Addressing Distribution Shift in Online Reinforcement Learning with Offline Datasets

Seunghyun Lee[*,†] **Younggyo Seo**[*,†] **Kimin Lee**[‡] **Pieter Abbeel**[‡] **Jinwoo Shin**[†]
[†]Korea Advanced Institute of Science and Technology
[‡]University of California, Berkeley

Abstract

Recent progress in offline reinforcement learning (RL) has made it possible to train strong RL agents from previously-collected, static datasets. However, depending on the quality of the trained agents and the application being considered, it is often desirable to improve such offline RL agents with further online interaction. As it turns out, fine-tuning offline RL agents is a non-trivial challenge, due to *distribution shift* – the agent encounters out-of-distribution samples during online interaction, which may cause bootstrapping error in Q-learning and instability during fine-tuning. In order to address the issue, we present a simple yet effective framework, which incorporates a balanced replay scheme and an ensemble distillation scheme. First, we propose to keep separate offline and online replay buffers, and carefully balance the number of samples from each buffer during updates. By utilizing samples from a wider distribution, i.e., both online and offline samples, we stabilize the Q-learning. Next, we present an ensemble distillation scheme, where we train an ensemble of independent actor-critic agents, then distill the policies into a single policy. In turn, we improve the policy using the Q-ensemble during fine-tuning, which allows the policy updates to be more robust to error in each individual Q-function. We demonstrate the superiority of our method on MuJoCo datasets from the recently proposed D4RL benchmark suite.

1 Introduction

Offline reinforcement learning (RL), the task of training a sequential decision-making agent with a static offline dataset, holds the promise of a data-driven approach to reinforcement learning, thereby bypassing the laborious and often dangerous process of sample collection [21]. Accordingly, various offline RL methods have been developed, some of which are often capable of training agents that are more performant than the best data-generating policy [1, 6, 14, 17, 18, 27, 31, 32]. However, agents trained via offline RL methods may be suboptimal, for (a) the dataset they were trained on may only contain suboptimal data; and (b) environment in which they are deployed may be different from the environment in which dataset was generated. This necessitates an online fine-tuning procedure, where the agent improves by interacting with the environment and gathering additional information.

Fine-tuning an offline RL agent, however, poses certain challenges. For example, Nair et al. [22] pointed out that offline RL algorithms based on modeling the dataset-generating policy [6, 17, 27, 31] are not amenable to fine-tuning, due to the difficulty of modeling the dataset-generating policy in the online setup. On the other hand, a more recent state-of-the-art offline RL algorithm, conservative Q-learning (CQL) [18], does not require explicit behavior modeling, and one might expect CQL to be amenable to fine-tuning. However, we make an observation that fine-tuning a CQL agent is a non-trivial task, due to the so-called *distribution shift* problem – the agent encounters out-of-distribution

[*]Equal Contribution. Correspondence to {seunghyun.lee, younggyo.seo}@kaist.ac.kr

Offline Reinforcement Learning Workshop at Neural Information Processing Systems, 2020.

samples, and in turn loses its good initial policy from offline RL training. This can be attributed to the bootstrapping error, i.e., error introduced when Q-function is updated with an inaccurate target value evaluated at unfamiliar states and actions. Such initial training instability is a severe limitation, given the appeal of offline RL lies in safe deployment at test time, and losing such safety guarantees directly conflicts with the goal of offline RL.

Contribution. In this paper, we first demonstrate that fine-tuning a CQL agent may lead to unstable training due to distribution shift (see Section 3 for more details). To handle this issue, we introduce a simple yet effective framework, which incorporates a balanced replay scheme and an ensemble distillation scheme. Specifically, we propose to maintain two separate replay buffers for offline and online samples, respectively. Then we modulate the sampling ratio between the two, in order to balance the effects of (a) widening the data distribution the agent sees (offline data), and (b) exploiting the environment feedback (online data). Furthermore, we propose an ensemble distillation scheme: first, we learn an ensemble of independent CQL agents, then distill the multiple policies into a single policy. During fine-tuning, we improve the policy using the mean of Q-functions, so that policy updates are more robust to error in each individual Q-function.

In our experiments, we demonstrate the strength of our method based on MuJoCo [29] datasets from the D4RL [4] benchmark suite. For evaluation, we measure the final performance and sample-efficiency of RL agents throughout the fine-tuning procedure. We demonstrate that our method achieves stable training, while significantly outperforming all baseline methods considered, including BCQ [6] and AWAC [22], both in terms of final performance and sample-efficiency.

2 Background

Reinforcement learning. We consider the standard RL framework, where an agent interacts with the environment so as to maximize the expected total return. More formally, at each timestep t, the agent observes a state s_t, and performs an action a_t according to its policy π. The environment rewards the agent with r_t, then transitions to the next state s_{t+1}. The agent's objective is to maximize the expected return $\mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k r_k]$, where $\gamma \in [0, 1)$ is the discount factor.

In this work, we mainly consider off-policy RL algorithms, a class of algorithms that can, in principle, train an agent with samples generated by any behavior policy. These algorithms are well-suited for fine-tuning a pre-trained RL agent, for they can leverage both offline and online samples. Offline RL algorithms are off-policy RL algorithms that only utilize static datasets for training. Here, we introduce an off-policy RL algorithm and an offline RL algorithm we build on in this work.

Soft Actor-Critic. SAC [9] is an off-policy actor-critic algorithm that learns a soft Q-function $Q_\theta(s, a)$ parameterized by θ and a stochastic policy π_ϕ modeled as a Gaussian with its parameters ϕ. To update parameters θ and ϕ, SAC alternates between a soft policy evaluation and a soft policy improvement. During soft policy evaluation, soft Q-function parameter θ is updated to minimize the soft Bellman residual:

$$\mathcal{L}^{\texttt{SAC}}_{\texttt{critic}}(\theta) = \mathbb{E}_{\tau_t \sim \mathcal{B}}[\mathcal{L}_Q(\tau_t, \theta)], \tag{1}$$

$$\mathcal{L}^{\texttt{SAC}}_{Q}(\tau_t, \theta) = \left(Q_\theta(s_t, a_t) - (r_t + \gamma \mathbb{E}_{a_{t+1} \sim \pi_\phi}[Q_{\bar{\theta}}(s_{t+1}, a_{t+1}) - \alpha \log \pi_\phi(a_{t+1}|s_{t+1})])\right)^2, \tag{2}$$

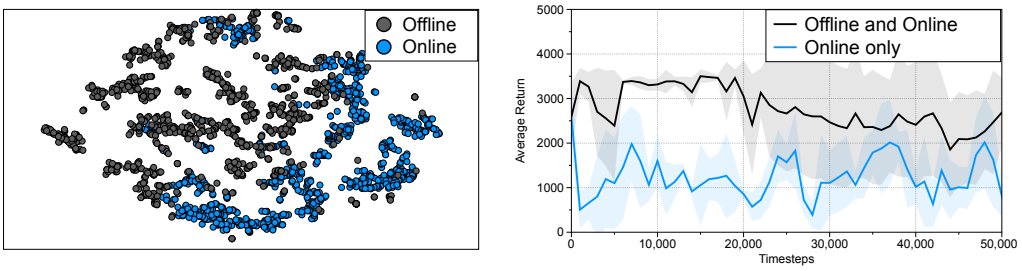
where $\tau_t = (s_t, a_t, r_t, s_{t+1})$ is a transition, $\mathcal{B}$ is the replay buffer, $\bar{\theta}$ is the moving average of soft Q-function parameter θ, and α is the temperature parameter. During soft policy improvement, policy parameter ϕ is updated to minimize the following objective:

$$\mathcal{L}^{\texttt{SAC}}_{\texttt{actor}}(\phi) = \mathbb{E}_{s_t \sim \mathcal{B}}[\mathcal{L}_\pi(s_t, \phi)], \text{ where } \mathcal{L}_\pi(s_t, \phi) = \mathbb{E}_{a_t \sim \pi_\phi}[\alpha \log \pi_\phi(a_t|s_t) - Q_\theta(s_t, a_t)]. \tag{3}$$

Conservative Q-Learning. CQL [18] is an offline RL algorithm that learns a lower bound of the Q-function $Q_\theta(s, a)$, in order to prevent extrapolation error – value overestimation caused by bootstrapping from out-of-distribution actions. To this end, CQL($\mathcal{H}$), a variant of CQL, imposes a regularization that minimizes the expected Q-value at unseen actions, and maximizes the expected Q-value at seen actions by minimizing the following objective:

$$\mathcal{L}^{\texttt{CQL}}_{\texttt{critic}}(\theta) = \frac{1}{2} \cdot \mathcal{L}^{\texttt{CQL}}_{Q}(\theta) + \alpha^{\texttt{CQL}} \cdot \mathcal{L}^{\texttt{CQL}}_{\texttt{reg}}(\theta), \tag{4}$$

$$\mathcal{L}^{\texttt{CQL}}_{Q}(\theta) = \mathbb{E}_{\tau_t \sim \mathcal{D}}\Big[\left(Q_\theta(s_t, a_t) - (r_t + \gamma \mathbb{E}_{a_{t+1} \sim \pi_\phi}[Q_{\bar{\theta}}(s_{t+1}, a_{t+1})])\right)^2\Big], \tag{5}$$



(a) t-SNE visualization (b) Walker2d-medium

Figure 1: (a) t-SNE visualization of offline samples from `walker2d-medium` dataset from the D4RL benchmark [4], and online samples collected by a CQL agent trained on the same dataset. (b) Performance on `walker2d-medium` task during fine-tuning. We first train a CQL agent via (4), and fine-tune it via (1), (3), either using both offline and online data, or using online samples only.

$$\mathcal{L}^{\texttt{CQL}}_{\texttt{reg}}(\theta) = \mathbb{E}_{s_t \sim \mathcal{D}}\left[\log \sum_a \exp(Q_\theta(s_t, a) - \mathbb{E}_{a_t \sim \widehat{\pi}_\beta}[Q_\theta(s_t, a_t)\right], \tag{6}$$

where $\alpha^{\texttt{CQL}}$ is the tradeoff parameter, $\mathcal{D}$ the offline dataset, and $\widehat{\pi}_\beta(a_t|s_t) = \frac{\sum_{(s,a)\in\mathcal{D}} \mathbb{1}\{s=s_t, a=a_t\}}{\sum_{s\in\mathcal{D}} \mathbb{1}\{s=s_t\}}$ the empirical behavior policy.

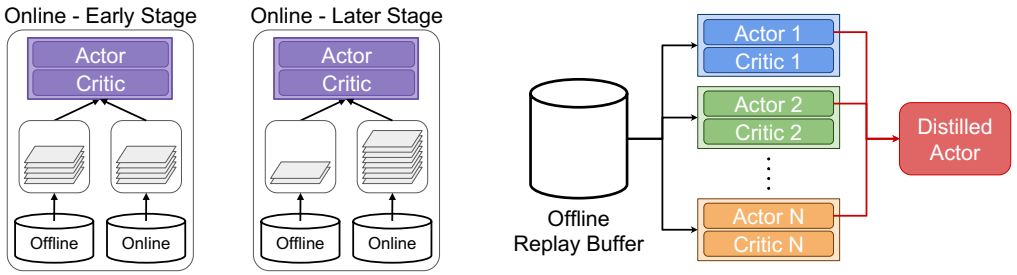
3 Challenge: Distribution shift

In the context of fine-tuning an offline RL agent, there may exist a distribution shift between offline and online data distribution: an offline RL agent encounters data distributed away from the offline data, as the agent starts interacting with the environment (see Figure 1a for a visualization). This shift in turn affects fine-tuning in a very complicated manner, for it involves an interplay between actor and critic updates with newly collected out-of-distribution samples. To elaborate in more detail, we train an agent offline via state-of-the-art CQL algorithm in (4), then fine-tune it via SAC algorithm in (1) and (3)[2], either using both offline and online samples as in Nair et al. [22], or using online samples exclusively. Figure 1b shows how fine-tuning may suffer from instability in both cases. Essentially, this instability occurs due to the shift between offline and online data distribution. In the case of using both offline and online data, the chance of agent seeing online samples for update becomes too low, especially when the offline dataset contains massive amount of data. This prevents timely updates at unfamiliar states encountered online, and may lead to performance degradation as seen in Figure 1b. On the other hand, when using online samples exclusively, the agent is exposed to unseen samples only, for which Q function does not provide a reliable value estimate. This may lead to bootstrapping error, and hence a dip in performance as seen in Figure 1b. This points to leveraging a mixture of offline and online samples for fine-tuning, so that the agent can balance the trade-off between them.

4 BRED: Balanced replay with ensemble distillation

In this section, we present BRED: **B**alanced **R**eplay with **E**nsemble **D**istillation, in order to address the distribution shift between offline and online data distributions. First, we introduce separate offline and online replay buffers in order to select a balanced mix of samples from them for better Q-function updates. This has the dual advantage of updating Q-function with a wide distribution of samples, while making sure that Q-values are updated at novel, unseen states from online interaction. Furthermore, we train multiple actor-critic offline RL agents, and distill the policies into a single policy. Then, during fine-tuning, we improve the distilled policy via Q-ensemble.

[2]Fine-tuning with CQL updates (4) leads to slow (if at all) improvement in most setups we consider (see Appendix E for more results).



(a) Balanced replay (b) Ensemble distillation

Figure 2: Illustrations of our framework. (a) For updating the agent during fine-tuning, we draw a balanced number of samples from both offline and online replay buffers initially, but focus on exploiting online samples at a later training stage. (b) We first train an ensemble of N independent offline RL agents with the given offline dataset. Then, we distill the ensemble of policies into a single policy, such that the policy may receive a more accurate learning signal from the Q-ensemble.

4.1 Balancing experiences from online and offline replay buffers

In order to address the distribution shift between offline and online samples, we introduce separate replay buffers for offline and online samples, respectively. Then, we update the agent using a mix of both offline and online samples. Specifically, at timestep t, we draw $B \cdot (1 - \rho_t^{\texttt{on}})$ samples from the offline replay buffer, and $B \cdot \rho_t^{\texttt{on}}$ samples from the online replay buffer, where B is the batch size, and the fraction of online samples at timestep t, $\rho_t^{\texttt{on}}$, is defined as follows:

$$\rho_t^{\texttt{on}} = \rho_0^{\texttt{on}} + (1 - \rho_0^{\texttt{on}}) \cdot \frac{\min(t, t_{\texttt{final}})}{t_{\texttt{final}}}, \tag{7}$$

where $t_{\texttt{final}}$ denotes the final timestep of the annealing schedule, and $\rho_0^{\texttt{on}}$ is the initial fraction of online samples. In other words, we initially let the agent see a mix of both offline samples and online samples, and linearly increase the proportion of online samples (see Figure 2a). This has the effect of (a) better Q-fuction updates with a wide distribution of both offline and online samples, and (b) eventually exploiting the online samples later when there are enough online samples gathered. We empirically show that this simple strategy improves the fine-tuning performance by balancing the experiences from online and offline replay buffers (see Figure 4 for supporting experimental results).

4.2 Ensemble of offline RL agents for online fine-tuning

During fine-tuning, each individual Q-function may be inaccurate due to bootstrapping error from unfamiliar online samples. In order to prevent the policy from learning with such erroneous Q-function, we propose to train multiple actor-critic offline RL agents, then distill the policies into a single policy. In turn, we improve the policy using a more accurate value estimate of the Q-ensemble.

Formally, we consider an ensemble of N CQL agents pre-trained via (4), i.e., $\{Q_{\theta_i}, \pi_{\phi_i}\}_{i=1}^N$, where θ_i and ϕ_i denote the parameters of the i-th Q-function and i-th policy, respectively. We then distill the ensemble of independent policies $\{\pi_{\phi_i}\}_{i=1}^N$ into a single distilled policy $\pi_{\phi_{\texttt{pd}}}$ (see Figure 2b) by minimizing the following objective before online interaction:

$$\mathcal{L}_{\texttt{distill}}^{\texttt{pd}}(\phi_{\texttt{pd}}) = \mathbb{E}_{s_t \sim \mathcal{D}} \left[||\mu_{\phi_{\texttt{pd}}}(s_t) - \widehat{\mu}(s_t)||^2 + ||\sigma_{\phi_{\texttt{pd}}}(s_t) - \widehat{\sigma}(s_t)||^2 \right], \tag{8}$$

$$\widehat{\mu}(s_t) = \frac{1}{N} \sum_{i=1}^{N} \mu_{\phi_i}(s_t), \;\; \widehat{\sigma}^2(s_t) = \frac{1}{N} \sum_{i=1}^{N} \left(\sigma_{\phi_i}^2(s_t) + \mu_{\phi_i}^2(s_t) \right) - \widehat{\mu}(s_t)^2, \tag{9}$$

with $\mu_{\texttt{pd}}, \sigma_{\texttt{pd}}^2$ denoting the mean and variance of the distilled policy $\pi_{\phi_{\texttt{pd}}}$, and $\widehat{\mu}$, $\widehat{\sigma}^2$ the mean and variance of the policy mixture $\frac{1}{N} \sum_{i=1}^N \pi_{\phi_i}$. Then, we update the distilled policy by minimizing the

following objective instead of (3):

$$\mathcal{L}^{\texttt{pd}}_{\texttt{actor}}(\phi_{\texttt{pd}}) = \mathbb{E}_{s_t \sim \mathcal{B}}\left[\mathcal{L}^{\texttt{pd}}_{\pi}(s_t, \phi_{\texttt{pd}})\right], \tag{10}$$

$$\mathcal{L}^{\texttt{pd}}_{\pi}(s_t, \phi_{\texttt{pd}}) = \mathbb{E}_{a_t \sim \pi_{\phi_{\texttt{pd}}}}\left[\alpha \log \pi_{\phi_{\texttt{pd}}}(a_t|s_t) - \frac{1}{N}\sum_{i=1}^{N} Q_{\theta_i}(s_t, a_t)\right], \tag{11}$$

where α is the temperature parameter. As for Q-function updates, we keep a separate target Q-function $Q_{\bar{\theta}_i}$ for each Q_{θ_i}, then minimize the loss (1) independently, in order to ensure diversity among the Q-functions. Trained this way, the distilled policy receives a more accurate learning signal from the Q-ensemble, although each individual Q-function may be relatively inaccurate. Alternatively, one may also consider an ensemble of independent policies, where actions are samples from $\mathcal{N}(\widehat{\mu}(s_t), \widehat{\sigma}(s_t))$ at timestep t, and each policy π_{ϕ_i} is updated with its corresponding Q_{θ_i} via (3). However, we show that our ensemble distillation scheme allows for a more stable improvement over using an ensemble of independent policies in more complex environments (see Figure 5 for supporting experiments).

5 Related work

Offline RL. Offline RL algorithms aim to learn RL agents exclusively with pre-collected datasets. While Agarwal et al. [1] showed that conventional off-policy RL algorithms work well for large and diverse datasets, it has been observed that Q-function evaluation can be erroneous due to out-of-distribution actions [5] otherwise. To address this problem, numerous approaches have been proposed, including policy constraint methods that constrain the policy to stay close to the modeled dataset behavior [6, 17, 27, 31], and conservative Q-learning methods that train the Q-function to be pessimistic in the unseen regime [14, 18, 32]. Our work builds on a conservative Q-learning method [18], so as to allow fast, unconstrained policy updates once fine-tuning starts.

Online RL with offline datasets. Several prior works have explored employing offline datasets for online RL, which offers the advantage of improving both sample-efficiency and exploration. Some assume access to expert demonstrations, then either improve the behavior-cloned policy via on-policy RL [12, 15, 24, 33], or train a (pre-trained) policy via off-policy RL [30]. These methods are limited, for their success highly depends on the optimality of the dataset. To overcome this, Nair et al. [22] proposed to train the policy so that it imitates actions with high advantage estimates in offline and online setups alike. However, this method relies on regression, hence the learned policy seldom surpasses the best data-generating policy. We instead advocate an alternative approach of improving the policy based on the generalization ability of Q-function.

Replay buffer. Using replay buffers for managing samples of different categories (e.g., expert and learner, different tasks, offline and online, etc.) has been explored for solving hard exploration problems [8, 30], or for continual learning [25]. Our work differs from these, in that we do not assume access to optimal data, nor do we consider multi-task setups. A similar balanced replay scheme as ours has been used in a robot learning setup [13], but their focus is on preventing overfitting. We instead study diverse scenarios with datasets of varying suboptimality, where the agent's initial fine-tuning procedure may suffer due to distribution shift.

Ensemble methods. In the context of model-free RL, ensemble methods have been studied for addressing Q-function's overestimation bias [2, 5, 10, 11, 19], for better exploration [3, 20, 23], or for reducing bootstrap error propagation [20]. The closest to our approach is Anschel et al. [2] that proposed to stabilize Q-learning by using the average of previously learned Q-values as a target Q-value. Our approach instead utilizes an ensemble of independently trained Q-functions, in order to maintain diversity among Q-functions.

6 Experiments

We designed our experiments to answer the following questions:

- How does BRED compare to other fine-tuning methods (see Figure 3)?
- How crucial is balanced replay for fine-tuning performance (see Figure 4)?
- Can ensemble distillation stabilize the fine-tuning procedure (see Figure 5)?
- How does ensemble size affect performance (see Figure 6)?

6.1 Setups

Tasks and implementation details. We evaluate BRED on MuJoCo tasks [29], i.e., `halfcheetah`, `hopper`, and `walker2d`, from the D4RL benchmark [4]. In order to demonstrate the applicability of our method on various suboptimal datasets, we use four dataset types, i.e., `random`, `medium`, `medium-replay`, and `medium-expert`. Specifically, `random` and `medium` datasets contain samples collected by a random policy and a medium-level policy, respectively. `medium-replay` datasets contain all samples encountered while training a medium-level agent from scratch, and `medium-expert` datasets contain samples collected by both medium-level and expert-level policies. Following the setup in Kumar et al. [18], we trained offline RL agents for 1000 epochs without early stopping. For our method, we use $N = 5$ for the number of CQL agents, $\rho_0^{\text{on}} \in \{0.5, 0.75\}$ for initial fraction of online samples, and $t_{\texttt{final}} = 125\text{K}$ for the final step of annealing schedule in (7). We report the mean and standard deviation across four runs for 250K timesteps. More details are provided in Appendix A.

Baselines. We consider the following methods as baselines:

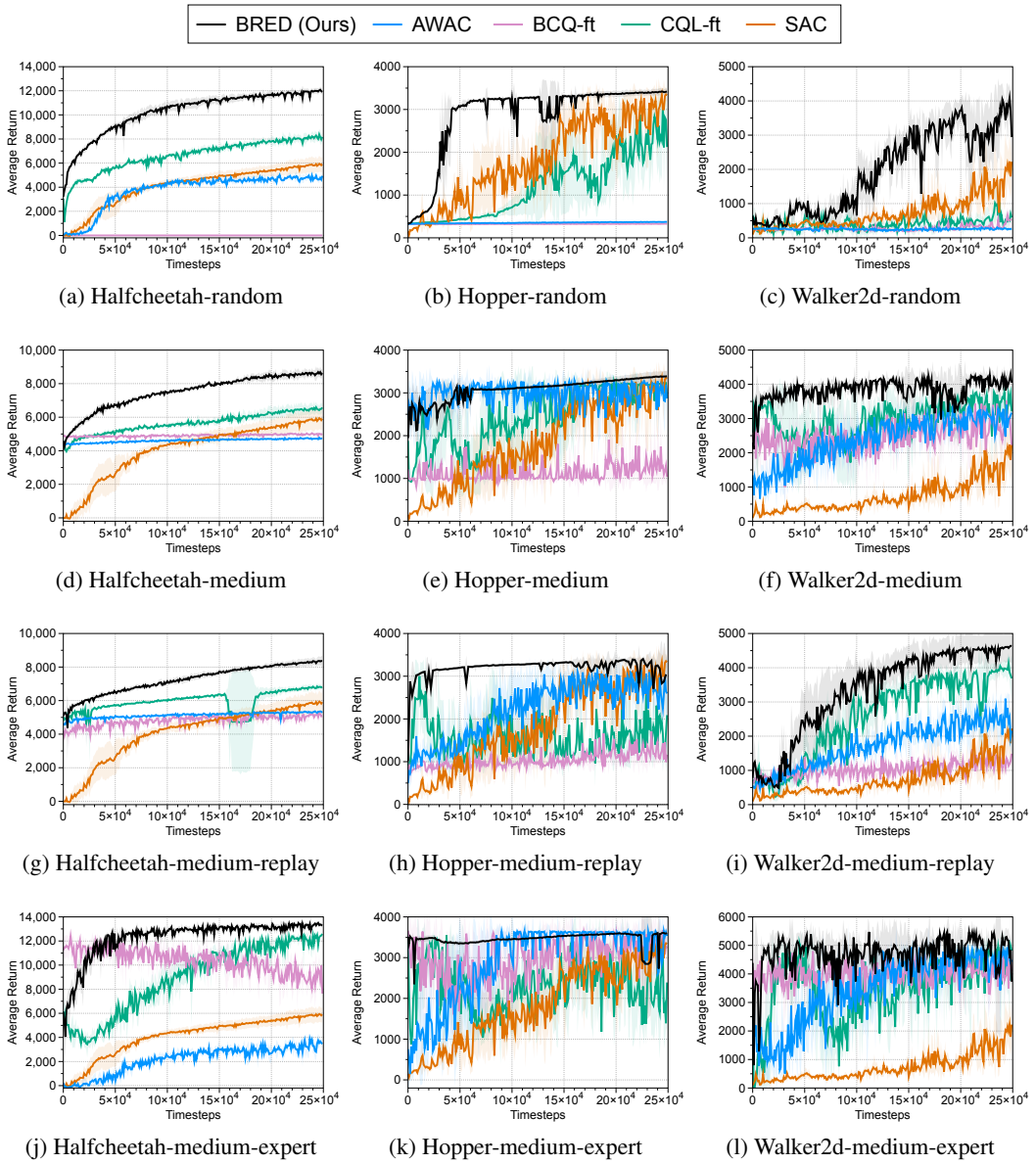
- Advantage-Weighted Actor Critic (AWAC) [22]: an actor-critic scheme for fine-tuning, where the policy is trained to imitate actions with high advantage estimates. Comparison of BRED to AWAC shows the benefit of exploiting the generalization ability of Q-function for policy learning.
- BCQ-ft: Batch-Constrained deep Q-learning (BCQ) [6], is an offline RL method that updates policy by modeling the data-generating policy using a conditional VAE [28]. We extend BCQ to the online fine-tuning setup by applying the same update rules as offline training.
- CQL-ft: Starting from a CQL agent trained via (4), we fine-tune the agent via SAC updates (1), (3). Justification for excluding the CQL regularization term (6) during fine-tuning can be found in Appendix E.
- SAC: a SAC agent trained from scratch. In particular, we assume the agent does not have access to the offline dataset. We include this baseline to show the benefit of fine-tuning a pre-trained agent in terms of sample-efficiency.

6.2 Comparative evaluation

Figure 3 shows the performances of BRED and baseline methods during fine-tuning, based on various type of datasets from the D4RL benchmark [4]. In most tasks, BRED outperforms all baselines in terms of both sample-efficiency and final performance. In particular, our method significantly outperforms CQL-ft, which shows that balanced replay and ensemble distillation play an essential role in stabilizing the fine-tuning process overall. Note that BRED removes the initial performance dip observed in CQL-ft by addressing the harmful effect coming from distribution shift. We also emphasize that BRED performs consistently well across all tasks, while the performances of AWAC and BCQ-ft are highly dependent on the quality of the dataset used. For example, we observe that AWAC and BCQ-ft show competitive performances in tasks where the datasets are generated by high-quality policies, i.e., `medium-expert` tasks, but perform worse than SAC on `random` tasks. This is because AWAC and BCQ-ft employ the same regularized update rule for offline and online setups alike, and fail to balance the offline and online experiences for replay.

6.3 Ablation and analysis

Effects of balanced replay. In order to investigate the effectiveness of balanced replay, we compare the performance of BRED with the performance of BRED without balanced replay. In particular, instead of balanced replay scheme, we consider two setups: (i) **Uniform replay**, where offline and online samples are sampled uniformly from the same buffer, and (ii) **Online only**, where the agent uses online samples exclusively for updates. For all methods, we applied the proposed ensemble distillation. As seen in Figure 4, BRED is the only method that learns in a stable manner, while the other two methods suffer from slow and unstable improvement. This shows that balanced replay is crucial for addressing distribution shift and stabilizing fine-tuning. Results for all other setups can be found in Appendix B.



(a) Halfcheetah-random (b) Hopper-random (c) Walker2d-random

(d) Halfcheetah-medium (e) Hopper-medium (f) Walker2d-medium

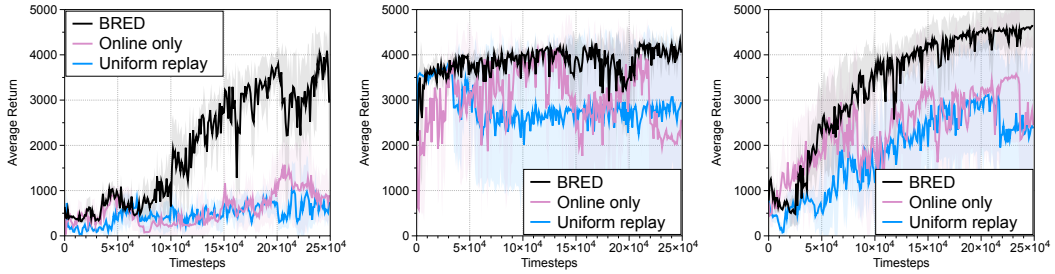
(g) Halfcheetah-medium-replay (h) Hopper-medium-replay (i) Walker2d-medium-replay

(j) Halfcheetah-medium-expert (k) Hopper-medium-expert (l) Walker2d-medium-expert

Figure 3: Performance on MuJoCo tasks from the D4RL benchmark [4] during online fine-tuning. The solid lines and shaded regions represent mean and standard deviation, respectively, across four runs.

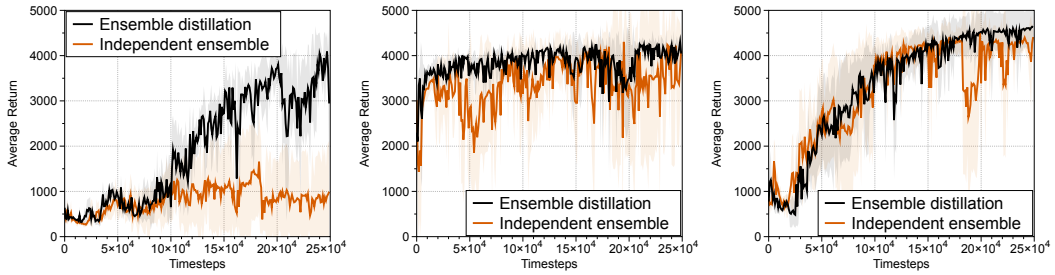
Effects of ensemble distillation. We also analyze the effect of the proposed ensemble distillation method on fine-tuning performance. To this end, we compare BRED to an ensemble of independent policies trained via (3). As shown in Figure 5, ensemble distillation significantly improves performance in complex tasks such as `walker2d`, where policy updates must be more robust to distribution shift. In particular, for the `walker2d-random` task, the ensemble of independent policies fails to learn a highly performant policy, while BRED achieves a near-expert score at the end. Overall trends are similar for other setups considered (see Appendix C).

Effects of ensemble size. We investigate the performance of our method while varying the ensemble size $N \in \{1, 2, 5\}$ on `walker2d-medium-replay` task. We remark that for the single-model case, i.e., $N = 1$, we did not use policy distillation. Figure 6 shows that that the fine-tuning performance of BRED improves as N increases, which shows that larger ensemble size leads to more stable policy updates. More experimental results for all tasks can be found in Appendix D, where the trends are similar.



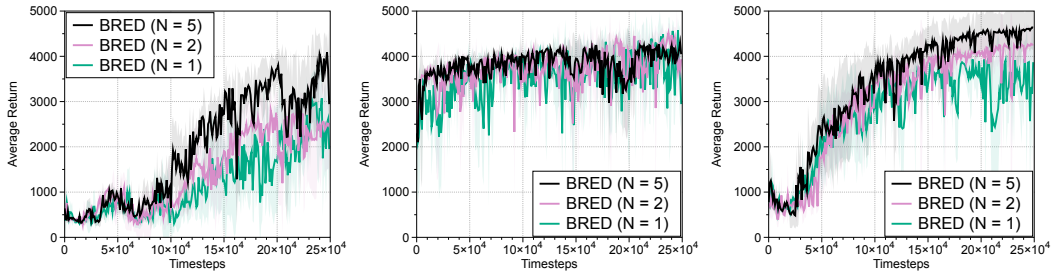
(a) Walker2d-random (b) Walker2d-medium (c) Walker2d-medium-replay

Figure 4: Performance on `walker2d` tasks from the D4RL benchmark [4] with and without balanced replay. Specifically, we consider two setups: **Uniform replay**, where offline and online samples are sampled uniformly from the same buffer for updates, and **Online only**, where the agent uses online samples exclusively for updates.



(a) Walker2d-random (b) Walker2d-medium (c) Walker2d-medium-replay

Figure 5: We compare BRED (using ensemble distillation) to its variant, an ensemble of independent policies (i.e., no distillation), on `walker2d` tasks. One can observe that ensemble distillation improves both stability and performance during online fine-tuning.



(a) Walker2d-random (b) Walker2d-medium (c) Walker2d-medium-replay

Figure 6: Performance on `walker2d` tasks from the D4RL benchmark [4] with varying ensemble size $N \in \{1, 2, 5\}$. We observe that performance of BRED improves as N increases.

7 Conclusion

In this paper, we present BRED, which mitigates the harmful effects of distribution shift between offline and online data distributions, thereby facilitating stable fine-tuning. BRED incorporates two components, namely, a balanced experience replay scheme that mixes offline and online samples for training, and an ensemble distillation scheme for stabilizing policy learning. Our experiments show that BRED performs well across many different setups, and in particular, outperforms prior works that tackle the problem of online reinforcement learning with offline datasets. We believe BRED could prove to be useful for other relevant topics such as sim-to-real transfer [26], scalable RL [13], and RL safety [7].

References

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

[2] Anschel, Oron, Baram, Nir, and Shimkin, Nahum. Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning. In *International Conference on Machine Learning*, 2017.

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

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

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

[6] Fujimoto, Scott, Meger, David, and Precup, Doina. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, 2019.

[7] Garcıa, Javier and Fernández, Fernando. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

[8] Gulcehre, Caglar, Paine, Tom Le, Shahriari, Bobak, Denil, Misha, Hoffman, Matt, Soyer, Hubert, Tanburn, Richard, Kapturowski, Steven, Rabinowitz, Neil, Williams, Duncan, Barth-Maron, Gabriel, Wang, Ziyu, de Freitas, Nando, and Team, Worlds. Making efficient use of demonstrations to solve hard exploration problems. In *International Conference on Learning Representations*, 2020.

[9] Haarnoja, Tuomas, Zhou, Aurick, Abbeel, Pieter, and Levine, Sergey. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, 2018.

[10] Hasselt, Hado V. Double q-learning. In *Advances in Neural Information Processing Systems*, 2010.

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

[12] Ijspeert, Auke J, Nakanishi, Jun, and Schaal, Stefan. Learning attractor landscapes for learning motor primitives. In *Advances in neural information processing systems*, 2003.

[13] Kalashnikov, Dmitry, Irpan, Alex, Pastor, Peter, Ibarz, Julian, Herzog, Alexander, Jang, Eric, Quillen, Deirdre, Holly, Ethan, Kalakrishnan, Mrinal, Vanhoucke, Vincent, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning*, 2018.

[14] Kidambi, Rahul, Rajeswaran, Aravind, Netrapalli, Praneeth, and Joachims, Thorsten. Morel: Model-based offline reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020.

[15] Kim, Beomjoon, Farahmand, Amir-massoud, Pineau, Joelle, and Precup, Doina. Learning from limited demonstrations. In *Advances in Neural Information Processing Systems*, 2013.

[16] Kingma, Diederik P and Ba, Jimmy. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

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

[18] Kumar, Aviral, Zhou, Aurick, Tucker, George, and Levine, Sergey. Conservative q-learning for offline reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020.

[19] Lan, Qingfeng, Pan, Yangchen, Fyshe, Alona, and White, Martha. Maxmin q-learning: Controlling the estimation bias of q-learning. In *International Conference on Learning Representations*, 2020.

[20] Lee, Kimin, Laskin, Michael, Srinivas, Aravind, and Abbeel, Pieter. Sunrise: A simple unified framework for ensemble learning in deep reinforcement learning. *arXiv preprint arXiv:2007.04938*, 2020.

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

[22] Nair, Ashvin, Dalal, Murtaza, Gupta, Abhishek, and Levine, Sergey. Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

[23] Osband, Ian, Blundell, Charles, Pritzel, Alexander, and Van Roy, Benjamin. Deep exploration via bootstrapped dqn. In *Advances in Neural Information Processing Systems*, 2016.

[24] Rajeswaran, Aravind, Kumar, Vikash, Gupta, Abhishek, Vezzani, Giulia, Schulman, John, Todorov, Emanuel, and Levine, Sergey. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. In *Robotics: Science and Systems*, 2018.

[25] Rolnick, David, Ahuja, Arun, Schwarz, Jonathan, Lillicrap, Timothy, and Wayne, Gregory. Experience replay for continual learning. In *Advances in Neural Information Processing Systems*, 2019.

[26] Rusu, Andrei A, Večerík, Matej, Rothörl, Thomas, Heess, Nicolas, Pascanu, Razvan, and Hadsell, Raia. Sim-to-real robot learning from pixels with progressive nets. In *Conference on Robot Learning*, 2017.

[27] Siegel, Noah Y, Springenberg, Jost Tobias, Berkenkamp, Felix, Abdolmaleki, Abbas, Neunert, Michael, Lampe, Thomas, Hafner, Roland, and Riedmiller, Martin. Keep doing what worked: Behavioral modelling priors for offline reinforcement learning. *arXiv preprint arXiv:2002.08396*, 2020.

[28] Sohn, Kihyuk, Lee, Honglak, and Yan, Xinchen. Learning structured output representation using deep conditional generative models. In *Advances in neural information processing systems*, 2015.

[29] Todorov, Emanuel, Erez, Tom, and Tassa, Yuval. Mujoco: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems*, 2012.

[30] Vecerik, Mel, Hester, Todd, Scholz, Jonathan, Wang, Fumin, Pietquin, Olivier, Piot, Bilal, Heess, Nicolas, Rothörl, Thomas, Lampe, Thomas, and Riedmiller, Martin. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

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

[32] Yu, Tianhe, Thomas, Garrett, Yu, Lantao, Ermon, Stefano, Zou, James, Levine, Sergey, Finn, Chelsea, and Ma, Tengyu. Mopo: Model-based offline policy optimization. In *Advances in Neural Information Processing Systems*, 2020.

[33] Zhu, Henry, Gupta, Abhishek, Rajeswaran, Aravind, Levine, Sergey, and Kumar, Vikash. Dexterous manipulation with deep reinforcement learning: Efficient, general, and low-cost. In *International Conference on Robotics and Automation*, 2019.

Appendix

A Experimental details for BRED

Offline datasets. For training offline RL agents with offline datasets, we use the publicly available datasets from the D4RL benchmark (https://github.com/rail-berkeley/d4rl) without any modification to the datasets. Note that we use datasets available from the commit `c4bd3de`, i.e., renewed version of datasets.

Training details for offline RL agents. For training CQL agents, following Kumar et al. [18], we built CQL on top of the publicly available implementation of SAC (https://github.com/vitchyr/rlkit) without any modification to hyperparameters. As for network architecture, we use 2-layer multi-layer perceptrons (MLPs) for value and policy networks (except for `halfcheetah-medium-expert` task where we found that 3-layer MLPs is more effective for training CQL agents). For training BCQ [6], we use the publicly available implementation from https://github.com/rail-berkeley/d4rl_evaluations/tree/master/bcq. For all experiments, following the setup in Kumar et al. [18], we trained offline RL agents for 1000 epochs without early stopping.

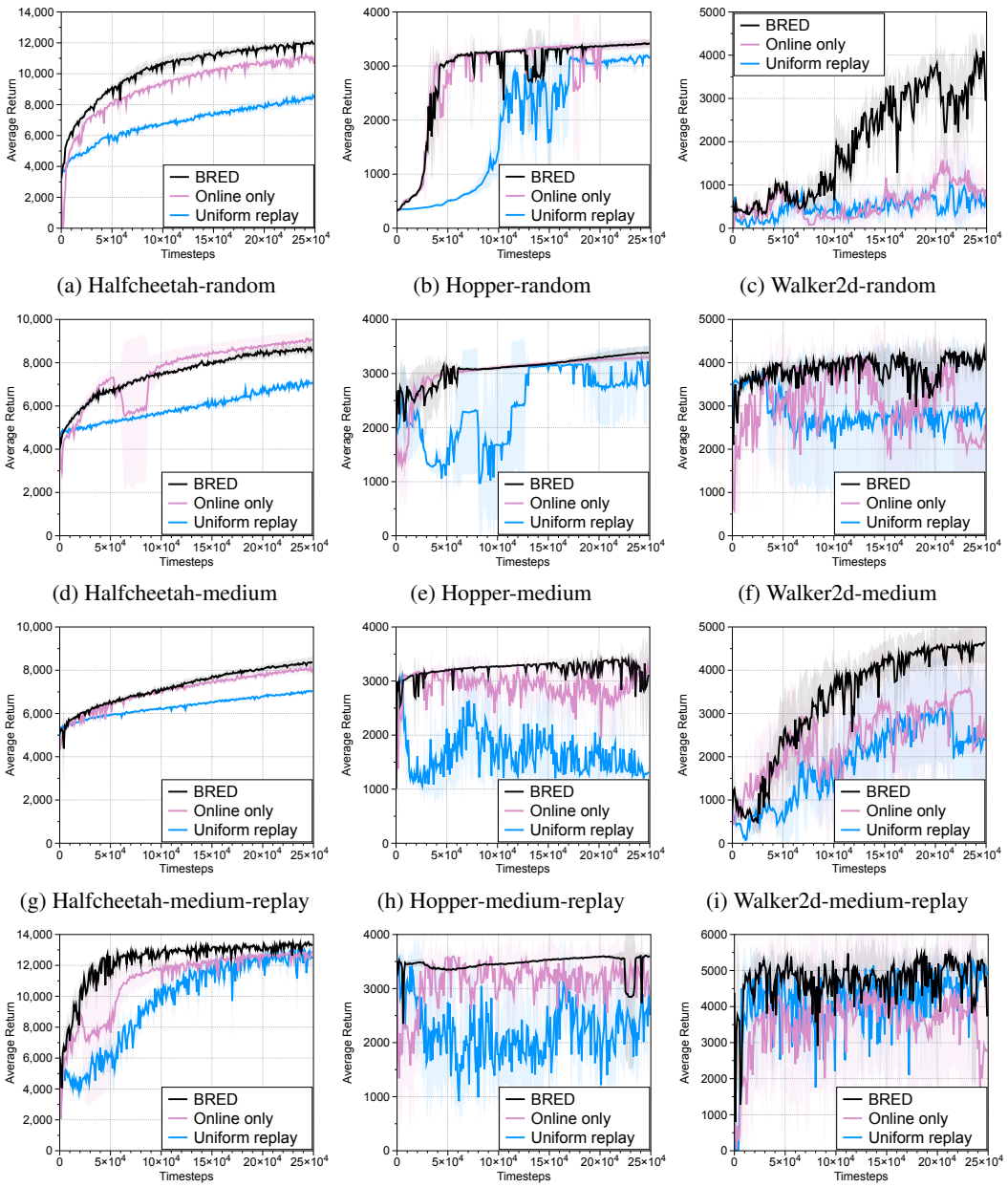
Training details for fine-tuning. To fine-tune a CQL agent with our method, we initialized parameters of policy and value networks using the parameters from the pre-trained offline CQL agent. For all experiments, we report the performance during the 250K timesteps for 4 random seeds. For our method, we used Adam optimizer [16] with policy learning rate of $\{3\text{e}{-4}, 3\text{e}{-5}, 5\text{e}{-6}\}$ and value learning rate of $3\text{e}{-4}$. We found that taking $10\times$ more training steps (10000 as opposed to the usual 1000 steps) after collecting the first 1000 online samples slightly improves the fine-tuning performance. After that, we trained for 1000 training steps every time 1000 additional samples were collected. For AWAC [22], we use the publicly available implementation from the authors (https://github.com/vitchyr/rlkit/tree/master/examples/awac) without any modification to hyperparameters and architectures. For BCQ-ft, we use the same implementation as the original BCQ, with the only difference being that we used additional online samples for training.

Training details for balanced replay. For fine-tuning offline RL agents with the proposed balanced replay, we use $\rho_0^{\texttt{on}} \in \{0.5, 0.75\}$ for initial fraction of online samples, 256 for batch size B, and $t_{\texttt{final}} = 125\text{K}$ for the final step of annealing schedule in (7).

Training details for ensemble distillation. For the proposed ensemble distillation method, we use $N = 5$ CQL agents. To distill the ensemble of policies into a single distilled policy, we optimize distillation objective (8) for 1000 epochs with early stopping using the pre-defined validation samples. For action selection during evaluation, we use the mean of distilled policy $\pi_{\phi_{\texttt{pd}}}$.

B Effects of balanced experience replay scheme

We provide learning curves for all tasks with and without balanced replay. The proposed balanced replay scheme is indeed crucial for a stable fine-tuning procedure.



(a) Halfcheetah-random (b) Hopper-random (c) Walker2d-random

(d) Halfcheetah-medium (e) Hopper-medium (f) Walker2d-medium

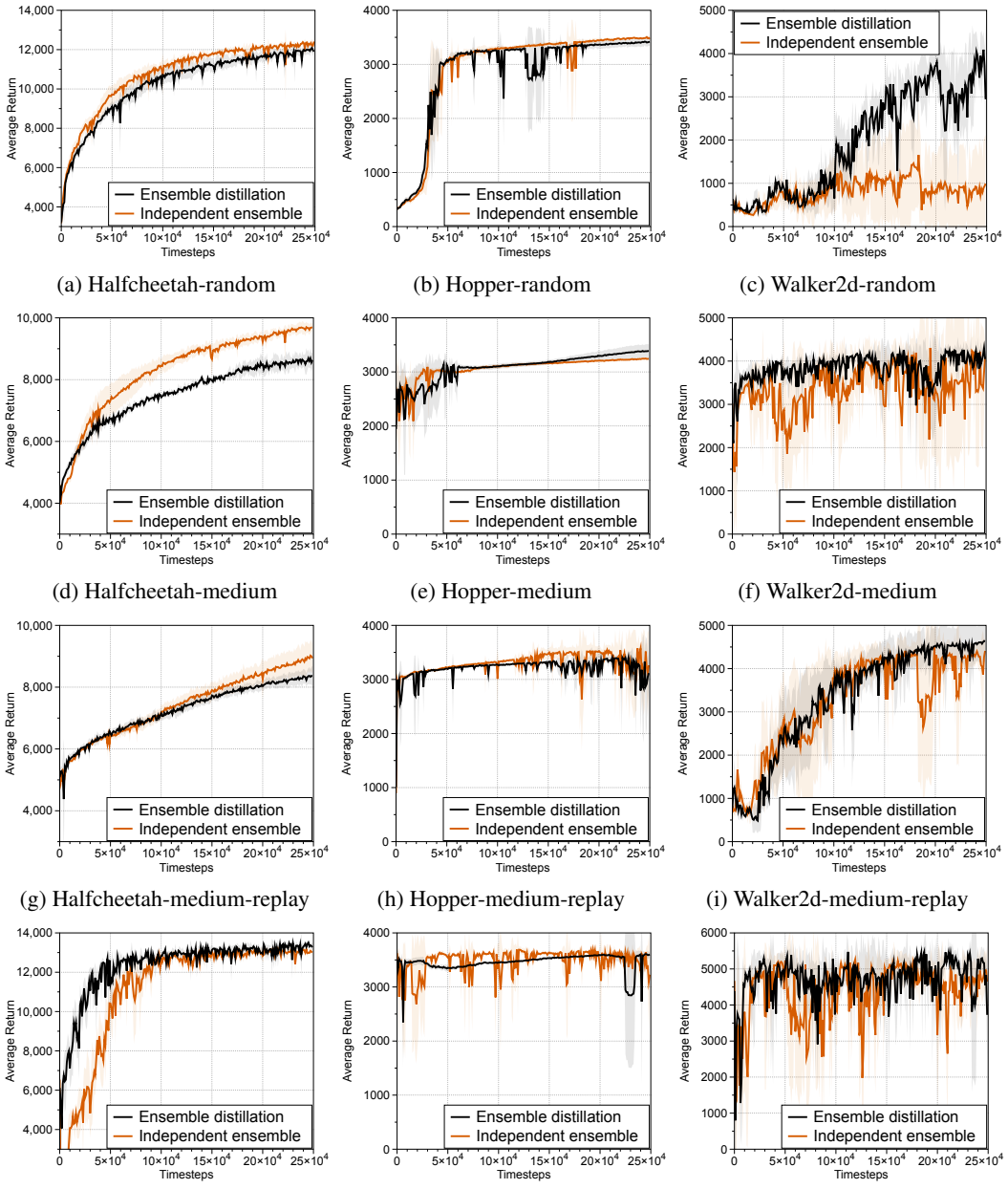
(g) Halfcheetah-medium-replay (h) Hopper-medium-replay (i) Walker2d-medium-replay

(j) Halfcheetah-medium-expert (k) Hopper-medium-expert (l) Walker2d-medium-expert

Figure 7: Performance on MuJoCo tasks from the D4RL benchmark [4] during online fine-tuning. The solid lines and shaded regions represent mean and standard deviation, respectively, across four runs.

C Effects of ensemble distillation

We provide learning curves for all tasks with and without ensemble distillation scheme. One can see that ensemble distillation is highly effective for more complex tasks such as `walker`. However, we observe that BRED without ensemble distillation method, i.e., Independent ensemble, sometimes performs better in simple tasks such as `halfcheetah`, as exploration with diverse policies is relatively more important for these tasks.



(a) Halfcheetah-random (b) Hopper-random (c) Walker2d-random

(d) Halfcheetah-medium (e) Hopper-medium (f) Walker2d-medium

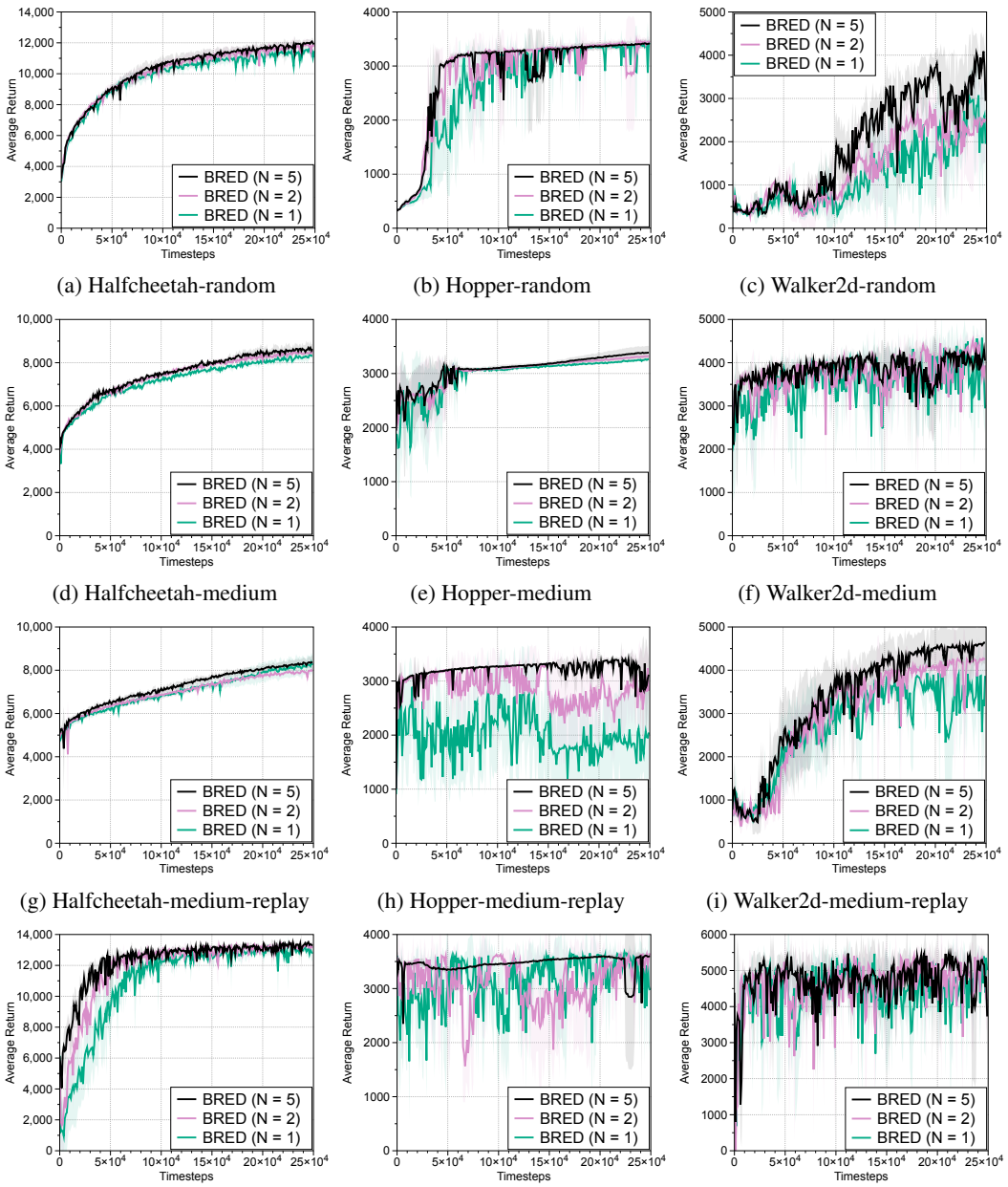
(g) Halfcheetah-medium-replay (h) Hopper-medium-replay (i) Walker2d-medium-replay

(j) Halfcheetah-medium-expert (k) Hopper-medium-expert (l) Walker2d-medium-expert

Figure 8: Performance on MuJoCo tasks from the D4RL benchmark [4] during online fine-tuning. The solid lines and shaded regions represent mean and standard deviation, respectively, across four runs.

D Effects of ensemble size

We provide learning curves for all tasks with varying ensemble size $N \in \{1, 2, 5\}$. One can see that performance of BRED improves as N grows.



(a) Halfcheetah-random (b) Hopper-random (c) Walker2d-random

(d) Halfcheetah-medium (e) Hopper-medium (f) Walker2d-medium

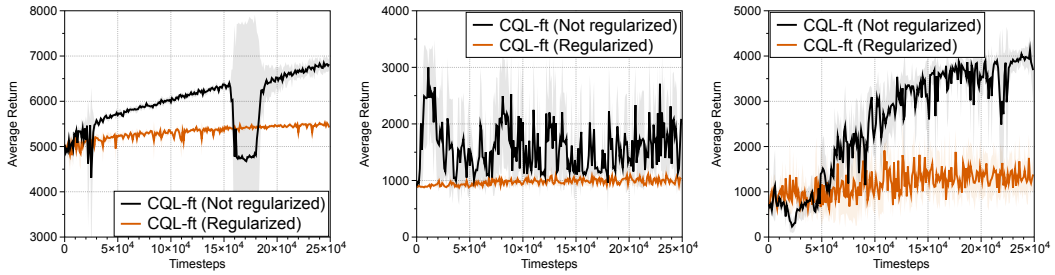
(g) Halfcheetah-medium-replay (h) Hopper-medium-replay (i) Walker2d-medium-replay

(j) Halfcheetah-medium-expert (k) Hopper-medium-expert (l) Walker2d-medium-expert

Figure 9: Performance on MuJoCo tasks from the D4RL benchmark [4] during online fine-tuning. The solid lines and shaded regions represent mean and standard deviation, respectively, across four runs.

E Effects of CQL regularization on fine-tuning

Here, we show that removing the regularization term (6) from the CQL update is crucial to the fine-tuning performance. As shown in Figure 10, fine-tuning with CQL regularization prevents the agent from improving via online interaction, for the Q-function is updated to be pessimistic for unseen actions. This essentially results in failure to explore. For BRED, we do away with regularization, and instead use the modified SAC update (10).



(a) Halfcheetah-medium-replay (b) Hopper-medium-replay (c) Walker2d-medium-replay

Figure 10: Performance of CQL-ft methods on (a) `halfcheetah-medium-replay`, (b) `hopper-medium-replay`, and (c) `walker2d-medium-replay` tasks from the D4RL benchmark [4] during online fine-tuning. The solid lines and shaded regions represent mean and standard deviation, respectively, across four runs.