Bellman Diffusion Models for Offline Reinforcement Learning

Liam Schramm
Department of Computer Science
Rutgers Universty
New Brunswick, NJ
liam.schramm@rutgers.edu

Abdeslam Boularias
Department of Computer Science
Rutgers Universty
New Brunswick, NJ
boularias@gmail.com

Abstract

The state occupancy measure and successor state measure are important theoretical tools in reinforcement learning that represent the distribution of future states. However, while these tools see extensive use in theory and theoretically-motivated algorithms, they have not seen significant use in practical settings because existing algorithms for learning SOM and SSM are high-variance or unstable in practice. To address this, we explore using diffusion models as a representation for the state successor measure. We find that enforcing the Bellman flow constraints on a diffusion model leads to a temporal difference update on the predicted noise, similar to the standard TD-learning update on the predicted reward. As a result, our method has the expressive power of a diffusion model, and a low variance that is comparable to that of TD-learning. To demonstrate this method's practicality, we propose a simple reinforcement learning algorithm based on regularizing the learned SSM. We test the proposed method on an array of offline RL problems, and find it has the highest average performance of all methods in the literature, as well as achieving state-of-the-art performance on several environments.

1 Introduction

The state occupancy measure (SOM) and successor state measure (SSM) are common objects of study in reinforcement learning (RL). A common statement of the objective is to find a policy that induces the SOM with the highest expected reward [13, 12, 15, 14, 16]. The SOM has also received considerable attention in the RL theory community, as a number of provably efficient exploration schemes revolve around regularizing the SOM [2, 11, 23]. The successor state measure (SSM) is a closely related concept, which describes the distribution over future states, given that the agent is currently at state s and takes action a.

These tools have been of particular interest in offline RL, where a central problem is keeping the future state distribution of a policy within the support of the dataset. For this reason, a large number of works have formulated the learning problem as an attempt to imitate the state distribution of the dataset [12, 14, 7]. However, it is difficult to implement this expression of the problem as a learning objective directly, for a number of reasons. If we learn a generative model of the state distribution, it is not clear how to extract the optimal policy from the model. Learning the probability of a given state by regression is challenging, because the learned function does not typically integrate to 1 and so is not a valid probability distribution. For this reason, most other works are forced to use a heavy set of mathematical tricks, based on Fenchel convex dual functions for example, to arrive at a learnable objective. This adds a great deal of technical overhead when adapting the problem to new settings [14, 17]. Additionally, these dual formulations struggle to match the performance of the primal formulations, when primal formulations are possible.

We propose *Bellman Diffusion Models* (BDM), the first SSM estimator that meets the following desideratum.

1. The proposed SSM estimator is off-policy, so it can be learned for arbitrary policies offline.
2. The proposed SSM estimator is generative, so it can be sampled from.
3. An upper bound on the KL divergence between the proposed SSM estimator and arbitrary continuous distributions is easy to calculate.

The combination of these three factors means that it can be calculated and used as part of a regularization term for the primal RL problem, without needing to derive a Fenchel dual formulation, which considerably simplifies its use. We also theoretically analyze this approach, and provide similar convergence guarantees to those that exist for deep Q-learning algorithms.

We present our algorithm for learning the Bellman Diffusion Model (BDM). We then additionally propose an offline RL algorithm called ReBRAC with State Behavior Cloning (ReBRAC-SBC). ReBRAC-SBC starts with ReBRAC (A variant of TD3-BC based on TD3 + a behavior cloning term), then learns a Bellman Diffusion Model and uses it to regularize the divergence between the SSM of the policy and the future trajectory of the state. This effectively encourages the agent to clone the state distribution as well as the action distribution, preventing distribution shift. We find that this method achieves state-of-the-art results on several offline RL tasks and also has the highest average performance of any method in the literature. To summarize, our contributions are as follows:

1. We present Bellman Diffusion Models (BDM), a successor state measure estimator that makes it possible to solve the simpler primal formulation of offline RL problems, instead of the more challenging dual formulation
2. We show that BDMs have the correct distribution as a fixed point of their update rule, which is the same guarantee given for common deep value-learning algorithms
3. We propose an offline RL algorithms based on regularizing the successor state measure, and show it achieves state of the art results.

In addition to our empirical results, we hope that this work lays the foundation for future work based on explicit regularization of the successor state measure and helps to bridge the gap between more traditional RL algorithms and works focused on state occupancy.

2 Background

Diffusion models are a form of generative model that has shown significant success in image generation [8]. In our work, we are primarily concerned with the loss function of diffusion models, and how it can be used to derive a Bellman update. For this reason, we begin with a review of diffusion models and the derivation of the standard diffusion model loss. Diffusion models are trained using a forward process and a backward process. In the forward process, noise is gradually added to a data point until only noise remains, and the data point is distributed as a multivariate unit Gaussian. If the noise is added successively over K steps, then this produces a sequence of increasingly random points from x_0 (a random point from the dataset D) to x_K. Let D be a dataset and x_0 be a data point in D. The probability of a sequence of noised points $x_{0:K}$ is then

$$q(x_{0:K}) = q(x_0) \prod_{i=1}^{K} q(x_i|x_{i-1}, x_0),$$

$q(x_0)$ is defined to be $\frac{1}{|D|}$ for each point in D and 0 for all other x_0. For all other time steps, the forward process probabilities are

$$q(x_i|x_{i-1}) = \mathcal{N}(\sqrt{1-\beta_i}x_{i-1}, \beta_i I),$$

where β_i is the forward variance at the i^{th} step. An advantage of the use of Gaussian noise is that it allows a closed form solution for the distribution after i steps, because the sum of all of additive Gaussian noises up to step i is also Gaussian. If we define $\alpha_i = 1 - \beta_i$ and $\bar{\alpha_i} = \prod_{j=0}^{i} \alpha_j$, then

$$q(x_i|x_0) = \mathcal{N}(\sqrt{\bar{\alpha_i}}x_0, (1-\bar{\alpha_i})I)$$

In the reverse process, a neural network parameterized by weights θ outputs a Gaussian distribution with mean ϵ_θ, predicting the noise that was added during the forward process. The backward process samples a predicted noise from this distribution and this noise is subtracted from the data point. This process repeats for the same number of steps as the forward process. The probability of a sequence of points $x_{0:K}$ in the reverse diffusion process is

$$p(x_{0:K}|\theta) = p(x_K)\prod_{i=1}^{K} p(x_{i-1}|x_i, \theta).$$

The diffusion model is trained to minimize the evidence lower bound. DDPM derives the following loss as an upper bound to the negative log probability of the data [8].[1]

$$E_{x\sim D}[-log(p_\theta(x))] \leq (K-1)E_{x_i}\left[\frac{1}{2\beta_i^2}||\tilde{\mu}(x_i, x_0) - \mu_\theta(x_i, i)||^2\right]$$

This can be reparameterized as follows, so that the neural network outputs ϵ_θ, an estimate of the noise ϵ added to the original sample.

$$E_{x\sim D}[-log(p_\theta(x))] \leq (K-1)E_{i,x_0,\epsilon}\left[\frac{1}{2\alpha_i(1-\bar{\alpha_i})}||\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha_i}}x_0 + \sqrt{1-\bar{\alpha_i}}\epsilon, i)||^2\right]$$

3 Related work

Diffusion models. Diffusion models have seen significant success as a class of generative models for image synthesis [8, 4]. More recently, there has been a growing interest in using diffusion models for imitation learning and reinforcement learning, especially to represent policies. Diffusion planners propose a model in which denoising is analogous to planning, and perform trajectory optimization by denoising [9]. Diffusion Policies extend this approach to behavior cloning [1]. Diffusion Q-learning proposes using diffusion models as an expressive class of policies for offline learning [21].

Learned successor state measures. Successor representations learn the distribution of future states, given the current state and action [3]. γ-models generalize this idea to continuous state distributions by learning a generative representation of future states [10]. Our method differs from γ-models in that γ-models only permit a low-variance score matching loss under special circumstances, when the log probability of the future state distribution can be directly calculated under the model, such as normalizing flows, and the environment dynamics are deterministic. By contrast, our method permits this kind of low-variance backup works for stochastic environments and for diffusion models, which are much more expressive than normalizing flows.

In contemporaneous work, [5] describe a similar learning rule which they call $TD^2 - DD$. They evaluate this rule for future state prediction, but do not explore its application to offline RL or try regularizing this distribution.

Successor state measure and state occupancy measure in imitation learning and offline reinforcement learning. GAIL frames the problem of imitation learning as a problem of state-occupancy matching. It then solves this problem by learning a cost function that maximally separates the real data from the policy data. Minimizing this cost causes the policy to imitate the expert [7]. Unlike some other methods, GAIL is online and assumes that the agent has access to the environment. Another approach to state occupancy matching in imitation learning and offline reinforcement learning is the DICE family of algorithms. AlgaeDICE poses the offline reinforcement

[1]Unlike DDPM, we assume the forward and backward process variances are the same.

learning problem as finding the state occupancy measure with the highest expected reward [17]. It solves this by first applying a Fenchel transform, and then solving the dual problem. In practice, this results in a value estimation problem, with weighted behavior cloning for the policy. SMODICE takes a similar approach to offline imitation learning, applying a Fenchel transform, learning a value function, and using the value function to produce a weighted behavior cloning method [14]

4 Derivation

We derive a temporal difference (TD) update for diffusion models representing the successor state measure. Just as Q-learning enables off-policy training of the value function, this update rule makes it possible to learn the successor measure of arbitrary policies from a fixed data set.

Our derivation consists of two main steps. First, we find an upper bound for the KL divergence between two diffusion models. This result may also be of independent interest. And second, we derive an update rule by minimizing the KL divergence between the successor state measure and its Bellman update, analogous to minimizing the Bellman error in value learning.

4.1 KL Divergence between Diffusion Models

Lemma. *Let q and p be K-step diffusion models with noise schedule β_i, parameterized by neural networks with outputs ϵ_q and ϵ_p, respectively. Let q_i and p_i be the distribution of the samples generated by the first $K - i$ steps of the forward process of q and p, respectively. Then*

$$KL(q_0||p_0) \leq (K-1)E_{i\sim[1,K],x_i\sim q_i}\left[\frac{1}{2\alpha_i(1-\bar{\alpha_i})}||\epsilon_q(x_i,i)-\epsilon_p(x_i,i)||^2\right]$$

Proof. By Jensen's inequality,

$$KL(q_0||p_0) \leq KL(q_{0:K}||p_{0:K})$$

Now recall the chain rule for KL divergences of joint distributions,

$$\begin{aligned}KL(q_{0:K}||p_{0:K}) &= \underbrace{KL(q(x_K)||p(x_K))}_{=0} + (T-1)E_{i\sim[1,K],x_i\sim q_i}[KL(q(x_{i-1}|x_i)||p(x_{i-1}|x_i))] \\ &= (K-1)E_{i\sim[1,K],x_i\sim q_i}[KL(q(x_{i-1}|x_i)||p(x_{i-1}|x_i))]\end{aligned}$$

Since $q(x_{i-1}|x_i)$ and $p(x_{i-1}|x_i)$ are both normal distributions with variance β_i^2 and means μ_q and μ_p respectively, this expression has the closed form solution

$$(K-1)E_{i\sim[1,K],x_i\sim q_i}\left[\frac{1}{2\beta_i^2}||\mu_q(x_i,i)-\mu_p(x_i,i)||^2\right].$$

Using the reparameterization from DDPM, we may rewrite this in terms of networks that predict the expected noise [8].

$$(K-1)E_{i\sim[1,K],x_i\sim q_i}\left[\frac{1}{2\alpha_i(1-\bar{\alpha_i})}||\epsilon_q(x_i,i)-\epsilon_p(x_i,i)||^2\right]$$

□

4.2 Bellman Update

Let M be a Markov Decision Process with state space S, action space A, transition distribution T, reward function R, and discount rate γ.

We consider the successor measure of a state and action $d^\pi(s_f|s,a)$, where s_f is some future state. This describes the probability that an agent following the policy π will stop at the state s_f if it begins in state s, takes action a, and has a $(1-\gamma)$ chance of stopping after taking each action. The action-conditioned value function $Q_\pi(s,a)$ is the expected reward of the distribution $d^\pi(\cdot|s,a)$,

$\frac{1}{1-\gamma} E_{s_f \sim d^\pi(\cdot|s,a)} [R(s_f)]$. The successor measure of a given policy is the unique probability distribution satisfying the Bellman flow constraints. These constraints are as follows:

$$d^\pi(s_f|s,a) = (1-\gamma)T(s' = s_f|s,a) + \gamma E_{a' \sim \pi(s'), s' \sim T(\cdot|s,a)}[d^\pi(s_f|s',a')]$$

We aim to learn a diffusion model $d^\pi_\theta(s_f|s,a)$, which is the distribution obtained by performing the forward diffusion process using the network $\epsilon_\theta(s_f, s, a, i)$. We do this by deriving an upper bound on the KL divergence between the left and right-hand sides of the Bellman flow equation, and then minimizing it by gradient descent. As is standard for deep RL, we will use a target network $\epsilon_{\text{target}}(s_f, s, a, i)$ for the right hand side of the Bellman equation.

$$KL_{Bellman} = KL(d^\pi_\theta(s_f|s,a)||(1-\gamma)T(s' = s_f|s,a) + \gamma E_{a' \sim \pi(s'), s' \sim T(\cdot|s,a)}[d^\pi_{\text{target}}(s_f|s',a')])$$

First, we note that the KL divergence is convex in both arguments. Then by Jensen's inequality,

$$\begin{aligned} KL_{Bellman} &= KL(d^\pi_\theta(s_f|s,a)||(1-\gamma)T(s' = s_f|s,a) + \gamma E_{a' \sim \pi(s'), s' \sim T(\cdot|s,a)}[d^\pi_{\text{target}}(s_f|s',a')]) \\ &\leq (1-\gamma)KL(d^\pi_\theta(s_f|s,a)||T(s' = s_f|s,a)) + \gamma KL(d^\pi_\theta(s_f|s,a)||E_{a' \sim \pi(s'), s' \sim T(\cdot|s,a)}[d^\pi_{\text{target}}(s_f|s',a')]) \\ &\leq (1-\gamma)KL(d^\pi_\theta(s_f|s,a)||T(s' = s_f|s,a)) + \gamma E_{a' \sim \pi(s'), s' \sim T(\cdot|s,a)}[KL(d^\pi_\theta(s_f|s,a)||d^\pi_{\text{target}}(s_f|s',a'))]. \end{aligned}$$

Now, recall that for two distributions p and q, $KL(p||q) = -E_{x\sim q}[log(p(x))] - H(q)$. Then,

$$\begin{aligned} KL_{Bellman} \leq &(1-\gamma)E_{s' \sim T(\cdot|s,a)}[-log(d^\pi_\theta(s'|s,a))] \\ &+ \gamma E_{a' \sim \pi(s'), s' \sim T(\cdot|s,a)}[KL(d^\pi_\theta(s_f|s,a)||d^\pi_{\text{target}}(s_f|s',a'))] - H(T(\cdot|s,a)) \end{aligned}$$

We can now apply the standard diffusion inequality to the first term, and Lemma 1 to the second. Let $s'_i = \sqrt{\bar{\alpha}_i}s' + \sqrt{1-\bar{\alpha}_i}\epsilon$ and $s_{f_i} = \sqrt{\bar{\alpha}_i}s_f + \sqrt{1-\bar{\alpha}_i}\epsilon$.

$$\begin{aligned} KL_{Bellman} \leq &E_{i,\epsilon,s' \sim T(\cdot|s,a)}[\frac{1}{2\alpha_i(1-\bar{\alpha}_i)}[(1-\gamma)||\epsilon - \epsilon_\theta(s'_i, s, a, i)||^2 \\ &+ \gamma E_{a' \sim \pi(s'), s_f \sim d^\pi_{\text{target}}(\cdot|s,a)}[||\epsilon_\theta(s_{f_i}, s, a, i) - \epsilon_{\text{target}}(s_{f_i}, s', a', i)||^2]]] - H(T(\cdot|s,a)) \end{aligned}$$

Since $-H(T(\cdot|s,a))$ does not depend on the parameters of the network, it will not affect the gradient and can be ignored. By dropping this term, we obtain the loss for the Bellman Diffusion Model.

$$\begin{aligned} L_{BDM} = &E_{i,\epsilon,s' \sim T(\cdot|s,a)}[\frac{1}{2\alpha_i(1-\bar{\alpha}_i)}[(1-\gamma)\underbrace{||\epsilon - \epsilon_\theta(s'_i, i)||^2}_{\text{Standard diffusion loss}} \\ &+ \gamma \underbrace{E_{a' \sim \pi(s'), s_f \sim d^\pi_{\text{target}}(\cdot|s,a)}[||\epsilon_\theta(s_{f_i}, s, ai) - \epsilon_{\text{target}}(s_{f_i}, s', a', i)||^2]]]}_{\text{Bellman backup loss}} \end{aligned}$$

The final loss contains two terms: a normal diffusion loss which attempts to make the network predict s', and a Bellman consistency term that attempts to make states that are probable under $d^\pi_{\text{target}}(\cdot|s',a')$ also be probable under $d^\pi_\theta(\cdot|s,a)$

As with DDPM [8], we note that the coefficient $\frac{1}{2\alpha_i(1-\bar{\alpha}_i)}$ does not affect the fixed point of the gradient update, so it may be removed during training without changing the optimal solution.

This learning rule offers two main advantages compared to directly using the cross entropy from the future state distribution. Firstly, the bound from Lemma 1 is lower variance than directly estimating the cross entropy. In the standard DDPM loss, the target value for a given x_i depends on the original unnoised value x_0. Since different values of x_0 can noise to the same x_i, this means the target values for x_i are random. By contrast, the target values of the loss derived in Lemma 1 are deterministic and depend only on x_i. Since we set $\gamma = 0.99$ in our experiments, this means that we eliminate about 99% of the target value variance from our training objective. This is effectively the same tradeoff made in using TD learning for the value function – we exchange an unbiased, high-variance estimator for a potentially biased but low-variance estimator. Given the success of TD learning in deep RL, we should expect this to be a very good tradeoff.

Second, the policy from which a' is sampled does not need to be the same policy used to gather the dataset. This means we can train the Bellman Diffusion Model off-policy by using the target network to generate samples, and then using the above loss to learn to generate those samples.

4.3 Algorithm

We now present an algorithm for learning d^π_θ.

Algorithm 1 Calculate d^π Loss

```
**Have:** Networks $\epsilon_\theta$, $\epsilon_{\text{target}}$ number of diffusion steps $K$, policy $\pi$;
**Input:** $(s, a, s')$
Sample $a' \sim \pi(s')$
Sample $i \sim Uniform([1, K])$
Sample $\epsilon \sim \mathcal{N}(0, 1)$
$s_f \sim d^\pi_{\text{target}}(\cdot|s', a')$
$s'_i = \sqrt{\bar{\alpha}_i} s' + \sqrt{1 - \bar{\alpha}_i}\epsilon$
$s_{f_i} = \sqrt{\bar{\alpha}_i} s_f + \sqrt{1 - \bar{\alpha}_i}\epsilon$
$L_1 = ||\epsilon - \epsilon_\theta(s'_i, s, a, i)||^2$
$L_2 = ||\epsilon_{target}(s_{f_i}, s', a', i) - \epsilon_\theta(s_{f_i}, s, a, i)||^2$
$L = (1 - \gamma)L_1 + \gamma L_2$
**return** $L$;
```

This is a simple change to the standard diffusion loss. Once we sample a state, action, next state tuple, we additionally find the next action a', and use it to sample a future state s_f from our target network. Then, we noise both the next state s' and the future state s_f. We find the squared error for both, using the normal diffusion target ϵ for the network at s'_i and the TD target $\epsilon_{target}(s_{f_i}, s', a', i)$ for the network at s'_{f_i}. Finally, we weight the losses by γ and $1 - \gamma$ respectively, and return their sum.

5 Theoretical analysis

A number of questions can now be asked about the proposed method, but perhaps the most pressing is whether this learns the correct distribution of future states. Since this method is related to Deep TD, full guarantees are not generally possible, as this family of methods can diverge in some cases [22]. However, it is possible to show that the optimal d^π_θ is a fixed point of the update operator, which is the same guarantee typically given for deep TD algorithms. We present the proof of this below.

Because we update ϵ_θ by gradient descent, there is a fixed point when the gradient is zero. Thus, to show that the fixed point of a Bellman diffusion model is the same as the normal diffusion model, we begin with the normal diffusion model, and show that if the gradient of the standard diffusion loss is zero and $d^\pi_\theta = d^\pi_{\text{target}} = d^\pi$, then the gradient of the Bellman diffusion loss is also zero.

Proposition. *Let* $L_{DDPM} = E_{x_0 \sim d^\pi}[||x_0 - x_\theta(x_i, s', \pi(s'), i)||^2]$. *Suppose* $d^\pi_\theta = d^\pi_{target} = d^\pi$ *and* $x_{target}(s, a, i) = E[x_0|x_i, s_i, a_i, i]$. *Then* $\nabla_\theta L_{DDPM} = 0$ *if and only* $\nabla_\theta L_{BDM} = 0$.

Proof:

For convenience, we use the form of both losses that predicts x_0, rather than the version that predicts ϵ. The proof is possible either way, but this form avoids needless calculation.

It is clear from the standard DDPM loss that the loss-minimizing value for the x_θ is $E[x_0|x_i, s, a, i]$.

Observe that

$$\begin{aligned}
L_{DDPM}(i,s,a) &= E_{x_0\sim d^\pi(\cdot|s,a)}[||x_0 - x_\theta(x_i,s,a,i)||^2]\\
&= E_{x_0\sim d^\pi(\cdot|s,a)}[||x_0 - x_\theta(x_i,s,a,i)||^2]]\\
&= (1-\gamma)E_{x_0\sim T(|s,a)}[||x_0 - x_\theta(x_i,s,a,i)||^2]\\
&+ \gamma E_{x_0\sim d^\pi_\theta(\cdot|s',a'),a'\sim\pi(s'),s'\sim T(|s,a)}[||x_0 - x_\theta(x_i,s,a,i)||^2]\\
\text{(Total Expectation)} &= (1-\gamma)E_{x_0\sim T(|s,a)}[||x_0 - x_\theta(x_i,s,a,i)||^2]\\
&+ \gamma E_{x_i}[E_{x_0}[||x_0 - x_\theta(x_i,s,a,i)||^2|x_i,s',a',i]]\\
\text{(Bias-variance decomposition)} &= (1-\gamma)E_{x_0\sim T(|s,a)}[||x_0 - x_\theta(x_i,s,a,i)||^2]\\
&+ \gamma E_{x_i}[||E[x_0|x_i,s',a',i] - x_\theta(x_i,s,a,i)||^2 + Var(x_0|x_i,s',a',i)]\\
\text{(Optimality of } x_{target}) &= (1-\gamma)E_{x_0\sim T(|s,a)}[||x_0 - x_\theta(x_i,s,a,i)||^2]\\
&+ \gamma E_{x_i}[||x_{target}(x_i,s',a',i) - x_\theta(x_i,s,a,i)||^2 + Var(x_0|x_i,s',a',i)]\\
&= L_{BDM} + \gamma Var(x_0|x_i,s',a',i)
\end{aligned}$$

As we can see, the DDPM loss decomposes to the Bellman Diffusion loss, plus a variance term when $x_{target}(s,a,i) = E[x_0|x_i,s_i,a_i,i]$. We can see then that not only does BDM have the correct value as a fixed point, but it is a lower-variance estimator than standard DDPM.

6 Offline Reinforcement Learning Algorithm

A central problem in offline RL is that, although it is possible to imitate the expert policy on the dataset, errors take the policy away from the dataset to new states where it cannot recover. Typically, it is difficult to control this future behavior because it is difficult to predict the future state occupancy of the agent.

We propose a simple solution: rather than performing behavior cloning on the actions alone, we perform behavior cloning on the actions and future states. Intuitively, this ensures that the future distribution of states remains close to the dataset, minimizing distribution shift. We do this by adding the cross entropy of the actual future state distribution and the Bellman Diffusion Model as a regularization term on the loss. Since the BDM depends on the action given by the policy, this loss can be backpropagated through the BDM to π. In Appendix A, we show that this regularization scheme can be viewed as an upper bound to the KL divergence between the data distribution and the policy's state-action occupancy measure for ergodic MDPs.

We base our algorithm on ReBRAC, a state of the art method in offline reinforcement learning [20]. We propose modifying ReBRAC to include this imitation loss, in place of the standard behavior cloning loss to encourage the policy to stay within the support of the dataset. We call this algorithm ReBRAC-SBC. This method can also be adapted for imitation learning by simply removing the value term and focusing on the imitation loss alone.

While other methods use a similar objective function, either in imitation learning or in offline reinforcement learning, it has not previously been possible to directly optimize this objective. Instead, other methods take the convex or Fenchel dual of the problem and optimize that, which typically results in a point-reweighting objective [7, 14, 16, 12, 15]. Unfortunately, many methods derived by this method, such as the DICE family of methods, fail to achieve results competitive with standard actor critic methods [18]. Our method allows us to control the distribution shift encountered in offline RL without abandoning the more performant primal problem.

7 Experiments

To evaluate ReBRAC-SBC, we compare its performance against ReBRAC with the standard behavior cloning loss. We focus on three D4RL tasks: hopper, halfcheetah, and walker2d [6]. We evaluate all methods on four datasets for each task.

1. One million data points from a partially-trained RL agent (medium)

Algorithm 2 ReBRAC-SBC policy loss

```
**Have:** Network $\epsilon_\theta$, state space with dimension $d_S$, diffusion generation algorithm $G$, diffusion model network $\epsilon_\theta$, Q-networks $Q_{\theta_1}$ and $Q_{\theta_2}$, imitation loss weights $w_s$(for states) and $w_a$ (for actions)
**Input:** Training batch $\{(s, a, s')\}$
$\{$Calculate action imitation loss$\}$
$L_a = \frac{||a-\mu_\theta(s)||}{2\sigma^2}$
$\{$Calculate state imitation loss$\}$
Sample future visited state $s_f$ from future trajectory of $s$
Sample $i \sim Uniform([1, K])$, where $K$ is the number of diffusion steps
Sample $\epsilon \sim \mathcal{N}(0, I_{d_S})$
$s_{f_i} \leftarrow \sqrt{\bar{\alpha}_i} s_f + \sqrt{1-\bar{\alpha}_i}\epsilon$
$a_\pi \sim \mathcal{N}(\mu_\theta(s), \sigma^2 I_{d_A})$
$\eta_i = \beta_i^2 \alpha_i(1-\alpha_i)$
$L_s \leftarrow \eta_i ||\epsilon - \epsilon_\theta(s_{f_i}, s, a_\pi, i)||^2$
$\{$Calculate value loss$\}$
$L_Q \leftarrow \min(Q_{\theta_2}(s, a_\pi), Q_{\theta_2}(s, a_\pi))$
**return** $L = L_Q + w_s L_s + w_a L_a$;
```

Table 1: D4RL performance. Reported values are the last-iterate return averaged across all training seeds. The ± symbol represents the standard deviation across seeds. All results other than ReBRAC-SBC are taken from [20]

.

Table 2: D4RL performance

Task Name	TD3+BC	IQL	CQL	SAC-RND	ReBRAC	ReBRAC-SBC
halfcheetah-medium	54.7 ± 0.9	50.0 ± 0.2	46.9 ± 0.4	**66.4** ± **1.4**	65.6 ± 1.0	65.0 ± 0.8
halfcheetah-expert	93.4 ± 0.4	95.5 ± 2.1	97.3 ± 1.1	102.6 ± 4.2	**105.9** ± **1.7**	**105.9** ± **0.7**
halfcheetah-medium-expert	89.1± 5.6	92.7 ± 2.8	95.0 ± 1.4	**108.1**± **1.5**	101.1 ± 5.2	105.8 ± 1.8
halfcheetah-medium-replay	45.0 ± 1.1	42.1 ± 3.6	45.3 ± 0.3	51.2 ± 3.2	51.0 ± 0.8	**54.7** ± **0.5**
hopper-medium	60.9 ± 7.6	65.2 ± 4.2	61.9 ± 6.4	91.1 ± 10.1	**102.0** ± **1.0**	101.6 ± 0.3
hopper-expert	109.6 ± 3.7	108.8 ± 3.1	106.5 ± 9.1	109.8 ± 0.5	100.1 ± 8.3	**111.5** ± **0.7**
hopper-medium-expert	87.8 ± 10.5	85.5 ± 29.7	96.9 ± 15.1	**109.8** ± **0.6**	107.0 ± 6.4	107.3 ± 2.7
hopper-medium-replay	55.1 ± 31.7	89.6 ± 13.2	86.3 ± 7.3	97.2 ± 9.0	98.1 ± 5.3	**101.1** ± **1.2**
walker2d-medium	77.7 ± 2.9	80.7 ± 3.4	79.5 ± 3.2	**92.7** ± **1.2**	82.5 ± 3.6	88.7 ± 1.5
walker2d-expert	110.0 ± 0.6	96.9 ± 32.3	109.3 ± 0.1	104.5 ± 22.8	**112.3** ± **0.2**	111.8 ± 0.1
walker2d-medium-expert	110.4 ± 0.6	112.1 ± 0.5	109.1 ± 0.2	104.6 ± 11.2	**111.6** ± **0.3**	110.7 ± 0.3
walker2d-medium-replay	68.0 ± 19.2	75.4 ± 9.3	76.8 ± 10.0	**89.4** ± **3.8**	77.3 ± 7.9	87.9 ± 3
Average	80.1	82.9	84.2	94.0	92.9	**96.0**

2. The full medium dataset as well as one million datapoints from the agent's replay buffer (medium-replay)

3. One million datapoints from a fully-trained RL agent (expert)

4. Union of the medium and expert datasets (medium-expert)

All methods are trained for 10^6 steps. For the most part, hyperparameters are taken from ReBRAC. However, the action behavior cloning weight, the state behavior cloning weight, and the actor and diffusion model learning rates were tuned. We found that for some environments, adding state behavior cloning allowed us to reduce the action behavior cloning weight, giving the model greater freedom to pursue high rewards. Additionally, we found that the state behavior cloning loss could still be too high variance for the policy, so we experimented with decreasing the actor learning rate when this led to instability.

We find that ReBRAC-SBC outperforms all baselines on three environments (halfcheetah-medium-replay, hopper-expert, and hopper-medium-replay) and ties with another method for a fourth (halfcheetah-expert). For another five environments, ReBRAC-SBC has the second highest expected reward of the group. As a result, ReBRAC-SBC has the highest average reward of the methods. Additionally, ReBRAC-SBC's reward has a substantially lower variance, which may indicate greater stability as a result of better regularization.

8 Conclusion

Optimization of the state occupancy measure and successor state measure are topics of great interest to offline reinforcement learning research, because they offer a valuable theoretical framework for controlling distribution shift. However, methods based on this approach have typically been held back by the difficulty of estimating these distributions. We propose a low-variance off-policy TD-based learning algorithm for estimating the state successor measure, and show that it can be used to regularize existing offline RL algorithms. This makes it possible to solve the primal problem by directly optimizing the state occupancy measure, instead of resorting to the Fenchel dual problem as many other methods do. This approach opens up a number of new possibilities for both online and offline reinforcement learning, as state occupancy formulations can more easily be proposed, experimented with, and combined with existing RL methods. Unlike GAIL or the DICE family of algorithms, the method we propose can be used, modified, and extended without lengthy derivation or extensive knowledge of convex analysis techiniques like convex duality [7, 17, 14]. Moreover, we find that the proposed algorithm has strong performance on the D4RL offline reinforcement learning dataset, and achieves new records on multiple environments.

References

[1] C. Chi, S. Feng, Y. Du, Z. Xu, E. Cousineau, B. Burchfiel, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *Proceedings of Robotics: Science and Systems (RSS)*, 2023.

[2] C. Dann, C.-Y. Wei, and J. Zimmert. Best of both worlds policy optimization, 2023.

[3] P. Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 5(4):613–624, 1993.

[4] P. Dhariwal and A. Nichol. Diffusion models beat gans on image synthesis, 2021.

[5] J. Farebrother, M. Pirotta, A. Tirinzoni, R. Munos, A. Lazaric, and A. Touati. Temporal difference flows, 2025.

[6] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2021.

[7] J. Ho and S. Ermon. Generative adversarial imitation learning, 2016.

[8] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models, 2020.

[9] M. Janner, Y. Du, J. B. Tenenbaum, and S. Levine. Planning with diffusion for flexible behavior synthesis, 2022.

[10] M. Janner, I. Mordatch, and S. Levine. Generative temporal difference learning for infinite-horizon prediction, 2021.

[11] T. Jin, J. Liu, and H. Luo. Improved best-of-both-worlds guarantees for multi-armed bandits: Ftrl with general regularizers and multiple optimal arms, 2023.

[12] J. Lee, W. Jeon, B.-J. Lee, J. Pineau, and K.-E. Kim. Optidice: Offline policy optimization via stationary distribution correction estimation, 2021.

[13] L. Lee, B. Eysenbach, E. Parisotto, E. Xing, S. Levine, and R. Salakhutdinov. Efficient exploration via state marginal matching, 2020.

[14] Y. J. Ma, A. Shen, D. Jayaraman, and O. Bastani. Versatile offline imitation from observations and examples via regularized state-occupancy matching, 2022.

[15] Y. J. Ma, J. Yan, D. Jayaraman, and O. Bastani. How far i'll go: Offline goal-conditioned reinforcement learning via f-advantage regression, 2022.

[16] O. Nachum and B. Dai. Reinforcement learning via fenchel-rockafellar duality, 2020.

[17] O. Nachum, B. Dai, I. Kostrikov, Y. Chow, L. Li, and D. Schuurmans. Algaedice: Policy gradient from arbitrary experience, 2019.

[18] S. Park, K. Frans, S. Levine, and A. Kumar. Is value learning really the main bottleneck in offline rl?, 2024.

[19] Y. Sun. Offlinerl-kit: An elegant pytorch offline reinforcement learning library. `https://github.com/yihaosun1124/OfflineRL-Kit`, 2023.

[20] D. Tarasov, V. Kurenkov, A. Nikulin, and S. Kolesnikov. Revisiting the minimalist approach to offline reinforcement learning, 2023.

[21] Z. Wang, J. J. Hunt, and M. Zhou. Diffusion policies as an expressive policy class for offline reinforcement learning, 2023.

[22] R. J. Williams and L. C. Baird. Analysis of some incremental variants of policy iteration: First steps toward understanding actor-cr. 1993.

[23] J. Zimmert and Y. Seldin. Tsallis-inf: An optimal algorithm for stochastic and adversarial bandits, 2022.

A Additional Proofs

One contribution of this work is bridging the gap between traditional policy gradient methods and DICE methods, which use a state occupancy objective. Here, we explore how our SSM regularization term can be seen as an upper bound on the divergence between the dataset and the state occupancy.

To do this, we begin by formally defining the state-action occupancy measure. Let ρ^π be the state occupancy measure of policy π, defined as

$$\rho^\pi(s_f) = E_{a\sim\pi(\cdot|s)s\sim q}[d^\pi(s_f|s,a)]$$

where q is the distribution of initial states. With a slight abuse of notation, we define

$$\rho^\pi(s,a) = \pi(a|s)\rho^\pi(s)$$

.

We then present an upper bound on the divergence between the data set and the state occupancy measure.

$$\begin{aligned}
KL(D(s,a)||\rho^\pi(s,a)) &= E_{s\sim D}[KL(D(a|s)||\pi(a|s))] + KL(D(s)||\rho^\pi(s)) \\
&= E_{s\sim D}[KL(D(a|s)||\pi(a|s))] + KL(\int_{s_0} D(s_f|s_0)q(s_0)|| \int_{s_0} d^\pi(s_f|s_0)q(s_0))] \\
(\textit{Jensen's Inquality}) &\leq E_{s\sim D}[KL(D(a|s)||\pi(a|s))] + E_{s_0\sim q}[KL(D(s_f|s_0)||d^\pi(s_f|s_0))] \\
(\textit{Jensen's Inquality}) &\leq E_{s\sim D}[KL(D(a|s)||\pi(a|s))] + E_{s_0\sim q, a_0\sim\pi(\cdot|s_0)}[KL(D(s_f|s_0)||d^\pi(s_f|s_0,a_0))]
\end{aligned}$$

In the case that the MDP is ergodic, this can be simplified further. If this is the case, then the occupancy measure for a sufficiently large γ approaches the same distribution regardless of the starting state [?]. As a result, the above inequality is approximately true for any q, because the choice of q does not affect the distribution of D or d^π. We choose $q = D$. Then,

$$KL(D(s,a)||\rho^\pi(s,a)) \lessapprox E_{s\sim D}[KL(D(a|s)||\pi(a|s))] + E_{s\sim D, a\sim\pi(\cdot|s)}[KL(D(s_f|s)||d^\pi(s_f|s,a))] \quad (1)$$

The left term is a standard behavior cloning loss. Recall that TD3 and ReBRAC both learn Gaussian policies with a fixed variance σ^2 and mean μ_θ [? 20]. Then if the action space has dimension d_A, we have

$$\begin{aligned}
KL(D(a|s)||\pi(a|s)) &= E_{a\sim D(\cdot|s)}[-\log(\pi(a|s))] - H(D(a|s)) \\
&= E_{a\sim D(\cdot|s)}\left[\frac{||a-\mu_\theta(s)||^2}{2\sigma^2}\right] + d_A\log(\sigma) + \frac{d_A}{2}\log(\pi) - H(D(a|s)) \\
&\leq E_{a\sim D(\cdot|s)}\left[\frac{||a-\mu_\theta(s)||^2}{2\sigma^2}\right] + d_A\log(\sigma) + \frac{d_A}{2}\log(\pi)
\end{aligned}$$

We can see that this is identical to the TD3 behavior cloning loss, up to constant terms. [2]

The right term of 1 is effectively a behavior cloning term on the state distribution. The KL divergence for this right term can easily be upper bounded using the standard diffusion model loss.

[2]Note that the $\frac{d_A}{2}\log(\pi)$ term is referring to the mathematical constant $\pi = 3.14...$, not the policy π. This term does not depend on the policy

As previously described, we learn a Bellman Diffusion Model to represent $d^\pi(s_f|s,a)$. We then minimize the imitation loss, backpropagating through $d^\pi(s_f|s,a)$ to train the policy.

Combining these two results, we find

$$\begin{aligned} KL(D(s,a)||\rho^\pi(s,a)) \lessapprox\ & E_{s\sim D}[E_{a\sim D(a|s)}\left[\frac{||a-\mu_\theta(s)||^2}{2\sigma^2}\right] \\ & + (K-1)E_{i,\epsilon,s_f\sim D(s_f|s),a\sim\pi(\cdot|s)}[\eta_i||\epsilon-\epsilon_\theta(s_{f_i},s,a_\pi,i)||^2]] + C \end{aligned}$$

where C is a constant that does not depend on the policy.

This is the regularization loss used by ReBRAC-SBM, up to constants scaling the balance between the action and state regularization losses.

B Experiment Details

All experiments were performed on an internal cluster, using a mixture of RTX A4500 and GeForce RTX A4500 GPUs. Each training process was assigned 1 GPU, and took between 12 and 24 hours to complete, requiring approximately 900 GPU hours total across all runs. Hyperparameter tuning took a similar amount of compute. Additionally, several previous iterations of the algorithm were tested, but the compute cost of these experiments was not tracked.

Our implementation was based on OfflineRLKit, and evaluated on the D4RL datasets [19, 6].

For the most part, we reused the hyperparameters from ReBRAC, which were found to work well. We tuned w_s, the state regularization weight, w_a, the action regularization weight, and the actor learning rate. We found that it was necessary to tune the actor and diffusion learning rates because the gradient from the diffusion model is stochastic, unlike the gradient from the Q function and the behavior cloning loss. Reducing the learning rate made convergence much more stable.

In order to reduce the number of hyperparameter search runs, we first turned off the value function training, and did a grid search over w_s, and the actor and diffusion model learning rates. The idea was to first find the optimal mix of state and action regularization, and the learning rate needed for that mix to be stable. Then, we searched for the optimal weighting of the value function, given a fixed mixture of state and action regularization. Since we searched three values each for lr and state regularization and five values for Q value weight, we reduced the number of runs needed from 3*3*5 = 45, to 3*3 + 5 = 14, a nearly 70% reduction.

We refer to ReBRAC's parameters in the following way:

$$\begin{aligned} w_a^0 &: \text{Action regularization coefficient used by ReBRAC} \\ lr_\pi^0 &: \text{Actor learning rate used by ReBRAC} \\ lr_Q^0 &: \text{Critic learning rate used by ReBRAC} \end{aligned}$$

The values we searched for in the initial imitation learning runs were parameterized as follows:

$$\begin{aligned} w_{s/a} &: \text{Relative state regularization coeffiecient} \\ \omega &: \text{Actor and diffusion model learning rate reduction factor} \\ w_a = w_a^0 &: \text{Our action regularization coefficient} \\ w_s = w_{s/a}w_a^0 &: \text{Our state regularization coefficient} \\ lr_\pi = \frac{lr_\pi^0}{\omega} &: \text{Our actor learning rate} \\ lr_d^\pi = \frac{lr_Q^0}{\omega} &: \text{Our diffusion model learning rate} \end{aligned}$$

Our hyperparameter search used values [1,10,100] for both $w_{s/a}$ and lr_reduce. For $w_{s/a}$, we found that values less than 1 were small enough to have effectively no impact. For ω, we only explored values greater than 1 because the use of the diffusion model increased variance, sometimes requiring lower learning rates to maintain stability. Convergence was fast enough that we never needed to increase learning rates.

For the offline RL portion, we used the following parameterizations:

$$\begin{aligned} w_q &: \text{Regularization reduction factor} \\ w_a = \frac{w_a^0}{w_q} &: \text{Our action regularization coefficient} \\ w_s = \frac{w_{s/a} w_a^0}{w_q} &: \text{Our state regularization coefficient} \end{aligned}$$

We searched the values [1,2,4,10,100] for w_q.

We found the following values to be optimal:

Table 3: Tuned hyperparameters for ReBRAC-SBC

Task Name	$w_{s/a}$	ω	w_Q
halfcheetah-medium	100	10	1
halfcheetah-expert	1	1	1
halfcheetah-medium-expert	1	1	1
halfcheetah-medium-replay	10	10	10
hopper-medium	100	100	2
hopper-expert	1	10	1
hopper-medium-expert	100	100	1
hopper-medium-replay	100	100	10
walker2d-medium	1	100	2
walker2d-expert	10	100	1
walker2d-medium-expert	1	1	1
walker2d-medium-replay	100	100	4

In terms of the original hyperparameters, these reduce to:

Table 4: Final hyperparameters for ReBRAC-SBC

Task Name	w_a	w_s	lr_π	$lr_{\text{diffusion}}$
halfcheetah-medium	0.001	0.1	0.0001	0.0001
halfcheetah-expert	0.01	0.01	0.001	0.001
halfcheetah-medium-expert	0.01	0.01	0.001	0.001
halfcheetah-medium-replay	0.001	0.01	0.0001	0.0001
hopper-medium	0.005	0.5	1e-05	1e-05
hopper-expert	0.1	0.1	0.0001	0.0001
hopper-medium-expert	0.1	10.0	1e-05	1e-05
hopper-medium-replay	0.005	0.5	1e-05	1e-05
walker2d-medium	0.025	0.025	1e-05	1e-05
walker2d-expert	0.01	0.1	1e-05	1e-05
walker2d-medium-expert	0.01	0.01	0.001	0.001
walker2d-medium-replay	0.0125	1.25	1e-05	1e-05

C Additional Experimental Results

In addition to our offline learning experiments, we also tried removing the rewards and Q function, and comparing State Behavior Cloning (SBC) alone against ordinary behavior cloning and

SMODICE, an offline imitation learning algorithm [14]. We find that SBC shows strong performance, outperforming BC and SMODICE on five of the twelve environments. Interestingly, SBC appears to be more stable than SMODICE, which collapsed and achieved near-zero reward on 3 of the 4 walker environments, and even crashed on walker2d-medium-expert from numerical overflows.

Table 5: Imitation learning experimental results on the D4RL dataset. Reported values are the last-iterate return averaged across all training seeds. The ± symbol represents the standard deviation across seeds.

Task Name	BC	SMODICE	SBC
halfcheetah-medium	42.51 ± 0.38	**55.99 ± 4.19**	42.21 ± 0.37
halfcheetah-expert	93.17 ± 0.11	**94.10 ± 1.05**	93.05 ± 0.20
halfcheetah-medium-expert	66.83 ± 7.08	**81.96 ± 5.22**	69.07 ± 7.88
halfcheetah-medium-replay	33.97 ± 4.84	**88.45 ± 2.73**	36.67 ± 2.06
hopper-medium	53.00 ± 0.57	57.01 ± 3.88	**65.40 ± 7.88**
hopper-expert	109.46 ± 1.80	111.12 ± 0.20	**111.46 ± 0.51**
hopper-medium-expert	56.20 ± 12.79	61.70 ± 7.91	**86.10 ± 16.55**
hopper-medium-replay	19.61 ± 2.63	**69.18 ± 24.71**	43.93 ± 25.58
walker2d-medium	71.72 ± 4.46	0.30 ± 0.53	**73.73 ± 3.61**
walker2d-expert	**108.51 ± 0.43**	107.62 ± 0.34	**108.51 ± 0.06**
walker2d-medium-expert	88.38 ± 14.41	crashed	**95.55 ± 13.15**
walker2d-medium-replay	**33.20 ± 12.52**	4.46 ± 4.97	33.11 ± 14.71

For these experiments, we used the same hyperparameters as the offline learning experiments, but omitted the Q function from the policy loss. For SMODICE, we used the author's original implementation with its original hyperparameters.