Continuous Actions, DDPG, TD3

Milan Straka

November 15, 2021

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

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

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

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

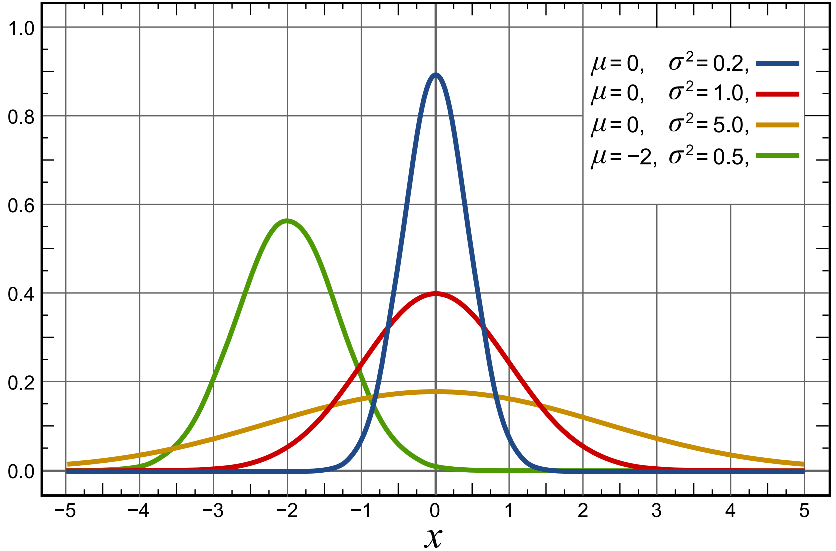


Figure from section 13.7 of "Reinforcement Learning: An Introduction, Second Edition".

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 regular 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 = tf.keras.layers.Dense(actions)(hidden_layer)
sds = tf.keras.layers.Dense(actions)(hidden_layer)
sds = tf.math.exp(sds)   # or sds = tf.math.softplus(sds)

action_dist = tfp.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.

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

Sometimes, the continuous action space is used even for discrete output -- when modeling pixels intensities (256 values) or sound amplitude (2^{16} values), instead of a softmax we use discretized mixture of distributions, usually $\operatorname{logistic}$ (a distribution with a sigmoid cdf). Then,

$$\pi(a) = \sum_i p_i \Big(\sigma\big((a + 0.5 - \mu_i)/\sigma_i\big) - \sigma\big((a - 0.5 - \mu_i)/\sigma_i\big)\Big).$$

However, such mixtures are usually used in generative modeling, not in reinforcement learning.

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 about which assumptions are required and when, can be found in Appendix B of *Deterministic Policy Gradient Algorithms: Supplementary Material* by David Silver et al.

Deterministic Policy Gradient Theorem – Proof

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

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:

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

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 predicted actions. Training is performed by Adam with learning rates of 1e-4 and 1e-3 for the policy and critic network, respectively.

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' 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 the paper "Continuous Control with Deep Reinforcement Learning" by Timothy P. Lillicrap et al.

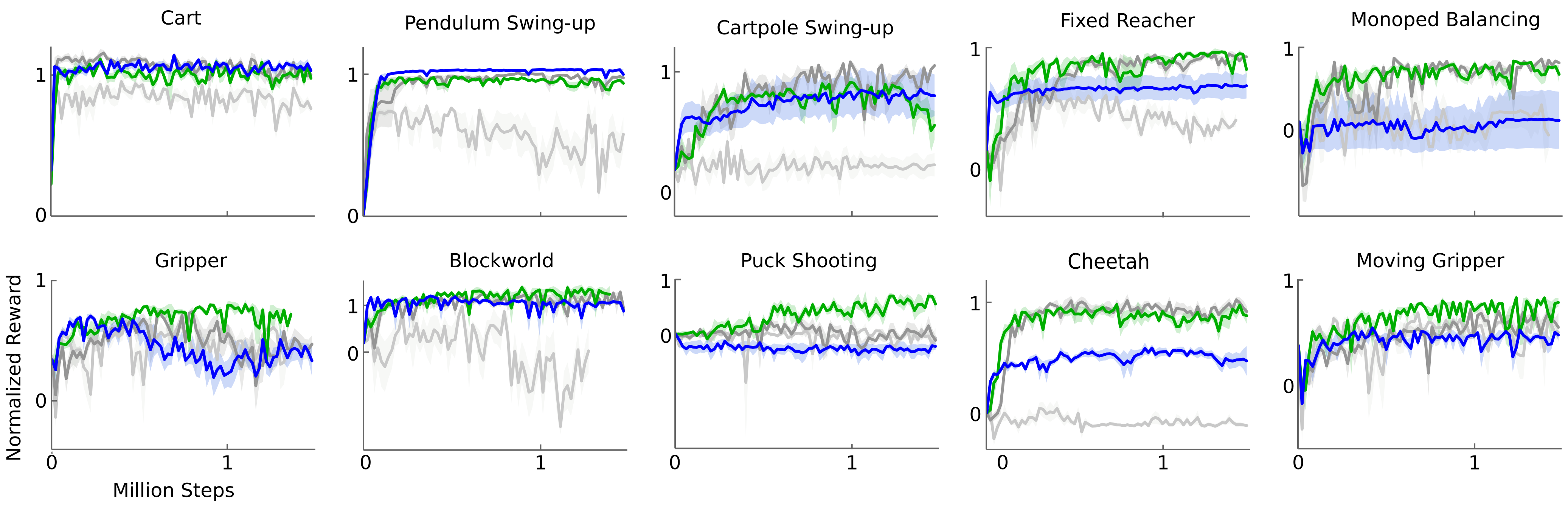


Figure 2: Performance curves for a selection of domains using variants of DPG: original DPG algorithm (minibatch NFQCA) with batch normalization (light grey), with target network (dark grey), with target networks and batch normalization (green), with target networks from pixel-only inputs (blue). Target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

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

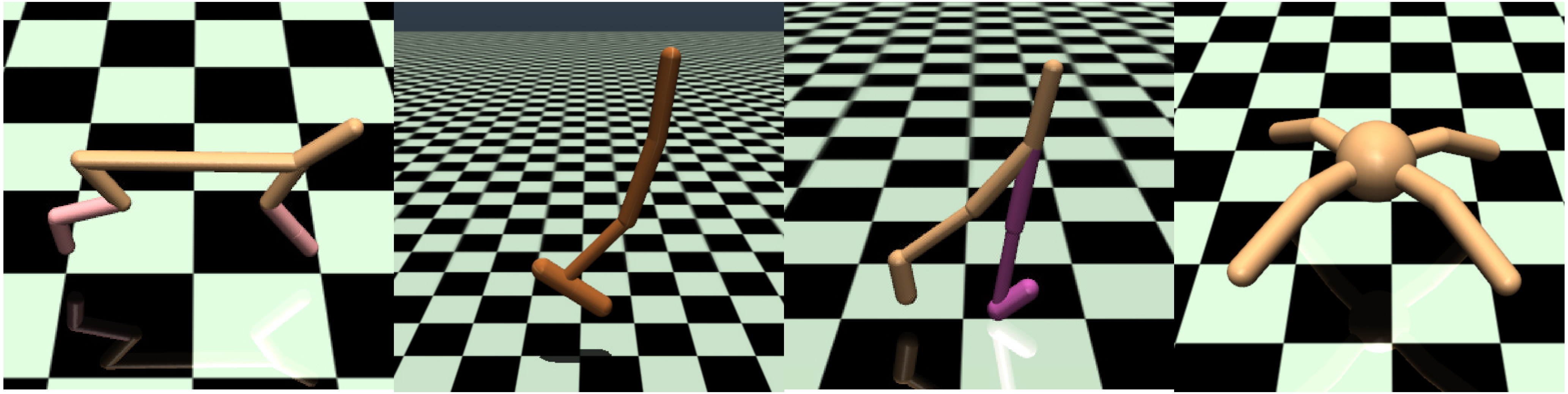
Ornstein-Uhlenbeck Exploration

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 $\tau = 0.15$ and $\sigma = 0.2$.



(a) (b) (c) (d)

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

Figure 4 of the paper "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 state-of-the-art algorithms for off-policy continuous-actions RL training (as of 2021).

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

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

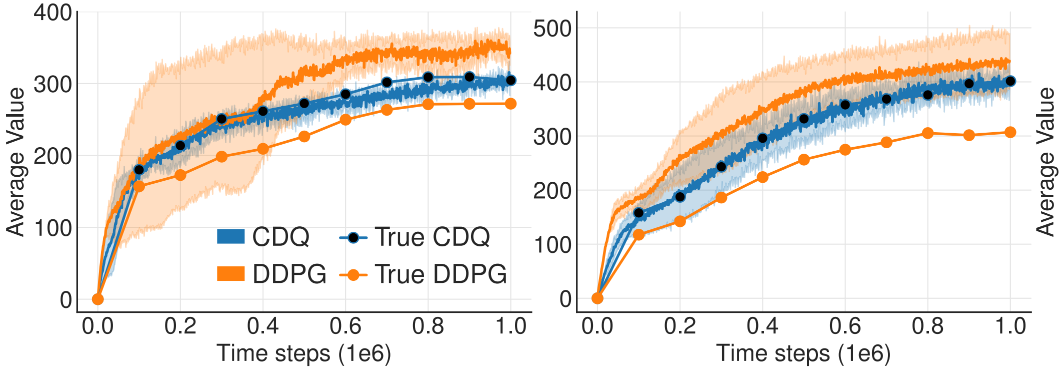
However, for real q_π and for sufficiently small $\alpha < \varepsilon_2$, it holds that

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

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

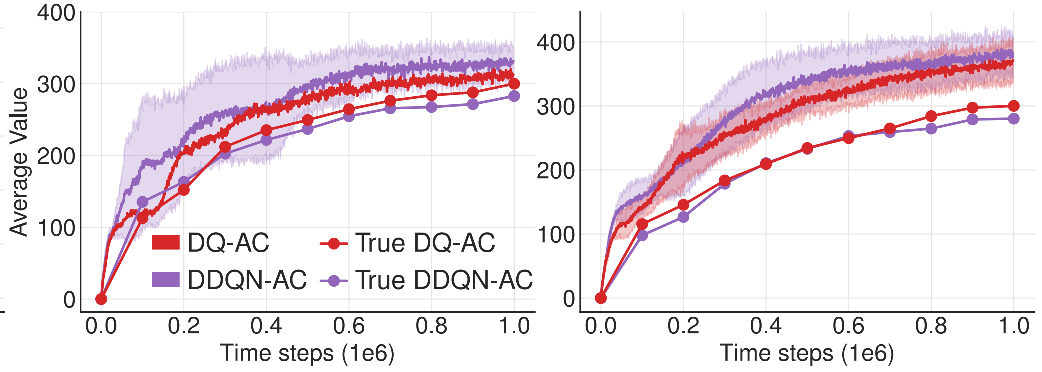
$$\mathbb{E}\big[q_{\boldsymbol{\theta}}(s, \pi_{approx})\big] \geq \mathbb{E}\big[q_\pi(s, \pi_{approx})\big].$$

TD3 – Maximization Bias



(a) Hopper-v1 (b) Walker2d-v1

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



(a) Hopper-v1 (b) Walker2d-v1

Figure 2 of the paper "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 target policy and 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

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

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 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_{θ_1}, Q_{θ_2}, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
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 $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \operatorname{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
  Update critics $\theta_i \leftarrow \operatorname{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
  if t mod d **then**
    Update ϕ 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 the paper "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 the paper "Addressing Function Approximation Error in Actor-Critic Methods" by Scott Fujimoto et al.

TD3 – Results

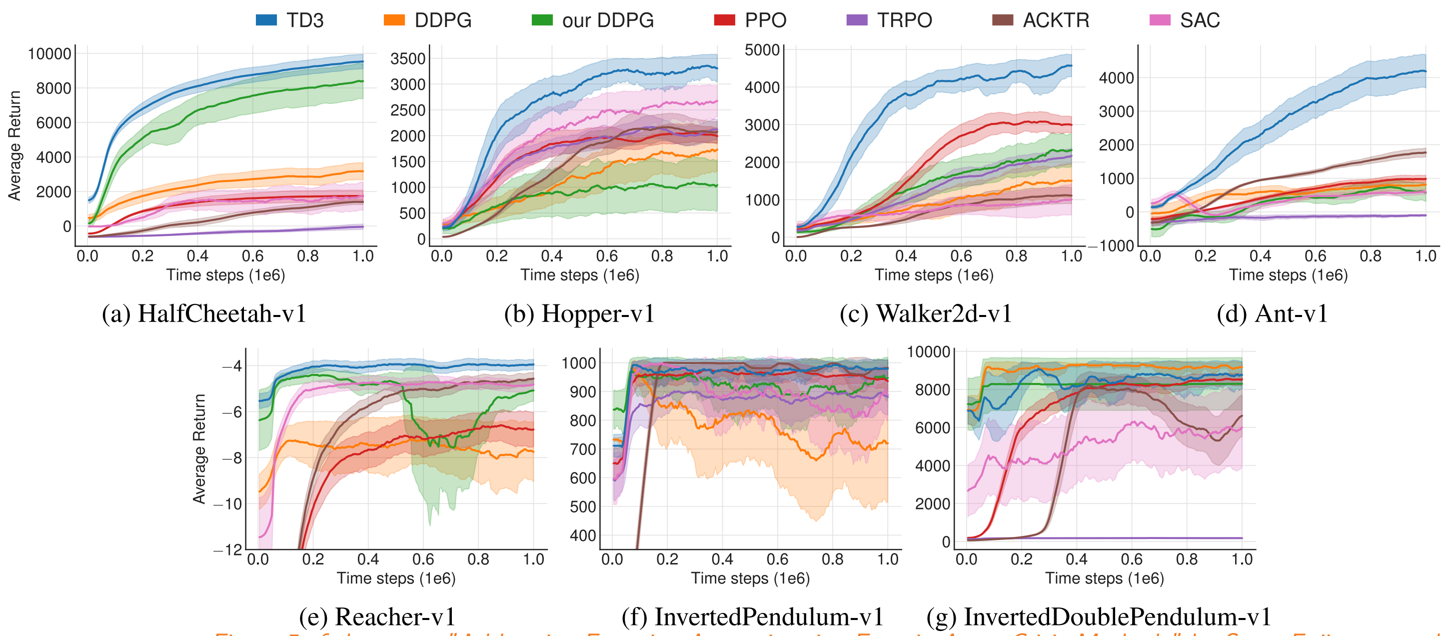


Figure 5 of the paper "Addressing Function Approximation Error in Actor-Critic Methods" by Scott Fujimoto et al.

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 the paper "Addressing Function Approximation Error in Actor-Critic Methods" by Scott Fujimoto et al.

TD3 – Ablations

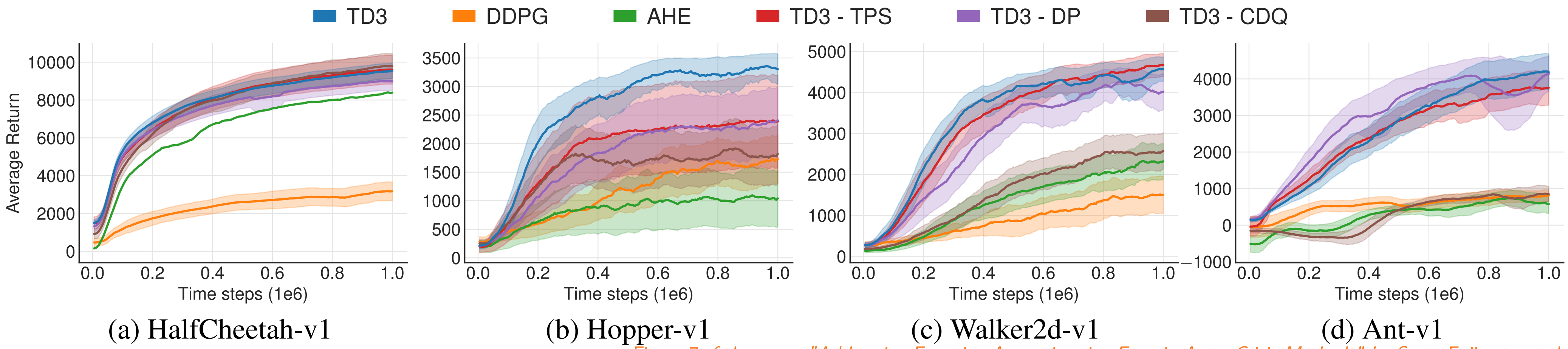


Figure 7 of the paper "Addressing Function Approximation Error in Actor-Critic Methods" by Scott Fujimoto et al.

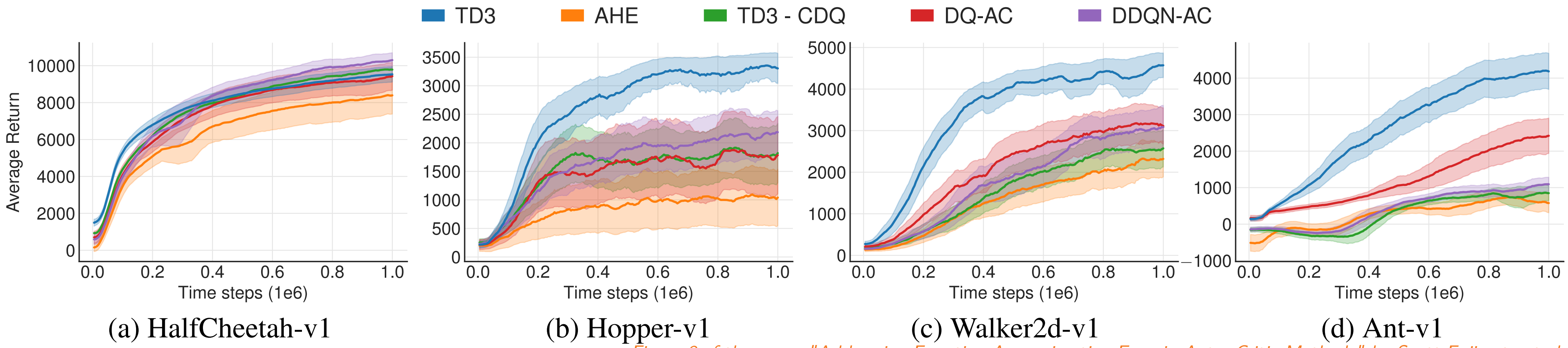


Figure 8 of the paper "Addressing Function Approximation Error in Actor-Critic Methods" by Scott Fujimoto et al.

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 the paper "Addressing Function Approximation Error in Actor-Critic Methods" by Scott Fujimoto et al.