loss

$$J_\pi(\boldsymbol\varphi) = \mathbb{E}_{s\sim\mu_\pi, a\sim\pi_{\boldsymbol\varphi}(s)}\Big[\alpha \log\big(\pi_{\boldsymbol\varphi}(a, s)\big) - \min_i\big(q_{\boldsymbol\theta_i}(s, a)\big)\Big].$$

Given that our critics are differentiable, we now reparametrize the policy as

$$a = f_{\boldsymbol\varphi}(s, \varepsilon).$$

Specifically, we sample $\varepsilon \sim \mathcal{N}(0, 1)$ and let $f_{\boldsymbol\varphi}$ produce an unbounded Gaussian distribution (a diagonal one if the actions are vectors).

Together, we obtain

$$J_\pi(\boldsymbol\varphi) = \mathbb{E}_{s\sim\mu_\pi, \varepsilon\sim\mathcal{N}(0,1)}\Big[\alpha \log\big(\pi_{\boldsymbol\varphi}(f_{\boldsymbol\varphi}(s, \varepsilon), s)\big) - \min_i\big(q_{\boldsymbol\theta_i}(s, f_{\boldsymbol\varphi}(s, \varepsilon))\big)\Big].$$

Soft Actor Critic – Bounding Actions

In practice, the actions need to be bounded.

The authors propose to apply an invertible squashing function $\tanh$ on the unbounded Gaussian distribution.

Consider that our policy produces an unbounded action $\pi(u|s)$. To define a distribution $\bar\pi(a|s)$ with $a = \tanh(u)$, we need to employ the change of variables, resulting in

$$\bar\pi(a|s) = \pi(u|s)\left(\frac{\partial a}{\partial u}\right)^{-1} = \pi(u|s)\left(\frac{\partial \tanh(u)}{\partial u}\right)^{-1}.$$

Therefore, the log-likelihood has quite a simple form

$$\log \bar\pi(a|s) = \log \pi(u|s) - \log\big(1 - \tanh^2(u)\big).$$

Soft Actor Critic – Automatic Entropy Adjustment

One of the most important hyperparameters is the entropy penalty α.

In the second paper, the authors presented an algorithm for automatic adjustment of its value.

Instead of setting the entropy penalty α, they propose to specify target entropy value $\mathcal{H}$ and then solve a constrained optimization problem

$$\pi_* = \arg\max_\pi \mathbb{E}_{s\sim\mu_\pi, a\sim\pi(s)}\big[r(s,a)\big] \text{ such that } \mathbb{E}_{s\sim\mu_\pi, a\sim\pi(s)}\big[-\log\pi(a|s)\big] \geq \mathcal{H}.$$

We can then form a Lagrangian with a multiplier α

$$\mathbb{E}_{s\sim\mu_\pi, a\sim\pi(s)}\Big[r(s,a) + \alpha\big(-\log\pi(a|s) - \mathcal{H}\big)\Big],$$

which should be maximized with respect to π and minimized with respect to $\alpha \geq 0$.

Soft Actor Critic – Automatic Entropy Adjustment

To optimize the Lagrangian, we perform *dual gradient descent*, where we alternate between maximization with respect to π and minimization with respect to α.

While such a procedure is guaranteed to converge only under the convexity assumptions, the authors report that the dual gradient descent works in practice also with nonlinear function approximation.

To conclude, the automatic entropy adjustment is performed by introducing a final loss

$$J(\alpha) = \mathbb{E}_{s\sim\mu_\pi, a\sim\pi(s)}\big[-\alpha\log\pi(a|s) - \alpha\mathcal{H}\big].$$

Soft Actor Critic

Algorithm 1 Soft Actor-Critic

```
Input: $\theta_1, \theta_2, \phi$                                         ▷ Initial parameters
  $\bar\theta_1 \leftarrow \theta_1, \bar\theta_2 \leftarrow \theta_2$                ▷ Initialize target network weights
  $\mathcal{D} \leftarrow \emptyset$                                    ▷ Initialize an empty replay pool
  for each iteration do
    for each environment step do
      $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$                         ▷ Sample action from the policy
      $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$        ▷ Sample transition from the environment
      $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$   ▷ Store the transition in the replay pool
    end for
    for each gradient step do
      $\theta_i \leftarrow \theta_i - \lambda_Q \hat\nabla_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$   ▷ Update the Q-function parameters
      $\phi \leftarrow \phi - \lambda_\pi \hat\nabla_\phi J_\pi(\phi)$                ▷ Update policy weights
      $\alpha \leftarrow \alpha - \lambda \hat\nabla_\alpha J(\alpha)$                 ▷ Adjust temperature
      $\bar\theta_i \leftarrow \tau\theta_i + (1-\tau)\bar\theta_i$ for $i \in \{1, 2\}$   ▷ Update target network weights
    end for
  end for
Output: $\theta_1, \theta_2, \phi$                                        ▷ Optimized parameters
```

Algorithm 1 of "Soft Actor-Critic Algorithms and Applications" by Tuomas Haarnoja et al.

Soft Actor Critic

Table 1: SAC Hyperparameters

Parameter	Value
optimizer	Adam (Kingma & Ba, 2015)
learning rate	$3 \cdot 10^{-4}$
discount (γ)	0.99
replay buffer size	10^6
number of hidden layers (all networks)	2
number of hidden units per layer	256
number of samples per minibatch	256
entropy target	$-\dim(\mathcal{A})$ (e.g. , -6 for HalfCheetah-v1)
nonlinearity	ReLU
target smoothing coefficient (τ)	0.005
target update interval	1
gradient steps	1

Table 1 of "Soft Actor-Critic Algorithms and Applications" by Tuomas Haarnoja et al.

Soft Actor Critic

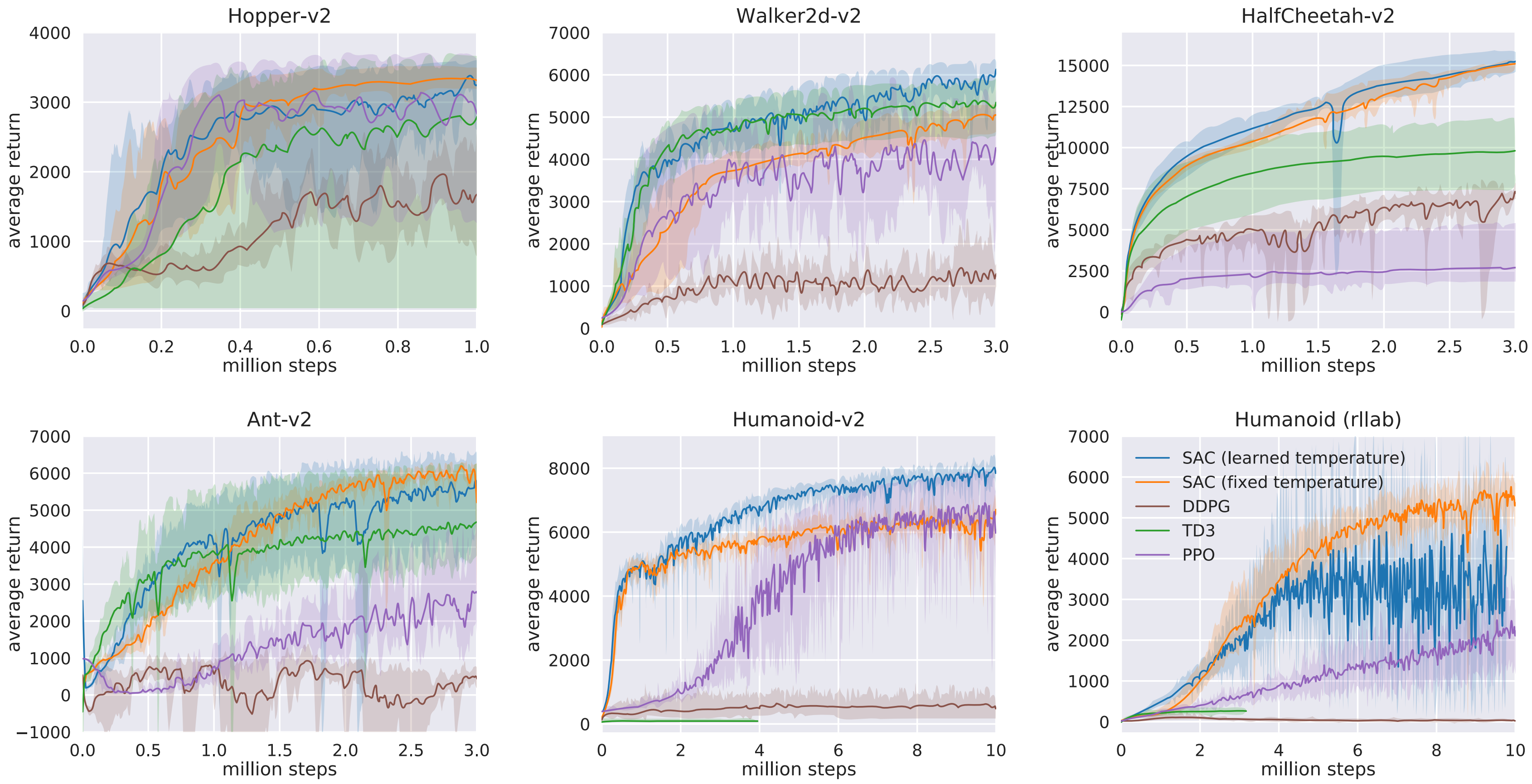
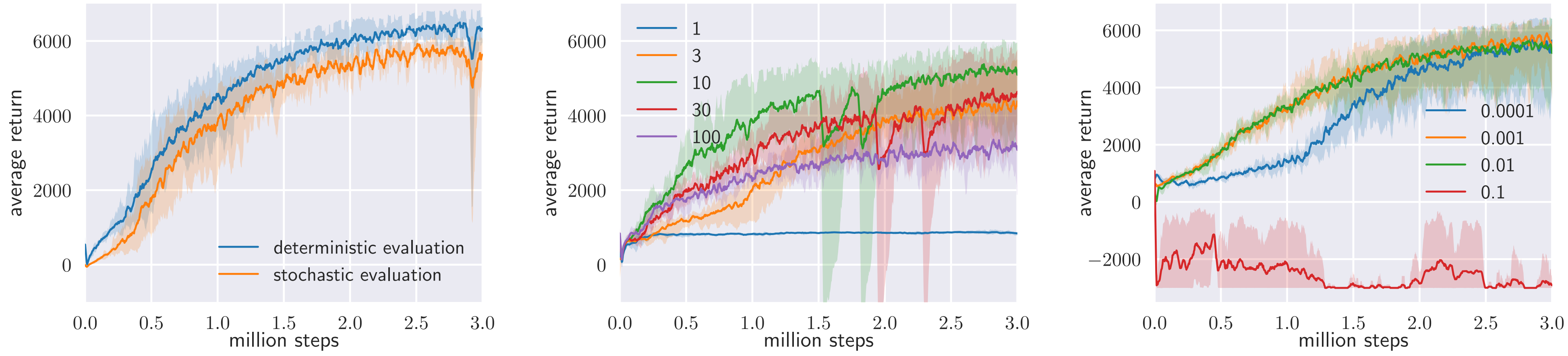


Figure 1: Training curves on continuous control benchmarks. Soft actor-critic (blue and yellow) performs consistently across all tasks and outperforming both on-policy and off-policy methods in the most challenging tasks.

Figure 1 of "Soft Actor-Critic Algorithms and Applications" by Tuomas Haarnoja et al.

Soft Actor Critic



(a) Evaluation (b) Reward Scale (c) Target Smoothing Coefficient (τ)

Figure 3. Sensitivity of soft actor-critic to selected hyperparameters on Ant-v1 task. (a) Evaluating the policy using the mean action generally results in a higher return. Note that the policy is trained to maximize also the entropy, and the mean action does not, in general, correspond the optimal action for the maximum return objective. (b) Soft actor-critic is sensitive to reward scaling since it is related to the temperature of the optimal policy. The optimal reward scale varies between environments, and should be tuned for each task separately. (c) Target value smoothing coefficient τ is used to stabilize training. Fast moving target (large τ) can result in instabilities (red), whereas slow moving target (small τ) makes training slower (blue).

Figure 3 of "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor" by Tuomas Haarnoja et al.