Adaptively Calibrated Critic Estimates for Deep Reinforcement Learning

Nicolai Dorka **Joschka Bödecker** **Wolfram Burgard**
University of Freiburg
dorka@cs.uni-freiburg.de

Abstract

Accurate value estimates are important for off-policy reinforcement learning. Algorithms based on temporal difference learning typically are prone to an over- or underestimation bias building up over time. In this paper, we propose a general method called Adaptively Calibrated Critics (ACC) that uses the most recent high variance but unbiased on-policy rollouts to alleviate the bias of the low variance temporal difference targets. We apply ACC to Truncated Quantile Critics [22], which is an algorithm for continuous control that allows regulation of the bias with a hyperparameter tuned per environment. The resulting algorithm adaptively adjusts the parameter during training rendering hyperparameter search unnecessary and sets a new state of the art on the OpenAI gym continuous control benchmark among all algorithms that do not tune hyperparameters for each environment. Additionally, we demonstrate that ACC is quite general by further applying it to TD3 [11] and showing an improved performance also in this setting.

1 Introduction

Off-policy reinforcement learning is an important research direction as the reuse of old experience promises to make these methods more sample efficient than their on-policy counterparts. This is an important property for many applications such as robotics where interactions with the environment are very time- and cost-intensive. Many successful off-policy methods make use of a learned Q-value function [11, 14, 18, 27]. If the action space is discrete the Q-function can be directly used to generate actions while for continuous action spaces it is usually used in an actor-critic setting where the policy is trained to choose actions that maximize the Q-function. In both cases accurate estimates of the Q-values are of crucial importance.

Unfortunately, learning the Q-function off-policy can lead to an overestimation bias [33]. Especially when a nonlinear function approximator is used to model the Q-function, there are many potential sources of bias. Different heuristics were proposed for their mitigation, such as the double estimator in the case of discrete action spaces [35] or taking the minimum of two estimates in the case of continuous actions [11]. While these methods successfully prevent extreme overestimation due to their coarse nature, they can still induce under- or overestimation bias to a varying degree depending on the environment [23].

To overcome these problems we propose a principled and general method to alleviate the bias called Adaptively Calibrated Critics (ACC). Our algorithm uses the most recent on-policy rollouts to determine the current bias of the Q-estimates and adjusts a bias controlling parameter accordingly. This parameter adapts the size of the temporal difference (TD) targets such that the bias can be corrected in the subsequent updates. As the parameter changes slower than the rollout returns, our method still benefits from stable and low-variance temporal difference targets, while it incorporates the information from unbiased but high variance samples from the recent policy to reduce the bias.

Deep Reinforcement Learning Workshop, 35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia.

We apply ACC to Truncated Quantile Critics (TQC) [22], which is a recent off-policy actor-critic algorithm for continuous control showing strong performance on various tasks. In TQC the bias can be controlled in a finegrained way with the help of a hyperparameter that has to be tuned for every environment. ACC allows to automatically adjusts this parameter online during the training in the environment. As a result, it eliminates the need to tune this hyperparameter in a new environment, which is very expensive or even infeasible for many applications.

We evaluate our algorithm on a range of continuous control tasks from OpenAI gym [4] and exceed the current state of the art results among all algorithms that do not need tuning of environment specific hyperparameters. For each environment ACC matches the performance of TQC with the optimal hyperparameter for that environment. Further, we show that the automatic bias correction allows to increase the number of value function updates performed per environment step, which results in even larger performance gains in the sample-efficient regime. We additionally apply ACC to the TD3 algorithm [11] where it also leads to notably improved performance, underscoring the generality of our proposed method.

To summarize, the main contributions of this work are:

1. We propose Adaptively Calibrated Critics, a new general algorithm that reduces the bias of value estimates in a principled fashion with the help of the most recent unbiased on-policy rollouts.
2. As a practical implementation we describe how ACC can be applied to learn a bias controlling hyperparameter of the TQC algorithm and show that the resulting algorithm sets a new state of the art on the OpenAI continuous control benchmark suite.
3. We demonstrate that ACC is a general algorithm by additionally applying it to TD3.

To allow for reproducibility of our results we describe our algorithm in detail, report all hyperparameters, use a large number of random seeds for evaluation, and open sourced the code[1].

2 Background

We consider model-free reinforcement learning for episodic tasks with continuous state and action spaces $\mathcal{S}$ and $\mathcal{A}$. An agent interacts with its environment by selecting an action $a_t \in \mathcal{A}$ in state $s_t \in \mathcal{S}$ for every discrete time step t. The agent receives a scalar reward r_t and observes the new state s_{t+1}. To model this in a mathematical framework we use a Markov decision process, defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$. Given an action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$ the unknown state transition density $\mathcal{P}$ defines a distribution over the next state. Rewards are given to the agent according to the reward function $\mathcal{R}$ and future rewards are discounted via the discount factor $\gamma \in [0, 1]$.

The goal is to learn a policy π that maps a state s to a distribution over actions such that the sum of future discounted rewards $R_t = \sum_{i=t}^{T} \gamma^{i-t} r_i$ is maximized. We use the term π_ϕ for the policy with parameters ϕ, that is trained to maximize the expected return $J(\phi) = \mathbb{E}_{s_i \sim \mathcal{P}, a_i \sim \pi}[R_0]$. For a given state-action pair (s, a) the value function is defined as $Q^\pi(s, a) = \mathbb{E}_{s_i \sim \mathcal{P}, a_i \sim \pi}[R_t | s, a]$, which is the expected return when executing action a in state s and following π afterwards.

2.1 Soft Actor Critic

TQC extends Soft Actor-Critic (SAC) [14], which is a strong off-policy algorithm for continuous control that uses entropy regularization. This means that while in the end we are interested in maximizing the performance with respect to the total amount of reward collected in the environment, SAC maximizes for an auxiliary objective that augments the original reward with the entropy of the policy $J(\phi) = \mathbb{E}_{s_t \sim \mathcal{P}, a_t \sim \pi}[\sum_t \gamma^t (r_t + \alpha \mathcal{H}(\pi(\cdot | s_t)))]$, where $\mathcal{H}$ denotes the entropy.

A critic is learned that evaluates the policy π in terms of its Q-value of the entropy augmented reward. The policy—called actor—is trained to choose actions such that the Q-function is maximized with an additional entropy regularization

$$J_\pi(\phi) = \mathop{\mathbb{E}}_{s_t \sim \mathcal{D}, a_t \sim \pi_\phi} [Q_\theta(s_t, a_t) - \alpha \log \pi_\phi(a_t | s_t)]. \tag{1}$$

The weighting parameter α of the entropy term can be automatically adjusted during the training [15]. Both the training of actor and critic happen off-policy with transitions sampled from a replay buffer.

[1] `https://github.com/Nicolinho/ACC`

2.2 Truncated Quantile Critics

The TQC algorithm uses distributional reinforcement learning [2] to learn a distribution over the future augmented reward instead of a Q-function which is a point estimate for the expectation of this quantity. In TQC this is done with quantile regression [6] which approximates the distribution with Dirac delta functions $Z_\theta(s_t, a_t) = \frac{1}{M}\sum_{m=1}^{M} \delta(\theta^m(s_t, a_t))$. The Diracs are located at the quantile locations for fractions $\tau_m = \frac{2m-1}{m}, m \in \{1, \ldots, M\}$. The network is trained to learn the quantile locations $\theta^m(s, a)$ by regressing the predictions $\theta^m(s_t, a_t)$ onto the Bellman targets $y_m(s_t, a_t) = r_t + \gamma(\theta^m(s_{t+1}, a_{t+1}) - \alpha \log \pi_\phi(a_{t+1}|s_{t+1}))$ via the Huber quantile loss.

TQC uses an ensemble of N networks $(\theta_1, \cdots, \theta_N)$ where each network θ_n predicts the distribution $Z_{\theta_n}(s_t, a_t) = \frac{1}{M}\sum_{m=1}^{M} \delta(\theta_n^m(s_t, a_t))$. A single Bellman target distribution is computed for all networks. This happens by first computing all targets for all networks, pooling all targets together in one set and sorting them in ascending order. Let $k \in \{1, \ldots, M\}$, then the kN smallest of these targets y_i are used to define the target distribution $Y(s_t, a_t) = \frac{1}{kN}\sum_{i=1}^{kN} \delta(y_i(s_t, a_t))$. The networks are trained by minimizing the quantile Huber loss which in this case is given by

$$L(s_t, a_t; \theta_n) = \frac{1}{kNM} \sum_{m,i=1}^{M,kN} \rho_{\tau_m}^H(y_i(s_t, a_t) - \theta_n^m(s_t, a_t)), \tag{2}$$

where $\rho_\tau^H(u) = |\tau - \mathbb{1}(u < 0)|\mathcal{L}_H^1(u)$ and $\mathcal{L}_H^1(u)$ is the Huber loss with parameter 1.

The rationale behind truncating some quantiles from the target distribution is to prevent overestimation bias. In TQC the number of dropped targets per network $d = M - k$ is a hyperparameter that has to be tuned per environment but allows for a finegrained control of the bias.

The policy is trained as in SAC by maximizing the entropy penalized estimate of the Q-value which is the expectation over the distribution obtained from the critic

$$J(\phi) = \mathop{\mathbb{E}}_{\substack{s\sim\mathcal{D}\\ a\sim\pi}}\left[\alpha \log \pi_\phi(a|s) - \frac{1}{NM}\sum_{m,n=1}^{M,N} \theta_n^m(s, a)\right]. \tag{3}$$

3 Adaptively Calibrated Critics

In this section, we will first introduce the problem of estimation bias in TD learning. Then we will present our method ACC followed by an explanation how it can be applied to TQC.

3.1 Over- and Underestimation Bias

The problem of overestimation bias in temporal difference learning with function approximation has been known for a long time [33]. In Q-learning [37] the predicted Q-value $Q(s_t, a_t)$ is regressed onto the target given by $y = r_t + \gamma \max_a Q(s_{t+1}, a)$. In the tabular case and under mild assumptions the Q-values converge to that of the optimal policy [37] with this update rule. However, using a function approximator to generate the Q-value introduces an approximation error. Even under the assumption of zero mean noise corruption of the Q-value $\mathbb{E}[\epsilon_a] = 0$, an overestimation bias occurs in the computation of the target value because of Jensen's inequality

$$\max_a Q(s_{t+1}, a) = \max_a \mathbb{E}[Q(s_{t+1}, a) + \epsilon_a] \leq \mathbb{E}\left[\max_a\{Q(s_{t+1}, a) + \epsilon_a\}\right]. \tag{4}$$

In continuous action spaces it is not possible to take the maximum over all actions. The most successful algorithms rely on an actor-critic structure where the actor is trained to choose actions that maximize the Q-value [11, 14, 24]. So the actor can be interpreted an approximation to the argmax of the Q-value.

With deep neural networks as function approximators other problems such as over-generalization [8, 27] can occur where the updates to $Q(s_t, a_t)$ also increases the target through $Q(s_{t+1}, a)$ for all a which could lead to divergence. There are many other potential sources for overestimation bias such as stochasticity of the environment [16] or computing the Q-target from actions that lie outside of the current training data distribution [21].

While for discrete action spaces the overestimation can be controlled with the double estimator [16, 35], it was shown that this estimator does not prevent overestimation when the action space is continuous [11]. As a solution the TD3 algorithm [11] uses the minimum of two separate estimators to compute the critic target. This approach was shown to prevent overestimation but can introduce an underestimation bias. In TQC [22] the problem is handled by dropping some targets from the pooled set of all targets of an ensemble of distributional critics. This allows for more finegrained control of over- or underestimation by choosing how many targets are dropped. TQC is able to achieve an impressive performance but the parameter d determining the number of dropped targets has to be set for each environment individually. This is highly undesirable for many applications since the hyperparameter sweep to determine a good choice of the parameter increases the actual number of environment interactions proportional to the number of hyperparameters tested. For many applications like robotics this makes the training prohibitively expensive.

3.2 Dynamically Adjusting the Bias

In the following we present a new general approach to adaptively control bias emerging in TD targets regardless of the source of the bias. Let $R^\pi(s,a)$ be the random variable denoting the sum of future discounted rewards when the agent starts in state s, executes action a and follows policy π afterwards. This means that the Q-value is defined as its expectation $Q^\pi(s,a) = \mathbb{E}[R^\pi(s,a)]$. For notational convenience we will drop the dependency on the policy π in the following. We start with the tabular case. Suppose for each state-action pair (s,a) we have a family $\{\hat{Q}_\beta(s,a)\}_{\beta\in[\beta_{min},\beta_{max}]\subset\mathbb{R}}$ of estimators for $Q(s,a)$ with the property that $\hat{Q}_{\beta_{min}}(s,a) \leq Q(s,a) \leq \hat{Q}_{\beta_{max}}(s,a)$, where $Q(s,a)$ is the true Q-value of the policy π and Q_β a continuous monotone increasing function in β .

If we have samples from $R_i \sim R(s,a)$ an unbiased estimator for $Q(s,a)$ is given by the average of the R_i. This is also called Monte Carlo estimation [32]. We further define the estimator $\hat{Q}_{\beta^*}(s,a)$, where β^* is given by

$$\beta^*(s,a) = \underset{\beta\in[\beta_{min},\beta_{max}]}{\arg\min} \left| \hat{Q}_\beta(s,a) - \frac{1}{N}\sum_{i=1}^{N} R_i(s,a) \right|. \tag{5}$$

In the following Theorem we show that the estimator is unbiased under some assumptions.

Theorem 1 *Let $Q_\beta(s,a)$ be a continuous monotone increasing function in β and assume that for all (s,a) it holds $\hat{Q}_{\beta_{min}}(s,a) \leq Q(s,a) \leq \hat{Q}_{\beta_{max}}(s,a)$, the returns $R(s,a)$ follow a symmetric probability distribution and that $\hat{Q}_{\beta_{min}}(s,a)$ and $\hat{Q}_{\beta_{max}}(s,a)$ have the same distance to $Q(s,a)$. Then Q_{β^*} from Equation 5 is an unbiased estimator for the true value Q for all (s,a).*

The proof is provided in the appendix. The symmetry and same distance assumption can be replaced by assuming that $\hat{Q}_{\beta_{min}}(s,a) \leq R_i \leq \hat{Q}_{\beta_{max}}(s,a)$ with probability one. In this case the proof is straightforward since Q_β can take any value for which R_i has positive mass.

We are interested in the case where $\hat{Q}$ is given by a function approximator such that there is generalization between state-action pairs and that it is possible to generate estimates for pairs for which there are no samples of the return available. Consider off-policy TD learning where the samples for updates of the Q-function are sampled from a replay buffer of past experience. While the above assumptions might not hold anymore in this case, we have an estimator for all state-action pairs and not just the ones for which we have samples of the return. Also in practice rolling out the policy several times from each state action pair is undesirable and so we set $N = 1$ which allows the use of the actual exploration rollouts. Our proposed algorithm starts by initializing the bias-controlling parameter β to some value. After a number of environment steps and when the next episode is finished, the Q-value estimates and actual observed returns are compared. Depending on the difference β is adjusted according to

$$\beta_{new} = \beta_{old} + \alpha \sum_{t=1}^{T_\beta} \Big[R(s_t,a_t) - \hat{Q}(s_t,a_t) \Big], \tag{6}$$

where α is a step size parameter and $(s_t,a_t)_{t=1}^{T_\beta}$ are the $T_\beta \in \mathbb{N}$ most recent state-action pairs. As a result β is decreased in the case of overestimation, where the Q-estimates are larger than the actual observed returns, and increased in the case of underestimation. We assumed that Q_β is continuous and monotonically increasing in β. Hence, increasing β increases Q_β and vice versa. For updating the Q-function the target will be computed from Q_β.

Only performing one update step and not the complete minimization from Equation 5 has the advantage that β is changing relatively slow which means the targets are more stable. Through this mechanism our method can incorporate the high variance on-policy samples to correct for under- or overestimation bias. At the same time our method can benefit from the low variance TD targets. ACC in its general form is summarized in Algorithm 1 in the appendix.

Other algorithms that attempt to control the bias arising in TD learning with non-linear function approximators usually use some kind of heuristic that includes more than one estimator. Some approaches use them to decouple the choice of the maximizing action and the evaluation of the maximum in the computation of the TD targets [35]. Alternative approaches take the minimum, maximum or a combination of both over the different estimators [1, 11, 12, 23]. All of these have in common that the same level of bias correction is done for every environment and for all time steps during training. In the deep case there are many different sources that can influence the tendency of TD learning building up bias in non-trivial ways. ACC is more principled in the regard that it allows to dynamically adjust the magnitude and direction of bias correction during training. Regardless of the source and amount of bias ACC provides a way to alleviate it. This makes ACC promising to work robustly on a wide range of different environments.

One assumption of ACC is that there is a way to adjust the estimated Q-value with a parameter β such that $\hat{Q}_\beta$ is continuous and monotonically increasing in β. There are many different functions that are in accordance with this assumption. We give one general example of how such a $\hat{Q}_\beta$ can be easily constructed for any algorithm that learns a Q-value. Let $\hat{Q}$ be the current estimate. Then define $\hat{Q}_\beta = \beta|\hat{Q}|/K + \hat{Q}$, where K is a constant (e.g. 100) and $[\beta_{min}, \beta_{max}]$ is some interval around 0. In the following section we will present an application of ACC in a more sophisticated way.

3.3 Applying ACC to TQC

As a practical instantiation of the general ACC algorithm we apply it to adjust the number of targets dropped from the set of all targets in TQC. Denote with $d_{max} \in \{0, \dots, M\}$ some upper limit of targets to drop per network. Define $\beta_{min} = 0$, $\beta_{max} = d_{max}$ and let $d = d_{max} - \beta$ be the current number of targets dropped for each network. Further, we write Q_β for the TQC estimate with dN targets dropped from the pooled set of all targets. If d_{max} is set high enough the TQC estimate without dropped targets $Q_{\beta_{max}}$ induces overestimation while the TQC estimate with d_{max} dropped targets per net $Q_{\beta_{min}}$ induces underestimation.

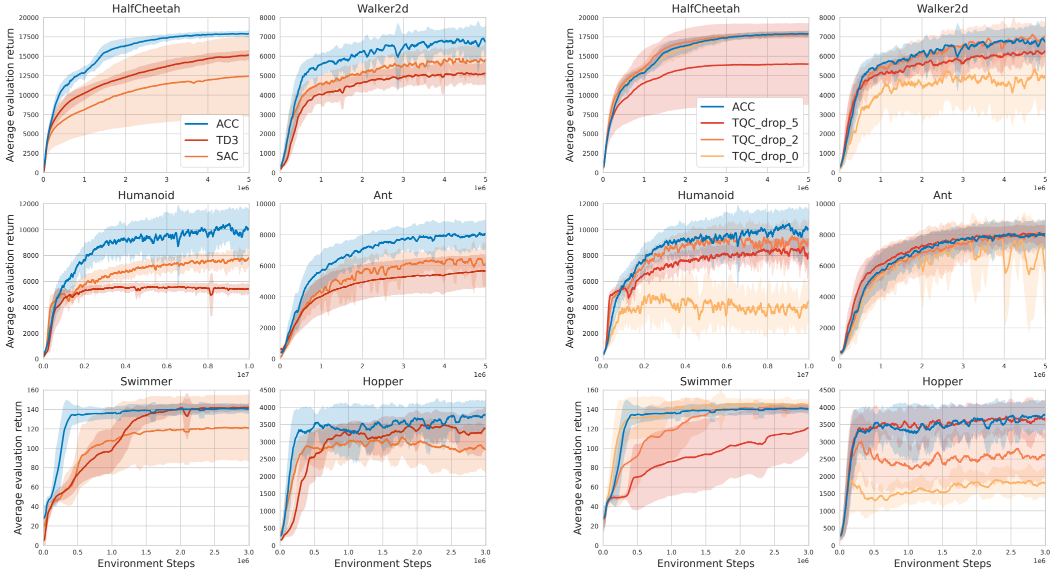
In general, $\beta \in [0, d_{max}]$ is continuous and hence also d is a continuous value. However, the number of dropped targets from the pooled set of all targets has to be a discrete number in $\{0, \dots, NM\}$. Thus, in the computation of the TD target the total number of dropped targets dN is rounded to the nearest integer. We use a normalization when updating β. A moving average of the absolute value of the difference between returns and estimated Q-values is stored. When updating β with Equation 6, the expectation is divided by the moving average.

4 Experiments

We evaluate our algorithm on a range of continuous control tasks from OpenAI Gym [4] that makes use of the physics engine MuJoCo [34] (version 1.5). First, we benchmark ACC against strong methods that do not use environment specific hyerparameters. Then we compare the performance of TQC with a fixed number of dropped targets per network with that of ACC. Next, we evaluate the effect of more critic updates for ACC and show results in the sample efficient regime. Further, we study the effect of ACC on the accuracy of the value estimate, and investigate the generality of ACC by applying it to TD3. To give more insight into the learning dynamics we also analyze how the calibration parameter develops during training in the appendix.

We implemented ACC on top of the PyTorch code published by the authors[2] to ensure a fair comparison. Because in TQC the tuned optimal number of dropped targets per network is for every

[2] https://github.com/bayesgroup/tqc_pytorch



(a)

(b)

Figure 1: Learning curves for six different continuous control tasks from OpenAi gym. For all environments version *v3* was used. The shaded area represents mean ± standard deviation over the 10 trials. For readability the curves showing the mean are filtered with a uniform filter of size 15. In **(a)** ACC applied to TQC is compared to other state of the arts methods for continuous control that use the same hyperparameters for all environments. **(b)** shows the results for TQC with the number of dropped quantiles per network fixed to different choices and the performance if this hyperparameter is adjusted online with ACC.

environment set in the interval $[0, 5]$, we set $d_{max} = 5$. At the beginning of the training we initialize $\beta = 2.5$ and set the step size parameter to $\alpha = 0.1$. We spend only a very limited amount of computation time into into the tuning of the previously mentioned hyperparameters and describe in the appendix in detail the process of choosing the selected hyperparameters. We also present pseudocode for our method as well as the complete list of all hyperparameters in the appendix.

Compared to TQC the additional computational overhead caused by ACC is minimal because there is only one update to β that is very cheap compared to one training step of the actor-critic and there are at least $T_\beta = 1000$ training steps in between one update to β.

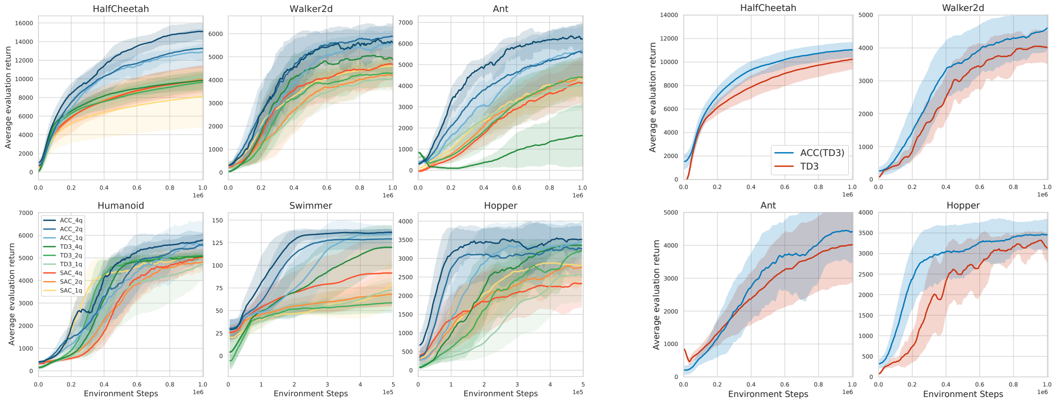
During training, the policy is evaluated every 1,000 environment steps by taking the average over the episode reward obtained by rolling out the current policy without sampling noise 10 times. For each task and algorithm we average the results of 10 trials each with a different random seed.

4.1 Comparative Evaluation

We compare ACC to the state of the art continuous control methods SAC [14] (with learned temperature parameter [15]) and TD3 [11]. The learning curves are shown in Figure 1a. On all six environments ACC achieves considerably better results setting a new state of the art among all algorithms without environment specific hyperparameters.

4.2 Fixing the Number of Dropped Targets

In this experiment we evaluate how well ACC performs when compared to TQC where we the number of dropped targets per network d is fixed to some value. Since in the original publication for each environment the optimal value was one of the three values 0, 2, and 5, we evaluated TQC with d fixed to one of these values for each environment. To ensure comparability we used the same codebase as for ACC. The results are shown in Figure 1b. It can be seen that ACC matches the performances of TQC with the best hyperparameter in every environment. Furthermore, there is no single choice



(a)

(b)

Figure 2: The mean ± standard deviation over 10 trials. **(a)** results in the sample efficient regime with different choices for the update number of the value function per environment step for each algorithm. **(b)** Results for ACC applied to TD3 compared to pure TD3.

that can compete with ACC on all environments. With $d = 0$, TQC is substantially worse on three environments and unstable on the *Ant* environment. Setting $d = 2$ is overall the best choice but still performs clearly worse for two environments and is also slightly worse for *Humanoid*. Dropping $d = 5$ targets per network leads to an algorithm that can compete with ACC only on two of the six environments. The experiments showed that it is not possible to find one value for d that performs well on all environments. Furthermore, even if there would be one tuned parameter that performs equally well as ACC on a given set of environments we hypothesize there are likely very different environments for which the specific parameter choice will not perform well. The principled nature of ACC on the other hand provides reason to believe that it can perform robustly on a wide range of different environments. This is supported by the robust performance on all considered environments.

4.3 Evaluation of Sample Efficient Variant

In principle more critic updates per environment step should make learning faster. However, because of the bootstrapping in the target computation this can easily become unstable. The problem is that as targets are changing faster, bias can build up easier and divergence becomes more likely. ACC provides a way to detect upbuilding bias in the TD targets and to correct the bias accordingly. This motivates to increase the number of gradient updates of the critic. In TD3, SAC and TQC one critic update is performed per environment step. We conducted an experiment to study the effect of increasing this rate up to 4. ACC using 4, 2 and 1 updates are denoted with ACC_4q, ACC_2q and ACC_1q respectively. ACC_1q is equal to ACC from the previous experiments. We use the same notation also for TD3 and SAC.

Scaling the number of critic updates by a factor of 4 increases the computation time by a factor of 4. But this can be worthwhile in the sample efficient regime, where a huge number of environment interactions is not possible or the interaction cost dominate the computational costs. This for example the case if the aim is to train robots in the real world. The results of our experiment are shown in Figure 2a. It can be seen that in the sample efficient regime ACC4q further increases over plain ACC. ACC4q reaches the performance that TD3 and SAC achieve at the end of the training in less than a third of the amount of steps for five environments. On *Humanoid* it needs roughly half the number of steps for that. Increasing the number of critic updates for TD3 and SAC shows mixed results, increasing performance on some environment while decreasing performance on others. Only ACC benefits from more critic updates on all environments, which supports the hypothesis that ACC is successful at calibrating the critic estimate.

4.4 Analysis of the Value Estimate

To better understand the effect of ACC on the bias of the value estimate, we analyze the difference between the value estimate and the corresponding observed return when ACC is applied to TQC. For

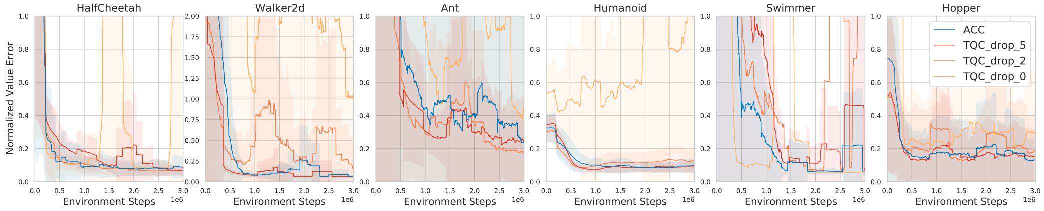


Figure 3: The normalized absolute error of the value estimate. Shown are the mean ± the standard deviation over 5 trials with a uniform filter of size 401 for readability.

each state-action pair encountered during exploration, we compute its value estimate at that time and at the end of the episode compare it with the actual discounted return from that state onwards. Hence, the state-action pair was not used to update the value function at the point when the value estimate has been computed. If an episode ends because the maximum number of episode time-steps has been reached, which is 1,000 for the considered environments, we ignore the last 100 state-action pairs. The reason is that in TQC the value estimator is trained to ignore the episode timeout and uses a bootstrapped target also at the end of the episode. We normalize for different value scales by computing the absolute error between the value estimate and the observed discounted return and divide that by the absolute value of the discounted return. Every 1,000 steps, the average over the errors of the last 1,000 state-action pairs is computed. The results are plotted in Figure 3 and show that ACC indeed achieves a low value error compared to TQC with fixed hyperparameter. This supports our hypothesis that the strong performance of ACC applied to TQC indeed stems from better values estimates.

4.5 Beyond TQC: Improving TD3 with ACC

To demonstrate the generality of ACC, we additionally applied it to TD3 algorithm [11], which uses a actor-critic framework with two critics. These are initialized differently but are trained with the same target value, which is the minimum over the two targets computed from the two critics. This is done to prevent overestimation in the value estimates. While successfully preventing the overestimation, using the minimum of the two target estimates is very coarse and can instead lead to an underestimation bias. We applied ACC to TD3 by defining the target for each critic network to be a convex combination between its own target and the target estimate coming from the minimum of both:

$$y_k = r + \gamma \left(\beta \, Q_{\bar{\theta}_k}(s_{t+1}, \pi_{\bar{\phi}}(s_{t+1})) + (1-\beta) \min_{i=1,2} Q_{\bar{\theta}_i}(s_{t+1}, \pi_{\bar{\phi}}(s_{t+1})) \right), \tag{7}$$

where $\beta \in [0, 1]$ is the ACC parameter that is adjusted to balance between under- and overestimation. The results are displayed in Figure 2b and show that ACC also improves the performance of TD3.

5 Related Work

5.1 Overestimation in Reinforcement Learning

The problem of overestimation in Q-learning with function approximation was introduced by [33]. For discrete actions the double estimator has been proposed [16] where two Q-functions are learned and one is used to determine the maximizing action, while the other evaluates the Q-function for that action. The Double DQN algorithm extended this to neural networks [35]. However, Zhang et al. [40] observed that the double estimator sometimes underestimates the Q-value and propose to use a weighted average of the single and the double estimator as target. This work is similar to ours in the regard that depending on the parameter over- or underestimation could be corrected. A major difference to our algorithm is that the weighting parameter is computed from the maximum and minimum of the estimated Q-value and does not use unbiased rollouts. Lv et al. [25] propose a similar weighting but suggest a stochastic selection of either the single or double estimator. The probability of choosing one or the other follows a predefined schedule. Other approaches compute the weighted average of the minimum and maximum over different estimates for the Q-value [12, 21]. However, the weighting parameter is a fixed hyperparameter. The TD3 algorithm [11] proposed to use the minimum over two Q-value estimates as the target. Maxmin Q-learning is another approach for discrete action spaces that uses an ensemble of Q-functions. In the target computation, first the minimum of over all Q-functions is computed followed by maximization with respect to the action [23]. Decreasing the ensemble size increases the estimated targets while increasing the size decreases the targets. Similarly to TQC this provides a way to control the bias in a more fine-grained way; the respective hyperparameter has to be set before the start of the training for each environment, however.

Other approaches have identified overgeneralization as a potential source of wrong or divergend TD targets. The idea is that, in the case of function approximation, updates to $Q(s_t, a_t)$ also change the estimate of $Q(s_{t+1}, a_{t+1})$ which again changes $Q(s_t, a_t)$ through the TD target [8]. In a recent work [31], it is proposed to use a regularization term to account for that. Furthermore, this work uses a combination of minimum and maximum over different Q-value estimates to compute the TD target. However, the method has several environment specific hyperparameters.

What sets ACC apart from the previously mentioned works is that unbiased on-policy rollouts are used to adjust a term that controls the bias correction instead of using some predefined heuristic.

5.2 Combining On- and Off-Policy Learning

There are many approaches that combine on- and off-policy learning by combining policy gradients with off-policy samples [7, 9, 20, 29, 36]. In Gu et al. [13] an actor-critic is used where the critic is updated off-policy and the actor is updated with a mixture of policy gradient and Q-gradient. This differs from our work in that we are interested only in better critic estimates through the information of on-policy samples. To learn better value estimates by combining on- and off-policy data many works propose the use of some form of importance sampling [26, 28, 30]. In Hausknecht and Stone [17] the TD target is computed by mixing Monte Carlo samples with the bootstrap estimator. These methods provide a tradeoff between variance and bias. They differ from our work in using the actual returns directly in the TD targets while we incorporate the returns indirectly via another parameter. Bhatt et al. [3] propose the use of a mixture of on- and off-policy transitions to generate a feature normalization that can be used in off-policy TD learning. Applied to TD3, learning becomes more stable eliminating the need to use a delayed target network.

5.3 Hyperparameter Tuning for Reinforcement Learning

Most algorithms that tune hyperparameters of RL algorithms use many different instances of the environment to find a good setting [5, 10, 19, 39]. There is, however, also work that adjusts a hyperparameter online during training [38]. In this work the meta-gradient (i.e., the gradient of the update rule) is used to adjust the discount factor and the length of bootstrapping intervals. However, it would not be straightforward to apply this method to control the bias of the value estimate. Their method also differs from ours in that they do not use a combination of on- and off-policy data.

6 Conclusion

We present Adaptively Calibrated Critics (ACC), a general off-policy algorithm that learns a Q-value function with bias calibrated TD targets. The bias correction in the targets is determined via a parameter that is adjusted by comparing the current value estimates with the most recently observed on-policy returns. Our method allows to incorporate information from the unbiased sample returns into the TD targets while keeping the high variance of the samples out. We apply ACC to Truncated Quantile Critics, a recent off-policy continuous control algorithm that allows fine-grained control of the TD target scale through a hyperparameter tuned per environment. With ACC, this parameter can automatically be adjusted during training, obviating the need for extensive tuning. The strong experimental results suggest that our method provides an efficient and general way to control the bias occurring in TD learning.

Interesting directions for future work are to evaluate the effectiveness of ACC applied to algorithms that work with discrete action spaces and when learning on a real robot where tuning of hyperparameters is very costly.

Acknowledgements

The authors would like to thank the following people for their help throughout the process of writing this paper, in alphabetical order: Daniel Honerkamp, Johannes Meyer, Niklas Wetzel and Tim Welschehold. This work was supported by the European Union's Horizon 2020 research and innovation program under grant agreement No 871449-OpenDR and by the Freiburg Graduate School of Robotics.

References

[1] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, pages 104–114. PMLR, 2020.

[2] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pages 449–458, 2017.

[3] Aditya Bhatt, Max Argus, Artemij Amiranashvili, and Thomas Brox. Crossnorm: Normalization for off-policy td reinforcement learning. *arXiv preprint arXiv:1902.05605*, 2019.

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[5] Hao-Tien Lewis Chiang, Aleksandra Faust, Marek Fiser, and Anthony Francis. Learning navigation behaviors end-to-end with autorl. *IEEE Robotics and Automation Letters*, 4(2): 2007–2014, 2019. doi: 10.1109/LRA.2019.2899918.

[6] Will Dabney, Mark Rowland, Marc Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[7] Thomas Degris, Martha White, and Richard Sutton. Off-policy actor-critic. In *International Conference on Machine Learning*, 2012.

[8] Ishan Durugkar and Peter Stone. Td learning with constrained gradients. In *Proceedings of the Deep Reinforcement Learning Symposium, NIPS 2017*, Long Beach, CA, USA, December 2017. URL `http://www.cs.utexas.edu/users/ai-lab?NIPS17-ishand`.

[9] Rasool Fakoor, Pratik Chaudhari, and Alexander J. Smola. P3o: Policy-on policy-off policy optimization. In *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, volume 115 of *Proceedings of Machine Learning Research*, pages 1017–1027. PMLR, 2020.

[10] Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1437–1446, 2018.

[11] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1582–1591, 2018.

[12] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pages 2052–2062. PMLR, 2019.

[13] Shixiang (Shane) Gu, Timothy Lillicrap, Richard E Turner, Zoubin Ghahramani, Bernhard Schölkopf, and Sergey Levine. Interpolated policy gradient: Merging on-policy and off-policy gradient estimation for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 30, pages 3846–3855, 2017.

[14] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1861–1870, 2018.

[15] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications. *CoRR*, abs/1812.05905, 2018. URL `http://arxiv.org/abs/1812.05905`.

[16] Hado V Hasselt. Double q-learning. In *Advances in Neural Information Processing Systems*, pages 2613–2621, 2010.

[17] Matthew Hausknecht and Peter Stone. On-policy vs. off-policy updates for deep reinforcement learning. In *Deep Reinforcement Learning: Frontiers and Challenges, IJCAI 2016 Workshop*, 2016.

[18] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[19] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

[20] Tang Jie and Pieter Abbeel. On a connection between importance sampling and the likelihood ratio policy gradient. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 23, pages 1000–1008, 2010.

[21] Aviral Kumar, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. In *Advances in Neural Information Processing Systems*, volume 32, pages 11784–11794, 2019.

[22] Arsenii Kuznetsov, Pavel Shvechikov, Alexander Grishin, and Dmitry Vetrov. Controlling overestimation bias with truncated mixture of continuous distributional quantile critics. In *International Conference on Machine Learning*, pages 5556–5566. PMLR, 2020.

[23] Qingfeng Lan, Yangchen Pan, Alona Fyshe, and Martha White. Maxmin q-learning: Controlling the estimation bias of q-learning. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=Bkg0u3Etwr`.

[24] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[25] P. Lv, X. Wang, Y. Cheng, and Z. Duan. Stochastic double deep q-network. *IEEE Access*, 7: 79446–79454, 2019. doi: 10.1109/ACCESS.2019.2922706.

[26] A. Rupam Mahmood, Hado P van Hasselt, and Richard S Sutton. Weighted importance sampling for off-policy learning with linear function approximation. In *Advances in Neural Information Processing Systems*, volume 27, pages 3014–3022, 2014.

[27] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[28] Remi Munos, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 29, pages 1054–1062, 2016.

[29] Brendan O'Donoghue, Remi Munos, Koray Kavukcuoglu, and Volodymyr Mnih. Combining policy gradient and q-learning. In *ICLR*, 2016.

[30] Doina Precup. Eligibility traces for off-policy policy evaluation. *Computer Science Department Faculty Publication Series*, page 80, 2000.

[31] Lin Shao, Yifan You, Mengyuan Yan, Qingyun Sun, and Jeannette Bohg. Grac: Self-guided and self-regularized actor-critic. *arXiv preprint arXiv:2009.08973*, 2020.

[32] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT Press, 2018. ISBN 78-0262039246.

[33] Sebastian Thrun and Anton Schwartz. Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 Connectionist Models Summer School*, pages 255–263, 1993.

[34] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IROS*, pages 5026–5033. IEEE, 2012.

[35] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[36] Ziyu Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. D. Freitas. Sample efficient actor-critic with experience replay. In *ICLR*, 2017.

[37] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[38] Zhongwen Xu, Hado P van Hasselt, and David Silver. Meta-gradient reinforcement learning. In *NeurIPS*, 2018.

[39] Baohe Zhang, Raghu Rajan, Luis Pineda, Nathan Lambert, André Biedenkapp, Kurtland Chua, Frank Hutter, and Roberto Calandra. On the importance of hyperparameter optimization for model-based reinforcement learning. In *International Conference on Artificial Intelligence and Statistics*, pages 4015–4023. PMLR, 2021.

[40] Zongzhang Zhang, Zhiyuan Pan, and Mykel J. Kochenderfer. Weighted double q-learning. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, IJCAI'17, pages 3455–3461. AAAI Press, 2017. ISBN 978-0-9992411-0-3. URL `http://dl.acm.org/citation.cfm?id=3172077.3172372`.

A Proof of Theorem 1

The estimator $\hat{Q}^{\pi}_{\beta^*}(s, a)$ was defined via

$$\beta^*(s, a) = \underset{\beta \in [\beta_{min}, \beta_{min}]}{\arg\min} \left| \hat{Q}_{\beta}(s, a) - \frac{1}{N} \sum_{i=1}^{N} R_i(s, a) \right|. \tag{8}$$

To declutter the notation we drop the dependencies on the state-action pairs (s, a) and the policy π. Further we write $\bar{R} = \frac{1}{N} \sum_{i=1}^{N} R_i$. First note that the average of symmetrically distributed random variables is still a symmetric distributed random variable and hence $\bar{R}$ is symmetrically distributed. By assumption $\hat{Q}_{\beta_{min}}$ and $\hat{Q}_{\beta_{max}}$ have the same distance to the true Q-value which is the mean $Q = \mathbb{E}[\bar{R}]$, i.e. there is a distance real valued value d such that $Q = \hat{Q}_{\beta_{min}} + d = \hat{Q}_{\beta_{max}} - d$ Denote the tail probabilty P$(\bar{R} < \hat{Q}_{\beta_{min}}) = p_t$. Because of the symmetry and the same distance to the mean we also have that $P(\bar{R} > \hat{Q}_{\beta_{max}}) = p_t$. In the computation of $\mathbb{E}[\hat{Q}_{\beta^*}]$ we can differentiate three events. If $\hat{Q}_{\beta_{min}} \leq \bar{R} \leq \hat{Q}_{\beta_{max}}$ then $\hat{Q}_{\beta^*} = \bar{R}$, if $\hat{Q}_{\beta_{min}} \geq \bar{R}$ then $\hat{Q}_{\beta^*} = \hat{Q}_{\beta_{min}}$ and if $\hat{Q}_{\beta_{min}} \geq \bar{R}$ then $\hat{Q}_{\beta^*} = \hat{Q}_{\beta_{max}}$. We denote the indicator function with $\mathbb{A}$, which is equal to 1 if the event A is true and 0 otherwise. Then we get

$$\begin{aligned}
\mathbb{E}\left[\hat{Q}_{\beta^*}\right] &= \mathbb{E}\left[\mathbb{1}\left[\hat{Q}_{\beta_{min}} \leq \bar{R} \leq \hat{Q}_{\beta_{max}}\right] \bar{R}\right] \\
&\quad + \mathbb{E}\left[\mathbb{1}\left[\hat{Q}_{\beta_{min}} \geq \bar{R}\right] \hat{Q}_{\beta_{min}}\right] \\
&\quad + \mathbb{E}\left[\mathbb{1}\left[\hat{Q}_{\beta_{max}} \leq \bar{R}\right] \hat{Q}_{\beta_{max}}\Bigg|\right] \\
&= (1 - 2p_t) \cdot \mathbb{E}[\bar{R}] + p_t\, \mathbb{E}\left[\hat{Q}_{\beta_{min}}\right] + p_t\, \mathbb{E}\left[\hat{Q}_{\beta_{max}}\right] \\
&= (1 - 2p_t)Q + p_t\hat{Q}_{\beta_{min}} + p_t\hat{Q}_{\beta_{max}} \\
&= (1 - 2p_t)Q + p_t(Q - d) + p_t(Q + d) \\
&= (1 - 2p_t)Q + 2p_tQ + p_t(d - d) \\
&= Q
\end{aligned}$$

B Using Fewer Critic Networks for Faster Runtime

Using 5 critic networks - the default in TQC - to approximate the value function leads to a high runtime of the algorithm. It is possible to trade off performance against runtime by changing the

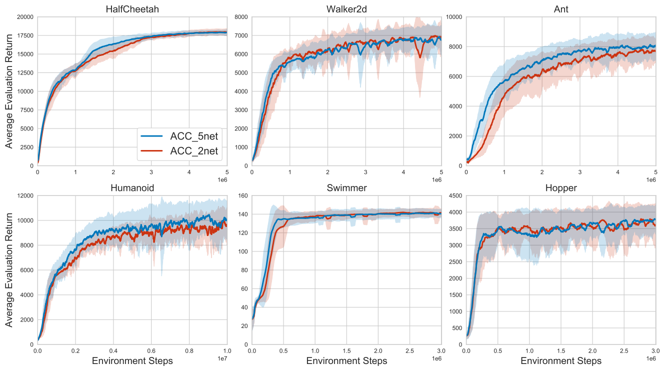


Figure 4: The mean ± standard deviation over 10 trials. Results with different choices for the number of critic networks for each algorithm.

Algorithm 1 ACC - General

```
Initialize: bias controlling parameter $\beta$, steps between $\beta$ updates $T_\beta$, $t_\beta = 0$
for $t = 1$ to total number of environment steps do
    Interact with environment according to $\pi$, store transitions in replay buffer $\mathcal{B}$ and store observed returns $R(s, a)$, increment $t_\beta \mathrel{+}= 1$
    if episode ended and $t_\beta >= T_\beta$ then
        Update $\beta$ with Eq. 6 using the most recent experience and set $t_\beta = 0$
    end if
    Sample mini-batch $b$ from $\mathcal{B}$
    Update $Q$ with target computed from $Q_\beta$ and $b$
end for
```

Algorithm 2 ACC - Applied to TQC

```
Initialize: $d$ the bias controlling parameter, $\alpha$ the learning rate for $d$, $T_d$ the minimum number of steps between updates to $d$, $T_d^{init}$ the initial steps before $d$ is updated, $S_R$ the size from which on episodes are removed from the batch storing the most recent returns, moving average parameter $\tau_d$, $t_d = 0$
for $t = 1$ to total number of environment steps do
    Interact with environment according to $\pi$, store transitions in replay buffer $\mathcal{B}$ and, increment $t_d \mathrel{+}= 1$
    if episode ended then
        Store observed returns $R(s, a)$ and corresponding state-action pairs $(s, a)$ in $\mathcal{B}_R$
        if $t_d >= T_d$ and $t > T_d^{init}$ then
            $C = \sum_{(s,a,R)\in\mathcal{B}_R} \Big[Q(s, a) - R(s, a)\Big]$, $ma = (1 - \tau_d)ma + \tau_d C$
            $d = d + \alpha\frac{C}{ma}$, clip $d$ in interval $[0, d_{max}]$, set $t_d = 0$
            Remove the oldest episodes from $\mathcal{B}_R$ until there are at most $S_R$ left
        end if
    end if
    Sample mini-batch from $\mathcal{B}$
    Update critic $Q$ as in TQC, where $dN$ (rounded to the next integer) number of targets are dropped from the set of pooled targets
    Update policy $\pi$ as in TQC
end for
```

number of critic networks. We evaluated ACC applied to TQC with 2 networks and compare it to the standard setting with 5 networks in Figure 4. The results show that reducing the number of critic networks to 2 leads only to a small drop in performance while the runtime is more than 2 times faster.

C Pseudocode

In Algorithm 1 the general version of ACC is presented. The pseudocode for ACC applied to TQC is in Algorithm 2. As the number of dropped targets per network is given by $d = d_{\max} - \beta$, we state the pseudocode in terms of the parameter d instead of β.

D Hyperparameters

At the beginning of the training we initialize $\beta = 2.5$ and set the step size parameter to $\alpha = 0.1$. After $T_\beta = 1000$ steps since the last update and when the next episode finishes, β is updated with a batch that stores the most recent state-action pairs encountered in the environment and their corresponding observed discounted returns. The choice of T_β was motivated by the fact that the maximum duration of an episode is 1000 steps for the considered environments. After every update of β the oldest episodes in this stored batch are removed until there are no more than 5000 state-action pairs left. This means that on average β is updated with a batch whose size is a bit over 5000. The updates of β are started as soon as 25000 environment steps as completed and the moving average parameter in the normalization of the $\beta-$update is set to 0.05. The first 5000 environment interactions are generated with a random policy after which learning starts. Apart from that all hyperparameters are the same as in TQC with $N = 5$ critic networks. In Table 1 we list all hyperparameters of ACC applied to TQC.

In the following we also desribe the process of hyperparameter selection. The range of values d is allowed to take is set to the interval $[0, 5]$ as it includes the optimal hyperparameters for TQC from all environments, which are in the set $\{0, 2, 5\}$. We did not try higher values than 5. The initial value for number of dropped targets per network was set to 2.5 as this value is in the middle of the allowed range and did not evaluated other choices. The learning rate α of d was set to 0.1 based on visual inspection of how fast d changes. We evaluated $\alpha = 0.05$ for a small subset of tasks and seeds, but $\alpha = 0.1$ gave slightly better results. T_d was set to 1000 as the episode length is 1000 and we did not evaluate other choices. For T_d^{init} we evaluated the choices 10000 and 25000 on a small subset of environments and seeds and did not found a big impact on performance. As d changes very quickly in the beginning we chose $T_d^{init} = 25000$. For S_R we evaluated the choices 1000 and 5000 also on a small subset of environments and seeds and found 5000 to perform slightly better. We did not tune the moving average parameter and set it to $\tau_d = 0.05$. For all hyperparameters for which we evaluated more than one choice we do not have definite results as the number of seeds and environments were limited. The hyperparameters shared with TQC were not changed.

Table 1: Hyperparameters values.

Hyperparameter	ACC
Optimizer	Adam
Learning rate	3×10^{-4}
Discount γ	0.99
Replay buffer size	1×10^6
Number of critics N	5
Number of atoms M	25
Huber loss parameter	1
Number of hidden layers in critic networks	3
Size of hidden layers in critic networks	512
Number of hidden layers in policy network	2
Size of hidden layers in policy network	256
Minibatch size	256
Entropy target	$-\dim \mathcal{A}$
Nonlinearity	ReLU
Target smoothing coefficient	0.005
Target updates per critic gradient step	1
Critic gradient steps per iteration	1
Actor gradient steps per iteration	1
Environment steps per iteration	1
Initial value for number of dropped targets per network	2.5
Maximum value for d denoted $d_{\max}$	5
Minimum value for d denoted $d_{\min}$	0
Learning rate for d denoted α	0.1
Minimum number of steps between updates to d denoted T_d	1000
Initial number of steps before d is updated denoted T_d^{init}	25000
Limiting size for batch used to update d denoted S_R	5000
Moving average parameter τ_d	0.05

Hyperparameter in Sample Efficient Experiment	ACC_1q	ACC_2q	ACC_4q
Critic gradient steps per iteration	1	2	4
Actor gradient steps per iteration	1	1	1
Target updates per critic gradient step	1	1	1

E Potential Limitations

One limitation of our work is that ACC can not be applied in the offline RL setting, as ACC also uses on-policy data. Furthermore, in the stated form ACC relies on the episodic RL setting. However, we believe that ACC could potentially be adapted to that setting. It is also not entirely clear how

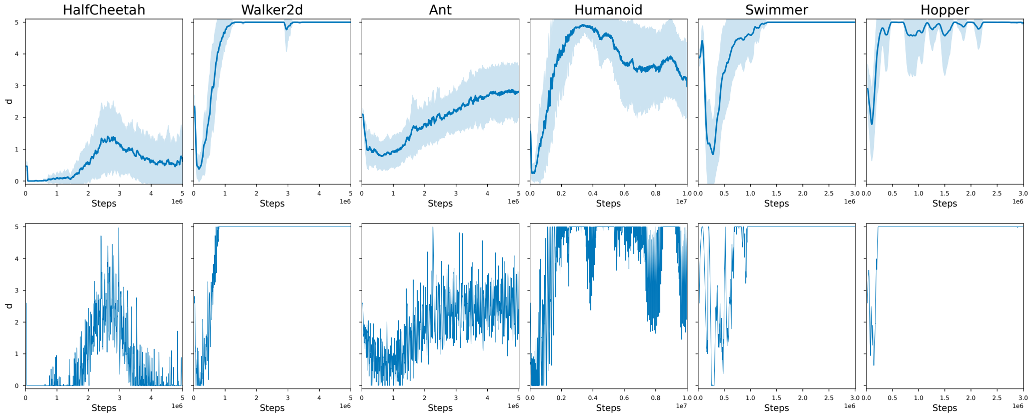


Figure 5: Development of the number of dropped targets per network $d = d_{max} - \beta$ in ACC over time for different environments. The top row shows the mean (thick line) and standard deviation (shaded area) over the 10 trials where for readability a uniform filter of size 15 is used. The bottom row shows the unfiltered development for one of the seeds.

the algorithm would perform in the terminal reward setting, where a reward of for example 1 is given upon successful completion of a specific task. While we do not have experiments for such environments we imagine that the positive effect of ACC could diminish as the true Q-values of states closer to the start of the episode are almost zero because of the discounting.

F Analysis of the ACC Parameter

To better understand the hidden training dynamics of ACC we show in Figure 5 how the number of dropped targets per network $d = d_{max} - \beta$ evolves during training. To do so we plotted d after every 5000 steps during the training of ACC. From the top row the first observation is that per environment the results are similar over the 10 seeds as can be seen from the relatively low standard deviation. We show the single runs for all seeds in the appendix to further support this observation. However, there are large differences between the environments which supports the argument that it might not be possible to find a single hyperparameter that works well on a wide variety of different environments. Another point that becomes clear from the plots is that the optimal amount of overestimation correction might change over time during the training even on a single environment.

In the bottom row of Figure 5 we plotted the evolution of d for one of the 10 trials in order to shed light on the actual training mechanics of a single run without lost information due to averaging. For each environment there is a trend but d is also fluctuating to a certain degree. While this shows that the initial value of d is not very important as the value quickly changes, this also highlights another interesting aspect of ACC. The rollouts give highly fluctuating returns. The parameter $d = d_{max} - \beta$ is changing more slowly and picks up the trend. So a lot of the variance of the returns is filtered out in ACC by incorporating on-policy samples via the detour over β. This leads to relatively stable TD targets computed from Q_β while an upbuilding under- or overestimation is prevented as β picks up the trend. On the other hand, if β would change too slowly the upbuilding of the bias might not be stopped.