Uncertainty-Based Offline Reinforcement Learning with Diversified Q-Ensemble

Gaon An[*12], **Seungyong Moon**[*12], **Jang-Hyun Kim**[12], **Hyun Oh Song**[†123]
Seoul National University[1]
Neural Processing Research Center[2]
DeepMetrics[3]
{white0234,symoon11,janghyun,hyunoh}@mllab.snu.ac.kr

Abstract

Offline reinforcement learning (offline RL), which aims to find an optimal policy from a previously collected static dataset, bears algorithmic difficulties due to function approximation errors from out-of-distribution (OOD) data points. To this end, offline RL algorithms adopt either a constraint or a penalty term that explicitly guides the policy to stay close to the given dataset. However, prior methods typically require accurate estimation of the behavior policy or sampling from OOD data points, which themselves can be a non-trivial problem. Moreover, these methods under-utilize the generalization ability of deep neural networks and often fall into suboptimal solutions too close to the given dataset. In this work, we propose an uncertainty-based offline RL method that takes into account the confidence of the Q-value prediction and does not require any estimation or sampling of the data distribution. We show that the clipped Q-learning, a technique widely used in online RL, can be leveraged to successfully penalize OOD data points with high prediction uncertainties. Surprisingly, we find that it is possible to substantially outperform existing offline RL methods on various tasks by simply increasing the number of Q-networks along with the clipped Q-learning. Based on this observation, we propose an ensemble-diversified actor-critic algorithm that reduces the number of required ensemble networks down to a tenth compared to the naive ensemble while achieving state-of-the-art performance on most of the D4RL benchmarks considered.

1 Introduction

Over the recent years, deep reinforcement learning (deep RL) has achieved considerable success in various domains such as robotics [20], recommendation systems [6], and strategy games [26]. However, a major drawback of RL algorithms is that they adopt an active learning procedure, where training steps require active interactions with the environment. This trial-and-error procedure can be prohibitive when scaling RL to real-world applications such as autonomous driving and healthcare, as exploratory actions can cause critical damage to the agent or the environment [19]. *Offline* RL, also known as *batch* RL, aims to overcome this problem by learning policies using only previously collected data without further interactions with the environment [2, 11, 19].

Even though offline RL is a promising direction to lead a more *data-driven* way of solving RL problems, recent works show offline RL faces new algorithmic challenges [19]. Typically, if the coverage of the dataset is not sufficient, vanilla RL algorithms suffer severely from extrapolation

[*]First two authors have equal contributions
[†]Corresponding author

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

error, overestimating the Q-values of out-of-distribution (OOD) state-action pairs [15]. To this end, most offline RL methods apply some constraints or penalty terms on top of the existing RL algorithms to enforce the learning process to be more conservative. For example, some prior works explicitly regularize the policy to be close to the behavior policy that was used to collect the data [11, 15]. A more recent work instead penalizes the Q-values of OOD state-action pairs to enforce the Q-values to be more pessimistic [16].

While these methods achieve significant performance gains over vanilla RL methods, they either require an estimation of the behavior policy or explicit sampling from OOD data points, which themselves can be non-trivial to solve. Furthermore, these methods do not utilize the generalization ability of the Q-function networks and prohibit the agent from approaching any OOD state-actions without any consideration on whether they are good or bad. However, if we can identify OOD data points where we can predict their Q-values with high confidence, it is more effective not to restrain the agent from choosing those data points.

From this intuition, we propose an uncertainty-based model-free offline RL method that effectively quantifies the uncertainty of the Q-value estimates by an ensemble of Q-function networks and does not require any estimation or sampling of the data distribution. To achieve this, we first show that a well-known technique from online RL, the clipped Q-learning [10], can be successfully leveraged as an uncertainty-based penalization term. Our experiments reveal that we can achieve state-of-the-art performance on various offline RL tasks by solely using this technique with increased ensemble size. To further improve the practical usability of the method, we develop an ensemble diversifying objective that significantly reduces the number of required ensemble networks. We evaluate our proposed method on D4RL benchmarks [9] and verify that the proposed method outperforms the previous state-of-the-art by a large margin on various types of environments and datasets.

2 Preliminaries

We consider an environment formulated as a Markov Decision Process (MDP) defined by a tuple $(\mathcal{S}, \mathcal{A}, T, r, d_0, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $T(\mathbf{s}' \mid \mathbf{s}, \mathbf{a})$ is the transition probability distribution, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, d_0 is the initial state distribution, and $\gamma \in (0, 1]$ is the discount factor. The goal of reinforcement learning is to find an optimal policy $\pi(\mathbf{a} \mid \mathbf{s})$ that maximizes the cumulative discounted reward $\mathbb{E}_{\mathbf{s}_t, \mathbf{a}_t}\left[\sum_{t=0}^{\infty} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t)\right]$, where $\mathbf{s}_0 \sim d_0(\cdot)$, $\mathbf{a}_t \sim \pi(\cdot \mid \mathbf{s}_t)$, and $\mathbf{s}_{t+1} \sim T(\cdot \mid \mathbf{s}_t, \mathbf{a}_t)$.

One of the major approaches for obtaining such a policy is Q-learning [12, 20] which learns a state-action value function $Q_\phi(\mathbf{s}, \mathbf{a})$ parameterized by a neural network that represents the expected cumulative discounted reward when starting from state $\mathbf{s}$ and action $\mathbf{a}$. Standard actor-critic approach [14] learns this Q-function by minimizing the Bellman residual $\left(Q_\phi(\mathbf{s}, \mathbf{a}) - \mathcal{B}^{\pi_\theta} Q_\phi(\mathbf{s}, \mathbf{a})\right)^2$, where $\mathcal{B}^{\pi_\theta} Q_\phi(\mathbf{s}, \mathbf{a}) = \mathbb{E}_{\mathbf{s}' \sim T(\cdot|\mathbf{s}, \mathbf{a})}\left[r(\mathbf{s}, \mathbf{a}) + \gamma\, \mathbb{E}_{\mathbf{a}' \sim \pi_\theta(\cdot|\mathbf{s}')} Q_\phi\left(\mathbf{s}', \mathbf{a}'\right)\right]$ is the Bellman operator. In the context of offline RL, where transitions are sampled from a static dataset $\mathcal{D}$, the objective for the Q-network becomes minimizing

$$J_q(Q_\phi) := \mathbb{E}_{(\mathbf{s}, \mathbf{a}, \mathbf{s}') \sim \mathcal{D}}\left[\left(Q_\phi(\mathbf{s}, \mathbf{a}) - \left(r(\mathbf{s}, \mathbf{a}) + \gamma\, \mathbb{E}_{\mathbf{a}' \sim \pi_\theta(\cdot|\mathbf{s}')}\left[Q_{\phi'}\left(\mathbf{s}', \mathbf{a}'\right)\right]\right)\right)^2\right], \tag{1}$$

where $Q_{\phi'}$ represents the target Q-network softly updated for algorithmic stability [20]. The policy, which is also parameterized by a neural network, is updated in an alternating fashion to maximize the expected Q-value: $J_p(\pi_\theta) := \mathbb{E}_{\mathbf{s} \sim \mathcal{D}, \mathbf{a} \sim \pi_\theta(\cdot|\mathbf{s})}\left[Q_\phi(\mathbf{s}, \mathbf{a})\right]$.

However, as the policy is updated to maximize the Q-values, the actions $\mathbf{a}'$ sampled from the current policy in Equation (1) can be biased towards OOD actions with erroneously high Q-values. In the offline RL setting, such errors cannot be corrected by feedback from the environment as in online RL. To handle the error propagation from these OOD actions, most offline RL algorithms regularize either the policy [11, 15] or the Q-function [16] to be biased towards the given dataset. However, the policy regularization methods typically require an accurate estimation of the behavior policy. The previous state-of-the-art method CQL [16] instead learns conservative Q-values without estimating the behavior policy by penalizing the Q-values of OOD actions by

$$\min_\phi\ J_q(Q_\phi) + \alpha\Big(\mathbb{E}_{\mathbf{s} \sim \mathcal{D}, \mathbf{a} \sim \mu(\cdot|\mathbf{s})}\left[Q_\phi\left(\mathbf{s}, \mathbf{a}\right)\right] - \mathbb{E}_{(\mathbf{s}, \mathbf{a}) \sim \mathcal{D}}\left[Q_\phi\left(\mathbf{s}, \mathbf{a}\right)\right]\Big),$$

where μ is an approximation of the policy that maximizes the current Q-function. While CQL does not need explicit behavior policy estimation, it requires sampling from an appropriate action distribution $\mu(\cdot \mid \mathbf{s})$.

3 Uncertainty penalization with Q-ensemble

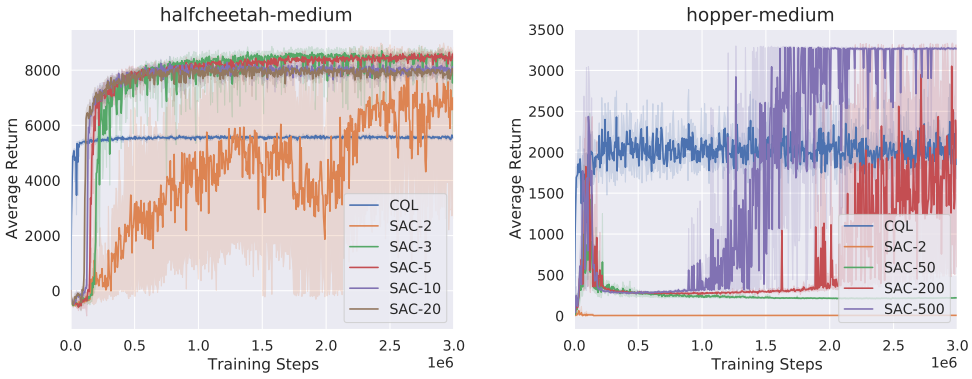


Figure 1: Performance of SAC-N on halfcheetah-medium and hopper-medium datasets while varying N, compared to CQL. 'Average Return' denotes the undiscounted return of each policies on evaluation. Results averaged over 4 seeds.

In this section, we turn our attention to a conventional technique from online RL, Clipped Double Q-learning [10], which uses the minimum value of two parallel Q-networks as the Bellman target: $y = r(\mathbf{s}, \mathbf{a}) + \gamma\, \mathbb{E}_{\mathbf{a}' \sim \pi_\theta(\cdot|\mathbf{s}')} \left[\min_{j=1,2} Q_{\phi'_j}(\mathbf{s}', \mathbf{a}') \right]$. Although this technique was originally proposed in online RL to mitigate the overestimation from general prediction errors, some offline RL algorithms [11, 15, 28] also utilize this technique to enforce their Q-value estimates to be more pessimistic. However, the isolated effect of the clipped Q-learning in offline RL was not fully analyzed in the previous works, as they use the technique only as an auxiliary term that adds up to their core methods.

To examine the ability of clipped Q-learning to prevent the overestimation in offline RL on its own, we modify SAC [12] by increasing the number of Q-ensembles from 2 to N:

$$\min_{\phi_i} \mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}' \sim \mathcal{D}} \left[\left(Q_{\phi_i}(\mathbf{s}, \mathbf{a}) - \left(r(\mathbf{s}, \mathbf{a}) + \gamma\, \mathbb{E}_{\mathbf{a}' \sim \pi_\theta(\cdot|\mathbf{s}')} \left[\min_{j=1,\ldots,N} Q_{\phi'_j}(\mathbf{s}', \mathbf{a}') - \beta \log \pi_\theta(\mathbf{a}' \mid \mathbf{s}') \right] \right) \right)^2 \right]$$

$$\max_{\theta} \mathbb{E}_{\mathbf{s} \sim \mathcal{D}, \mathbf{a} \sim \pi_\theta(\cdot|\mathbf{s})} \left[\min_{j=1,\ldots,N} Q_{\phi_j}(\mathbf{s}, \mathbf{a}) - \beta \log \pi_\theta(\mathbf{a} \mid \mathbf{s}) \right], \tag{2}$$

for $i = 1, \ldots, N$. We denote this modified algorithm as SAC-N.

Figure 1 shows the preliminary experiments on D4RL halfcheetah-medium and hopper-medium datasets [9] while varying N. Note that these datasets are constructed from suboptimal behavior policies. Surprisingly, as we gradually increase N, we can successfully find policies that outperform the previous state-of-the-art method (CQL) by a large margin. In fact, as we will present in Section 5, SAC-N outperforms CQL on various types of environments and data-collection policies.

To understand why this simple technique works so well, we can first interpret the clipping procedure (choosing the minimum value from the ensemble) as penalizing state-action pairs with high-variance Q-value estimates, which encourages the policy to favor actions that appeared in the dataset [11]. The dataset samples will naturally have lower variance compared to the OOD samples as the Bellman residual term in Equation (2) explicitly aligns the Q-value predictions for the dataset samples. More formally, we can regard this difference in variance as accounting for *epistemic uncertainty* [8] which refers to the uncertainty stemming from limited data and knowledge.

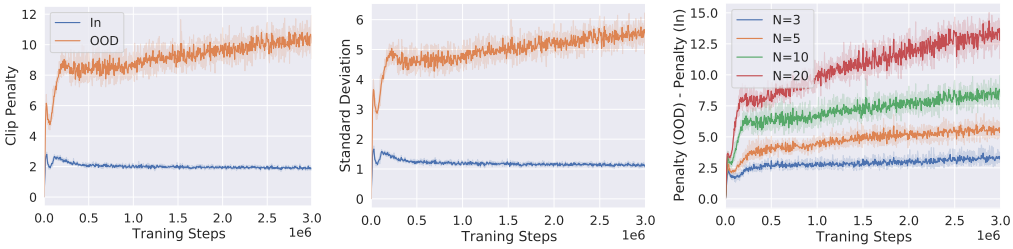
Utilization of the clipped Q-value relates to methods that consider the confidence bound of the Q-value estimates [24]. Online RL methods typically utilize the Q-ensemble to form an optimistic estimate of the Q-value, by adding the standard deviation to the mean of the Q-ensembles [18]. This optimistic Q-value, also known as the upper-confidence bound (UCB), can encourage the exploration

of unseen actions with high uncertainty. However, in offline RL, the dataset available during training is fixed, and we have to focus on *exploiting* the given data. For this purpose, it is natural to utilize the lower-confidence bound (LCB) of the Q-value estimates, for example by subtracting the standard deviation from the mean, which allows us to avoid risky state-actions.

The clipped Q-learning algorithm, which chooses the worst-case Q-value instead to compute the pessimistic estimate, can also be interpreted as utilizing the LCB of the Q-value predictions. Suppose $Q(\mathbf{s}, \mathbf{a})$ follows a Gaussian distribution with mean $m(\mathbf{s}, \mathbf{a})$ and standard deviation $\sigma(\mathbf{s}, \mathbf{a})$. Also, let $\{Q_j(\mathbf{s}, \mathbf{a})\}_{j=1}^N$ be realizations of $Q(\mathbf{s}, \mathbf{a})$. Then, we can approximate the expected minimum of the realizations following the work of Royston [23] as

$$\mathbb{E}\left[\min_{j=1,\dots,N} Q_j(\mathbf{s}, \mathbf{a})\right] \approx m(\mathbf{s}, \mathbf{a}) - \Phi^{-1}\left(\frac{N - \frac{\pi}{8}}{N - \frac{\pi}{4} + 1}\right)\sigma(\mathbf{s}, \mathbf{a}), \tag{3}$$

where Φ is the CDF of the standard Gaussian distribution. This relation indicates that using the clipped Q-value is similar to penalizing the ensemble mean of the Q-values with the standard deviation scaled by a coefficient dependent on N.



(a) Clip penalty (b) Standard deviation (c) Clip penalty gap

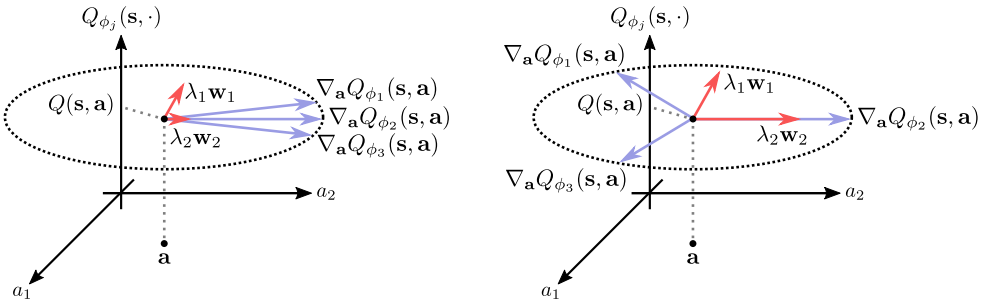
Figure 2: (a) and (b) each plots the size of the clip penalty and the standard deviation of the Q-value estimates for in-distribution (behavior) and OOD (random) actions while training SAC-10 on halfcheetah-medium dataset. (c) plots the gap of the clip penalty between the in-distribution and OOD actions while varying N. Results averaged over 4 seeds.

We now move on to the empirical analysis of the clipped Q-learning. Figure 2a compares the strength of the uncertainty penalty on in-distribution and OOD actions. Specifically, we compare actions sampled from two types of policies: (1) the behavior policy which was used to collect the dataset, and (2) the random policy which samples actions uniformly from the action space. For each policy, we measure the size of the penalty from the clipping as $\mathbb{E}_{\mathbf{s}\sim\mathcal{D},\mathbf{a}\sim\pi(\cdot|\mathbf{s})}[\frac{1}{N}\sum_{j=1}^N Q_{\phi_j}(\mathbf{s}, \mathbf{a}) - \min_{j=1,\dots,N} Q_{\phi_j}(\mathbf{s}, \mathbf{a})]$. Figure 2a shows that the clipping term penalizes the random state-action pairs much stronger than the in-distribution pairs throughout the training. For comparison, we also measure the standard deviation of the Q-values for each policy. The results in Figure 2b show that as we conjectured, the Q-value predictions for the OOD actions have a higher variance. We also find that the size of the penalty and the standard deviation are highly correlated, as we noted in Equation (3).

As we observe that OOD actions have higher variance on Q-value estimates, the effect of increasing N becomes obvious: it strengthens the penalty applied to the OOD samples compared to the dataset samples. To verify this, we measured the relative penalty applied to the OOD samples in Figure 2c and found that indeed the OOD samples are penalized relatively further as N increases.

4 Ensemble gradient diversification

Even though SAC-N outperforms existing methods on various tasks, it sometimes requires an excessively large number of ensembles to learn stably (*e.g.*, $N = 500$ for hopper-medium). While investigating its reason, we found that the performance of SAC-N is negatively correlated with the degree to which the input gradients of Q-functions $\nabla_{\mathbf{a}} Q_{\phi_j}(\mathbf{s}, \mathbf{a})$ are aligned, which increases with N. Figure 4 measures the minimum cosine similarity between the gradients of the Q-functions $\min_{i\neq j}\langle \nabla_{\mathbf{a}} Q_{\phi_i}(\mathbf{s}, \mathbf{a}), \nabla_{\mathbf{a}} Q_{\phi_j}(\mathbf{s}, \mathbf{a})\rangle$ to examine the alignment of the gradients while varying N on the D4RL hopper-medium dataset. The results imply that the performance of the learned policy degrades significantly when the Q-functions share a similar local structure.



(a) Without ensemble gradient diversification

(b) With ensemble gradient diversification

Figure 3: Illustration of the ensemble gradient diversification. The vector $\lambda_i\mathbf{w}_i$ represents the normalized eigenvector $\mathbf{w}_i$ of $\mathrm{Var}(\nabla_\mathbf{a}Q_{\phi_j}(\mathbf{s},\mathbf{a}))$ multiplied by its eigenvalue λ_i.

We now show that the alignment of the input gradients can induce insufficient penalization of near-distribution data points, which leads to requiring a large number of ensemble networks. Let $\nabla_\mathbf{a}Q_{\phi_j}(\mathbf{s},\mathbf{a})$ be the gradient of the j-th Q-function with respect to the behavior action $\mathbf{a}$ and assume the gradient is normalized for simplicity. If the gradients of the Q-functions are well-aligned as illustrated in Figure 3a, then there exists a unit vector $\mathbf{w}$ such that the Q-values for the OOD actions along the direction of $\mathbf{w}$ have a low variance. To show this, we first assume the Q-value predictions for the in-distribution state-action pairs coincide, *i.e.*, $Q_{\phi_j}(\mathbf{s},\mathbf{a}) = Q(\mathbf{s},\mathbf{a})$ for $j = 1,\dots,N$. Note that this can be optimized by minimizing the Bellman error. Then, using the first-order Taylor approximation, the sample variance of the Q-values at an OOD action along $\mathbf{w}$ can be represented as

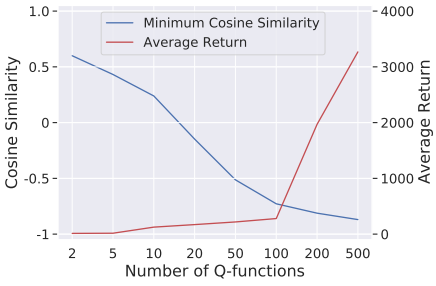


Figure 4: Plot of the minimum cosine similarity between the input gradients of Q-functions and the average return while varying the number of Q-functions.

$$\begin{aligned}\mathrm{Var}\left(Q_{\phi_j}(\mathbf{s},\mathbf{a}+k\mathbf{w})\right) &\approx \mathrm{Var}\left(Q_{\phi_j}(\mathbf{s},\mathbf{a}) + k\left\langle\mathbf{w}, \nabla_\mathbf{a}Q_{\phi_j}(\mathbf{s},\mathbf{a})\right\rangle\right)\\ &= \mathrm{Var}\left(Q(\mathbf{s},\mathbf{a}) + k\left\langle\mathbf{w}, \nabla_\mathbf{a}Q_{\phi_j}(\mathbf{s},\mathbf{a})\right\rangle\right)\\ &= k^2\mathrm{Var}\left(\left\langle\mathbf{w}, \nabla_\mathbf{a}Q_{\phi_j}(\mathbf{s},\mathbf{a})\right\rangle\right)\\ &= k^2\mathbf{w}^\intercal\mathrm{Var}\left(\nabla_\mathbf{a}Q_{\phi_j}(\mathbf{s},\mathbf{a})\right)\mathbf{w},\end{aligned}$$

where $\langle\cdot,\cdot\rangle$ denotes an inner-product, $k \in \mathbb{R}$, and $\mathrm{Var}\left(\nabla_\mathbf{a}Q_{\phi_j}(\mathbf{s},\mathbf{a})\right)$ is the sample variance matrix for the input gradients $\nabla_\mathbf{a}Q_{\phi_j}(\mathbf{s},\mathbf{a})$. One interesting property of the variance matrix is that its total variance, which is equivalent to the sum of its eigenvalues, can be represented as a function of the norm of the average gradients by Lemma 1.

Lemma 1. *The total variance of the matrix* $\mathrm{Var}\left(\nabla_\mathbf{a}Q_{\phi_j}(\mathbf{s},\mathbf{a})\right)$ *is equal to* $1 - \|\bar{q}\|_2^2$*, where* $\bar{q} = \frac{1}{N}\sum_{j=1}^{N}\nabla_\mathbf{a}Q_{\phi_j}(\mathbf{s},\mathbf{a})$.

Let $\lambda_{\min}$ be the smallest eigenvalue of $\mathrm{Var}\left(\nabla_\mathbf{a}Q_{\phi_j}(\mathbf{s},\mathbf{a})\right)$ and $\mathbf{w}_{\min}$ be the corresponding normalized eigenvector. Also, let $\epsilon > 0$ be the value such that $\min_{i\neq j}\left\langle\nabla_\mathbf{a}Q_{\phi_i}(\mathbf{s},\mathbf{a}), \nabla_\mathbf{a}Q_{\phi_j}(\mathbf{s},\mathbf{a})\right\rangle = 1 - \epsilon$. Then, using Lemma 1, we can prove that the variance of the Q-values for an OOD action along $\mathbf{w}_{\min}$ is upper-bounded by some constant multiple of ϵ, which is given by Proposition 1.

Proposition 1. *Suppose* $Q_{\phi_j}(\mathbf{s},\mathbf{a}) = Q(\mathbf{s},\mathbf{a})$ *and* $Q_{\phi_j}(\mathbf{s},\cdot)$ *is locally linear in the neighborhood of* $\mathbf{a}$ *for all* $j \in [N]$*. Let* $\lambda_{\min}$ *and* $\mathbf{w}_{\min}$ *be the smallest eigenvalue and the corresponding normalized eigenvector of the matrix* $\mathrm{Var}\left(\nabla_\mathbf{a}Q_{\phi_j}(\mathbf{s},\mathbf{a})\right)$ *and* $\epsilon > 0$ *be the value such that*

$\min_{i \neq j} \left\langle \nabla_{\mathbf{a}} Q_{\phi_i}(\mathbf{s}, \mathbf{a}), \nabla_{\mathbf{a}} Q_{\phi_j}(\mathbf{s}, \mathbf{a}) \right\rangle = 1 - \epsilon$. *Then, the variance of the Q-values for an OOD action in the neighborhood along the direction of* $\mathbf{w}_{\min}$ *is upper-bounded as follows:*

$$\mathrm{Var}\left(Q_{\phi_j}(\mathbf{s}, \mathbf{a} + k\mathbf{w}_{\min})\right) \leq \frac{1}{|\mathcal{A}|} \frac{N-1}{N} k^2 \epsilon,$$

where $|\mathcal{A}|$ *is the action space dimension.*

We provide the proofs in Appendix A.1. Proposition 1 implies that if there exists such $\epsilon > 0$ that is small, which means the gradients of Q-function are well-aligned, then the variance of the Q-values for an OOD action along a specific direction will also be small. This in turn degrades the ability of the ensembles to penalize OOD actions, which ultimately leads to requiring a large number of ensemble networks.

To address this problem, we propose a regularizer that effectively increases the variance of the Q-values for near-distribution OOD actions. Note that the variance is lower-bounded by some constant multiple of the smallest eigenvalue $\lambda_{\min}$:

$$\begin{aligned}
\mathrm{Var}\left(Q_{\phi_j}(\mathbf{s}, \mathbf{a} + k\mathbf{w})\right) &\approx k^2 \mathbf{w}^\intercal \mathrm{Var}\left(\nabla_{\mathbf{a}} Q_{\phi_j}(\mathbf{s}, \mathbf{a})\right) \mathbf{w} \\
&\geq k^2 \mathbf{w}_{\min}^\intercal \mathrm{Var}\left(\nabla_{\mathbf{a}} Q_{\phi_j}(\mathbf{s}, \mathbf{a})\right) \mathbf{w}_{\min} \\
&= k^2 \lambda_{\min}.
\end{aligned}$$

Therefore, an obvious way to increase this variance is to maximize the smallest eigenvalue of $\mathrm{Var}\left(\nabla_{\mathbf{a}} Q_{\phi_j}(\mathbf{s}, \mathbf{a})\right)$, which can be formulated as

$$\underset{\phi}{\text{maximize}} \ \mathbb{E}_{\mathbf{s}, \mathbf{a} \sim \mathcal{D}} \left[\lambda_{\min} \left(\mathrm{Var}\left(\nabla_{\mathbf{a}} Q_{\phi_j}(\mathbf{s}, \mathbf{a}) \right) \right) \right],$$

where ϕ denotes the collection of the parameters $\{\phi_j\}_{j=1}^N$. There are several methods to compute the smallest eigenvalue, such as the power method or the QR algorithm [27]. However, these iterative methods require constructing huge computation graphs, which makes optimizing the eigenvalue using back-propagation inefficient. Instead, we aim to maximize the sum of all eigenvalues, which is equal to the total variance. By Lemma 1, it is equivalent to minimizing the norm of the average gradients:

$$\underset{\phi}{\text{minimize}} \ \mathbb{E}_{\mathbf{s}, \mathbf{a} \sim \mathcal{D}} \left[\left\langle \frac{1}{N} \sum_{i=1}^{N} \nabla_{\mathbf{a}} Q_{\phi_i}(\mathbf{s}, \mathbf{a}), \frac{1}{N} \sum_{j=1}^{N} \nabla_{\mathbf{a}} Q_{\phi_j}(\mathbf{s}, \mathbf{a}) \right\rangle \right]. \tag{4}$$

With simple modification, we can reformulate Equation (4) as diversifying the gradients of each Q-function network for in-distribution actions:

$$\underset{\phi}{\text{minimize}} \ J_{\mathrm{ES}}(Q_\phi) := \mathbb{E}_{\mathbf{s}, \mathbf{a} \sim \mathcal{D}} \left[\frac{1}{N-1} \sum_{1 \leq i \neq j \leq N} \underbrace{\left\langle \nabla_{\mathbf{a}} Q_{\phi_i}(\mathbf{s}, \mathbf{a}), \nabla_{\mathbf{a}} Q_{\phi_j}(\mathbf{s}, \mathbf{a}) \right\rangle}_{\mathrm{ES}_{\phi_i, \phi_j}(\mathbf{s}, \mathbf{a})} \right].$$

Concretely, our final objective can be interpreted as measuring the pairwise alignment of the gradients using cosine similarity, which we denote as the Ensemble Similarity (ES) metric $\mathrm{ES}_{\phi_i, \phi_j}(\mathbf{s}, \mathbf{a})$, and minimizing the ES values for every pair in the Q-ensemble with regard to the dataset state-actions. The illustration of the ensemble gradient diversification is shown in Figure 3b. Note that we instead maximize the total variance to reduce the computational burden. Nevertheless, the modified objective is closely related to maximizing the smallest eigenvalue. The detailed explanation can be found in Appendix A.2.

We name the resulting actor-critic algorithm as Ensemble-Diversified Actor Critic (EDAC) and present the detailed procedure in Algorithm 1 (differences with the original SAC algorithm marked in blue). Note that Algorithm 1 reduces to SAC-N when $\eta = 0$, and further reduces to vanilla SAC when also $N = 2$.

5 Experiments

We evaluate our proposed methods against the previous offline RL algorithms on the standard D4RL benchmark [9] . Concretely, we perform our evaluation on MuJoCo Gym (Section 5.1) and Adroit

Algorithm 1 Ensemble-Diversified Actor Critic (EDAC)

1: Initialize policy parameters θ, Q-function parameters $\{\phi_j\}_{j=1}^N$, target Q-function parameters $\{\phi'_j\}_{j=1}^N$, and offline data replay buffer $\mathcal{D}$
2: **repeat**
3: Sample a mini-batch $B = \{(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')\}$ from $\mathcal{D}$
4: Compute target Q-values (shared by all Q-functions):

$$y(r, \mathbf{s}') = r + \gamma \left(\min_{j=1,\dots,N} Q_{\phi'_j}(\mathbf{s}', \mathbf{a}') - \beta \log \pi_\theta(\mathbf{a}' \mid \mathbf{s}') \right), \quad \mathbf{a}' \sim \pi_\theta(\cdot \mid \mathbf{s}')$$

5: Update each Q-function Q_{ϕ_i} with gradient descent using

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(\mathbf{s},\mathbf{a},r,\mathbf{s}') \in B} \left(\left(Q_{\phi_i}(\mathbf{s}, \mathbf{a}) - y(r, \mathbf{s}') \right)^2 + \frac{\eta}{N-1} \sum_{1 \le i \ne j \le N} \mathrm{ES}_{\phi_i, \phi_j}(\mathbf{s}, \mathbf{a}) \right)$$

6: Update policy with gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{\mathbf{s} \in B} \left(\min_{j=1,\dots,N} Q_{\phi_j}(\mathbf{s}, \tilde{\mathbf{a}}_\theta(\mathbf{s})) - \beta \log \pi_\theta(\tilde{\mathbf{a}}_\theta(\mathbf{s}) \mid \mathbf{s}) \right),$$

where $\tilde{\mathbf{a}}_\theta(\mathbf{s})$ is a sample from $\pi_\theta(\cdot \mid \mathbf{s})$ which is differentiable w.r.t. θ via the reparametrization trick.
7: Update target networks with $\phi'_i \leftarrow \rho \phi'_i + (1-\rho)\phi_i$

(Section 5.2) domains. We consider the following baselines: SAC, the backbone algorithm of our method, CQL, the previous state-of-the-art on the D4RL benchmark, REM [2], an offline RL method which utilized Q-network ensemble on discrete control environments, and BC, the behavior cloning method. We evaluate each method under the normalized average return metric where the average return is scaled such that 0 and 100 each equals the performance of a random policy and an online expert policy. In addition to the performance evaluation, we compare the computational cost of each method (Section 5.3). For the implementation details of our algorithm and the baselines, please refer to Appendix B and Appendix C. Also, we provide more experiments such as comparison with more baselines, CQL with N Q-networks, and hyperparameter sensitivity from Appendix E to Appendix H.

5.1 Evaluation on D4RL MuJoCo Gym tasks

We first evaluate each method on D4RL MuJoCo Gym tasks which consist of three environments, halfcheetah, hopper, and walker2d, each with six datasets from different data-collecting policies. In detail, the considered policies are *random*: a uniform random policy, *expert*: a fully trained online expert, *medium*: a suboptimal policy with approximately 1/3 the performance of the expert, *medium-expert*: a mixture of medium and expert policies, *medium-replay*: the replay buffer of a policy trained up to the performance of the medium agent, and *full-replay*: the final replay buffer of the expert policy. Each dataset consists of 1M transitions except for medium-expert and medium-replay.

The experiment results in Table 1 show EDAC and SAC-N both outperform or are competitive with the previous state-of-the-art on all of the tasks considered. Notably, the performance gap is especially high for random, medium, and medium-replay datasets, where the performances of the previous works are relatively low. Both the proposed methods achieve average normalized scores over 80, reducing the gap with the online expert by 40% compared to CQL. While the performance of EDAC is marginally better than the performance of SAC-N, EDAC achieves this result with a much smaller Q-ensemble size. As noted in Figure 5, on hopper tasks, SAC-N requires 200 to 500 Q-networks, while EDAC requires less than 50.

Figure 6 compares the distance between the actions chosen by each method and the dataset actions. Concretely, we measure $\mathbb{E}_{(\mathbf{s},\mathbf{a}) \sim \mathcal{D}, \hat{\mathbf{a}} \sim \pi_\theta(\cdot|\mathbf{s})}[\|\hat{\mathbf{a}} - \mathbf{a}\|_2^2]$ for EDAC, SAC-N, CQL, SAC-2, and a random policy on $*$-medium datasets. We find that our proposed methods choose from a more diverse range of actions compared to CQL. This shows the advantage of the uncertainty-based penalization which considers the prediction confidence other than penalizing all OOD actions.

Table 1: Normalized average returns on D4RL Gym tasks, averaged over 4 random seeds. CQL (Paper) denotes the results reported in the original paper.

Task Name	BC	SAC	REM	CQL (Paper)	CQL (Reproduced)	SAC-N (Ours)	EDAC (Ours)
halfcheetah-random	2.2±0.0	29.7±1.4	-0.8±1.1	**35.4**	31.3±3.5	28.0±0.9	28.4±1.0
halfcheetah-medium	43.2±0.6	55.2±27.8	-0.8±1.3	44.4	46.9±0.4	**67.5±1.2**	**65.9±0.6**
halfcheetah-expert	91.8±1.5	-0.8±1.8	4.1±5.7	104.8	97.3±1.1	**105.2±2.6**	**106.8±3.4**
halfcheetah-medium-expert	44.0±1.6	28.4±19.4	0.7±3.7	62.4	95.0±1.4	**107.1±2.0**	**106.3±1.9**
halfcheetah-medium-replay	37.6±2.1	0.8±1.0	6.6±11.0	46.2	45.3±0.3	**63.9±0.8**	**61.3±1.9**
halfcheetah-full-replay	62.9±0.8	**86.8±1.0**	27.8±35.4	-	76.9±0.9	84.5±1.2	84.6±0.9
hopper-random	3.7±0.6	9.9±1.5	3.4±2.2	10.8	5.3±0.6	**31.3±0.0**	**25.3±10.4**
hopper-medium	54.1±3.8	0.8±0.0	0.7±0.0	86.6	61.9±6.4	**100.3±0.3**	**101.6±0.6**
hopper-expert	107.7±9.7	0.7±0.0	0.8±0.0	109.9	106.5±9.1	**110.3±0.3**	**110.1±0.1**
hopper-medium-expert	53.9±4.7	0.7±0.0	0.8±0.0	**111.0**	96.9±15.1	110.1±0.3	110.7±0.1
hopper-medium-replay	16.6±4.8	7.4±0.5	27.5±15.2	48.6	86.3±7.3	**101.8±0.5**	**101.0±0.5**
hopper-full-replay	19.9±12.9	41.1±17.9	19.7±24.6	-	101.9±0.6	**102.9±0.3**	**105.4±0.7**
walker2d-random	1.3±0.1	0.9±0.8	6.9±8.3	7.0	5.4±1.7	**21.7±0.0**	**16.6±7.0**
walker2d-medium	70.9±11.0	-0.3±0.2	0.2±0.7	74.5	79.5±3.2	**87.9±0.2**	**92.5±0.8**
walker2d-expert	108.7±0.2	0.7±0.3	1.0±2.3	**121.6**	109.3±0.1	107.4±2.4	115.1±1.9
walker2d-medium-expert	90.1±13.2	1.9±3.9	-0.1±0.0	98.7	109.1±0.2	**116.7±0.4**	**114.7±0.9**
walker2d-medium-replay	20.3±9.8	-0.4±0.3	12.5±6.2	32.6	76.8±10.0	**78.7±0.7**	**87.1±2.3**
walker2d-full-replay	68.8±17.7	27.9±47.3	-0.2±0.3	-	94.2±1.9	**94.6±0.5**	**99.8±0.7**
Average	49.9	16.2	6.2	-	73.7	**84.5**	**85.2**

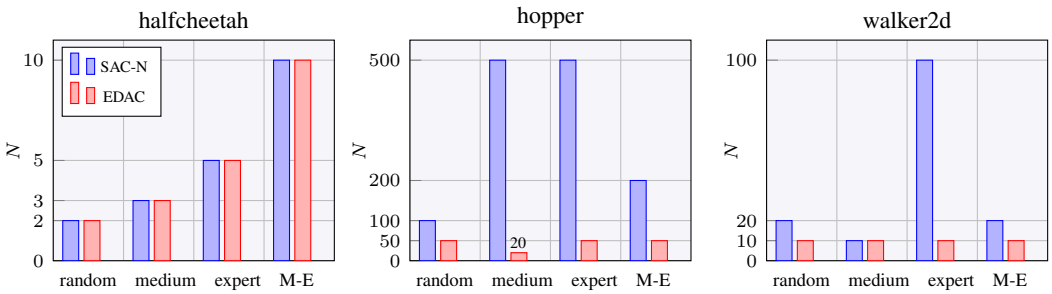


Figure 5: Minimum number of Q-ensembles (N) required to achieve the performance reported in Table 1. M-E denotes medium-expert. We omit the results of medium-replay and full-replay as SAC-N already works well with a small number of ensembles (less than or equal to 5). For more details of the experiment, please refer to Appendix C.

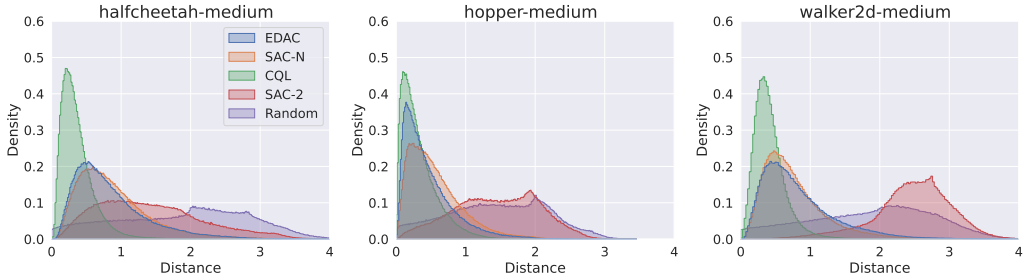


Figure 6: Histograms of the distances between the actions from each methods (EDAC, SAC-N, CQL, SAC-2, and a random policy) and the actions from the dataset. For more details of the experiment, please refer to Appendix C.

5.2 Evaluation on D4RL Adroit tasks

We also experiment on the more complex D4RL Adroit tasks that require controlling a 24-DoF robotic hand to perform tasks such as aligning a pen, hammering a nail, opening a door, or relocating a ball.

We use two types of datasets for each environment: *human*, containing 25 trajectories of human demonstrations, and *cloned*, a 50-50 mixture between the demonstration data and the behavioral cloned policy on the demonstrations. Note that for the Adroit tasks, we could not reproduce the CQL results from the paper completely. For the detailed procedure of reproducing the results of CQL, please refer to Appendix D.

Table 2: Normalized average returns on D4RL Adroit tasks, averaged over 4 random seeds.

Task Name	**BC**	**SAC**	**REM**	**CQL (Paper)**	**CQL (Reproduced)**	**SAC-N (Ours)**	**EDAC (Ours)**
pen-human	25.8±8.8	4.3±3.8	5.4±4.3	**55.8**	35.2±6.6	9.5±1.1	52.1±8.6
hammer-human	3.1±3.2	0.2±0.0	0.3±0.0	2.1	0.6±0.5	0.3±0.0	0.8±0.4
door-human	2.8±0.7	-0.3±0.0	-0.3±0.0	9.1	1.2±1.8	-0.3±0.0	**10.7±6.8**
relocate-human	0.0±0.0	-0.3±0.0	-0.3±0.0	0.35	0.0±0.0	-0.1±0.1	0.1±0.1
pen-cloned	38.3±11.9	-0.8±3.2	-1.0±0.1	40.3	27.2±11.3	**64.1±8.7**	**68.2±7.3**
hammer-cloned	0.7±0.3	0.1±0.1	-0.3±0.0	5.7	1.4±2.1	0.2±0.2	0.3±0.0
door-cloned	0.0±0.0	-0.3±0.1	-0.3±0.0	3.5	2.4±2.4	-0.3±0.0	**9.6±8.3**
relocate-cloned	0.1±0.0	-0.1±0.1	-0.2±0.2	-0.1	0.0±0.0	0.0±0.0	0.0±0.0

The evaluation results are summarized in Table 2. For pen-$*$ tasks, where the considered algorithms achieve meaningful performance, EDAC outperforms or matches with the previous state-of-the-art. Especially, for pen-cloned, both EDAC and SAC-N achieve 75% higher score compared to CQL. Unlike the results from the Gym tasks, we find that SAC-N falls behind in some datasets, for example, pen-human, which could in part due to the size of the dataset being exceptionally small (5000 transitions). However, our method with ensemble diversification successfully overcomes this difficulty.

5.3 Computational cost comparison

We compared the computational cost of our methods with vanilla SAC and CQL on hopper-medium-v2, where our methods require the largest number of Q-networks. For each method, we measure the runtime per training epoch (1000 gradient steps) along with GPU memory consumption. We run our experiments on a single machine with one RTX 3090 GPU and provide the results in Table 3.

Table 3: Computational costs of each method.

	Runtime (s/epoch)	**GPU Mem.** (GB)
SAC	21.4	1.3
CQL	38.2	1.4
SAC-500	44.1	5.1
EDAC	30.8	1.8

As the result shows, our method EDAC runs faster than CQL with comparable memory consumption. Note that CQL is about twice as slower than vanilla SAC due to the additional computations for Q-value regularization (e.g., dual update and approximate logsumexp via sampling). Meanwhile, the inference to the Q-network ensemble in SAC-N and EDAC is embarrassingly parallelizable, minimizing the runtime increase with the number of Q-networks. Also, we emphasize that our gradient diversification term in Equation (4) has linear computational complexity, as we can reformulate the term using the sum of the gradients.

6 Related Works

Model-free offline RL A popular approach for offline RL is to regularize the learned policy to be close to the behavior policy where the offline dataset was collected. BCQ [11] uses a generative model to produce actions with high similarity to the dataset and trains a restricted policy to choose the best action from the neighborhood of the generated actions. Another line of work, such as BEAR [15] or BRAC [28], stabilizes policy learning by penalizing the divergence from the dataset measured by KL divergence or MMD. While these policy-constraint methods demonstrate high performance on datasets from expert behavior policies, they fail to find optimal policies from datasets with suboptimal policies due to the strict policy constraints [9]. Also, these methods require an accurate estimation of the behavior policy, which might be difficult in complex settings with multiple behavior sources or high-dimensional environments. To address these issues, CQL [16] directly regularizes Q-functions

by introducing a term that minimizes the Q-values for out-of-distribution actions and maximizes the Q-values for in-distribution actions. Without such explicit regularizations, REM [2] proposes to use a random convex combination of Q-network ensembles on environments with discrete action spaces [4].

Estimation bias in Q-learning While Q-learning is one of the most popular algorithms in reinforcement learning, it suffers from overestimation bias due to the maximum operation $\max_{\mathbf{a}' \in \mathcal{A}} Q(\mathbf{s}', \mathbf{a}')$ used during Q-function updates [10, 25]. This overestimation bias, together with the bootstrapping, can lead to a catastrophic build-up of errors during the Q-learning process. To resolve this issue, TD3 [10] introduces a clipped version of Double Q-learning [25] that takes the minimum value of two critics. Subsequently, Maxmin Q-learning [17] theoretically shows that the overestimation bias can be controlled by the number of ensembles in the clipped Q-learning. The overestimation problem in Q-learning can be exacerbated in the offline setting since the extrapolation error cannot be corrected with further interactions with the environment, and existing offline RL algorithms handle the bias by introducing constrained policy optimization [11, 15] or conservative Q-learning frameworks [16].

Uncertainty measures in RL Uncertainty estimates have been widely used in RL for various purposes including exploration, Q-learning, and planning. Bootstrapped DQN [21] leverages an ensemble of Q-functions to quantify the uncertainty of the Q-value, and utilizes it for efficient exploration. Following this work, the UCB exploration algorithm [5] constructs an upper confidence bound [3] of the Q-values using the empirical mean and standard deviation of Q-ensembles, which is used to promote efficient exploration by applying the principle of optimism in the face of uncertainty [7]. Osband et al. [22] proposes a randomly initialized Q-ensemble that reflects the concept of prior functions in Bayesian inference and Abbas et al. [1] introduces an uncertainty incorporated planning with imperfect models. The notion of uncertainty has also been considered in offline RL, mostly in the framework of model-based offline RL. Especially, MOPO [30] and MOReL [13] measure the uncertainty of the model's prediction to formulate an uncertainty-penalized policy optimization problem in the offline RL setting. These methods introduce an ensemble of dynamics models for the quantification of the uncertainty, whereas our work adopts an ensemble of Q-functions for uncertainty-aware Q-learning.

7 Conclusion

We have shown that clipped Q-learning can be efficiently leveraged to construct an uncertainty-based offline RL method that outperforms previous methods on various datasets. Based on this observation, we proposed Ensemble-Diversifying Actor-Critic (EDAC) that effectively reduces the required number of ensemble networks for quantifying and penalizing the epistemic uncertainty. Our method does not require any explicit estimation of the data collecting policy or sampling from the out-of-distribution data and respects the epistemic uncertainty of each data point during penalization. EDAC, while requiring up to 90% less number of ensemble networks compared to the vanilla Q-ensemble, exhibits state-of-the-art performance on various datasets.

Acknowledgements

This work was supported in part by Samsung Advanced Institute of Technology, Samsung Electronics Co., Ltd., Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2020-0-00882, (SW STAR LAB) Development of deployable learning intelligence via self-sustainable and trustworthy machine learning and No. 2019-0-01371, Development of brain-inspired AI with human-like intelligence), and Research Resettlement Fund for the new faculty of Seoul National University. This material is based upon work supported by the Air Force Office of Scientific Research under award number FA2386-20-1-4043.

References

[1] Zaheer Abbas, Samuel Sokota, Erin Talvitie, and Martha White. Selective dyna-style planning under limited model capacity. In *ICML*, 2020.

[2] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *ICML*, 2020.

[3] Jean-Yves Audibert, Rémi Munos, and Csaba Szepesvári. Exploration–exploitation tradeoff using variance estimates in multi-armed bandits. *Theoretical Computer Science*, 410(19): 1876–1902, 2009.

[4] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[5] Richard Y Chen, Szymon Sidor, Pieter Abbeel, and John Schulman. Ucb exploration via q-ensembles. *arXiv preprint arXiv:1706.01502*, 2017.

[6] Xinshi Chen, Shuang Li, Hui Li, Shaohua Jiang, and Le Song. Generative adversarial user model for reinforcement learning based recommendation system. In *ICML*, 2019.

[7] Kamil Ciosek, Quan Vuong, Robert Loftin, and Katja Hofmann. Better exploration with optimistic actor-critic. In *NeurIPS*, 2019.

[8] William R Clements, Bastien Van Delft, Benoît-Marie Robaglia, Reda Bahi Slaoui, and Sébastien Toth. Estimating risk and uncertainty in deep reinforcement learning. *arXiv preprint arXiv:1905.09638*, 2019.

[9] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[10] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *ICML*, 2018.

[11] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *ICML*, 2019.

[12] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, 2018.

[13] Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. Morel: Model-based offline reinforcement learning. In *NeurIPS*, 2020.

[14] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *NeurIPS*, 2000.

[15] Aviral Kumar, Justin Fu, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. In *NeurIPS*, 2019.

[16] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In *NeurIPS*, 2020.

[17] Qingfeng Lan, Yangchen Pan, Alona Fyshe, and Martha White. Maxmin q-learning: Controlling the estimation bias of q-learning. In *ICLR*, 2020.

[18] Kimin Lee, Michael Laskin, Aravind Srinivas, and Pieter Abbeel. Sunrise: A simple unified framework for ensemble learning in deep reinforcement learning. In *ICML*, 2021.

[19] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[20] 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):529–533, 2015.

[21] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. In *NeurIPS*, 2016.

[22] Ian Osband, John Aslanides, and Albin Cassirer. Randomized prior functions for deep reinforcement learning. In *NeurIPS*, 2018.

[23] JP Royston. Expected normal order statistics(exact and approximate). *Applied Statistics*, 31(2): 161–5, 1982.

[24] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *arXiv preprint arXiv:1206.2944*, 2012.

[25] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, 2016.

[26] Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, Wojciech M Czarnecki, Andrew Dudzik, Aja Huang, Petko Georgiev, Richard Powell, et al. Alphastar: Mastering the real-time strategy game starcraft ii. *DeepMind blog*, 2, 2019.

[27] David S Watkins. Understanding the qr algorithm. *SIAM review*, 24(4):427–440, 1982.

[28] Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

[29] Yue Wu, Shuangfei Zhai, Nitish Srivastava, Joshua M. Susskind, Jian Zhang, Ruslan Salakhutdinov, and Hanlin Goh. Uncertainty weighted actor-critic for offline reinforcement learning, 2021.

[30] Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization. In *NeurIPS*, 2020.

A Ensemble gradient diversification

A.1 Proofs

Lemma 1. *The total variance of the matrix* $\mathrm{Var}\left(\nabla_{\mathbf{a}} Q_{\phi_j}(\mathbf{s}, \mathbf{a})\right)$ *is equal to* $1 - \|\bar{q}\|_2^2$*, where* $\bar{q} = \frac{1}{N}\sum_{j=1}^{N} \nabla_{\mathbf{a}} Q_{\phi_j}(\mathbf{s}, \mathbf{a})$.

Proof. For simplicity, we denote $\nabla_{\mathbf{a}} Q_{\phi_j}(\mathbf{s}, \mathbf{a})$ by q_j and their average by $\bar{q} = \frac{1}{N}\sum_j q_j$. Then, the total variance of the matrix, which is equivalent to the trace of the matrix by definition, formulates as below:

$$
\begin{aligned}
\mathrm{tr}\left(\mathrm{Var}\left(\nabla_{\mathbf{a}} Q_{\phi_j}(\mathbf{s}, \mathbf{a})\right)\right) &= \mathrm{tr}\left(\frac{1}{N}\sum_{j=1}^{N}\left(q_j - \bar{q}\right)\left(q_j - \bar{q}\right)^{\intercal}\right) \\
&= \frac{1}{N}\sum_{j=1}^{N}\mathrm{tr}\left(\left(q_j - \bar{q}\right)\left(q_j - \bar{q}\right)^{\intercal}\right) \\
&= \frac{1}{N}\sum_{j=1}^{N}\mathrm{tr}\left(\left(q_j - \bar{q}\right)^{\intercal}\left(q_j - \bar{q}\right)\right) && (\mathrm{tr}(AB) = \mathrm{tr}(BA)) \\
&= \frac{1}{N}\sum_{j=1}^{N}\left(q_j - \bar{q}\right)^{\intercal}\left(q_j - \bar{q}\right) \\
&= \frac{1}{N}\sum_{j=1}^{N}\left(q_j^{\intercal} q_j - 2 q_j^{\intercal}\bar{q} + \bar{q}^{\intercal}\bar{q}\right) \\
&= 1 - 2\left(\frac{1}{N}\sum_{j=1}^{N} q_j\right)^{\intercal}\bar{q} + \bar{q}^{\intercal}\bar{q} && (\|q_j\|_2 = 1) \\
&= 1 - \bar{q}^{\intercal}\bar{q} \\
&= 1 - \|\bar{q}\|_2^2.
\end{aligned}
$$

□

Proposition 1. *Suppose* $Q_{\phi_j}(\mathbf{s}, \mathbf{a}) = Q(\mathbf{s}, \mathbf{a})$ *and* $Q_{\phi_j}(\mathbf{s}, \cdot)$ *is locally linear in the neighborhood of* $\mathbf{a}$ *for all* $j \in [N]$*. Let* $\lambda_{\min}$ *and* $\mathbf{w}_{\min}$ *be the smallest eigenvalue and the corresponding normalized eigenvector of the matrix* $\mathrm{Var}\left(\nabla_{\mathbf{a}} Q_{\phi_j}(\mathbf{s}, \mathbf{a})\right)$ *and* $\epsilon > 0$ *be the value such that* $\min_{i \neq j}\left\langle \nabla_{\mathbf{a}} Q_{\phi_i}(\mathbf{s}, \mathbf{a}), \nabla_{\mathbf{a}} Q_{\phi_j}(\mathbf{s}, \mathbf{a})\right\rangle = 1 - \epsilon$*. Then, the variance of the Q-values for an OOD action in the neighborhood along the direction of* $\mathbf{w}_{\min}$ *is upper-bounded as follows:*

$$
\mathrm{Var}\left(Q_{\phi_j}(\mathbf{s}, \mathbf{a} + k\mathbf{w}_{\min})\right) \leq \frac{1}{|\mathcal{A}|}\frac{N-1}{N}k^2\epsilon,
$$

where $|\mathcal{A}|$ *is the action space dimension.*

Proof. We first prove that the smallest eigenvalue $\lambda_{\min}$ of $\mathrm{Var}\left(\nabla_{\mathbf{a}} Q_{\phi_j}(\mathbf{s}, \mathbf{a})\right)$ is upper-bounded by some constant multiple of ϵ. For simplicity, we denote $\nabla_{\mathbf{a}} Q_{\phi_j}(\mathbf{s}, \mathbf{a})$ by q_j and their average by

$\bar{q} = \frac{1}{N}\sum_j q_j$. We first compute the norm of the average of the gradients, which can be expressed by

$$\begin{aligned}
\|\bar{q}\|_2^2 &= \langle \bar{q}, \bar{q} \rangle \\
&= \left\langle \frac{1}{N}\sum_{i=1}^{N} q_i, \frac{1}{N}\sum_{j=1}^{N} q_j \right\rangle \\
&= \frac{1}{N^2} \sum_{1 \le i,j \le N} \langle q_i, q_j \rangle \\
&= \frac{1}{N^2} \left(\sum_{j=1}^{N} \langle q_j, q_j \rangle + \sum_{1 \le i \ne j \le N} \langle q_i, q_j \rangle \right) \\
&\ge \frac{1}{N^2} \left(N + N(N-1)(1-\epsilon) \right) \\
&= 1 - \frac{(N-1)}{N}\epsilon.
\end{aligned}$$

By Lemma 1, the total variance of the matrix is less or equal to $\frac{N-1}{N}\epsilon$. Using the fact that the total variance is equivalent to the sum of the eigenvalues and the eigenvalues of a variance matrix is non-negative, we have

$$\begin{aligned}
\lambda_{\min} &\le \frac{1}{|\mathcal{A}|} \sum_{j=1}^{|\mathcal{A}|} \lambda_j \\
&= \frac{1}{|\mathcal{A}|} \mathrm{tr}\left(\mathrm{Var}\left(\nabla_{\mathbf{a}} Q_{\phi_j}(\mathbf{s}, \mathbf{a}) \right) \right) \\
&\le \frac{1}{|\mathcal{A}|} \frac{N-1}{N} \epsilon,
\end{aligned} \tag{5}$$

where $\lambda_1, \ldots, \lambda_{|\mathcal{A}|}$ are the eigenvalues of $\mathrm{Var}\left(\nabla_{\mathbf{a}} Q_{\phi_j}(\mathbf{s}, \mathbf{a}) \right)$.

Note that, using the fact that the Q-values coincide at the action $\mathbf{a}$ and the local linearity of the Q-functions, we have derived

$$\mathrm{Var}(Q_{\phi_j}(\mathbf{s}, \mathbf{a} + k\mathbf{w})) = k^2 \mathbf{w}^\intercal \mathrm{Var}\left(\nabla_{\mathbf{a}} Q_{\phi_j}(\mathbf{s}, \mathbf{a}) \right) \mathbf{w}. \tag{6}$$

Plugging $\mathbf{w} = \mathbf{w}_{\text{min}}$ in Equation (6) and using Equation (5), we have

$$\begin{aligned}
\mathrm{Var}(Q_{\phi_j}(\mathbf{s}, \mathbf{a} + k\mathbf{w}_{\text{min}})) &= k^2 \mathbf{w}_{\text{min}}^\intercal \mathrm{Var}\left(\nabla_{\mathbf{a}} Q_{\phi_j}(\mathbf{s}, \mathbf{a}) \right) \mathbf{w}_{\text{min}} \\
&= k^2 \lambda_{\text{min}} \\
&\le \frac{1}{|\mathcal{A}|} \frac{N-1}{N} k^2 \epsilon.
\end{aligned}$$

. □

A.2 Relationship between maximizing the total variance and maximizing the smallest eigenvalue

As we have shown in Section 4, maximizing the total variance of the matrix $\mathrm{Var}\left(\nabla_{\mathbf{a}} Q_{\phi_i}(\mathbf{s}, \mathbf{a}) \right)$ is equivalent to minimizing the cosine similarity of all distinct pairs of the gradients $\nabla_{\mathbf{a}} Q_{\phi_i}(\mathbf{s}, \mathbf{a})$, which makes the gradients uniformly distributed on the unit sphere $S^{|\mathcal{A}|-1}$. Therefore, if the trace is sufficiently maximized, then we can see $\mathrm{Var}\left(\nabla_{\mathbf{a}} Q_{\phi_i}(\mathbf{s}, \mathbf{a}) \right)$ as a sample variance matrix of a uniform spherical distribution. It can be easily proved that the variance matrix of a uniform distribution on is $\frac{1}{|\mathcal{A}|} I$, whose all eigenvalues are equal to $\frac{1}{|\mathcal{A}|}$, by Proposition 2.

Proposition 2. *The variance matrix of the uniform spherical distribution $X \sim \mathcal{U}(S^{n-1})$ is $\frac{1}{n} I$.*

Proof. Let $X = (X_1, \ldots, X_n)$. Then $X_{-i} = (X_1, \ldots, -X_i, \ldots, X_n)$ is also from the uniform spherical distribution. Therefore, we have $\mathbb{E}[X_i] = \mathbb{E}[-X_i] = 0$ and $\mathbb{E}[X_i X_j] = \mathbb{E}[-X_i X_j] =$

$0,\ \forall i \neq j$. For the diagonal entries of the variance matrix, we have $\mathbb{E}[\sum_{i=1}^{n} X_i^2] = \sum_{i=1}^{n} \mathbb{E}[X_i^2] = 1$ by the definition of the spherical distribution and $\mathbb{E}[X_i^2] = \mathbb{E}[X_j^2]$ by the symmetry of the distribution. Therefore, we have $\mathbb{E}[X_i^2] = \frac{1}{n}$ and $\mathrm{Var}(X) = \frac{1}{n} I$. □

Note that the smallest eigenvalue of $\mathrm{Var}\left(\nabla_{\mathbf{a}} Q_{\phi_i}(\mathbf{s}, \mathbf{a})\right)$ is less or equal to $\frac{1}{|\mathcal{A}|}$, since the total variance is upper-bounded by 1 due to Lemma 1. Therefore, as the number of Q-ensembles goes to infinity, $\mathrm{Var}\left(\nabla_{\mathbf{a}} Q_{\phi_i}(\mathbf{s}, \mathbf{a})\right)$ converges to $\frac{1}{|\mathcal{A}|} I$, attaining the maximum value for the smallest eigenvalue.

B Implementation details

SAC We use the SAC implementation from rlkit[3]. We use its default parameters except for increasing the number of layers for both the policy network and the Q-function networks from 2 to 3, following the protocol of CQL.

REM We implement a continuous control version of REM on top of SAC by modifying the Bellman residual term to

$$\min_{\phi} \mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}' \sim \mathcal{D}, \xi \sim \mathrm{P}_{\Delta}} \left[\left(\sum_{j=1}^{N} \xi_j Q_{\phi_j}(\mathbf{s}, \mathbf{a}) - \left(r(\mathbf{s}, \mathbf{a}) + \gamma \, \mathbb{E}_{\mathbf{a}' \sim \pi_\theta(\cdot|\mathbf{s}')} \left[\sum_{j=1}^{N} \xi_j Q_{\phi'_j}\left(\mathbf{s}', \mathbf{a}'\right) \right] \right) \right)^2 \right],$$

where P_{Δ} represents a probability distribution over the standard $(N-1)$-simplex $\Delta^{N-1} = \{\xi \in \mathbb{R}^N : \xi_1 + \xi_2 + \cdots + \xi_N = 1,\ \xi_n \geq 0,\ n = 1, \ldots, N\}$. Following the original REM paper, we use a simple probability distribution: $\xi_n = \xi'_n / \sum_k \xi'_k$, where $\xi'_k \sim U(0, 1)$ for $k = 1, \ldots, N$. For a fair comparison with our ensemble algorithms, we sweep the ensemble size N within $\{2, 5, 10, 20, 50, 100, 200, 500, 1000\}$ and report the best number.

CQL We use the official implementation by the authors[4]. For MuJoCo Gym tasks, the recommended hyperparameters in the codebase differ from the original paper due to the updates in the D4RL datasets. We tried both versions of hyperparameter settings and found the codebase version outperforms the paper version while matching the numbers in the paper reasonably well. Therefore, we follow the guidelines from the official code and use the fixed α version, searching for the parameters within $\alpha \in \{5, 10\}$ and policy learning rate $\in \{1e-4, 3e-4\}$. We chose $\alpha = 10.0$ with policy learning rate $= 1e-4$ as the default as it gives the best results in most of the datasets. However, we use the dual gradient descent version with $\tau = 10.0$ and policy learning rate$= 1e-4$ on some datasets, such as halfcheetah-random, since the fixed α version could not reproduce the results from the paper on those datasets. For the Adroit tasks, the codebase does not provide separate guidelines, and we use the hyperparameters listed in the paper.

SAC-N (Ours) We keep the default setting from the SAC experiments other than the ensemble size N. On halfcheetah and walker2d environments, we tune N in the range of $\{5, 10, 20\}$ except for walker2d-expert, which requires up to $N = 100$. For hopper, we tune within $N \in \{100, 200, 500, 1000\}$. The hyperparameters selected are listed in Table 4. As we noted in Figure 5, some datasets can be dealt with less N (*e.g.*, $*$-replay). However, we tried to keep the hyperparameters within an environment consistent in order to reduce hyperparameter sensitivity. For Adroit tasks, we sweep N in the range of $\{20, 50, 100, 200\}$ and report the selected N in Table 5.

EDAC (Ours) For Mujoco Gym tasks, we tune the ensemble size N within the range of $\{10, 20, 50\}$ and the weight of the ensemble gradient diversity term η within $\{0.0, 1.0, 5.0\}$. Note that we use the same N on each environment. For Adroit tasks, we sweep the parameters on $N \in \{20, 50, 100\}$ and $\eta \in \{100, 200, 500, 1000\}$ except for pen-cloned, which uses $\eta = 10.0$. While we can also achieve competitive performance on pen-cloned with larger η, we found lower η helps to mitigate the performance degradation on further training steps. The selected hyperparameters on each environment are listed in Table 4 and Table 5, respectively.

[3] `https://github.com/vitchyr/rlkit`

[4] `https://github.com/aviralkumar2907/CQL`

Table 4: Hyperparameters used in the D4RL MuJoCo Gym experiments.

Task Name	**SAC-N** (N)	**EDAC** (N, η)
halfcheetah-random	10	10, 0.0
halfcheetah-medium	10	10, 1.0
halfcheetah-expert	10	10, 1.0
halfcheetah-medium-expert	10	10, 5.0
halfcheetah-medium-replay	10	10, 1.0
halfcheetah-full-replay	10	10, 1.0
hopper-random	500	50, 0.0
hopper-medium	500	50, 1.0
hopper-expert	500	50, 1.0
hopper-medium-expert	200	50, 1.0
hopper-medium-replay	200	50, 1.0
hopper-full-replay	200	50, 1.0
walker2d-random	20	10, 1.0
walker2d-medium	20	10, 1.0
walker2d-expert	100	10, 5.0
walker2d-medium-expert	20	10, 5.0
walker2d-medium-replay	20	10, 1.0
walker2d-full-replay	20	10, 1.0

Table 5: Hyperparameters used in the D4RL Adroit experiments.

Task Name	**SAC-N** (N)	**EDAC** (N, η)
pen-human	100	20, 1000.0
pen-cloned	100	20, 10.0
hammer-human	100	50, 200.0
hammer-cloned	100	50, 200.0
door-human	100	50, 200.0
door-cloned	100	50, 200.0
relocate-human	100	50, 200.0
relocate-cloned	100	50, 200.0

C Experimental settings

MuJoCo Gym We use the v2 version of each dataset (*e.g.*, halfcheetah-random-v2) which fixes some of the bugs from the previous versions. We run each algorithm for 3 million training steps and report the normalized average return of each policy. While the CQL paper originally used 1 million steps, we found increasing this to 3 million helps the algorithms to converge on more complex datasets such as $*$-medium-expert.

Adroit We use the v1 version of each dataset. On these datasets, we adopt max Q backup from CQL and normalize the rewards for training stability. As we will discuss in Appendix D, the performance of the baseline algorithm CQL degrades after some steps of training. Therefore, for a fair comparison, we run each algorithm for 200,000 steps and report the normalized average return.

Minimum required Q-ensembles (Figure 5) To check the minimum required number of Q-ensembles for each dataset, we sweep N within the range of $\{2, 3, 5, 10, 20, 50, 100, 200, 500, 1000\}$ and report the minimum N that achieves the performance similar to Table 1. We find EDAC successes to reduce the required N significantly when the original requirement is high (*e.g.*, hopper, walker2d-expert).

Action distance histograms (Figure 6) To draw the histogram, we sample 500,000 random $(\mathbf{s}, \mathbf{a})$ pairs from each dataset and measure the ℓ_2 distance between the action sampled from each policy after full training and the dataset action.

D Reproducing CQL in Adroit

Since the pen-$*$ tasks are where the considered algorithms show meaningful performance, we focused on reproducing the reported results for those tasks. After running CQL with the parameters given in the original paper, we found that the performance of CQL degrades after about 200,000 steps, as shown in Figure 7. While we are not sure of the cause of this performance gap, it could be due to the difference in the `min_q_weight` parameter setting, which was not specified in the original paper, or a minor modification we applied to the code to fix the backpropagation issue[5]. Meanwhile, for a fair comparison, on Adroit we chose to use early-stopping and train each algorithm for 200,000 steps. Also, we include the reported CQL numbers for all experiments.

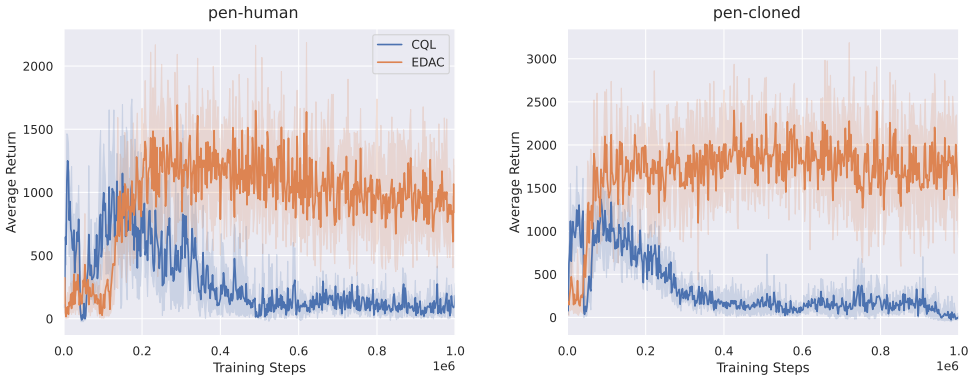


Figure 7: Performance of EDAC and CQL on pen-$*$ datasets. 'Average Return' denotes the undiscounted return of each policy on evaluation. Results averaged over 4 seeds.

E Comparison with more baselines

We additionally compared our methods with more baselines on D4RL Gym datasets. First, we add comparisons with some of the well-known offline RL methods, BCQ [11], BEAR [15], BRAC [28], and MOReL [13]. Also, we include the results of UWAC [29], a concurrent work that also utilizes Q-value uncertainty. We reproduced all the methods by following the hyperparameter search procedure listed in each paper and selected the best results. We report the normalized average return results in Table 6.

Table 6: Extended results of normalized average returns on D4RL Gym tasks, averaged over 4 random seeds. CQL (Paper) denotes the results reported in the original paper.

Task Name	BC	SAC	REM	BCQ	BEAR	BRAC	MOReL	UWAC	CQL (Paper)	CQL (Reproduced)	SAC-N (Ours)	EDAC (Ours)
halfcheetah-random	2.2±0.0	29.7±1.4	-0.8±1.1	2.2±0.0	12.6±1.0	24.3±0.7	**38.9±1.8**	2.3±0.0	35.4	31.3±3.5	28.0±0.9	28.4±1.0
halfcheetah-medium	43.2±0.6	55.2±27.8	-0.8±1.3	46.6±0.4	42.8±0.1	51.9±0.3	60.7±4.4	43.7±0.4	44.4	46.9±0.4	**67.5±1.2**	**65.9±0.6**
halfcheetah-expert	91.8±1.5	-0.8±1.8	4.1±5.7	89.9±9.6	92.6±0.6	39.0±13.8	8.4±11.8	94.7±1.1	104.8	97.3±1.1	**105.2±2.6**	**106.8±3.4**
halfcheetah-medium-expert	44.0±1.6	28.4±19.4	0.7±3.7	95.4±2.0	45.7±4.2	52.3±0.1	80.4±11.7	47.0±6.0	62.4	95.0±1.4	**107.1±2.0**	**106.3±1.9**
halfcheetah-medium-replay	37.6±2.1	0.8±1.0	6.6±11.0	42.2±0.9	39.4±0.8	48.6±0.4	44.5±5.6	38.9±1.1	46.2	45.3±0.3	**63.9±0.8**	**61.3±1.9**
halfcheetah-full-replay	62.9±0.8	**86.8±1.0**	27.8±35.4	69.5±4.0	60.1±3.2	78.0±0.7	70.1±5.1	65.1±0.5	-	76.9±0.9	84.5±1.2	84.6±0.9
hopper-random	3.7±0.6	9.9±1.5	3.4±2.2	7.8±0.6	3.6±3.6	8.1±0.6	**38.1±10.1**	2.6±0.3	10.8	5.3±0.6	31.3±0.0	25.3±10.4
hopper-medium	54.1±3.8	0.8±0.0	0.7±0.0	59.4±8.3	55.3±3.2	77.8±6.1	84.0±17.0	52.6±4.0	86.6	61.9±6.4	**100.3±0.3**	**101.6±0.6**
hopper-expert	107.7±9.7	0.7±0.0	0.8±0.0	109±4.0	39.4±20.5	78.1±52.3	80.4±34.9	**111.0±0.8**	109.9	106.5±9.1	110.3±0.3	110.1±0.1
hopper-medium-expert	53.9±4.7	0.7±0.0	0.8±0.0	106.9±5.0	66.2±8.5	81.3±8.0	105.6±8.2	54.8±3.2	**111.0**	96.9±15.1	110.1±0.3	110.7±0.1
hopper-medium-replay	16.6±4.8	7.4±0.5	27.5±15.2	60.9±14.7	57.7±16.5	62.7±30.4	81.8±17.0	31.1±14.8	48.6	86.3±7.3	**101.8±0.5**	**101.0±0.5**
hopper-full-replay	19.9±12.9	41.1±17.9	19.7±24.6	46.6±13.0	54.0±24.0	**107.4±0.5**	94.4±20.5	21.9±8.4	-	101.9±0.6	102.9±0.3	105.4±0.7
walker2d-random	1.3±0.1	0.9±0.8	6.9±8.3	4.9±0.1	4.3±1.2	1.3±1.4	16.0±7.7	1.5±0.3	7.0	5.4±1.7	**21.7±0.0**	**16.6±7.0**
walker2d-medium	70.9±11.0	-0.3±0.2	0.2±0.7	71.8±7.2	59.8±40.0	59.7±39.9	72.8±11.9	66.0±9.0	74.5	79.5±3.2	**87.9±0.2**	**92.5±0.8**
walker2d-expert	108.7±0.2	0.7±0.3	1.0±2.3	106.3±5.0	110.1±0.6	55.2±62.2	62.6±29.9	108.4±0.5	**121.6**	109.3±0.1	107.4±2.4	115.1±1.9
walker2d-medium-expert	90.1±13.2	1.9±3.9	-0.1±0.0	107.7±3.8	107.0±2.9	9.3±18.9	107.5±5.6	85.7±14.0	98.7	109.1±0.2	**116.7±0.4**	**114.7±0.9**
walker2d-medium-replay	20.3±9.8	-0.4±0.3	12.5±6.2	57.0±9.6	12.2±4.7	40.1±47.9	40.8±20.4	27.1±9.6	32.6	76.8±10.0	**78.7±0.7**	**87.1±2.3**
walker2d-full-replay	68.8±17.7	27.9±47.3	-0.2±0.3	71.0±21.8	79.6±15.6	96.9±2.2	84.8±13.1	60.7±15.6	-	94.2±1.9	**94.6±0.5**	**99.8±0.7**
Average	49.9	16.2	6.2	64.2	52.4	54.0	65.1	50.8	-	73.7	**84.5**	**85.2**

[5] `https://github.com/aviralkumar2907/CQL/issues/5`

The results show our methods outperform all the baseline methods on most of the datasets considered. Also, we reiterate that while the performance of EDAC is marginally better than SAC-N, EDAC achieves this result with a much smaller Q-ensemble size.

F CQL with N Q-networks

Since other offline RL methods may also benefit from larger N or ensemble diversification, here we evaluate CQL-N and CQL with ensemble diversification for ablation. For CQL-N, we tried $N \in \{2, 5, 10, 50, 100\}$, where $N = 2$ denotes the original version of CQL. For CQL with ensemble diversification, we added our diversification term to the CQL loss function and swept the coefficient η in the range of $\{0.5, 1.0, 5.0\}$, which is the same range used in EDAC. The normalized return evaluation results on D4RL Gym $*$-medium datasets are shown in Table 7.

Table 7: Performance of CQL with N Q-networks on D4RL Gym $*$-medium datasets.

		halfcheetah-medium	hopper-medium	walker2-medium
CQL-N	$N = 2$	46.9±0.4	61.9±6.4	79.5±3.2
	$N = 5$	47.1±0.3	61.6±6.0	80.8±4.9
	$N = 10$	45.9±0.3	60.1±4.8	70.9±0.9
	$N = 50$	44.2±0.4	54.3±2.0	69.4±0.0
	$N = 100$	43.7±0.2	43.7±0.8	71.3±3.8
CQL w/ diversification	$\eta = 0.5$	46.5±0.4	65.8±11.2	82.2±0.6
	$\eta = 1.0$	47.2±0.1	69.2±8.8	80.5±3.1
	$\eta = 5.0$	47.4±0.5	60.9±3.2	82.1±1.2
SAC-N		**67.5±1.2**	**100.3±0.3**	**87.9±0.2**
EDAC		**65.9±1.6**	**101.6±0.6**	**92.5±0.8**

We observe that even though increasing the number of Q-networks or applying gradient diversification do help CQL on some of the datasets, the improved performance still falls far behind our methods (SAC-N, EDAC).

G Comparison to variance regularization

In this section, we compare EDAC with increasing the variance of the Q-estimates for in-distribution actions, which is another possible option for ensemble diversification. Table 8 shows the average return and the Q-value estimation statistics on the walker2d-expert dataset when using the Q-estimate variance regularizer, compared to EDAC. *Var reg* adds to SAC-N a regularizing term that explicitly increases the variance of the Q-estimates, weighted by a coefficient c. *Q Avg* denotes the estimated Q-values of each model in evaluation. *Q Std* means the standard deviation of Q-estimates from a ensemble on the given actions. *Q Std gap* means the gap of standard deviations from behavior and random actions.

Table 8: Comparison of EDAC with Q-estimate variance regularization on walker2d-expert dataset. Same number of Q-networks is used for all methods.

		Return	Q Avg	Q Std (behavior action)	Q Std (random action)	Q Std gap
Var reg	$c = 50$	511	overflow	N/A	N/A	N/A
	$c = 100$	20	-95	5.3	7	1.7
	$c = 200$	368	-929	10.6	15.1	4.5
EDAC		5236	392	1.2	10.5	9.3

On the walker2d-expert dataset, adding the variance-enhancing regularizer either leads to two results: (1) Exploding Q-values when the regularization is not strong ($c = 50$) or (2) severe Q-value underestimation when the regularization is stronger ($c = 100, 200$). The reason behind these two

extreme modes is that the gap of the Q-estimate variance between behavior actions and OOD actions, which is crucial for conservative learning, increases much slower than the absolute increase of the Q-estimate variances. For example, on $c = 200$, the Q-estimate Std gap is 4.5. This gap is about half of EDAC, whereas the absolute Q-estimate Stds on both actions are much higher. In EDAC, the variance of Q-estimates on behavior actions remains small even though the OOD actions are sufficiently penalized, as we only diversify the Q-networks' gradients instead of the Q-values themselves.

H Hyperparameter sensitivity

To measure the hyperparameter sensitivity of EDAC, we sweep the weight of the gradient diversification term η in the range of {0.0, 0.5, 1.0, 2.0, 5.0} on the hopper datasets, fixing the number of Q-networks to $N = 50$, and present the results in Table 9.

Table 9: Performance of EDAC over various η on the D4RL Gym hopper datasets.

Dataset type	$\eta = 0.0$	$\eta = 0.5$	$\eta = 1.0$	$\eta = 2.0$	$\eta = 5.0$
random	25.3±10.4	9.3±6.2	6.7±0.8	3.8±1.5	1.9±0.8
medium	7.3±0.1	102.2±0,4	101.6±0.5	94.5±12.4	75.5±24.1
expert	2.3±0.1	110.3±0.2	110.1±0.1	109.8±0.2	109.9±0.2
medium-expert	46.9±33.0	103.8±12.4	110.7±0.1	109.8±0.2	109.8±0.2
medium-replay	100.9±0.4	100.3±0.8	101.0±0.4	100.2±0.5	20.6±0.7
full-replay	105.6±0.4	104.9±0.5	4105.4±0.6	104.0±0.2	106.3±0.9

The results show that except for the random dataset, there exists a large well of hyperparameters where EDAC achieves expert-level performance. We also observe that increasing η sometimes degrades the performance on random, medium, and medium-replay datasets which contain trajectories drawn from suboptimal policies. Intuitively, the gradient diversification term induces the learned policy to favor in-distribution actions over OOD actions. Therefore, increasing η can lead to a more conservative policy, which is undesirable if the behavior policy is suboptimal.