State Deviation Correction for Offline Reinforcement Learning

Hongchang Zhang[1], Jianzhun Shao[1], Yuhang Jiang[1], Shuncheng He[1], Guanwen Zhang[2], Xiangyang Ji[1]*

[1] Tsinghua University,
[2] Northwestern Polytechnical University
hc-zhang19@mails.tsinghua.edu.cn

Abstract

Offline reinforcement learning aims to maximize the expected cumulative rewards with a fixed collection of data. The basic principle of current offline reinforcement learning methods is to restrict the policy to the offline dataset action space. However, they ignore the case where the dataset's trajectories fail to cover the state space completely. Especially, when the dataset's size is limited, it is likely that the agent would encounter unseen states during test time. Prior policy-constrained methods are incapable of correcting the state deviation, and may lead the agent to its unexpected regions further. In this paper, we propose the state deviation correction (SDC) method to constrain the policy's induced state distribution by penalizing the out-of-distribution states which might appear during the test period. We first perturb the states sampled from the logged dataset, then simulate noisy next states on the basis of a dynamics model and the policy. We then train the policy to minimize the distances between the noisy next states and the offline dataset. In this manner, we allow the trained policy to guide the agent to its familiar regions. Experimental results demonstrate that our proposed method is competitive with the state-of-the-art methods in a GridWorld setup, offline Mujoco control suite, and a modified offline Mujoco dataset with a finite number of valuable samples.

Introduction

In recent years, reinforcement learning has progressed in substantial leaps in areas ranging from video games (Mnih et al. 2015) to simulated robotic tasks (Schulman et al. 2015; Haarnoja et al. 2018). However, the applicability of reinforcement learning in real-world domains is hampered by several challenges. On the one hand, the online data collection – reinforcement learning's innate characteristic – threatens to introduce risks in safety-critical settings (Berkenkamp et al. 2017). On the other hand, the high sample complexity of reinforcement learning poses a significant challenge for industry practitioners with limited resources.

Offline reinforcement learning seemingly holds the promise of tackling the above difficulties (Levine et al. 2020). The interactive data generation process is eliminated, and a fixed collection of data is offered. It turns out that in this case, the agent can get rid of risky behaviors during the training process. Additionally, experiences from a variety of sources such as video websites and published datasets(Gulcehre et al. 2020) could be employed to train the agent. By exploiting large-scale datasets, offline reinforcement learning is also a solution to high sample complexity.

Nevertheless, most commonly used reinforcement learning methods easily fail to learn satisfactory strategies when they are applied in offline settings. The cause is that these algorithms suffer from the gap between the trained policy and the offline policy (Fujimoto, Meger, and Precup 2019), which is referred as the extrapolation error. Due to the extrapolation error, these methods tend to opt for the out-of-distribution target actions when running the Bellman backup, and are bound to overestimate the Q-value function, thus undermining their efficacies.

In the principle of reducing the extrapolation error, most previous works attempt to constrain the trained policy to the offline dataset's action space (Fujimoto, Meger, and Precup 2019; Kumar et al. 2019; Wu, Tucker, and Nachum 2019; Kumar et al. 2020). We now refer to these methods as policy-constrained algorithms. Although these offline variants gain considerable success in the offline setting, they neglect the scenario where the dataset cannot cover the state space completely. In this setting, the trained policy is likely to fail to generalize well in the state space. This issue is not severe when presented with large-scale datasets. It comes into effect when offered a dataset with a modest number of samples since the dataset is incapable of reflecting the actual state space and the dynamics. Without the loss of generality, we assume that an agent's initial position during the test time is slightly different from the dataset. The aforementioned policy-constrained offline methods concern little about whether the state at the next time step is close to the dataset. When the agent follows these methods, it stands a chance to encounter a new state that it never observes before. If the agent insists on selecting the 'correct' action later, the state deviation will inevitably accumulate over time.

We show in Figure 1 that state deviation could undermine the policy's performance. The majority of the actions in the dataset are 'right'. The offline policy tends to choose the 'up' action when the agent runs into the wall. If the agent is initialized at the purple point close to the static dataset and complies with the default policy during the test time, it

*Corresponding author.

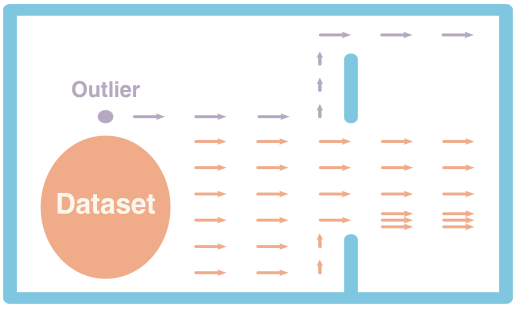


Figure 1: The agent diverges from the dataset when it starts form the purple point.

will diverge considerably from the dataset. As this example points out, it is vital for the agent to simultaneously constrain both its state and action in the support set.

In this paper, we propose the state deviation correction (SDC) method to reduce the state visitation mismatch between the trained policy and the offline dataset. Our basic idea is to predict the outcome of executing the policy and ensure that the generated state is in the support of the dataset. The framework of our proposed method consists of three components: a dynamics model, a state transition model, and an actor-critic agent. The dynamics model is trained to predict the next state conditioned on the state and the action. Meanwhile, we maintain a state transition model that takes a state as input and predicts the next state. Note that the state transition model, which is a generative model and specifies the support of the state space, is independent of actions. We train the policy so that it can yield the action which directs the agent to the in-distribution region when perturbing the starting state. The perturbed state is used to represent the situation where the state deviation appears. The in-distribution region is modeled by the state transition model. In the meantime, the dynamics model is utilized to predict the state at the next time step when starting from the noisy state and following the policy. The agent is thus expected to predict potential next states and move to the state close to the static dataset, therefore reducing state deviation.

To aid understanding of SDC, we show the existence of state deviation of most commonly used offline RL methods in a GridWorld setting, highlighting the advantage of our state-constrained method. We then evaluate our proposed method on D4RL (Fu et al. 2020) tasks. The experimental results demonstrate that our method is competitive with the state-of-the-art algorithms in the Mujoco control suite. Additionally, to test the validity of our methods when presented with limited data, we design a new task by reducing the number of expert samples gradually. Our proposed method performs better than the policy-constrained methods by a large margin in this setting.

Related Work

Offline reinforcement learning (Lange, Gabel, and Riedmiller 2012; Riedmiller 2005) focuses on training a policy given a fixed dataset and receives increasing attention recently (Siegel et al. 2020; Kostrikov et al. 2021; Urpí, Curi, and Krause 2021; Jin, Yang, and Wang 2021; Yu et al. 2021). It has been applied in healthcare (Gottesman et al. 2018), recommendation systems (Swaminathan and Joachims 2015), dialogue systems (Zhou et al. 2017), and autonomous driving (Sallab et al. 2017). In this case, standard off-policy methods (Mnih et al. 2015) fail to perform robustly and they are sensitive to the dataset distribution. The problem has been studied in approximate dynamic programming (Bertsekas and Tsitsiklis 1995). This study ascribes it to the errors arising from distribution shift and function approximation. Recently, Van Hasselt et al. (2018) shows that the temporal difference algorithms will diverge when represented with function approximators and trained under off-policy data.

Some works mitigate the extrapolation error problem in the view of restricting the value function. Agarwal, Schuurmans, and Norouzi (2019) introduces value function ensembles to stabilize the update of the Q-function. Kumar et al. (2020) learns a lower bound of the Q-value of policy by penalizing out-of-distribution state-action samples and maximizing the Q-value of the in-distribution samples. Our proposed method is in the line of restraining the policy distribution. Fujimoto, Meger, and Precup (2019) minimizes the state-action distribution distance between the trained and offline policies. Kumar et al. (2019) directly constrains the policy by minimizing the Maximum Mean Discrepancy (MMD) (Gretton et al. 2012) between the trained policy and the offline policy. Jaques et al. (2019) takes advantage of KL-divergence and proposes a regularization term to train the policy. Wu, Tucker, and Nachum (2019) proposes an extra reward which denotes the KL-divergence between the current policy and a generative policy model. Ghasemipour, Schuurmans, and Gu (2021) introduces EMaQ which simplifies BCQ and considers the number of samples and the proposal distribution. These methods attempt to constrain the policy in the offline action space. Unlike these works, we observe that the out-of-distribution states during the test time could weaken the policy's performance and we turn to control the state visitation to be in the support of the dataset.

Furthermore, our proposed method is related to Yu et al. (2020); Kidambi et al. (2020) since they also utilize a learned model. Yu et al. (2020) learns a dynamics model and simulates samples from the dynamics model to train the policy network. Kidambi et al. (2020) draws on the static dataset to learn a pessimistic MDP and trains a near-optimal policy in this pessimistic MDP. This method employs an unknown state-action detector to determine if a state-action pair is unknown and uses a model-based method (Rajeswaran, Mordatch, and Kumar 2020) to train the policy. Swazinna, Udluft, and Runkler (2021) uses a dynamics model to rollout trajectories and punish the out-of-distribution states and actions. Our method predicts one step where the model has high prediction accuracy and pulls the agent back at once when it deviates from the dataset.

Compared with them, our proposed method does not utilize imaginary samples to train the policy directly and we focus on state deviation. We generate noisy next states based on the dynamic model, minimize their distance from the dataset, and backpropagate the gradient through the policy.

Background

In general, we consider a reinforcement learning problem which could be modeled by a Markov decision process (S, A, P, R, γ), with the state space S, the action space A, the transition probability matrix P, the reward function R, and the discount factor γ. Each term of P denotes the probability of arriving at s' when selecting a at state s. At each time step, an agent is located at s and chooses an action a. After interacting with an environment, it arrives at next state s' and obtains a reward r. The goal of reinforcement learning is to train a policy $\pi(a|s)$ to maximize the expected return:

$$\max_{\pi} \ \mathbb{E}_{\pi}[\sum_{t=0} \gamma^t r_t]. \tag{1}$$

In reinforcement learning, the cumulative rewards expectation is defined as value function $Q_\pi(s,a) = \mathbb{E}_\pi[\sum_{t=0} \gamma^t r_t|s,a]$. Q-learning method(Watkins and Dayan 1992) trains the value function by forming a Bellman target:

$$\mathcal{T}Q(s,a) := \mathbb{E}[r + \gamma \max_{a'} Q(s',a')], \tag{2}$$

where $\mathcal{T}$ is the Bellman operator. When the state space is huge or continuous, it is hard to enumerate all the states. Neural networks are recently utilized to approximate the actual value function. DQN updates the value function by minimizing:

$$\mathbb{E}_{s,a,r,s' \sim \mathcal{D}}(Q(s,a;\theta) - (r + \gamma(\max_{a'} Q(s',a';\theta^-)))^2, \tag{3}$$

where $\mathcal{D}$ is a replay buffer collecting previous samples, and θ, θ^- are parameters of the Q neural network and target Q network, respectively. When the action space is continuous, selecting an optimal action is challenging. A policy $\pi(a|s)$ is employed to sample actions, and the loss function is defined as:

$$\mathbb{E}_{s,a,r,s' \sim \mathcal{D}}(Q(s,a;\theta) - (r + \gamma(Q(s', \pi(a'|s';\omega);\theta^-)))^2, \tag{4}$$

where ω is the parameter of the policy. The policy is updated to yield the action with the maximal Q-value:

$$\max_{\omega} Q(s, \pi(a|s;\omega);\theta) \tag{5}$$

State Deviation

State deviation is introduced by the state space and the transition function's mismatches between the offline data and the actual environment. To be specific, there are two ways in the following sections that the mismatches could lead to state deviation and impair the trained policy's performance.

Initial State Difference

The initial state distribution $\hat{\rho}$ of the offline dataset might be different from that of the actual environment ρ. Considering that the initial states of the agent may vary drastically across episodes in many scenarios, a modest number of trajectories are likely to induce an empirical distribution far from ρ. For instance, the initial state of the Halfcheetah agent in the Mujoco control suite has 17 dimensions, each of which has a range of $[-0.1, 0.1]$. When the game is reset, the initial state is sampled randomly from this region. However, the offline dataset in D4RL(Fu et al. 2020) has an insufficient amount of initial data. Because D4RL dataset has up to $1,000,000$ samples for each difficulty level and merely $1,000$ initial state samples for the Halfcheetah task. During the test period, there is a high possibility that the agent has never encountered the initial state in the training dataset.

Dynamics Bias

The empirical distribution of the dynamics of the offline dataset $\hat{P}$ might differ from the true dynamics P. The dynamics bias has no effect if $\hat{P}(s'|s,a) > 0$ where $P(s'|s,a) > 0$ for all s, a. However, for some s, a, if $\hat{P}(s'|s,a) = 0$ where $P(s'|s,a) > 0$, the agent might move to a foreign state when it executes the action a at the state s.

Based on initial state difference and dynamics bias, state deviation Δ_t might appear at each time step. Especially in continuous domains, it is challenging to observe the exact same states or actions in a dataset. What's more, the approximation error of the neural network would enlarge the deviation further. Since the deviation at the previous time step would prompt the later departure from the support, the cumulative variations might end up affecting the policy's performance in the long run. We will alleviate this issue and illustrate our proposed method in the next section.

State Deviation Correction

As explained in the previous section, the state deviation could be readily generated, but the current offline reinforcement learning methods ignore this problem. As a result, out-of-distribution states would be met during the test time and influence the performance of the trained policy. We propose state deviation correction (SDC) to suppress the state deviation caused by the limited data.

Specifically, SDC is trained to help the policy yield actions which are reward-seeking and also guide the agent located in the out-of-distribution states to the regions close to the offline dataset. In this manner, any departure from the offline state space would be penalized immediately.

Networks

We maintain a dynamics model network M, a state transition model network U, an actor-critic architecture for our proposed SDC. The dynamic model is trained to model the environment's dynamics. The state transition model learns to predict the next states given current states.

For the dynamics model, we use an ensemble of models parameterized by neural networks (Chua et al. 2018). Each component of the ensemble is a Bayesian neural network that represents a Gaussian distribution. The outputs of each network are mean and diagonal covariance of the distribution: $p^i(s', r \mid s, a) = \mathcal{N}\left(\mu^i(s,a), \Sigma^i(s,a)\right)$. Each Bayesian neural network represents aleatoric uncertainty of the dynamics and the bootstrap ensemble accounts for epistemic uncertainty. As Chua et al. (2018) mentions, capturing aleatoric uncertainty and epistemic uncertainty allows the model to predict accurately.

The state transition model is a network that takes input s and outputs s'. Note that it is independent of the action. We

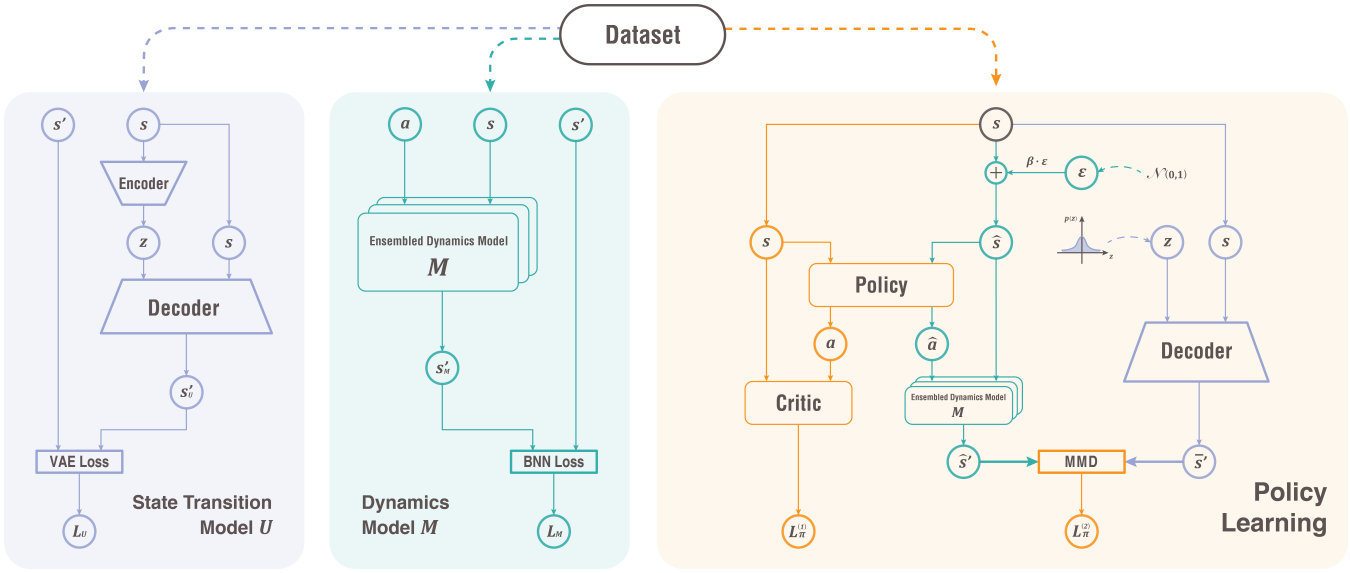


Figure 2: The framework of SDC. The left, middle and right parts are the training processes of the state transition model, the dynamics model, and the policy network, respectively.

use a conditional variational auto-encoder (CVAE) (Kingma and Welling 2013; Sohn, Lee, and Yan 2015) to model the state transition network. The generator is capable of sampling the next states given current states. The state transition model is used to provide supervision for out-of-distribution states. We will illustrate its usage in detail in the following subsection.

Besides, we maintain and update two Q networks and a policy network according to the soft actor-critic (Haarnoja et al. 2018) algorithm.

Implementation

Given a state s, SDC first adds a noise ϵ with small magnitude to the state and formulates a noisy state as:

$$\hat{s} = s + \beta \cdot \epsilon, \tag{6}$$

where ϵ is sampled from a Gaussian distribution $\mathcal{N}(0, 1)$ and β is a small constant. The perturbation aims to construct a broader state space that contains substantial out-of-distribution states. For initial positions, the induced state space is likely to cover the range of the most initial states.

We then obtain an action $\hat{a}$ by feeding $\hat{s}$ to the policy network π. The dynamics model takes $\hat{s}$ and $\hat{a}$ as inputs and outputs a noisy next state $\hat{s'}$. It simulates the scenario where an agent starts from an out-of-distribution state and follows the trained policy. To reduce state deviation, we expect to train the policy so that it can lead the agent to a reasonable $\hat{s'}$, which is close to the dataset. To that end, we generate a next state $\bar{s'}$ by feeding s to the state transition network, which acts as a signal for $\hat{s'}$. It represents the state the agent will arrive at when it executes the offline policy at state s.

Finally, SDC minimizes the distance from $\hat{s'}$ to $\bar{s'}$ with respect to the parameter of the policy. We use maximum mean discrepancy (MMD) between $\hat{s'}$ and $\bar{s'}$ to denote the distance: $\mathrm{MMD}^2(x, y) = \frac{1}{n^2}\sum_{i,i'} k(x_i, x_{i'}) - \frac{2}{nm}\sum_{i,j} k(x_i, y_j) + \frac{1}{m^2}\sum_{j,j'} k(y_j, y_{j'})$, where $k(\cdot, \cdot)$ is a kernel function. x is the set of size n, which is constructed by the samples generated by the dynamics model M, and y is the set of size m induced by the state transition model U. In our implementation, we use Gaussian kernels and set $n = m = 4$. $\bar{s'}$ is offered as a label for the noisy next state and the gradient does not backpropagate through the state transition model. The parameter of the dynamics model is fixed when we minimize the MMD loss. In this way, SDC encourages the agent to move to familiar regions when the agent encounters a noisy state.

The optimization in the policy improvement is defined as:

$$\begin{aligned}\pi(\cdot|s) := &\max_{\pi} \mathbb{E}_{a\sim\pi(\cdot|s)}\left[Q(s,a)\right] \\ &\text{s.t. } \mathbb{E}_{s\sim\mathcal{D}}[\mathrm{MMD}(M(\hat{s}, \pi(\cdot \mid \hat{s})), U(\cdot \mid s))] \le \eta,\end{aligned} \tag{7}$$

where η is a threshold to control the degree of the MMD loss compared to the policy loss. We choose $\eta = 0.05$ in our experiment. In addition, the policy is trained to maximize the likelihood of the actions which correspond with high Q-values. In our implementation, we use dual gradient descent with Lagrange multiplier α. The policy loss function is translated into:

$$\begin{aligned}L_\pi := L_\pi^{(1)} + L_\pi^{(2)} = &-\mathbb{E}_{a\sim\pi(\cdot|s)}\left[Q(s,a)\right] \\ +\alpha(&\mathbb{E}_{s\sim\mathcal{D}}[\mathrm{MMD}(M(\hat{s}, \pi(\cdot \mid \hat{s})), U(\cdot \mid s))] - \eta),\end{aligned} \tag{8}$$

The two Q-networks are updated by the conservative Q-learning algorithm (Kumar et al. 2020). The target of Q-learning is the minimum of the two Bellman targets. Meanwhile, the Q-networks are updated to maximize the value of in-distribution state-action pairs and minimize the value of out-of-distribution state-action pairs.

$$\begin{aligned}&\min_Q(\mathbb{E}_{s\sim\mathcal{D},a\sim\pi(\cdot|s)}[Q(s,a)] - \mathbb{E}_{s,a\sim\mathcal{D}}[Q(s,a)]) \\ &+ \mathbb{E}_{s,a,s'\sim\mathcal{D}}\left[\left(Q(s,a) - \mathcal{T}Q(s,a)\right)^2\right]\end{aligned} \tag{9}$$

Algorithm 1: State Deviation Correction

Input: offline dataset $\mathcal{D}$, maximal update iterations t_{max}
Parameter: policy network π, Q-networks Q_1, Q_2, dynamics model M, state transition model U,
Output: learnt policy network π

Initialize the policy network, Q-networks, the dynamics model and the state transition model.
Let $t = 0$.
while $t < t_{max}$ **do**
  Sample mini-batch of N samples (s, a, r, s') from $\mathcal{D}$.
  Train the dynamics model M and state transition model U.
  Perturb the state s and get $\hat{s}$ according to Equation 6.
  Feed $\hat{s}$ to the policy π and get $\hat{a}$.
  Feed $\hat{s}$ and $\hat{a}$ to the dynamics model M and get $\hat{s'}$.
  Feed s to the transition model U and get $\bar{s'}$.
  Update the policy π according to Equation 8.
  Update the Q-networks according to Equation 9.
end while

The whole process is summarized in Algorithm 1. The diagram of our proposed method is shown as Figure 2.

It should be noted that our method still differs from the policy-constrained methods even when the additional noise is eliminated. Unlike imitation learning, the policy in offline reinforcement learning is allowed to deviate from the dataset, which enables itself to receive a higher return than the behavior policy. The policy-constrained methods might produce promising actions but care little about whether these actions will produce out-of-distribution states. Our method, on the contrary, constrains the policy to generate actions that lead to in-distribution states.

We now discuss the adoption of different architectures to model the dynamics model and the state transition network. For the state transition model, it is important to fit the distribution and capture the mode of the distribution for generative ability. As with BCQ (Fujimoto, Meger, and Precup 2019) which learns a parametric behavior policy by CVAE, we use CVAE to model the state transition network. For the dynamics model, the ensemble of Gaussian enjoys the highest accuracy in high-dimensional continuous domains than other architectures (Chua et al. 2018).

Experiments

In this section, we expect to understand our proposed method by conducting experiments in diverse offline settings. The main objectives of our evaluation are to answer the following three questions: (1) Does our proposed method actually reduce state deviation compared to the policy-constrained counterparts? (2) How well does SDC perform on the benchmark offline reinforcement learning tasks-Mujoco control suite? (3) Compared with previous methods, is our method less sensitive to the decreasing number of valuable samples? For the dataset, we use a GridWorld setting and the Mujoco datasets in the D4RL benchmarks (Fu et al. 2020). D4RL datasets introduce standard datasets for Hopper, Halfcheetah, and Walker2d benchmarks. For each type of control environment, D4RL provides five kinds of datasets("random", "medium", "medium-replay", "medium-expert", and "expert"). The "random" dataset is produced by rolling out a randomly initialized policy for 10^6 steps. The "medium" dataset is generated by a soft actor-critic policy trained to reach approximately $1/3$ of the expert's performance. The "medium-replay" dataset comprises all samples in the replay buffer during training until the policy reaches the medium level of performance. The "expert" dataset consists of $1,000,000$ samples produced by a soft actor-critic policy, which is trained until convergence. The "medium-expert" dataset is introduced by mixing equal numbers of expert-level samples and medium-level samples.

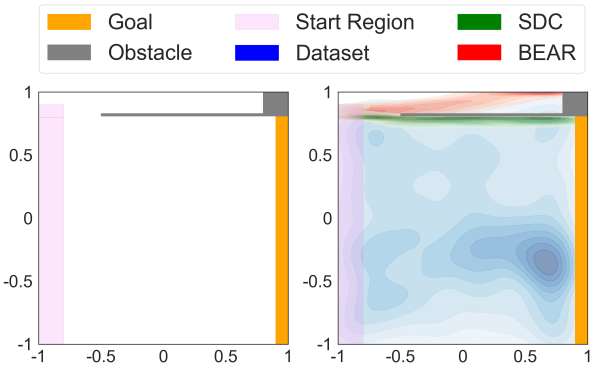


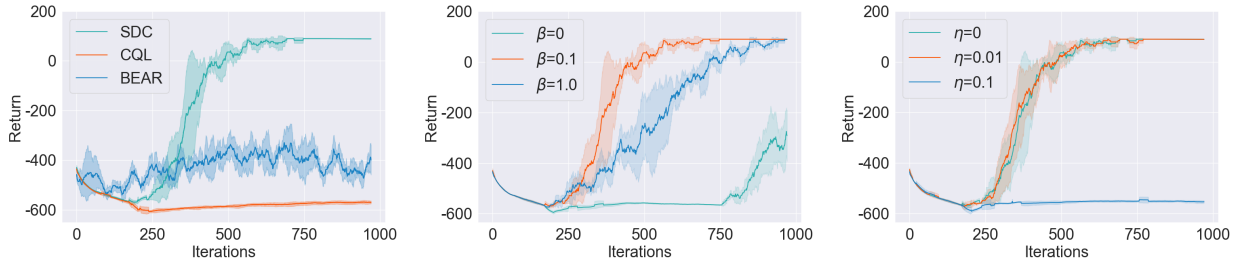
Figure 3: The visualization of a GridWorld example.

Visualization of State Deviation Correction

GridWorld Example To understand the utility of our approach, we deploy the SDC method on a modified GridWorld task. The visualization of the environment is shown in the left part of Figure 3. The map ranges from -1 to 1 in two dimensions. An agent starts from the left side(purple region) of the environment, with the aim to reach the goal district depicted by the orange rectangle. The gray region denotes the obstacle which the agent cannot pass through. The state space is continuous and contains the position of the agent. The action is the agent's velocity. At each time step, the agent will receive a penalty -0.1. When it steps into the goal district, it will get a bonus 100 and the episode is over. The offline dataset is constituted by the samples stored in the replay buffer over the course of the training of a soft-actor-critic agent until convergence. Note that all trajectories start from the positions with $y \leq 0.8$, which simulates the setting where the training data fails to cover the state space.

For offline training, we compare our method with CQL (Kumar et al. 2020) and BEAR (Kumar et al. 2019), which are both policy-constrained methods. Note that when evaluating the algorithms, the agent is initialized at the position whose y-coordinate ranges from 0.8 to 0.9. This distinction from the dataset requires the agent to bypass the obstacle and track the states of the dataset.

We run five instances for each method. The learning



(a) GridWorld Results (b) Ablation on β (c) Ablation on η

Figure 4: (a) The learning curves of the three algorithms SDC, CQL and BEAR on the GridWorld. (b) Ablation study on β. (c) Ablation study on η.

curves shown in Figure 4(a) suggest that our method surpasses the other methods by a large margin. The result corresponds with the state density distribution visualized in the right part of Figure 3. The blue region denotes the state density distribution of the dataset. The green patch and the red patch denote the state distribution of SDC and BEAR, respectively. (We omit the visualization of CQL since its state distribution is similar to BEAR.) Our method learns to constrain the state in the support of the dataset. By contrast, BEAR and CQL simply follow the behavior policy, thus end up failing to perform the task when the initial location is slightly different from the dataset. This study implies that when the dataset can not fully represent the environment, there is a necessity to guarantee the policy's induced state distribution to remain in the support of the static dataset.

Ablation Study We also present ablation studies to study the relationship between the hyperparameters and our proposed method.

The scale of the additional noise with respect to the state affects the performance of our method. Figure 4(b) demonstrates the learning progress of our method when tuning β. It is evident that when β is large, the target is too far from the noisy next state and is not able to act as a valid supervision signal, thus the performance of the algorithm degrades. When β is small, the policy has no chance to be trained on alien states, thus fails to complete the task.

Figure 4(c) demonstrates SDC's performance varies according to the scale of η in the MMD loss. When η is too large, the constraint of Equation 7 is too loose, the algorithm thus fails.

High-dimensional Settings To evaluate the effectiveness of SDC in high-dimensional settings, we investigate how our method constrains the state distribution on D4RL datasets. We compare SDC with an implicit policy-constrained method CQL and a direct policy-constrained method BEAR on "Halfcheetah-medium-expert". We also present the result of "Halfcheetah-expert" in the Appendix.

For each task, we train a CQL agent, a BEAR agent, and an SDC agent for $1,000,000$ updates. Then we collect $100,000$ samples by running these trained policies separately. To visualize the results clearly, we plot the distributions of the offline dataset, CQL-induced dataset and SDC-induced dataset with t-Distributed Stochastic Neighbor Embedding (t-SNE) (Hinton and Roweis 2002) in the left part of Figure 5. The blue region depicts the distribution of the offline dataset, while the red and green represent the dataset generated by the CQL policy and the SDC policy, respectively. We also show the distributions of the offline dataset, BEAR-induced dataset and SDC-induced dataset by a scatter figure(the right part of Figure 5). The blue points denote the offline dataset, while the red and green represent the dataset generated by the BEAR policy and the SDC policy, respectively.

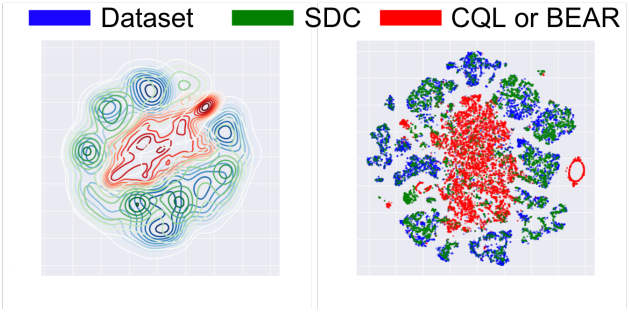


Figure 5: The visualizations of the state distributions of experiments on the "Halfcheetah-medium-expert" task. The left part is CQL(red) vs SDC(green). The right part is BEAR(red) vs SDC(green)

The result shows that CQL and BEAR fail to follow offline state distribution in high-dimensional settings. They only induce state distributions that cover a part of the offline dataset and produce out-of-distribution samples. By contrast, there is a merely slight divergence between the state distribution of the SDC policy and the offline dataset. Our proposed method produces fewer outliers and follows the support of the dataset more accurately.

Offline Mujoco Control Datasets

In this section, we evaluate our proposed method on the Mujoco datasets in the D4RL benchmarks (Fu et al. 2020). We compare our method with BC, SAC (Haarnoja et al. 2018), BEAR (Kumar et al. 2019), BRAC (Wu, Tucker, and Nachum 2019), and MOPO (Yu et al. 2020), which enjoy significant empirical success in offline setting. The results

Task Name	SAC	BC	BEAR	BRAC-p	BRAC-v	CQL	MOPO	SDC
Halfcheetah-random	30.5	2.1	25.5	23.5	28.1	**35.4**	31.9	**36.2±1.3**
Walker2d-random	4.1	1.6	6.7	0.8	0.5	7.0	**13.3**	**14.3±4.5**
Hopper-random	11.3	9.8	9.5	11.1	12.0	10.8	**13.0**	10.6±0.3
Halfcheetah-medium	-4.3	36.1	38.6	44.0	45.4	44.4	40.2	**47.1±0.2**
Walker2d-medium	0.9	6.6	33.2	72.7	**81.3**	79.2	26.5	**81.1±0.4**
Hopper-medium	0.8	29.0	47.6	31.2	32.3	58.0	14.0	**91.3±0.8**
Halfcheetah-medium-replay	-2.4	38.4	36.2	45.6	46.9	46.2	**54.0**	47.3±1.0
Walker2d-medium-replay	1.9	11.3	10.8	-0.3	0.9	26.7	**92.5**	30.3±4.7
Hopper-medium-replay	3.5	11.8	25.3	0.7	0.8	**48.6**	42.7	**48.2±2.7**
Halfcheetah-medium-expert	1.8	35.8	51.7	43.8	45.3	62.4	57.9	**101.3±3.6**
Walker2d-medium-expert	1.9	11.3	10.8	-0.3	0.9	98.7	51.7	**105.3±4.0**
Hopper-medium-expert	1.6	111.9	4.0	1.1	0.8	**111.0**	55.0	**112.9±0.2**
Halfcheetah-expert	-1.9	**107.0**	**108.2**	3.8	-1.1	104.8	-	**106.6±1.1**
Walker2d-expert	-0.3	125.7	106.1	-0.2	-0.0	**153.9**	-	108.3±3.7
Hopper-expert	0.7	**109.0**	**110.3**	6.6	3.7	**109.9**	-	**112.6±0.1**

Table 1: Results of SDC, SAC, BC, BEAR, BRAC, and CQL on offline Mujoco control suite tasks, on the normalized return metric, averaged over four seeds. Note that SDC performs better or similar to other methods on most of the tasks.

Task Name	CQL	SDC
Halfcheetah-random-expert-0.5	67.8±2.2	**88.4±1.3**
Halfcheetah-random-expert-0.6	44.5±4.5	**90.9±2.9**
Halfcheetah-random-expert-0.7	46.7±6.8	**85.2±1.7**
Halfcheetah-random-expert-0.8	42.6±2.1	**88.4±3.2**
Halfcheetah-random-expert-0.9	34.7±3.4	**70.2±5.7**
Walker2d-random-expert-0.5	**88.3±4.9**	**88.8±3.9**
Walker2d-random-expert-0.6	**81.5±6.3**	65.9±5.0
Walker2d-random-expert-0.7	48.3±5.8	**85.5±4.7**
Walker2d-random-expert-0.8	**50.5±7.2**	**49.8±5.6**
Walker2d-random-expert-0.9	17.6±3.0	**26.8±2.1**
Hopper-random-expert-0.5	111.2±0.1	112.8±0.2
Hopper-random-expert-0.6	111.0±0.1	111.6±0.1
Hopper-random-expert-0.7	110.4±0.3	112.0±0.3
Hopper-random-expert-0.8	110.8±0.2	112.3±0.1
Hopper-random-expert-0.9	102.2 ±0.3	**108.5±0.2**

Table 2: Results of CQL and SDC on the datasets with limited valuable samples, on the normalized return metric, averaged over four seeds. Note that SDC outperforms CQL greatly on Halfcheetah datasets.

for BEAR, BRAC, SAC, and BC are obtained by Fu et al. (2020). Results for these 15 tasks are shown in Table 1. We bold the approximate highest scores across all the algorithms. Our proposed method performs similarly or better than the state-of-art methods on most tasks. Especially, our proposed method exceeds the best previous methods by a large margin on "Hopper-medium" and "Halfcheetah-medium-expert" tasks.

Limited Valuable Data

In this section, we try to understand our method's efficacy when presented with limited valuable data. We propose a new task by mixing the expert-level dataset and random-level dataset with different ratios for Halfcheetah, Walker2d, and Hopper. We design five datasets for each environment where the proportions of random samples are 0.5, 0.6, 0.7, 0.8, and 0.9. Accordingly, the ratios of expert samples decline. All datasets consist of $1,000,000$ samples.

We compare our proposed method with CQL on these tasks. The results are shown in Table 2. As expected, the trained policy's performance shows degradation with decreased expert samples. Our proposed method achieves better results than CQL in most tasks. For Halfcheetah tasks, the performance of SDC decreases more slowly as the size of valuable data reduces. For Hopper tasks, our proposed method performs competitively to CQL in the former four tasks and outperforms CQL with a small margin when the random ratio is 0.9. As the results show, our proposed method is less sensitive to the reduction of the expert samples and can yield a policy whose state distribution is close to the support of the dataset.

Conclusions

In conclusion, we introduce the state deviation correction(SDC) algorithm. We attribute the state deviation to the initial data difference, the dynamics bias, and deviation accumulation. We build a dynamics model and a state transition model. By perturbing the state, we obtain a noisy state and feed it to the policy and the dynamics model sequentially. By minimizing the distance of the generated noisy next state to the next state sampled from the state transition model, we encourage the policy to guide the agent to arrive at its familiar regions. In the experiment, we compare our method with current offline reinforcement learning methods on a GridWorld setup, Mujoco control suite tasks, and a modified Mujoco dataset with limited valuable samples. Our method is competitive with the state-of-art methods on these offline reinforcement learning benchmarks.

Acknowledgements

This work was supported by the National Key R&D Program of China under Grant 2018AAA0102800, National Natural Science Foundation of China under Grant 61620106005, and Beijing Municipal Science and Technology Commission under Grant Z201100005820005.

References

Agarwal, R.; Schuurmans, D.; and Norouzi, M. 2019. Striving for simplicity in off-policy deep reinforcement learning.

Berkenkamp, F.; Turchetta, M.; Schoellig, A. P.; and Krause, A. 2017. Safe model-based reinforcement learning with stability guarantees. *arXiv preprint arXiv:1705.08551*.

Bertsekas, D. P.; and Tsitsiklis, J. N. 1995. Neuro-dynamic programming: an overview. In *Proceedings of 1995 34th IEEE conference on decision and control*, volume 1, 560–564. IEEE.

Chua, K.; Calandra, R.; McAllister, R.; and Levine, S. 2018. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *arXiv preprint arXiv:1805.12114*.

Fu, J.; Kumar, A.; Nachum, O.; Tucker, G.; and Levine, S. 2020. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*.

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

Ghasemipour, S. K. S.; Schuurmans, D.; and Gu, S. S. 2021. Emaq: Expected-max q-learning operator for simple yet effective offline and online rl. In *International Conference on Machine Learning*, 3682–3691. PMLR.

Gottesman, O.; Johansson, F.; Meier, J.; Dent, J.; Lee, D.; Srinivasan, S.; Zhang, L.; Ding, Y.; Wihl, D.; Peng, X.; et al. 2018. Evaluating reinforcement learning algorithms in observational health settings. *arXiv preprint arXiv:1805.12298*.

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

Gulcehre, C.; Wang, Z.; Novikov, A.; Le Paine, T.; Gomez Colmenarejo, S.; Zolna, K.; Agarwal, R.; Merel, J.; Mankowitz, D.; Paduraru, C.; et al. 2020. Rl unplugged: Benchmarks for offline reinforcement learning. *arXiv e-prints*, arXiv–2006.

Haarnoja, T.; Zhou, A.; Hartikainen, K.; Tucker, G.; Ha, S.; Tan, J.; Kumar, V.; Zhu, H.; Gupta, A.; Abbeel, P.; et al. 2018. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*.

Hinton, G.; and Roweis, S. T. 2002. Stochastic neighbor embedding. In *NIPS*, volume 15, 833–840. Citeseer.

Jaques, N.; Ghandeharioun, A.; Shen, J. H.; Ferguson, C.; Lapedriza, A.; Jones, N.; Gu, S.; and Picard, R. 2019. Way off-policy batch deep reinforcement learning of implicit human preferences in dialog. *arXiv preprint arXiv:1907.00456*.

Jin, Y.; Yang, Z.; and Wang, Z. 2021. Is Pessimism Provably Efficient for Offline RL? In *International Conference on Machine Learning*, 5084–5096. PMLR.

Kidambi, R.; Rajeswaran, A.; Netrapalli, P.; and Joachims, T. 2020. Morel: Model-based offline reinforcement learning. *arXiv preprint arXiv:2005.05951*.

Kingma, D. P.; and Welling, M. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

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

Kumar, A.; Fu, J.; Tucker, G.; and Levine, S. 2019. Stabilizing off-policy q-learning via bootstrapping error reduction. *arXiv preprint arXiv:1906.00949*.

Kumar, A.; Zhou, A.; Tucker, G.; and Levine, S. 2020. Conservative q-learning for offline reinforcement learning. *arXiv preprint arXiv:2006.04779*.

Lange, S.; Gabel, T.; and Riedmiller, M. 2012. Batch reinforcement learning. In *Reinforcement learning*, 45–73. Springer.

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

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533.

Rajeswaran, A.; Mordatch, I.; and Kumar, V. 2020. A game theoretic framework for model based reinforcement learning. In *International Conference on Machine Learning*, 7953–7963. PMLR.

Riedmiller, M. 2005. Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method. In *European conference on machine learning*, 317–328. Springer.

Sallab, A. E.; Abdou, M.; Perot, E.; and Yogamani, S. 2017. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19): 70–76.

Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; and Moritz, P. 2015. Trust region policy optimization. In *International conference on machine learning*, 1889–1897. PMLR.

Siegel, N. Y.; Springenberg, J. T.; Berkenkamp, F.; Abdolmaleki, A.; Neunert, M.; Lampe, T.; Hafner, R.; Heess, N.; and Riedmiller, M. 2020. Keep doing what worked: Behavioral modelling priors for offline reinforcement learning. *arXiv preprint arXiv:2002.08396*.

Sohn, K.; Lee, H.; and Yan, X. 2015. Learning structured output representation using deep conditional generative models. *Advances in neural information processing systems*, 28: 3483–3491.

Swaminathan, A.; and Joachims, T. 2015. Batch learning from logged bandit feedback through counterfactual risk minimization. *The Journal of Machine Learning Research*, 16(1): 1731–1755.

Swazinna, P.; Udluft, S.; and Runkler, T. 2021. Overcoming model bias for robust offline deep reinforcement learning. *Engineering Applications of Artificial Intelligence*, 104: 104366.

Urpí, N. A.; Curi, S.; and Krause, A. 2021. Risk-Averse Offline Reinforcement Learning. *arXiv preprint arXiv:2102.05371*.

Van Hasselt, H.; Doron, Y.; Strub, F.; Hessel, M.; Sonnerat, N.; and Modayil, J. 2018. Deep reinforcement learning and the deadly triad. *arXiv preprint arXiv:1812.02648*.

Watkins, C. J.; and Dayan, P. 1992. Q-learning. *Machine learning*, 8(3-4): 279–292.

Wu, Y.; Tucker, G.; and Nachum, O. 2019. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*.

Yu, T.; Kumar, A.; Rafailov, R.; Rajeswaran, A.; Levine, S.; and Finn, C. 2021. Combo: Conservative offline model-based policy optimization. *arXiv preprint arXiv:2102.08363*.

Yu, T.; Thomas, G.; Yu, L.; Ermon, S.; Zou, J.; Levine, S.; Finn, C.; and Ma, T. 2020. Mopo: Model-based offline policy optimization. *arXiv preprint arXiv:2005.13239*.

Zhou, L.; Small, K.; Rokhlenko, O.; and Elkan, C. 2017. End-to-end offline goal-oriented dialog policy learning via policy gradient. *arXiv preprint arXiv:1712.02838*.