DROPOUT Q-FUNCTIONS FOR DOUBLY EFFICIENT REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Randomized ensemble double Q-learning (REDQ) (Chen et al., 2021b) has recently achieved state-of-the-art sample efficiency on continuous-action reinforcement learning benchmarks. This superior sample efficiency is possible by using a large Q-function ensemble. However, REDQ is much less computationally efficient than non-ensemble counterparts such as Soft Actor-Critic (SAC) (Haarnoja et al., 2018a). To make REDQ more computationally efficient, we propose a method of improving computational efficiency called Dr.Q, which is a variant of REDQ that uses a small ensemble of dropout Q-functions. Our dropout Q-functions are simple Q-functions equipped with dropout connection and layer normalization. Despite its simplicity of implementation, our experimental results indicate that Dr.Q is doubly (sample and computationally) efficient. It achieved comparable sample efficiency with REDQ and much better computational efficiency than REDQ and comparable computational efficiency with that of SAC.

1 INTRODUCTION

In the reinforcement learning (RL) community, improving **sample efficiency** of RL methods has been important. Traditional RL methods have been shown to be promising for solving complex control tasks such as dexterous in-hand manipulation (OpenAI et al., 2018). However, RL methods generally require millions of training samples to solve a task (Mendonca et al., 2019). This poor sample efficiency of RL methods is a severe obstacle to practical RL applications (e.g., applications on limited computational resources or in real-world environments without simulators). Motivated by these issues, many RL methods have been proposed to achieve higher sample efficiency. For example, Haarnoja et al. (2018a;b) proposed Soft Actor-Critic (SAC), which achieved higher sample efficiency than the previous state-of-the-art RL methods (Lillicrap et al., 2015; Fujimoto et al., 2018; Schulman et al., 2017).

Since 2019, RL methods that use a high **update-to-data (UTD)** ratio to achieve high sample efficiency have emerged. The UTD ratio is defined as the number of Q-function updates divided by the number of actual interactions with the environment. A high UTD ratio promotes sufficiently training Q-functions within a few interactions, which leads to sample-efficient learning. Model-Based Policy Optimization (MBPO) (Janner et al., 2019) is a seminal RL method that uses a high UTD ratio of 20–40 and achieves significantly higher sample efficiency than SAC, which uses a UTD ratio of 1. Encouraged by the success of MBPO, many RL methods with high UTD ratios have been proposed (Shen et al., 2020; Lai et al., 2020).

With such methods, randomized ensembled double Q-learning (REDQ) proposed by Chen et al. (2021b) is currently the most sample-efficient method for the Mujoco benchmark. REDQ uses a high UTD ratio and large ensemble of Q-functions. The use of a high UTD ratio promotes estimation bias in policy evaluation, which degrades sample-efficient learning. REDQ uses the ensemble of Q-functions to suppress the estimation bias and improve sample efficiency. Chen et al. (2021b) demonstrated that the sample efficiency of REDQ is equal to or even better than that of MBPO.

However, REDQ leaves room for improvement in terms of **computational efficiency**. REDQ runs 1.1 to 1.4 times faster than MBPO (Chen et al., 2021b) but is still less computationally efficient than non-ensemble-based RL methods (e.g., SAC) due to the use of large ensembles. In Section 4.2,

we show that REDQ runs more than two times slower than SAC and requires much larger memory. Computational efficiency is important in several scenarios, e.g., in RL applications with much lighter on-device computation (e.g. mobile phones or other light-weight edge devices) (Chen et al., 2021a), or in situations in which rapid trials and errors are required for developing RL agents (e.g., hyperparameter tuning or proof of concept for RL applications with limited time resources). Therefore, RL methods that are superior not only in terms of sample efficiency but also in computational efficiency are preferable.

We propose a method of improving computational efficiency. Our method is called Dr.Q and is a REDQ variant that *uses a small ensemble of dropout Q-functions, in which dropout (Srivastava et al., 2014) and layer normalization (Ba et al., 2016) are used (Section 3).* We experimentally show that Dr.Q is doubly (computationally and sample) efficient: (i) Dr.Q significantly improves computational efficiency over REDQ (Section 4.2) by more than two times and (ii) achieves sample efficiency comparable to REDQ (Section 4.1).

Although our primary contribution is **proposing a doubly efficient RL method**, we also make three significant contributions from other perspectives:
1. Simplicity of implementation. Dr.Q can be implemented by basically adding a few lines of readily available functions (dropout and layer normalization) to Q-functions in REDQ (and SAC). This simplicity enables one to easily replicate and extend it.
2. First successful demonstration of the usefulness of dropout in *high* UTD ratio settings. Previous studies incorporated dropout into RL (Gal & Ghahramani, 2016; Harrigan, 2016; Moerland et al., 2017; Gal et al., 2016; 2017; Kahn et al., 2017) (see Section 5 for details). However, these studies focused on *low* UTD ratio settings (i.e., UTD ratio ≤ 1). Dropout approaches generally do not work as well as ensemble approaches (Ovadia et al., 2019; Lakshminarayanan et al., 2017; Durasov et al., 2021). For this reason, instead of dropout approaches, ensemble approaches have been used in RL with high UTD ratio settings (Chen et al., 2021b; Janner et al., 2019; Shen et al., 2020; Hiraoka et al., 2020; Lai et al., 2020). In Section 4, we argue that Dr.Q achieves almost the same or better bias reduction ability and sample/computationally efficiency compared with ensemble-based RL methods in high UTD ratio settings. This sheds light on dropout approaches once again and promotes their use as a reasonable alternative (or complement) to ensemble approaches in high UTD ratio settings.
3. Discovery of engineering insights to effectively apply dropout to RL. Specifically, we discovered that the following three engineering practices are effective in reducing bias and improving sample efficiency: (i) using the dropout and ensemble approaches together for constructing Q-functions (i.e., using *multiple* dropout Q-functions) (Section 4.1 and Appendix A.3); (ii) introducing layer normalization into dropout Q-functions (Section 4.3); and (iii) using dropout for both the current and target Q-functions (Appendix C). These engineering insights were not revealed in previous RL studies and would be useful to practitioners who attempt to apply dropout to RL.

2 PRELIMINARIES

2.1 MAXIMUM ENTROPY REINFORCEMENT LEARNING (MAXIMUM ENTROPY RL)

RL addresses the problem of an agent learning to act in an environment. At each discrete time step t, the environment provides the agent with a state s_t, the agent responds by selecting an action a_t, and then the environment provides the next reward r_t and state s_{t+1}. For convenience, as needed, we use the simpler notations of r, s, a, s', and a' to refer to a reward, state, action, next state, and next action, respectively.

We focus on *maximum entropy* RL, in which an agent aims to find its policy that maximizes the expected return with an entropy bonus: $\arg\max_{\pi} \sum_{t=0}^{T} \gamma^t \mathbb{E}_{s_t,a_t}\left[r_t + \alpha\mathcal{H}(\pi(\cdot|s_t))\right]$. Here, $\pi : \mathcal{S} \times \mathcal{A} \to [0, \infty)$ is a policy and $\mathcal{H}$ is entropy. Temperature α balances exploitation and exploration and affects the stochasticity of the policy.

2.2 RANDOMIZED ENSEMBLE DOUBLE Q-LEARNING (REDQ)

REDQ (Chen et al., 2021b) is a sample-efficient model-free method for solving maximum-entropy RL problems (Algorithm 1). It has two primary components to achieve high sample efficiency.

Algorithm 1 REDQ

1: Initialize policy parameters θ, N Q-function parameters ϕ_i, $i = 1, ..., N$, and empty replay buffer $\mathcal{D}$. Set target parameters $\bar{\phi}_i \leftarrow \phi_i$, for $i = 1, ..., N$.
2: **repeat**
3: Take action $a_t \sim \pi_\theta(\cdot|s_t)$; Observe reward r_t, next state s_{t+1}; $\mathcal{D} \leftarrow \mathcal{D} \bigcup (s_t, a_t, r_t, s_{t+1})$.
4: **for** G updates **do**
5: Sample a mini-batch $\mathcal{B} = \{(s, a, r, s')\}$ from $\mathcal{D}$.
6: Sample a set $\mathcal{M}$ of M distinct indices from $\{1, 2, ..., N\}$.
7: Compute the Q target y (same for all N Q-functions):

$$y = r + \gamma \left(\min_{i \in \mathcal{M}} Q_{\bar{\phi}_i}(s', a') - \alpha \log \pi_\theta(a'|s') \right), \quad a' \sim \pi_\theta(\cdot|s')$$

8: **for** $i = 1, ..., N$ **do**
9: Update ϕ_i with gradient descent using

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

10: Update target networks with $\bar{\phi}_i \leftarrow \rho \bar{\phi}_i + (1 - \rho)\phi_i$.
11: Update θ with gradient ascent using

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

1. High UTD ratio: It uses a high UTD ratio G, which is the number of Q-function updates (lines 4–10) divided by the number of actual interactions with the environment (line 3). The high UTD ratio promotes sufficient training of Q-functions within a few interactions. However, this also promotes overestimation bias in the Q-function training, which degrades sample-efficient learning (Chen et al., 2021b).
2. Ensemble of Q-functions: To reduce the overestimation bias, it uses an ensemble of N Q-functions for the target to be minimized (lines 6–7). Specifically, a random subset $\mathcal{M}$ of the ensemble is selected (line 6) then used for calculating the target (line 7). The size of the subset $\mathcal{M}$ is kept fixed and is denoted as M. In addition, each Q-function in the ensemble is randomly and independently initialized but updated with the same target (lines 8–9). Chen et al. (2021b) showed that using a large ensemble ($N = 10$) and small subset ($M = 2$) successfully reduces the bias.

Although using a large ensemble of Q-functions is beneficial for reducing bias and improving sample efficiency, this makes REDQ computationally intensive. In the next section, we discuss reducing the ensemble size.

3 Injecting Model Uncertainty into Target with Dropout Q-functions

In this section, we discuss replacing the large ensemble of Q-functions in REDQ with a small ensemble of dropout Q-functions. We start our discussion by reviewing what the ensemble in REDQ does from the viewpoint of model uncertainty injection. Then, we propose to use dropout for model uncertainty injection instead of the large ensemble. Specifically, we propose (i) the dropout Q-function that is a Q-function equipped with dropout and layer normalization, and (ii) Dr.Q, a variant of REDQ that uses a small ensemble of dropout Q-functions. Finally, we explain that the size of the ensemble can be smaller in Dr.Q than in REDQ.

We first explain our insight that, in REDQ, model (Q-function parameters') uncertainty is injected into the target. In REDQ, the subset $\mathcal{M}$ of N Q-functions is used to compute the target value $\min_{i \in \mathcal{M}} Q_{\bar{\phi}_i}(s', a')$ (lines 6–7 in Algorithm 1). This can be interpreted as an approximation for

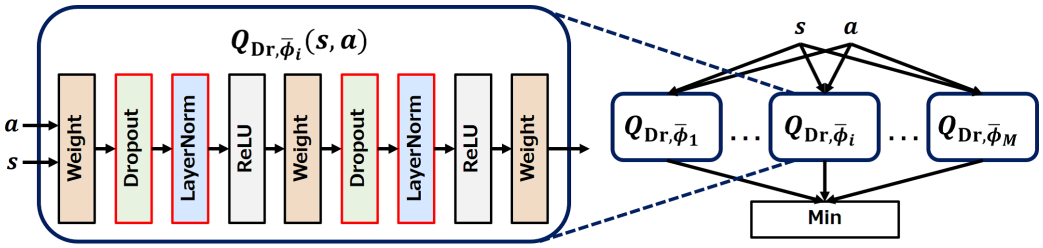

Figure 1: Dropout Q-function implementation (left part) and how dropout Q-functions are used in target (right part). **Dropout Q-function implementation:** Our dropout function is implemented by modifying that used by Chen et al. (2021b). Our modification (highlighted in red) is adding dropout (Dropout) and layer normalization (LayerNorm). "Weight" is a weight layer and "ReLU" is the activation layer of rectified linear units. Parameters $\bar{\phi}_i$ represent the weights and biases in weight layers. **How dropout Q-functions are used in target:** M dropout Q-functions are used to calculate the target value as $\min_{i=1,\dots,M} Q_{\mathrm{Dr},\bar{\phi}_i}(s,a)$.

$\mathbb{E}_{\bar{\phi}_1,\dots,\bar{\phi}_M}\left[\min_{i=1,\dots,M} Q_{\bar{\phi}_i}(s',a')\right]$, the expected target value with respect to model uncertainty:

$$\mathbb{E}_{\bar{\phi}_1,\dots,\bar{\phi}_M}\left[\min_{i=1,\dots,M} Q_{\bar{\phi}_i}(s',a')\right] = \int \min_{i=1,\dots,M} Q_{\bar{\phi}_i}(s',a') \prod_{i=1,\dots,M} p(\bar{\phi}_i) d\bar{\phi}_i$$

$$\approx \int \min_{i=1,\dots,M} Q_{\bar{\phi}_i}(s',a') \prod_{i=1,\dots,M} q_{\mathrm{en}}(\bar{\phi}_i) d\bar{\phi}_i \approx \underbrace{\min_{i\in\mathcal{M}} Q_{\bar{\phi}_i}(s',a')}_{\text{(used in line 7 in Algorithm 1)}} .$$

On the left hand side in the second line, the model distribution $p(\bar{\phi}_i)$ is replaced with a proposal distribution q_{en}, which is based on resampling from the ensemble in line 6 in Algorithm 1[1]. On the right hand side (RHS) in the second line, the expected target value is further approximated by one sample average on the basis of q_{en}. The resulting approximation is used in line 7 in Algorithm 1. In the remainder of this section, we discuss another means to approximate the expected target value.

We use M dropout Q-functions Q_{Dr} for the target value approximation (Figure 1). Here, Q_{Dr} is a Q-function that has dropout connections (Srivastava et al., 2014). The left part of Figure 1 shows Q_{Dr} implementation by adding dropout layers to the Q-function implementation used by Chen et al. (2021b). Layer normalization (Ba et al., 2016) is applied after dropout for more effectively using dropout as with Vaswani et al.; Xu et al. (2019). By using Q_{Dr}, the target value is approximated as

$$\mathbb{E}_{\bar{\phi}_1,\dots,\bar{\phi}_M}\left[\min_{i=1,\dots,M} Q_{\bar{\phi}_i}(s',a')\right] \approx \int \min_{i=1,\dots,M} Q_{\bar{\phi}_i}(s',a') \prod_{i=1,\dots,M} q_{\mathrm{dr}}(\bar{\phi}_i) d\bar{\phi}_i$$

$$\approx \min_{i=1,\dots,M} Q_{\mathrm{Dr},\bar{\phi}_i}(s',a').$$

Instead of q_{en}, a proposal distribution q_{dr} based on the dropout is used on the RHS in the first line. The expected target value is further approximated by one sample average on the basis of q_{dr} in the second line. We use the resulting approximation $\min_{i=1,\dots,M} Q_{\mathrm{Dr},\bar{\phi}_i}(s',a')$ for injecting model uncertainty into the target value (the right part of Figure 1). For calculating $\min_{i=1,\dots,M} Q_{\mathrm{Dr},\bar{\phi}_i}(s',a')$, we use M dropout Q-functions, which have independently initialized and trained parameters $\bar{\phi}_1,\dots,\bar{\phi}_M$. Using M dropout Q-functions improves the performance of Dr.Q (our RL method described in the next paragraph), compared with that using a single dropout Q-function (further details are given in A.3).

We now explain Dr.Q[2], in which Q_{Dr} is used for considering model uncertainty. The algorithmic description of Dr.Q is shown in Algorithm 2. Dr.Q is a variant of REDQ, and the modified parts from REDQ are highlighted in red in the algorithm. In line 6, Q_{Dr} is used to inject model uncertainty into the target, as we discussed in the previous paragraph. In lines 8 and 10, M dropout Q-functions are used instead of N Q-functions ($N \geq M$) to make Dr.Q more computationally efficient.

[1]We assume that $\forall_{i=1,\dots,N},\ \bar{\phi}_i$ independently follow an identical distribution.

[2]The period between "Dr" and "Q" is necessary to avoid name conflict with DrQ (Yarats et al., 2021).

Algorithm 2 Dr.Q

1: Initialize policy parameters θ, M Q-function parameters ϕ_i, $i = 1, ..., M$, and empty replay buffer $\mathcal{D}$. Set target parameters $\bar{\phi}_i \leftarrow \phi_i$, for $i = 1, ..., M$.
2: **repeat**
3: Take action $a_t \sim \pi_\theta(\cdot|s_t)$. Observe reward r_t, next state s_{t+1}; $\mathcal{D} \leftarrow \mathcal{D} \bigcup (s_t, a_t, r_t, s_{t+1})$.
4: **for** G updates **do**
5: Sample a mini-batch $\mathcal{B} = \{(s, a, r, s')\}$ from $\mathcal{D}$.
6: Compute the Q target y for the dropout Q-functions:

$$y = r + \gamma \left(\min_{i=1,\dots,M} Q_{\mathrm{Dr},\bar{\phi}_i}(s', a') - \alpha \log \pi_\theta(a'|s') \right), \quad a' \sim \pi_\theta(\cdot|s')$$

7: **for** $i = 1, ..., M$ **do**
8: Update ϕ_i with gradient descent using

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

9: Update target networks with $\bar{\phi}_i \leftarrow \rho \bar{\phi}_i + (1 - \rho)\phi_i$.
10: Update θ with gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in \mathcal{B}} \left(\frac{1}{M} \sum_{i=1}^{M} Q_{\mathrm{Dr},\phi_i}(s, a) - \alpha \log \pi_\theta(a|s) \right), \quad a \sim \pi_\theta(\cdot|s)$$

The ensemble size of the dropout Q-functions for Dr.Q (i.e., M) should be smaller than that of Q-functions for REDQ (i.e., N). M for Dr.Q is equal to the subset size for REDQ, which is not greater than N. In practice, M is much smaller than N (e.g., $M = 2$ and $N = 10$ in Chen et al. (2021b)). This reduction in the number of Q-functions makes Dr.Q more computationally efficient. Specifically, Dr.Q is faster due to the reduction in the number of Q-function updates (line 8 in Algorithm 2). Dr.Q also requires less memory for holding Q-function parameters. In Section 4.2, we show that Dr.Q is computationally faster and less memory intensive than REDQ.

4 Experiments

We conducted experiments to evaluate and analyse Dr.Q. In Section 4.1, we discuss the evaluation of Dr.Q's performance (sample efficiency and bias-reduction ability). In Section 4.2, we discuss the evaluation of the computational efficiency of Dr.Q. In Section 4.3, we explain the ablation study for Dr.Q.

4.1 Sample efficiency and bias-reduction ability of Dr.Q

To evaluate the performances of Dr.Q, we compared Dr.Q with three baseline methods in MuJoCo benchmark environments (Todorov et al., 2012; Brockman et al., 2016). Following Chen et al. (2021b); Janner et al. (2019), we prepared the following environments: **Hopper**, **Walker2d**, **Ant**, and **Humanoid**. In these environments, we compared the following four methods:
REDQ: Baseline method that follows the REDQ algorithm (Chen et al., 2021b) (Algorithm 1).
SAC: Baseline method that follows the SAC algorithm (Haarnoja et al., 2018a;b). To improve the sample efficiency of this method, delayed policy update and high UTD ratio $G = 20$ were used, as suggested by Chen et al. (2021b).
Dr.Q: Proposed method that follows Algorithm 2.
DUVN: Baseline method. This method is a variant of Dr.Q that uses double uncertainty value networks (Harrigan, 2016; Moerland et al., 2017) for policy evaluation. Specifically, a single (i.e., $M = 1$) target dropout Q-function is used for target calculation on line 6 in Algorithm 2: $y = r + \gamma \left(Q_{\mathrm{Dr},\bar{\phi}_i}(s', a') - \alpha \log \pi_\theta(a'|s') \right), \quad a' \sim \pi_\theta(\cdot|s')$. As with Harrigan (2016); Moerland et al. (2017), layer normalization is not applied in $Q_{\mathrm{Dr},\bar{\phi}_i}(s', a')$. Following Chen et al. (2021b), we set

Table 1: Process times (in msec) required for executing overall loop (e.g., lines 3–10 in Algorithm 2). Times for updating Q-functions (e.g., lines 5–9 in Algorithm 2) are shown in parentheses.

	Hopper-v2	Walker2d-v2	Ant-v2	Humanoid-v2
SAC	910 (870)	870 (835)	888 (848)	893 (854)
REDQ	2328 (2269)	2339 (2283)	2336 (2277)	2400 (2340)
DUVN	664 (636)	762 (731)	733 (700)	692 (660)
Dr.Q	948 (905)	933 (892)	954 (913)	989 (946)

the hyperparameters as $G = 20$, $N = 10$, and $M = 2$ for all methods except DUVN ($M = 1$). More detailed hyperparameter settings are given in Appendix D.

The methods were compared on the basis of average return and estimation bias.
Average return: An average return over episodes. We regarded 1000 environment steps in Hopper and 3000 environment steps in the other environments as one epoch, respectively. After every epoch, we ran ten test episodes with the current policy and recorded the average return.
Average/std. bias: Average and standard deviation of the normalized estimation error (bias) of Q-functions Q_{ϕ_i} (Chen et al., 2021b). The error represents how significantly the Q-value estimation differs from the actual return. Formally, the error is defined as $|Q^\pi(s,a) - \hat{Q}(s,a)|/\mathbb{E}_{\bar{s},\bar{a}\sim\pi}\left[Q^\pi(\bar{s},\bar{a})\right]$, where $Q^\pi(s,a)$ is the discounted Monte Carlo return obtained from the current policy π in the test episodes and $\hat{Q}(s,a)$ is its estimation. The $\hat{Q}(s,a)$ was evaluated as $\hat{Q}(s,a) = 1/N \sum_{i=1}^{N} Q_{\phi_i}(s,a)$ for SAC and REDQ and as $\hat{Q}(s,a) = 1/M \sum_{i=1}^{M} Q_{\mathrm{Dr},\phi_i}(s,a)$ for Dr.Q and DUVN, respectively.

The comparison results (Figure 2) indicate that Dr.Q achieved almost the same level of performance as REDQ. Regarding the average return, REDQ and Dr.Q achieved almost the same sample efficiency overall. In Walker2d and Ant, their learning curves highly overlapped, and there was no significant difference between them. In Humanoid, REDQ was slightly better than Dr.Q. In Hopper, Dr.Q was better than REDQ. In all environments except Hopper, REDQ and Dr.Q improved their average return significantly earlier than SAC and DUVN. Regarding bias, REDQ and Dr.Q consistently kept the value-estimation bias closer to zero than SAC and DUVN in all environments.

4.2 Computational efficiency of Dr.Q

We next evaluated the computational efficiency of Dr.Q. We compared Dr.Q with the baseline methods on the basis of the following criteria: (i) **Process time** required for executing methods; (ii) **Number of parameters** of each method; (iii) **Bottleneck memory consumption** suggested from the Pytorch profiler[3]. Bottleneck memory consumption is the maximum memory consumption recorded when running the methods. For evaluation, we ran each method on a machine equipped with two Intel(R) Xeon(R) CPU E5-2667 v4 and one NVIDIA Tesla K80.

The process times of the methods (Table 1) indicate that Dr.Q runs more than two times faster than REDQ. Dr.Q (and SAC) requires process times in the 800—900-msec range. REDQ requires process times in the 2300—2400-msec range. The numbers in parentheses show that learning Q-functions is dominant in the overall process. This suggests that using more compact (e.g., small numbers of) Q-functions is important for improving process times.

The number of parameters and bottleneck memory consumption of each method indicate that Dr.Q is more memory efficient than REDQ. Regarding the numbers of parameters (Table 2), we can see that those of Dr.Q (and SAC and DUVN) are about one-fifth those of REDQ. Note that the number of parameters of Dr.Q is equivalent to that of SAC since Dr.Q and SAC use the same number (two) of Q-functions. Regarding the bottleneck memory consumption (Table 3), we can see that that for Dr.Q (SAC and DUVN) is about one-third that for REDQ. We can also see that the bottleneck memory consumption is almost independent of the environment. This is because the most memory-intensive process is evaluation of the hidden layers in Q-functions (e.g., applying the ReLU layer), which is not environment specific (number of dimensions of states and actions)[4].

[3] `https://pytorch.org/tutorials/recipes/recipes/profiler_recipe.html`
[4] In our experiment, the number of hidden units is invariant over all environments (Appendix D).

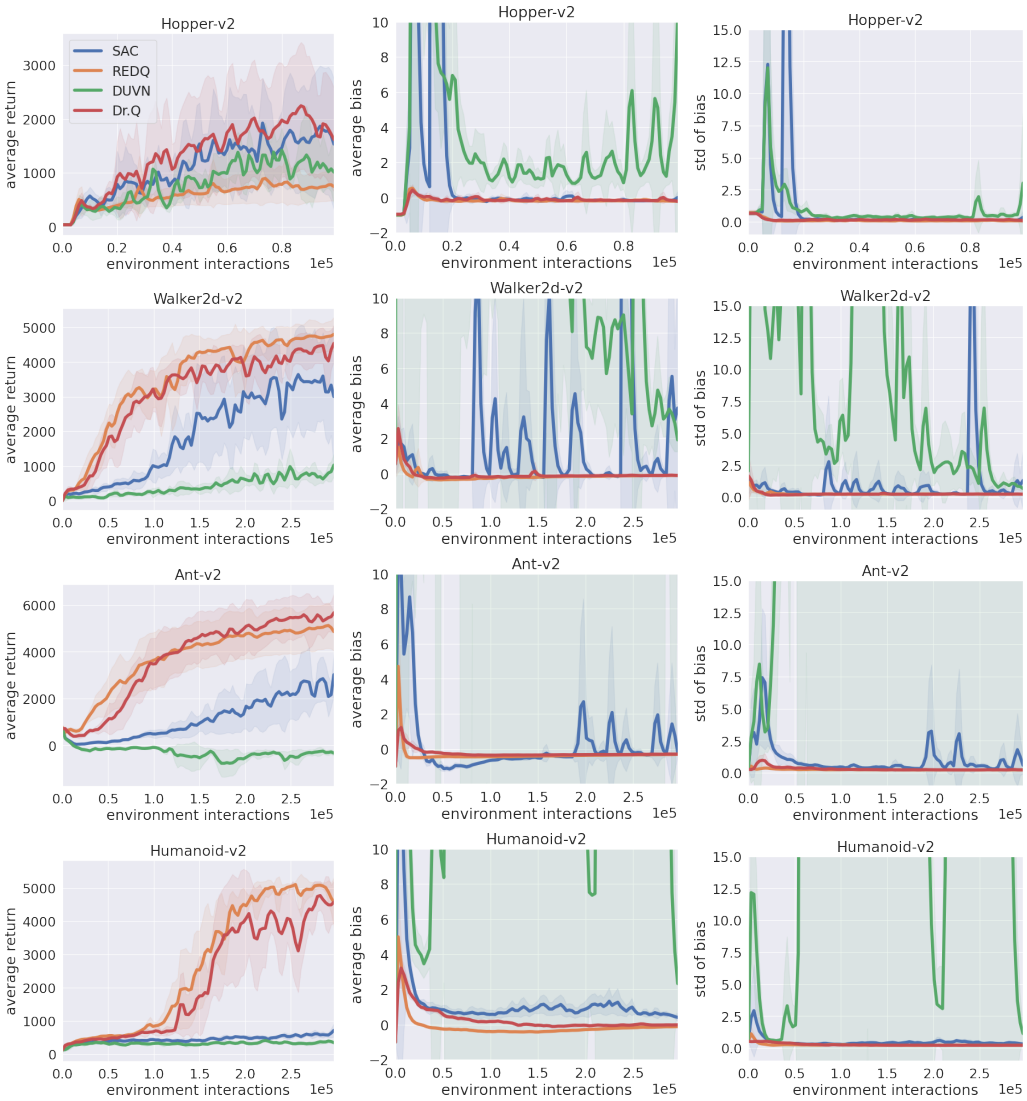

Figure 2: Average return and average and standard deviation of estimation bias for REDQ, SAC, DUVN, and Dr.Q. The horizontal axis represents the number of interactions with the environment (e.g., the number of executions of line 3 of Algorithm 2). For each method, average score of five independent trials are plotted as solid lines, and standard deviation across trials is plotted as transparent shaded region.

Table 2: Number of parameters with methods.

	Hopper-v2	Walker2d-v2	Ant-v2	Humanoid-v2
SAC	141,826	146,434	152,578	166,402
REDQ	698,890	721,930	752,650	821,770
DUVN	139,778	144,386	150,530	164,354
Dr.Q	141,826	146,434	152,578	166,402

4.3 Ablation Study

As an ablation study of Dr.Q, we investigated the performance of Dr.Q variants that do not use either dropout, or layer normalization, or both. We refer to the one without dropout as **-DO**, that without layer normalization as **-LN**, and that without both as **-DO-LN**. The results (Figure 3) indicate that the synergic effect of using dropout and layer normalization is high, especially in complex environments (Ant and Humanoid). In these environments, Dr.Q significantly outperformed its ablation variants (-DO, -LN and -DO-LN) in terms of both average return and bias reduction.

Table 3: Bottleneck memory consumption (in megabytes) with methods. Three worst bottleneck memory consumptions are shown in form of "1st worst bottleneck memory consumption / 2nd worst bottleneck memory consumption / 3rd worst bottleneck memory consumption."

	Hopper-v2	Walker2d-v2	Ant-v2	Humanoid-v2
SAC	73 / 64 / 62	73 / 64 / 62	73 / 64 / 62	73 / 65 / 62
REDQ	241 / 211/ 200	241 / 211 / 200	241 / 211 / 200	241 / 212 / 201
DUVN	72 / 71 / 51	72 / 71 / 51	72 / 71 / 51	72 / 71 / 52
Dr.Q	73 / 72 / 69	73 / 72 / 69	73 / 72 / 70	73 / 72 / 70

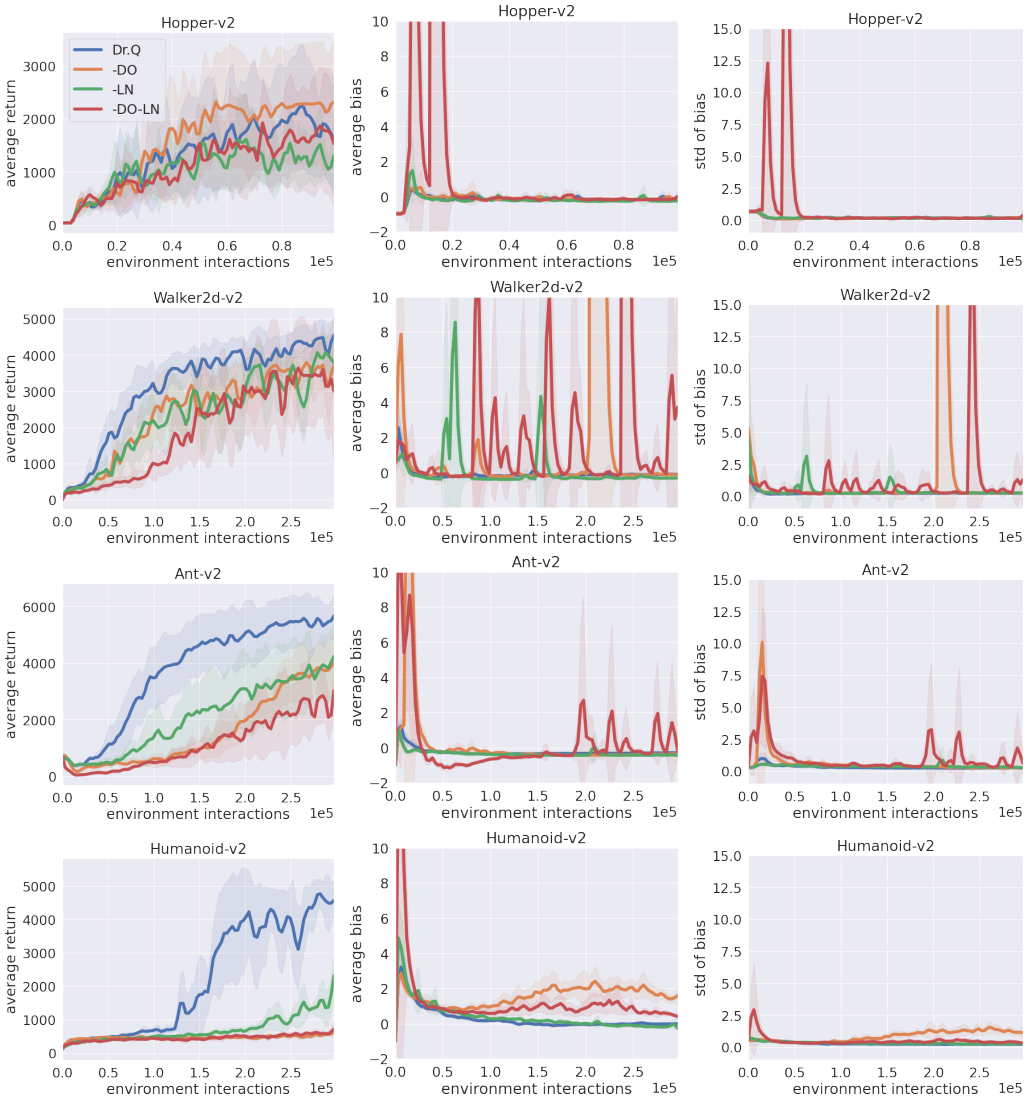

Figure 3: Ablation study results

5 Related Work

In this section, we review related studies and compare them with ours (Table 4).

Ensemble Q-functions: Ensembles of Q-functions have been used in RL to consider model uncertainty (Faußer & Schwenker, 2015; Osband et al., 2016; Anschel et al., 2017; Agarwal et al., 2020; Lee et al., 2021; Lan et al., 2020; Chen et al., 2021b). **Ensemble transition models:** Ensembles of transition (and reward) models have been introduced to model-based RL, e.g., (Chua et al., 2018; Kurutach et al., 2018; Janner et al., 2019; Shen et al., 2020; Yu et al., 2020; Lee et al., 2020; Hiraoka et al., 2020; Abraham et al., 2020). The methods proposed in the above studies use a large

Table 4: Comparison between related studies and ours. We classify related studies into five types (e.g., "Ensemble Q-functions") on basis of three criteria (e.g. "Type of model uncertainty").

Type of study	Method for injecting model uncertainty	Type of model uncertainty	Focus on high UTD ratio (and maximum–entropy RL) setting?
Ensemble Q-functions	Ensemble	Q-functions	Partially yes Chen et al. (2021b)
Ensemble transition models	Ensemble	Transition models	Partially yes e.g., Janner et al. (2019)
Dropout Q-functions	Dropout	Q-functions	No
Dropout transition models	Dropout	Transition models	No
Normalization	–	–	No
Our study	Dropout (with ensemble)	Q-functions	Yes

ensemble of Q-functions or transition models, thus are computationally intensive. Dr.Q does not use a large ensemble of Q-functions, thus is computationally lighter.

Dropout transition models: Gal et al. (2016; 2017); Kahn et al. (2017) introduced dropout and its modified variant to transition models of model-based RL methods. **Dropout Q-functions:** Gal & Ghahramani (2016) introduced dropout to a Q-function at action selection for considering model uncertainty in exploration. Harrigan (2016) and Moerland et al. (2017) introduced dropout to policy evaluation in the same vein as us. However, there are three main differences between these studies and ours. (i) Target calculation: they introduced dropout to a *single* Q-function and used it for a target value, whereas we introduce dropout to *multiple* Q-functions use the minimum of their outputs for the target value. On the basis of insights from Fujimoto et al. (2018); Haarnoja et al. (2018b), overestimation bias should be significantly suppressed by this minimum operation in the target. (ii) Use of engineering to stabilize learning: their methods do not use engineering to stabilize the learning of dropout Q-functions, whereas Dr.Q uses layer normalization to stabilize the learning. (iii) RL setting: they focused on a low UTD ratio (and standard RL) setting, whereas we focused on a high UTD ratio (and maximum entropy RL) setting. As we explained in Section 2.2, a high UTD ratio setting promotes high estimation bias, thus is a more challenging setting than a low UTD ratio setting. In Section 4.1, we showed that their methods do not perform successfully in a high UTD setting.

Normalization in RL: Normalization (e.g., batch normalization (Ioffe & Szegedy, 2015) or layer normalization (Ba et al., 2016)) has been introduced into RL. Batch normalization and its variant are introduced in deep deterministic policy gradient (DDPG) (Lillicrap et al., 2015) and twin delayed DDPG (Fujimoto et al., 2018) (Bhatt et al., 2020). Layer normalization is introduced in the implementation of maximum a posteriori policy optimisation (Abdolmaleki et al., 2018; Hoffman et al., 2020). It is also introduced in SAC extensions (Ma et al., 2020; Zhang et al., 2021). Unlike our study, the above studies did not introduce dropout to consider model uncertainty. In addition, although some studies (Ma et al., 2020; Zhang et al., 2021) focused on the maximum entropy RL setting, none focused on a high UTD ratio setting.

6 Conclusion

We proposed, Dr.Q, an RL method based on a small ensemble of Q-functions that are equipped with dropout connection and layer normalization. We experimentally demonstrated that Dr.Q significantly improves computational and memory efficiencies over REDQ while achieving sample efficiency comparable with REDQ. In the ablation study, we found that using both dropout and layer normalization had a synergistic effect on improving sample efficiency, especially in complex environments (e.g., Humanoid).

References

Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller. Maximum a posteriori policy optimisation. In Proc. ICLR, 2018.

Ian Abraham, Ankur Handa, Nathan Ratliff, Kendall Lowrey, Todd D Murphey, and Dieter Fox. Model-based generalization under parameter uncertainty using path integral control. IEEE Robotics and Automation Letters, 5(2):2864–2871, 2020.

Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In Proc. ICML, 2020.

Oron Anschel, Nir Baram, and Nahum Shimkin. Averaged-DQN: Variance reduction and stabilization for deep reinforcement learning. In Proc. ICML, 2017.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. arXiv preprint arXiv:1607.06450, 2016.

Aditya Bhatt, Max Argus, Artemij Amiranashvili, and Thomas Brox. CrossNorm: On normalization for off-policy reinforcement learning, 2020.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. arXiv preprint arXiv:1606.01540, 2016.

Lili Chen, Kimin Lee, Aravind Srinivas, and Pieter Abbeel. Improving computational efficiency in visual reinforcement learning via stored embeddings. arXiv preprint arXiv:2103.02886, 2021a.

Xinyue Chen, Che Wang, Zijian Zhou, and Keith W. Ross. Randomized ensembled double q-learning: Learning fast without a model. In Proc. ICLR, 2021b.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In Proc. NeurIPS, 2018.

Nikita Durasov, Timur M. Bagautdinov, Pierre Baqué, and Pascal Fua. Masksembles for uncertainty estimation. In Proc. CVPR, 2021.

Stefan Faußer and Friedhelm Schwenker. Neural network ensembles in reinforcement learning. Neural Processing Letters, 41(1):55–69, 2015.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In Proc. ICML, pp. 1587–1596. PMLR, 2018.

Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In Proc. ICLR, pp. 1050–1059. PMLR, 2016.

Yarin Gal, Rowan McAllister, and Carl Edward Rasmussen. Improving PILCO with Bayesian neural network dynamics models. In Data-Efficient Machine Learning workshop on ICML, 2016.

Yarin Gal, Jiri Hron, and Alex Kendall. Concrete dropout. In Proc. NeurIPS, 2017.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Proc. ICML, pp. 1861–1870. PMLR, 2018a.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, and Pieter Abbeel. Soft actor-critic algorithms and applications. arXiv preprint arXiv:1812.05905, 2018b.

Cosmo Harrigan. Deep reinforcement learning with regularized convolutional neural fitted Q iteration. `https://www.machineintelligence.org/papers/rc-nfq.pdf`, 2016.

Takuya Hiraoka, Takahisa Imagawa, Voot Tangkaratt, Takayuki Osa, Takashi Onishi, and Yoshimasa Tsuruoka. Meta-model-based meta-policy optimization. arXiv preprint arXiv:2006.02608, 2020.

Matt Hoffman, Bobak Shahriari, John Aslanides, Gabriel Barth-Maron, Feryal Behbahani, Tamara Norman, Abbas Abdolmaleki, Albin Cassirer, Fan Yang, Kate Baumli, Sarah Henderson, Alex Novikov, Sergio Gomez Colmenarejo, Serkan Cabi, Caglar Gulcehre, Tom Le Paine, Andrew Cowie, Ziyu Wang, Bilal Piot, and Nando de Freitas. Acme: A research framework for distributed reinforcement learning. arXiv preprint arXiv:2006.00979, 2020.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Proc. ICML, 2015.

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. In Proc. NeurIPS, 2019.

Gregory Kahn, Adam Villaflor, Vitchyr Pong, Pieter Abbeel, and Sergey Levine. Uncertainty-aware reinforcement learning for collision avoidance. arXiv preprint arXiv:1702.01182, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 2015.

Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. In Proc. ICLR, 2018.

Hang Lai, Jian Shen, Weinan Zhang, and Yong Yu. Bidirectional model-based policy optimization. In Proc. ICML, 2020.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In Proc. NeurIPS, pp. 6402–6413, 2017.

Qingfeng Lan, Yangchen Pan, Alona Fyshe, and Martha White. Maxmin Q-learning: Controlling the estimation bias of Q-learning. In Proc. ICLR, 2020.

Kimin Lee, Younggyo Seo, Seunghyun Lee, Honglak Lee, and Jinwoo Shin. Context-aware dynamics model for generalization in model-based reinforcement learning. In Proc. ICML, 2020.

Kimin Lee, Michael Laskin, Aravind Srinivas, and Pieter Abbeel. SUNRISE: A simple unified framework for ensemble learning in deep reinforcement learning. In Proc. ICML, 2021.

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

Xiaoteng Ma, Li Xia, Zhengyuan Zhou, Jun Yang, and Qianchuan Zhao. DSAC: Distributional soft actor critic for risk-sensitive reinforcement learning. In Reinforcement Learning for Real Life Workshop at ICML 2019, 2020.

Russell Mendonca, Abhishek Gupta, Rosen Kralev, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Guided meta-policy search. In Proc. NeurIPS, pp. 9653–9664, 2019.

Thomas M Moerland, Joost Broekens, and Catholijn M Jonker. Efficient exploration with double uncertain value networks. arXiv preprint arXiv:1711.10789, 2017.

OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub W. Pachocki, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation. CoRR, abs/1808.00177, 2018.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped DQN. In Proc. NeurIPS, 2016.

Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, D. Sculley, Sebastian Nowozin, Joshua Dillon, Balaji Lakshminarayanan, and Jasper Snoek. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. In Proc. NeurIPS, 2019.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.

Jian Shen, Han Zhao, Weinan Zhang, and Yong Yu. Model-based policy optimization with unsupervised model adaptation. In Proc. NeurIPS, 2020.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15(56):1929–1958, 2014.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In Proc. IROS, pp. 5026–5033. IEEE, 2012.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Proc. NeurIPS.

Jingjing Xu, Xu Sun, Zhiyuan Zhang, Guangxiang Zhao, and Junyang Lin. Understanding and improving layer normalization. In Proc. NeurIPS, 2019.

Denis Yarats, Ilya Kostrikov, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In Proc. ICLR, 2021.

Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Y Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. MOPO: model-based offline policy optimization. In Proc. NeurIPS, 2020.

Amy Zhang, Rowan Thomas McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction. In Proc. ICLR, 2021.

A EFFECT OF DROPOUT RATE ON DR.Q AND ITS VARIANTS

A.1 DR.Q

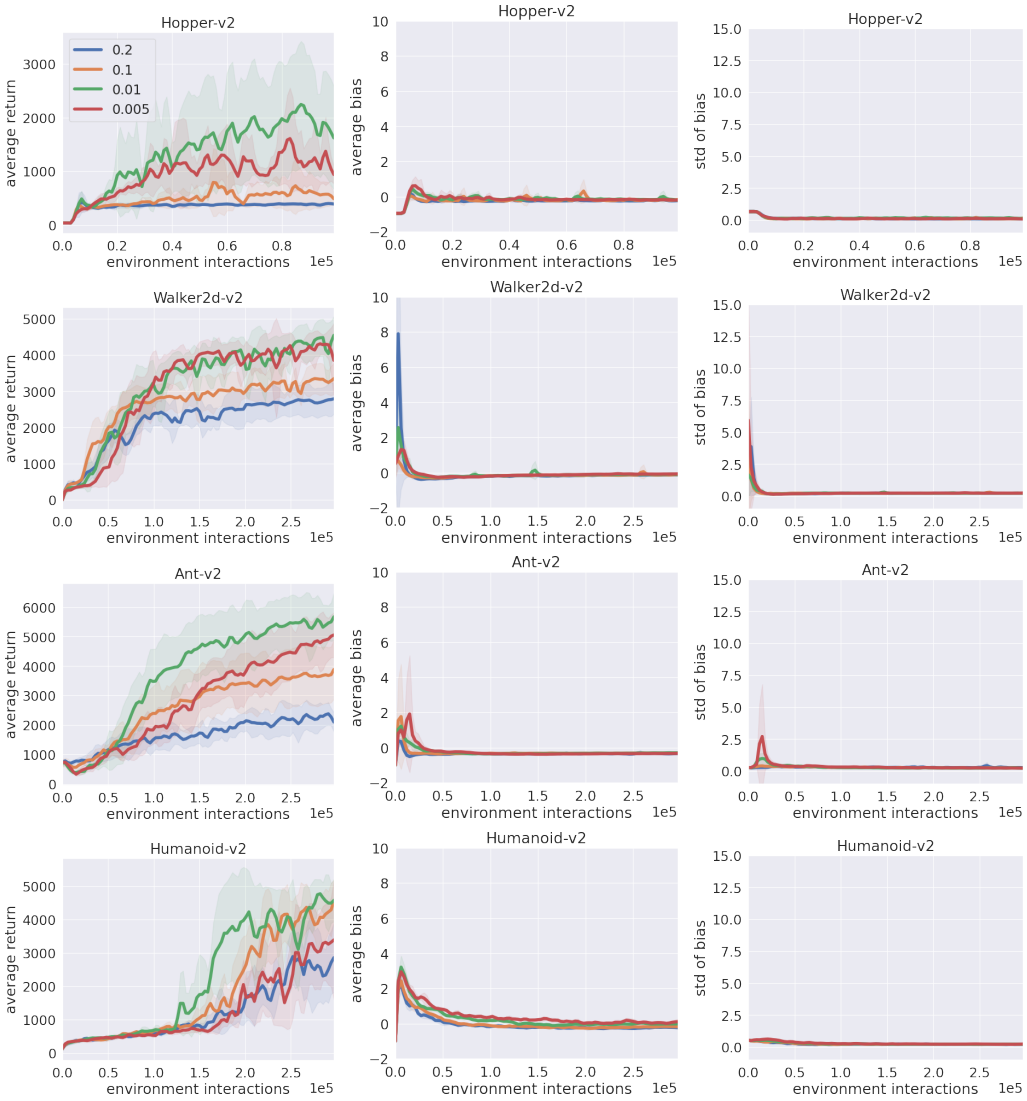

Figure 4: Average return and average and standard deviation of estimation bias for Dr.Q with different dropout rates. Scores for Dr.Q are plotted as solid lines and labelled in accordance with dropout rates (e.g.,“0.2”).

A.2 DR.Q WITHOUT LAYER NORMALIZATION

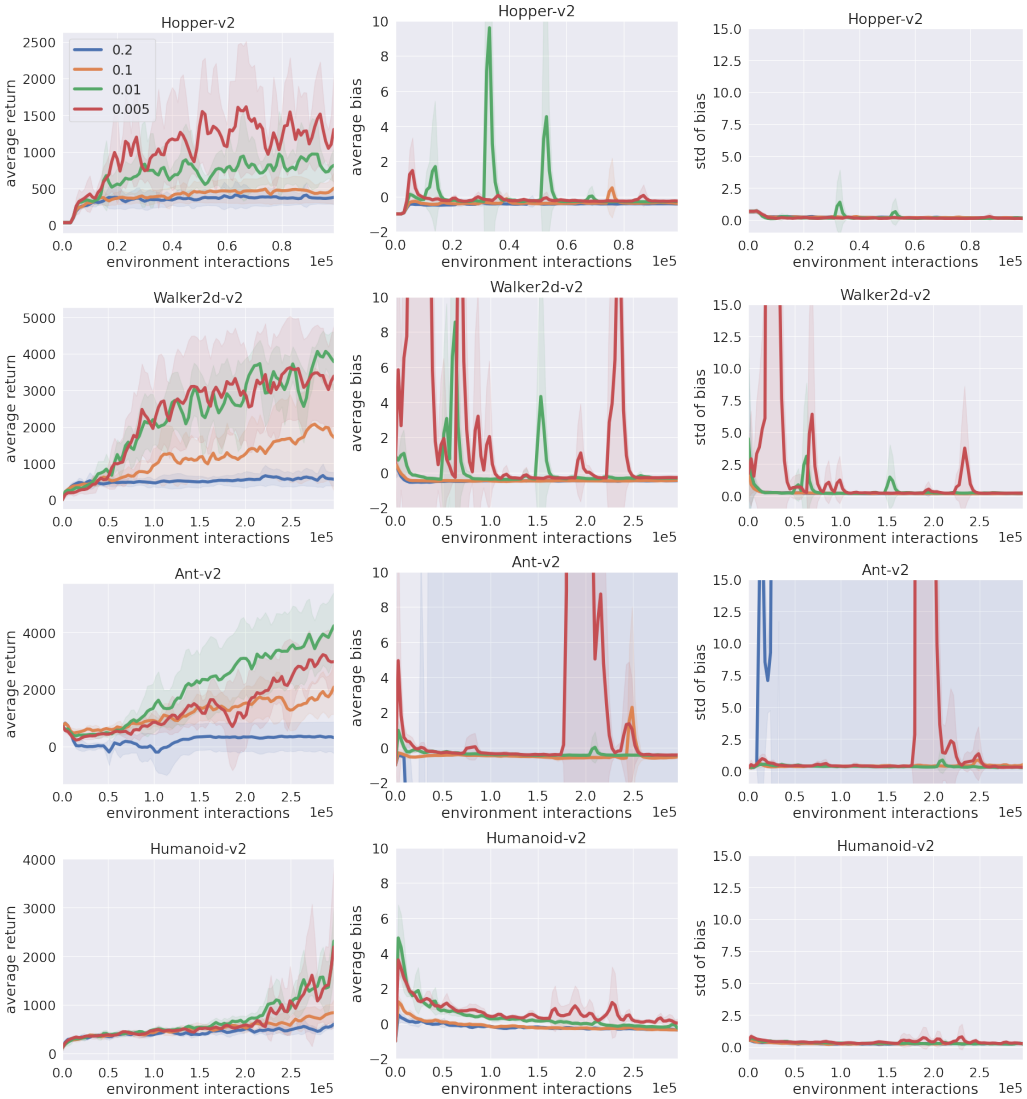

Figure 5: Average return and average and standard deviation of estimation bias for Dr.Q without layer normalization (i.e., **-LN** in Section 4.3) with different dropout rates

A.3 SIN-DR.Q: DR.Q VARIANT USING A SINGLE DROPOUT Q-FUNCTION

Dr.Q (Algorithm 2) uses an ensemble of multiple (M) dropout Q-functions. This raises the question: "Why not use a single dropout Q-function for Dr.Q?" To answer this question, we compared Dr.Q with **Sin-Dr.Q**, a variant of Dr.Q that uses a single dropout Q-function. Specifically, with Sin-Dr.Q, the target in line 6 in Algorithm 2 is calculated by evaluating $Q_{\mathrm{Dr}\bar{\phi}_1}(s', a')$, a single dropout Q-function, M times:

$$y = r + \gamma \left(\min(\underbrace{Q_{\mathrm{Dr},\bar{\phi}_1}(s', a'), ..., Q_{\mathrm{Dr},\bar{\phi}_1}(s', a')}_{M \text{ evaluation results of } Q_{\mathrm{Dr},\bar{\phi}_1}(s',a')}) - \alpha \log \pi_\theta(a'|s') \right), \quad a' \sim \pi_\theta(\cdot|s').$$

Note that, the output of $Q_{\mathrm{Dr},\bar{\phi}_1}(s', a')$ can differ in each evaluation due to the use of a dropout connection. The remaining part of Sin-Dr.Q is the same as Dr.Q.

From the comparison results of Dr.Q and Sin-Dr.Q (Figure 6), we can see that the average return of Sin-Dr.Q was lower than that of Dr.Q. This result indicates that using multiple dropout Q-functions is preferable to using a single dropout Q-function with Dr.Q.

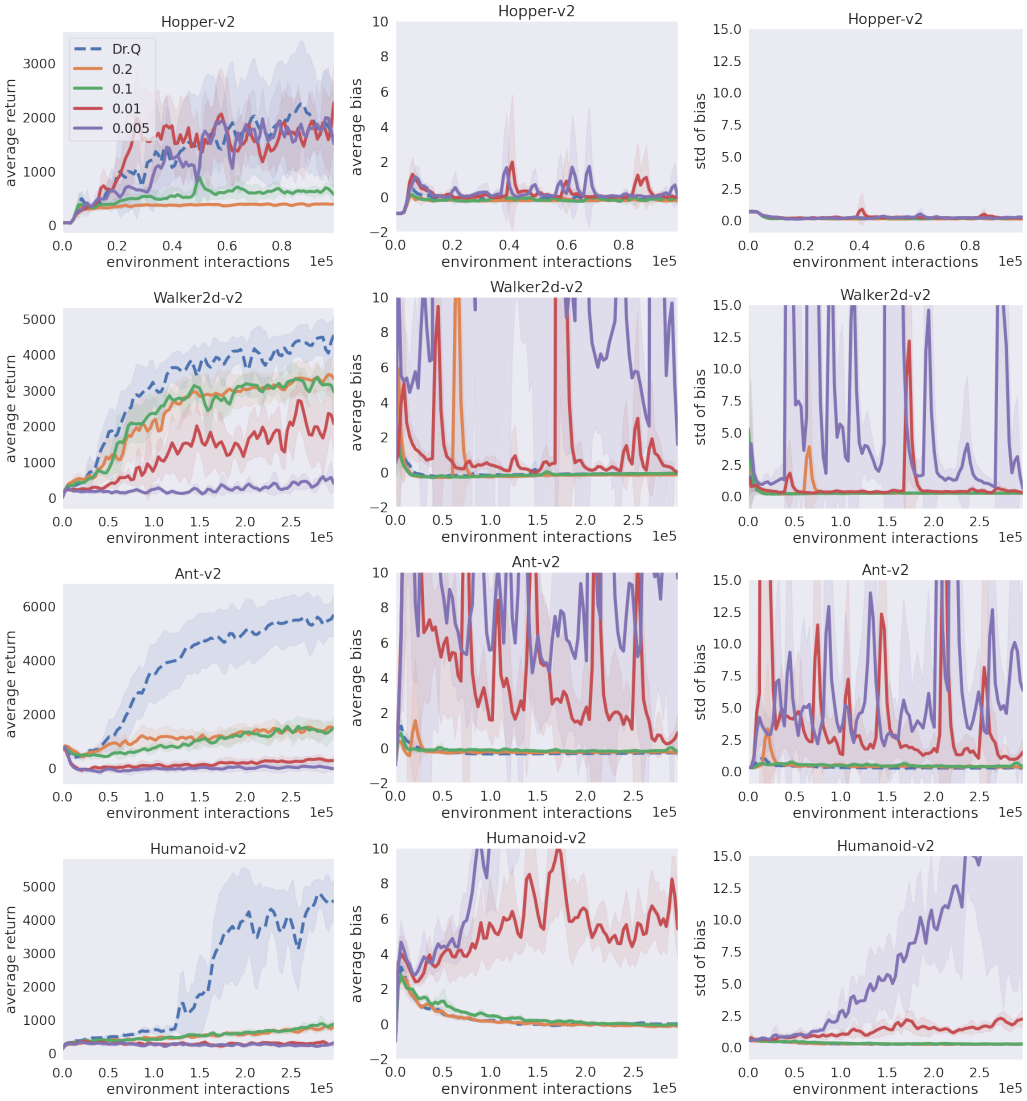

Figure 6: Average return and average and standard deviation of estimation bias for Dr.Q and Sin-Dr.Q. Scores for Dr.Q are plotted as dash lines. Scores for Sin-Dr.Q are plotted as solid lines and labelled in accordance with dropout rates (e.g.,“0.2”).

B REDQ WITH DIFFERENT ENSEMBLE SIZE N

In Section 4, we discussed comparing Dr.Q with REDQ, which uses an ensemble size of 10 (i.e., N=10). To make a more detailed comparison, we compared Dr.Q and REDQ by varying the ensemble size for REDQ. We denote REDQ that uses an ensemble size of N as "REDQN" (e.g., REDQ5 for REDQ with an ensemble size of five).

Regarding average return (left part of Figure 7), overall, Dr.Q was superior to REDQ2–5. Dr.Q was comparable with REDQ2 and REDQ3 in Hopper but superior in more complex environments (Walker, Ant, and Humanoid). In addition, Dr.Q was somewhat better than REDQ5 in all environments. Regarding estimation bias (middle and right parts of Figure 7), overall, Dr.Q was significantly better than REDQ2 and comparable with REDQ3–10. Regarding the processing speed (Table 5), Dr.Q ran as fast as REDQ3 and faster than REDQ5 by 1.4 to 1.5 times. Regarding memory efficiency (Tables 6 and 7), Dr.Q was less memory intensive than REDQ3–10.

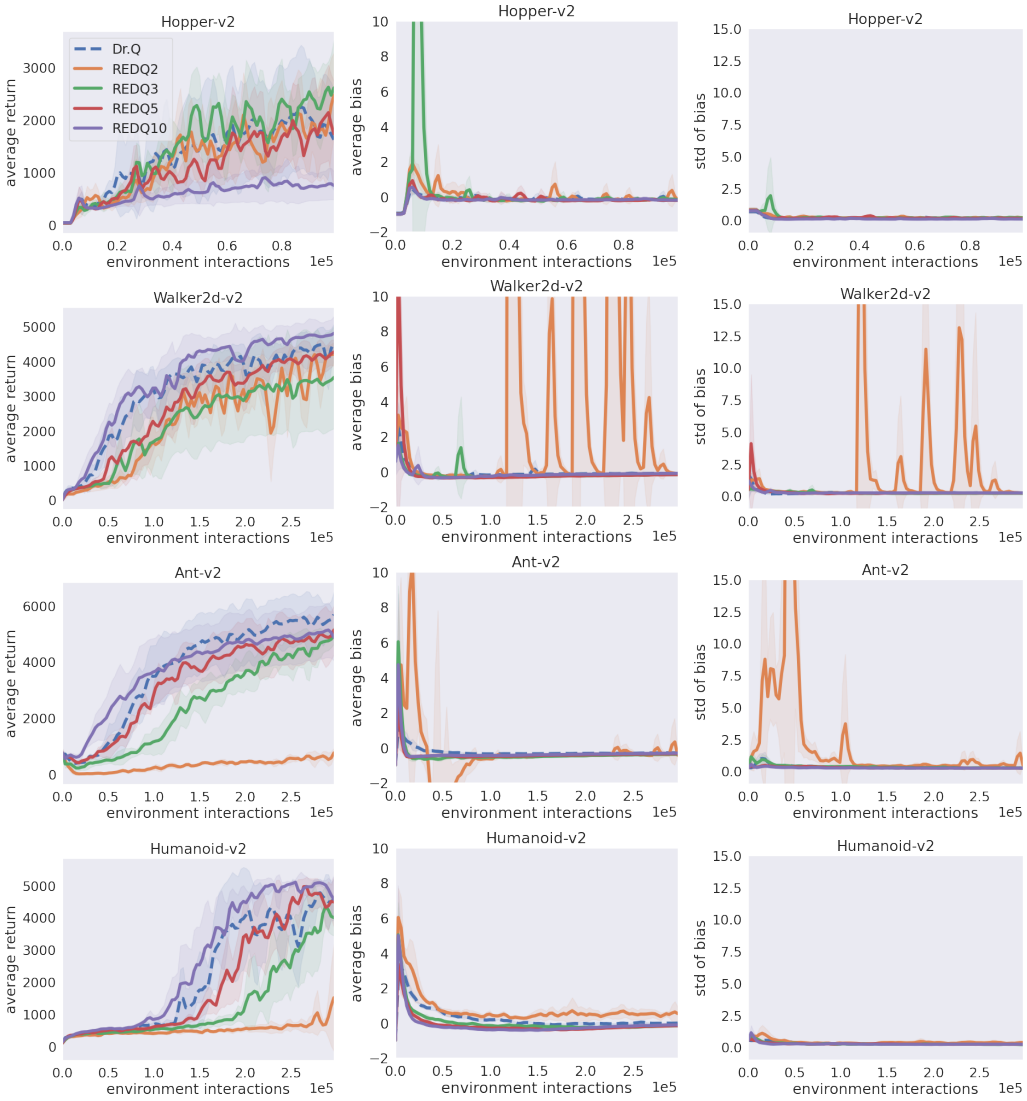

Figure 7: Average return and average and standard deviation of estimation bias for Dr.Q and REDQ. Scores for Dr.Q are plotted as dash lines. Scores for REDQ are plotted as solid lines and labelled as "REDQN".

Table 5: Process times (in msec) required for executing overall loop (e.g., lines 3–10 in Algorithm 2). Times for updating Q-functions (e.g., lines 5–9 in Algorithm 2) are shown in parentheses.

	Hopper-v2	Walker2d-v2	Ant-v2	Humanoid-v2
Dr.Q	948 (905)	933 (892)	954 (913)	989 (946)
REDQ2	832 (792)	802 (768)	675 (641)	820 (773)
REDQ3	1052 (1014)	876 (838)	919 (881)	950 (906)
REDQ5	1414 (1368)	1425 (1378)	1373 (1327)	1552 (1503)
REDQ10	2328 (2269)	2339 (2283)	2336 (2277)	2400 (2340)

Table 6: Number of parameters with methods.

	Hopper-v2	Walker2d-v2	Ant-v2	Humanoid-v2
Dr.Q	141,826	146,434	152,578	166,402
REDQ2	139,778	144,386	150,530	164,354
REDQ3	209,667	216,579	225,795	246,531
REDQ5	349,445	360,965	376,325	410,885
REDQ10	698,890	721,930	752,650	821,770

Table 7: Bottleneck memory consumption (in megabytes) with methods. Three worst bottleneck memory consumptions are shown in form of "1st worst bottleneck memory consumption / 2nd worst bottleneck memory consumption / 3rd worst bottleneck memory consumption."

	Hopper-v2	Walker2d-v2	Ant-v2	Humanoid-v2
Dr.Q	73 / 72 / 69	73 / 72 / 69	73 / 72 / 70	73 / 72 / 70
REDQ2	73 / 51 / 51	73 / 51 / 51	73 / 51 / 51	73 / 52 / 52
REDQ3	94 / 64 / 62	94 / 64 / 62	94 / 64 / 62	94 / 65 / 62
REDQ5	136 / 106 / 100	136 / 106 / 100	136 / 106 / 100	136 / 107 / 101
REDQ10	241 / 211 / 200	241 / 211 / 200	241 / 211 / 200	241 / 212 / 201

C ADDITIONAL ABLATION STUDY OF DR.Q

Dropout is introduced into three parts of the algorithm for Dr.Q (i.e., lines 6, 8 and 10 of Algorithm 2 in Section 3). In this section, we conducted an ablation study to answer the question "which dropout introduction contributes to overall performance improvement of Dr.Q?" We compared Dr.Q with its following variants:

-DO@TargetQ: A Dr.Q variant that does not use dropout in line 6. Specifically, dropout is not used in $Q_{\mathrm{Dr},\bar{\phi}_i}(s', a')$ in the following part in line 6.

$$y = r + \gamma \left(\min_{i=1,\ldots,M} Q_{\mathrm{Dr},\bar{\phi}_i}(s', a') - \alpha \log \pi_\theta(a'|s') \right), \quad a' \sim \pi_\theta(\cdot|s')$$

-DO@CurrentQ: A Dr.Q variant that does not use dropout in line 8. Specifically, dropout is not used in $Q_{\mathrm{Dr},\phi_i}(s, a)$ in the following part in line 8.

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

-DO@PolicyOpt: A Dr.Q variant that does not use dropout in line 10. Specifically, dropout is not used in $Q_{\mathrm{Dr},\phi_i}(s, a)$ in the following part in line 10.

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in \mathcal{B}} \left(\frac{1}{M} \sum_{i=1}^{M} Q_{\mathrm{Dr},\phi_i}(s, a) - \alpha \log \pi_\theta(a|s) \right), \quad a \sim \pi_\theta(\cdot|s)$$

-DO: A Dr.Q variant that does not use dropout in lines 6, 8, and 10.
In this ablation study, we compared the methods on the basis of average return and estimation bias.

The comparison results (Figure 8) indicate that the use of dropout for the target and current Q-functions (i.e., $Q_{\mathrm{Dr},\bar{\phi}_i}(s', a')$ and $Q_{\mathrm{Dr},\phi_i}(s, a)$ in lines 6 and 8) is effective. Regarding average return, the Dr.Q variants that do not use dropout for either target Q-functions in line 6 or current Q-functions in line 8 perform significantly worse than that of Dr.Q. For example, in Ant, the variants that do not use dropout for target Q-functions (-DO@TargetQ and -DO) perform much worse than Dr.Q. In addition, in Humanoid, the variants that do not use dropout for current Q-functions (-DO@CurrentQ and -DO) perform much worse than Dr.Q. Regarding estimation bias, that of the variants that do not use dropout for target Q-functions (-DO@TargetQ and -DO) is significantly worse than that of Dr.Q in all environments.

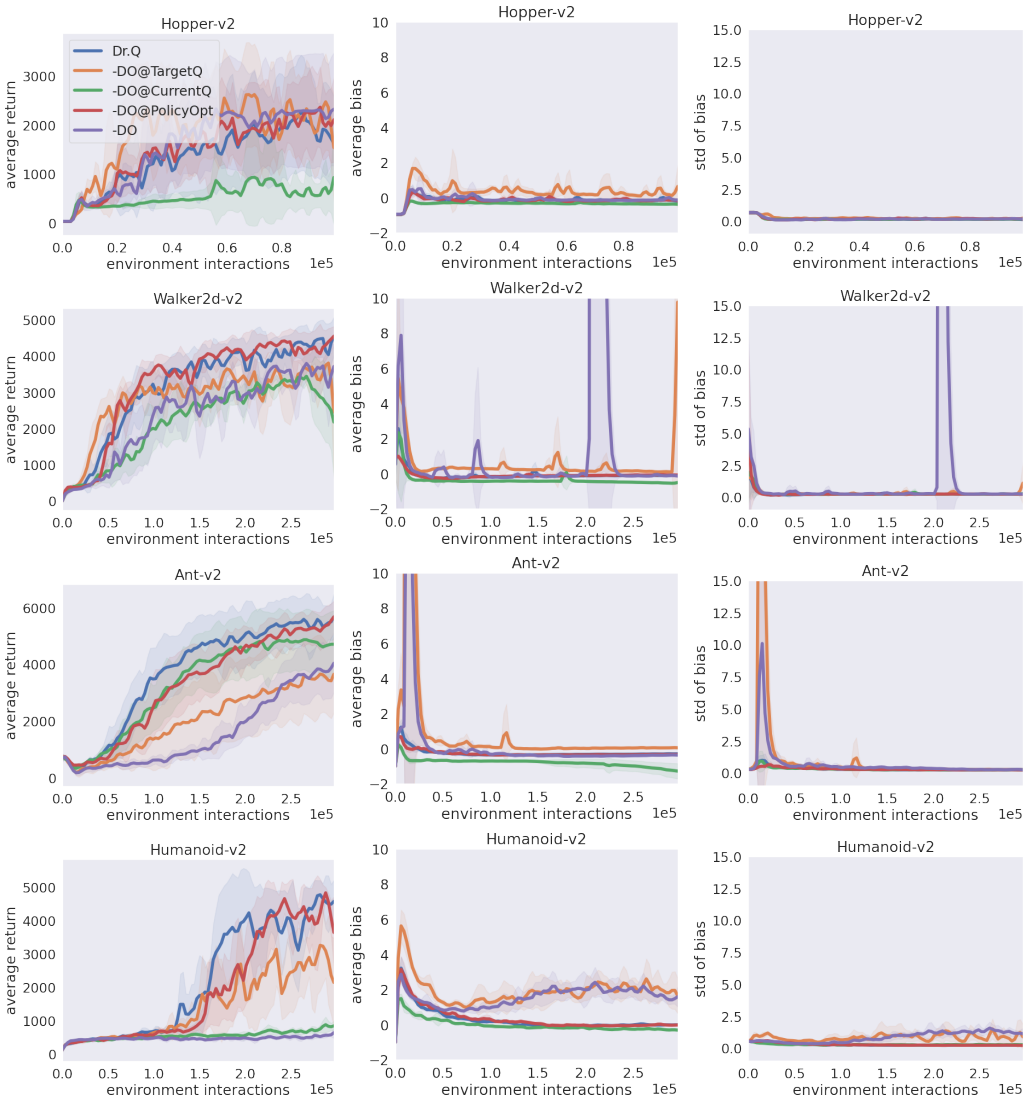

Figure 8: Additional ablation study result

D HYPERPARAMETER SETTINGS

The hyperparameter settings for each method in the experiments discussed in Section 4 are listed in Table 8. Parameter values, except for (i) dropout rate for Dr.Q and DUVN and (ii) M for DUVN, were set according to Chen et al. (2021b). The dropout rate (i) was set through line search, and M for DUVN (ii) was set according to Harrigan (2016); Moerland et al. (2017).

Table 8: Hyperparameter settings

Method	Parameter	Value
SAC, REDQ, Dr.Q, and DUVN	optimizer	Adam (Kingma & Ba, 2015)
	learning rate	$3 \cdot 10^2$
	discount rate (γ)	0.99
	target-smoothing coefficient (ρ)	0.005
	replay buffer size	10^6
	number of hidden layers for all networks	2
	number of hidden units per layer	256
	mini-batch size	256
	random starting data	5000
	UTD ratio G	20
REDQ and Dr.Q	in-target minimization parameter M	2
REDQ	ensemble size N	10
Dr.Q and DUVN	dropout rate	0.01
DUVN	in-target minimization parameter M	1