MUNI
FI

DDPG & Friends

Vítek Unčovský

Faculty of Informatics, Masaryk University

March 28, 2025

Topics

- DDPG - extending DPG with neural network approximators (2015)
- TD3 - deals with overestimation bias & update variance in DDPG. (2018)
- SAC - AC method with an entropy-augmented objective (2018)

DDPG

- Lillicrap et al. 2015 - Continuous control with deep reinforcement learning - Arxiv
- Deep DPG - represent $Q-$values with neural networks.
- Resembles a “continuous-action” version of DQN.

DQN refresh

Good to look at DDPG with DQN in mind.

- QL loss - $(Q_\theta(s,a) - y)^2$, where $y = r + \gamma * max_{a'} Q_\theta(s', a')$
- DQN showed that it is possible to make deep value-based RL work through a series of tricks from supervised learning.
  1. Replay buffer - transitions (s, a, r, s') are stored in a circular buffer $\mathcal{B}$. QL loss optimized by sampling batches from $\mathcal{B}$
  2. Target networks - use second network $Q_{\theta'}$ to calculate targets, θ' set to θ every K learning steps.
- Continuous action spaces - need to handle the maximization in target. Discretization is infeasible (7 DoF, 3 bins $\approx$ 2000 actions).

DDPG

- Deterministic Actor-critic, $Q_\theta : S \times A \to \mathbb{R}$, $\mu_\phi : S \to A$.
- Learn a Q-function and a second network that is an approximate maximizer:
  1. $\mu_\phi(s) \approx \max_a Q_\theta(s, a)$
  2. $Q_\theta(s, a) \approx Q^{\mu_\phi}(s, a)$
- For both of the above, store target networks with parameters ϕ', θ' that are updated much slower.
- Both losses are optimized via SGD on batches from replay buffer, note the difference in L from DQN:
  1. $J(\phi) = \mathbb{E}_{s\sim\mathcal{B}}[\nabla_\phi \mu_\phi(s) \cdot \nabla_a Q_\theta(s, a)|_{a=\mu_\phi(s)}]$
  2. $L(\theta) = \mathbb{E}_{s,a,r,s'\sim\mathcal{B}}[(Q_\theta(s, a) - r - \gamma * Q_{\theta'}(s', \mu_{\phi'}(s')))^2]$
- $J(\phi)$ is used for improving the policy (refining the maximizer)

Algorithm - Summary

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

Implementation details

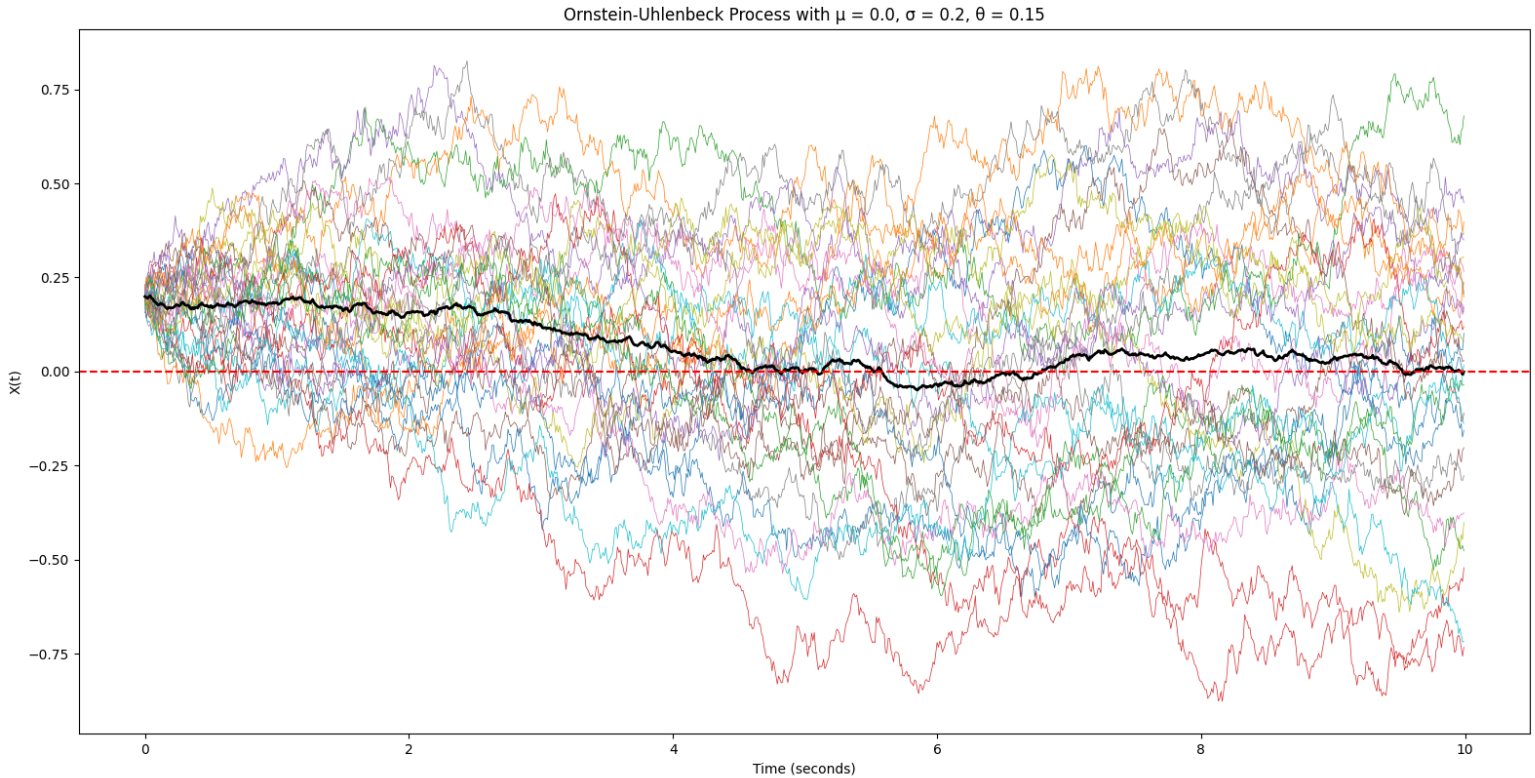
- Weight decay for Q net, action only included at second hidden layer of Q. Several architecture / weight init adjustments.
- Used **batch normalization** extensively:
  1. All layers of Q_θ until action input
  2. all layers of μ_ϕ.
- **Ornstein-Uhlenbeck** (OU) noise for exploration:

$$dX_t = \theta(\mu - X_t)dt + \sigma dW_t \tag{1}$$

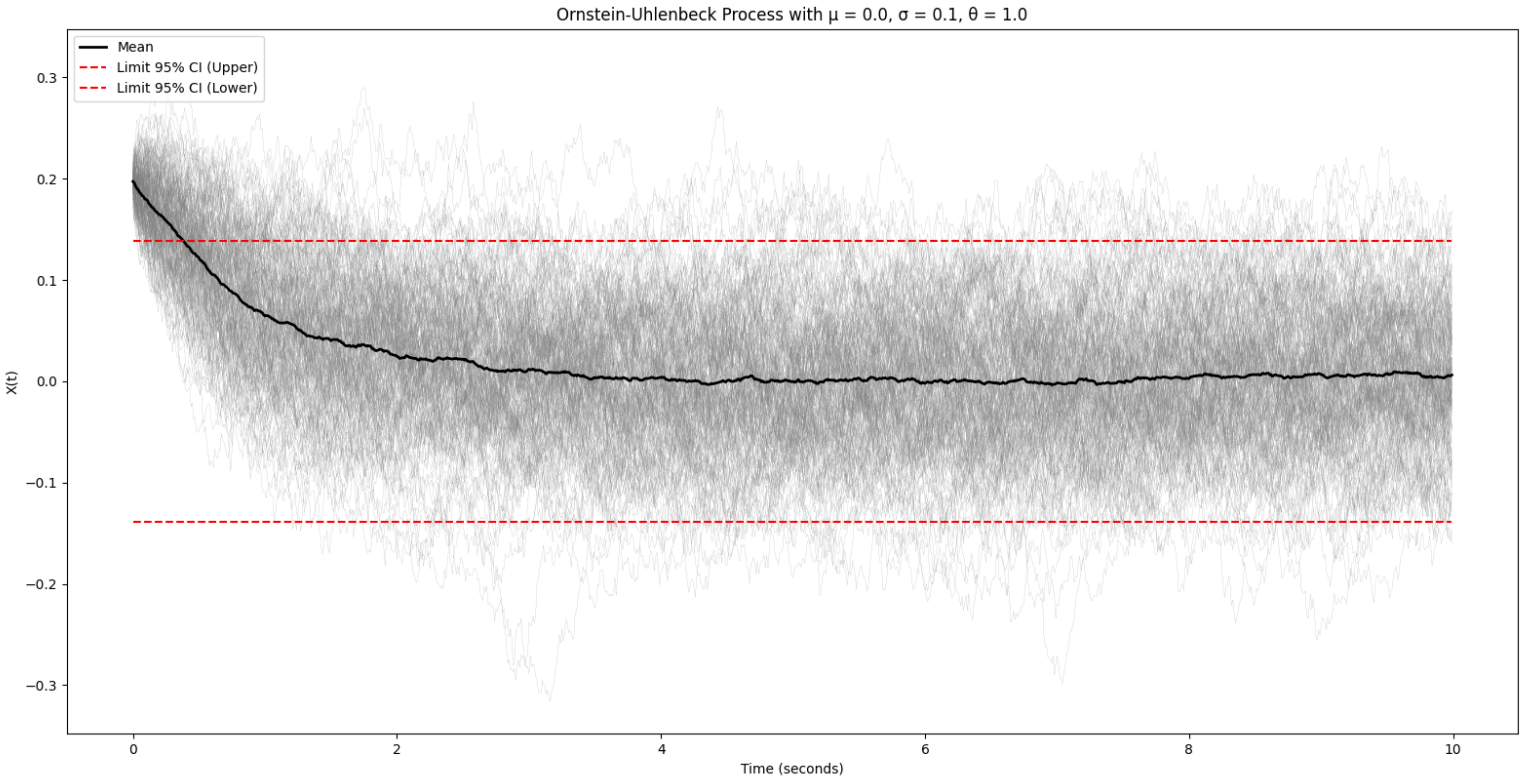
- Continuous Gaussian process, approaches μ at exponential rate θ, subject to gaussian noise.
- $X_{t+\Delta t} \leftarrow X_t + (\mu - X_t) \cdot \Delta t \ + \sigma \cdot \varepsilon$, where $\varepsilon \sim \mathcal{N}(0, \Delta t)$.

Later findings: Neither batch norm nor OU noise are necessary for good performance (Although ablation study in DDPG claims otherwise).

OU noise


Ornstein-Uhlenbeck Process with μ = 0.0, σ = 0.2, θ = 0.15
X(t)
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
0
2
4
6
8
10
Time (seconds)

OU noise


Ornstein-Uhlenbeck Process with μ = 0.0, σ = 0.1, θ = 1.0
Mean
Limit 95% CI (Upper)
Limit 95% CI (Lower)
X(t)
Time (seconds)
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0
2
4
6
8
10

Experiments

- Tested on tasks like Cartpole, Gripper, Puck Striking, Cheetah (MuJoCo).
- Normalized returns, 0 - Uniform policy, 1 - iLQG (model-based planner).
- Evaluated on high-dimensional (image-based) and low-dimensional (state-based) inputs.

Results and Observations

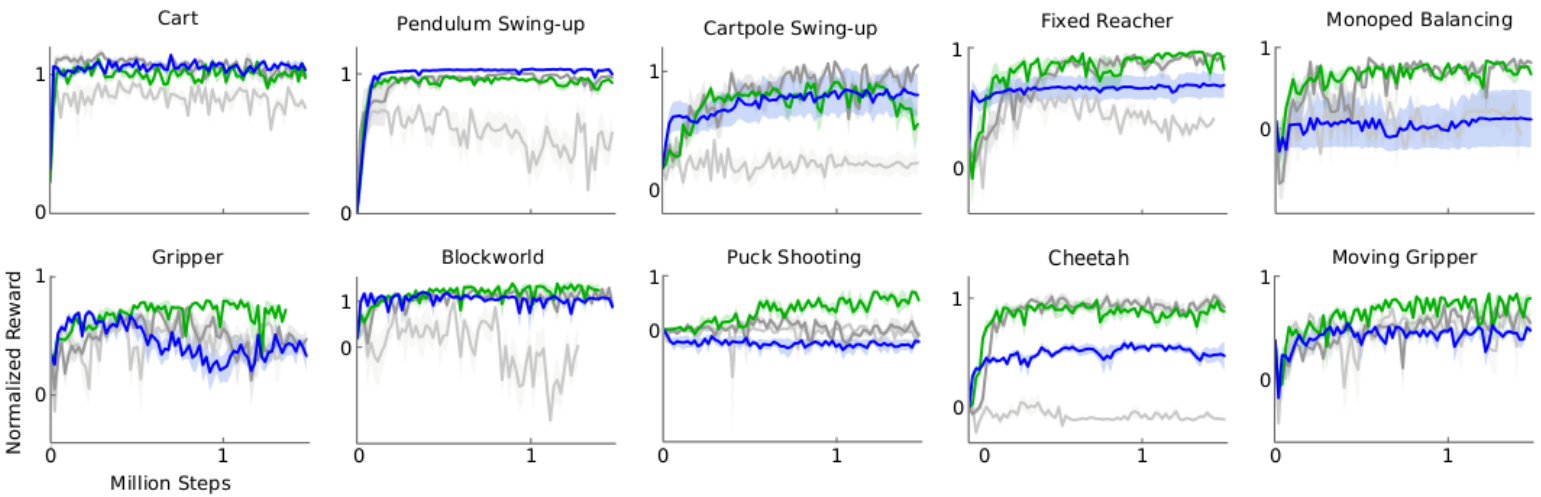


Figure 2: Performance curves for a selection of domains using variants of DPG: original DPG algorithm (minibatch NFQCA) with batch normalization (light grey), with target network (dark grey), with target networks and batch normalization (green), with target networks from pixel-only inputs (blue). Target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

Maximization Bias

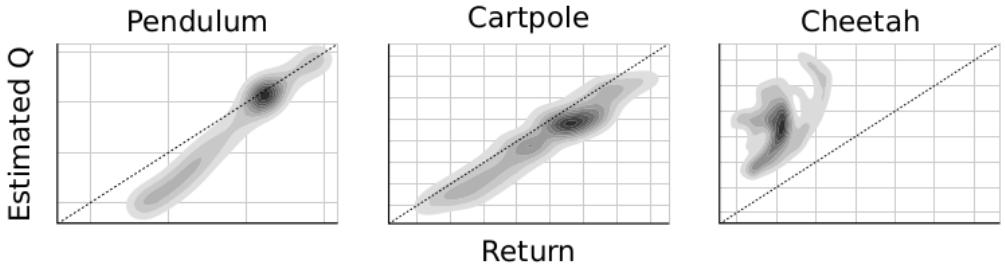


Figure 3: Density plot showing estimated Q values versus observed returns sampled from test episodes on 5 replicas. In simple domains such as pendulum and cartpole the Q values are quite accurate. In more complex tasks, the Q estimates are less accurate, but can still be used to learn competent policies. Dotted line indicates unity, units are arbitrary.

- $\approx$ Q-values larger than actual observed returns
- No double Q trick like in DQN, but this gets addressed in TD3.
- Bias not an issue on simple envs, does happen on harder MuJoCo envs though.

Conclusion

- Extends deterministic policy gradients with deep networks.
- Extends DQN to continuous, high-dimensional action spaces, leverages the same tricks.
- Replay buffer and target networks are crucial for stability.
- Still suffers from overestimation bias.

TD3

- *Twin Delayed DDPG*
- Addressing Function Approximation Error in AC methods - Fujimoto et al. 2018 - Arxiv
- DDPG is sensitive to hyperparameters, generally quite unstable.
- Improves DDPG by addressing overestimation bias and error propagation.
- Introduces three tricks which stabilize DDPG; these are still relevant for deep value-based RL.

TD3: Three Key Tricks

- **Clipped Double Q-learning**
  Conservative target calculation.
- **Delayed Policy Updates**
  Actor updates less frequently than critic.
- **Target Policy Smoothing**
  SARSA-like regularization (noisy target actions).
- These aid in reducing bias and variance in target calculations, which in turn reduces the amount of error propagated via bootstrapping.

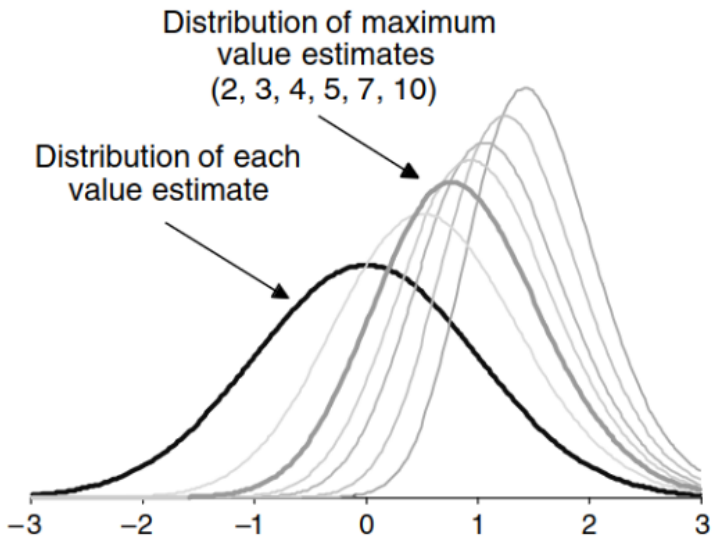
Overestimation Bias

- Known issue with value-based RL algorithms, some references:
  1. Issues in Using Function Approximation for Reinforcement Learning, Thrun & Schwartz 1993 - Link.
  2. Double Q-Learning, 2010 - Link.
  3. Double DQN, 2015 - Link.
- Essentially, the maximization $\max_{a'} Q(s', a')$ in the target leads to positive bias, even if estimates $Q(s', a')$ are unbiased.
- Particularly, if a function approximator is used, some degree of noise is inevitable.

Overestimation Bias - Example

- Consider same value $Q^*(s', a') = 0$ for every a', with estimates $Q(s', a')$ subject to zero-mean gaussian noise.
- Although $\max_{a'} \mathbb{E}(Q(s', a')) = 0$, $\mathbb{E} \max_{a'}(Q(s', a')) > 0$, since the maximum is sensitive to positive noise.

Figure 2 The Distribution of the Maximum of n Standard Normal Value Estimates



Number of alternatives	Expected disappointment
1	0.00
2	0.56
3	0.85
4	1.03
5	1.16
6	1.27
7	1.35
8	1.43
9	1.48
10	1.54

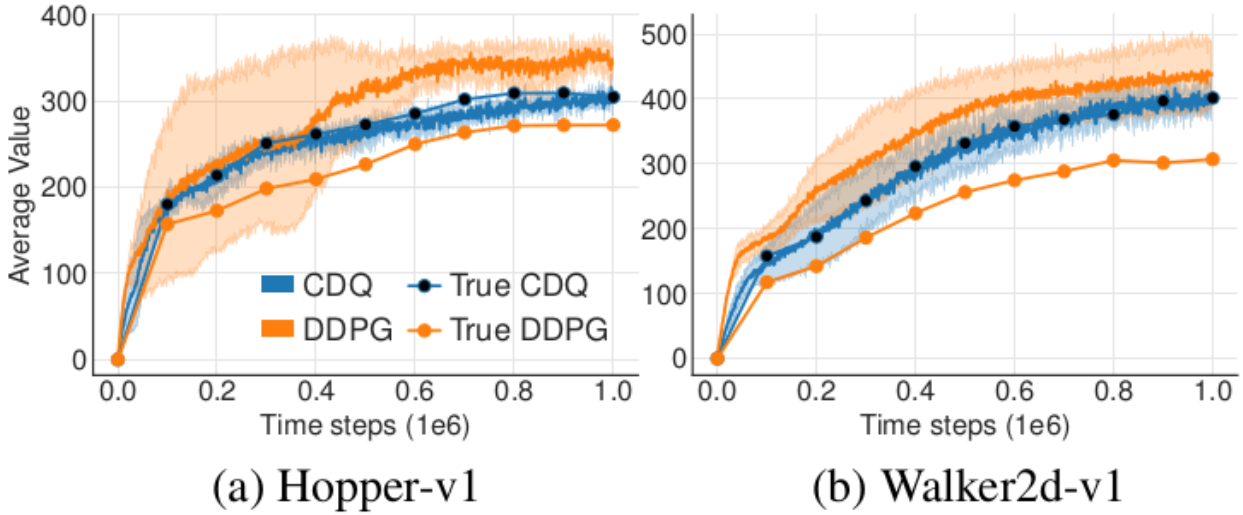
Overestimation Bias - cont.

- The bias is an issue, since it gets propagated by bootstrapping, leading to a feedback loop
- Overestimated Q-value $\rightarrow$ Policy selects wrong action, overestimated value gets propagated via bootstrapping, . . .
- Greatly destabilized training in DQN
- Solution - use different Q estimates to select and evaluate the action. Ideally, the estimates should be independent (Double Q-Learning), but using the target network (Double DQN) works quite well.
- Target calculation is changed to $r + \gamma \cdot Q_{\theta'}(s', argmax_{a'}(Q_{\theta}(s', a'))$

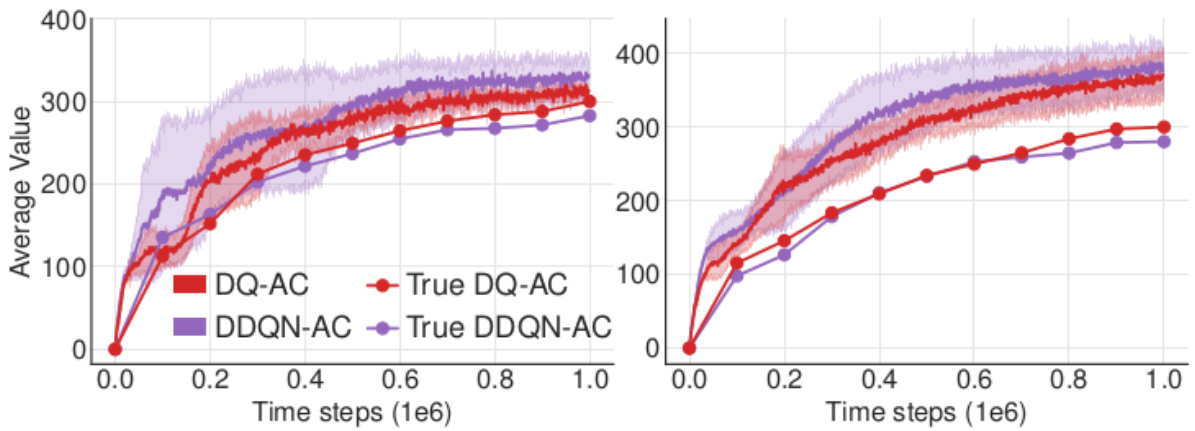
Overestimation bias in DDPG

- But is it even an issue here? We don't directly maximize as in the discrete case (DQN)
- TD3 authors show that overestimation still happens and hinders the performance of DDPG
- Authors try both Double-DQN and Double-QLearning (two independent estimates) to address this bias, but find that neither works very well.
- "*Unfortunately, due to the slow-changing policy in an actor-critic setting, the current and target value estimates remain too similar to avoid maximization bias.*"

Overestimation bias visualized



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



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

Clipped Q-Learning

- The authors propose to train two critics Q_{θ_1}, Q_{θ_2}, each with their own target.
- Both critics are trained using the same target $r + \gamma \cdot \min_i Q_{\theta'_i}(s', \mu_\phi(s'))$
- Pessimistic average - penalizes uncertain Q-values.
- > 2 critics do not add much benefit for computation.

D for Delayed

- Target networks crucial in reducing residual error after updates
- Reduce error in Q estimates over multiple updates before updating the actor
- The actor is updated every d learning steps, $d = 2$ in the authors' implementation.

Target Smoothing

- When calculating the maximizing action, add some noise from clipped gaussian $\varepsilon \sim \mathcal{N}(0, 0.2)$, clip $|\varepsilon| \leq 0.5$.
- Regularization, prior, similar actions should have similar values
- Should help the actor not overfit to small peaks in Q, assign higher value to actions resistant to perturbations

Algorithm Summary

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 $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
  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 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

Results

Table 1. Max Average Return over 10 trials of 1 million time steps. Maximum value for each task is bolded. ± corresponds to a single standard deviation over trials.

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

Ablation Study

Table 2. Average return over the last 10 evaluations over 10 trials of 1 million time steps, comparing ablation over delayed policy updates (DP), target policy smoothing (TPS), Clipped Double Q-learning (CDQ) and our architecture, hyper-parameters and exploration (AHE). Maximum value for each task is bolded.

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

Maximum Entropy RL

- $J_{ME}(\pi) = \mathbb{E}_\pi \sum_{t=0}^{T} \gamma^t (r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t))$
- Maximum entropy policies are more robust, objective encourages exploration, multiple modes
- Theoretical foundations in soft optimality
- SAC - maximum entropy off-policy actor-critic method - Arxiv.

Soft Policy Evaluation

- Q/V-values reflect new objective.
- $Q^\pi(s,a) = r(s,a) + \gamma \mathbb{E}_{s' \sim \delta(\cdot|s,a)}[V^\pi(s')]$
- $V^\pi(s,a) = \mathbb{E}_{a \sim \pi}[Q(s,a) - \log \pi(a|s)]$
- Can show Q fixpoint, via modified Bellman operator (augment rewards with policy entropy)

Soft Policy Improvement

- Given Q^π, how should π change to maximize value?
- Set $\pi(a|s) \propto \exp(Q^\pi(s,a))$, soft optimality
- For each state $\pi_{new} = \min_{\pi'} D_{KL}(\pi'(\cdot|s) \;||\; \frac{exp(Q^\pi(s,\cdot))}{Z^\pi(s)})$, where Z^π is the softmax normalizer.

Derivation - Soft Policy Improvement

- For simplicity, I'll fix $\alpha = 1$ here, otherwise Q values scaled by α^{-1}, the rest stays the same.
- Look at the "local" improvement $\max_{\pi_n} \mathbb{E}_{a\sim\pi_n(\cdot|s)}[Q^\pi(s,a) - \log \pi_n(a|s)]$
- Then for any s,

$$
\begin{aligned}
V^\pi(s) &= \mathbb{E}_{a\sim\pi}[Q^\pi(s,a) - \log \pi(a|s)] && (2)\\
&\leq \mathbb{E}_{a\sim\pi_n}[Q^\pi(s,a) - \log \pi_n(a|s)] && (3)\\
&= \mathbb{E}_{a\sim\pi_n}[r + \gamma \mathbb{E}_{s'\sim\delta(\cdot|s,a)}[V^\pi(s')] - \log \pi_n(a|s)] && (4)\\
&\leq \mathbb{E}_{a\sim\pi_n}[r + \gamma \mathbb{E}_{s'\sim\delta(\cdot|s,a),a'\sim\pi_n(\cdot|s')}[Q^\pi(s',a') && (5)\\
&\quad - \log \pi_n(a'|s')] - \log \pi_n(a|s)] && (6)\\
&\dots && (7)\\
&\leq V^{\pi_n}(s) && (8)
\end{aligned}
$$

Derivation 2 - Soft Policy Improvement

- For simplicity, I'll fix $\alpha = 1$ here, otherwise Q values scaled by α^{-1}, the rest stays the same.
- How does the policy π_n look like?

$$\max_{\pi_n} \mathbb{E}_{a\sim\pi_n(\cdot|s)}[Q^\pi(s,a) - \log \pi_n(a|s)] = \tag{9}$$

$$\min_{\pi_n} \mathbb{E}_{a\sim\pi_n(\cdot|s)}[\log \pi_n(a|s) - \log(\exp^\pi(s,a))] = \tag{10}$$

$$\min_{\pi_n} \mathbb{E}_{a\sim\pi_n(\cdot|s)}[\log Z^\pi(s) + \log \pi_n(a|s) - \log(\exp Q^\pi(s,a))] = \tag{11}$$

$$\min_{\pi_n} \mathbb{E}_{a\sim\pi_n(\cdot|s)}[\log \pi_n(a|s) - \log(\frac{\exp Q^\pi(s,a)}{Z^\pi(s)})] = \tag{12}$$

$$\min_{\pi_n} D_{KL}(\pi_n(\cdot|s) \;||\; \frac{\exp(Q^\pi(s,a))}{Z^\pi(s)}) \tag{13}$$

SAC

- SAC does the two previous steps approximately, utilizing off-policy data.
- Neural nets for $Q_{\theta_1}, Q_{\theta_2}, \pi_\phi$, along with targets for Q. In the original paper, also an additional V_ψ approximator, but later removed.
- $L_Q(\theta_1) = \mathbb{E}_{s,a,r,s' \sim \mathcal{B}}[(Q_{\theta_1}(s,a) - y)^2]$, where $y = r + \gamma \cdot (\min_i Q_{\theta'_i}(s', a') - \log \pi(a'|s'))$, a' is sampled from $\pi(\cdot|s')$ [1]
- Single sample estimate of V from Q, use clipped Q trick from TD3 & target nets.
- Adaptive temperature α optimization, set a target entropy value $\overline{\mathcal{H}}$

[1] The TD3 Q-value clipping helps a lot, see old version of the paper

Algorithm Summary

Algorithm 1 Soft Actor-Critic

Input: θ_1, θ_2, ϕ ▷ 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: θ_1, θ_2, ϕ ▷ Optimized parameters

Learning the Actor

- $J_\pi(\phi) = \mathbb{E}_{s\sim\mathcal{B},\, a\sim\pi_\phi(\cdot|s)}[\alpha \cdot \log \pi_\phi(a|s) - Q(s,a)]$ [2]
- Want to estimate gradient $\nabla_\phi J_\pi(\phi)$, reparametrize the expectation $\mathbb{E}_{a\sim\pi_\phi} \ldots = \mathbb{E}_\varepsilon \ldots$, $\bar{a} = f_\phi(s,\varepsilon)$
- Reparametrization trick, pathwise gradient estimator - see e.g. this. The sampling path f needs to be differentiable in ϕ.
- For example, consider $\pi_\phi(a|s)$ to be a Gaussian distribution with $\phi = (\mu, \sigma)$, then $f_\phi(s,\varepsilon) = \mu + \sigma \cdot \varepsilon$, $\varepsilon \sim N(0,1)$.
- $\nabla_\phi \mathbb{E}_{a\sim\pi_\phi} \ldots = \mathbb{E}_\varepsilon \nabla_\phi \alpha \log \pi_\phi(\bar{a}|s) + \nabla_\phi Q(s,\bar{a})$, $\bar{a} = \mu + \sigma \cdot \varepsilon$

[2] Again, minimum of both Q values is used here

Results

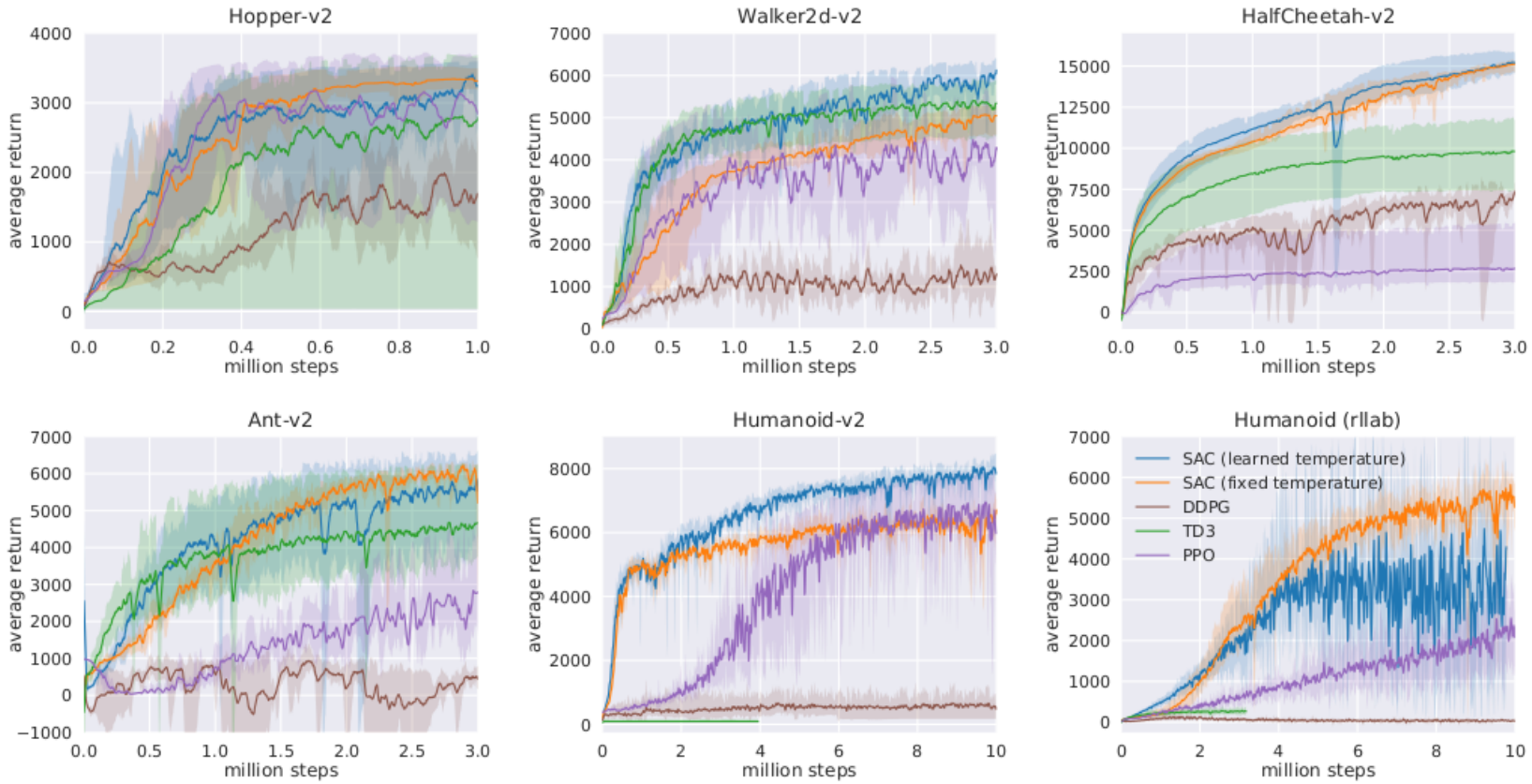


Figure 1: Training curves on continuous control benchmarks. Soft actor-critic (blue and yellow) performs consistently across all tasks and outperforming both on-policy and off-policy methods in the most challenging tasks.

Ablation Study

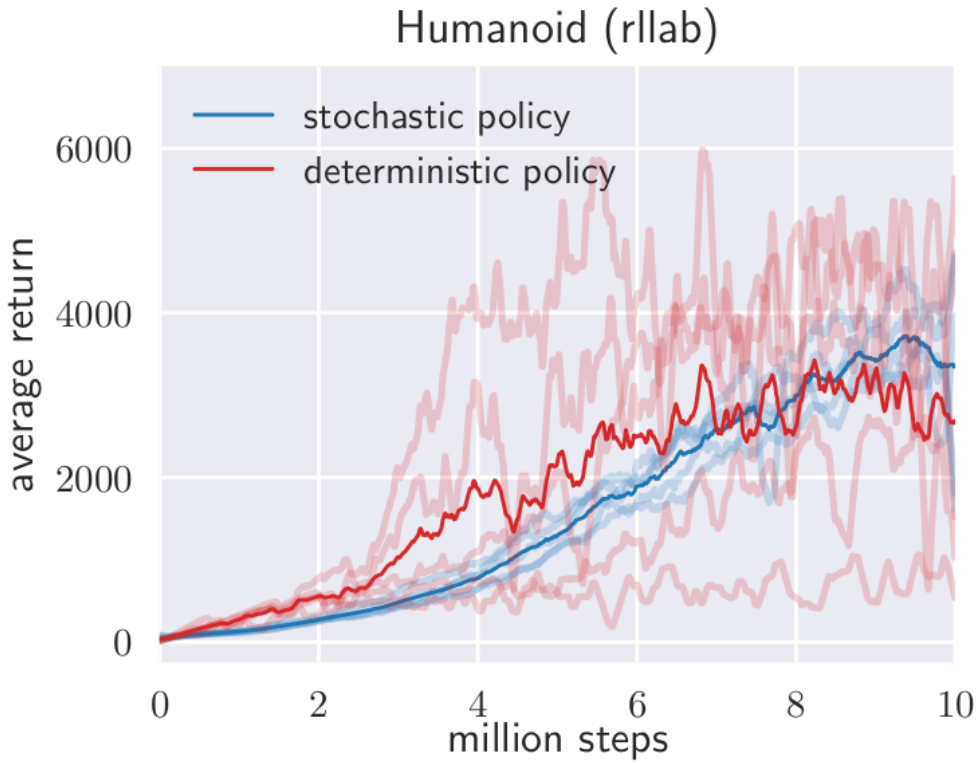


Figure 2. Comparison of SAC (blue) and a deterministic variant of SAC (red) in terms of the stability of individual random seeds on the Humanoid (rllab) benchmark. The comparison indicates that stochasticity can stabilize training as the variability between the seeds becomes much higher with a deterministic policy.

Maximum Entropy RL - resources

- If MaxEnt RL is the Answer, What is the Question? - ME RL policies robust to reward changes and under some partial observability.
- RL as probabilistic inference
- Composable Deep Reinforcement Learning for Robotic Manipulation, compose Q functions learned from multiple rewards/tasks via ME RL, then extract the policy for the sum of the rewards.
- Learning locomotion from real world interactions - A walk in the park, and safe exploration in training