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 Policy Regularization with Dataset Constraint (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."
We consider the infinite-horizon Markov Decision Process (MDP) defined by a tuple \((\mathcal{S}, \mathcal{A}, \rho_0, \mathcal{P}, r, \gamma)\), where \(\mathcal{S}\) is the state space, \(\mathcal{A}\) is the action space, \(\rho_0\) is the initial state distribution, \(\mathcal{P} : \mathcal{S} \times \mathcal{A} \rightarrow \Delta_\mathcal{S}\) is the transition function\(^1\), \(r : \mathcal{S} \rightarrow \mathbb{R}\) 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 \(\forall (s, a) \in \mathcal{S} \times \mathcal{A}, |r(s)| \leq R_{\text{max}},\) and \(a \in [-A, A]\).

Given the MDP and the agent’s policy \(\pi : \mathcal{S} \rightarrow \mathcal{A}\), 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 \(\pi\),

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

where the expectation takes over the randomnesses of the initial state distribution \(\rho_0\) and the transition function \(\mathcal{P}\). The objective of RL is to learn an optimal policy \(\pi^*\) that has maximal expected discounted reward, i.e.,

\[
\pi^* = \arg\max_\pi J(\pi).
\]

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

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

We further define the occupancy measure \(d^\pi : \mathcal{S} \rightarrow \mathbb{R}\) of \(\pi\),

\[
d^\pi(s') = (1 - \gamma) \int_{s}^{\infty} \gamma^{t} p(s, a, s') d\pi ds,
\]

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

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

### 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 \(\mu\), 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 \(Q^\pi\),

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

where \(\delta_t = [r(s) + (1 - d)\hat{Q}^\pi(s', a') - \hat{Q}^\pi(s, a)]\), \(a' = \pi(s')\) and \(\eta\) is a hyper-parameter controlling the step size. With sufficiently enough samples, \(Q^\pi\) will converge to \(Q^\pi\) (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 \(\pi\), not the behavior

---

\(^1\)We use \(\Delta_X\) 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.
policy \( \mu \). 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, where the former forces \( \pi \) to be similar to \( \mu \) 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 \( \beta \) 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 proven 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 \( \pi \) 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 \( \beta \), our state-aware regularization will allow \( \pi \) to learn optimal actions from a different state but still maintain enough conservatism.

### 3. 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 \( \theta_1, \theta_2, \phi \) 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 \( Q_{\theta_1} \) and \( Q_{\theta_2} \):

\[
\mathcal{L}_{\text{TD}}(\theta_i) = \mathbb{E}_{(s,a,r,s',d) \sim \mathcal{D}} [(Q_{\theta_i}(s,a) - y(r,s',d))^2].
\]

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

\[
\mathcal{L}_{\text{TD3}}(\phi) = \mathbb{E}_{s \sim \mathcal{D}, \hat{a} = \pi_{\phi}(s)} [-Q_{\theta_1}(s, \hat{a})].
\]

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

\[
\mathcal{L}_{\text{PRDC}}(\phi) = \lambda \mathcal{L}_{\text{TD3}}(\phi) + \mathcal{L}_{\text{DC}}(\phi).
\]

Following TD3+BC (Fujimoto & Gu, 2021), we set \( \lambda = \frac{\alpha N}{\sum_{i,j} Q(\pi_{\phi'}(s^{(i,j)}) \in \mathbb{N})} \) with \( \alpha \) a hyper-parameter and \( N \) the batch size. The pseudo-code is summarized in Algorithm 1.

\[\text{The function clip}_X(\cdot) \text{ clips its input into } [-X, X].\]
Algorithm 1 PRDC

1: **Input:** Initial policy parameters $\phi$, Q-function parameters $\theta_1, \theta_2$, offline dataset $D$, 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 $D$.
5: Update $\theta_i, i \in \{1, 2\}$ using gradient descent with Equation (5).
6: Use KD-Tree to find the nearest neighbor in $D$ of every $(s, \pi_\phi(s))$.
7: Update $\phi$ using gradient descent with Equation (7).
8: Update target network with $\theta'_1 \leftarrow (1 - \tau)\theta'_1 + \tau \theta_1, \theta'_2 \leftarrow (1 - \tau)\theta'_2 + \tau \theta_2, \phi' \leftarrow (1 - \tau)\phi'$. 
9: end for

Remark 3.2 (Highly efficient implementation). Our method requires searching $(s, \pi_\phi(s))$’s nearest neighbor in $D$, which will be time-consuming if $D$ is large. To speed up the search, we use KD-Tree (Bentley, 1975). KD-Tree has a time complexity of $O(M \log |D|)$ 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 $\pi_\phi$ and the behavior policy $\mu$.

**Definition 3.3** (Lipschitz function). A function $f$ from $S \subset \mathbb{R}^n$ into $\mathbb{R}^n$ is called a Lipschitz 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 $\mu$, and the transition function $P$.

**Assumption 3.4.** Suppose that the Q-function we learn is a Lipschitz function with $K_D$ the Lipschitz constant, i.e.,

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

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

**Assumption 3.5.** Suppose that $\mu$ is deterministic and a Lipschitz function with $K_\mu$ the Lipschitz constant, i.e.,

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

for all $s_1, s_2 \in S$.

**Assumption 3.6.** $\forall a_1, a_2 \in A$, there exists a positive constant $K_P$ such that

$$\|P(s'|s, a_1) - P(s'|s, a_2)\| \leq K_P\|a_1 - a_2\|,$$

for any $s, s' \in S$.

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 $\max_{s \in S} d_D^\beta(s, \pi_\phi(s)) \leq \epsilon$, which can be achieved by PRDC. Then with Assumption 3.4 and Assumption 3.5, we have

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

for any $s \in S$.

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 $D$, we could still treat it as an in-distribution sample since $s' \in D$ and $D$ is constructed by $\mu$. That is,

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

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

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

$$Q^\pi(s', \pi(s')) \approx Q^\pi(s', \mu(s')).$$

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 $\max_{s \in S} d_D^\beta(s, \pi_\phi(s)) \leq \epsilon$ and $\max_{s \in S} |\pi^*(s) - \mu(s)| \leq \epsilon_{\text{opt}}$, which can be achieved by PRDC. Then we have

$$|J(\pi^*) - J(\pi)| \leq \frac{C K_P R_{\text{max}}}{1 - \gamma} \left[ (1 + \frac{K_q}{\beta}) \epsilon_{\text{opt}} + \epsilon_{\text{opt}} \right],$$

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 $\beta$. However, as discussed in 3.1, a small $\beta$ may not keep the policy from OOD actions and thus degrades the performance. With a proper $\beta$, our method can reach higher performance...
than TD3+BC (corresponding to $\beta \rightarrow \infty$). Moreover, for any state $s$, if we assume that there exists another state-action pair $(\hat{s}, \hat{a}) \in D$ such that $\hat{a}$ is the optimal action for state $s$, then with a properly selected $\beta$, we can get that $(\hat{s}, \hat{a}) = \arg \min_{(\tilde{s}, \tilde{a}) \in D} D^2(s, \pi(s)), \text{ where we use (}\tilde{s}, \tilde{a} ) \in D \text{ to denote all the state-action pairs in } D$. Regularizing $\pi(s)$ toward $\hat{a}$ will eventually give us an optimal policy. That is, we can obtain a zero performance gap between $\pi^*$ and $\pi$ 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 $\mu$ and then learns the policy $\pi$, which has a similar distribution to that of $\mu$. Nevertheless, the fitting error will backpropagate to $\pi$ and eventually affect $\pi$’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 $\pi$ without an explicit fit of $\mu$. 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 $\pi$ to match $\mu$’s distribution. However, distribution constraint limits $\pi$’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 $\pi$. Empirically, they found that when computing MMD over a small number of samples, the sampled MMD between $\mu$ and $\pi$ was similar to the MMD between the supports of $\mu$ and $\pi$. In their experiments, they showed that this support constraint could find optimal policies even when the offline dataset $D$ 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 $\pi$ with the whole dataset $D$. Moreover, we want to learn optimal actions from all those in $D$ 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 SCAAN (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 $\beta$ 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 $\mu$ and then learns a $\pi$ similar to $\mu$.
- **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-
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 on 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 $[0, 101]^2$, 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.

We collect four datasets on lineworld. They are named lineworld-easy, lineworld-medium, lineworld-hard and lineworld-superhard. ✔ means accomplish, while × means not.

<table>
<thead>
<tr>
<th>Task Name</th>
<th>BC</th>
<th>BCQ</th>
<th>BEAR</th>
<th>AWAC</th>
<th>CQL</th>
<th>IQL</th>
<th>TD3+BC</th>
<th>SPOT</th>
<th>PRDC (Ours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>halfcheetah-random</td>
<td>0.2</td>
<td>8.8</td>
<td>15.1</td>
<td>—</td>
<td>20.0</td>
<td>11.2</td>
<td>11.0</td>
<td>—</td>
<td>26.9 ± 1.0</td>
</tr>
<tr>
<td>hopper-random</td>
<td>4.9</td>
<td>7.1</td>
<td>14.2</td>
<td>—</td>
<td>8.3</td>
<td>7.9</td>
<td>8.5</td>
<td>—</td>
<td>26.8 ± 9.3</td>
</tr>
<tr>
<td>walker2d-random</td>
<td>1.7</td>
<td>6.5</td>
<td>10.7</td>
<td>—</td>
<td>8.3</td>
<td>5.9</td>
<td>1.6</td>
<td>—</td>
<td>5.0 ± 1.2</td>
</tr>
<tr>
<td>halfcheetah-medium</td>
<td>42.6</td>
<td>47.0</td>
<td>41.0</td>
<td>43.5</td>
<td>44.0</td>
<td>47.4</td>
<td>48.3</td>
<td>58.4</td>
<td>63.5 ± 0.9</td>
</tr>
<tr>
<td>hopper-medium</td>
<td>52.9</td>
<td>56.7</td>
<td>51.9</td>
<td>57.0</td>
<td>58.5</td>
<td>66.2</td>
<td>59.3</td>
<td>86.0</td>
<td>100.3 ± 0.2</td>
</tr>
<tr>
<td>walker2d-medium</td>
<td>75.3</td>
<td>72.6</td>
<td>80.9</td>
<td>72.4</td>
<td>72.5</td>
<td>78.3</td>
<td>83.7</td>
<td>86.4</td>
<td>85.2 ± 0.4</td>
</tr>
<tr>
<td>halfcheetah-medium-replay</td>
<td>36.6</td>
<td>40.4</td>
<td>29.7</td>
<td>40.5</td>
<td>45.5</td>
<td>44.2</td>
<td>44.6</td>
<td>52.2</td>
<td>55.0 ± 1.1</td>
</tr>
<tr>
<td>hopper-medium-replay</td>
<td>18.1</td>
<td>53.3</td>
<td>37.3</td>
<td>37.2</td>
<td>95.0</td>
<td>94.7</td>
<td>60.9</td>
<td>100.2</td>
<td>101.1 ± 1.6</td>
</tr>
<tr>
<td>walker2d-medium-replay</td>
<td>26.0</td>
<td>52.1</td>
<td>18.5</td>
<td>27.0</td>
<td>77.2</td>
<td>73.8</td>
<td>81.8</td>
<td>91.6</td>
<td>92.0 ± 1.6</td>
</tr>
<tr>
<td>halfcheetah-medium-expert</td>
<td>55.2</td>
<td>89.1</td>
<td>38.9</td>
<td>42.8</td>
<td>91.6</td>
<td>86.7</td>
<td>90.7</td>
<td>86.9</td>
<td>94.5 ± 0.5</td>
</tr>
<tr>
<td>hopper-medium-expert</td>
<td>52.5</td>
<td>81.8</td>
<td>17.7</td>
<td>55.8</td>
<td>105.4</td>
<td>91.5</td>
<td>98.0</td>
<td>99.3</td>
<td>109.2 ± 4.0</td>
</tr>
<tr>
<td>walker2d-medium-expert</td>
<td>107.5</td>
<td>109.5</td>
<td>95.4</td>
<td>74.5</td>
<td>108.8</td>
<td>109.6</td>
<td>110.1</td>
<td>112.0</td>
<td>111.2 ± 0.6</td>
</tr>
<tr>
<td>antmaze-umaze</td>
<td>65.0</td>
<td>73.0</td>
<td>56.7</td>
<td>84.8</td>
<td>88.2</td>
<td>91.3</td>
<td>93.5</td>
<td>—</td>
<td>98.8 ± 1.0</td>
</tr>
<tr>
<td>antmaze-umaze-diverse</td>
<td>55.6</td>
<td>61.0</td>
<td>61.0</td>
<td>49.3</td>
<td>43.3</td>
<td>66.7</td>
<td>54.6</td>
<td>40.7</td>
<td>90.0 ± 6.8</td>
</tr>
<tr>
<td>antmaze-medium-play</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>65.2</td>
<td>70.4</td>
<td>0.0</td>
<td>74.7</td>
<td>82.8 ± 4.8</td>
</tr>
<tr>
<td>antmaze-medium-diverse</td>
<td>0.0</td>
<td>0.0</td>
<td>8.0</td>
<td>0.7</td>
<td>54.0</td>
<td>74.6</td>
<td>0.0</td>
<td>79.1</td>
<td>78.8 ± 6.9</td>
</tr>
<tr>
<td>antmaze-large-play</td>
<td>0.0</td>
<td>6.7</td>
<td>0.0</td>
<td>0.0</td>
<td>18.8</td>
<td>43.5</td>
<td>0.0</td>
<td>35.3</td>
<td>54.8 ± 10.9</td>
</tr>
<tr>
<td>antmaze-large-diverse</td>
<td>0.0</td>
<td>2.2</td>
<td>0.0</td>
<td>1.0</td>
<td>31.6</td>
<td>45.6</td>
<td>0.0</td>
<td>36.3</td>
<td>50.0 ± 5.4</td>
</tr>
</tbody>
</table>

Table 2. Accomplishments of BEAR, TD3+BC, and PRDC on lineworld-easy, lineworld-medium, lineworld-hard and lineworld-superhard.
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.

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 $\epsilon$ 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, 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.
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 $\beta$

The hyper-parameter $\beta$ controls how much conservatism is imposed when updating policy. To see how it influences the policy’s performance, we conduct an ablation study of $\beta$ 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 $\beta = 0.1$, PRDC performs bad. It is because a small $\beta$ 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 $\beta = 1000$, PRDC performs not well. A large $\beta$ 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 $\beta$ on all datasets, meaning we do not need much effort to tune it.

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.

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


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 \( \|\hat{a}\| + \|\hat{s}\| \leq \beta \), \( \beta > 0 \), \( \epsilon > 0 \), then the following inequalities hold,
\[
\begin{align*}
\|a - \hat{a}\| & \leq \epsilon, \\
\|\hat{s} - \hat{a}\| & \leq \epsilon/\beta, \\
\|s + a - \hat{s} - \hat{a}\| & \leq (1/\beta + 1)\epsilon.
\end{align*}
\]

Proof. Since \( \beta > 0 \), we have
\[
\beta \|s - \hat{s}\| \leq \|\hat{a}\| + \|\hat{s}\| \leq \epsilon.
\]
By dividing both sides of the above inequality by \( \beta \), we achieve \( \|s - \hat{s}\| \leq \epsilon/\beta \). Similarly, we can get \( \|a - \hat{a}\| \leq \epsilon \). Thus,
\[
\|s + a - \hat{s} - \hat{a}\| \leq \|s - \hat{s}\| + \|a - \hat{a}\| \leq (1/\beta + 1)\epsilon,
\]
where the first inequality comes from Lemma A.1.

Theorem 3.7. Let \( \max_{s \in \mathcal{S}} d^2_{\mathcal{D}}(s, \pi_\phi(s)) \leq \epsilon \), which can be achieved by PRDC. Then with Assumption 3.4 and Assumption 3.5, we have
\[
\|Q(s, \pi_\phi(s)) - Q(s, \mu(s))\| \leq ((K_\mu + 2)/\beta + 1) K_Q \epsilon,
\]
for any \( s \in \mathcal{S} \).

Proof. Let
\[
(\hat{s}, \hat{a}) = \arg \min_{(\hat{s}, \hat{a}) \in \mathcal{D}} d^2_{\mathcal{D}}(s, \pi_\phi(s)),
\]
where \( \hat{a} = \mu(\hat{s}) \). By expanding the left side of Equation (8), we get
\[
\begin{align*}
\|Q(s, \pi_\phi(s)) - Q(s, \mu(s))\| &= \|Q(s, \pi_\phi(s)) - Q(\hat{s}, \hat{a}) + Q(\hat{s}, \hat{a}) - Q(s, \mu(s))\| \\
& \leq (i) \|Q(s, \pi_\phi(s)) - Q(\hat{s}, \hat{a})\| + \|Q(\hat{s}, \hat{a}) - Q(s, \mu(s))\| \\
& \leq (ii) K_Q (\|s \oplus \pi(s) - \hat{s} \oplus \hat{a}\| + \|\hat{a} \oplus \mu(\hat{s}) - s \oplus \mu(s)\|) \\
& \leq (iii) K_Q ((1/\beta + 1)\epsilon + \|\hat{s} \oplus \mu(\hat{s}) - s \oplus \mu(s)\|) \\
& \leq (iv) K_Q ((1/\beta + 1)\epsilon + \|\hat{s} - \hat{a}\| + \|\mu(\hat{s}) - \mu(s)\|) \\
& \leq (v) K_Q ((1/\beta + 1)\epsilon + \|\hat{s} - \hat{a}\| + K_\mu \|\hat{s} - \hat{a}\|) \\
& \leq (vi) ((K_\mu + 2)/\beta + 1) K_Q \epsilon.
\end{align*}
\]
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 \( f : \mathcal{S} \subset \mathbb{R}^m \rightarrow \mathbb{R} \), we have that
\[
\left| \int_a^b f(x) \, dx \right| \leq \int_a^b |f(x)| \, dx,
\]
where \( a \leq b \), and \([a, b] \subseteq \mathcal{S} \).
Proof. For any \( x \in S \), the following inequality holds,

\[-|f(x)| \leq f(x) \leq |f(x)|.\]

Thus,

\[-\int_a^b |f(x)| \, dx \leq \int_a^b f(x) \, dx \leq \int_a^b |f(x)| \, dx.\]

Therefore,

\[\left| \int_a^b f(x) \, dx \right| \leq \int_a^b |f(x)| \, dx.\]

Thus, we finish the proof. \( \square \)

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

\[\int_S |\pi_s - \mu_s| \, ds \leq CK \max_{s \in S} \|\pi_s - \mu_s\|,\]

where \( C \) is a positive constant.

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

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

\[\|\pi(s) - \mu(s)\| \leq \left( 1 + \frac{1}{K} \beta \right) \epsilon, \forall s \in S,\]

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

\[(\hat{s}, \hat{a}) = \arg \min_{(\hat{s}, \hat{a}) \in D} d_D^D(s, \pi(s)),\]

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

\[\|\pi(s) - \mu(s)\| = \|\pi(s) - \mu(\hat{s}) + \mu(\hat{s}) - \mu(s)\|
\leq \|\pi(s) - \mu(\hat{s})\| + \|\mu(\hat{s}) - \mu(s)\|
\leq \|\pi(s) - \mu(\hat{s})\| + K\mu\|\hat{s} - s\|
\leq \left( 1 + \frac{K}{\beta} \right) \epsilon.\]

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

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

\[|J(\pi^*) - J(\pi)| \leq \frac{CK \max_{s \in S} \pi^*(s)}{1 - \gamma} \left( 1 + \frac{K}{\beta} \right) \epsilon + \epsilon_{\text{opt}}.\]

where \( C \) is a positive constant.

Proof. With Lemma A.2, we have

\[|J(\pi^*) - J(\pi)| = |J(\pi^*) - J(\mu) + J(\mu) - J(\pi)|\]
\[\leq |J(\pi^*) - J(\mu)| + |J(\pi) - J(\mu)|.\]
We first bound $|J(\pi) - J(\mu)|$. By taking into Equation (2), we get

$$|J(\pi) - J(\mu)| = \left| \frac{1}{1-\gamma} \mathbb{E}_{s \sim d'(s)}[r(s)] - \frac{1}{1-\gamma} \mathbb{E}_{s \sim d'(s)}[r(s)] \right| ds$$

$$= \left| \frac{1}{1-\gamma} \int_{S} (d'(s) - d''(s)) r(s) ds \right|$$

$$(i) \leq \frac{1}{1-\gamma} \int_{S} |d'(s) - d''(s)||r(s)| ds$$

$$(i) \leq \frac{R_{\max}}{1-\gamma} \int_{S} |d'(s) - d''(s)| ds.$$  

$$\leq \frac{C K_{p} R_{\max}}{1-\gamma} \max_{s \in S} \|\pi(s) - \mu(s)\|$$

$$(ii) \leq \frac{C K_{p} R_{\max}}{1-\gamma} (1 + \frac{K_{\mu}}{\beta}) \epsilon_{\pi}.$$  

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

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

(15)

Thus,

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

(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
Policy Regularization with Dataset Constraint for Offline Reinforcement Learning

C. More Results

C.1. Sensitivity on the Hyper-parameter $\beta$

This section investigates how $\beta$ 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 $\beta$ 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 $\beta$ 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

$$\min_{\phi} L(\phi) := ||\pi(s) - \bar{\pi}||,$$

where $\bar{\pi}$ 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.

<table>
<thead>
<tr>
<th>Hyper-parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Network</strong></td>
<td></td>
</tr>
<tr>
<td>Actor learning rate</td>
<td>$3 \times 10^{-4}$</td>
</tr>
<tr>
<td>Critic learning rate</td>
<td>$3 \times 10^{-4}$ for MuJoCo</td>
</tr>
<tr>
<td></td>
<td>$1 \times 10^{-3}$ for AntMaze</td>
</tr>
<tr>
<td>Batch size</td>
<td>256</td>
</tr>
<tr>
<td>Optimizer</td>
<td>Adam</td>
</tr>
<tr>
<td>Q-network</td>
<td>3 layers ReLU activated MLPs with 256 units</td>
</tr>
<tr>
<td>Policy Network</td>
<td>3 layers ReLU activated MLPs with 256 units</td>
</tr>
<tr>
<td><strong>TD3</strong></td>
<td></td>
</tr>
<tr>
<td>Critic learning rate</td>
<td>0.99 for MuJoCo</td>
</tr>
<tr>
<td></td>
<td>0.995 for AntMaze</td>
</tr>
<tr>
<td>Number of iterations</td>
<td>$10^6$</td>
</tr>
<tr>
<td>Target update rate $\tau$</td>
<td>0.005</td>
</tr>
<tr>
<td>Policy noise</td>
<td>0.2</td>
</tr>
<tr>
<td>Policy noise clipping</td>
<td>0.5</td>
</tr>
<tr>
<td>Policy update frequency</td>
<td>2</td>
</tr>
<tr>
<td><strong>PRDC</strong></td>
<td></td>
</tr>
<tr>
<td>Normalized state $k$</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>$\beta$</td>
<td>2.0 for MuJoCo</td>
</tr>
<tr>
<td></td>
<td>${2.0, 7.5, 15.0}$ for AntMaze {umaze, medium, large}</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>2.5 for MuJoCo</td>
</tr>
<tr>
<td></td>
<td>${2.5, 7.5, 20.0}$ for AntMaze {umaze, medium, large}</td>
</tr>
</tbody>
</table>
Figure 8. Performances of PRDC with different $\beta$, a hyper-parameter in Definition 3.1.
Figure 9. Performances of PRDC with $k$ nearest neighbors constraint.