Policy Regularization with Dataset Constraint for Offline Reinforcement Learning

Yuhang Ran [*1] Yi-Chen Li [*1] Fuxiang Zhang [1 2] Zongzhang Zhang [1] Yang Yu [1 2]

Abstract

We consider the problem of learning the best possible policy from a fixed dataset, known as offline Reinforcement Learning (RL). A common taxonomy of existing offline RL works is policy regularization, which typically constrains the learned policy by distribution or support of the behavior policy. However, distribution and support constraints are overly conservative since they both force the policy to choose similar actions as the behavior policy when considering particular states. It will limit the learned policy's performance, especially when the behavior policy is sub-optimal. In this paper, we find that regularizing the policy towards the nearest state-action pair can be more effective and thus propose **P**olicy **R**egularization with **D**ataset **C**onstraint (PRDC). When updating the policy in a given state, PRDC searches the entire dataset for the nearest state-action sample and then restricts the policy with the action of this sample. Unlike previous works, PRDC can guide the policy with proper behaviors from the dataset, allowing it to choose actions that do not appear in the dataset along with the given state. It is a softer constraint but still keeps enough conservatism from out-of-distribution actions. Empirical evidence and theoretical analysis show that PRDC can alleviate offline RL's fundamentally challenging value overestimation issue with a bounded performance gap. Moreover, on a set of locomotion and navigation tasks, PRDC achieves state-of-the-art performance compared with existing methods. Code is available at `https://github.com/LAMDA-RL/PRDC`.

[*]Equal contribution [1]National Key Laboratory for Novel Software Technology, Nanjing University [2]Polixir Technologies. Correspondence to: Zongzhang Zhang <zzzhang@nju.edu.cn>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

1. Introduction

Online Reinforcement Learning (RL) has shown remarkable success in a variety of domains such as games (Silver et al., 2017), robotics (Li et al., 2021), and recommendation systems (Chen et al., 2018). However, learning an optimal policy online demands continual and possibly huge environmental interactions because of *trial-and-error* (Sutton & Barto, 2018). For expense or safety concerns, this may be impractical in real-world applications. On the other hand, offline RL learns from a fixed, previously collected dataset, thus eliminating the need for additional interactions during training. Due to the promise of turning datasets into powerful decision-making engines, offline RL has attracted significant interest in recent years (Levine et al., 2020).

One of the fundamental challenges of offline RL is value overestimation in Out-Of-Distribution (OOD) actions (see Section 2.3). According to the methodology of dealing with OOD actions, existing works on offline RL could be roughly categorized into the following two taxonomies (Jin et al., 2021): (i) Pessimistic value-based approaches that learn an underestimated or conservative value to discourage choosing OOD actions (Kumar et al., 2020; Yu et al., 2021; An et al., 2021; Lyu et al., 2022; Yang et al., 2022). (ii) Regularized policy-based approaches that constrain the policy to avoid visiting the states and actions that are less covered by the dataset (Fujimoto et al., 2019; Nair et al., 2020; Kostrikov et al., 2021a;b; Fujimoto & Gu, 2021; Wu et al., 2022).

Our work focuses on policy regularization. Generally, previous policy regularization approaches have constrained the learned policy by either the distribution (Wu et al., 2019) or the support (Kumar et al., 2019) of the behavior policy. Considering a particular state, however, distribution and support constraints are overly conservative since they both restrict the policy by actions from the behavior policy. It will limit the performance of the policy, especially when the actions from the behavior policy in the dataset are not optimal for the given state. Nevertheless, there are far more actions in the dataset than in a particular state. A natural question thus arises: *Can we guide the policy by all actions in the dataset rather than the limited ones in a given state?*

It motivates us to propose a new approach on policy regularization. When updating the policy in a particular state, our method will search the dataset for the nearest neighbor of the state-action pair, where the action comes from the policy's prediction. Then we will constrain the policy toward the action of the nearest neighbor. This novel constraint can be interpreted as minimizing the *point-to-set* distance between the state-action pair and the dataset. Thus it is neither a distribution constraint nor a support constraint but a *dataset constraint* method.

One benefit of our proposed dataset constraint is that it can relieve excessive pessimism from sub-optimal behaviors of the behavior policy, allowing the policy to choose better actions that do not appear in the dataset along with the given state but still keeping sufficient conservatism from OOD actions. We name our method **P**olicy **R**egularization with **D**ataset **C**onstraint (PRDC). PRDC can be combined with any actor-critic algorithm, and we instantiate a practical algorithm upon TD3 (Fujimoto et al., 2018) with a highly efficient implementation (Section 3). Empirical evidence and theoretical analysis show that PRDC is able to effectively alleviate the fundamentally challenging value overestimation issue of offline RL with a bounded performance gap (Section 3). On the Gym and AntMaze tasks from D4RL (Fu et al., 2020), PRDC achieves state-of-the-art performance compared with previous methods (Section 5).

2. Preliminaries

This section will briefly introduce the background, problem setting, and some notations.

2.1. Reinforcement Learning

We consider the infinite-horizon Markov Decision Process (MDP) defined by a tuple $(\mathcal{S}, \mathcal{A}, p_0, \mathcal{P}, r, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, p_0 is the initial state distribution, $\mathcal{P}: \mathcal{S} \times \mathcal{A} \to \Delta_{\mathcal{S}}$ is the transition function[1], $r: \mathcal{S} \to \mathbb{R}$ is the reward function, and $\gamma \in [0, 1)$ is a discount factor. This paper considers *deterministic* policies and continuous state and action spaces. We assume $\forall (s,a) \in \mathcal{S} \times \mathcal{A}$, $|r(s)| \le R_{\max}$, and $a \in [-A, A]$.

Given the MDP and the agent's policy $\pi: \mathcal{S} \to \mathcal{A}$, the whole decision process runs as follows: At time step $t \in \mathbb{N}$, the agent perceives the environment state s_t; then decides to take action $a_t = \pi(s_t)$, resulting in the environment to transit to the next state s_{t+1} and return a reward $r(s_t)$ to the agent. Let $J(\pi)$ be the expected discounted reward of π,

$$J(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t)\right], \tag{1}$$

where the expectation takes over the randomnesses of the initial state distribution p_0 and the transition function $\mathcal{P}$. The objective of RL is to learn an optimal policy π^* that has maximal expected discounted reward, i.e.,

$$\pi^* = \arg\max_{\pi} J(\pi).$$

Let $Q^\pi: \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ be the state-action value function (or Q-function),

$$Q^\pi(s,a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t) \middle| s_0 = s, a_0 = a\right].$$

We further define the occupancy measure $d^\pi: \mathcal{S} \to \mathbb{R}$ of π,

$$d^\pi(s') = (1-\gamma) \int_{\mathcal{S}} \sum_{t=0}^{\infty} \gamma^t p_0(s) p(s \to s', t, \pi) \mathrm{d}s,$$

where $p(s \to s', t, \pi)$ denotes the density at state s' after taking t steps from state s under policy π. Then Equation (1) could be reformulated (Xiong et al., 2022) as

$$J(\pi) = \frac{1}{1-\gamma} \mathbb{E}_{s \sim d^\pi(s)}[r(s)]. \tag{2}$$

2.2. Offline Reinforcement Learning

As stated in the above subsection, the classical setting of RL requires interactions with the environment during training. However, interaction is sometimes not allowed, especially when the task demands highly in safety or cost. To this end, offline RL, a.k.a., batch RL or data-driven RL, considers learning in an offline manner. Formally speaking, let $\mathcal{D} = \{(s, a, s', r, d)\}$ denote the set of transitions collected by a behavior policy μ, where s, a, s', and r are state, action, next state, and reward, respectively; d is a done flag indicating whether s' is a terminal state[2]. The goal of offline RL is to learn the best possible policy from $\mathcal{D}$ without further interactions (Ernst et al., 2005).

2.3. Value Overestimation Issue of Offline RL

We often use the following *one-step* Temporal Difference (TD) update (Sutton & Barto, 2018) to approximate Q^π,

$$\hat{Q}^\pi(s,a) \leftarrow \hat{Q}^\pi(s,a) + \eta \delta_t, \tag{3}$$

where $\delta_t = \left[r(s) + \gamma(1-d)\hat{Q}^\pi(s', a') - \hat{Q}^\pi(s,a)\right]$, $a' = \pi(s')$ and η is a hyper-parameter controlling the step size. With sufficiently enough samples, $\hat{Q}^\pi$ will converge to Q^π (Singh et al., 2000). But in offline RL, the dataset $\mathcal{D}$ is limited, with partial coverage of the state-action space. Thus, (s', a') may not exist in $\mathcal{D}$ (a.k.a., distribution shift) because a' is predicted by the learned policy π, not the behavior

[1]We use Δ_X to denote the set of distributions over X.

[2]In infinite-horizon MDPs, we also use d to denote whether s' is an absorbing state since there may be no terminal states.

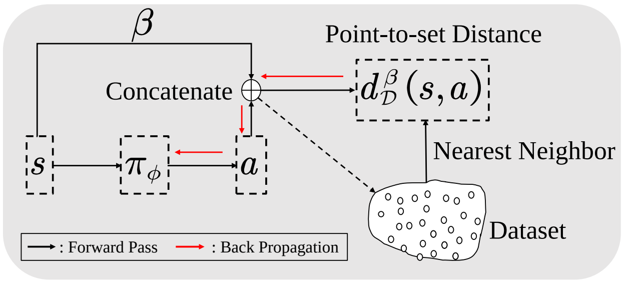


Figure 1. Illustration of the forward calculation and back propagation of our proposed policy regularization with dataset constraint.

policy μ. If $\hat{Q}^\pi(s',a')$ is overestimated, the error will continuously backpropagate to the updates of $\hat{Q}^\pi$, eventually causing $\hat{Q}^\pi$ to have overly large outputs for any input state-action. It is known as the value overestimation issue. Policy regularization (Wu et al., 2019) has been proven to be effective in tackling the value overestimation issue. We can broadly categorize existing works on policy regularization into distribution constraint and support constraint.

3. Our Method

We now introduce our method, **P**olicy **R**egularization with **D**ataset **C**onstraint (PRDC). First, we will begin by defining our proposed dataset constraint objective. Then, we will instantiate a practical algorithm. Finally, we will give a theoretical analysis of why PRDC works in offline RL and a bound of the performance gap.

3.1. Dataset Constraint

The basic motivation of our *Dataset Constraint (DC)* is to allow the policy π to choose optimal actions from all actions in the offline dataset $\mathcal{D}$. Since either distribution constraint or support constraint regularizes π by only selecting actions from the same state in the dataset, DC empowers a better generalization ability on π. However, as claimed in Section 2.3, we still have to impose enough conservatism on π to avoid the value overestimation issue. This trade-off needs to be carefully balanced.

Definition 3.1 (Point-to-set distance). Given the offline dataset $\mathcal{D}$, for any state-action pair $(s,a)\in\mathcal{S}\times\mathcal{A}$, we define its point-to-set distance to $\mathcal{D}$ as

$$d_{\mathcal{D}}^{\beta}(s,a)=\min_{(\hat{s},\hat{a})\in\mathcal{D}}\|(\beta s)\oplus a-(\beta\hat{s})\oplus\hat{a}\|,$$

where $\oplus$ denotes the vector concatenation operation and β is a hyper-parameter trading off the differences in s and a.

Based on Definition 3.1, we give the following objective:

$$\min_{\phi}\mathcal{L}_{\mathrm{DC}}(\phi):=\mathbb{E}_{s\sim\mathcal{D}}\left[d_{\mathcal{D}}^{\beta}\left(s,\pi_\phi(s)\right)\right], \tag{4}$$

where ϕ denotes parameters of the policy π_ϕ. Figure 1 illustrates the forward calculation and back propagation when using stochastic gradient descent (Ruder, 2016) to optimize Equation (4). That is, we will regularize π_ϕ by minimizing the distance between $(s,\pi_\phi(s))$ and its nearest neighbor in $\mathcal{D}$. This is essentially different from the distribution constraint and the support constraint, where the former forces π to be similar to μ and the latter requires $\pi(s)$ to be supported by $\mu(\cdot|s)$, since the state of the retrieved nearest neighbor may not be the same as the given state.

We note that β is a key hyper-parameter, controlling the strength of conservatism. Intuitively, when $\beta\to\infty$ or the state space is high-dimensional (e.g., image), Equation (4) will be dominated by the difference in s, and it reduces to behavioral cloning (Pomerleau, 1991), which has been proved to be effective on some tasks in TD3+BC (Fujimoto & Gu, 2021); when $\beta\to 0$, Equation (4) will ignore the difference in s and it reduces to regularizing π such that $\pi(s)$ is close to at least one action in $\mathcal{D}$. However, in-distribution actions are coupled with states. Thus this reduced regularization may not sufficiently constrain the policy from OOD actions. With a proper β, our state-aware regularization will allow π to learn optimal actions from a different state but still maintain enough conservatism.

3.2. A Practical Algorithm

Equation (4) can be combined with any modern actor-critic algorithm, such as TD3 (Fujimoto et al., 2018) or SAC (Haarnoja et al., 2018). In this paper, we choose TD3 due to its simplicity and high performance.

Let θ_1,θ_2,ϕ be the parameters of TD3's two Q-networks and the policy network, respectively; and $\theta_1',\theta_2',\phi'$ denote the corresponding target networks' parameters. TD3 uses the following TD error to update Q_{θ_1} and Q_{θ_2}:

$$\mathcal{L}_{\mathrm{TD}}(\theta_i)=\mathbb{E}_{(s,a,r,s',d)\sim\mathcal{D}}\left[\left(Q_{\theta_i}(s,a)-y(r,s',d)\right)^2\right], \tag{5}$$

where $y(r,s',d)=r+\gamma(1-d)\min_i Q_{\theta_i'}(s',a')$, $a'=\mathtt{clip}_A(\pi_{\phi'}(s')+\epsilon)$, $\epsilon\sim\mathtt{clip}_c(\mathcal{N}(0,\tilde{\sigma}^2))$[3], $i\in\{1,2\}$. $\tilde{\sigma}$ and c are two hyper-parameters for exploration. To update the policy π_ϕ, TD3 uses the loss below:

$$\mathcal{L}_{\mathrm{TD3}}(\phi)=\mathbb{E}_{s\sim\mathcal{D},\tilde{a}=\pi_\phi(s)}[-Q_{\theta_1}(s,\tilde{a})]. \tag{6}$$

Combining Equation (4) and Equation (6), we get the following policy update loss for offline RL:

$$\mathcal{L}_{\mathrm{PRDC}}(\phi)=\lambda\mathcal{L}_{\mathrm{TD3}}(\phi)+\mathcal{L}_{\mathrm{DC}}(\phi). \tag{7}$$

Following TD3+BC (Fujimoto & Gu, 2021), we set $\lambda=\frac{\alpha N}{\sum_{s_i,a_i}Q(s_i,a_i)}$ with α a hyper-parameter and N the batch size. The pseudo-code is summarized in Algorithm 1.

[3]The function $\mathtt{clip}_X(\cdot)$ clips its input into $[-X,X]$.

Algorithm 1 PRDC

```
1: Input: Initial policy parameters $\phi$, Q-function parameters $\theta_1, \theta_2$, offline dataset $\mathcal{D}$, hyper-parameters $\alpha, \beta, \tau$.
2: Set $\theta'_1 \leftarrow \theta_1, \theta'_2 \leftarrow \theta_2, \phi' \leftarrow \phi$.
3: for step $t = 1$ to $T$ do
4:     Sample a mini-batch of transitions $\{(s, a, r, s', d)\}$ from $\mathcal{D}$.
5:     Update $\theta_i, i \in \{1, 2\}$ using gradient descent with Equation (5).
6:     Use KD-Tree to find the nearest neighbor in $\mathcal{D}$ of every $(s, \pi_\phi(s))$.
7:     Update $\phi$ using gradient descent with Equation (7).
8:     Update target network with $\theta'_1 \leftarrow \tau\theta_1 + (1-\tau)\theta'_1$, $\theta'_2 \leftarrow \tau\theta_2 + (1-\tau)\theta'_2$, $\phi' \leftarrow \tau\phi + (1-\tau)\phi'$.
9: end for
```

Remark 3.2 (Highly efficient implementation). Our method requires searching $(s, \pi_\phi(s))$'s nearest neighbor in $\mathcal{D}$, which will be time-consuming if $\mathcal{D}$ is large. To speed up the search, we use KD-Tree (Bentley, 1975). KD-Tree has a time complexity of $\mathcal{O}(M \log |\mathcal{D}|)$ for every nearest neighbor retrieval in average, where M is the feature dimension size. It can greatly improve the run time.

3.3. Theoretical Analysis

We will begin with a quantitative analysis of why our method, PRDC, can alleviate the value overestimation issue. Then we will give a performance gap between the learned policy π_ϕ and the behavior policy μ.

Definition 3.3 (Lipschitz function)**.** A function f from $S \subset \mathbb{R}^m$ into $\mathbb{R}^n$ is called a Lispschitz function if there is a real constant $K \geq 0$ such that

$$\|f(x) - f(y)\| \leq K\|x - y\|,$$

for all $x, y \in S$. K is called the Lipschitz constant. Unless explicitly stated, we onward use $\|\cdot\|$ to denote the L2 norm.

We make the following assumptions about the Q-function, the behavior policy μ, and the transition function $\mathcal{P}$.

Assumption 3.4. Suppose that the Q-function we learn is a Lipschitz function with K_Q the Lispchitz constant, i.e.,

$$\|Q(s_1, a_1) - Q(s_2, a_2)\| \leq K_Q\|s_1 \oplus a_1 - s_2 \oplus a_2\|,$$

for all $(s_1, a_1), (s_2, a_2) \in \mathcal{S} \times \mathcal{A}$.

Assumption 3.5. Suppose that μ is deterministic and a Lipschitz function with K_μ the Lispchitz constant, i.e.,

$$\|\mu(s_1) - \mu(s_2)\| \leq K_\mu\|s_1 - s_2\|,$$

for all $s_1, s_2 \in \mathcal{S}$.

Assumption 3.6. $\forall a_1, a_2 \in \mathcal{A}$, there exists a positive constant $K_\mathcal{P}$ such that

$$\|\mathcal{P}(s'|s, a_1) - \mathcal{P}(s'|s, a_2)\| \leq K_\mathcal{P}\|a_1 - a_2\|,$$

for any $s, s' \in \mathcal{S}$.

Since we often use neural networks or linear models to parameterize the value function and policy, Assumption 3.4 and Assumption 3.5 can be easily satisfied (Gouk et al., 2021). Assumption 3.6 is standard in the theoretical studies of RL (Dufour & Prieto-Rumeau, 2013).

Theorem 3.7. *Let* $\max_{s\in\mathcal{S}} d^\beta_\mathcal{D}(s, \pi_\phi(s)) \leq \epsilon$, *which can be achieved by PRDC. Then with Assumption 3.4 and Assumption 3.5, we have*

$$\|Q(s, \pi_\phi(s)) - Q(s, \mu(s))\| \leq ((K_\mu + 2)/\beta + 1)\, K_Q\epsilon, \tag{8}$$

for any $s \in \mathcal{S}$.

The proof is in Appendix A. Recall that the one-step TD update in Equation (3) requires $\hat{Q}^\pi(s', \pi(s'))$ to be an approximately correct estimate of $Q^\pi(s', \pi(s'))$. (Li et al., 2022) has shown that $\hat{Q}^\pi$ will have low approximation errors on in-distribution samples. Although $(s', \mu(s'))$ may not exist in $\mathcal{D}$, we could still treat it as an in-distribution sample since $s' \in \mathcal{D}$ and $\mathcal{D}$ is constructed by μ. That is,

$$\hat{Q}^\pi(s', \mu(s')) \approx Q^\pi(s', \mu(s')). \tag{9}$$

Suppose that μ satisfies Assumption 3.5, and both Q^π and $\hat{Q}^\pi$ satisfies Assumption 3.4. Then with Theorem 3.7, we have that

$$\hat{Q}^\pi(s', \pi(s')) \approx \hat{Q}^\pi(s', \mu(s')), \tag{10}$$

$$Q^\pi(s', \pi(s')) \approx Q^\pi(s', \mu(s')). \tag{11}$$

Combining Equation (9), Equation (10), and Equation (11), we get $\hat{Q}^\pi(s', \pi(s')) \approx Q^\pi(s', \pi(s'))$. We thus conclude that PRDC can alleviate the value overestimation issue.

Theorem 3.8 (Performance gap of PRDC)**.** *With Assumption 3.5 and Assumption 3.6, let* $\max_{s\in\mathcal{S}} d^\beta_\mathcal{D}(s, \pi_\phi(s)) \leq \epsilon_\pi$ *and* $\max_{s\in\mathcal{S}} |\pi^*(s) - \mu(s)| \leq \epsilon_{\text{opt}}$, *which can be achieved by PRDC. Then we have*

$$|J(\pi^*) - J(\pi)| \leq \frac{CK_\mathcal{P}R_{\max}}{1-\gamma}\left[(1 + \frac{K_\mu}{\beta})\epsilon_\pi + \epsilon_{\text{opt}}\right], \tag{12}$$

where C *is a positive constant.*

We defer the proof to Appendix A. From Theorem 3.8, we see that the performance gap is inversely related to β. However, as discussed in 3.1, a small β may not keep the policy from OOD actions and thus degrades the performance. With a proper β, our method can reach higher performance than TD3+BC (corresponding to $\beta \to \infty$). Moreover, for any state s, if we assume that there exists another state-action pair $(\tilde{s}, \tilde{a}) \in \mathcal{D}$ such that $\tilde{a}$ is the optimal action for state s, then with a properly selected β, we can get that $(\tilde{s}, \tilde{a}) = \arg\min_{(\hat{s},\hat{a})\in\mathcal{D}} d_{\mathcal{D}}^{\beta}(s, \pi(s))$, where we use $(\hat{s}, \hat{a}) \in \mathcal{D}$ to denote all the state-action pairs in $\mathcal{D}$. Regularizing $\pi(s)$ toward $\tilde{a}$ will eventually give us an optimal policy. That is, we can obtain a **zero** performance gap between π^* and π under the assumption above.

4. Related Work

Policy regularization is a typical way in offline RL to avoid OOD actions. The basic idea is to augment an actor-critic algorithm with a penalty measuring the divergence of the policy from the offline dataset (Kostrikov et al., 2021a). One of the first policy regularization methods in offline RL is BCQ (Fujimoto et al., 2019). BCQ firstly uses CVAE (Sohn et al., 2015) to fit the behavior policy μ and then learns the policy π, which has a similar distribution to that of μ. Nevertheless, the fitting error will backpropagate to π and eventually affect π's performance. TD3+BC (Fujimoto & Gu, 2021) adds a behavioral cloning (Pomerleau, 1991) term to the policy improvement loss of TD3 (Fujimoto et al., 2018), successfully constraining π without an explicit fit of μ. Although simple, TD3+BC achieves competitive results on the Gym-MuJoCo suite of the D4RL benchmark (Fu et al., 2020), compared with state-of-the-art methods. TD3+BC shows that even a simple regularization term can achieve superior performance, which greatly inspires the designation of our method.

Both BCQ and TD3+BC constrain π to match μ's distribution. However, distribution constraint limits π's performance since it cannot distinguish the optimal actions from the poor ones. To this end, BEAR (Kumar et al., 2019) uses the maximum mean discrepancy divergence (MMD) (Gretton et al., 2012) with a Gaussian kernel as the f-divergence to constrain π. Empirically, they found that when computing MMD over a small number of samples, the sampled MMD between μ and π was similar to the MMD between the supports of μ and π. In their experiments, they showed that this support constraint could find optimal policies even when the offline dataset $\mathcal{D}$ was composed of several sub-optimal behaviors. Our method motivates from a quite different perspective. It is neither a distribution constraint method nor a support constraint method. Instead, we constrain π with the whole dataset $\mathcal{D}$. Moreover, we want to learn optimal actions from all those in $\mathcal{D}$ instead of just those in a particular state. For a thorough overview of existing offline RL methods and their difference from ours, we recommend referring to (Prudencio et al., 2022).

Apart from policy regularization, there are also some works associated with dataset constraint. (Goyal et al., 2022) augmented an RL agent with a retrieval process (parameterized as a neural network) that has direct access to a dataset of experiences. They tested the retrieval process in multi-task offline RL settings, showing that it could learn good task representations. However, it is about something other than regularizing the policy from OOD actions. (Humphreys et al., 2022) proposed to search the dataset for the current state's nearest neighbor, which would be fed as an additional input into the policy and value networks. They use SCaNN (Guo et al., 2020) for fast approximate nearest-neighbor retrieval, while we use KD-Tree to speed up the retrieval for the exact nearest neighbor.

5. Experiments

In this section, we will conduct extensive evaluations of the empirical performance of our method, PRDC. We would like to answer the following questions:

(i) *How does PRDC perform on the generally used benchmarks? (Section 5.1)*

(ii) *Can PRDC learn optimal actions, even when they only come with different states in the dataset? (Section 5.2)*

(iii) *Does PRDC really alleviate the value overestimation issue? (Section 5.3)*

(iv) *How does the additional hyper-parameter β influence PRDC's performance? (Section 5.4)*

Due to the nearest neighbor searching operation in Equation (4), PRDC may be time-consuming if not implemented well. Therefore, we present an empirical comparison of the running time of PRDC and existing methods (Section 5.5).

5.1. Main Results on Benchmark

First,we choose a set of locomotion and navigation tasks from the D4RL benchmark (Fu et al., 2020) to be the testbed of performance comparison. All datasets take the "**-v2**" version. Below is a brief introduction to the baselines:

• *BC* (Pomerleau, 1991), which uses mean squared error (MSE) minimization to regress the behavior policy.

• *BCQ* (Fujimoto et al., 2019), which firstly uses CVAE (Sohn et al., 2015) to fit μ and then learns a π similar to μ.

• *BEAR* (Kumar et al., 2019), which regularizes the policy with its MMD with a Gaussian kernel to the behavior policy.

• *AWAC* (Nair et al., 2020), which applies a KL divergence constraint in the policy improvement step.

• *CQL* (Kumar et al., 2020), which learns a conservative Q-function that lower-bounds the policy's true value.

• *IQL* (Kostrikov et al., 2021b), which uses expectile regres-

Table 1. Average normalized score over the final 10 evaluations and 5 seeds. Scores with the highest mean are highlighted.

Task Name	BC	BCQ	BEAR	AWAC	CQL	IQL	TD3+BC	SPOT	PRDC (Ours)
halfcheetah-random	0.2	8.8	15.1	—	20.0	11.2	11.0	—	**26.9** ± 1.0
hopper-random	4.9	7.1	14.2	—	8.3	7.9	8.5	—	**26.8** ± 9.3
walker2d-random	1.7	6.5	**10.7**	—	8.3	5.9	1.6	—	5.0 ± 1.2
halfcheetah-medium	42.6	47.0	41.0	43.5	44.0	47.4	48.3	58.4	**63.5** ± 0.9
hopper-medium	52.9	56.7	51.9	57.0	58.5	66.2	59.3	86.0	**100.3** ± 0.2
walker2d-medium	75.3	72.6	80.9	72.4	72.5	78.3	83.7	**86.4**	85.2 ± 0.4
halfcheetah-medium-replay	36.6	40.4	29.7	40.5	45.5	44.2	44.6	52.2	**55.0** ± 1.1
hopper-medium-replay	18.1	53.3	37.3	37.2	95.0	94.7	60.9	**100.2**	100.1 ± 1.6
walker2d-medium-replay	26.0	52.1	18.5	27.0	77.2	73.8	81.8	91.6	**92.0** ± 1.6
halfcheetah-medium-expert	55.2	89.1	38.9	42.8	91.6	86.7	90.7	86.9	**94.5** ± 0.5
hopper-medium-expert	52.5	81.8	17.7	55.8	105.4	91.5	98.0	99.3	**109.2** ± 4.0
walker2d-medium-expert	107.5	109.5	95.4	74.5	108.8	109.6	110.1	**112.0**	111.2 ± 0.6
antmaze-umaze	65.0		73.0	56.7	84.8	88.2	91.3	93.5	**98.8** ± 1.0
antmaze-umaze-diverse	55.6	61.0	61.0	49.3	43.3	66.7	54.6	40.7	**90.0** ± 6.8
antmaze-medium-play	0.0	0.0	0.0	0.0	65.2	70.4	0.0	74.7	**82.8** ± 4.8
antmaze-medium-diverse	0.0	0.0	8.0	0.7	54.0	74.6	0.0	**79.1**	78.8 ± 6.9
antmaze-large-play	0.0	6.7	0.0	0.0	18.8	43.5	0.0	35.3	**54.8** ± 10.9
antmaze-large-diverse	0.0	2.2	0.0	1.0	31.6	45.6	0.0	36.3	**50.0** ± 5.4

sion[4] to learn the policy without evaluating OOD actions.

• *TD3+BC* (Fujimoto & Gu, 2021), which adds an additional BC regularization term to TD3's policy update loss.

• *SPOT* (Wu et al., 2022), which explicitly models the support set of π and presents a density-based regularization.

We then train PRDC for 1M steps over five seeds on every dataset, with the implementation details deferred to Appendix B. The average normalized scores in the final ten evaluations are shown in Table 1, where we use the scores from (Prudencio et al., 2022) for AWAC, the scores from (Wu et al., 2022) for SPOT, and all other baselines take the results reported in (Li et al., 2022). From Table 1, we can see that PRDC performs well and achieves state-of-the-art performance in **13 out of 18** total tasks.

5.2. Generalization

To test whether PRDC can learn optimal actions even when they do not appear with the current state but only with different states in the dataset, we create a lineworld environment inspired by (Kumar, 2019). It is a deterministic environment, with 102 cells of length 1 connected from left to right. The state space is continuous and in $[0, 101]$[5], where the initial state randomly falls at $[0, 1]$. The action space is also continuous and in $[-1, 1]$, where a positive action means the agent goes right and a negative action means it goes left. At every step, the reward is 100 if the agent reaches the goal, i.e., its state falls at $(100, 101]$; otherwise the reward is 0. The episode ends only if the agent reaches the goal or the episode length exceeds 105.

Table 2. Accomplishments of BEAR, TD3+BC, and PRDC on lineworld-easy, lineworld-medium, lineworld-hard and lineworld-superhard. $\surd$ means accomplish, while $\times$ means not.

	BEAR	TD3+BC	PRDC
lineworld-easy	$\surd$	$\surd$	$\surd$
lineworld-medium	$\times$	$\surd$	$\surd$
lineworld-hard	$\times$	$\times$	$\surd$
lineworld-superhard	$\times$	$\times$	$\surd$

We collect four datasets on lineworld. They are named *lineworld-easy*, *lineworld-medium*, *lineworld-hard*, and *lineworld-superhard* in order based on their difficulties. A visual illustration and description of them are in Figure 2. We then train BEAR, TD3+BC, and PRDC for 10k steps on each of the four datasets and evaluate the learned policies for ten episodes over five seeds when the training finishes. Moreover, the method accomplishes the task only when the learned policy successfully reaches the goal in all the 5×10 evaluations. The result is shown in Table 2, where we can see that only PRDC can always learn an optimal policy even on the most difficult lineworld-superhard task.

[4]See `http://www.sp.unipg.it/surwey/events/28-tutorial.html` for a tutorial on expectile regression.

[5]We use $[a, b]$ to denote the set $\{x|a \leq x \leq b, x \in \mathbb{R}\}$ and $(a, b]$ to denote the set $\{x|a < x \leq b, x \in \mathbb{R}\}$.

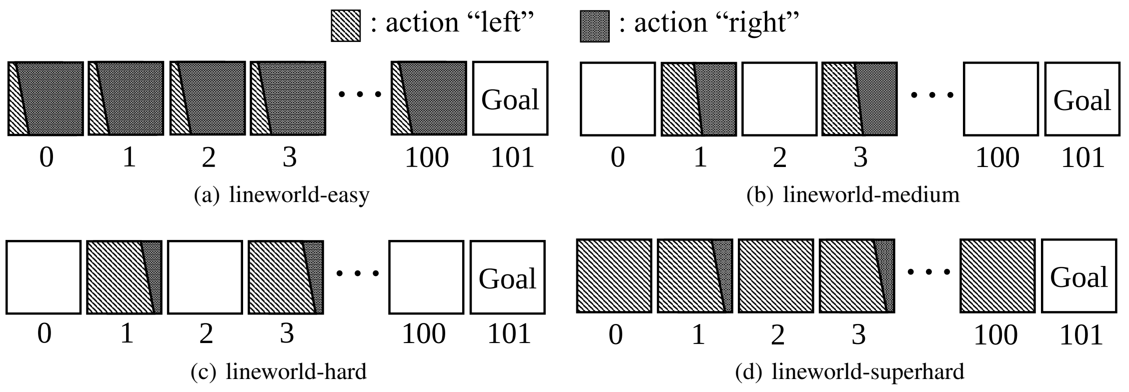


Figure 2. Compositions of four lineworld datasets. In all of the four datasets, the actions are in $\{-1, +1\}$. (a) *lineworld-easy*: the states are in $\{0, 1, 2, 3, \cdots, 100\}$, where at each state, the ratio of action -1 and $+1$ is $1:99$. (b) *lineworld-medium*: The states are in $\{1, 3, 5, 7, \cdots, 99\}$, where at each state, the ratio of action -1 and $+1$ is $1:1$. (c) *lineworld-hard*: the states are in $\{1, 3, 5, 7, \cdots, 99\}$, where at each state, the ratio of action -1 and $+1$ is $99:1$. (d) *lineworld-superhard*: the states are in $\{0, 1, 2, 3, \cdots, 100\}$. If the states are in $\{1, 3, 5, 7, \cdots, 99\}$, the ratio of action -1 and $+1$ is $99:1$. If the states are in $\{0, 2, 4, 6, \cdots, 100\}$, all actions are -1. In all the four datasets, we collect 100 samples for every state if there has actions.

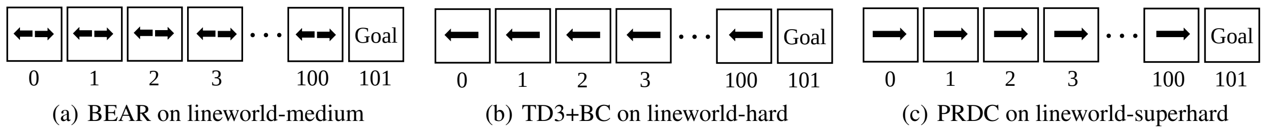

Goal
0
1
2
3
100
101
(a) BEAR on lineworld-medium
(b) TD3+BC on lineworld-hard
(c) PRDC on lineworld-superhard

Figure 3. Visualization of the learned policy by BEAR, TD3+BC, and PRDC on selected tasks. (a) On the lineworld-medium task, BEAR outputs $[-\epsilon, \epsilon]$ whatever the input state is, where ϵ is a small positive number. (b) On the lineworld-hard task, TD3+BC always outputs -1. (c) On the lineworld-superhard task, PRDC always outputs $+1$. That is, only PRDC has learned an optimal policy.

The policies learned by all three methods are visualized in Figure 3. BEAR fails on line-medium where the ratio of action -1 and $+1$ is $1:1$. BEAR always outputs actions approaching 0, the average of -1 and $+1$. The MMD distance minimization objective in BEAR will converge to action 0 if not combined with RL optimization. While in this lineworld environment, the reward is sparse, and the regularization is too conservative. Thus the RL optimization may make little difference. The same reason applies to why TD3+BC always outputs actions approaching -1 on lineworld-hard, where the ratio of action -1 and $+1$ is $99:1$. On the other hand, PRDC can learn an optimal policy that always outputs action $+1$. But from Figure 2(d), we see that there are no action $+1$ if the state is in $\{0, 2, 4, \cdots, 100\}$ and it only comes with states in $\{1, 3, 5, \cdots, 99\}$. Thanks to the softer nearest neighbor regularization, PRDC can learn optimal actions even when they never occur with the current state.

5.3. Value Estimation Error

As discussed in Section 2.3, value overestimation is a detrimental issue in offline RL. Theoretical analysis in Section 3.3 has shown that PRDC can alleviate this issue. Here is some empirical evidence. Since PRDC is built upon TD3,

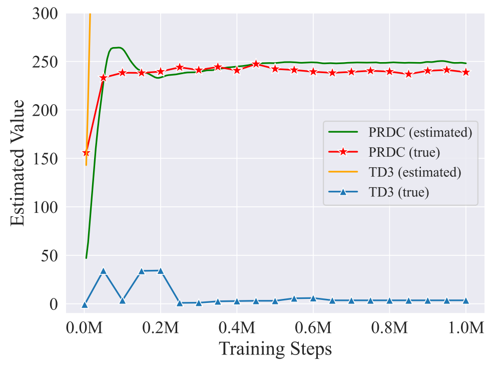


Figure 4. Comparison of PRDC's estimated value, PRDC's true value, TD3's estimated value, and TD3's true value.

we train PRDC and TD3 on the hopper-medium-v2 dataset for 1M steps. During training, we randomly sample 10 states from the initial state distribution every 5k steps and predict actions on these states by the current policy. We then get these state-action pairs' mean estimated Q-value. True values are gotten by Monte Carlo roll out (Sutton & Barto, 2018). The result is shown in Figure 4, where we see that our proposed point-to-set distance regularization effectively improves the value overestimation problem.

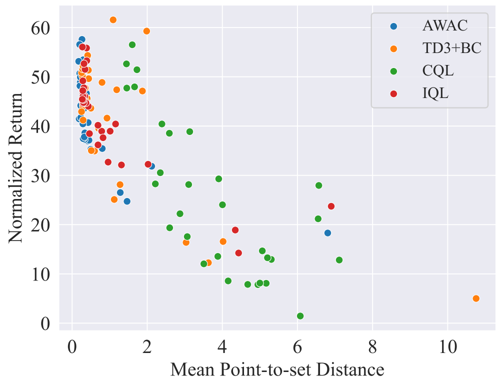


Figure 5. Normalized returns and mean point-to-set distances of baseline methods during training.

Regarding the point-to-set distance, we also observe an interesting phenomenon. We re-train AWAC, CQL, TD3+BC, and IQL on the hopper-medium-v2 dataset for 3k steps. All methods take the implementations from `https://github.com/tinkoff-ai/CORL`. During training, we evaluate the policy every 100 steps. We collect all the generated state-action pairs in the evaluation and calculate their mean point-to-set distance with $\beta = 1$. Figure 5 illustrates the result. From it we can observe an inverse relationship between the policy's normalized return and the mean point-to-set distance, which empirically motivates us that the point-to-set distance minimization objective is a reasonable target. Moreover, we speculate that this inverse relationship exists because a small mean point-to-set distance corresponds to an accurate value function, on which the policy update will return us a safely improved policy.

5.4. Sensitivity on the Hyper-parameter β

The hyper-parameter β controls how much conservatism is imposed when updating policy. To see how it influences the policy's performance, we conduct an ablation study of β on the hopper, halfcheetah, and walker2d datasets. We train PRDC on each dataset for 1M steps with $\beta \in \{0.1, 1, 2, 5, 10, 1000\}$ and keep other hyper-parameters the same. Section 5.4 shows some of the results, with the complete results deferred to Appendix C.

From Section 5.4, we can see: (i) When β is 0.1, PRDC performs bad. It is because a small β will ignore the difference in the state when searching for the nearest neighbor. However, in-distribution actions are coupled with states. Thus just constraining the policy to output an action that has occurred in the dataset may not be enough to avoid OOD actions. (ii) When β is 1000, PRDC performs not well. A large β is too conservative. (iii) When $\beta \in \{1, 2, 5\}$, PRDC performs well. We also observe that PRDC has high performance in a wide range of β on all datasets, meaning we do not need much effort to tune it.

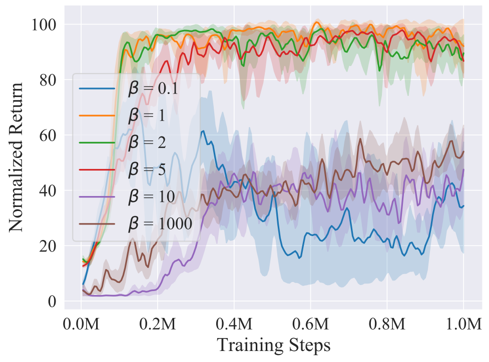


Figure 6. Performances of PRDC on hopper-medium-v2 with different β, a hyper-parameter in Definition 3.1.

5.5. Run Time

Time complexity is also a challenge in offline RL. We run PRDC and baselines on the same dataset and machine for 1M steps. Baseline implementations are the same as those in Section 5.3. The result is shown in Figure 7, where we can read that PRDC runs at an acceptable speed and even performs faster than CQL on the hopper-medium-v2 dataset. Thanks to KD-Tree, we have a highly efficient implementation of PRDC.

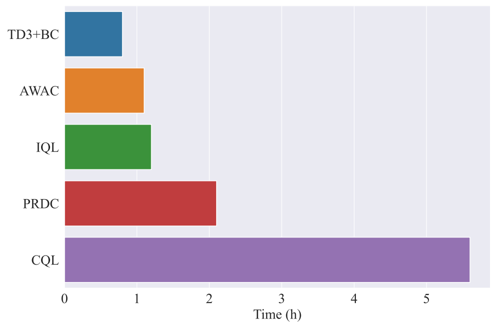


Figure 7. Run time of PRDC and baselines on hopper-medium-v2.

6. Discussion

Currently, the dataset constraint is implemented as the nearest neighbor restriction. However, our fundamental idea of dataset constraint is to leverage as much information as possible while constraining the policy into the dataset. To this end, one direct idea is to search for more than one nearest neighbor and use them together to construct a new constraint. Nevertheless, how to aggregate multiple nearest neighbors needs further consideration. We made a naive attempt by regularizing the policy towards the k nearest neighbors' *average action* with different $k \in \{1, 2, 4\}$. Due to the space limit, we defer the result to Figure 9 in Appendix C. From Figure 9, we see that this average action restriction does not improve or even hurt the performance, perhaps due to the non-optimal ones in actions of the k nearest neighbors. Indeed, the aggregation of regularization on multiple nearest neighbors is a promising direction worth in-depth study. Moreover, parameterizing the searching process as a neural network and learning to retrieve information optimally from the dataset is also worth investigating.

Although we have shown that an efficient implementation can effectively speed up PRDC in Figure 7, the nearest neighbor retrieval may still be a bottleneck to minimizing run time, especially when the dataset is large or the states are high-dimensional images. Some ideas from existing studies may help. For example, we can learn a low-dimensional representation, which is then used to calculate the point-to-set distance. We can also use methods like SCaNN for fast approximate nearest neighbor retrieval.

These further discussions are beyond the scope of this paper, and we leave them for future works.

7. Conclusion

We propose dataset constraint, a new type of policy regularization method. Unlike the commonly used distribution constraint and support constraint, which limit the policy to actions that have occurred with the current state, the proposed dataset constraint is less conservative and allows the policy to learn from all actions in the dataset. Empirical and theoretical evidence shows that the dataset constraint can effectively alleviate offline RL's fundamentally challenging value overestimation issue. It is simple and effective with new state-of-the-art performance on the D4RL (Fu et al., 2020) benchmark but only one additional hyper-parameter compared to (Fujimoto & Gu, 2021). Overall, this is the first attempt to constrain the policy from a dataset perspective in offline RL. We hope our work can inspire more relevant research as stated in Section 6.

Acknowledgements

This work is supported by the National Key R&D Program of China (2022ZD0114804), the National Science Foundation of China (62276126), and the Fundamental Research Funds for the Central Universities (14380010). The authors would like to thank Chengxing Jia, Chenxiao Gao, Chenghe Wang, and the anonymous reviewers for their support and helpful discussions on improving the paper.

References

An, G., Moon, S., Kim, J., and Song, H. O. Uncertainty-based offline reinforcement learning with diversified q-ensemble. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 7436–7447, 2021.

Bentley, J. L. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18 (9):509–517, 1975.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. OpenAI Gym. *arXiv Preprint arxiv:1606.01540*, 2016.

Chen, S., Yu, Y., Da, Q., Tan, J., Huang, H., and Tang, H. Stabilizing reinforcement learning in dynamic environment with application to online recommendation. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 1187–1196, 2018.

Dufour, F. and Prieto-Rumeau, T. Finite linear programming approximations of constrained discounted markov decision processes. *SIAM Journal on Control and Optimization*, 51(2):1298–1324, 2013.

Ernst, D., Geurts, P., and Wehenkel, L. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556, 2005.

Fu, J., Kumar, A., Nachum, O., Tucker, G., and Levine, S. D4RL: Datasets for deep data-driven reinforcement learning. *arxiv Preprint arxiv:2004.07219*, 2020.

Fujimoto, S. and Gu, S. S. A minimalist approach to offline reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 20132–20145, 2021.

Fujimoto, S., van Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning (ICML)*, pp. 1582–1591, 2018.

Fujimoto, S., Meger, D., and Precup, D. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning (ICML)*, pp. 2052–2062, 2019.

Gouk, H., Frank, E., Pfahringer, B., and Cree, M. J. Regularisation of neural networks by enforcing lipschitz continuity. *Machine Learning*, 110(2):393–416, 2021.

Goyal, A., Friesen, A. L., Banino, A., Weber, T., Ke, N. R., Badia, A. P., Guez, A., Mirza, M., Humphreys, P. C., Konyushkova, K., Valko, M., Osindero, S., Lillicrap, T. P., Heess, N., and Blundell, C. Retrieval-augmented reinforcement learning. In *International Conference on Machine Learning (ICML)*, pp. 7740–7765, 2022.

Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., and Alexander, S. A kernel two-sample test. *Journal of Machine Learning Research*, 13(1):723–773, 2012.

Guo, R., Sun, P., Lindgren, E., Geng, Q., Simcha, D., Chern, F., and Kumar, S. Accelerating large-scale inference with anisotropic vector quantization. In *International Conference on Machine Learning (ICML)*, pp. 3887–3896, 2020.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning (ICML)*, pp. 1856–1865, 2018.

Humphreys, P. C., Guez, A., Tieleman, O., Sifre, L., Weber, T., and Lillicrap, T. Large-scale retrieval for reinforcement learning. *arxiv Preprint arxiv:2206.05314*, 2022.

Jin, Y., Yang, Z., and Wang, Z. Is pessimism provably efficient for offline RL? In *International Conference on Machine Learning (ICML)*, pp. 5084–5096, 2021.

Kostrikov, I., Fergus, R., Tompson, J., and Nachum, O. Offline reinforcement learning with fisher divergence critic regularization. In *International Conference on Machine Learning (ICML)*, pp. 5774–5783, 2021a.

Kostrikov, I., Nair, A., and Levine, S. Offline reinforcement learning with implicit Q-learning. In *International Conference on Learning Representations (ICLR)*, 2021b.

Kumar, A. Data-driven deep reinforcement learning. https://bair.berkeley.edu/blog/2019/12/05/bear/, 2019.

Kumar, A., Fu, J., Soh, M., Tucker, G., and Levine, S. Stabilizing off-policy Q-learning via bootstrapping error reduction. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 11761–11771, 2019.

Kumar, A., Zhou, A., Tucker, G., and Levine, S. Conservative Q-learning for offline reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Levine, S., Kumar, A., Tucker, G., and Fu, J. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arxiv Preprint arxiv:2005.01643*, 2020.

Li, J., Zhan, X., Xu, H., Zhu, X., Liu, J., and Zhang, Y. Distance-sensitive offline reinforcement learning. In *NeurIPS'22 Workshop on Deep RL*, 2022.

Li, Z., Cheng, X., Peng, X. B., Abbeel, P., Levine, S., Berseth, G., and Sreenath, K. Reinforcement learning for robust parameterized locomotion control of bipedal robots. In *International Conference on Robotics and Automation (ICRA)*, pp. 2811–2817, 2021.

Lyu, J., Ma, X., Li, X., and Lu, Z. Mildly conservative Q-learning for offline reinforcement learning. *arxiv Preprint arxiv:2206.04745*, 2022.

Nair, A., Dalal, M., Gupta, A., and Levine, S. Accelerating online reinforcement learning with offline datasets. *arxiv Preprint arxiv:2006.09359*, 2020.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 8024–8035. 2019.

Pomerleau, D. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1): 88–97, 1991.

Prudencio, R. F., Máximo, M. R. O. A., and Colombini, E. L. A survey on offline reinforcement learning: Taxonomy, review, and open problems. *arXiv Preprint arxiv:2203.01387*, 2022.

Ruder, S. An overview of gradient descent optimization algorithms. *arxiv Preprint arxiv:1609.04747*, 2016.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T. P., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, 2017.

Singh, S., Jaakkola, T. S., Littman, M. L., and Szepesvári, C. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 38(3):287–308, 2000.

Sohn, K., Lee, H., and Yan, X. Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 3483–3491, 2015.

Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction (Second Edition)*. MIT press, 2018.

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

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

Wu, J., Wu, H., Qiu, Z., Wang, J., and Long, M. Supported policy optimization for offline reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 31278–31291, 2022.

Wu, Y., Tucker, G., and Nachum, O. Behavior regularized offline reinforcement learning. *arxiv Preprint arxiv:1911.11361*, 2019.

Xiong, H., Xu, T., Zhao, L., Liang, Y., and Zhang, W. Deterministic policy gradient: Convergence analysis. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 2159–2169, 2022.

Yang, R., Bai, C., Ma, X., Wang, Z., Zhang, C., and Han, L. RORL: robust offline reinforcement learning via conservative smoothing. *arxiv Preprint arxiv:2206.02829*, 2022.

Yu, T., Kumar, A., Rafailov, R., Rajeswaran, A., Levine, S., and Finn, C. COMBO: conservative offline model-based policy optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 28954–28967, 2021.

A. Proofs

Lemma A.1. *For $\forall a, b \in \mathbb{R}_+$, the following inequality holds,*

$$\sqrt{a^2+b^2} \leq a+b.$$

Lemma A.2 (Triangle inequality)**.** *For $\forall x, y \in \mathbb{R}^m$, the following inequality holds,*

$$\|x+y\| \leq \|x\| + \|y\|.$$

Lemma A.3. *For $\forall (s,a), (\hat{s},\hat{a}) \in \mathcal{S} \times \mathcal{A}$, if $\|(\beta s) \oplus a - (\beta \hat{s}) \oplus \hat{a}\| \leq \epsilon, \beta > 0, \epsilon > 0$, then the following inequalities hold,*

- $\|s - \hat{s}\| \leq \epsilon/\beta$,
- $\|a - \hat{a}\| \leq \epsilon$,
- $\|s \oplus a - \hat{s} \oplus \hat{a}\| \leq (1/\beta + 1)\epsilon$.

Proof. Since $\beta > 0$, we have

$$\beta\|s - \hat{s}\| \leq \|(\beta s) \oplus a - (\beta \hat{s}) \oplus \hat{a}\| \leq \epsilon.$$

By dividing both sides of the above inequality by β, we achieve $\|s - \hat{s}\| \leq \epsilon/\beta$. Similarly, we can get $\|a - \hat{a}\| \leq \epsilon$. Thus,

$$\|s \oplus a - \hat{s} \oplus \hat{a}\| \leq \|s - \hat{s}\| + \|a - \hat{a}\| \leq (1/\beta + 1)\epsilon,$$

where the first inequality comes from Lemma A.1. □

Theorem 3.7. *Let $\max_{s \in \mathcal{S}} d^{\beta}_{\mathcal{D}}(s, \pi_\phi(s)) \leq \epsilon$, which can be achieved by PRDC. Then with Assumption 3.4 and Assumption 3.5, we have*

$$\|Q(s, \pi_\phi(s)) - Q(s, \mu(s))\| \leq ((K_\mu + 2)/\beta + 1)\, K_Q \epsilon, \tag{8}$$

for any $s \in \mathcal{S}$.

Proof. Let

$$(\hat{s}, \hat{a}) = \arg\min_{(\hat{s},\hat{a}) \in \mathcal{D}} d^{\beta}_{\mathcal{D}}(s, \pi(s)),$$

where $\hat{a} = \mu(\hat{s})$. By expanding the left side of Equation (8), we get

$$\begin{aligned}
\|Q(s, \pi_\phi(s)) - Q(s, \mu(s))\| &= \|Q(s, \pi_\phi(s)) - Q(\hat{s}, \hat{a}) + Q(\hat{s}, \hat{a}) - Q(s, \mu(s))\| \\
&\overset{(i)}{\leq} \|Q(s, \pi_\phi(s)) - Q(\hat{s}, \hat{a})\| + \|Q(\hat{s}, \hat{a}) - Q(s, \mu(s))\| \\
&\overset{(ii)}{\leq} K_Q(\|s \oplus \pi(s) - \hat{s} \oplus \hat{a}\| + \|\hat{s} \oplus \mu(\hat{s}) - s \oplus \mu(s)\|) \\
&\overset{(iii)}{\leq} K_Q\left((1/\beta + 1)\epsilon + \|\hat{s} \oplus \mu(\hat{s}) - s \oplus \mu(s)\|\right) \\
&\overset{(iv)}{\leq} K_Q\left((1/\beta + 1)\epsilon + \|\hat{s} - s\| + \|\mu(\hat{s}) - \mu(s)\|\right) \\
&\overset{(v)}{\leq} K_Q\left((1/\beta + 1)\epsilon + \|\hat{s} - s\| + K_\mu\|\hat{s} - s\|\right) \\
&\overset{(vi)}{\leq} ((K_\mu + 2)/\beta + 1)\, K_Q \epsilon.
\end{aligned}$$

Here, (i) is due to Lemma A.2; (ii) is due to Assumption 3.4; (iii) is due to Lemma A.3; (iv) is due to Lemma A.1; (v) is due to Assumption 3.5; and (vi) is due to Lemma A.3. □

Lemma A.4. *For function $f : S \subset \mathbb{R}^m \to \mathbb{R}$, we have that*

$$\left|\int_a^b f(x)\mathrm{d}x\right| \leq \int_a^b |f(x)|\mathrm{d}x,$$

where $a \leq b$, and $[a,b] \subseteq S$.

Proof. For any $x \in S$, the following inequality holds,

$$-|f(x)| \leq f(x) \leq |f(x)|.$$

Thus,

$$-\int_a^b |f(x)|\mathrm{d}x \leq \int_a^b f(x)\mathrm{d}x \leq \int_a^b |f(x)|\mathrm{d}x.$$

Therefore,

$$\left|\int_a^b f(x)\mathrm{d}x\right| \leq \int_a^b |f(x)|\mathrm{d}x.$$

Thus, we finish the proof. □

Lemma A.5 (Lemma 1 of (Xiong et al., 2022))**.** *With Assumption 3.6, the following inequality holds,*

$$\int_{\mathcal{S}} |d^\pi(s) - d^\mu(s)|\mathrm{d}s \leq CK_{\mathcal{P}} \max_{s\in\mathcal{S}} \|\pi(s) - \mu(s)\|,$$

where C is a positive constant.

Proof. Please see the appendix of (Xiong et al., 2022). □

Lemma A.6. *Let $\max_{s\in\mathcal{S}} d^\beta_{\mathcal{D}}(s, \pi_\phi(s)) \leq \epsilon$, which can be acvieved by PRDC. Then with Assumption 3.5, we have*

$$\|\pi(s) - \mu(s)\| \leq (1 + \frac{1}{K_\mu}\beta)\epsilon, \forall s \in \mathcal{S},$$

Proof. We follow a similar proof of Theorem 3.7. Let

$$(\hat{s}, \hat{a}) = \operatorname*{arg\,min}_{(\hat{s},\hat{a})\in\mathcal{D}} d^\beta_{\mathcal{D}}(s, \pi(s)),$$

where $\hat{a} = \mu(\hat{s})$. Then,

$$\begin{aligned}\|\pi(s) - \mu(s)\| &= \|\pi(s) - \mu(\hat{s}) + \mu(\hat{s}) - \mu(s)\| \\ &\overset{(i)}{\leq} \|\pi(s) - \mu(\hat{s})\| + \|\mu(\hat{s}) - \mu(s)\| \\ &\overset{(ii)}{\leq} \|\pi(s) - \mu(\hat{s})\| + K_\mu\|\hat{s} - s\| \\ &\overset{(iii)}{\leq} (1 + \frac{K_\mu}{\beta})\epsilon.\end{aligned}$$

Here, (i) is due to Lemma A.2; (ii) is due to Assumption 3.5; and (iii) is due to Lemma A.3. □

Theorem 3.8 (Performance gap of PRDC)**.** *With Assumption 3.5 and Assumption 3.6, let $\max_{s\in\mathcal{S}} d^\beta_{\mathcal{D}}(s, \pi_\phi(s)) \leq \epsilon_\pi$ and $\max_{s\in\mathcal{S}} |\pi^*(s) - \mu(s)| \leq \epsilon_{\text{opt}}$, which can be achieved by PRDC. Then we have*

$$|J(\pi^*) - J(\pi)| \leq \frac{CK_{\mathcal{P}}R_{\max}}{1-\gamma}\left[(1 + \frac{K_\mu}{\beta})\epsilon_\pi + \epsilon_{\text{opt}}\right], \tag{12}$$

where C is a positive constant.

Proof. With Lemma A.2, we have

$$\begin{aligned}|J(\pi^*) - J(\pi)| &= |J(\pi^*) - J(\mu) + J(\mu) - J(\pi)| \\ &\leq |J(\pi^*) - J(\mu)| + |J(\pi) - J(\mu)|.\end{aligned} \tag{13}$$

We first bound $|J(\pi) - J(\mu)|$. By taking into Equation (2), we get

$$\begin{aligned}
|J(\pi) - J(\mu)| &= \left| \frac{1}{1-\gamma} \mathbb{E}_{s\sim d^{\pi}(s)}[r(s)] - \frac{1}{1-\gamma} \mathbb{E}_{s\sim d^{\mu}(s)}[r(s)] \right| \mathrm{d}s \\
&= \frac{1}{1-\gamma} \left| \int_{\mathcal{S}} (d^{\pi}(s) - d^{\mu}(s)) r(s) \mathrm{d}s \right| \\
&\overset{(i)}{\leq} \frac{1}{1-\gamma} \int_{\mathcal{S}} |d^{\pi}(s) - d^{\mu}(s)| |r(s)| \mathrm{d}s \\
&\leq \frac{R_{\max}}{1-\gamma} \int_{\mathcal{S}} |d^{\pi}(s) - d^{\mu}(s)| \mathrm{d}s. \\
&\overset{(ii)}{\leq} \frac{C K_{\mathcal{P}} R_{\max}}{1-\gamma} \max_{s\in\mathcal{S}} \|\pi(s) - \mu(s)\| \\
&\overset{(iii)}{\leq} \frac{C K_{\mathcal{P}} R_{\max}}{1-\gamma} (1 + \frac{K_{\mu}}{\beta}) \epsilon_{\pi}.
\end{aligned} \tag{14}$$

Here, (i) is due to Lemma A.4; (ii) is due to Lemma A.5; and (iii) is due to Lemma A.6.

Similarly, we have that

$$|J(\pi^*) - J(\mu)| \leq \frac{C K_{\mathcal{P}} R_{\max}}{1-\gamma} \epsilon_{\text{opt}}. \tag{15}$$

Thus,

$$|J(\pi^*) - J(\pi)| \leq \frac{C K_{\mathcal{P}} R_{\max}}{1-\gamma} \left[(1 + \frac{K_{\mu}}{\beta}) \epsilon_{\pi} + \epsilon_{\text{opt}} \right]. \tag{16}$$

The proof is finished. □

B. Implementation Details

We implement PRDC based on the author-provided implementation of TD3+BC[6], and the KD-tree implementation comes from SciPy (Virtanen et al., 2020). The full hyper-parameters setting is in Table 3.

B.1. Software

We use the following software versions:

- Python 3.8
- MuJoCo 2.2.0 (Todorov et al., 2012)
- Gym 0.21.0 (Brockman et al., 2016)
- MuJoCo-py 2.1.2.14
- PyTorch 1.12.1 (Paszke et al., 2019)

B.2. Hardware

We use the following hardware:

- NVIDIA RTX A4000
- 12th Gen Intel(R) Core(TM) i9-12900K

[6] https://github.com/sfujim/TD3_BC

C. More Results

C.1. Sensitivity on the Hyper-paramter β

This section investigates how β influences the policy's performance. On the hopper, halfcheetah, and walker2d (-random-v2, -medium-replay-v2, -medium-v2, -medium-expert-v2) datasets, we train PRDC for 1M steps over 5 seeds with $\beta \in \{0.1, 1, 2, 5, 10, 1000\}$ and keep other hyper-parameters the same. The result is shown in Figure 8. Apart from the discussion in Section 5.4, we notice that PRDC performs better on the random datasets when $\beta \in \{0.1, 1, 2\}$ than when $\beta \in \{5, 10, 1000\}$. The reason is that a bigger β makes the regularization in Equation (4) more like behavior cloning (Pomerleau, 1991). However, on the random datasets, behavior cloning will return a poor policy. On the other hand, a smaller β will leave more room for RL optimization.

C.2. k Nearest Neighbor Constraint

This section investigates whether more than one nearest-neighbor constraint can improve the policy's performance. To this end, we modify the regularization defined in Equation (4) to minimize the distance between $\pi(s)$ and the average action of $(s, \pi(s))$'s k nearest neighbors. That is, we define a new regularization as

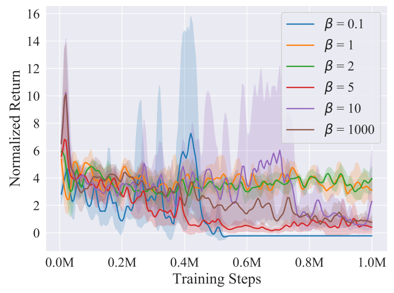
$$\min_{\phi} \mathcal{L}(\phi) := \|\pi(s) - \overline{a}\|, \tag{17}$$

where $\overline{a}$ denotes the average action of $(s, \pi(s))$'s k nearest neighbors.

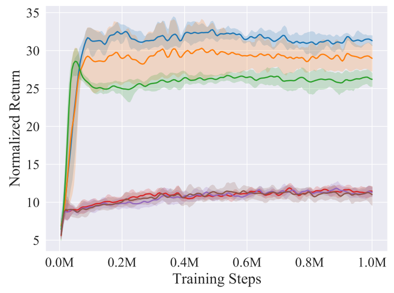
We choose $k \in \{1, 2, 4\}$ and train PRDC for 1M steps on the hopper, halfcheetah, and walker2d (-random-v2, -medium-replay-v2, -medium-v2, -medium-expert-v2) datasets. The result is shown in Figure 9, where we see that this average action regularization does not improve the performance on almost the tasks. The learning curves become much more unstable, where we speculate that it is because there are non-optimal ones in the actions of the k nearest neighbors.

Table 3. Full hyper-parameters setting of PRDC.

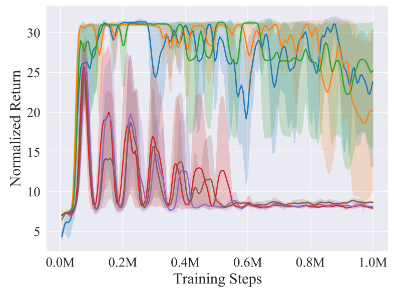
	Hyper-parameters	Value
Network	Actor learning rate	3×10^{-4}
	Critic learning rate	3×10^{-4} for MuJoCo 1×10^{-3} for AntMaze
	Batch size	256
	Optimizer	Adam
	Q-network	3 layers ReLU activated MLPs with 256 units
	Policy Network	3 layers ReLU activated MLPs with 256 units
TD3	Critic learning rate	0.99 for MuJoCo 0.995 for AntMaze
	Number of iterations	10^6
	Target update rate τ	0.005
	Policy noise	0.2
	Policy noise clipping	0.5
	Policy update frequency	2
PRDC	Normalized state	True
	k	1
	β	2.0 for MuJoCo {2.0, 7.5, 15.0} for AntMaze {umaze, medium, large}
	α	2.5 for MuJoCo {2.5, 7.5, 20.0} for AntMaze {umaze, medium, large}



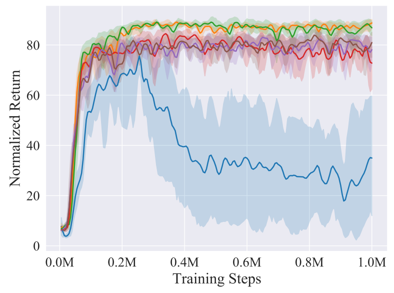
(a) walker2d-random-v2



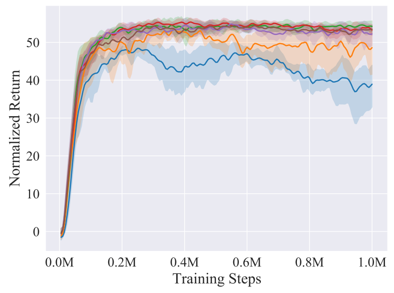
(b) halfcheetah-random-v2



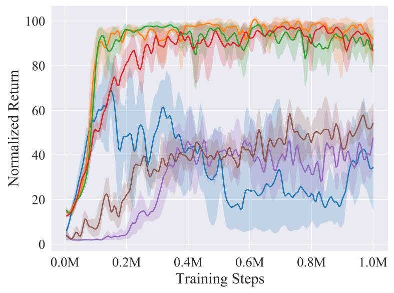
(c) hopper-random-v2



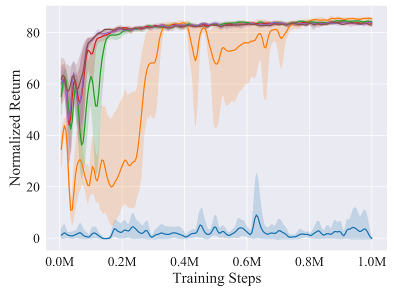
(d) walker2d-medium-replay-v2



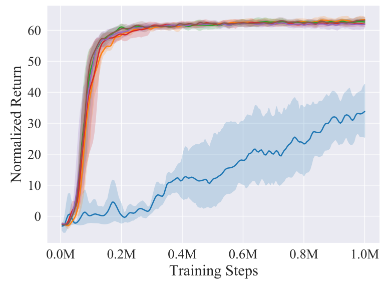
(e) halfcheetah-medium-replay-v2



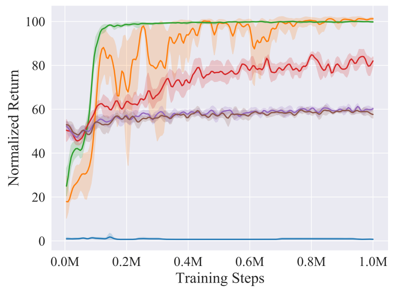
(f) hopper-medium-replay-v2



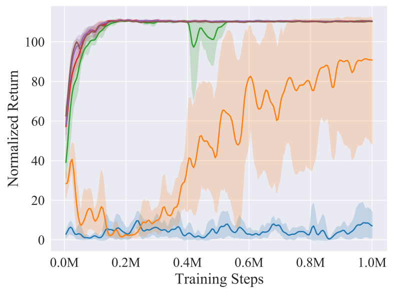
(g) walker2d-medium-v2



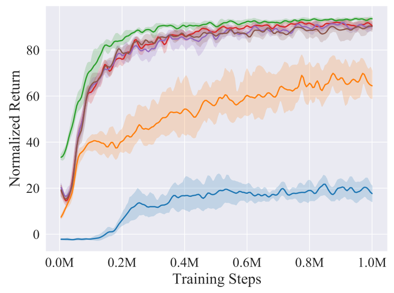
(h) halfcheetah-medium-v2



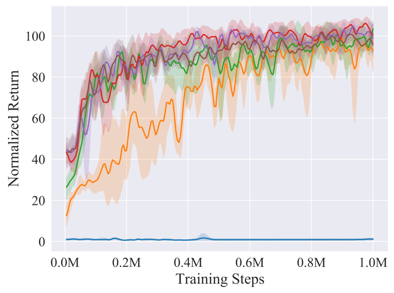
(i) hopper-medium-v2



(j) walker2d-medium-expert-v2

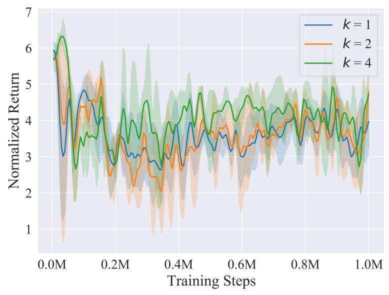


(k) halfcheetah-medium-expert-v2

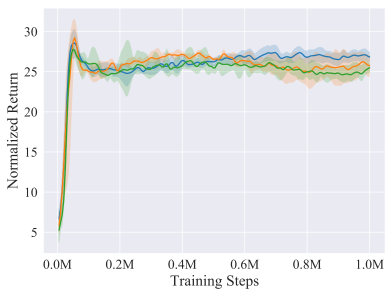


(l) hopper-medium-expert-v2

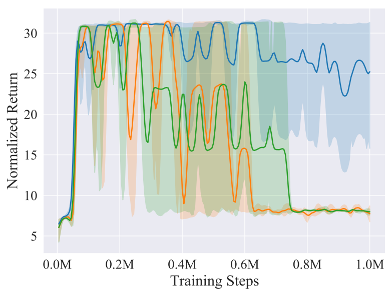
Figure 8. Performances of PRDC with different β, a hyper-parameter in Definition 3.1.



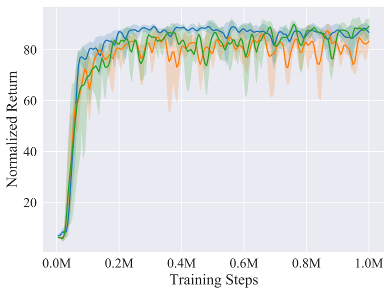
(a) walker2d-random-v2



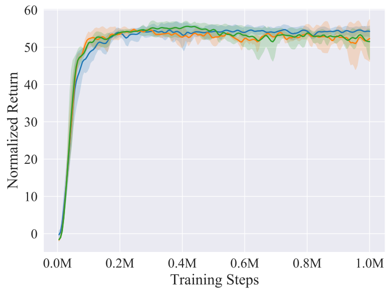
(b) halfcheetah-random-v2



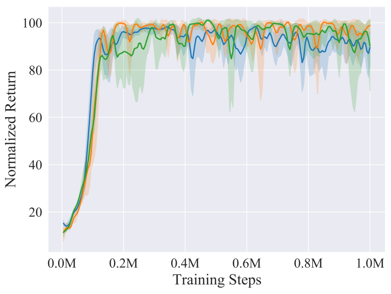
(c) hopper-random-v2



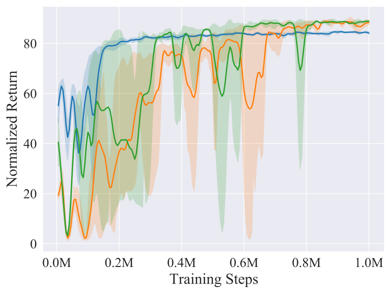
(d) walker2d-medium-replay-v2



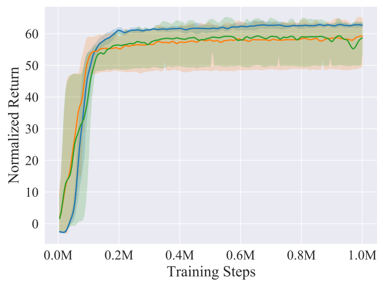
(e) halfcheetah-medium-replay-v2



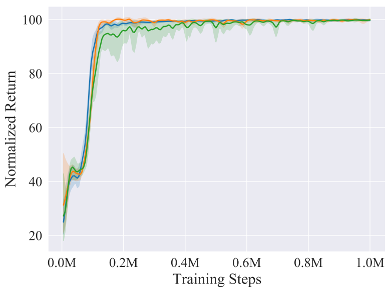
(f) hopper-medium-replay-v2



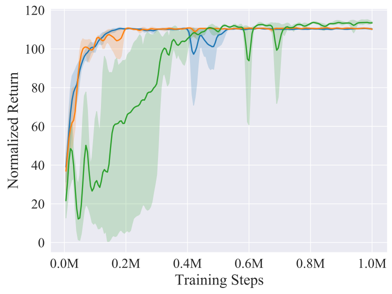
(g) walker2d-medium-v2



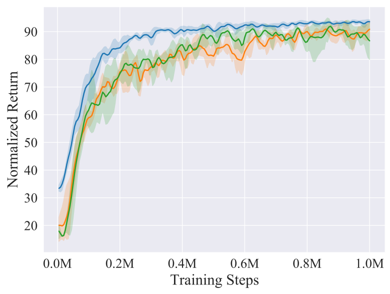
(h) halfcheetah-medium-v2



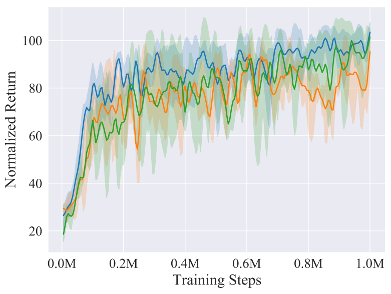
(i) hopper-medium-v2



(j) walker2d-medium-expert-v2



(k) halfcheetah-medium-expert-v2



(l) hopper-medium-expert-v2

Figure 9. Performances of PRDC with k nearest neighbors constraint.