Lecture 12: Deterministic Policy Gradient Methods

Oliver Wallscheid

Table of contents

Background and motivation

Recap on policy gradient so far:

- The previously discussed policy functions and the policy gradient theorem were assuming stochastic polices.
- The resulting on-policy algorithms may not provide top-class learning performance:
  - Non-guided exploration with step-by-step updates and
  - Greedy actions only in the limit (i.e., infeasible long learning).

The alternative:

- Apply a deterministic policy with separate exploration.
- Enable off-policy learning (with experience replay as a possible extension).
- Hence, we will focus on a deterministic policy function

$$\pi(\boldsymbol{x}, \boldsymbol{\theta}) = \mu(\boldsymbol{x}, \boldsymbol{\theta}). \tag{12.1}$$

Deterministic policy gradient (DPG) theorem

Theorem 12.1: Deterministic Policy Gradient

Given a metric $J(\boldsymbol{\theta})$ for the undiscounted episodic (11.7) or continuing tasks (11.8) and a parameterizable policy $\mu(\boldsymbol{x}, \boldsymbol{\theta})$ the deterministic policy gradient is

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\mu}\left[\nabla_{\boldsymbol{\theta}} \mu(\boldsymbol{x}, \boldsymbol{\theta}) \nabla_{\boldsymbol{u}} q(\boldsymbol{x}, \boldsymbol{u})|_{\boldsymbol{u}=\mu(\boldsymbol{x})}\right]. \tag{12.2}$$

- Again, q needs to be approximated using samples, e.g., implementing a critic via TD learning.
- It turns out that (12.2) is also (approximately) valid in the off-policy case, i.e., if the sample distribution is obtained from a behavior policy.
- Proof can be found in D. Silver et al., *Deterministic Policy Gradient Algorithms*, International Conference on Machine Learning, 2014

Exploration with a deterministic policy

- If the DPG approach is applied on-policy there is no inherent exploration.
- How to learn something?
  - The environment itself is sufficiently noisy (random impacts, measurement noise).
  - Or we have to add noise to the actions, i.e., making the approach off-policy.
  - Hence, utilizing a behavior policy is also possible.
- That additional action noise could be:
  - Simple Gaussian noise or
  - a shaped noise process like a discrete-time Ornstein-Uhlenbeck (OU) process

$$\nu_{k+1} = \lambda \nu_k + \sigma \epsilon_k$$

where ν_k is the OU noise output, $0 < \lambda < 1$ is a smoothing factor and σ is the variance scaling a standard Gaussian sequence (no mean) ϵ_k.

Algo. implementation: deterministic actor-critic

input: a differentiable deterministic policy function $\mu(\boldsymbol{x}, \boldsymbol{\theta})$
input: a differentiable action-value function $\hat{q}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w})$
parameter: step sizes $\{\alpha_w, \alpha_\theta\} \in \{\mathbb{R} | 0 < \alpha < 1\}$
init: parameter vectors $\boldsymbol{w} \in \mathbb{R}^\zeta$ and $\boldsymbol{\theta} \in \mathbb{R}^d$ arbitrarily
for $j = 1, 2, \dots,$ *episodes* **do**
    initialize $\boldsymbol{x}_0$;
    for $k = 0, 1, \dots, T-1$ *time steps* **do**
        $\boldsymbol{u}_k \leftarrow$ apply from $\mu(\boldsymbol{x}_k, \boldsymbol{\theta})$ w/wo noise or from behavior policy;
        observe $\boldsymbol{x}_{k+1}$ and r_{k+1};
        choose $\boldsymbol{u}'$ from $\mu(\boldsymbol{x}_{k+1}, \boldsymbol{\theta})$;
        $\delta \leftarrow r_{k+1} + \gamma \hat{q}(\boldsymbol{x}_{k+1}, \boldsymbol{u}', \boldsymbol{w}) - \hat{q}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{w})$;
        $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha_w \delta \nabla_{\boldsymbol{w}} \hat{q}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{w})$;
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_\theta \gamma^k \nabla_{\boldsymbol{\theta}} \mu(\boldsymbol{x}_k, \boldsymbol{\theta}) \nabla_{\boldsymbol{u}} \hat{q}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{w})|_{\boldsymbol{u}=\mu(\boldsymbol{x})}$;

Algo. 12.1: Deterministic actor-critic for episodic tasks using SARSA(0) targets applicable on- and off-policy (output: parameter vector $\boldsymbol{\theta}^*$ for $\mu^*(\boldsymbol{x}, \boldsymbol{\theta}^*)$) and $\boldsymbol{w}^*$ for $\hat{q}^*(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}^*)$)

Exemplary comparison to stochastic policy gradient

- DPG-based approach uses compatible function approximation, i.e., suitable linear $\hat{q}$ estimation. A fixed Gaussian behavior policy is applied for exploration.
- SAC uses a Gaussian policy with linear function approximation.

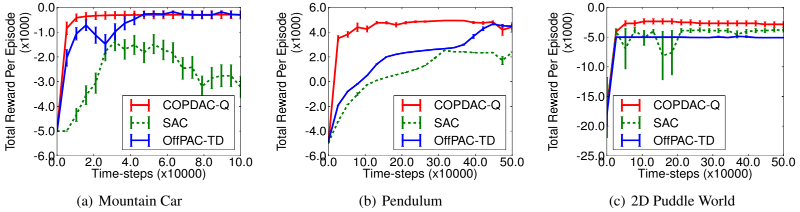

Fig. 12.1: Comparison of stochastic on-policy actor-critic (SAC), stochastic off-policy actor-critic (OffPAC), and deterministic off-policy actor-critic (COPDAC) on continuous-action reinforcement learning (source: D. Silver et al., *Deterministic Policy Gradient Algorithms*, International Conference on Machine Learning, 2014)

Table of contents

Motivation / general idea

- The upcoming deep deterministic policy gradient (DDPG) algorithm was very much inspired by the successes of DQNs (cf. Algo. 10.6 and landmark paper by Mnih et al.) on discrete action spaces.
- However, DQNs are not directly applicable to (quasi-)continuous action spaces.
- Recall the incremental Q-learning equation using function approximation

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \left[r + \gamma \max_{u} \hat{q}(\boldsymbol{x}', u, \boldsymbol{w}) - \hat{q}(\boldsymbol{x}, u, \boldsymbol{w}) \right] \nabla_{\boldsymbol{w}} \hat{q}(\boldsymbol{x}, u, \boldsymbol{w}).$$

- For every policy inference and updating step we need to find $\max_u \hat{q}(\boldsymbol{x}', u, \boldsymbol{w})$.
- If $u \in \mathcal{U} \subset \mathbb{Z}$ (i.e., using integer-encoded actions) is a sufficiently small discrete set, that is straightforward by an exhaustive search.
- In contrast, if $\boldsymbol{u} \in \mathcal{U} \subset \mathbb{R}^m$ is a (quasi-)continuous variable solving $\max_{\boldsymbol{u}} \hat{q}(\boldsymbol{x}', \boldsymbol{u}, \boldsymbol{w})$ requires an own optimization routine which is computationally expensive if we use nonlinear function approximation.

The deterministic policy trick

- When using a greedy, deterministic policy $\boldsymbol{\pi}(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{\mu}(\boldsymbol{x}, \boldsymbol{\theta})$ we can utilize it to approximate
$$\max_{\boldsymbol{u}} \hat{q}(\boldsymbol{x}', \boldsymbol{u}, \boldsymbol{w}) \approx \hat{q}(\boldsymbol{x}', \boldsymbol{\mu}(\boldsymbol{x}', \boldsymbol{\theta}), \boldsymbol{w}). \tag{12.3}$$
- Hence, we can obtain explicit Q-learning targets for continuous actions when using a deterministic policy.
- For improving the policy we reuse the deterministic policy gradient theorem in an off-policy fashion
$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_b \left[\nabla_{\boldsymbol{\theta}} \boldsymbol{\mu}(\boldsymbol{X}, \boldsymbol{\theta}) \nabla_{\boldsymbol{u}} q(\boldsymbol{X}, \boldsymbol{U}) \left| \boldsymbol{U} = \boldsymbol{\mu}(\boldsymbol{X}, \boldsymbol{\theta}) \right. \right] \tag{12.4}$$
given a behavior policy $b(\boldsymbol{u}|\boldsymbol{x})$.

DDPG $\approx$ DQN + DPG

- Hence, we can consider the DDPG approach as a combination of DQN + DPG rendering it an actor-critic off-policy approach for continuous state and action spaces.
- Similarly to DQN we will introduce several 'tweaks' to stabilize and improve the DDPG learning process.

Tweak #1: experience replay buffer

- We store $\langle \boldsymbol{x}, \boldsymbol{u}, r, \boldsymbol{x}' \rangle$ in $\boldsymbol{\mathcal{D}}$ after each transition step.
- The replay buffer $\boldsymbol{\mathcal{D}}$ is of limited capacity, i.e., it discards the oldest data sample when updating once it is full (ring memory).
- This allows us to improve the Q-learning critic minimizing the mean-squared Bellman error (MSBE):

$$\mathcal{L}(\boldsymbol{w}) = \left[\left(r + \gamma q(\boldsymbol{x}', \boldsymbol{\mu}(\boldsymbol{x}', \boldsymbol{\theta}), \boldsymbol{w})\right) - q(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w})\right]^2_{\boldsymbol{\mathcal{D}}}. \tag{12.5}$$

Additional DDPG tweaks (1)

Tweak #2: target networks

- Similar to DQN we introduce a (delayed) target network to estimate the Q-learning target

$$r + \gamma q(\boldsymbol{x}', \boldsymbol{\mu}(\boldsymbol{x}', \boldsymbol{\theta}), \boldsymbol{w})$$

  since it depends on the same parameters $\boldsymbol{w}$ which we want to update.
- Hence, the target network's purpose it to mimic the generation of i.i.d. data as the ground truth to minimize (12.5).
- Since the policy parameters $\boldsymbol{\theta}$ are also part of the target calculation it turns out that an additional policy target network is also beneficial to stabilize the Q-learning.
- In contrast to the classical DQN implementation, the original DDPG algorithm does not perform periodically hard target network updates but continuous ones using a low-pass filter characteristic

$$\boldsymbol{w}^- \leftarrow (1-\tau)\boldsymbol{w}^- + \tau\boldsymbol{w}, \quad \boldsymbol{\theta}^- \leftarrow (1-\tau)\boldsymbol{\theta}^- + \tau\boldsymbol{\theta} \tag{12.6}$$

  with τ representing the equivalent filter constant (hyperparameter).

Additional DDPG tweaks (2)

Tweak #3: mini-batch sampling

- Given a sufficiently filled memory $\boldsymbol{\mathcal{D}}$ and the target networks parametrized by $\boldsymbol{w}^-$ and $\boldsymbol{\theta}^-$ we draw uniformly distributed mini-batch samples $\boldsymbol{\mathcal{D}}_b$ from $\boldsymbol{\mathcal{D}}$.
- The actual Q-learning is then based on the loss

$$\mathcal{L}(\boldsymbol{w}) = \left[\left(r + \gamma q(\boldsymbol{x}', \boldsymbol{\mu}(\boldsymbol{x}', \boldsymbol{\theta}^-), \boldsymbol{w}^-)\right) - q(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w})\right]^2_{\boldsymbol{\mathcal{D}}_b} . \tag{12.7}$$

Tweak #4: batch normalization

- Minimizing (12.7) is a supervised learning step within the DDPG.
- The original DDPG paper by Lillicrap et al. back in 2015/16 suggested to use batch normalization, i.e., re-centering and re-scaling the inputs of each layer in an ANN.
- This idea of batch normalization was presented at that time shortly before by Ioffe and Szegedy (cf. original paper).
- Today's perspective: stick to the current state-of-the-art supervised ML algorithms for top-class Q-learning stability and speed (which are normally well-covered in popular supervised ML toolboxes).

Additional DDPG tweaks (3)

Tweak #5: exploration

- Since our policy is deterministic we require an exploratory behavior policy.
- Similar to DPG the standard approach is to add noise to the greedy actions, e.g., again from an Ornstein-Uhlenbeck (OU) process

$$\boldsymbol{u}_k \sim \boldsymbol{b}(\boldsymbol{u}|\boldsymbol{x}_k) = \boldsymbol{\mu}(\boldsymbol{x}_k, \boldsymbol{\theta}_k) + \boldsymbol{\nu}_k, \quad \boldsymbol{\nu}_k = \lambda \boldsymbol{\nu}_{k-1} + \sigma \boldsymbol{\epsilon}_{k-1}.$$

- One might also add a schedule for λ and σ along the training procedure, e.g., starting with significant noise levels (increased exploration) while reducing it over time (focusing exploitation)[1].
- However, many other behavior policies are possible, e.g., using model or expert-based guidance.

[1] Please note that this 'lambda' is not related to TD(λ), SARSA(λ), etc. Here, it is representing the stiffness of the OU noise process.

Visual summary of DDPG working principle

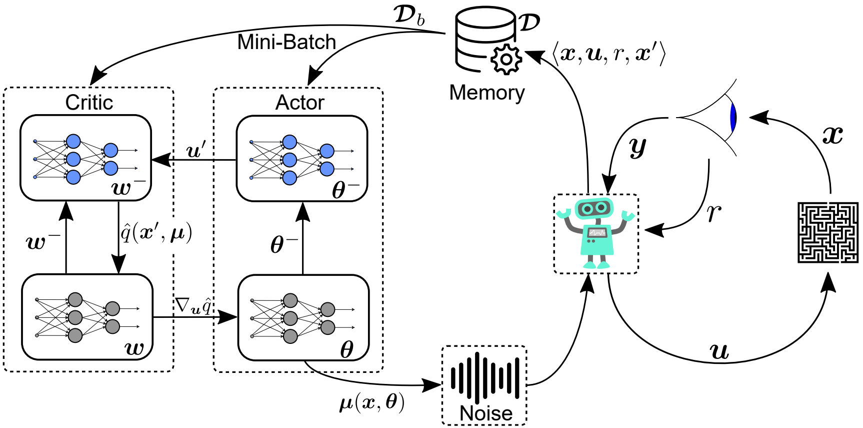

Fig. 12.2: DDPG structure from a bird's-eye perspective (derivative work of Fig. 1.1 and wikipedia.org, CC0 1.0)

Algo. implementation: DDPG

input: diff. deterministic policy function $\boldsymbol{\mu}(\boldsymbol{x}, \boldsymbol{\theta})$ and action-value function $\hat{q}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w})$
parameter: step sizes and filter constant $\{\alpha_w, \alpha_\theta, \tau\} \in \{\mathbb{R}|0 < \alpha, \tau < 1\}$
init: weights $\boldsymbol{w} = \boldsymbol{w}^- \in \mathbb{R}^\zeta$ and $\boldsymbol{\theta} = \boldsymbol{\theta}^- \in \mathbb{R}^d$ arbitrarily, memory $\boldsymbol{\mathcal{D}}$
for $j = 1, 2, \ldots,$ *episodes* **do**
  initialize $\boldsymbol{x}_0$;
  for $k = 0, 1, \ldots, T-1$ *time steps* **do**
    $\boldsymbol{u}_k \leftarrow$ apply from $\boldsymbol{\mu}(\boldsymbol{x}_k, \boldsymbol{\theta})$ w/wo noise or from behavior policy;
    observe $\boldsymbol{x}_{k+1}$ and r_{k+1};
    store tuple $\langle \boldsymbol{x}_k, \boldsymbol{u}_k, r_{k+1}, \boldsymbol{x}_{k+1} \rangle$ in $\boldsymbol{\mathcal{D}}$;
    sample mini-batch $\boldsymbol{\mathcal{D}}_b$ from $\boldsymbol{\mathcal{D}}$ (after initial memory warmup);
    for $i = 1, \ldots, b$ *samples* **do** calculate Q-targets
      if $\boldsymbol{x}_{i+1}$ *is terminal* **then** $y_i = r_{i+1}$;
      else $y_i = r_{i+1} + \gamma \hat{q}(\boldsymbol{x}_{i+1}, \boldsymbol{\mu}(\boldsymbol{x}_{i+1}, \boldsymbol{\theta}^-), \boldsymbol{w}^-)$;
    fit $\boldsymbol{w}$ on loss $\mathcal{L}(\boldsymbol{w}) = [y - \hat{q}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w})]^2_{\boldsymbol{\mathcal{D}}_b}$ with step size α_w;
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_\theta [\nabla_{\boldsymbol{\theta}} \boldsymbol{\mu}(\boldsymbol{x}, \boldsymbol{\theta}) \nabla_{\boldsymbol{u}} \hat{q}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w})|_{\boldsymbol{u}=\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{x})}]_{\boldsymbol{\mathcal{D}}_b}$;
    Update target net. $\boldsymbol{w}^- \leftarrow (1-\tau)\boldsymbol{w}^- + \tau \boldsymbol{w}, \ \boldsymbol{\theta}^- \leftarrow (1-\tau)\boldsymbol{\theta}^- + \tau \boldsymbol{\theta}$;

Algo. 12.2: Deep deterministic policy gradient (output: parameter vectors $\boldsymbol{\theta}^*$ for $\boldsymbol{\mu}^*(\boldsymbol{x}, \boldsymbol{\theta}^*)$)

Table of contents

Overestimation bias

- For Q-learning in the tabular case we have already discussed the maximization bias (cf. Fig. 5.13) issue.
- Recap: Due to the greedy policy targets, $\hat{q}$ was overestimated when calculated using sampled values of stochastic MDPs.
- Additional problem when applying function approximation: the estimator itself introduces additional variance during the learning process which represents another source of the maximization bias problem.

This issue is already known in the DQN context (cf. Algo. 10.6). Similar to the tabular case, double DQN introduces a second Q-network counteracting the overestimation issue (cf. paper by van Hasselt et al.).

However, we did not address this possible problem in an actor-critic context using function approximation (e.g., DDPG).

Overestimation bias in actor-critic approaches (1)

- It turns out that the overestimation bias is also an issue for actor-critic methods[1].
- Consider an actor-critic policy with the current policy parameters $\boldsymbol{\theta}$.
- Let $\tilde{\boldsymbol{\theta}}$ define the parameters from the actor update induced by the maximization of the approximate critic $\hat{q}_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{u})$.
- Let $\boldsymbol{\theta}^*$ be the parameters from the hypothetical actor update w.r.t. the true underlying value function $q^{\boldsymbol{\pi}}(\boldsymbol{x}, \boldsymbol{u})$.
- Then, we perform the policy update

$$\begin{aligned}
\tilde{\boldsymbol{\theta}} &= \boldsymbol{\theta} + \frac{\alpha}{Z_1}\mathbb{E}_\pi\left[\nabla_{\boldsymbol{\theta}}\pi_{\boldsymbol{\theta}}(\boldsymbol{X})\nabla_{\boldsymbol{u}}\hat{q}_{\boldsymbol{w}}(\boldsymbol{X}, \boldsymbol{U})\,|\boldsymbol{U} = \boldsymbol{\pi}_{\boldsymbol{\theta}}(\boldsymbol{X})\right], \\
\boldsymbol{\theta}^* &= \boldsymbol{\theta} + \frac{\alpha}{Z_2}\mathbb{E}_\pi\left[\nabla_{\boldsymbol{\theta}}\pi_{\boldsymbol{\theta}}(\boldsymbol{X})\nabla_{\boldsymbol{u}}q^{\boldsymbol{\pi}}(\boldsymbol{X}, \boldsymbol{U})\,|\boldsymbol{U} = \boldsymbol{\pi}_{\boldsymbol{\theta}}(\boldsymbol{X})\right],
\end{aligned} \tag{12.8}$$

where Z_1 and Z_2 normalize the gradient such that $Z^{-1}||\mathbb{E}\left[\cdot\right]|| = 1$.

[1]Source: S. Fujimoto et al., *Addressing Function Approximation Error in Actor-Critic Methods*, https://arxiv.org/abs/1802.09477, 2018

Overestimation bias in actor-critic approaches (2)

- Lets denote $\tilde{\boldsymbol{\pi}}$ and $\boldsymbol{\pi}^*$ as the policies with updated parameters $\tilde{\boldsymbol{\theta}}$ and $\boldsymbol{\theta}^*$ respectively.
- As the gradient direction is a local maximizer, there exists ϵ_1 sufficiently small such that if $\alpha \leq \epsilon_1$ then the *approximate* value of $\tilde{\boldsymbol{\pi}}$ will be bounded below by the *approximate* value of $\boldsymbol{\pi}^*$:

$$\mathbb{E}\left[\hat{q}_{\boldsymbol{w}}(\boldsymbol{X}, \tilde{\boldsymbol{\pi}}(\boldsymbol{X}))\right] \geq \mathbb{E}\left[\hat{q}_{\boldsymbol{w}}(\boldsymbol{X}, \boldsymbol{\pi}^*(\boldsymbol{X}))\right]. \tag{12.9}$$

- Conversely, there exists ϵ_2 sufficiently small such that if $\alpha \leq \epsilon_2$ then the *true* value of $\tilde{\boldsymbol{\pi}}$ will be bounded above by the *true* value of $\boldsymbol{\pi}^*$:

$$\mathbb{E}\left[q^{\boldsymbol{\pi}}(\boldsymbol{X}, \boldsymbol{\pi}^*(\boldsymbol{X}))\right] \geq \mathbb{E}\left[q^{\boldsymbol{\pi}}(\boldsymbol{X}, \tilde{\boldsymbol{\pi}}(\boldsymbol{X}))\right]. \tag{12.10}$$

- In other words: if the approximate and true critics differ from each other, the according policy gradient updates cannot lead to better policy updates of the respective other framework.

Overestimation bias in actor-critic approaches (3)

- If the expected, estimated action value will be at least as large as the *true* action value w.r.t. $\boldsymbol{\theta}^*$
$$\mathbb{E}\left[\hat{q}_{\boldsymbol{w}}(\boldsymbol{X}, \boldsymbol{\pi}^*(\boldsymbol{X}))\right] \geq \mathbb{E}\left[q^{\boldsymbol{\pi}}(\boldsymbol{X}, \boldsymbol{\pi}^*(\boldsymbol{X}))\right], \tag{12.11}$$
then (12.9) and (12.10) imply
$$\mathbb{E}\left[\hat{q}_{\boldsymbol{w}}(\boldsymbol{X}, \tilde{\boldsymbol{\pi}}(\boldsymbol{X}))\right] \geq \mathbb{E}\left[q^{\boldsymbol{\pi}}(\boldsymbol{X}, \tilde{\boldsymbol{\pi}}(\boldsymbol{X}))\right] \tag{12.12}$$
with a sufficiently small $\alpha < \min\{\epsilon_1, \epsilon_2\}$.
- Hence, the maximization bias is also present in actor-critic updates.
- It can add up over several estimation updates and, therefore, may lead to suboptimal policy updates.
- A proof for unnormalized gradients can be also found in S. Fujimoto et al., *Addressing Function Approximation Error in Actor-Critic Methods*, 2018.

Overestimation example for DDPG

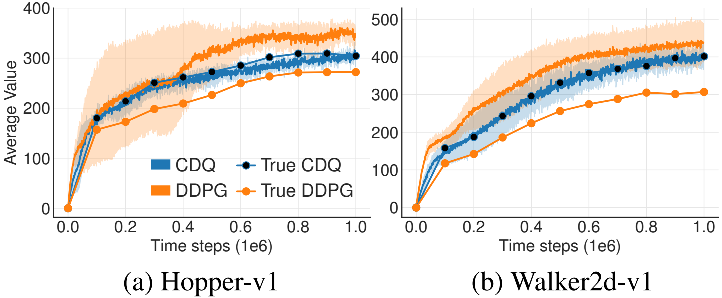

Fig. 12.3: Comparison of true and estimated values averaged over 10000 states in two robotic examples from OpenAI Gym. Estimated values originate from the approximate DDPG critic while the true values are based on the average discounted return over 1000 episodes following the current policy, starting from states sampled from the replay buffer (source: S. Fujimoto et al., *Addressing Function Approximation Error in Actor-Critic Methods*, 2018.

Increased variance due to accumulating TD errors

- Using function approximation, the Bellman equation is never exactly satisfied leaving room for some amount of residual TD-error $\tilde{\delta}(\boldsymbol{x}, \boldsymbol{u})$:

$$\hat{q}_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{u}) = r + \gamma \mathbb{E}_\pi \left[\hat{q}_{\boldsymbol{w}}(\boldsymbol{X}', \boldsymbol{U}') | \boldsymbol{X}' = \boldsymbol{x}', \boldsymbol{U}' = \boldsymbol{u}' \right] - \tilde{\delta}(\boldsymbol{x}, \boldsymbol{u}). \tag{12.13}$$

- Although this error might be considered small per update step, it may accumulate over future steps if biased:

$$\hat{q}_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{u}) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k \left(R_k - \tilde{\delta}_k(\boldsymbol{X}, \boldsymbol{U}) \right) \middle| \boldsymbol{X} = \boldsymbol{x}, \boldsymbol{U} = \boldsymbol{u} \right]. \tag{12.14}$$

- Observation: the variance of $\hat{q}$ will be proportional to the variance of future reward and residual TD-errors.
- If γ is large, the estimation variance might increase significantly.
- Mini-batch sampling will contribute to this variance issue.

TD3 extensions and modifications (1)

In order to reduce both the maximization bias and the learning variance, TD3 introduces mainly three measures on top of the DDPG algorithm. Hence, TD3 is a direct successor of DDPG.

Measure #1: clipped double Q-learning for actor-critic

- Following double Q-learning, a pair of critics $\{\hat{q}_{\boldsymbol{w}_1}, \hat{q}_{\boldsymbol{w}_2}\}$ is introduced.
- In contrast, the clipped target (with target networks $\{\boldsymbol{w}_1^-, \boldsymbol{w}_2^-\}$)

$$y = r + \gamma \min_{i=1,2} \hat{q}_{\boldsymbol{w}_i^-}(\boldsymbol{x}', \boldsymbol{u}') \tag{12.15}$$

  provides an upper-bound on the estimated action value.
- May introduce some underestimation, which is considered less critical than overestimation, since the value of underestimated actions will not be explicitly propagated through the policy update.
- The $\min$ operator will also (indirectly) favor actions leading to values with estimation errors of lower variance.

TD3 extensions and modifications (2)

Measure #2: target policy smoothing regularization

- Background: deterministic policies $\boldsymbol{\mu}$ tend to overfit to narrow peaks in the action-value estimate.
- Counteraction: fit the action value of a small area around the target action (i.e., smoothing $\hat{q}$ in the action space):

$$y = r + \gamma \hat{q}_{\boldsymbol{w}^-}(\boldsymbol{x}', \boldsymbol{\mu}_{\boldsymbol{\theta}^-}(\boldsymbol{x}') + \boldsymbol{\epsilon}). \tag{12.16}$$

- Here, $\boldsymbol{\epsilon} \sim \mathrm{clip}\left(\mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}), -\boldsymbol{c}, \boldsymbol{c}\right)$ is a mean-free, Gaussian noise with covariance $\boldsymbol{\Sigma}$, which is clipped at $\pm\boldsymbol{c}$ while $\boldsymbol{\theta}^-$ are the policy target network parameters.
- To satisfy possible action constraints (denoted by upper and lower box constraints $\{\underline{\boldsymbol{u}}, \overline{\boldsymbol{u}}\}$), we add an additional clipping:

$$\boldsymbol{u}' = \mathrm{clip}\left(\boldsymbol{\mu}_{\boldsymbol{\theta}^-}(\boldsymbol{x}') + \boldsymbol{\epsilon}, \underline{\boldsymbol{u}}, \overline{\boldsymbol{u}}\right). \tag{12.17}$$

- This modified action is then used for the target calculation (12.15).

TD3 extensions and modifications (3)

Measure #3: delayed policy updates

- Similar to DDPG, TD3 uses policy target networks $\boldsymbol{\theta}^-$ and (two) critic target networks $\{\boldsymbol{w}_1^-, \boldsymbol{w}_2^-\}$ in order to provide (rather) fixed Q-learning targets trying to stabilize the learning of $\hat{q}$.
- The target networks are also continuously updated using

$$\boldsymbol{w}_i^- \leftarrow (1-\tau)\boldsymbol{w}_i^- + \tau \boldsymbol{w}_i, \quad \boldsymbol{\theta}^- \leftarrow (1-\tau)\boldsymbol{\theta}^- + \tau\boldsymbol{\theta}.$$

- However, each policy update will inherently change the (true) Q-learning target directly adding variance to the learning process (cf. Fig. 12.4 on next slide).
- Therefore, it is argued that a policy update should not follow after each Q-learning update such that the critic can adapt properly to the previous policy update.
- The original TD3 implementation suggests a policy update every second Q-learning update, however, we can consider this update rate a hyperparameter.

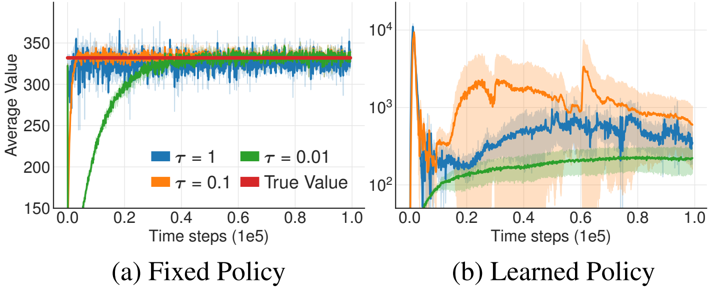

Fig. 12.4: Average estimated action value of a randomly selected state on Hopper-v1 environment from OpenAI Gym (source: S. Fujimoto et al., *Addressing Function Approximation Error in Actor-Critic Methods*, 2018.

input: diff. deterministic policy function $\boldsymbol{\mu}(\boldsymbol{x}, \boldsymbol{\theta})$ and action-value function $\hat{q}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w})$
parameter: step sizes and filter constant $\{\alpha_w, \alpha_\theta, \tau\} \in \{\mathbb{R}|0 < \alpha, \tau < 1\}$, policy update rate $k_w \in \{\mathbb{N}|1 \leq k_w\}$, target noise $\boldsymbol{\Sigma} \in \mathbb{R}^{m \times m}$ and $\boldsymbol{c} \in \mathbb{R}^m$
init: weights $\{\boldsymbol{w}_1 = \boldsymbol{w}_1^-, \boldsymbol{w}_2 = \boldsymbol{w}_2^-\} \in \mathbb{R}^\zeta$, $\boldsymbol{\theta} = \boldsymbol{\theta}^- \in \mathbb{R}^d$ arbitrarily, memory $\boldsymbol{\mathcal{D}}$
for $j = 1, 2, \ldots,$ *episodes* **do**
    initialize $\boldsymbol{x}_0$;
    for $k = 0, 1, \ldots, T-1$ *time steps* **do**
        $\boldsymbol{u}_k \leftarrow$ apply from $\boldsymbol{\mu}(\boldsymbol{x}_k, \boldsymbol{\theta})$ w/wo noise or from behavior policy;
        observe $\boldsymbol{x}_{k+1}$ and r_{k+1};
        store tuple $\langle \boldsymbol{x}_k, \boldsymbol{u}_k, r_{k+1}, \boldsymbol{x}_{k+1} \rangle$ in $\boldsymbol{\mathcal{D}}$;
        sample mini-batch $\boldsymbol{\mathcal{D}}_b$ from $\boldsymbol{\mathcal{D}}$ (after initial memory warmup);
        for $i = 1, \ldots, b$ *samples* **do** calculate Q-targets
            if $\boldsymbol{x}_{i+1}$ *is terminal* **then** $y_i = r_{i+1}$;
            else
                $\boldsymbol{u}' = \mathrm{clip}\left(\boldsymbol{\mu}_{\boldsymbol{\theta}^-}(\boldsymbol{x}_{i+1}) + \mathrm{clip}\left(\mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}), -\boldsymbol{c}, \boldsymbol{c}\right), \underline{\boldsymbol{u}}, \overline{\boldsymbol{u}}\right)$;
                $y_i = r_{i+1} + \gamma \min_{l=1,2} \hat{q}(\boldsymbol{x}_{i+1}, \boldsymbol{u}', \boldsymbol{w}_l^-)$;
        fit $\boldsymbol{w}_l$ on loss $\mathcal{L}(\boldsymbol{w}_l) = [y - \hat{q}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}_l)]^2_{\boldsymbol{\mathcal{D}}_b}$ with step size α_w $\forall\, l$;
        if $k \mod k_w = 0$ **then**
            $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_\theta [\nabla_{\boldsymbol{\theta}} \boldsymbol{\mu}(\boldsymbol{x}, \boldsymbol{\theta}) \nabla_{\boldsymbol{u}} \hat{q}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}_1)|_{\boldsymbol{u}=\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{x})}]_{\boldsymbol{\mathcal{D}}_b}$;
            $\boldsymbol{w}_l^- \leftarrow (1-\tau)\boldsymbol{w}_l^- + \tau \boldsymbol{w}_l, \; \boldsymbol{\theta}^- \leftarrow (1-\tau)\boldsymbol{\theta}^- + \tau \boldsymbol{\theta}$;

Algo. 12.3: Twin delayed deep deterministic policy gradient (TD3)

Summary: what you've learned today

- The deep deterministic policy gradient (DDPG) approach 'transfers' many deep Q-network (DQN) ideas to continuous action spaces.
- It mainly combines DQN + deterministic policy gradients + policy and value target networks (plus additional minor tweaks).
- However, the DDPG actor-critic suffers from value overestimation and high variance during learning. Hence, sampled policy gradients might not be optimal (pointing towards overrated action values).
- Twin delayed DDPG (TD3) adds clipped double Q-learning, delayed policy updates and target policy smoothing to counteract these issues.