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

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

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

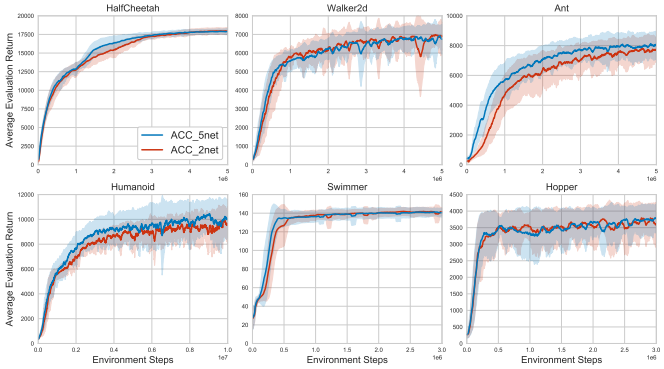

Figure 4: The mean ± standard deviation over 10 trials. Results with different choices for the number of critic networks for each algorithm.

Algorithm 1 ACC - General

Initialize: bias controlling parameter β, steps between β updates T_β, $t_\beta = 0$
for $t = 1$ **to** total number of environment steps **do**
  Interact with environment according to π, 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 β 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_β and b
end for

Algorithm 2 ACC - Applied to TQC

Initialize: d the bias controlling parameter, α 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 τ_d, $t_d = 0$
for $t = 1$ **to** total number of environment steps **do**
  Interact with environment according to π, 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 π 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

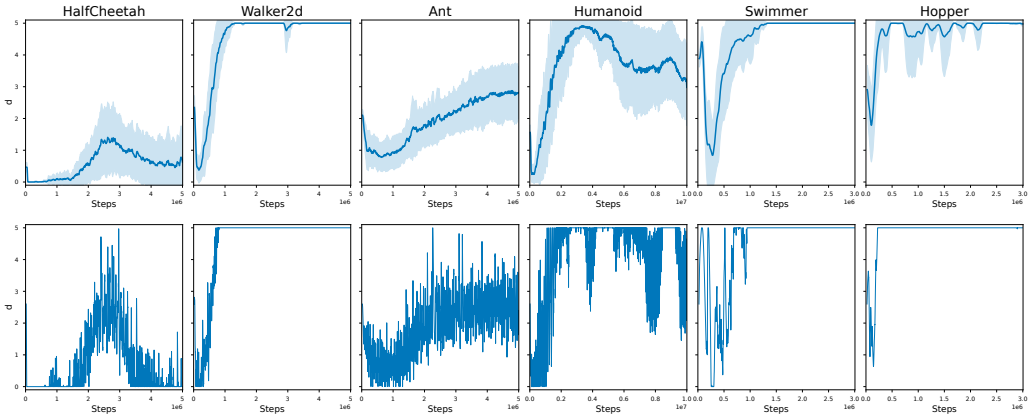

Figure 5: Development of the number of dropped targets per network $d = d_{max} - \beta$ in ACC over time for different environments. The top row shows the mean (thick line) and standard deviation (shaded area) over the 10 trials where for readability a uniform filter of size 15 is used. The bottom row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