Offline RL Without Off-Policy Evaluation

David Brandfonbrener **William F. Whitney** **Rajesh Ranganath** **Joan Bruna**
Department of Computer Science, Center for Data Science
New York University
david.brandfonbrener@nyu.edu

Abstract

Most prior approaches to offline reinforcement learning (RL) have taken an iterative actor-critic approach involving off-policy evaluation. In this paper we show that simply doing one step of constrained/regularized policy improvement using an on-policy Q estimate of the behavior policy performs surprisingly well. This one-step algorithm beats the previously reported results of iterative algorithms on a large portion of the D4RL benchmark. The simple one-step baseline achieves this strong performance without many of the tricks used by previously proposed iterative algorithms and is more robust to hyperparameters. We argue that the relatively poor performance of iterative approaches is a result of the high variance inherent in doing off-policy evaluation and magnified by the repeated optimization of policies against those high-variance estimates. In addition, we hypothesize that the strong performance of the one-step algorithm is due to a combination of favorable structure in the environment and behavior policy.

1 Introduction

An important step towards effective real-world RL is to improve sample efficiency. One avenue towards this goal is offline RL (also known as batch RL) where we attempt to learn a new policy from data collected by some other behavior policy without interacting with the environment. Recent work in offline RL is well summarized by Levine et al. [2020].

In this paper, we challenge the dominant paradigm in the deep offline RL literature that primarily relies on actor-critic style algorithms that alternate between policy evaluation and policy improvement [Fujimoto et al., 2018a, 2019, Peng et al., 2019, Kumar et al., 2019, 2020, Wang et al., 2020b, Wu et al., 2019, Kostrikov et al., 2021, Jaques et al., 2019, Siegel et al., 2020, Nachum et al., 2019]. All these algorithms rely heavily on off-policy evaluation to learn the critic. Instead, we find that a simple baseline which only performs one step of policy improvement using the behavior Q function often outperforms the more complicated iterative algorithms. Explicitly, we find that our one-step algorithm beats prior results of iterative algorithms on most of the gym-mujoco [Brockman et al., 2016] and Adroit [Rajeswaran et al., 2017] tasks in the the D4RL benchmark suite [Fu et al., 2020].

We then dive deeper to understand why such a simple baseline is effective. First, we examine what goes wrong for the iterative algorithms. When these algorithms struggle, it is often due to poor off-policy evaluation leading to inaccurate Q values. We attribute this to two causes: (1) distribution shift between the behavior policy and the policy to be evaluated, and (2) iterative error exploitation whereby policy optimization introduces bias and dynamic programming propagates this bias across the state space. We show that empirically both issues exist in the benchmark tasks and that one way to avoid these issues is to simply avoid off-policy evaluation entirely.

Finally, we recognize that while the the one-step algorithm is a strong baseline, it is not always the best choice. In the final section we provide some guidance about when iterative algorithms can perform better than the simple one-step baseline. Namely, when the dataset is large and behavior policy has good coverage of the state-action space, then off-policy evaluation can succeed and iterative

Preprint. Under review.

algorithms can be effective. In contrast, if the behavior policy is already fairly good, but as a result does not have full coverage, then one-step algorithms are often preferable.

Our main contributions are:

- A demonstration that a simple baseline of one step of policy improvement outperforms more complicated iterative algorithms on a broad set of offline RL problems.
- An examination of failure modes of off-policy evaluation in iterative offline RL algorithms.
- A description of when one-step algorithms are likely to outperform iterative approaches.

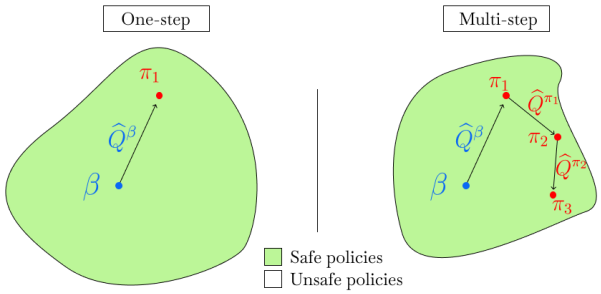

Figure 1: A cartoon illustration of the difference between one-step and multi-step methods. All algorithms constrain themselves to a neighborhood of "safe" policies around β. A one-step approach (left) only uses the on-policy $\widehat{Q}^\beta$, while a multi-step approach (right) repeatedly uses off-policy $\widehat{Q}^{\pi_i}$.

2 Setting and notation

We will consider an offline RL setup as follows. Let $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \rho, P, R, \gamma\}$ be a discounted infinite-horizon MDP. In this work we focus on applications in continuous control, so we will generally assume that both $\mathcal{S}$ and $\mathcal{A}$ are continuous and bounded. We consider the offline setting where rather than interacting with $\mathcal{M}$, we only have access to a dataset D_N of N tuples of (s_i, a_i, r_i) collected by some behavior policy β with initial state distribution ρ. Let $r(s,a) = \mathbb{E}_{r|s,a}[r]$ be the expected reward. Define the state-action value function for any policy π by $Q^\pi(s,a) := \mathbb{E}_{P,\pi|s_0=s,\ a_0=a}[\sum_{t=0}^\infty \gamma^t r(s_t, a_t)]$. The objective is to maximize the expected return J of the learned policy:

$$J(\pi) := \mathop{\mathbb{E}}_{\rho,P,\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right] = \mathop{\mathbb{E}}_{\substack{s\sim\rho \\ a\sim\pi|s}}[Q^\pi(s,a)]. \tag{1}$$

Following Fu et al. [2020] and others in this line of work, we allow access to the environment to tune a small (< 10) set of hyperparameters. See Paine et al. [2020] for a discussion of the active area of research on hyperparameter tuning for offline RL. We also discuss this further in Appendix C.

3 Related work

Most prior work on deep offline RL consists of iterative actor-critic algorithms. The primary innovation of each paper is to propose a different mechanism to ensure that he learned policy does not stray too far from the data generated by the behavior policy. Broadly, we group these methods into three camps: policy constraints/regularization, modified of imitation learning, and Q regularization:

1. The majority of prior work acts directly on the policy. Some authors have proposed explicit constraints on the learned policy to only select actions where (s,a) has sufficient support under the data generating distribution [Fujimoto et al., 2018a, 2019, Laroche et al., 2019]. Another proposal is to regularize the learned policy towards the behavior policy Wu et al. [2019] usually either with a KL divergence [Jaques et al., 2019] or MMD [Kumar et al., 2019]. This is a very straighforward way to stay close to the behavior with a hyperparameter that determines just how close. All of these algorithms are iterative and rely on off-policy evaluation.
2. Siegel et al. [2020], Wang et al. [2020b], Chen et al. [2020] all use algorithms that filter out datapoints with low Q values and then perform imitation learning. Wang et al. [2018], Peng et al. [2019] use a weighted imitation learning algorithm where the weights are determined by exponentiated Q values. These algorithms are iterative.
3. Another way to prevent the learned policy from choosing unknown actions is to incorporate some form of regularization to encourage staying near the behavior and being pessimistic about unknown state, action pairs [Wu et al., 2019, Nachum et al., 2019, Kumar et al., 2020, Kostrikov et al., 2021]. However, properly being able to quantify uncertainty about unknown states is notoriously difficult when dealing with neural network value functions [Buckman et al., 2020]. Again all of these algorithms are iterative.

Some recent work has also noted that optimizing policies based on the behavior value function can perform surprisingly well [Gulcehre et al., 2020, Goo and Niekum, 2020]. However, these papers propose complicated variants of the one-step approach involving ensembles, non-standard regularizers and paraterizations or ensembles and distributional Q functions. In contrast, we implement the simplest possible one-step algorithms without any modifications to the network architecture or standard regularizers/constraints. Moreover, we focus on providing an analysis of when and why this simple baseline works.

There are also important connections between the one-step algorithm and the literature on conservative policy improvement [Kakade and Langford, 2002, Schulman et al., 2015, Achiam et al., 2017], which we discuss in more detail in Appendix B.

4 Defining the algorithms

In this section we provide a unified algorithmic template for offline RL algorithms as offline approximate modified policy iteration. We show how this template captures our one-step algorithm as well as a multi-step policy iteration algorithm and an iterative actor-critic algorithm. Then any choice of policy evaluation and policy improvement operators defines one-step, multi-step, and iterative algorithms.

4.1 Algorithmic template

We consider a generic offline approximate modified policy iteration (OAMPI) scheme, shown in Algorithm 1. Essentially the algorithm alternates between two steps. First, there is a policy evaluation step where we estimate the Q function of the current policy π_{k-1} by $\widehat{Q}^{\pi_{k-1}}$ using only the dataset D_N. Implementations also often use the prior Q estimate $\widehat{Q}^{\pi_{k-2}}$ to warm-start the approximation process. Second, there is a policy improvement step. This step takes in the estimated Q function $\widehat{Q}^{\pi_{k-1}}$, the estimated behavior $\hat{\beta}$, and the dataset D_N and produces a new policy π_k. Again an algorithm may use π_{k-1} to warm-start the optimization. Moreover, we expect this improvement step to be regularized or constrained to ensure that π_k remains in the support of β and D_N. Choices for this regularization/constraint are discussed below. Now we discuss a few ways to instantiate the template.

Algorithm 1: OAMPI

input : K, dataset D_N, estimated behavior $\hat{\beta}$

Set $\pi_0 = \hat{\beta}$. Initialize $\widehat{Q}^{\pi_{-1}}$ randomly.

for *k = 1, . . . , K* **do**

  Policy evaluation: $\widehat{Q}^{\pi_{k-1}} = \mathcal{Q}(\pi_{k-1}, D_N, \widehat{Q}^{\pi_{k-2}})$

  Policy improvement: $\pi_k = \mathcal{I}(\widehat{Q}^{\pi_{k-1}}, \hat{\beta}, D_N, \pi_{k-1})$

end

One-step. The simplest algorithm sets the number of iterations $K = 1$. We train the policy evaluation to estimate Q^β, and then use one of the policy improvement operators discussed below to find the resulting π_1.

Multi-step. The multi-step algorithm now sets $K > 1$. The evaluation operator must evaluate off-policy since D_N is collected by β, but evaluation steps for $K \geq 2$ require evaluating policies $\pi_{k-1} \neq \beta$. Each iteration is trained to convergence in both the estimation and improvement steps.

Iterative actor-critic. An actor critic approach looks somewhat like multistep policy iteration, but does not attempt to train to convergence at each iteration. Instead, each iteration consists of one gradient step to update the Q estimate and one gradient step to improve the policy. Since all of the evaluation and improvement operators that we consider are gradient-based, this algorithm can adapt the same evaluation and improvement operators used by the multi-step algorithm. Most algorithms from the literature fall into this category [Fujimoto et al., 2018a, Kumar et al., 2019, 2020, Wu et al., 2019, Wang et al., 2020b, Siegel et al., 2020].

4.2 Policy evaluation operators

Following prior work on continuous state and action problems, we always evaluate by simple fitted Q evaluation [Fujimoto et al., 2018a, Kumar et al., 2019, Siegel et al., 2020, Wang et al., 2020b, Paine et al., 2020, Wang et al., 2021]. Explicitly the evaluation step for the one-step or multi-step

algorithms looks like

$$\mathcal{Q}(\pi_{k-1}, D_N, \widehat{Q}^{\pi_{k-2}}) = \arg\min_Q \sum_{i=1}^{N} (r(s_i, a_i) + \gamma \mathop{\mathbb{E}}_{a' \sim \pi_{k-1}|s'_i} Q(s'_i, a') - Q(s_i, a_i))^2, \quad (2)$$

where the right hand side may depend on $\widehat{Q}^{\pi_{k-2}}$ to warm-start optimization. In practice this is optimized by stochastic gradient descent with the use of a target network [Mnih et al., 2015]. For the iterative algorithm the $\arg\min$ is replaced by a single stochastic gradient step. We estimate the expectation over next state by a single sample from π_{k-1} (or from the dataset in the case when $\pi_{k-1} = \hat{\beta}$). See Voloshin et al. [2019], Wang et al. [2021] for more comprehensive examinations of this evaluation step.

4.3 Policy improvement operators

To instantiate the template, we also need to choose a specific policy improvement operator $\mathcal{I}$. We consider the following improvement operators selected from those discussed in the related work section. Each operator has a hyperparameter controlling deviation from the behavior policy.

Behavior cloning. The simplest baseline worth including is to just return $\hat{\beta}$ as the new policy π. Any policy improvement operator ought to perform at least as well as this baseline.

Constrained policy updates. Algorithms like BCQ [Fujimoto et al., 2018a] and SPIBB [Laroche et al., 2019] constrain the policy updates to be within the support of the data/behavior. In favor of simplicity, we implement a simplified version of the BCQ algorithm that removes the policy correction network which we call Easy BCQ. We define a new policy $\hat{\pi}_k^M$ by drawing M samples from $\hat{\beta}$ and then executing the one with the highest value according to $\widehat{Q}^{\beta}$. Explicitly:

$$\hat{\pi}_k^M(a|s) = \mathbb{1}[a = \arg\max_{a_j} \{\widehat{Q}^{\pi_{k-1}}(s, a_j) : a_j \sim \pi_{k-1}(\cdot|s),\ 1 \leq j \leq M\}]. \quad (3)$$

Regularized policy updates. Another common idea proposed in the literature is to regularize towards the behavior policy [Wu et al., 2019, Jaques et al., 2019, Kumar et al., 2019, Ma et al., 2019]. For a general divergence D we can define an algorithm that maximizes a regularized objective:

$$\hat{\pi}_k^{\alpha} = \arg\max_{\pi} \sum_i \mathop{\mathbb{E}}_{a \sim \pi|s} [\widehat{Q}^{\pi_{k-1}}(s_i, a)] - \alpha D(\hat{\beta}(\cdot|s_i), \pi(\cdot|s_i)) \quad (4)$$

A comprehensive review of different variants of this method can be found in Wu et al. [2019] which does not find dramatic differences across regularization techniques. In practice, we will use reverse KL divergence, i.e. $KL(\pi(\cdot|s_i) \| \hat{\beta}(\cdot|s_i))$. To compute the reverse KL, we draw samples from $\pi(\cdot|s_i)$ and use the density estimate $\hat{\beta}$ to compute the divergence. Intuitively, this regularization forces π to remain within the support of β rather than incentivizing π to cover beta.

Variants of imitation learning. Another idea, proposed by [Wang et al., 2018, Siegel et al., 2020, Wang et al., 2020b, Chen et al., 2020] is to modify an imitation learning algorithm either by filtering or weighting the observed actions so as to get a policy improvement. The weighted version that we implement uses exponentiated advantage estimates to weight the observed actions:

$$\hat{\pi}_k^{\tau} = \arg\max_{\pi} \sum_i \exp(\tau(\widehat{Q}^{\pi_{k-1}}(s_i, a_i) - \widehat{V}(s_i))) \log \pi(a_i|s_i). \quad (5)$$

5 Benchmark Results

Our main empirical finding is that one step of policy improvement is sufficient to beat state of the art results on much of the D4RL benchmark suite Fu et al. [2020]. This is striking since prior work focuses on iteratively estimating the Q function of the current policy iterate, but we only use one-step derived from $\widehat{Q}^{\beta}$. Results are shown in Table 1. Full experimental details are in Appendix C.

As we can see in the table, all of the one-step algorithms usually outperform the best iterative algorithms tested by Fu et al. [2020]. The one notable exception is the case of random data (especially

Table 1: Results of one-step algorithms on the D4RL benchmark. The first column gives the best results across several iterative algorithms considered in Fu et al. [2020]. We run 3 seeds and each algorithm is tuned over 6 values of their respective hyperparameter. We report the mean and standard deviation over seeds on 100 evaluation episodes per seed. We **bold** the best result on each dataset and blue any result where a one-step algorithm beat the best reported iterative result from Fu et al. [2020]. We use m for medium, m-e for medium-expert, m-re for medium-replay, r for random, and c for cloned.

	Iterative	One-step			
	Fu et al. [2020]	BC	Easy BCQ	Rev. KL Reg	Exp. Weight
halfcheetah-m	46.3	41.9 ± 0.1	52.6 ± 0.2	**55.2 ± 0.4**	48.4 ± 0.1
walker2d-m	81.1	68.6 ± 6.3	**87.2 ± 1.3**	85.9 ± 1.4	81.8 ± 2.2
hopper-m	58.8	49.9 ± 3.1	74.5 ± 6.2	**83.7 ± 4.5**	59.6 ± 2.5
halfcheetah-m-e	64.7	61.1 ± 2.7	78.2 ± 1.6	**93.8 ± 0.5**	93.4 ± 1.6
walker2d-m-e	111.0	78.5 ± 22.4	112.2 ± 0.3	111.2 ± 0.2	**113.0 ± 0.4**
hopper-m-e	**111.9**	49.1 ± 4.3	85.1 ± 2.2	98.7 ± 7.5	103.3 ± 9.1
halfcheetah-m-re	**47.7**	34.6 ± 0.9	38.3 ± 0.3	41.9 ± 0.5	38.1 ± 1.3
walker2d-m-re	26.7	26.6 ± 3.4	69.1 ± 4.2	**74.9 ± 6.6**	49.5 ± 12.0
hopper-m-re	48.6	23.1 ± 2.7	78.4 ± 7.2	92.3 ± 1.1	**97.5 ± 0.7**
halfcheetah-r	**35.4**	2.2 ± 0.0	5.4 ± 0.3	8.8 ± 3.8	3.2 ± 0.1
walker2d-r	**7.3**	0.9 ± 0.1	3.7 ± 0.1	6.2 ± 0.7	5.6 ± 0.8
hopper-r	**12.2**	2.0 ± 0.1	6.6 ± 0.1	7.9 ± 0.7	7.5 ± 0.4
pen-c	56.9	46.9 ± 11.0	**65.9 ± 3.6**	57.4 ± 3.5	60.0 ± 4.1
hammer-c	2.1	0.4 ± 0.1	**2.9 ± 0.5**	0.2 ± 0.1	2.1 ± 0.7
relocate-c	-0.1	-0.1 ± 0.0	**0.3 ± 0.2**	0.2 ± 0.1	0.2 ± 0.1
door-c	0.4	0.0 ± 0.1	**0.6 ± 0.6**	0.2 ± 0.7	0.2 ± 0.3

on halfcheetah), where iterative algorithms have a clear advantage. We will discuss potential causes of this further in Section 7.

To give a more direct comparison that controls for any potential implementation details, we use our implementation of reverse KL regularization to create multi-step and iterative algorithms. We are not using algorithmic modifications like Q ensembles, regularized Q values, or early stopping that have been used in prior work. But, our iterative algorithm recovers similar performance to prior regularized actor-critic approaches. These results are shown in Table 2.

Put together, these results immediately suggest some guidance to the practitioner: it is worthwhile to run the one-step algorithm as a baseline before trying something more elaborate. The one-step algorithm is substantially simpler than prior work, but usually achieves better performance.

Table 2: Results of reverse KL regularization on the D4RL benchmark across one-step, multi-step, and iterative algorithms. Again we run 3 seeds and 6 hyperparameters and report the mean and standard deviation across seeds using 100 evaluation episodes.

	One-step	Multi-step	Iterative
halfcheetah-m	55.2 ± 0.4	**59.3 ± 0.7**	51.2 ± 0.2
walker2d-m	**85.9 ± 1.4**	74.5 ± 2.8	74.8 ± 0.7
hopper-m	**83.7 ± 4.5**	54.8 ± 4.3	54.7 ± 1.9
halfcheetah-m-e	93.8 ± 0.5	**94.2 ± 0.5**	93.7 ± 0.6
walker2d-m-e	**111.2 ± 0.2**	109.8 ± 0.3	108.7 ± 0.6
hopper-m-e	**98.7 ± 7.5**	90.6 ± 18.8	94.5 ± 11.9
halfcheetah-r	8.8 ± 3.8	18.3 ± 6.5	**21.2 ± 5.2**
walker2d-r	**6.2 ± 0.7**	5.4 ± 0.2	5.4 ± 0.4
hopper-r	7.9 ± 0.7	**21.9 ± 8.9**	9.7 ± 0.4

6 What goes wrong for iterative algorithms?

The benchmark experiments show that one step of policy improvement often beats iterative and multi-step algorithms. In this section we dive deeper to understand why this happens. First, by examining the learning curves of each of the algorithms we note that iterative algorithms require stronger regularization to avoid instability. Then we identify two causes of this instability: *distribution shift* and *iterative error exploitation*.

Distribution shift causes evaluation error by reducing the effective sample size in the fixed dataset for evaluating the current policy and has been extensively considered in prior work as discussed below.

Iterative error exploitation occurs when we repeatedly optimize policies against our Q estimates and exploit their errors. This introduces a bias towards overestimation at each step (much like the training error in supervised learning is biased to be lower than the test error). Moreover, by iteratively re-using the data and using prior Q estimates to warmstart training at each step, the errors from one step are amplified at the next. This type of error is particular to multi-step and iterative algorithms.

6.1 Learning curves and hyperparameter sensitivity

To begin to understand why iterative and multi-step algorithms can fail it is instructive to look at the learning curves. As shown in Figure 2, we often observe that the iterative algorithm will begin to learn and then crash. Regularization can help to prevent this crash since strong enough regularization towards the behavior policy ensures that the evaluation is nearly on-policy.

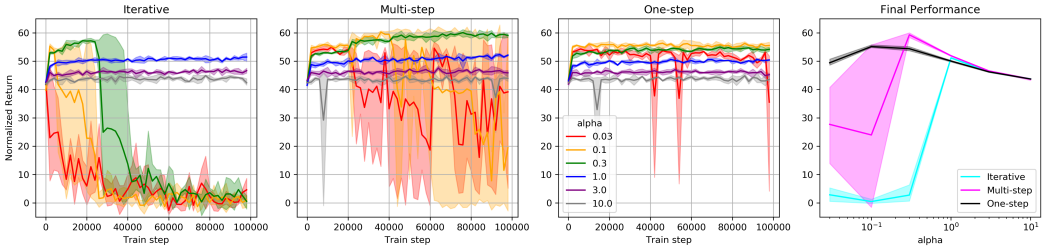

Figure 2: Learning curves and final performance on halfcheetah-medium across different algorithms and regularization hyperparameters. Error bars show min and max over 3 seeds. Similar figures for other datasets from D4RL can be found in Appendix D.

In contrast, the one-step algorithm is more robust to the regularization hyperparameter. The rightmost panel of the figure shows this clearly. While iterative and multi-step algorithms can have their performance degrade very rapidly with the wrong setting of the hyperparameter, the one-step approach is more stable. Moreover, we usually find that the optimal setting of the regularization hyperparameter is lower for the one-step algorithm than the iterative or multi-step approaches.

6.2 Distribution shift

Any algorithm that relies on off-policy evaluation will struggle with distribution shift in the evaluation step. Trying to evaluate a policy that is substantially different from the behavior reduces the effective sample size and increases the variance of the estimates. Explicitly, by distribution shift we mean the shift between the behavior distribution (the distribution over state-action pairs in the dataset) and the evaluation distribution (the distribution that would be induced by the policy π we want to evaluate).

Prior work. There is a substantial body of prior theoretical work that suggests that off-policy evaluation can be difficult and this difficulty scales with some measure of distribution shift. Wang et al. [2020a], Amortila et al. [2020], Zanette [2021] give exponential (in horizon) lower bounds on sample complexity in the linear setting even with good feature representations that can represent the desired Q function and assuming good data coverage. Upper bounds generally require very strong assumptions on both the representation and limits on the distribution shift [Wang et al., 2021, Duan et al., 2020, Chen and Jiang, 2019]. Moreover, the assumed bounds on distribution shift can be exponential in horizon in the worst case. On the empirical side, Wang et al. [2021] demonstrates issues with distribution shift when learning from pre-trained features and provides a nice discussion of why distribution shift causes error amplification. Fujimoto et al. [2018a] raises a similar issue under the name "extrapolation error". Regularization and constraints are meant to reduce issues stemming from distribution shift, but also reduce the potential for improvement over the behavior.

Empirical evidence. Both the multi-step and iterative algorithms in our experiments rely on off-policy evaluation as a key subroutine. We examine how easy it is to evaluate the policies encountered along the learning trajectory. To control for issues of iterative error exploitation (discussed in the next subsection), we train Q estimators from scratch on a heldout evaluation dataset sampled from the behavior policy. We then evaluate these trained Q function on rollouts from 1000 datapoints sampled from the replay buffer. Results are shown in Figure 3.

The results show a correlation betweed KL and MSE. Moreover, we see that the MSE generally increases over training. One way to mitigate this, as seen in the figure, is to use a large value of α. We just cannot take a very large step before running into problems with distribution shift. But, when we take such a small step, the information from the on-policy $\widehat{Q}^{\beta}$ is about as useful as the newly estimated $\widehat{Q}^{\pi}$. This is seen, for example, in Figure 2 where we get very similar performance across algorithms at high levels of regularization.

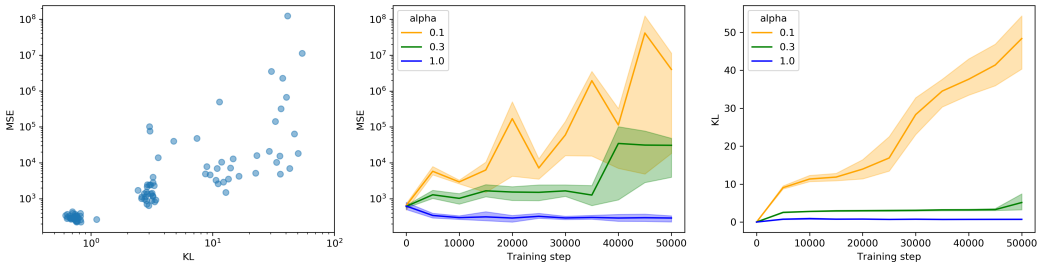

Figure 3: Results of running the iterative algorithm on halfcheetah-medium. Each checkpointed policy is evaluated by a Q function trained from scratch on heldout data. MSE refers to $\mathbb{E}_{s,a\sim\beta}[\hat{Q}^{\pi_i}(s,a) - Q^{\pi_i}(s,a)]$ and KL refers to $\mathbb{E}_{s\sim\beta}[KL(\pi(\cdot|s)\|\beta(\cdot|s)]$. Left: 90 policies taken from various points in training with various hyperaparmeters and random seeds. Center: MSE learning curves. Right: KL learning curves. Error bars show min and max over 3 random seeds.

6.3 Iterative error exploitation

The previous subsection identifies how any algorithm that uses off-policy evaluation is fundamentally limited by distribution shift, even if we were given fresh data and trained Q functions from scratch at every iteration. But, in practice, iterative algorithms repeatedly iterate between optimizing policies against estimated Q functions and re-estimating the Q functions using the *same data* and using the Q function from the previous step to warm-start the re-estimation. This induces dependence between steps that causes a problem that we call iterative error exploitation.

Intuition about the problem. In short, iterative error exploitation happens because π_i tends to choose overestimated actions in the policy improvement step, and then this overestimation propagates via dynamic programming in the policy evaluation step. To illustrate this issue more formally, consider the following: at each s, a we suffer some Bellman error $\varepsilon^{\pi}_{\beta}(s,a)$ based on our fixed dataset collected by β. Formally,

$$\widehat{Q}^{\pi}(s,a) = r(s,a) + \gamma \mathop{\mathbb{E}}_{\substack{s'|s,a \\ a'\sim\pi|s'}} [\widehat{Q}^{\pi}(s',a')] + \varepsilon^{\pi}_{\beta}(s,a). \tag{6}$$

Intuitively, $\varepsilon^{\pi}_{\beta}$ will be larger at state-actions with less coverage in the dataset collected by β. Note that $\varepsilon^{\pi}_{\beta}$ can absorb all noise due to our finite dataset as well as function approximation error.

All that is needed to cause iterative error exploitation is that the ϵ^{π}_{β} are highly correlated across different π, but for simplicity, we will assume that $\varepsilon^{\pi}_{\beta}$ is *the same* for all policies π estimated from our fixed offline dataset and instead write ε_{β}. Now that the errors do not depend on the policy we can treat the errors as auxiliary rewards that obscure the true rewards and see that

$$\widehat{Q}^{\pi}(s,a) = Q^{\pi}(s,a) + \widetilde{Q}^{\pi}_{\beta}(s,a), \qquad \widetilde{Q}^{\pi}_{\beta}(s,a) := \mathop{\mathbb{E}}_{\pi|s_0,a_0=s,a} \left[\sum_{t=0}^{\infty} \gamma^t \varepsilon_{\beta}(s_t,a_t) \right]. \tag{7}$$

This assumption is somewhat reasonable since we expect the error to primarily depend on the data. And, when the prior Q function is used to warm-start the current one (as is generally the case in practice), the approximation errors are automatically passed between steps.

Now we can explain the problem. Recall that under our assumption the ε_{β} are fixed once we have a dataset and likely to have larger magnitude the further we go from the support of the dataset. So, with each step π_i is able to better maximize ε_{β}, thus moving further from β and increasing the magnitude

of $\widetilde{Q}_\beta^{\pi_i}$ relative to Q^{π_i}. Even though Q^{π_i} may provide better signal than Q^β, it can easily be drowned out by $\widetilde{Q}_\beta^{\pi_i}$. In contrast, $\widetilde{Q}_\beta^{\beta}$ has small magnitude, so the one-step algorithm is robust to errors[1].

An example. Now we consider a simple gridworld example to illustrate iterative error exploitation. This example fits exactly into the setup outlined above since all errors are due to reward estimation so the ε_β is indeed constant over all π. The gridworld we consider has one deterministic good state with reward 1 and many stochastic bad states that have rewards distributed as $\mathcal{N}(-0.5, 1)$. We collect a dataset of 100 trajectories, each of length 100. One run of the multi-step offline regularized policy iteration algorithm is illustrated in Figure 4.

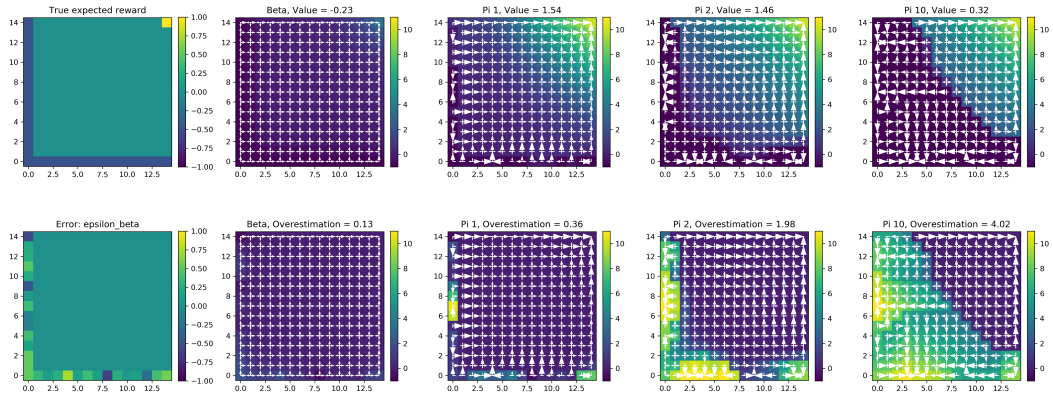

Figure 4: An illustration of multi-step offline regularized policy iteration. The leftmost panel in each row shows the true reward (top) or error ε_β (bottom). Then each subsequent panel plots π_i (with arrow size proportional to $\pi_i(a|s)$) over either Q^{π_i} (top) or $\widetilde{Q}_\beta^{\pi}$ (bottom), averaged over actions at each state. The one-step policy (π_1) has the highest value. The behavior policy here is a mixture of optimal π^* and uniform u with coefficient 0.2 so that $\beta = 0.2 \cdot \pi^* + 0.8 \cdot u$. We set $\alpha = 0.1$ as the regularization parameter for reverse KL regularization.

In the example, like in the D4RL benchmark, we see that one step outperforms multiple steps of improvement. Intuitively, when there are so many noisy states, it is likely that a few of them will be overestimated. Since the data is re-used for each step, these overestimations persist and propagate across the state space due to iterative error exploitation. This property of having many bad, but poorly estimated states likely also exists in the high-dimensional control problems encountered in the benchmark where there are many ways for the robots to fall down that are not observed in the data for non-random behavior. Moreover, both settings have larger errors in areas where we have less data. So even though the errors in the gridworld are caused by noise in the rewards, while errors in D4RL are caused by function approximation, we think this is a useful mental model of the problem.

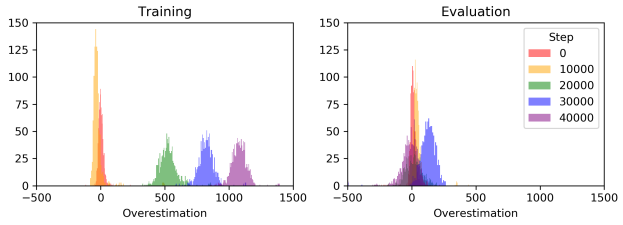

Figure 5: Histograms of overestimation error ($\widehat{Q}^{\pi_i}(s,a) - Q^{\pi_i}(s,a)$) on halfcheetah-medium with the iterative algorithm. Left: errors from the training Q function. Right: errors from an independently trained Q function.

Empirical evidence. In practice we cannot easily visualize the progression of errors. However, the dependence between steps still arises

[1]We should note that iterative error exploitation is similar to the overestimation addressed by double Q learning [Van Hasselt et al., 2016, Fujimoto et al., 2018b], but distinct. Since we are in the offline setting, the errors due to our finite dataset can be iteratively exploited more and more, while in the online setting considered by double Q learning, fresh data prevents this issue. We are also considering an algorithm based on policy iteration rather than value iteration.

as overestimation of the Q values. We can track the overestimation of the Q values over training as a way to measure how much bias is being induced by optimizing against our dependent Q estimators. As a control we can also train Q estimators from scratch on independently sampled evaluation data. These independently trained Q functions do not have the same overestimation bias even though the squared error does tend to increase as the policy moves further from the behavior (as seen in Figure 3). Explicitly, we track 1000 state, action pairs from the replay buffer over training. For each checkpointed policy we perform 3 rollouts at each state to get an estimate of the true Q value and compare this to the estimated Q value. Results are shown in Figure 5.

7 When are multiple steps useful?

So far we have focused on why the one-step algorithm often works better than the multi-step and iterative algorithms. However, we do not want to give the impression that one-step is always better. Indeed, our own experiments in Section 5 show a clear advantage for the multi-step and iterative approaches when we have randomly collected data. While we cannot offer a precise delineation of when one-step will outperform multi-step, in this section we offer some intuition as to when we can expect to see benefits from multiple steps of policy improvement.

As seen in Section 6, multi-step and iterative algorithms have problems when they propagate estimation errors. This is especially problematic in noisy and/or high dimensional environments. While the multi-step algorithms propagate this noise more widely than the one-step algorithm, they also propagate the signal. So, when we have sufficient coverage to reduce the magnitude of the noise, this increased propagation of signal can be beneficial. The D4RL experiments suggest that we are usually on the side of the tradeoff where the errors are large enough to make one-step preferable.

In Appendix A we illustrate a simple gridworld example where a slight modification of the behavior policy from Figure 4 makes multi-step dramatically outperform one-step. This modified behavior policy (1) has better coverage of the noisy states (which reduces error, helping multi-step), and (2) does a worse job propagating the reward from the good state (hurting one-step).

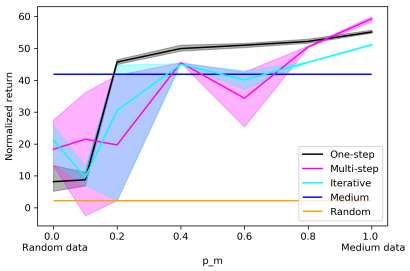

Figure 6: Performance of all three algorithms with reverse KL regularization across mixtures between halfcheetah-random and halfcheetah-medium. Error bars indicate min and max over 3 seeds.

We can also test empirically how the behavior policy effects the tradeoff between error and signal propagation. To do this we construct a simple experiment where we mix data from the random behavior policy with data from the medium behavior policy. Explicitly we construct a dataset D out of the datasets D_r for random and D_m for medium such that each trajectory in D comes from the medium dataset with probability p_m. So for $p_m = 0$ we have the random dataset and $p_m = 1$ we have the medium dataset, and in between we have various mixtures. Results are shown in Figure 6. It takes surprisingly little data from the medium policy for one-step to outperform the iterative algorithm.

8 Discussion, limitations, and future work

This paper presents the surprising effectiveness of a simple one-step baseline for offline RL. We examine the failure modes of iterative algorithms and the conditions where we might expect them to outperform the simple one-step baseline. This provides guidance to a practitioner that the simple one-step baseline is a good place to start when approaching an offline RL problem.

But, we leave many questions unanswered. One main limitation is that we lack a clear theoretical characterization of which environments and behaviors can guarantee that one-step outperforms multi-step or visa versa. Such results will likely require strong assumptions, but could provide useful insight. We don't expect this to be easy as it requires understanding policy iteration which has been notoriously difficult to analyze, often converging much faster than the theory would suggest [Sutton and Barto, 2018, Agarwal et al., 2019]. Another limitation is that while only using one step is perhaps the simplest way to avoid the problems of off-policy evaluation, there are possibly other more elaborate algorithmic solutions that we did not consider here. However, our strong empirical results suggest that the one-step algorithm is at least a strong baseline.

References

Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *International Conference on Machine Learning*, pages 22–31. PMLR, 2017.

Alekh Agarwal, Nan Jiang, and S. Kakade. Reinforcement learning: Theory and algorithms. 2019.

P. Amortila, Nan Jiang, and Tengyang Xie. A variant of the wang-foster-kakade lower bound for the discounted setting. *ArXiv*, abs/2011.01075, 2020.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016. URL `http://arxiv.org/abs/1606.01540`.

Jacob Buckman, Carles Gelada, and Marc G. Bellemare. The importance of pessimism in fixed-dataset policy optimization, 2020.

John Burkhardt. The truncated normal distribution, 2014.

Jinglin Chen and Nan Jiang. Information-theoretic considerations in batch reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*. PMLR, 2019.

Xinyue Chen, Zijian Zhou, Zheng Wang, Che Wang, Yanqiu Wu, and Keith Ross. Bail: Best-action imitation learning for batch deep reinforcement learning. *Advances in Neural Information Processing Systems*, 33, 2020.

Yaqi Duan, Zeyu Jia, and Mengdi Wang. Minimax-optimal off-policy evaluation with linear function approximation. In *International Conference on Machine Learning*, pages 2701–2709. PMLR, 2020.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. *arXiv preprint arXiv:1812.02900*, 2018a.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018b.

Scott Fujimoto, Edoardo Conti, Mohammad Ghavamzadeh, and Joelle Pineau. Benchmarking batch deep reinforcement learning algorithms. *arXiv preprint arXiv:1910.01708*, 2019.

Wonjoon Goo and Scott Niekum. You only evaluate once – a simple baseline algorithm for offline rl. In *Offline Reinforcement Learning Workshop at Neural Information Processing Systems*, 2020.

Caglar Gulcehre, Ziyu Wang, Alexander Novikov, Tom Le Paine, Sergio Gómez Colmenarejo, Konrad Zolna, Rishabh Agarwal, Josh Merel, Daniel Mankowitz, Cosmin Paduraru, et al. Rl unplugged: Benchmarks for offline reinforcement learning. *arXiv preprint arXiv:2006.13888*, 2020.

Natasha Jaques, Asma Ghandeharioun, Judy Hanwen Shen, Craig Ferguson, Agata Lapedriza, Noah Jones, Shixiang Gu, and Rosalind Picard. Way off-policy batch deep reinforcement learning of implicit human preferences in dialog, 2019.

Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *ICML*, volume 2, pages 267–274, 2002.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Ilya Kostrikov, Jonathan Tompson, Rob Fergus, and Ofir Nachum. Offline reinforcement learning with fisher divergence critic regularization. *arXiv preprint arXiv:2103.08050*, 2021.

Aviral Kumar, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. In *Advances in Neural Information Processing Systems*, pages 11761–11771, 2019.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *arXiv preprint arXiv:2006.04779*, 2020.

Romain Laroche, Paul Trichelair, and Remi Tachet Des Combes. Safe policy improvement with baseline bootstrapping. In *International Conference on Machine Learning*, pages 3652–3661. PMLR, 2019.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Yifei Ma, Yu-Xiang Wang, et al. Imitation-regularized offline learning. *arXiv preprint arXiv:1901.04723*, 2019.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Ofir Nachum, Bo Dai, Ilya Kostrikov, Yinlam Chow, Lihong Li, and Dale Schuurmans. Algaedice: Policy gradient from arbitrary experience. *arXiv preprint arXiv:1912.02074*, 2019.

Tom Le Paine, Cosmin Paduraru, Andrea Michi, Caglar Gulcehre, Konrad Zolna, Alexander Novikov, Ziyu Wang, and Nando de Freitas. Hyperparameter selection for offline reinforcement learning, 2020.

Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.

Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

Noah Siegel, Jost Tobias Springenberg, Felix Berkenkamp, Abbas Abdolmaleki, Michael Neunert, Thomas Lampe, Roland Hafner, Nicolas Heess, and Martin Riedmiller. Keep doing what worked: Behavior modelling priors for offline reinforcement learning. In *International Conference on Learning Representations*, 2020.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

Cameron Voloshin, Hoang M Le, Nan Jiang, and Yisong Yue. Empirical study of off-policy policy evaluation for reinforcement learning. *arXiv preprint arXiv:1911.06854*, 2019.

Qing Wang, Jiechao Xiong, Lei Han, Han Liu, Tong Zhang, et al. Exponentially weighted imitation learning for batched historical data. In *Advances in Neural Information Processing Systems*, pages 6288–6297, 2018.

Ruosong Wang, Dean P. Foster, and Sham M. Kakade. What are the statistical limits of offline rl with linear function approximation?, 2020a.

Ruosong Wang, Yifan Wu, Ruslan Salakhutdinov, and Sham M Kakade. Instabilities of offline rl with pre-trained neural representation. *arXiv preprint arXiv:2103.04947*, 2021.

Ziyu Wang, Alexander Novikov, Konrad Zolna, Josh S Merel, Jost Tobias Springenberg, Scott E Reed, Bobak Shahriari, Noah Siegel, Caglar Gulcehre, Nicolas Heess, et al. Critic regularized regression. *Advances in Neural Information Processing Systems*, 33, 2020b.

Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning, 2019.

Andrea Zanette. Exponential lower bounds for batch reinforcement learning: Batch rl can be exponentially harder than online rl, 2021.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 8 and Section 7.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See supplement.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Appendix C
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] In all relevant figures.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix C
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] Data from Fu et al. [2020].
   (b) Did you mention the license of the assets? [Yes] The license is Apache 2.0.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Code in supplement.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] Data is simulated.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] Data is simulated.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Gridworld example where multi-step outperforms one-step

As explained in the main text, this section presents an example that is only a slight modification of the one in Figure 4, but where a multi-step approach is clearly preferred over just one step. The data-generating and learning processes are exactly the same (100 trajectories of length 100, discount 0.9, $\alpha = 0.1$ for reverse KL regularization). The only difference is that rather than using a behavior that is a mixture of optimal and uniform, we use a behavior that is a mixture of maximally suboptimal and uniform. If we call the suboptimal policy π^- (which always goes down and left in our gridworld), then the behavior for the modified example is $\beta = 0.2 \cdot \pi^- + 0.8 \cdot u$, where u is uniform. Results are shown in Figure 7.

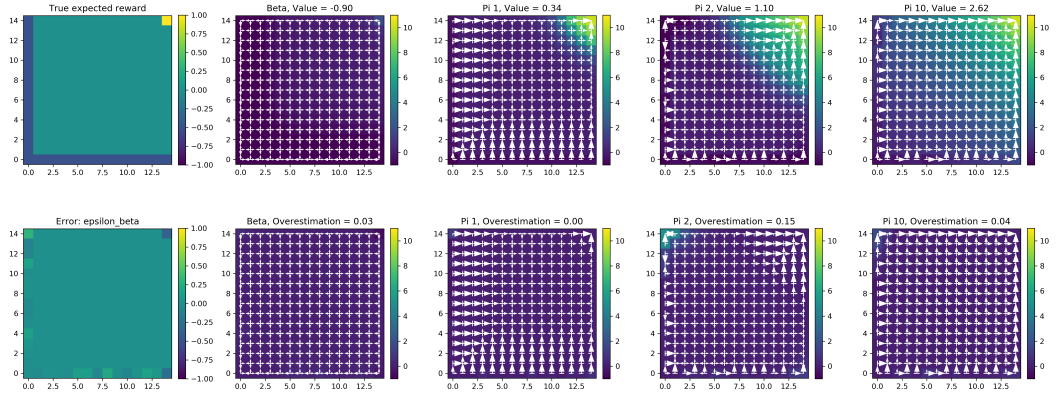

Figure 7: A gridworld example with modified behavior where multi-step is much better than one-step.

By being more likely to go to the noisy states, this behavior policy allows us to get lower variance estimates of the rewards. Essentially, the coverage of the behavior policy in this example reduces the magnitude of the evaluation errors. This allows for more aggressive planning using multi-step methods. Moreover, since the behavior is less likely to go to the good state, the behavior Q function does not propagate the signal from the rewarding state as far, harming the one-step method.

B Connection to policy improvement guarantees

The regularized or constrained one-step algorithm performs an update that directly inherits guarantees from the literature on conservative policy improvement [Kakade and Langford, 2002, Schulman et al., 2015, Achiam et al., 2017]. These original papers consider an online setting where more data is collected at each step, but the guarantee at each step applies to our one-step offline algorithm.

The key idea of this line of work begins with the performance difference lemma of Kakade and Langford [2002], and then lower bounds the amount of improvement over the behavior policy. Define the discounted state visitation distribution for a policy π by $d^\pi(s) := (1-\gamma)\sum_{t=0}^{\infty}\gamma^t \mathbb{P}_{\rho,P,\pi}(s_t = s)$. We will also use the shorthand $Q(s,\pi)$ to denote $\mathbb{E}_{a\sim\pi|s}[Q(s,a)]$. Then we have the performance difference lemma as follows.

Lemma 1 (Performance difference, Kakade and Langford [2002])**.** *For any two policies π and β,*

$$J(\pi) - J(\beta) = \frac{1}{1-\gamma}\mathop{\mathbb{E}}_{s\sim d^\pi}[Q^\beta(s,\pi) - Q^\beta(s,\beta)]. \tag{8}$$

Then, Corollary 1 from Achiam et al. [2017] (reproduced below) gives a guarantee for the one-step algorithm. The key idea is that when π is sufficiently close to β, we can use Q^β as an approximation to Q^π.

Lemma 2 (Conservative Policy Improvement, Achiam et al. [2017])**.** *For any two policies π and β, let $\|A^\beta_\pi\|_\infty = \sup_s |Q^\beta(s,\pi) - Q^\beta(s,\beta)|$. Then,*

$$J(\pi) - J(\beta) \geq \frac{1}{1-\gamma}\mathop{\mathbb{E}}_{s\sim d^\beta}\left[\left(Q^\beta(s,\pi) - Q^\beta(s,\beta)\right) - \frac{2\gamma\|A^\beta_\pi\|_\infty}{1-\gamma} D_{TV}(\pi(\cdot|s)\|\beta(\cdot|s))\right] \tag{9}$$

where D_{TV} denotes the total variation distance.

Replacing Q^β with $\widehat{Q}^\beta$ and the TV distance by the KL, we get precisely the objective that we optimize in the one-step algorithm. This shows that the one-step algorithm indeed optimizes a lower bound on the performance difference. Of course, in practice we replace the potentially large multiplier on the divergence term by a hyperparameter, but this theory at least motivates the soundness of the approach.

We are not familiar with similar guarantees for the iterative or multi-step approaches that rely on off-policy evaluation.

C Experimental setup

C.1 Benchmark experiments (Tables 1 and 2, Figure 2)

Data. We use the datasets from the D4RL benchmark [Fu et al., 2020]. We use the latest versions, which are v2 for the mujoco datasets and v1 for the adroit datasets.

Hyperparameter tuning. We follow the practice of Fu et al. [2020] and tune a small set of hyperparameters by interacting with the simulator to estimate the value of the policies learned under each hyperparameter setting. The hyperparameter sets for each algorithm can be seen in Table 3.

Table 3: Hyperparameter sweeps for each algorithm.

Algorithm	Hyperparameter set
Reverse KL (α)	{0.03, 0.1, 0.3, 1.0, 3.0, 10.0}
Easy BCQ (M)	{2, 5, 10, 20, 50, 100}
Exponentially weighted (τ)	{0.1, 0.3, 1.0, 3.0, 10.0, 30.0}

This may initially seem like "cheating", but can be a reasonable setup if we are considering applications like robotics where we can feasibly test a small number of trained policies on the real system. Also, since prior work has used this setup, it makes it easiest to compare our results if we use it too. While beyond the scope of this work, we do think that better offline model selection procedures will be crucial to make offline RL more broadly applicable. A good primer on this topic can be found in Paine et al. [2020].

Models. All of our Q functions and policies are simple MLPs with ReLU activations and 2 hidden layers of width 1024. Our policies output a truncated normal distribution with diagonal covariance where we can get reparameterized samples by sampling from a uniform distribution and computing the differentiable inverse CDF [Burkhardt, 2014]. We found this to be more stable than the tanh of normal used by e.g. Fu et al. [2020], but to achieve similar performance when both are stable. We use these same models across all experiments.

One-step training procedure. For all of our one-step algorithms, we train our $\hat{\beta}$ behavior estimate by imitation learning for 500k gradient steps using Adam [Kingma and Ba, 2014] with learning rate 1e-4 and batch size 512. We train our $\widehat{Q}^\beta$ estimator by fitted Q evaluation with a target network for 2 million gradient steps using Adam with learning rate 1e-4 and batch size 512. The target is updated softly at every step with parameter $\tau = 0.005$. All policies are trained for 100k steps again with Adam using learning rate 1e-4 and batch size 512.

Easy BCQ does not require training a policy network and just uses $\hat{\beta}$ and $\widehat{Q}^\beta$ to define it's policy. For the exponentially weighted algorithm, we clip the weights at 100 to prevent numerical instability. To estimate reverse KL at some state we use 10 samples from the current policy and the density defined by our estimated $\hat{\beta}$.

Each random seed retrains all three models (behavior, Q, policy) from different initializations. We use three random seeds.

Multi-step training procedure. For multi-step algorithms we use all the same hyperparameters as one-step. We initialize our policy and Q function from the same pre-trained $\hat{\beta}$ and $\widehat{Q}^\beta$ as we use for the one-step algorithm trained for 500k and 2 million steps respectively. Then we consider 5 policy steps. To ensure that we use the same number of gradient updates on the policy, each step consists of 20k gradient steps on the policy followed by 200k gradient steps on the Q function. Thus, we take the same 100k gradient steps on the policy network. Now the Q updates are off-policy so the next action a' is sampled from the current policy π_i rather than from the dataset.

Iterative training procedure. For iterative algorithms we again use all the same hyperparameters and initialize from the same $\hat{\beta}$ and $\widehat{Q}^{\beta}$. We again take the same 100k gradient steps on the policy network. For each step on the policy network we take 2 off-policy gradient steps on the Q network.

Evaluation procedure. To evaluate each policy we run 100 trajectories in the environment and compute the mean. We then report the mean and standard deviation over three training seeds.

C.2 MSE experiment (Figure 3)

Data. To get an independently sampled dataset of the same size as the training set, we use the behavior cloned policy $\hat{\beta}$ to sample 1000 trajectories. The checkpointed policies are taken at intervals of 5000 gradient steps from each of the three training seeds.

Training procedure. The $\widehat{Q}^{\pi_i}$ training procedure is the same as before so we use Adam with step size 1e-4 and batch size 512 and a target network with soft updates with parameter 0.005. We train for 1 million steps.

Evaluation procedure. To evaluate MSE, we sample 1000 state, action pairs from the original training set and from each state, action pair we run 3 rollouts. We take the mean over the rollouts and then compute squared error at each state, action pair and finally get MSE by taking the mean over state, action pairs. The reported reverse KL is evaluated by samples during training. At each state in a batch we take 10 samples to estimate the KL at that state and then take the mean over the batch.

C.3 Gridworld experiment (Figure 4)

Environment. The environment is a 15 x 15 gridworld with deterministic transitions. The rewards are deterministically 1 for all actions taken from the state in the top right corner and stochastic with distribution $\mathcal{N}(-0.5, 1)$ for all actions taken from states on the left or bottom walls. The initial state is uniformly random. The discount is 0.9.

Data. We collect data from a behavior policy that is a mixture of the uniform policy (with probability 0.8) and an optimal policy (with probability 0.2). We collect 100 trajectories of length 100.

Training procedure. We give the agent access to the deterministic transitions. The only thing for the agent to do is estimate the rewards from the data and then learn in the empirical MDP. We perform tabular Q evaluation by dynamic programming. We initialize with the empirical rewards and do 100 steps of dynamic programming with discount 0.9. Regularized policy updates are solved for exactly by setting $\pi_i(a|s) \propto \beta(a|s) \exp(\frac{1}{\alpha}\widehat{Q}^{\pi_{i-1}}(s,a))$.

C.4 Overestimation experiment (Figure 5)

This experiment uses the same setup as the MSE experiment. The main difference is we also consider the Q functions learned during training and demonstrate the overestimation relative to the Q functions trained on the evaluation dataset as in the MSE experiment.

C.5 Mixed data experiment (Figure 6)

We construct datasets with $p_m = \{0.0, 0.1, 0.2, 0.4, 0.6, 0.8, 1.0\}$ by mixing the random and medium datasets from D4RL and then run the same training procedure as we did for the benchmark experiments. Each dataset has the same size, but a different proportion of trajectories from the medium policy.

D Learning curves

In this section we reproduce the learning curves and hyperparameter plots across the one-step, multi-step, and iterative algorithms with reverse KL regularization, as in Figure 2.

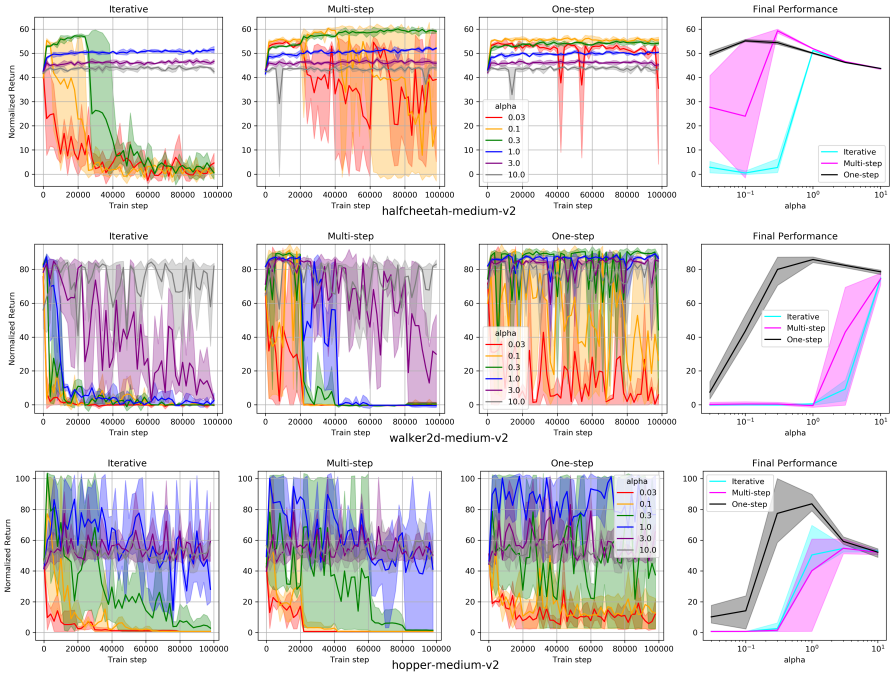

Figure 8: Learning curves on the medium datasets.

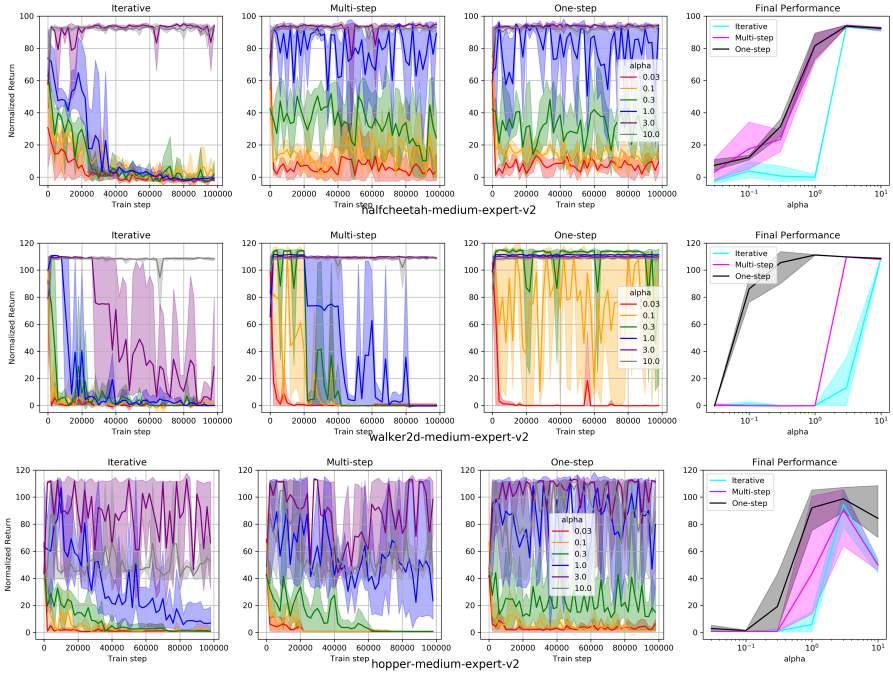

Figure 9: Learning curves on the medium-expert datasets.

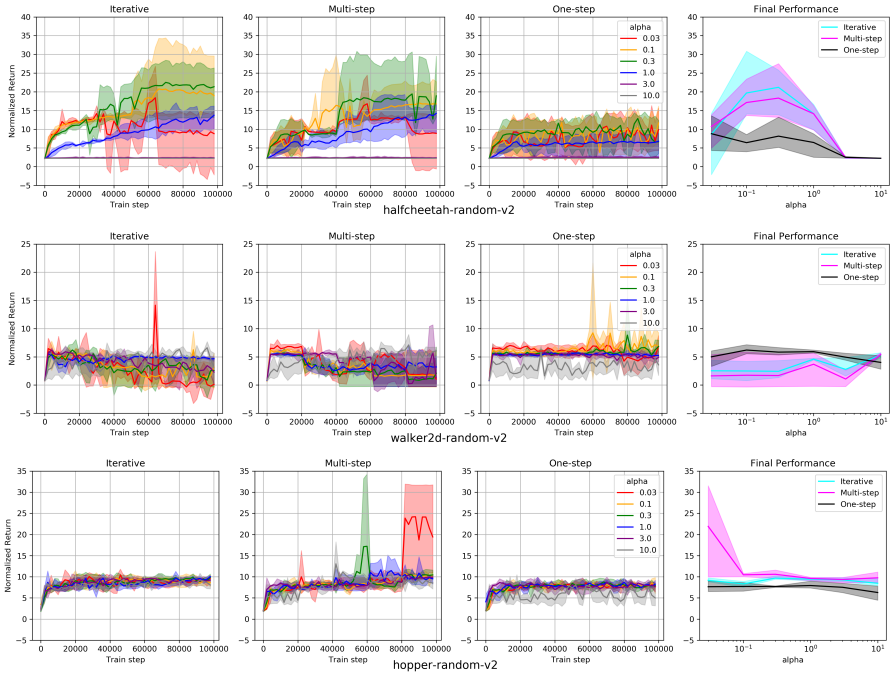

Figure 10: Learning curves on the random datasets.