Model-Based Offline Policy Optimization with Distribution Correcting Regularization

Jian Shen, Mingcheng Chen, Zhicheng Zhang, Zhengyu Yang, Weinan Zhang(✉), and Yong Yu

Shanghai Jiao Tong University, Shanghai, China
{rockyshen,mcchen,zyyang,yyu}@apex.sjtu.edu.cn,
{zhangzhicheng1,wnzhang}@sjtu.edu.cn

Abstract. Offline Reinforcement Learning (RL) aims at learning effective policies by leveraging previously collected datasets without further exploration in environments. Model-based algorithms, which first learn a dynamics model using the offline dataset and then conservatively learn a policy under the model, have demonstrated great potential in offline RL. Previous model-based algorithms typically penalize the rewards with the uncertainty of the dynamics model, which, however, is not necessarily consistent with the model error. Inspired by the lower bound on the return in the real dynamics, in this paper we present a model-based alternative called DROP for offline RL. In particular, DROP estimates the density ratio between model-rollouts distribution and offline data distribution via the DICE framework [45], and then regularizes the model-predicted rewards with the ratio for pessimistic policy learning. Extensive experiments show our DROP can achieve comparable or better performance compared to baselines on widely studied offline RL benchmarks.

Keywords: Offline Reinforcement Learning · Model-based Reinforcement Learning · Occupancy measure

1 Introduction

Reinforcement learning (RL) has achieved great success in various simulated domains, such as Go [33] and video games [11]. However, these promising results mostly rely on numerous online trial-and-error learning. Unfortunately, such an online learning manner is typically costly, even dangerous, and thus cannot be directly applied to complex real-world problems, such as recommender systems [2]. Instead, in real-world applications, there usually exist large and diverse datasets that are previously collected by one or multiple logging policies. These scenarios motivate the study of offline RL [21], also known as batch RL [20], which provides a promising alternative for widespread use of RL. At a colloquial

J. Shen and M. Chen—Equal contribution.

N. Oliver et al. (Eds.): ECML PKDD 2021, LNAI 12975, pp. 174–189, 2021.
https://doi.org/10.1007/978-3-030-86486-6_11

level, offline RL algorithms aim to learn highly rewarding policies by leveraging the static offline datasets without further interactions with environments.

Although it seems to be promising, offline RL faces enormous challenges. Previous works have observed that directly utilizing existing off-policy RL algorithms (*e.g.*, DDPG [22]) in an offline setting without online data collection performs poorly [7]. These failures are mainly caused by evaluating the target Q-function on out-of-training-distribution state-actions in the bootstrapping process [18]. Such distribution shift issue may introduce errors when updating the Q-function, making policy optimization unstable and potentially diverging.

In literature, many efforts have been devoted to mitigating the state-action distribution shift challenge in offline RL. One fruitful line of prior offline RL works consists of model-free algorithms [1,7,14,18,19,26,28,32,40], which constrain the learned policy to be close to the data collecting policy or incorporate conservatism into the Q-function training. However, such model-free algorithms can only learn on the states in the offline dataset, making it overly conservative to learn an effective policy. Therefore, it is necessary to leave the offline data support for better policy optimization. To that end, model-based counterparts have been studied to offer such a possibility by using a dynamics model.

However, directly applying model-based techniques to offline settings also suffers from the state-action distribution shift between model learning and model using. Therefore previous model-based offline RL algorithms [16,42] learn a pessimistic model by penalizing the model predicted rewards according to the estimated model uncertainty and then interact with the model to sample transitions for policy training. Due to the capability of sampling states that are not contained in the offline dataset, model-based algorithms have shown to generalize better. However, the existing model-based offline RL methods rely on heuristic uncertainty quantification techniques, and there is no guarantee that the estimated uncertainty is in proportion to the groundtruth model prediction error.

Based on this consideration, in this paper we present density ratio regularized offline policy learning (DROP), a simple yet effective model-based algorithm for offline RL. DROP directly builds upon a theoretical lower bound of the return in the real dynamics, providing a sound theoretical guarantee for our algorithm. To be more specific, DROP leverages the GradientDICE technique [45] to estimate the density ratio between the model rollout distribution and the offline data distribution, and then regularizes the model predicted rewards according to the density ratio. As an intuitive example, the reward will be severely penalized if the state-action pair is much more likely to be sampled by the model than in the offline dataset, in which case the penalization could be viewed as inducing a conservative estimate of the value for out-of-support state-action pairs. We show through extensive experiments that our proposed DROP can achieve comparable or better performance compared with prior model-free and model-based offline RL methods on the offline RL benchmark D4RL [6].

2 Preliminary

We first introduce the notations used throughout the paper, and briefly discuss the problem setup of offline RL and the basic idea of model-based RL.

2.1 Markov Decision Processes

A Markov decision process (MDP) is defined by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, r, \gamma, \mu_0)$, where $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces, respectively. $\mu_0(s)$ denotes the initial state distribution, and $\gamma \in (0, 1)$ is the discount factor. $T(s' \mid s, a)$ is the transition density of state s' given action a made under state s, and the reward function is denoted as $r(s, a)$. In (online) reinforcement learning, an agent interacts with the MDP, which is also called its environment, and aims to learn a policy $\pi(a|s)$ to maximize the expectation of the return (the sum of discounted rewards) with the collected transitions (s, a, s', r):

$$\max_{\pi} \mathcal{J}(\pi, \mathcal{M}) := \mathbb{E}_{s_0 \sim \mu_0, a_t \sim \pi(\cdot|s_t), s_{t+1} \sim T(\cdot|s_t, a_t)} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right]. \tag{1}$$

For a policy π, we define its discounted state visitation distribution on the real dynamics as $\mu_T^\pi(s) := (1-\gamma) \sum_{t=0}^{\infty} \gamma^t P_{T,t}^\pi(s)$, where $P_{T,t}^\pi(s)$ is the probability of visiting state s at time t. Similarly, we also define the normalized occupancy measure [12] of policy π on dynamics T: $\rho_T^\pi(s, a) := (1-\gamma) \cdot \pi(a \mid s) \sum_{t=0}^{\infty} \gamma^t P_{T,t}^\pi(s)$, and then we have $\rho_T^\pi(s, a) = \pi(a|s) \mu_T^\pi(s)$. Using this definition, we can equivalently express the RL objective as follows:

$$\mathcal{J}(\pi, \mathcal{M}) = \mathbb{E}_{\rho_T^\pi(s,a)}[r(s, a)] = \int \rho_T^\pi(s, a) r(s, a) \, \mathrm{d}s \, \mathrm{d}a. \tag{2}$$

2.2 Offline RL

In offline RL (also known as batch RL [20]), the agent cannot interact with the environment to collect additional transitions. Instead, the agent is provided with a static dataset of transitions $\mathcal{D}_b = \{(s_i, a_i, s_i', r_i)\}_{i=1}^N$, which is collected by one or a mixture of behavior policies, also called logging policies, denoted by π_b. In this case, the state-action pairs in dataset $\mathcal{D}_b$ can be viewed as sampling from the distribution $\rho_T^{\pi_b}$. Typically, we do not assume the behavior policy is known in the formulation of offline RL, but we can approximate it via imitation learning over the dataset $\mathcal{D}_b$ if needed. The goal of offline RL is still to search a policy that maximizes the objective function $\mathcal{J}(\pi, \mathcal{M})$ in Eq. 1 or 2.

2.3 Model-Based RL

In practice, the groundtruth state transition T is unknown and model-based RL methods learn a dynamics model $\hat{T}$ to mimic the real one via maximum likelihood estimation, using the data $\mathcal{D}_b$ collected in the real environment. If the reward function r is unknown, a reward function $\hat{r}$ can also be learned in the dynamics model. Similarly, we define $\rho_{\hat{T}}^\pi(s, a)$ to represent the discounted occupancy measure visited by π under the model $\hat{T}$. Once the model is learned, we can construct a model MDP $\hat{\mathcal{M}} = (\mathcal{S}, \mathcal{A}, \hat{T}, \hat{r}, \gamma, \mu_0)$. Then subsequently, any off-the-shelf model-free algorithms can be used to interact with the model MDP to find the optimal policy: $\pi^* = \arg\max_\pi \mathcal{J}(\pi, \hat{\mathcal{M}})$. Besides, one can also use planning algorithms [4,27] in the model to derive a high-performance agent.

3 A Lower Bound of the True Expected Return

As discussed above, model-based offline RL methods aim to optimize $\mathcal{J}(\pi, \mathcal{M})$ by optimizing $\mathcal{J}(\pi, \hat{\mathcal{M}})$ instead. However, it is not guaranteed that the policy optimized under the model MDP $\hat{\mathcal{M}}$ can achieve good performance in the real environment $\mathcal{M}$ due to the potential model bias. Unfortunately, the model bias is inevitable due to the state-action distribution shift between the offline data and the model rollout trajectories, and in the offline settings this phenomenon is more severe since the error can not be corrected by collecting new data. Therefore, it is important to control the trade-off between the potential gain in performance by leaving the offline data support and the consequent increased model bias.

To overcome this challenge, previous works like MOPO [42] construct a lower bound of the true expected return $\mathcal{J}(\pi, \mathcal{M})$ in the following form:

$$\mathcal{J}(\pi, \mathcal{M}) \geq \mathcal{J}(\pi, \hat{\mathcal{M}}) - C. \tag{3}$$

Once the lower bound is constructed, one can naturally design algorithms to optimize the RL objective by maximizing the lower bound. Following the theoretical analysis in previous works, we first present the following lemma.

Lemma 1. *([23], Lemma 4.3; [42], Lemma 4.1; [31], Lemma E.1) Let two MDPs $\mathcal{M}$ and $\hat{\mathcal{M}}$ share the same reward function $r(s,a)$, but have two different dynamics transition functions $T(\cdot|s,a)$ and $\hat{T}(\cdot|s,a)$, respectively. Define $G^{\pi}_{\hat{T}}(s,a) := \mathbb{E}_{s' \sim \hat{T}(\cdot|s,a)}[V^{\pi}_{T}(s')] - \mathbb{E}_{s' \sim T(\cdot|s,a)}[V^{\pi}_{T}(s')]$. For any policy π, we have*

$$\mathcal{J}(\pi, \hat{\mathcal{M}}) - \mathcal{J}(\pi, \mathcal{M}) = \kappa \cdot \mathbb{E}_{(s,a) \sim \rho^{\pi}_{\hat{T}}}[G^{\pi}_{\hat{T}}(s,a)], \tag{4}$$

where $\kappa = \gamma(1-\gamma)^{-1}$.

MOPO then further bounds the value discrepancy $G^{\pi}_{\hat{T}}(s,a)$ by integral probability metric (IPM) [24]. According to the definition of IPM, let $\mathcal{F}$ be a collection of functions from $\mathcal{S}$ to $\mathbb{R}$, under the assumption that $V^{\pi}_{T} \in \mathcal{F}$, we have

$$G^{\pi}_{\hat{T}}(s,a) \leq \sup_{f \in \mathcal{F}} \left| \mathbb{E}_{s' \sim \hat{T}(\cdot|s,a)}[f(s')] - \mathbb{E}_{s' \sim T(\cdot|s,a)}[f(s')] \right| =: d_{\mathcal{F}}(\hat{T}(\cdot|s,a), T(\cdot|s,a)), \tag{5}$$

where $d_{\mathcal{F}}$ is the IPM defined by the function class $\mathcal{F}$. By choosing different $\mathcal{F}$, IPM reduces to many well-known distance metrics between probability distributions, such as Wasserstein distance [37] and maximum mean discrepancy [9]. Now it becomes the following lower bound

$$\mathcal{J}(\pi, \mathcal{M}) \geq \mathbb{E}_{s' \sim \hat{T}(\cdot|s,a)}[r(s,a) - \kappa \cdot d_{\mathcal{F}}(\hat{T}(\cdot|s,a), T(\cdot|s,a))]. \tag{6}$$

According to this lower bound, MOPO estimates the uncertainty $u(s,a)$ in model prediction and then reshapes the reward by $\tilde{r}(s,a) = \hat{r}(s,a) - \eta u(s,a)$. However, there exists a neglected gap between the practical algorithm (estimated uncertainty $u(s,a)$) and the theoretical result (real model prediction error $d_{\mathcal{F}}$). To bridge this gap, we now present the main theoretical result in our paper.

Theorem 1. *Let two MDPs $\mathcal{M}$ and $\hat{\mathcal{M}}$ share the same reward function $r(s,a)$, but with different dynamics transition $T(\cdot|s,a)$ and $\hat{T}(\cdot|s,a)$, respectively. Assume there exists a function collection $\mathcal{F}_1 = \{f : \mathcal{S}\times\mathcal{A} \to \mathbb{R} \mid \|f\|_\infty \leq \alpha\}$ such that $G^\pi_{\hat{T}}(s,a) \in \mathcal{F}_1$ and $\mathcal{F}_2 = \{f : \mathcal{S} \to \mathbb{R} \mid \|f\|_\infty \leq \beta\}$ such that $V^\pi_T(s) \in \mathcal{F}_2$, we have that*

$$\mathcal{J}(\pi,\mathcal{M}) \geq \mathcal{J}(\pi,\hat{\mathcal{M}}) - \frac{\kappa\alpha}{\sqrt{2}}\sqrt{D_{KL}(\rho^\pi_{\hat{T}}\|\rho^{\pi_b}_T)} - \frac{\kappa\beta}{\sqrt{2}}\mathbb{E}_{(s,a)\sim\rho^{\pi_b}_T}\left[\sqrt{D_{KL}(T\|\hat{T})}\right]. \quad (7)$$

Proof. According to Lemma 1, we have

$$\begin{aligned}
&\mathcal{J}(\pi,\hat{\mathcal{M}}) - \mathcal{J}(\pi,\mathcal{M}) \\
=&\kappa\cdot\mathbb{E}_{(s,a)\sim\rho^\pi_{\hat{T}}}[G^\pi_{\hat{T}}(s,a)] \\
=&\kappa\cdot\mathbb{E}_{(s,a)\sim\rho^\pi_{\hat{T}}}[G^\pi_{\hat{T}}(s,a)] - \kappa\cdot\mathbb{E}_{(s,a)\sim\rho^{\pi_b}_T}[G^\pi_{\hat{T}}(s,a)] + \kappa\cdot\mathbb{E}_{(s,a)\sim\rho^{\pi_b}_T}[G^\pi_{\hat{T}}(s,a)] \\
\leq&\kappa\cdot\sup_{f\in\mathcal{F}_1}\left|\mathbb{E}_{(s,a)\sim\rho^{\pi_b}_T}[f(s,a)] - \mathbb{E}_{(s,a)\sim\rho^\pi_{\hat{T}}}[f(s,a)]\right| \quad (8) \\
+&\kappa\cdot\mathbb{E}_{(s,a)\sim\rho^{\pi_b}_T}\left[\sup_{g\in\mathcal{F}_2}\left|\mathbb{E}_{s'\sim\hat{T}(\cdot|s,a)}[g(s')] - \mathbb{E}_{s'\sim T(\cdot|s,a)}[g(s')]\right|\right] \\
=&\kappa\alpha\cdot d_{\mathrm{TV}}(\rho^{\pi_b}_T,\rho^\pi_{\hat{T}}) + \kappa\beta\cdot\mathbb{E}_{(s,a)\sim\rho^{\pi_b}_T}[d_{\mathrm{TV}}(\hat{T}(\cdot|s,a),T(\cdot|s,a))],
\end{aligned}$$

where the last inequality holds due to the IPM form of total variation distance. To be more specific, when using the witness function class $\mathcal{F} = \{f : \|f\|_\infty \leq 1\}$, IPM $d_{\mathcal{F}}(\mathbb{P},\mathbb{Q})$ reduces to the total variation $d_{\mathrm{TV}}(\mathbb{P},\mathbb{Q})$ where $\mathcal{P}$ and $\mathcal{Q}$ are two probability distributions. Then by applying Pinsker's inequality, we have

$$\mathcal{J}(\pi,\hat{\mathcal{M}}) - \mathcal{J}(\pi,\mathcal{M}) \leq \frac{\kappa\alpha}{\sqrt{2}}\sqrt{D_{\mathrm{KL}}(\rho^\pi_{\hat{T}}\|\rho^{\pi_b}_T)} + \frac{\kappa\beta}{\sqrt{2}}\mathbb{E}_{(s,a)\sim\rho^{\pi_b}_T}\left[\sqrt{D_{\mathrm{KL}}(T\|\hat{T})}\right], \quad (9)$$

which completes the proof.

□

Theorem 1 gives a lower bound of the true expected return in the true environment. In this bound, the last term corresponds to the model training error on offline data which is sampled from $\rho^{\pi_b}_T$, since training the model via maximum likelihood is an empirical approximation of minimizing the Kullback–Leibler divergence between the model and the true environment. To optimize the first term and third term in this lower bound, we are supposed to train a dynamics model on offline data and then optimize a policy on the model, which forms the naive version of model-based offline RL algorithm. Then the key is how to optimize the second term, which corresponds to the KL divergence between the model rollout data distribution and the offline data distribution. It is intuitively reasonable since this state-action distribution shift problem will lead to large bias in model predicted rollouts and result in a poor policy. Therefore, a principled technique to minimize this distribution shift problem is needed.

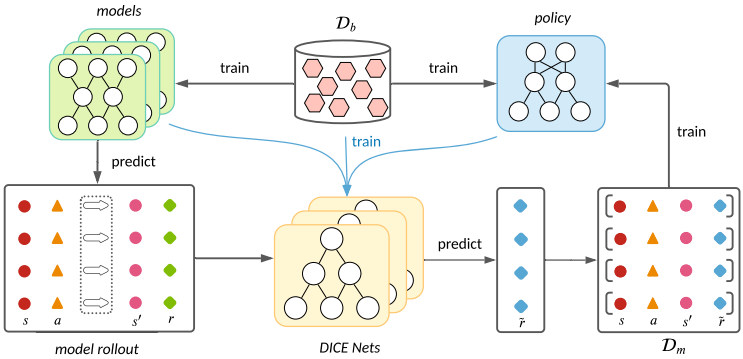


Fig. 1. Illustration of our DROP framework, which uses the distribution ratio correction (DICE) framework to penalize the rewards predicted by the model ensemble.

4 Method

In this section, we give a detailed introduction of our DROP method, which is able to optimize the lower bound in Theorem 1. The overall framework is illustrated in Fig. 1, and the detailed training procedure is shown in Algorithm 1.

4.1 Overall Framework

The primary contribution of DROP is a principled way to minimize the KL divergence between the offline data distribution $\rho_T^{\pi_b}$ and the model rollout data distribution $\rho_{\hat{T}}^{\pi}$. By the definition of KL divergence, the second term can be written as (for clarity, the constant coefficient is neglected):

$$D_{\mathrm{KL}}(\rho_{\hat{T}}^{\pi} \| \rho_T^{\pi_b}) = \mathbb{E}_{(s,a)\sim\rho_{\hat{T}}^{\pi}} \left[\log(\rho_{\hat{T}}^{\pi}(s,a)/\rho_T^{\pi_b}(s,a)) \right]. \tag{10}$$

If we can obtain the real occupancy measure ratio $\tau^*(s,a) = \rho_{\hat{T}}^{\pi}(s,a)/\rho_T^{\pi_b}(s,a)$, we can use the negative log-ratio $-\log \tau^*(s,a)$ as an additional reward when training the policy. In this way, the second term of the lower bound in Theorem 1 can be optimized.

Then it comes to the question of how to obtain the occupancy measure ratio. One direct way to estimate this ratio is constructing a binary classification problem where the model rollout data from $\rho_{\hat{T}}^{\pi}(s,a)$ is labeled positive and the offline data in $\rho_T^{\pi_b}(s,a)$ is labeled negative [43]. In practice, however, the amount of model rollout data and the offline data is not balanced since in model-based methods we hope to use the model to generate much more data to help improve the policy when offline data is limited. This imbalanced classification problem may cause the learning process to be biased [17], which could lead to poor estimation. To sidestep the above obstacle, we will turn to the DICE (DIstribution Correction Estimation) framework to estimate the ratio by exploiting the Markov property of the environments.

Algorithm 1. DROP Algorithmic Framework

Require: Offline dataset D_b, rollout horizon h, reward penalty coefficient η.
1: Learn ensemble dynamics models $\{\hat{T}_\theta^i\}_{i=1}^B : S \times A \mapsto \Pi(S)$ using $D_b(s)$.
2: Randomly initialize π_ϕ, empty buffer $\mathcal{D}_m$.
3: **for** G epochs **do**
4: sample a state $s_0 \sim D_b$.
5: **for** $j = 0, 1, \ldots, h-1$ **do**
6: sample an action $a_j \sim \pi(s_j)$.
7: randomly choose a model $\hat{T}^k$ from $\{\hat{T}_\theta\}_{i=1}^B$ and sample $s_{j+1}, r_j \sim \hat{T}^k(s_j, a_j)$.
8: $u_j = \tau^k(s_j, a_j)$. ▷ Use corresponding DICE net to estimate the ratio
9: $\tilde{r}_j \leftarrow \hat{r}_j - \eta \cdot \log u_j$. ▷ Penalize reward with the calculated ratio
10: add sample $(s_j, a_j, s_{j+1}, \tilde{r}_j)$ to buffer $\mathcal{D}_m$.
11: **end for**
12: Train π_ϕ several times using SAC with mini-batches sampled from $\mathcal{D}_b \cup \mathcal{D}_m$.
13: Train the DICE networks $\{\tau, f, \nu\}$ using $\mathcal{D}_b$, $\{\hat{T}_\theta\}_{i=1}^B$ and π_ϕ.
14: **end for**

By incorporating the ratio estimation and policy regularization into an effective model-based method MBPO [13], we obtain our algorithm DROP. To be more specific, DROP first uses the offline data $\mathcal{D}_b$ to train an ensemble of dynamics models $\{\hat{T}_\theta\}_{i=1}^B$ parameterized by θ, where B is the ensemble size. Given a state-action pair (s, a), the model ensemble can predict the reward $\hat{r}$ and next state $\hat{s}'$. The dynamics model ensemble will be fixed after sufficient training. Then a policy π_ϕ parameterized by ϕ collects samples $\{(s, a, \hat{s}', \hat{r})\}$ by interacting with the dynamics models. To alleviate the distribution shift problem, we use DICE networks to predict the distribution density ratio $u = \tau(s, a)$ and then penalize the predicted reward with the ratio $\tilde{r} = \hat{r} - \eta \cdot \log u$ according to Eq. (10) where η is the coefficient to control the degree of penalty. The modified samples $\{(s, a, \hat{s}', \tilde{r})\}$ are added into the model buffer $\mathcal{D}_m$. Then the policy π_ϕ is trained using data from both $\mathcal{D}_b$ and $\mathcal{D}_m$. After several iterations of policy optimization, we update the DICE networks using data constructed by the offline data, dynamics models and the current policy since the ratio $\rho^\pi_{\hat{T}}(s, a)/\rho^{\pi_b}_T(s, a)$ is related to these three parts. Below we elaborate on several modeling details of model learning, policy optimization and density ratio estimation.

Dynamics Model Learning. We use a bootstrapped ensemble of probabilistic dynamics models $\{\hat{T}_\theta(s'|s, a)\}_{i=1}^B$ to capture both epistemic and aleatoric uncertainty [4]. Each individual dynamics model is a probabilistic neural network which outputs a Gaussian distribution with diagonal covariance conditioned on the state s_n and the action a_n: $\hat{T}_\theta^i(s_{n+1} \mid s_n, a_n) = \mathcal{N}(\mu_\theta^i(s_n, a_n), \Sigma_\theta^i(s_n, a_n))$ where μ and Σ are the mean and covariance, respectively. Each model is trained using the offline data buffer $\mathcal{D}_b$ with a negative log-likelihood loss:

$$\mathcal{L}^i_{\hat{T}}(\theta) = \sum_{n=1}^{N} \left[\mu_\theta^i(s_n, a_n) - s_{n+1}\right]^\top {\Sigma_\theta^i}^{-1}(s_n, a_n) \left[\mu_\theta^i(s_n, a_n) - s_{n+1}\right] + \log\det \Sigma_\theta^i(s_n, a_n). \tag{11}$$

Policy Optimization. Following previous works [4,13], we randomly choose a state from offline data $\mathcal{D}_b$ and use the current policy π_ϕ to perform h-step rollouts on the model ensemble. In detail, at each step, a probabilistic model from the ensemble is selected at random to predict the reward and the next state. The policy is trained on both offline data and model generated data using soft actor-critic (SAC) [10] by minimizing the expected KL-divergence: $\mathcal{L}_\pi(\phi) = \mathbb{E}_s[D_{\mathrm{KL}}(\pi_\phi(\cdot|s) \parallel \exp(Q(s_t,\cdot) - V(s))]$.

4.2 Ratio Estimation via DICE

In an off-policy evaluation where we want to estimate the performance of a target policy using data generated by a behavioral policy, one promising solution is to re-weight the reward by the occupancy measure density ratio. Recently several works have proposed to estimate this ratio such as DualDICE [25], GenDICE [44] and GradientDICE [45]. In this paper, we adapt the GradientDICE architecture to estimate the density ratio in Eq. (10). We will briefly introduce the general framework and refer the readers to [45] for more detail. To begin with, the occupancy measure satisfies the following equation:

$$\rho^\pi_{\hat{T}}(s',a') = (1-\gamma)\mu_0(s')\pi(a'|s') + \gamma \int \rho^\pi_{\hat{T}}(s,a)\hat{T}(s'|s,a)\pi(a'|s')\,\mathrm{d}s\,\mathrm{d}a, \quad (12)$$

where μ_0 is the initial state distribution. Then using $\rho^{\pi_D}_T(s,a)\tau(s,a)$ to replace $\rho^\pi_{\hat{T}}(s,a)$ and minimizing some divergence between the LHS and RHS of Eq. 12 over the function τ with an additional constraint can finally estimate the ratio $\tau^*(s,a)$. Denoting $\delta(s',a') = \gamma \int \rho^{\pi_D}_T(s,a)\tau(s,a)\hat{T}(s'|s,a)\pi(a'|s')\,\mathrm{d}s\,\mathrm{d}a + (1-\gamma)\mu_0(s')\pi(a'|s') - \rho^{\pi_D}_T(s',a')\tau(s',a')$, we have the following objective function

$$\mathcal{L}(\tau) = \frac{1}{2}\mathbb{E}_{\rho^{\pi_D}_T}\left[\left(\frac{\delta(s,a)}{\rho^{\pi_D}_T(s,a)}\right)^2\right] + \frac{\lambda}{2}(\mathbb{E}_{\rho^{\pi_D}_T}[\tau(s,a)] - 1)^2\ , \quad (13)$$

where $\lambda > 0$ is a constant coefficient. By further applying Fenchel conjugate [29] and the interchangeability principle as in [44], optimizing the final objective of GradientDICE is a minimax problem as

$$\begin{aligned}
&\min_\tau \max_{f,\nu} \mathcal{L}_{\mathrm{DICE}}(\tau,\nu,f) \\
&= (1-\gamma)\mathbb{E}_{s\sim\mu_0,a\sim\pi(\cdot|s)}[f(s,a)] + \gamma\mathbb{E}_{(s,a)\sim\rho^{\pi_D}_T,s'\sim\hat{T}(\cdot|s,a),a'\sim\pi(\cdot|s')}[\tau(s,a)f(s',a')] \\
&- \mathbb{E}_{(s,a)\sim\rho^{\pi_D}_T}[\tau(s,a)f(s,a) + \frac{1}{2}f(s,a)^2] + \lambda(\mathbb{E}_{(s,a)\sim\rho^{\pi_D}_T}[\nu\tau(s,a) - \nu] - \frac{1}{2}\nu^2)\ ,
\end{aligned}$$

where we have $\tau : \mathcal{S}\times\mathcal{A} \to \mathbb{R}$, $f : \mathcal{S}\times\mathcal{A} \to \mathbb{R}$ and $\nu \in \mathbb{R}$. In practical implementation, we use feed-forward neural networks to model the function f and τ. Moreover, since we use an ensemble of models and the ratio is related to one specific model, we need to construct a separate DICE network for each dynamics model. Finally, we could use the offline states to approximate the initial state distribution $\mu_0(s)$ since the simulated rollouts are generated starting from some state in $\mathcal{D}_b$.

5 Experiment

Through our experiments, we aim to answer the following questions: i) How does DROP perform compared to prior model-free and model-based offline RL algorithms? ii) How does the quality of uncertainty quantification used in DROP compare against prior offline model-based methods?

5.1 Comparative Evaluation

Compared Methods. We compare DROP with several representative baselines. Firstly, BC (behavior cloning) and SAC-off, directly learning a policy using offline data by supervised learning and soft actor-critic algorithm, serves as basic baselines. For model-free baselines, we compare to BEAR [18], BRAC-v [40], AWR [28] and CQL [19], which are all effective offline RL methods (as discussed in Sect. 6) with CQL achieving SoTA results. For model-based baselines, MBPO [13] and MOPO [42] are taken into comparison. By comparing to MBPO, the effectiveness of reward penalty can be investigated while the comparison with MOPO helps reveal the potential of density ratio as the reward penalty. Note that we don't compare to MOReL [16] to avoid unfair comparison since there are other different factors (*e.g.*policy training algorithm and known reward assumption) besides the reward penalty form. We leave incorporating our density ratio regularization into MOReL framework for future work.

Offline Datasets. All the compared methods are evaluated over 12 offline datasets provided in a large-scale open-source benchmark D4RL [6]. The 12 datasets have different experimental settings with three Gym-MuJoCo tasks (Hopper, Walker2d, and Halfcheetah), and four types of offline datasets. More specifically, the four types of dataset are defined as follows:

(1) The "random" dataset consists of data generated by executing a randomly initialized SAC policy for 1M steps.
(2) The "medium" dataset is generated by executing a suboptimal SAC policy (achieves "medium" level that depends on specific environment) for 1M steps.
(3) The "medium-replay" dataset collects all the samples observed in the replay buffer during the process of training a SAC policy to "medium" level.
(4) The "medium-expert" dataset mixes equal amount of samples generated by executing an "expert" level and a "medium" level SAC policies.

Implementation Details. We implement DROP using TensorFlow[1]. We train the GraidientDICE networks with fixed learning rate $1e-4$ and mini-batch size of 256. The hyperparameter λ in DICE objective is searched in range of $[0.1, 1.0]$, and the reward penalty coefficient is searched in $\{0.1, 0.5, 1.0, 2.0, 5.0\}$. We pretrain the DICE net with offline dataset and initialized SAC policy for

[1] Experiment code can be found at https://github.com/cmciris/DROP.

Table 1. Evaluation results on D4RL benchmark. The values reported are normalized scores roughly to the range between 0 and 100, which are calculated with $normalized\ score = 100 \times \frac{score - random\ score}{expert\ score - random\ score}$ according to [6]. Results of model-free offline methods and BC are taken from the original paper of D4RL [6]. MOPO and DROP are evaluated over six random seeds. We bold the best, and underline the second results across all methods.

Dataset	Env.	DROP	MOPO	MBPO	BC	SAC-off	BEAR	BRAC-v	AWR	CQL
random	hopp.	**18.2**	11.2	4.5	9.8	11.3	11.4	<u>12.2</u>	10.2	10.8
medium	hopp.	<u>52.1</u>	26.3	4.9	29.0	0.8	<u>52.1</u>	31.1	35.9	**58.0**
med-replay	hopp.	**88.2**	<u>78.6</u>	49.8	11.8	3.5	33.7	0.6	28.4	48.6
med-expert	hopp.	54.7	33.9	56.0	**111.9**	1.6	96.3	0.8	27.1	<u>98.7</u>
random	walk.	<u>9.2</u>	**12.1**	8.6	1.6	4.1	7.3	1.9	1.5	7.0
medium	walk.	73.4	11.9	12.7	6.6	0.9	59.1	**81.1**	17.4	<u>79.2</u>
med-replay	walk.	**39.2**	<u>39.0</u>	22.2	11.3	1.9	19.2	0.9	15.5	26.7
med-expert	walk.	65.3	59.2	7.6	6.4	−0.1	40.1	<u>81.6</u>	53.8	**111.0**
random	halfch.	**38.0**	33.8	30.7	2.1	30.5	25.1	31.2	2.5	<u>35.4</u>
medium	halfch.	**50.2**	42.3	28.3	36.1	−4.3	41.7	46.3	37.4	<u>44.4</u>
med-replay	halfch.	**58.6**	54.4	47.3	38.4	−2.4	38.6	<u>47.7</u>	40.3	46.2
med-expert	halfch.	**65.7**	<u>64.2</u>	9.7	35.8	1.8	53.4	41.9	52.7	62.4

adequate steps in the beginning of training. Since log function has extremely negative value when the input is near 0, we clip the ratio into $[\sigma_1, \sigma_2]$, and we search σ_1 from $\{0.01, 0.1\}$ and search σ_2 from $\{10, 20, 50\}$. Besides, we find that other monotonically increasing functions such as $\tanh(\cdot)$ can also be used to replace $\log(\cdot)$ in practice. The model rollout length h used in DROP is set the same as in MOPO for fair comparison.

Results. The comparative results are presented in Table 1. From the comparison we observe that: (i) Our proposed DROP algorithm outperforms the SoTA results in 7 out of the 12 dataset settings, and achieves comparable results (the second best) in other 2 out of 5 dataset. Especially in the halfcheetah environment, DROP shows its superior performance. This verifies the effectiveness of DROP. (ii) DROP performs better than the model-based baseline MOPO in 11 out of the 12, which demonstrates the strength of using density ratio as the reward penalty. (iii) Compared to the SoTA model-free baseline CQL, DROP outperforms it in 8 out of 12 datasets, although it achieves extremely high score in some settings like hopper medium-expert and walker2d medium-expert. By comparison, DROP achieves relatively stable performance across all the settings with hyperparameters searched in a small range while the performance of model-free offline baselines is closely related to the careful tuning in the specific environment and dataset type as mentioned in [40].

5.2 Empirical Analysis

To answer the question ii), we empirically analyze the behaviour of reward penalty used in the two model-based offline methods DROP and MOPO [42]. By

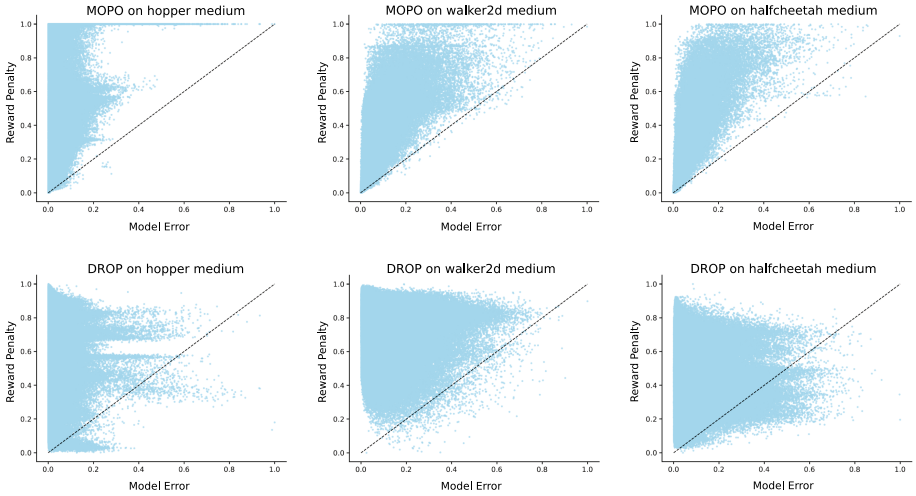
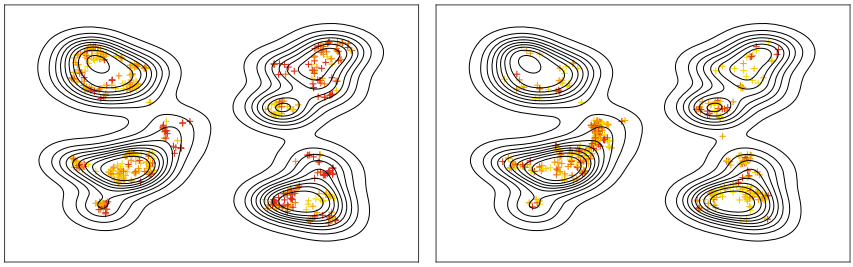


Fig. 2. Visualization of correlation between groundtruth model error and different reward penalty forms. In each Fig. 10000 state-action tuples randomly sampled from the offline dataset are plotted. The top row shows the results of MOPO while the bottom row shows the results of DROP and columns from left to right are the environments of hopper, walker2d, and halfcheetah.

rolling out trajectories on the dynamics model, the agent is allowed to explore around the support of offline data and thus is able to seek possible improvements to behavioral policies without any further interaction with real environment. In this way, how to control the trade-off between informative exploration around the support of offline data and avoiding going far away from the support without exploiting the model deficiency becomes a key problem in model-based offline RL algorithms. MOPO solves this problem by estimating the model uncertainty as the maximum standard deviation of the learned models in the ensemble $u_{\text{MOPO}}(s,a) = \max_i ||\Sigma_\phi^i(s,a)||_{\text{F}}, i \in \{1,2,\ldots,B\}$ and then penalizing the predicted rewards with the uncertainty. However, this uncertainty quantification used in MOPO is not guaranteed to consistently reflect the error between $\hat{T}(\cdot|s,a)$ and $T(\cdot|s,a)$. Differently, instead of using heuristic uncertainty quantification, DROP uses the density ratio $u_{\text{DROP}}(s,a) = \frac{\rho_{\hat{T}}^{\pi}(s,a)}{\rho_{T}^{\pi_b}(s,a)}$ as reward penalty in a principled manner.

To better compare the behavior of different reward penalty forms, we visualize the correlation between the groundtruth model error and the two reward penalty where the model error is computed as $||\hat{T}(s,a) - T(s,a)||_2$, as shown in Fig. 2. We present the comparative results in all three environments, hopper, walker2d, and halfcheetah, on medium dataset, while the results of the other dataset types are similar. We normalize the reward penalty and model errors to $[0,1]$ interval, and thus scattered points should lie along the diagonal line $y = x$ in an ideal situation. As shown in Fig. 2, the scattered points in the fig-



(a) MOPO behaves conservatively. (b) DROP behaves optimistically.

Fig. 3. Visualization of offline data support and different penalized reward. Black lines show the distribution of offline data. We randomly choose 500 state-action pairs from model rollouts, and plot them with colors range form yellow to red according to the reward penalty. Points with high reward penalty are drawn in red while points with low reward penalty are drawn in yellow. (Color figure online)

ures of MOPO tend to be located at the left top of the figures, which indicates that the uncertainty quantification strategy in MOPO is overly conservative. In contrast, in the figures of DROP the scattered points are closer to the diagonal lines compared to MOPO, which means the reward penalty in DROP is less conservative. This may be the reason why DROP performs better than MOPO on D4RL benchmark since MOPO can not effectively utilize the data with low true model error.

To verify the above findings, we further visualize model rollout samples with computed reward penalty and the offline samples with corresponding data support in halfcheetah medium. To be more specific, we embed offline state-action pairs into a 2-dimensional space with the Uniform Manifold Approximation and Projection (UMAP) technique, and then project the model rollout samples using the same mapping. The visualization is shown in Fig. 3. The black lines represent the distribution of offline samples, which is estimated by Kernel Density Estimation (KDE). Thus it can briefly indicates the offline data support on the 2-dimensional space. Again we find that MOPO is quite conservative in assigning the reward penalty. For example, the penalty for the data in the minor mode at the right bottom is quite large. In contrast, DROP is able to give appropriate reward penalty.

6 Related Work

Offline RL [21], also known as batch RL [20], studies the problem of how to train an RL agent only using static offline dataset without any further interactions with the true environment. It means no exploration is allowed and the agent should focus on making full exploitation of existing knowledge. This property makes offline RL technique naturally suitable for practical application scenarios requiring high exploration cost, like healthcare [8,38,41], recommender systems [3,5,34,35], dialogue systems [14,15,46] and autonomous driving [30]. On the

algorithmic front, offline RL methods can be broadly categorized into two groups as described below.

6.1 Model-Free Offline RL

Model-free offline RL algorithms are mostly designed in the principle of keeping the learned policy to stay close to the data collecting policy with explicit [7,14,18,26,40] or implicit [28,32] constraints. According to the specific implementation, model-free offline RL methods can be further grouped into two categories. The first type is Q-value based methods. For example, BCQ [7] optimizes over a subset of actions generated by a trained generative model instead of optimizing over all actions, and thus are less susceptible to over-estimation of Q-values. BEAR [18] employs maximum mean discrepancy (MMD) [9] constraint to make the policy stay close to the behavior policy approximated by a generative model, which guides the target policy to choose those actions that lie in the support of offline distribution. CQL [19] adds regularization in the original Bellman error objective to penalize the Q value of out-of-distribution data, which enables a conservative value function. BRAC [40] penalizes the value function with policy (BRAC-p) or value (BRAC-v) regularization by discrepancy measure (e.g., MMD or KL-divergence) between the learned policy and the behaviour policy. On the other hand, imitation learning based methods like MARWIL [39] and AWR [28] directly train a policy with a reweighted imitation objective. The reweighting coefficient advantage value is estimated by regression and thus the value overestimation due to the Bellman update will not occur.

6.2 Model-Based Offline RL

Model-free offline RL methods can only train the policy with offline data, which may limit the ability to learn a better policy. In contrast, by introducing a dynamics model, model-based offline RL algorithms [16,36,42], is able to provide pseudo exploration around the offline data support for the agent, and thus has potential to learn a better policy with sub-optimal offline dataset. Typically, previous model-based methods rely on uncertainty quantification. MOReL [16] constructs a pessimistic MDP and employs discrepancy of the model prediction over the ensemble as uncertainty to penalize the rewards. A concurrent work MOPO [42] uses maximum learned variance over the ensemble as the uncertainty and performs a similar reward shaping. MOOSE [36] trains a dynamics model as well as a VAE, and uses the reconstruction error as the uncertainty. Similarly, the estimated uncertainty is then used to penalize the policy training. In contrast, our proposed DROP do not rely on the uncertainty quantification. Instead, DROP estimates the density ratio and uses the ratio as the reward penalty to tackle the distribution shift problem.

7 Conclusion

In this paper, we propose a simple yet effective model-based method called DROP for offline reinforcement learning. DROP adds a penalty on rewards to

discourage the policy from visiting out-of-distribution state-action tuples like previous work MOPO [42], but uses a novel reward penalty, *i.e.*, the density ratio between the model generated data distribution and the offline data distribution. This form of reward penalty is directly inspired by the lower bound of the true expected return derived in this paper, and thus has strong theoretical guarantee for policy improvement compared to heuristic uncertainty. We validate the performance of our proposed DROP in widely used benchmark D4RL, and the results show DROP achieves promising performance. One major limitation of DROP is the higher computational complexity introduced by the DICE network training. For future work, we plan to investigate more accurate and quick density ratio estimation strategy to boost the performance and computational efficiency of DROP. Also, incorporating some model-free techniques into DROP may be a promising way to improve performance in some specific datasets, such as walker2d medium-expert.

Acknowledgements. The corresponding author Weinan Zhang is supported by "New Generation of AI 2030" Major Project (2018AAA0100900), Shanghai Municipal Science and Technology Major Project (2021SHZDZX0102) and National Natural Science Foundation of China (62076161, 61772333).

References

1. Agarwal, R., Schuurmans, D., Norouzi, M.: Striving for simplicity in off-policy deep reinforcement learning (2019)
2. Chen, H., et al.: Large-scale interactive recommendation with tree-structured policy gradient. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 3312–3320 (2019)
3. Chen, M., Beutel, A., Covington, P., Jain, S., Belletti, F., Chi, E.H.: Top-k off-policy correction for a reinforce recommender system. In: Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining, pp. 456–464 (2019)
4. Chua, K., Calandra, R., McAllister, R., Levine, S.: Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In: Advances in Neural Information Processing Systems, pp. 4754–4765 (2018)
5. Covington, P., Adams, J., Sargin, E.: Deep neural networks for Youtube recommendations. In: Proceedings of the 10th ACM Conference on Recommender Systems, pp. 191–198 (2016)
6. Fu, J., Kumar, A., Nachum, O., Tucker, G., Levine, S.: D4rl: datasets for deep data-driven reinforcement learning. arXiv preprint arXiv:2004.07219 (2020)
7. Fujimoto, S., Meger, D., Precup, D.: Off-policy deep reinforcement learning without exploration. In: International Conference on Machine Learning, pp. 2052–2062. PMLR (2019)
8. Gottesman, O., et al.: Evaluating reinforcement learning algorithms in observational health settings. Computing Research Repository (CoRR) (2018)
9. Gretton, A., Borgwardt, K.M., Rasch, M.J., Schölkopf, B., Smola, A.: A kernel two-sample test. J. Mach. Learn. Res. **13**(Mar), 723–773 (2012)
10. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor. arXiv preprint arXiv:1801.01290 (2018)

11. Hessel, M., et al.: Rainbow: combining improvements in deep reinforcement learning. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32 (2018)
12. Ho, J., Ermon, S.: Generative adversarial imitation learning. In: Advances in Neural Information Processing Systems, pp. 4565–4573 (2016)
13. Janner, M., Fu, J., Zhang, M., Levine, S.: When to trust your model: model-based policy optimization. In: Advances in Neural Information Processing Systems, pp. 12498–12509 (2019)
14. Jaques, N., et al.: Way off-policy batch deep reinforcement learning of implicit human preferences in dialog. arXiv preprint arXiv:1907.00456 (2019)
15. Karampatziakis, N., Kochman, S., Huang, J., Mineiro, P., Osborne, K., Chen, W.: Lessons from real-world reinforcement learning in a customer support bot. Computing Research Repository (CoRR) (2019)
16. Kidambi, R., Rajeswaran, A., Netrapalli, P., Joachims, T.: Morel: model-based offline reinforcement learning. arXiv preprint arXiv:2005.05951 (2020)
17. Krawczyk, B.: Learning from imbalanced data: open challenges and future directions. Progr, Artif. Intell. **5**(4), 221–232 (2016). https://doi.org/10.1007/s13748-016-0094-0
18. Kumar, A., Fu, J., Tucker, G., Levine, S.: Stabilizing off-policy Q-learning via bootstrapping error reduction. arXiv preprint arXiv:1906.00949 (2019)
19. Kumar, A., Zhou, A., Tucker, G., Levine, S.: Conservative Q-learning for offline reinforcement learning. arXiv preprint arXiv:2006.04779 (2020)
20. Lange, S., Gabel, T., Riedmiller, M.: Batch reinforcement learning. In: Wiering, M., van Otterlo, M. (eds.) Reinforcement Learning, vol. 12, pp. 45–73. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-27645-3_2
21. Levine, S., Kumar, A., Tucker, G., Fu, J.: Offline reinforcement learning: tutorial, review, and perspectives on open problems. arXiv preprint arXiv:2005.01643 (2020)
22. Lillicrap, T.P., et al.: Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971 (2015)
23. Luo, Y., Xu, H., Li, Y., Tian, Y., Darrell, T., Ma, T.: Algorithmic framework for model-based deep reinforcement learning with theoretical guarantees. arXiv preprint arXiv:1807.03858 (2018)
24. Müller, A.: Integral probability metrics and their generating classes of functions. Adv. Appl. Probab. **29**(2), 429–443 (1997)
25. Nachum, O., Chow, Y., Dai, B., Li, L.: DualDICE: behavior-agnostic estimation of discounted stationary distribution corrections. arXiv preprint arXiv:1906.04733 (2019)
26. Nachum, O., Dai, B., Kostrikov, I., Chow, Y., Li, L., Schuurmans, D.: AlgaeDICE: policy gradient from arbitrary experience. arXiv preprint arXiv:1912.02074 (2019)
27. Nagabandi, A., Kahn, G., Fearing, R.S., Levine, S.: Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In: 2018 IEEE International Conference on Robotics and Automation (ICRA), pp. 7559–7566. IEEE (2018)
28. Peng, X.B., Kumar, A., Zhang, G., Levine, S.: Advantage-weighted regression: simple and scalable off-policy reinforcement learning. arXiv preprint arXiv:1910.00177 (2019)
29. Rockafellar, R.T.: Convex Analysis, vol. 36. Princeton University Press, Princeton (1970)
30. Sallab, A.E., Abdou, M., Perot, E., Yogamani, S.: Deep reinforcement learning framework for autonomous driving. Electron. Imaging **2017**(19), 70–76 (2017)

31. Shen, J., Zhao, H., Zhang, W., Yu, Y.: Model-based policy optimization with unsupervised model adaptation. In: Advances in Neural Information Processing Systems, vol. 33 (2020)
32. Siegel, N.Y., et al.: Keep doing what worked: behavioral modelling priors for offline reinforcement learning. arXiv preprint arXiv:2002.08396 (2020)
33. Silver, D., et al.: Mastering the game of go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016)
34. Strehl, A., Langford, J., Kakade, S., Li, L.: Learning from logged implicit exploration data. Computing Research Repository (CoRR) (2010)
35. Swaminathan, A., Joachims, T.: Batch learning from logged bandit feedback through counterfactual risk minimization. J. Mach. Learn. Res. **16**(1), 1731–1755 (2015)
36. Swazinna, P., Udluft, S., Runkler, T.: Overcoming model bias for robust offline deep reinforcement learning. arXiv preprint arXiv:2008.05533 (2020)
37. Villani, C.: Optimal Transport: Old and New, vol. 338. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-71050-9
38. Wang, L., Zhang, W., He, X., Zha, H.: Supervised reinforcement learning with recurrent neural network for dynamic treatment recommendation. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 2447–2456 (2018)
39. Wang, Q., Xiong, J., Han, L., Sun, P., Liu, H., Zhang, T.: Exponentially weighted imitation learning for batched historical data. In: NeurIPS, pp. 6291–6300 (2018)
40. Wu, Y., Tucker, G., Nachum, O.: Behavior regularized offline reinforcement learning. arXiv preprint arXiv:1911.11361 (2019)
41. Yu, C., Ren, G., Liu, J.: Deep inverse reinforcement learning for sepsis treatment. In: 2019 IEEE International Conference on Healthcare Informatics (ICHI), pp. 1–3. IEEE (2019)
42. Yu, T., et al.: MOPO: model-based offline policy optimization. arXiv preprint arXiv:2005.13239 (2020)
43. Zadrozny, B.: Learning and evaluating classifiers under sample selection bias. In: Proceedings of the Twenty-First International Conference on Machine Learning, p. 114 (2004)
44. Zhang, R., Dai, B., Li, L., Schuurmans, D.: GenDICE: generalized offline estimation of stationary values. arXiv preprint arXiv:2002.09072 (2020)
45. Zhang, S., Liu, B., Whiteson, S.: GradientDICE: rethinking generalized offline estimation of stationary values. In: International Conference on Machine Learning, pp. 11194–11203. PMLR (2020)
46. Zhou, L., Small, K., Rokhlenko, O., Elkan, C.: End-to-end offline goal-oriented dialog policy learning via policy gradient. Computing Research Repository (CoRR) (2017)