Aggressive Q-Learning with Ensembles: Achieving Both High Sample Efficiency and High Asymptotic Performance

Yanqiu Wu[1] **Xinyue Chen**[2,3]

Che Wang[1] **Yiming Zhang**[1] **Keith Ross**[1,2*]

[1] New York University
[2]New York University Shanghai
[3] University of California, Berkeley

Abstract

Recent advances in model-free deep reinforcement learning (DRL) show that simple model-free methods can be highly effective in challenging high-dimensional continuous control tasks. In particular, Truncated Quantile Critics (TQC) achieves state-of-the-art asymptotic training performance on the MuJoCo benchmark with a distributional representation of critics; and Randomized Ensemble Double Q-Learning (REDQ) achieves high sample efficiency that is competitive with state-of-the-art model-based methods using a high update-to-data ratio and target randomization. In this paper, we propose a novel model-free algorithm, Aggressive Q-Learning with Ensembles (AQE), which improves the sample-efficiency performance of REDQ and the asymptotic performance of TQC, thereby providing overall state-of-the-art performance during all stages of training. Moreover, AQE is very simple, requiring neither distributional representation of critics nor target randomization. The effectiveness of AQE is further supported by our extensive experiments, ablations, and theoretical results.

1 Introduction

Off-policy Deep Reinforcement Learning algorithms aim to improve sample efficiency by reusing past experience. A number of off-policy Deep RL algorithms have been proposed for control tasks with continuous state and action spaces, including Deep Deterministic Policy Gradient (DDPG), Twin Delayed DDPG (TD3) and Soft Actor Critic (SAC) (Lillicrap et al., 2016; Fujimoto et al., 2018; Haarnoja et al., 2018a,b). TD3 introduced clipped double-Q learning, and was shown to be significantly more sample efficient than popular on-policy methods for a wide range of MuJoCo benchmarks. Soft Actor Critic (SAC) has similar off-policy structures with clipped double-Q learning, but it also employs maximum entropy reinforcement learning. SAC was shown to provide excellent sample efficiency and asymptotic performance in a wide-range of MuJoCo environments, including the high-dimensional Humanoid environment for which both DDPG and TD3 perform poorly.

More recently, Kuznetsov et al. (2020) proposed Truncated Quantile Critics (TQC), a model-free algorithm which includes distributional representations of critics, truncation of critics prediction, and ensembling of multiple critics. Instead of the usual modeling of the Q-function of the expectation of

*Correspondence to: Keith Ross <keithwross@nyu.edu>.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

return, TQC approximates the distribution of the return random variable conditioned on the state and action. By dropping several of the top-most "atoms" and varying the number of dropped atoms of the return distribution approximation, TQC can control the over-estimation bias. TQC's asymptotic performance (that is after a long period of training) was shown to be better than that of SAC on the continuous control MuJoCo benchmark suite, including a 25% improvement on the most challenging Humanoid environment. However, TQC is not sample efficient in that it generally requires a large number of samples to reach even moderate performance levels.

Chen et al. (2021) proposed Randomized Ensembled Double Q-learning(REDQ), a model-free algorithm which includes a high Update-To-Data (UTD) ratio, an ensemble of Q functions, and in-target minimization across a random subset of Q functions from the ensemble. Using a UTD ratio much larger than one, meaning that several gradient steps are taken for each environment interaction, improves sample efficiency, while the ensemble and in-target minimization allows the algorithm to maintain stable and near-uniform bias under the high UTD ratio. The algorithm was shown to attain much better performance than SAC at the early stage of training, and to match or improve the sample-efficiency of the state-of-the-art model-based algorithms for the MuJoCo benchmarks. However, although REDQ is highly sample efficient for early-stage training, its asymptotic performance is significantly below that of TQC.

Is it possible to design a simple, streamlined model-free algorithm which can achieve REDQ's high sample efficiency in early-stage training and also achieve TQC's high asymptotic performance in late stage training? In this paper, we achieve this goal with a new model-free algorithm, Aggressive Q-Learning with Ensembles (AQE). Like TQC and REDQ, AQE uses an ensemble of Q-functions, and like REDQ it uses a UTD ratio > 1. However AQE is very simple, requiring neither distributional representation of critics as in TQC nor target randomization and double-Q learning as in REDQ. AQE controls overestimation bias and the standard deviation of the bias by varying the number of ensemble members N and the number of ensembles $K \leq N$ that are kept when calculating the targets.

Through carefully designed experiments, we provide a detailed analysis of AQE. We perform extensive and comprehensive experiments for both MuJoCo and DeepMind Control Suite (DMC) environments. We first show that for the five most challenging MuJoCo benchmark, AQE provides state-of-the-art performance, surpassing the performance of SAC, REDQ, and TQC at *all stages of training.* When averaged across the five MuJoCo environments, AQE's early stage performance is 2.9 times better than SAC, 1.6 times better than TQC and 1.1 times better than REDQ. AQE's asymptotic performance is 26%, 22%, and 6% higher than SAC, REDQ, and TQC, respectively. Then we provide additional experimental results for the nine most challenging DeepMind Control Suite (DMC) environments, which TQC and REDQ did not consider. We show that AQE also provides state-of-the-art performance at both early stage and late-stage of training. When averaged over nine environments, AQE's early stage performance is 13.71 times better than SAC, 7.59 times better than TQC and 1.02 times better than REDQ. AQE's asymptotic performance is 37% better than SAC, 3% better than REDQ, and 8% better than TQC. We also perform an ablation study, and show that AQE is robust to choices of hyperparameters: AQE can work well with small ensembles consisting of 10-20 ensemble members, and performance does not vary significantly with small changes in the keep parameter K. We show that that AQE performs better than several variations, including using the median of all ensemble members and removing the most extreme minimum and maximum outlier in the targets. In order to improve computational time, we also consider different multi-head architectures for the ensemble of critics: consistent with the supervised convolutional network literature, we find that a two-head architecture not only reduces computational time but can actually improve performance for some environments. Additionally, we show that AQE continues to out-perform SAC and TQC even when these algorithms are made aggressive with a UTD $\gg 1$.

To ensure a fair comparison and to obtain reliable and reproducible results (Henderson et al., 2018; Islam et al., 2017; Duan et al., 2016), we provide open source code[2]. For all algorithmic comparisons, we use the the authors' code.

2 Additional Related Work

Overestimation bias due to in target maximization in Q-learning can significantly slow down learning (Thrun and Schwartz, 1993). For tabular Q-learning, van Hasselt (2010) introduced Double Q-

[2]https://github.com/AutumnWu/Aggressive-Q-Learning-with-Ensembles

Learning, and showed that it removes the overestimation basis and in general leads to an under-estimation bias. Hasselt et al. (2016) showed that adding Double Q-learning to deep-Q networks can have a similar effect, leading to a major performance boost for the Atari games benchmark. As stated in the Introduction, for continuous-action spaces, TD3 and SAC address the overestimation bias using clipped-double Q-learning, which brings significant performance improvements (Fujimoto et al., 2018; Haarnoja et al., 2018a,b).

As mentioned in the Introduction, Kuznetsov et al. (2020) control the over-estimation bias by estimating the distribution of the return random variable, and then by dropping several of the top-most "atoms" from the estimated distribution. The distribution estimate is based on a methodology developed in Bellemare et al. (2017); Dabney et al. (2018a,b), which employs an asymmetric Huber loss function to minimize the Wasserstein distance between the neural network output distribution and the target distribution. In this paper, in order to counter over-estimation bias, we also drop the top-most estimates, although we do so solely with an ensemble of Q-function mean estimators rather than with an ensemble of the more complex distributional models employed in (Kuznetsov et al., 2020).

It is well-known that using ensembles can improve the performance of DRL algorithms (Faußer and Schwenker, 2015; Osband et al., 2016; Lee et al., 2021). For Q-learning based methods, Anschel et al. (2017) use the average of multiple Q estimates to reduce variance. Lan et al. (2020) introduced Maxmin Q-learning, which uses the minimum of all the Q-estimates rather than the average. Agarwal et al. (2020) use Random Ensemble Mixture (REM), which employs a random convex combination of multiple Q estimates.

Model-based methods often attain high sample efficiency by using a high UTD ratio. In particular, Model-Based Policy Optimization (MBPO) (Janner et al., 2019) uses a large UTD ratio of 20-40. Compared to Soft-Actor-Critic (SAC), which is model-free and uses a UTD of 1, MBPO achieves much higher sample efficiency in the OpenAI MuJoCo benchmark (Todorov et al., 2012; Brockman et al., 2016). REDQ (Chen et al., 2021), a model-free algorithm, also successfully employs a high UTD ratio to achieve high sample efficiency.

3 Algorithm

We propose Aggressive Q-learning with Ensembles (AQE), a simple model-free algorithm which provides state-of-the-art performance for the MuJoCo benchmark for both early and late stage of training. The pseudocode can be found in Algorithm 1. As is the case with most off-policy continuous-control algorithms, AQE has a single actor (policy network) and multiple critics (Q-function networks), and employs Polyak averaging of the target parameters to enhance stability. Building on this algorithmic base, it also employs an update-to-data ratio $G > 1$, an ensemble of $N \geq 3$ Q-functions (rather than just two as in TD3 and SAC), and targets that average all the Q-functions excluding the Q-functions with the highest $N - K$ values. In the Appendix, we demonstrate theoretically in the tabular case of the algorithm that we can control over-estimation through adjusting K and N. More concretely, we can bring the bias term from above zero (i.e. overestimation) to under zero (i.e. underestimation) by decreasing K and/or increasing N.

For comparison, REDQ employs two randomly chosen ensemble members when calculating the target, the bias does not depend on the number of ensemble models N (Chen et al., 2021). As discussed in Appendix, with $M = 2$ fixed, increasing the size of ensemble N with the multi-head architecture does not necessarily improve the performance of REDQ. Unlike REDQ, AQE can control the bias term through both the number of ensemble models used in the average calculation K and the total number of ensembles N, allowing for more flexibility. One other drawback for REDQ is that it ignores the estimates of all other ensemble estimates except for the minimal one in the randomly chosen set, which diminishes the power of the multiple ensemble sets. In contrast, AQE utilizes most of the ensemble models when calculating the target.The resulting algorithm is not only simple and streamlined, but also provides state-of-the art performance. For exploration, it uses entropy maximization as in SAC, although it could easily incorporate alternative exploration schemes.

AQE has three key hyperparameters, G, N, and K. If we set $N = 2$, $K = 1$ and $G = 1$, AQE is simply the underlying off-policy algorithm such as SAC. When $N > 2$, $K = 1$ and $G = 1$, then AQE becomes similar to, but not equivalent to, Maxmin Q-learning (Lan et al., 2020).

Algorithm 1 Aggressive Q-Learning with Ensembles

Initialize: Initial policy parameters θ, N Q-function parameters $\phi_i, i = 1,\ldots,N$, empty replay buffer $\mathcal{D}$. Set target parameters $\phi_{\text{targ},i} \leftarrow \phi_i$ for $i = 1, 2,\ldots, N$.

repeat
  Take one action $a_t \sim \pi_\theta(\cdot|s_t)$. Observe reward r_t, new state s_{t+1}.
  Add data to replay buffer: $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r_t, s_{t+1})\}$
  for G updates **do**
    Randomly sample a mini-batch $B = \{(s, a, r, s')\}$ from $\mathcal{D}$.
    for each $(s, a, r, s') \in B$ **do**
      Sample $\tilde{a}' \sim \pi_\theta(\cdot|s')$.
      Determine the K indices from $i = 1, \ldots, N$ that minimize $Q_{\text{target},i}(s', \tilde{a}')$.
      Compute the Q target y:

$$y(s,a) = r + \gamma\left(\frac{1}{K}\sum_{i\in K} Q_{\phi_{\text{targ},i}}(s',\tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s')\right)$$

    for $i = 1, \ldots, N$ **do**
      Update ϕ_i with gradient descent using

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s')\in B} \left(Q_{\phi_i}(s,a) - y(s,a)\right)^2$$

      Update target networks with $\phi_{\text{targ},i} \leftarrow \rho\phi_{\text{targ},i} + (1-\rho)\phi_i$
  Update policy parameters θ with gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s\in B} \left(\frac{1}{N}\sum_{i=1}^{N} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta(\tilde{a}_\theta(s)|s)\right) \qquad \tilde{a}_\theta(s) \sim \pi_\theta(\cdot|s)$$

until Convergence

AQE uses an ensemble of Q networks (as does REDQ and TQC). Employing multiple networks, one for each Q-function output, can be expensive in terms of computation and memory. In order to reduce the computation and memory requirements, we combine network ensemble with multi-head architectures to generate multiple Q-function outputs. We consider N separate Q networks each with h heads, providing a total of $h \cdot N$ estimates. The h heads from one network share all of the layers except the final fully-connected layer. In practice, we found h = 2 heads works well for AQE, consistent with work in ensembles of convolutional neural networks for computer vision tasks (Lee et al., 2015). When properly sharing low-level weights, multi-headed networks may not only retain the performance of full ensembles but can sometimes outperform them. In the next section, we discuss our experimental results.

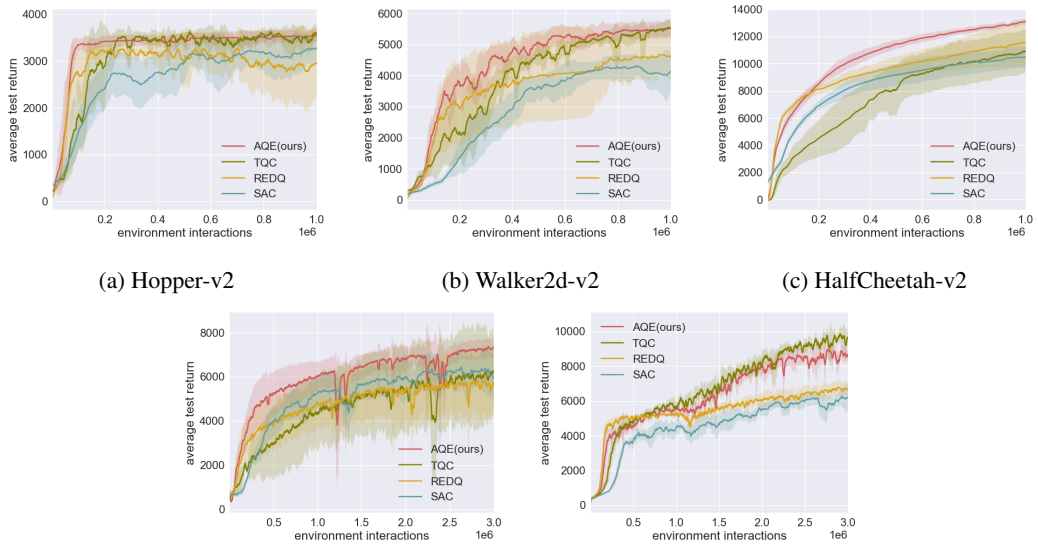
4 Experimental Results

We perform extensive and comprehensive experiments for two sets of popular benchmarks. First we provide experimental results for AQE, TQC, REDQ and SAC for the five most challenging MuJoCo environments, namely, Hopper, Walker2d, HalfCheetah, Ant and Humanoid. Then we provide additional experimental results for the nine most challenging DeepMind Control Suite (DMC) environments, namely, Cheetah-run, Fish-swim, Hopper-hop, Humanoid-stand, Humanoid-walk, Humanoid-run, Quadruped-walk, Quadruped-run and Walker-run. To make a fair comparison, the TQC, REDQ and SAC results are reproduced using the authors' open source code, and use the same network sizes and hyperparameters reported in their papers. In particular, for the MuJoCo environments, TQC employs 5 critic networks with 25 distributional samples for a total of 125 atoms. TQC drops 5 atoms per critic for Hopper, 0 atoms per critic for Half Cheetah, and 2 atoms per critic for Walker, Ant, and Humanoid. For REDQ, we also use the authors' suggested values of N = 10 and M = 2, where M is the number of ensemble members used in the target calculation.

The REDQ paper uses $G = 20$ for the update-to-data ratio, and provides results for up to 300K environment interactions. Using such a high value for G is computationally infeasible in our experimental setting, since we use 3 million environment interactions for Ant and Humanoid and over 4 million environment interactions for Humanoid-run in order to investigate asymptotic performance

as well early-stage sample efficiency. In the experiments reported here, we use a value of $G = 5$ for both REDQ and AQE.

For AQE, we use 10 Q-networks each with 2 heads, producing 20 Q-values for each input. The AQE networks are the same size as those in the REDQ paper. For MuJoCo benchmark, AQE keeps 10 out of 20 values for Hopper, all 20 values for half-Cheetah, and 16 out of 20 values for Walker, Ant and Humanoid.



(a) Hopper-v2 (b) Walker2d-v2 (c) HalfCheetah-v2

(d) Ant-v2 (e) Humanoid-v2

Figure 1: AQE versus TQC, REDQ and SAC for MuJoCo environments. AQE is the only algorithm that beats SAC in all five environments during all stages of training, and it typically beats SAC by a wide margin.

Figure 1 shows the training curves for AQE, TQC, REDQ, and SAC for the five MuJoCo environments. For each algorithm, we plot the average return of 5 independent trials as the solid curve, and plot the standard deviation across 5 seeds as the shaded region. For each environment, we train each algorithm for exactly the same number of environment interactions as done in the SAC paper. We use the same evaluation protocol as in the TQC paper. Specifically, after every epoch, we run ten test episodes with the current policy, record the undiscounted sum of all the rewards in each episode and take the average of the sums as the performance. A more detailed discussion on hyperparameters and implementation details is given in the Appendix.

We see from Figure 1 that AQE is the only algorithm that beats SAC in all five environments during all stages of training. Moreover, it typically beats SAC by a very wide margin. Table 1 shows that, when averaged across the five environments, AQE achieves SAC asymptotic performance approximately 3x faster than SAC and 2x faster than REDQ and TQC. As seen from Figure 1 and Table 2, in the early stages of training, AQE matches the excellent performance of REDQ in all five environments, and both algorithms are much more sample efficient than SAC and TQC. As seen from Figure 1 and Table 3, in late-stage training, AQE always matches or beats all other algorithms, except for Humanoid, where TQC is about 10% better. Table 3 shows that, when averaged across all five environments, AQE's asymptotic performance is 26%, 22%, and 6% higher than SAC, REDQ, and TQC, respectively.

4.1 Fixed Hyperparameters across Environments

Following the TQC paper, in Figure 1 we used different drop atoms for TQC for the different MuJoCo environments. To make the comparison fair, we also used different keep values K for AQE for the different environments. We repeat the experiment on the five MuJoCo environments, but now use the same hyperparameter values across environments for TQC (drop two atoms per network) and AQE ($K = 16$). These choices of fixed hyperparameters appear to give the best overall performance for the

Table 1: Sample efficiency comparison of SAC, TQC, REDQ and AQE. The numbers show the amount of data collected when the specified performance level is reached (roughly corresponding to 90% of SAC's final performance). The last three columns show how many times AQE is more sample efficient than SAC, TQC and REDQ in reaching that performance level. For each task, the lowest sample use, and those within 5% difference are highlighted.

Performance	SAC	TQC	REDQ	AQE	AQE/SAC	AQE/TQC	AQE/REDQ
Hopper at 3000	506K	184K	136K	**77K**	6.57	2.39	1.77
Walker2d at 4000	631K	371K	501K	**277K**	2.28	1.34	1.81
HalfCheetah at 10000	763K	737K	552K	**304K**	2.51	2.42	1.82
Ant at 5500	1445K	1759K	1749K	**632K**	2.29	2.78	2.77
Humanoid at 6000	2469K	**1043K**	1862K	1345K	1.84	0.78	1.38
Average	-	-	-	-	3.10	1.94	1.91

Table 2: Early-stage performance comparison of SAC, TQC, REDQ and AQE. The numbers show the performance achieved when the specific amount of data is collected. On average, AQE performs 2.9 times better than SAC, 1.6 times better than TQC and 1.1 times better than REDQ. For each task, the highest score and those within 5% difference are highlighted.

Amount of data	SAC	TQC	REDQ	AQE	AQE/SAC	AQE/TQC	AQE/REDQ
Hopper at 100K	1456	1807	2747	**3345**	2.30	1.85	1.22
Walker2d at 100K	501	1215	1810	**2150**	4.29	1.77	1.19
HalfCheetah at 100K	3055	4801	**6876**	6378	2.09	1.33	0.93
Ant at 250K	2107	2344	3279	**4153**	1.97	1.77	1.27
Humanoid at 250K	1094	3038	**4535**	3973	3.63	1.31	0.88
Average at early stage	-	-	-	-	2.86	1.61	1.10

two algorithms. The training curves for AQE, TQC, REDQ, and SAC and detailed early-stage and late-stage performance comparisons of all algorithms for this experiment are shown in the Appendix.

We can see from the results that with fixed hyperparamters, the conclusions for AQE remain largely unchanged, except for Hopper, where REDQ becomes the strongest algorithm. When averaged across environments, AQE still matches the high sample efficiency of REDQ during the early stages of training. Furthermore, on average, AQE's asymptotic performance is still 16%, 11% and 9% higher than SAC, REDQ and TQC, respectively.

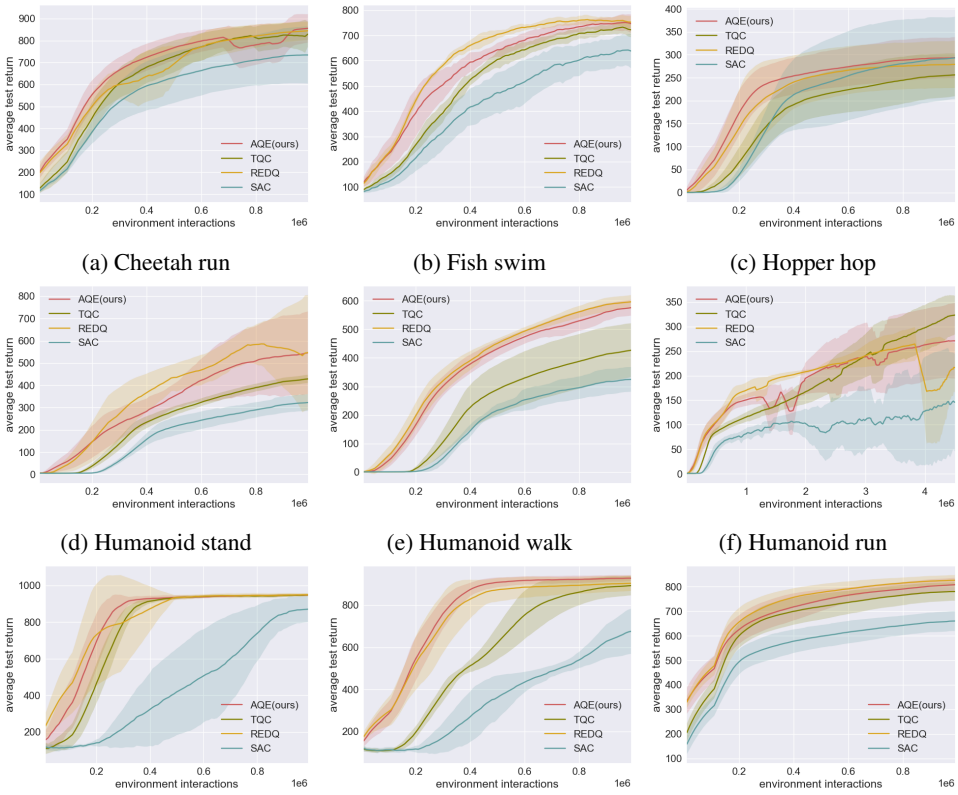
4.2 DeepMind Control Suite Results

In this section, we provide detailed experimental results for AQE, TQC, REDQ and SAC for the nine most challenging DeepMind Control Suite (DMC) environments. The TQC and REDQ papers do not consider DMC benchmark, so we employ the same hyperparameters for TQC and REDQ as for the MuJoCo environments. For TQC, we employ 5 critic networks with 25 distributional samples and drop 2 atoms per critic across environments. For REDQ, we keep using $N = 10$ and $M = 2$. To make the comparison fair, we also use the same hyperparameter values across environments for AQE. We present the learning curves in Figure 2. Similar to MuJoCo benchmark, for each algorithm, we plot the average return of 5 independent trials as the solid curve, and plot the standard deviation across 5 seeds as the shaded region. We run all algorithms to 1 million environment interactions except for the most challenging environment, Humanoid-run, where we run up to 4.5 million environment interactions. We use the same evaluation protocol as for the MuJoCo environments.

Figure 2 shows that in DMC environments with fixed hyperparameters, AQE continues to outperform TQC except for the Humanoid-run environment, where TQC performs better than AQE in the final stage of training. AQE and REDQ have comparable results in some of the DMC environments during traning, however, AQE usually outperforms REDQ in the more challenging environments, such as Hopper-hop, Humanoid-run, and Quadruped-run. We report detailed early-stage and late-stage performance comparisons of all algorithms in Appendix.

Table 3: Late-stage performance comparison of SAC, TQC, REDQ and AQE. The numbers show the performance achieved when a specific amount of data is collected. The last three columns show the ratio of AQE performance compared to SAC, TQC, and REDQ performance. On average, during late-stage training, AQE performs 1.26 times better than SAC, 1.06 times better than TQC, and 1.22 times better than REDQ. For each task, the highest score and those within 5% difference are highlighted.

Amount of data	SAC	TQC	REDQ	AQE	AQE/SAC	AQE/TQC	AQE/REDQ
Hopper at 1M	3282	**3612**	2954	**3541**	1.08	0.98	1.20
Walker2d at 1M	4134	**5532**	4637	**5517**	1.33	1.00	1.19
HalfCheetah at 1M	10475	10887	11562	**13093**	1.25	1.20	1.13
Ant at 3M	5903	6186	5785	**7345**	1.24	1.19	1.27
Humanoid at 3M	6177	**9593**	6649	8680	1.41	0.91	1.31
Average at late stage	-	-	-	-	1.26	1.06	1.22



(a) Cheetah run (b) Fish swim (c) Hopper hop

(d) Humanoid stand (e) Humanoid walk (f) Humanoid run

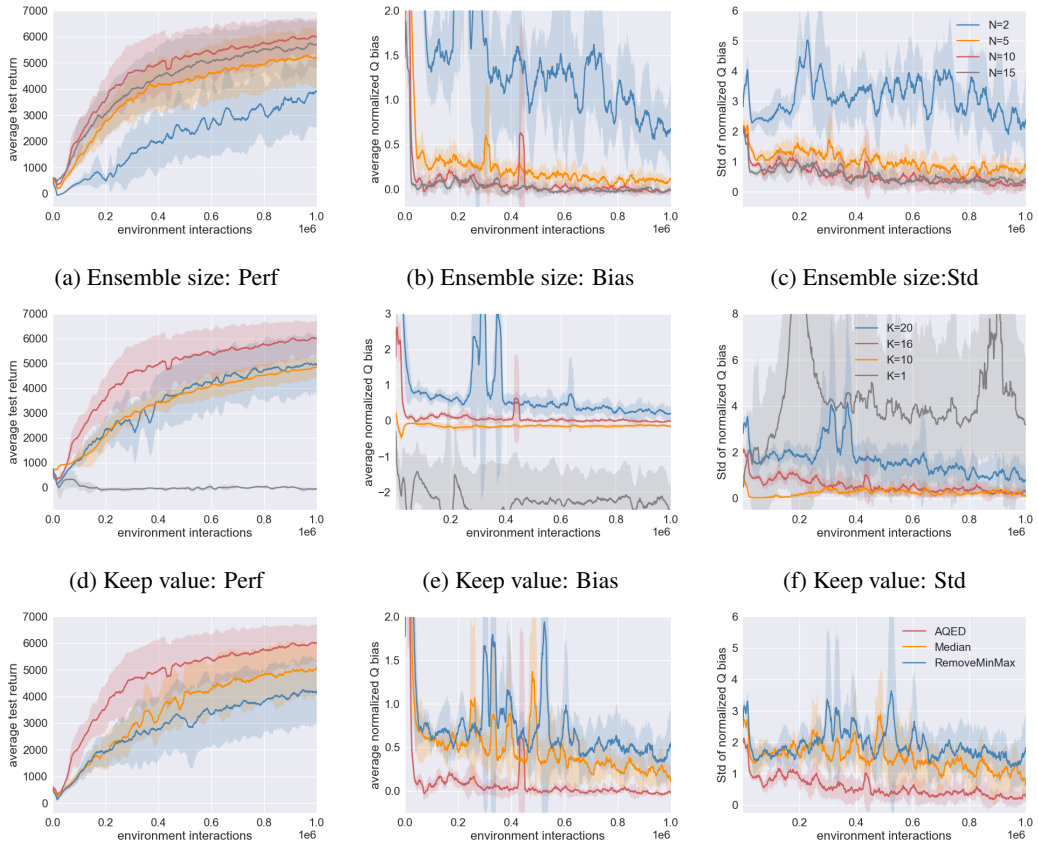
(g) Quadruped walk (h) Quadruped run (i) Walker run

Figure 2: AQE versus TQC, REDQ and SAC in DeepMind Control Suite benchmark. AQE and TQC use same hyperparameters across the nine environments.

In summary, in the early stage of training (100K data), AQE performs 13.71x better than SAC, 7.59x better than TQC and matches the excellent performance of REDQ in nine environments. In the late-stage training (1M data), AQE always matches or outperforms all other algorithms, except for Humanoid-run, where TQC performs the best. On average, AQE performs 37% better than SAC, 8% better than TQC, and 3% better than REDQ. Using the same hyperparameters and averaged across nine DMC environments, AQE achieves the asymptotic performance of SAC approximately 3x faster than SAC, 1.57x faster than TQC, and 1.05x faster than REDQ.

4.3 Ablations

In this section, we use ablations to provide further insight into AQE. We focus on the Ant environment, and run the experiments up to 1M time steps. (In the Appendix we provide ablations for the other four environments.) As in the REDQ paper, we consider not only performance but normalized bias and standard deviation of normalized bias as defined by the REDQ authors. We first look at how the ensemble size N affects AQE. The first row in Figure 3 shows AQE with N equal to 2, 5, 10 and 15, with two heads for each Q network, and the percentage of kept Q-values unchanged. As the ensemble size N increases, we generally obtain a more stable average bias, a lower std of bias, and stronger performance. When trained with high UTD value, a relatively small ensemble size, for example, $N = 5$, can greatly reduce bias accumulation, resulting in much stronger performance. This experimental finding is consistent with the results in Theorem 1 in Appendix G.



(a) Ensemble size: Perf (b) Ensemble size: Bias (c) Ensemble size:Std

(d) Keep value: Perf (e) Keep value: Bias (f) Keep value: Std

(g) Variations: Perf (h) Variations: Bias (i) Variations: Std

Figure 3: AQE ablation results for Ant. The top row shows the effect of the ensemble size N. The second row shows the effect of keep number parameter K. The third row compares AQE to some variants.

The second row in Figure 3 shows how the keep parameter can affect the algorithm's performance: under the same high UTD value, as K decreases, the average normalized Q bias goes from over-estimation to under-estimation. Consistent with the theoretical result in Theorem 1, by decreasing K we lower the average bias. When K becomes too small, the Q estimate becomes too conservative and starts to have negative bias, which makes learning difficult. We see that $K = 16$ has an average bias closest to 0 and also a consistently small std of bias. These results are similar for the other four environments, as shown in the Appendix.

The third row in Figure 3 shows results for variants of the target computation methods. The Median curve uses the median value of all the Q estimates in the ensemble to compute the Q target. The

RemoveMinMax curve drops the minimum and maximum values of all the Q estimates in the ensemble to compute the Q target. We see that these two variants give larger positive Q bias values.

We also considered different combinations of ensemble size N and the number of multi-heads h while keeping the total number of Q-function estimates $N \cdot h$ fixed. We performed these experiments for all five environments. In terms of performance, we found the two best combinations to be $N = 20$, $h = 1$ and $N = 10$, $h = 2$, with the former being about 50% slower than the latter in terms of computation time. We also consider endowing REDQ with the same multi-head ensemble architecture as AQE and find that it does not improve REDQ substantially (additional details in Appendix F).

We also consider comparing AQE with SAC and TQC, all using a more aggressive UTD ratio of $G = 5$. Although SAC becomes more sample efficient with $G = 5$, AQE continues to outperform both algorithms except for Humanoid, where once again TQC performs somewhat better than AQE for the final stage of training (Additional details in appendix E).

5 Conclusion

Perhaps the most important takeaway from this study is that a simple model-free algorithm can do surprisingly well, providing state-of-art performance at all stages of training. There is no need for a model, distributional representation of the return, or in-target randomization to achieve high sample efficiency and asymptotic performance.

With extensive experiments and ablations, we show that AQE is both performant and robust. In both OpenAI Gym and DMControl, AQE is able to achieve superior performance in all stages of training with the same hyperparameters, and it can be further improved with per-task finetuning. Our ablations show that AQE is robust to small changes in the hyperparameters. Our theoretical results complement the experimental results, showing that the estimation bias can be controlled by either varying the ensemble size N or the keep parameter K.

AQE along with prior works show that a high update-to-data ratio combined with refined bias control can lead to very significant performance gain. An interesting future work direction is to investigate whether there are other critical factors (in addition to bias control) that can allow us to further benefit from a high update-to-data ratio, and achieve even better sample efficiency with simple model-free methods.

References

R. Agarwal, D. Schuurmans, and M. Norouzi. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, 2020.

O. Anschel, N. Baram, and N. Shimkin. Averaged-DQN: Variance reduction and stabilization for deep reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning*, pages 176–185, 2017.

M. G. Bellemare, W. Dabney, and R. Munos. A distributional perspective on reinforcement learning. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 449–458. PMLR, 06–11 Aug 2017.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

X. Chen, C. Wang, Z. Zhou, and K. W. Ross. Randomized ensembled double q-learning: Learning fast without a model. In *International Conference on Learning Representations*, 2021.

W. Dabney, G. Ostrovski, D. Silver, and R. Munos. Implicit quantile networks for distributional reinforcement learning. In J. G. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 1104–1113, 2018a.

W. Dabney, M. Rowland, M. G. Bellemare, and R. Munos. Distributional reinforcement learning with quantile regression. In S. A. McIlraith and K. Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018b.

Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 1329–1338, 2016.

S. Faußer and F. Schwenker. Neural network ensembles in reinforcement learning. *Neural Process. Lett.*, page 55–69, 2015.

S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1587–1596, 2018.

T. Haarnoja, A. Zhou, P. Abbeel, and S. 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, 2018a.

T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine. Soft actor-critic algorithms and applications. Technical report, 2018b.

H. v. Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, page 2094–2100. AAAI Press, 2016.

P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. In *AAAI*, 2018.

R. Islam, P. Henderson, M. Gomrokchi, and D. Precup. Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. *CoRR*, 2017.

M. Janner, J. Fu, M. Zhang, and S. Levine. When to trust your model: Model-based policy optimization. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 2019.

A. Kuznetsov, P. Shvechikov, A. Grishin, and D. Vetrov. Controlling overestimation bias with truncated mixture of continuous distributional quantile critics. In *Proceedings of the 37th International Conference on Machine Learning*, pages 5556–5566, 2020.

Q. Lan, Y. Pan, A. Fyshe, and M. White. Maxmin q-learning: Controlling the estimation bias of q-learning. In *8th International Conference on Learning Representations*, 2020.

K. Lee, M. Laskin, A. Srinivas, and P. Abbeel. SUNRISE: A simple unified framework for ensemble learning in deep reinforcement learning. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021*, volume 139, pages 6131–6141, 2021.

S. Lee, S. Purushwalkam, M. Cogswell, D. J. Crandall, and D. Batra. Why M heads are better than one: Training a diverse ensemble of deep networks. *CoRR*, 2015.

T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In *4th International Conference on Learning Representations*, 2016.

I. Osband, C. Blundell, A. Pritzel, and B. Van Roy. Deep exploration via bootstrapped dqn. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

S. Thrun and A. Schwartz. Issues in using function approximation for reinforcement learning. In *Proceedings of the Fourth Connectionist Models Summer School*, pages 255–263. Hillsdale, NJ, 1993.

E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

H. van Hasselt. Double q-learning. In *Advances in Neural Information Processing Systems*, pages 2613–2621, 2010.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Aggressive Q-Learning with Ensembles: Achieving Both High Sample Efficiency and High Asymptotic Performance Supplementary Material

A Hyperparameters and implementation details

Table 4 gives a list of hyperparameters used in the experiments. Most of AQE's hyperparameters are made the same as in the REDQ paper to ensure fairness and consistency in comparisons, except that AQE has 2-head critic networks. As compared with AQE and REDQ, TQC uses a larger critic network with 3 layers of 512 units per layer. In table 5, we report the dropped atoms d for TQC and the number of Q values we keep in the ensemble to calculate the target in AQE. In algorithm 1, we provide detailed pseudo-code.

Table 4: Hyperparameter values.

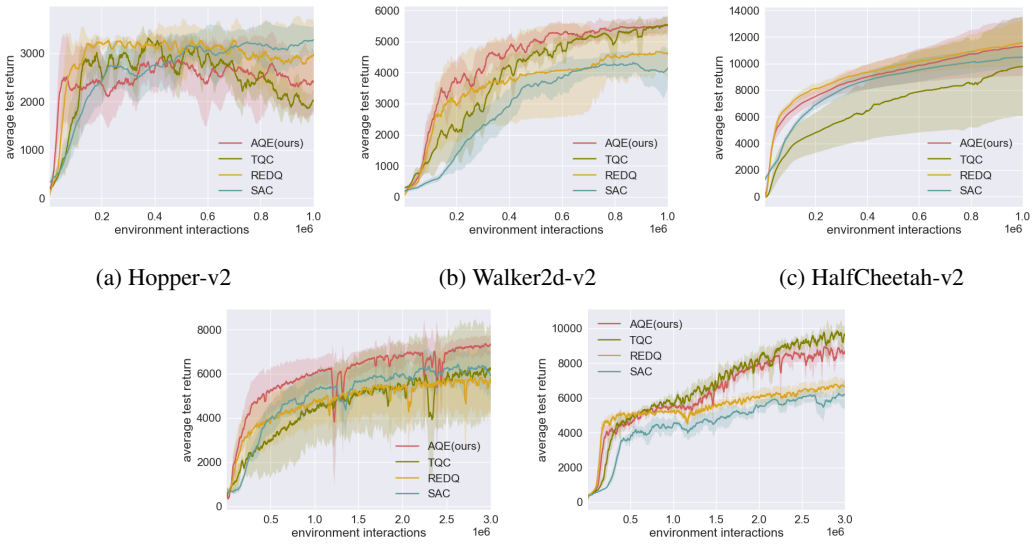
Hyperparameters	AQE	SAC	REDQ	TQC
optimizer	Adam			
learning rate	$3 \cdot 10^{-4}$			
discount(γ)	0.99			
target smoothing coefficient(ρ)	0.005			
replay buffer size	$1 \cdot 10^{6}$			
number of critics N	10	2	10	5
number of hidden layers in critic networks	2	2	2	3
size of hidden layers in critic networks	256	256	256	512
number of heads in critic networks h	2	1	1	25
number of hidden layers in policy network	2			
size of hidden layers in policy network	256			
mini-batch size	256			
nonlinearity	ReLU			
UTD ratio G	5	1	5	1

Table 5: Environment-dependent hyper-parameters for TQC and AQE.

Environment	Dropped atoms per critic	Kept Q values out of $N \cdot h$ values
Hopper	5	10
HalfCheetah	0	20
Walker	2	16
Ant	2	16
Humanoid	2	16

B Additional Results for AQE, TQC, REDQ and SAC in MuJoCo Benchamrk with Fixed Hyper-parameters

We present the experiment on the five MuJoCo environments with the same hyperparameter values across environments for TQC (drop two atoms per network) and AQE (K = 16) in Figure 4.



(a) Hopper-v2 (b) Walker2d-v2 (c) HalfCheetah-v2

(d) Ant-v2 (e) Humanoid-v2

Figure 4: Performance for AQE and TQC using same hyper-parameters across the five environments. AQE uses $K = 16$ and TQC uses atoms = 2 per critic.

Table 6: Early-stage performance comparison of SAC, TQC, REDQ and AQE when AQE and TQC using the same hyperparameters across the environments. On average, AQE performs 2.71 times better than SAC, 1.59 times better than TQC and 1.02 times better than REDQ.

Amount of data	SAC	TQC	REDQ	AQE	AQE/SAC	AQE/TQC	AQE/REDQ
Hopper at 100K	1456	1719	2747	2294	1.58	1.33	0.84
Walker2d at 100K	501	1215	1810	2150	4.29	1.77	1.19
HalfCheetah at 100K	3055	3594	6876	6325	2.10	1.76	0.92
Ant at 250K	2107	2344	3279	4153	1.97	1.77	1.27
Humanoid at 250K	1094	3038	4535	3973	3.63	1.31	0.88
Average at early stage	-	-	-	-	2.71	1.59	1.02

Table 7: Late-stage performance comparison of SAC, TQC, REDQ and AQE when AQE and TQC using the same hyperparameters across the environments. On average, AQE performs 16% better than SAC, 9% better than TQC and 11% times better than REDQ.

Amount of data	SAC	TQC	REDQ	AQE	AQE/SAC	AQE/TQC	AQE/REDQ
Hopper at 1M	3282	2024	2954	2404	0.73	1.19	0.81
Walker2d at 1M	4134	5532	4637	5517	1.33	1.00	1.19
HalfCheetah at 1M	10475	9792	11562	11293	1.08	1.15	0.98
Ant at 3M	5903	6186	5785	7345	1.24	1.19	1.27
Humanoid at 3M	6177	9593	6649	8680	1.41	0.91	1.31
Average at late stage	-	-	-	-	1.16	1.09	1.11

C Additional Results for AQE, TQC, REDQ and SAC in DeepMind Control Suite Benchmark

Figure 2 presents the performance of AQE, TQC, REDQ and SAC for 9 DeepMind Control Suite (DMC) environments. We can see that AQE continues to outperform TQC except for Humanoid run environment, where TQC performs better than AQE in the final stage training. AQE and REDQ have comparable results in some of the DMC environments, however, AQE usually outperforms REDQ in the more challenging environments, such as Hopper-hop, Humanoid-run and Quadruped-run. We report detailed early-stage and late-stage performance comparisons of all algorithms in Table 8 and Table 9. On average, in the early stage of training, AQE performs 13.71 times better than SAC, 7.59 times better than TQC and 1.02 times better than REDQ. In the late-stage training, on average, AQE performs 1.37 times better than SAC, 1.08 times better than TQC and 1.03 times better than REDQ.

Table 8: Early-stage performance comparison of SAC, TQC, REDQ and AQE when AQE and TQC using the same hyperparameters across the DMC environments. On average, AQE performs 13.71 times better than SAC, 7.59 times better than TQC and 1.02 times better than REDQ.

Amount of data	SAC	TQC	REDQ	AQE	AQE/SAC	AQE/TQC	AQE/REDQ
Cheetah-run at 100K	205	235	317	339	1.65	1.44	1.07
Fish-swim at 100K	121	149	234	230	1.90	1.54	0.98
Hopper-hop at 100K	2	11	50	64	32	5.81	1.28
Quadruped-walk at 100K	116	172	452	341	2.94	1.98	0.75
Quadruped-run at 100K	114	111	294	284	2.49	2.56	0.97
Walker-run at 100K	305	372	468	457	1.50	1.23	0.98
Humanoid-stand at 100K	5	5	37	52	10.4	10.4	1.41
Humanoid-walk at 100K	1	1	57	40	40	40	0.70
Humanoid-run at 250K	2	18	59	61	30.5	3.39	1.03
Average at early stage	-	-	-	-	13.71	7.59	1.02

Table 9: Late-stage performance comparison of SAC, TQC, REDQ and AQE when AQE and TQC using the same hyperparameters across the DMC environments. On average, AQE performs 1.37 times better than SAC, 1.08 times better than TQC and 1.03 times better than REDQ.

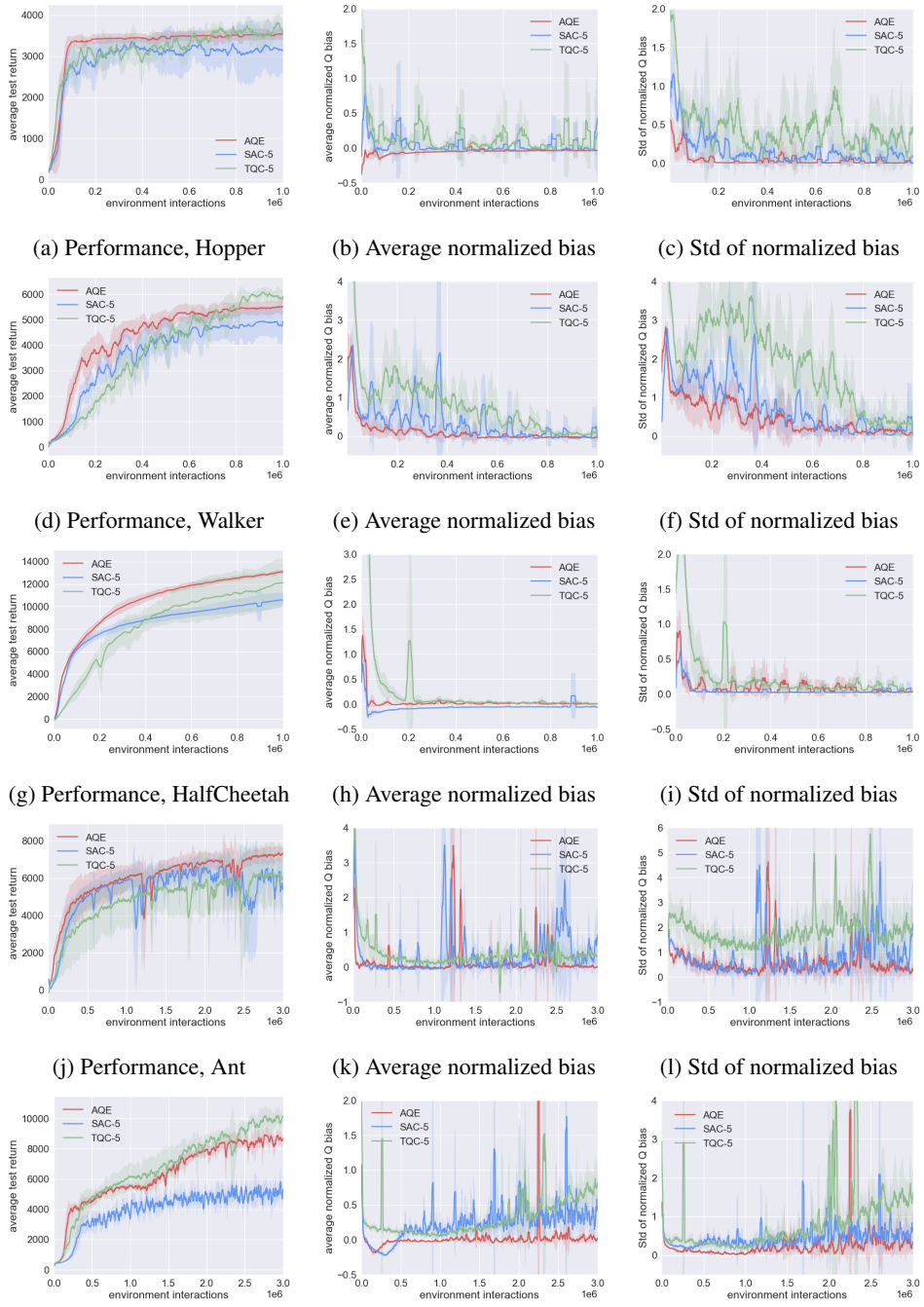
Amount of data	SAC	TQC	REDQ	AQE	AQE/SAC	AQE/TQC	AQE/REDQ
Cheetah-run at 1M	734	829	844	856	1.17	1.03	1.01
Fish-swim at 1M	639	722	753	747	1.17	1.03	0.99
Hopper-hop at 1M	293	256	279	294	1.00	1.15	1.05
Quadruped-walk at 1M	871	948	949	948	1.09	1.00	1.00
Quadruped-run at 1M	676	893	904	928	1.37	1.04	1.03
Walker-run at 1M	660	780	826	808	1.22	1.04	0.98
Humanoid-stand at 1M	323	429	547	546	1.69	1.27	1.00
Humanoid-walk at 1M	325	427	596	576	1.77	1.35	0.97
Humanoid-run at 4.5M	146	324	216	271	1.86	0.84	1.25
Average at late stage	-	-	-	-	1.37	1.08	1.03

Table 10: Sample efficiency comparison of SAC, TQC, REDQ and AQE. The numbers show the amount of data collected when the specified performance level is reached (roughly corresponding to 90% of SAC's final performance). The last three columns show how many times AQE is more sample efficient than SAC, TQC and REDQ in reaching that performance level.

Performance	SAC	TQC	REDQ	AQE	AQE/SAC	AQE/TQC	AQE/REDQ
Cheetah-run at 700	746K	440K	506K	350K	2.13	1.26	1.45
Fish-swim at 600	794K	494K	317K	417K	1.90	1.18	0.76
Hopper-hop at 250	580K	856K	451K	371K	1.56	2.31	1.22
Quadruped-walk at 800	844K	301K	302K	236K	3.58	1.28	1.28
Quadruped-run at 650	942K	521K	267K	248K	3.80	2.10	1.08
Walker-run at 600	516K	201K	156K	174K	2.97	1.16	0.90
Humanoid-stand at 250	626K	429K	279K	342K	1.83	1.25	0.82
Humanoid-walk at 300	820K	523K	279K	300K	2.73	1.74	0.93
Humanoid-run at 120	3940K	1100K	602K	603K	6.53	1.82	1.00
Average	-	-	-	-	3.00	1.57	1.05

D Additional Results for AQE, SAC-5 and TQC-5

Figure 5 presents the performance of AQE, SAC-5 and TQC-5 for all the environments. SAC-5 and TQC-5 uses UTD ratio G = 5 for SAC and TQC, respectively. We can see that AQE continues to outperform both algorithms except for Humanoid, where TQC performs somewhat better than AQE in the final stage training. SAC becomes more sample efficient with $G = 5$; however, AQE still beats SAC-5 by a large margin.



(a) Performance, Hopper (b) Average normalized bias (c) Std of normalized bias

(d) Performance, Walker (e) Average normalized bias (f) Std of normalized bias

(g) Performance, HalfCheetah (h) Average normalized bias (i) Std of normalized bias

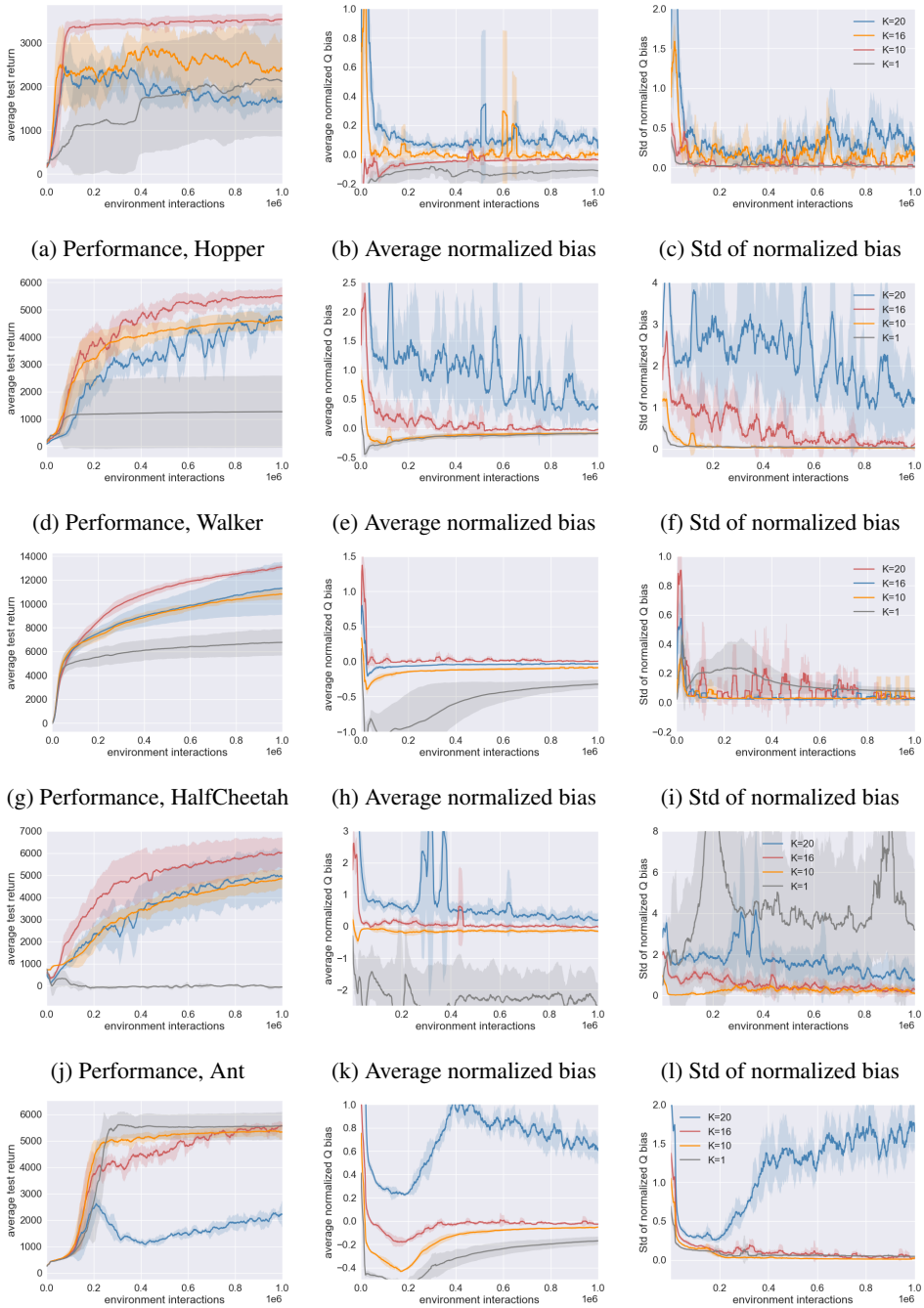
(j) Performance, Ant (k) Average normalized bias (l) Std of normalized bias

(m) Performance, Humanoid (n) Average normalized bias (o) Std of normalized bias

Figure 5: Performance, average and std of normalized Q bias for AQE, SAC-5 and TQC-5. All of the algorithms in this experiment use UTD = 5.

E Additional Results for parameter K

Due to lack of space, Figure 3 only compares different AQE keep numbers K for Ant. Figure 6 shows the performance, average estimation bias and standard deviation for all five environments. Consistent with the theoretical result in Theorem 1, by decreasing K, we lower the average bias.



(a) Performance, Hopper (b) Average normalized bias (c) Std of normalized bias

(d) Performance, Walker (e) Average normalized bias (f) Std of normalized bias

(g) Performance, HalfCheetah (h) Average normalized bias (i) Std of normalized bias

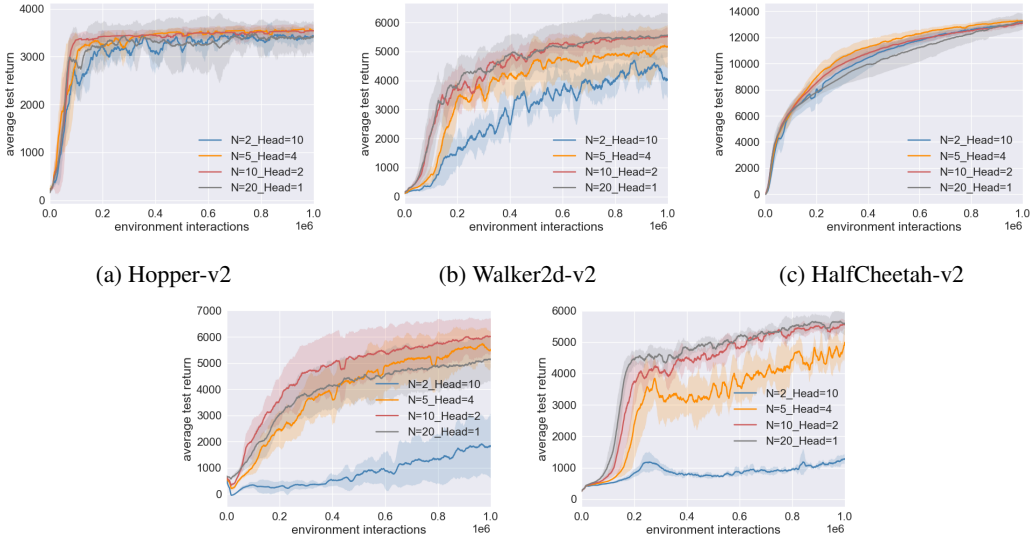
(j) Performance, Ant (k) Average normalized bias (l) Std of normalized bias

(m) Performance, Humanoid (n) Average normalized bias (o) Std of normalized bias

Figure 6: Performance, average and std of normalized Q bias for AQE with different values of K.

F Additional Results for Multi-head Architecture

Due to lack of space, Figure 3 only compares the different size of the ensemble N and the number of heads h for Ant. Figure 7 shows the results for all five environments. We can see that the combination of $N = 10, h = 2$ and $N = 20, h = 1$ have comparable performance. However, $N = 10$ and $h = 2$ is faster in terms of computation time.

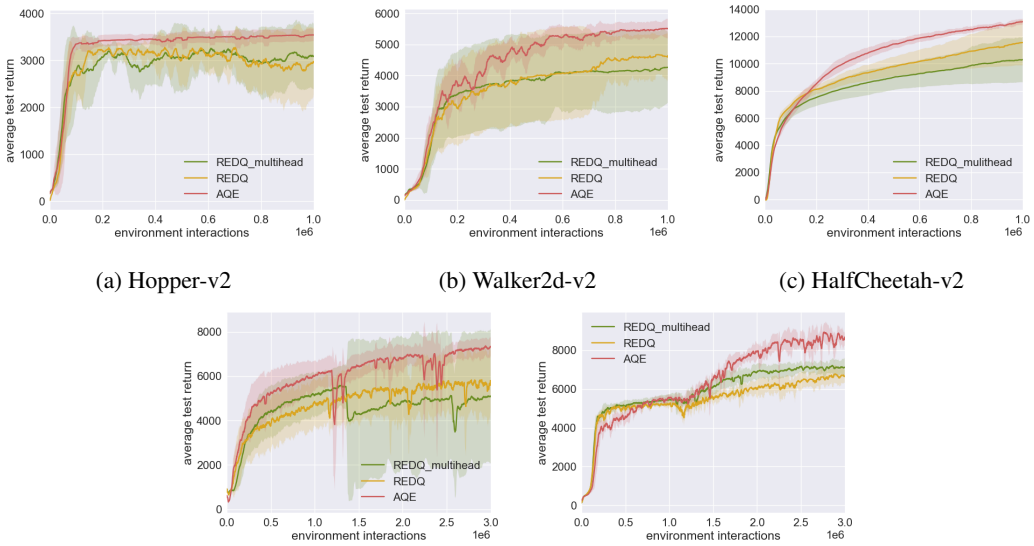


(a) Hopper-v2 (b) Walker2d-v2 (c) HalfCheetah-v2

(d) Ant-v2 (e) Humanoid-v2

Figure 7: Performance for AQE with different combinations of number of Q networks and number of heads.

Will the performance of REDQ match that of AQE if we also provide REDQ a multi-head architecture? Figure 8 examines the performance of REDQ when it is endowed with the same multi-head architecture as AQE. We see that the performance of REDQ does not substantially improve.



(a) Hopper-v2 (b) Walker2d-v2 (c) HalfCheetah-v2

(d) Ant-v2 (e) Humanoid-v2

Figure 8: Performance of REDQ with N=10 and heads = 2 as compared with REDQ and AQE.

G Theoretical Results

In this section, we characterize how changing the size of the ensemble N and the keep parameter K affects the estimation bias term in the AQE algorithm. We will restrict our analysis to the tabular version of AQE shown in algorithm 2.

Our analysis will follow similar lines of reasoning as Lan et al. (2020) and Chen et al. (2021) which extends upon the theoretical framework introduced in Thrun and Schwartz (1993).

For each $a \in \mathcal{A}$, let $E_{K,N}(s,a)$ be the ensemble members in $\{1,\ldots,N\}$ with the K lowest values of $Q^j(s,a)$, $j = 1,\ldots,N$. In the tabular case, the target for the Q networks take the form:

$$r + \gamma \max_{a'} \left(\frac{1}{K} \sum_{j \in E_{K,N}(s',a')} Q^j(s',a') \right). \tag{1}$$

Define the *post-update estimation bias* as

$$\begin{aligned} Z_{K,N} &:= r + \gamma \max_{a'} \left(\frac{1}{K} \sum_{j \in E_{K,N}(s',a')} Q^j(s',a') \right) - \left(r + \gamma \max_{a'} Q^\pi(s',a') \right) \\ &= \gamma \left[\max_{a'} \left(\frac{1}{K} \sum_{j \in E_{K,N}(s',a')} Q^j(s',a') \right) - \max_{a'} Q^\pi(s',a') \right] \end{aligned} \tag{2}$$

Under this definition, if $\mathbb{E}[Z_{K,N}] > 0$, then the expected post-update estimation bias is positive and there is a tendency for the positive bias to accumulate during updates. Similarly, if $\mathbb{E}[Z_{K,N}] < 0$, then the expected post-update estimation bias is negative and there is a tendency for the negative bias to accumulate during updates. Ideally, we would like $\mathbb{E}[Z_{K,N}] \approx 0$

Also let

$$Q^j(s,a) = Q^\pi(s,a) + e^j(s,a) \tag{3}$$

where $e^j(s,a)$ is an independent and identically distributed error term across all j's and all a's for each fixed s. We further assume that $\mathbb{E}[e^j(s,a)] = 0$. Note that with this assumption

$$\mathbb{E}\left[\frac{1}{N} \sum_{j=1}^{N} Q^j(s,a) \right] - Q^\pi(s,a) = 0,$$

that is the pre-update estimation bias is zero. The following theorem shows how the expected estimation bias changes with N and K:

Theorem 1. *The following results hold for* $\mathbb{E}[Z_{K,N}]$*:*

1. $\mathbb{E}[Z_{N,N}] \geq 0$ *for all* $N \geq 1$.
2. $\mathbb{E}[Z_{K-1,N}] \leq \mathbb{E}[Z_{K,N}]$ *for all* $K \leq N$.
3. $\mathbb{E}[Z_{K,N+1}] \leq \mathbb{E}[Z_{K,N}]$.
4. *Suppose that* $e^j_{sa} \leq c$ *for some* $c > 0$ *for all* s, a *and* j*. Then there exists an* N *sufficiently large and* $K < N$ *such that* $\mathbb{E}[Z_{K,N}] < 0$.

Proof Sketch. Part 1 is a result of Jensen's Inequality, and Parts 2 and 3 can be shown by analyzing how the average of the K smallest ensembles changes when adding an extra ensemble model. Given the first three parts, we only need to show that $\mathbb{E}[Z_{1,N}] < 0$ to show that there exists a K for a sufficiently large N where the expected bias is negative. See the next section for full proof. □

Theorem 1 shows that we can control the expected post-update bias $\mathbb{E}[Z_{K,N}]$ through adjusting K and N. More concretely, we can bring the bias term from above zero (i.e. over estimation) to under zero (i.e. under estimation) by decreasing K and/or increasing N.

We note also that similar to Chen et al. (2021), we make very few assumptions on the error term $e_{s,a}$. This is in contrary to Thrun and Schwartz (1993) and Lan et al. (2020), both of whom assume that the error term is uniformly distributed.

G.1 Tabular AQE with N ensemble members and d drops

Algorithm 2 Tabular AQE

Initialize: $Q^j(s,a)$ for all $s \in \mathcal{S}$, $a \in \mathcal{A}$, $j = 1, \ldots, N$.

repeat

For some state $s \in \mathcal{S}$, choose $a \in \mathcal{A}$ based on $\left\{Q^j(s,a)\right\}_{j=1}^{N}$, observe r, s'.

For each $a' \in \mathcal{A}$, let $E_{K,N}(s',a')$ be the ensemble members in $\{1, \ldots, N\}$ with the K lowest values of $Q^j(s',a')$, $j = 1, \ldots, N$.

Get target

$$y = r + \gamma \max_{a' \in \mathcal{A}} \frac{1}{K} \sum_{j \in E_{K,N}(s',a')} Q^j(s',a')$$

for $j = 1, \ldots, N$ **do**

Update each $Q^j(s,a)$

$$Q^j(s,a) \leftarrow Q^j(s,a) + \alpha(y - Q^j(s,a))$$

$s \leftarrow s'$

until end

H Proofs

We first present the following lemma:

Lemma 1 (Chen et al., 2021)**.** *Let $X_1, X_2, \ldots$ be an infinite sequence of $i.i.d.$ random variables with cdf $F(x)$ and let $\tau = \inf x : F(x) > 0$. Also let $Y_N = \min\{X_1, X_2, \ldots, X_N\}$. Then $Y_1, Y_2, \ldots$ converges to τ almost surely.*

Proof. See Appendix A.2 of Chen et al. (2021) □

Theorem 1. *The following results hold for $\mathbb{E}[Z_{K,N}]$:*

1. $\mathbb{E}[Z_{N,N}] \geq 0$ *for all* $N \geq 1$.
2. $\mathbb{E}[Z_{K-1,N}] \leq \mathbb{E}[Z_{K,N}]$ *for all* $K \leq N$.
3. $\mathbb{E}[Z_{K,N+1}] \leq \mathbb{E}[Z_{K,N}]$.
4. *Suppose that $e^j_{sa} \leq c$ for some $c > 0$ for all s, a and j. Then there exists an N sufficiently large and $K < N$ such that $\mathbb{E}[Z_{K,N}] < 0$.*

Proof.

1. By definition,

$$\begin{aligned}\mathbb{E}[Z_{N,N}] &= \gamma\,\mathbb{E}\left[\max_{a'}\left(\frac{1}{N}\sum_{j=1}^{N} Q^j(s',a')\right) - \max_{a'} Q^\pi(s',a')\right] \\ &\geq \gamma\left[\max_{a'} E\left[\left(\frac{1}{N}\sum_{j=1}^{N} Q^j(s',a')\right)\right] - \max_{a'} Q^\pi(s',a')\right] \\ &= \gamma\left[\max_{a'} Q^\pi(s',a') - \max_{a'} Q^\pi(s',a')\right] = 0\end{aligned} \tag{4}$$

2. Let

$$\bar{Q}_{K,N}(s,a) = \frac{1}{K}\sum_{j\in E_{K,N}} Q^j(s,a). \tag{5}$$

Since for any state s, $\max_a \bar{Q}_{K+1,N}(s,a) \geq \max_a \bar{Q}_{K,N}(s,a)$,

$$\begin{aligned}\mathbb{E}[Z_{K+1,N}] &= \gamma\,\mathbb{E}\left[\max_{a'} \bar{Q}_{K+1,N}(s',a') - \max_{a'} Q^\pi(s',a')\right] \\ &\geq \gamma\,\mathbb{E}\left[\max_{a'} \bar{Q}_{K,N}(s',a') - \max_{a'} Q^\pi(s',a')\right] \\ &= \mathbb{E}[Z_{K,N}]\end{aligned} \tag{6}$$

3. Comparing $\mathbb{E}[Z_{K,N}]$ and $\mathbb{E}[Z_{K,N+1}]$ is equivalent to comparing $\bar{Q}_{K,N}(s,a)$ and $\bar{Q}_{K,N+1}(s,a)$. Since $e^j(s,a)$ is i.i.d., by extension $Q^j(s,a)$ is also i.i.d. for $j = 1, 2, \cdots$. Suppose $Q^j(s,a)$ is drawn from some probability distribution F, then given $\bar{Q}_{K,N}(s,a)$, $\bar{Q}_{K,N+1}(s,a)$ can be calculated by generating an additional $Q^i(s,a)$ from F. The new sample $Q^i(s,a)$ affects the calculation of $\bar{Q}_{K,N+1}(s,a)$ under the following two cases:

   - If $Q^i(s,a) > \max_{j\in E_{K,N}} Q^j(s,a)$, then the lowest K values remain unchanged hence $\bar{Q}_{K,N}(s,a) = \bar{Q}_{K,N+1}(s,a)$.
   - Else if $Q^i(s,a) \leq \max_{j\in E_{K,N}} Q^j(s,a)$, then $\max_{j\in E_{K,N}} Q^j(s,a)$ would be removed from and $Q^i(s,a)$ would be added to the set of lowest K values, therefore $\bar{Q}_{K,N+1}(s,a) \leq \bar{Q}_{K,N}(s,a)$.

   Combining the two cases $\bar{Q}_{K,N+1}(s,a) \leq \bar{Q}_{K,N}(s,a)$, therefore $\mathbb{E}[Z_{K,N+1}] \leq \mathbb{E}[Z_{K,N}]$

4. Since $\mathbb{E}[Z_{N,N}] \geq 0$, $\mathbb{E}[Z_{K,N}] \leq \mathbb{E}[Z_{K+1,N}]$ and $\mathbb{E}[Z_{K,N+1}] \leq \mathbb{E}[Z_{K,N}]$. It is suffice to show that $\mathbb{E}[Z_{1,N}] < 0$ for some N. The rest of the proof largely follows Theorem 1 of Chen et al. (2021).

   Let $\tau = \inf\{x : F_a(x) > 0\}$ where $F_a(x)$ is the cdf of $Q^j(s,a)$, $j = 1, 2, \ldots$. By Lemma 1, $\bar{Q}_{1,N}(s,a) = \min_{1 \leq j \leq N} Q^j(s,a)$ converges almost surely to to τ_a for each a. Since the action space is finite, it then follows that $\max_a \bar{Q}_{1,N}(s,a)$ converges almost surely to to $\tau = \max_a \tau_a$. Due to our assumption that $e^j(s,a) \leq c$ and that $Q^\pi(s,a)$ is finite, it then follows that $\max_a \bar{Q}_{1,N}(s,a)$ is also bounded above. By Part 3 of the theorem, $\bar{Q}_{1,N}(s,a)$ is monotonoically decreasing w.r.t. N. and since $\max_a \bar{Q}_{1,N}(s,a)$ is also bounded above and converges almost surely to τ, we have

$$\begin{aligned}\mathbb{E}[Z_{1,N}] &= \gamma \left(\mathbb{E}[\max_a \min_{1 \leq j \leq N} Q^j(s,a)] - \max_a Q^\pi(s,a) \right) \\ &= \gamma \left(\mathbb{E}[\max_a Y_a^N] - \max_a Q^\pi(s,a) \right) \overset{N \to \infty}{\longrightarrow} \gamma \left(\max_a \tau_a - \max_a Q^\pi(s,a) \right) < 0 \end{aligned} \tag{7}$$

   where the last equality follows from the assumption that the error $e^j(s,a)$ is non-trivial, and hence $\tau_a < \max_a Q^\pi(s,a)$ for all a. Therefore for a sufficiently large N, there exists a $1 \leq K \leq N$ such that $\mathbb{E}_{K,N} < 0$.

□

I Computing Infrastructure

Each experiment is run on a single Nvidia 2080-Ti GPU with CentOS Linux System.