Offline Learning for Scalable Decision Making

Justin Fu

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2021-168
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-168.html

July 29, 2021

Copyright © 2021, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Offline Learning for Scalable Decision Making

by

Justin Fu

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Sergey Levine, Chair
Professor Pieter Abbeel
Professor Roberto Horowitz

Summer 2021

The dissertation of Justin Fu, titled Offline Learning for Scalable Decision Making, is approved:

Chair _____ Date _____

_____ Date _____

_____ Date _____

University of California, Berkeley

Offline Learning for Scalable Decision Making

Copyright 2021
by
Justin Fu

Abstract

Offline Learning for Scalable Decision Making

by

Justin Fu

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Sergey Levine, Chair

The remarkable success of modern machine learning has arguably been due to the ability of algorithms to combine powerful models, such as neural networks, with large-scale datasets. This data-driven paradigm has been applied to a variety of applications from computer vision and speech processing, to machine translation and question answering. However, the majority of these successes have been in prediction problems, such as supervised learning. In contrast, many real-world applications of machine learning involve decision making problems, where one must leverage learned models to select optimal actions that maximize some objective of interest. Unfortunately, learned models can often fail in these situations, due to issues such as distribution shift and model exploitation. This thesis proposes methods and algorithms which are designed to handle these shortcomings in modern machine learning methods in order to produce reliable decision making agents. We begin in the area of reinforcement learning, where we study robust algorithms for offline reinforcement learning and model-based reinforcement learning. We discuss considerations for benchmarking offline reinforcement learning and off-policy evaluation, and propose a variety of domains and datasets designed to stress test state-of-the-art algorithms in the area. Finally, we study the more general problem of model-based optimization, and show how information-theoretic principles can guide us to construct uncertainty-aware models that mitigate exploitation.

Contents

Contents **i**

1 Introduction **1**

2 Preliminaries and Notation **4**
2.1 Reinforcement Learning 4
2.2 Model-based Optimization 7

3 Related Work **9**

I Robust Algorithms **12**

4 Bootstrapping Error in Model-Free Reinforcement Learning **13**
4.1 Relation to Prior Work 14
4.2 Out-of-Distribution Actions in Q-Learning 15
4.3 BEAR: Bootstrapping Error Accumulation Reduction 18
4.4 Experiments 21
4.5 Discussion 23

5 Model-Based Policy Optimization **25**
5.1 Relation to Prior Work 26
5.2 Monotonic Improvement with Model Bias 27
5.3 Model-Based Policy Optimization with Deep Reinforcement Learning 31
5.4 Experiments 32
5.5 Discussion 36

6 Offline Model-Based Optimization **38**
6.1 Relation to Prior Work 39
6.2 Normalized Maximum Likelihood 40
6.3 NEMO: Normalized Maximum Likelihood Estimation for Model-Based Optimization 41
6.4 Theoretical Results 44

6.5 Experiments . . . 46
6.6 Results and Discussion . . . 48
6.7 Discussion . . . 54

II Benchmarks and Evaluation 56

7 Benchmarking Offline Reinforcement Learning 57
7.1 Relation to Prior Work . . . 58
7.2 Task Design Factors . . . 58
7.3 Tasks and Datasets . . . 60
7.4 Evaluation . . . 64
7.5 Discussion . . . 65

8 Benchmarking Off-Policy Evaluation 69
8.1 Relation to Prior Work . . . 70
8.2 Policy Evaluation Metrics . . . 70
8.3 Task Design Factors . . . 72
8.4 Tasks . . . 72
8.5 Evaluation . . . 75
8.6 Discussion . . . 79

9 Conclusion 80

Bibliography 81

A Bootstrapping Error Accumulation Reduction 96
A.1 Distribution-Constrained Backup Operator . . . 96
A.2 Error Propagation Proofs . . . 97
A.3 Additional Details Regarding BEAR-QL . . . 99
A.4 Additional Experimental Details . . . 102
A.5 Additional Experimental Results . . . 103

B Model-Based Policy Optimization 109
B.1 Model-Based Policy Optimization with Performance Guarantees . . . 109
B.2 Useful Lemmas . . . 113
B.3 Hyperparameter Settings . . . 118

C Model-Based Optimization 119
C.1 Experimental Details . . . 119

D Benchmarking Offline Reinforcement Learning 121
D.1 Results by Domain . . . 121

D.2 Task Properties 122
D.3 Task and Datasets 124
D.4 Training and Evaluation Task Split 124
D.5 Experiment Details 125
D.6 Assessing the Feasibility of Hard Tasks 126
D.7 Maze Domain Trajectories 126

E Benchmarking Off-Policy Evaluation **128**
E.1 Off-policy Evaluation Metrics 128
E.2 Detailed Off-policy Evaluation Results 128

Acknowledgments

First, I would like to thank my thesis committee and qualifying exam committee members - Sergey Levine, Pieter Abbeel, Roberto Horowitz, and Trevor Darrell. This thesis would not have been possible without your guidance and advice. I would especially like to thank my graduate advisor Sergey Levine and my undergraduate advisor Pieter Abbeel. Pieter's enthusiastic teaching is what originally got me excited for artificial intelligence, and gave me the idea of joining his research lab in my junior year. It was so exhilarating to be able to work at the forefront of human knowledge, and get robots to do some really cool things. During my time in Pieter's lab, I was also fortunate enough to be mentored by Sergey, who inspired me to pursue a Ph.D and accepted me into his lab as a graduate student. Your dedication, work ethic, and commitment to excellence will inspire me for many years to come.

In addition to my advisors, I would not be here without the guidance of my mentors who taught me how to be a better researcher. My first mentor was Arjun Singh, who taught me the ropes on my first research project. I would especially like to thank Kelvin Guu for his mentorship during my year at Stanford, where he gave me extraordinary insight into improving my research process, and I still draw upon those experiences to guide me today. I was lucky to be able to find excellent mentors in both of my internships at Google Brain and DeepMind: Sergio Guadarrama, Anoop Korattikara, Yoram Bachrach, and Andrea Tacchetti. Not only did you give me excellent advice, but also encouraged me to be an independent thinker and provided me with the support to be successful.

Throughout my career in graduate school, I have been fortunate to work with many collaborators. I want to thank Chelsea Finn, Kevin Yu, J.D. Co-Reyes, Katie Luo, Avi Singh, Dibya Ghosh, Larry Yang, Aviral Kumar, Matthew Soh, George Tucker, Ofir Nachum, Michael Janner, Marvin Zhang, Anoop Korattikara, Sergio Guadarrama, Yoram Bachrach, Andrea Tacchetti, Julien Perolat, Ashwin Reddy, Abhishek Gupta, Coline Devin, Ben Eysenbach, Mohammad Norouzi, Thomas Paine, and Siddharth Verma. This thesis would not have been possible without you, and it has been an amazing journey working together towards cracking the problem of artificial intelligence.

My amazing roommates, Siddhartho Bhattacharya and Ryan Hang, have been some of my biggest supporters throughout my journey. Thank you for always being by my side, and may TP North live on even as this chapter ends and a new one begins. No acknowledgements would be complete without thanking the friends I have made a long the way in graduate school: J.D. Co-Reyes, Avi Singh, Vitchyr Pong, Marvin Zhang, Dierdre Quillen, Erin Grant, Kelvin Xu, Aviral Kumar, Michael Chang, and many others. Not only was it a blast having hot pots, making ramen, and other cooking parties, it was great to have conversations on intelligence, philosophy, and experience the ups and downs of graduate school together!

Finally and most importantly, nobody has provided me more encouragement than my family - Annette Wu, Zach Fu, and Isaac Fu. Even when the challenges felt insurmountable, you were always the ones who were willing to lend a hand and pick me up so I could keep pushing forward. It is only through your love and support that I made it this far!

Chapter 1

Introduction

Over the past decade, machine learning researchers have seen ever increasing success by combining more and more powerful models with increasingly large datasets. Beginning in 2012 with Alexnet [Krizhevsky et al., 2012] (trained on over 14 million images from the ImageNet dataset, with a network size of 62 million parameters), researchers have scaled methods such that today's largest models, such as GPT-3 [Brown et al., 2020], are trained on hundreds of billions of tokens with a network of 175 billion parameters. The impact of large models has been enormous and we have seen these models achieve superhuman performance on applications ranging from image speech recognition [Karita et al., 2019], recommender systems [Zhang et al., 2019], object detection [He et al., 2017, Redmon and Farhadi, 2017, Carion et al., 2020], medical imaging [Rajpurkar et al., 2017], machine translation [Sutskever et al., 2014, Vaswani et al., 2017, Raffel et al., 2020], and many more.

A common theme across many of these models is that they are primarily trained on prediction problems, using supervised or unsupervised learning. However, in many cases the end goal of training a model is not to use it for the prediction problem it was trained on, but to perform educated decision making on a downstream task. For example, in economics one may wish to construct models of labor or goods markets to select policies that try to elicit a certain outcome (such as reducing unemployment). Engineers may build physical models of vehicles or robots to design morphologies or controllers that achieve the highest performance or efficiency. And advertising agencies may wish to construct models of user behavior in order to select advertisements that are the most relevant to a user.

Scaling these decision making problems to large datasets can often be difficult. A straightforward formulation for these problems is to cast them under the bandit or reinforcement learning framework, and in these settings, data collection is ideally done in an online process, where one can explore, execute decisions, and learn from their consequences. However, online exploration and data collection can easily become impractical, such as in safety-critical applications (e.g. healthcare, autonomous driving) or in domains where evaluating a decision can be expensive or time-consuming. The alternative of optimizing decisions within a model learned from data is often not ideal either, since complex models are notorious for being susceptible to problems such as adversarial examples [Szegedy et al., 2014] and distribution

shift [Shankar et al., 2019]. While in some cases these may only be a nuisance or curiosity, if one is directly performing optimization on these models, one can easily produce decisions that are out-of-distribution and exploit the learned model.

Robust Algorithms for Offline Decision Making

The goal of this thesis is to present algorithms and ideas that allow decision-making agents to effectively leverage large, static datasets. The primary mechanism by which we construct reliable decision making agents is through "robustness" and mitigating model exploitation. Robustness in this context refers to constructing decision making agents that are reliable in the face of uncertainty in the environment or problem specifications (e.g. as in robust control [Zhou and Doyle, 1998]). Because the offline setting restricts us to viewing only a finite number of samples from problem at hand, (epistemic) uncertainty is an unavoidable fact that algorithms must cope with in order to succeed. For example, if one were to collect a dataset for autonomous driving entirely from local roads a low speed, it would be difficult to learn a controller for driving on the freeway or highway, and at the associated high speeds. And even if one were to scale up data collection to many possible situations, it would still be extremely difficult and expensive to capture every possible edge case an agent may experience during execution (known as the "long-tail" problem in AI). Thus, as with human decision making, it is crucial for algorithms and agents to be able to reason about where their models are accurate, and where they lack enough data to make an educated decision.

The work presented in this thesis primarily takes a "pessimistic" approach to handling uncertainty, which attempts to avoid situations under which an agent will experience rarely-seen situations that could produce erroneous results. We believe a risk-averse approach to offline learning is especially suited to many real-world problems where safety is a primary concern. In the context of reinforcement learning, Chapter 4 explores how by avoiding Q-function queries in low-support regions of data, we can avoid accumulation of errors during dynamic programming. Chapter 5 explores similar accumulation of errors in the model-based RL setting, where we address the issue by careful truncation of rollouts during planning. Finally, Chapter 6 explores explicit use of uncertainty-aware models for data-driven optimization problems when a dataset of function evaluations are provided in lieu of the ability to query the objective function itself.

Organization and Summary of Contributions

This thesis is divided into two main parts: algorithm design (Part I) and benchmarking recent progress in the field of offline decision making (Part II).

Part I motivates the development of robust algorithms, targeted primarily towards building decision making agents from static datasets. In Chapter 4, we explore the offline reinforcement learning setting. Traditional reinforcement learning requires agents to interact in

an online fashion, learning from mistakes and rewards as the agent collects more experience. Offline reinforcement learning presents an alternate paradigm under which learning is done from a fixed, static dataset. We analyze error propagation introduced by action selection in reinforcement learning algorithms, which leads us to develop the BEAR (**B**ootstrapping **E**rror **A**ccumulation **R**eduction) algorithm, which minimizes error propagation and model exploitation by selecting conservative actions to use in the Bellman operator.

In Chapter 5, we investigate robust algorithms in the context of the classic model-based reinforcement learning problem, where a learned dynamics model is used to generate simulated experience for a policy optimizer. We see in this setting that model-based reinforcement learning suffers from similar issues with model exploitation and distribution shift as offline reinforcement learning. To address these issues, we propose MBPO, a model-based policy optimization algorithm which controls the length of rollouts within the model. We provide theoretical analysis showing how the horizon length and regularization limit the worst-case policy performance, and empirically, we demonstrated improvements in sample complexity on simulated robotics problems. While the empirical evaluation in this work is in the more traditional online RL setting, the ideas are nevertheless relevant for constructing algorithms in the offline setting as well.

Chapter 6 explores the offline model-based optimization problem. In this setting, we are interested in a more traditional optimization setup where we wish to select an input x that maximizes some function $f(x)$, but unlike the traditional setting we are not allowed to query f directly and are instead given an offline dataset of input-output pairs. We propose an uncertainty-aware method for mitigating model exploitation in this setting based on the normalized maximum likelihood framework, and propose the NEMO (**N**ormalized Maximum Likelihood **E**stimation for **M**odel-based **O**ptimization) algorithm, which achieves state of the art results on a selection of design tasks for applications ranging from biology to robotics and materials science.

As part of an effort to accelerate progress in offline learning, in Part II, we discuss considerations for benchmarking offline reinforcement learning methods. Motivated by the difficulties of using ad-hoc evaluations, we introduce a set of standardized, open-source benchmarks designed to provide easy but meaningful evaluations of algorithm performance. We propose benchmarks in two problem settings: offline reinforcement learning (Chapter 7), where the goal is to optimize a policy for performance, and off-policy evaluation (Chapter 8), where the goal is either to select the best policy out of a set of candidates or evaluate the absolute performance of a given policy. We supplement each benchmark with a comprehensive evaluation of state-of-the-art algorithms in each field, and discuss both settings in which current methods work well along with areas which need improvement.

Chapter 2

Preliminaries and Notation

Offline decision making draws upon a wide array of concepts in machine learning and asks many fundamental questions about what it means to learn from prior experience. We primarily formalize decision making within two frameworks: reinforcement learning for problems with sequential structure, and model-based optimization for more general problems without obvious structure. In this chapter we review both of these frameworks, and introduce notation that we will be using throughout this thesis.

2.1 Reinforcement Learning

In this section, we cover background on reinforcement learning, offline reinforcement learning and general classes of solution methods.

Markov Decision Processes

Reinforcement learning is commonly formalized as learning in a Markov decision process (MDP). We define an MDP as a tuple, $(\mathcal{S}, \mathcal{A}, r, \gamma, P\rho_0)$. These symbols are defined as the following:

- $\mathcal{S}, \mathcal{A}$ denotes the state and action space, respectively. These can be either continuous or discrete, depending on the problem.
- $r(s, a)$ is the reward or utility function.
- $\gamma \in (0, 1)$ represents the discount factor. A smaller discount factor encourages more short-sighted behavior, whereas a longer discount factor allows an agent to reason about delayed rewards.
- $P(s'|s, a)$ represents a distribution over the dynamics or transition probabilities.
- $\rho_0(s)$, represents the initial state distribution.

An MDP is typically chosen to model sequential decision making problems, where an agent selects actions and generates a sequence of states and actions known as a trajectory, $\tau = (s_0, a_0, s_1, a_1, \dots)$. The agent is formalized as a policy $\pi(a|s)$ which maps a state to a distribution over actions (or, in the deterministic case, a single action).

Solving an MDP typically means to find a policy that maximizes the expected cumulative discounted rewards $J(\pi)$ (also known as the returns). This can be described by the following equation:

$$J(\pi) = E_{\pi,P,\rho_0}[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)].$$

When a model of a system is available (formally, the transition and reward functions are known), this problem is commonly referred to as **planning**. This can occur in many real-world scenarios when the dynamics of the system can be written down or simulated. In **reinforcement learning**, however, it is assumed that the transition and reward models are unknown, and must be learned through experience and data. This brings in an element of statistical learning, and with it interesting questions related to generalization and distribution shift that we will explore in this thesis.

Many approaches have been proposed to solve this objective in the reinforcement learning setting. One commonly used classification divides methods between *model-based* versus *model-free* algorithms. Model-based methods use pre-collected experience to estimate the transition and/or reward functions, and can then use a planning method inside the learned models to obtain a policy. We will explore model-based methods in Chapter 5. In contrast, model-free methods avoid explicitly modeling the dynamics or reward directly. For example, policy gradient methods form an estimate of the gradient of $J(\pi)$ with respect to the policy and optimize the returns directly [Peters and Schaal, 2006]. An alternative class of methods are those based on estimating **value functions**, or estimates of the future returns given a current state. We review these in the next section, and discuss offline reinforcement learning methods based on this framework in Chapter 7.

Value Functions

A key concept in reinforcement learning is the value function, or estimate of the future returns expected from a given state and/or action. The state value function is typically denoted by $V^\pi(s)$, and the state-action value is more commonly referred to as the Q-function, denoted by $Q^\pi(s, a)$. These values are defined by the following equations:

$$V^\pi(s) = \mathbb{E}_{\pi,P,\rho_0}[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)|s_0 = s].$$

$$Q^\pi(s, a) = \mathbb{E}_{\pi,P,\rho_0}[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)|s_0 = s, a_0 = a].$$

The notation Q^* and V^* are commonly used to denote the values of the optimal policy, π^*. The optimal Q-value Q^* is of special importance since we can obtain a deterministic optimal policy by selecting actions that maximize the Q-function: $\pi(s) = \underset{a}{\text{argmax}}\, Q^*(s, a)$.

All value functions satisfy a recurrence relation known as the Bellman consistency equation, written from the perspective of Q^π or V^π as:

$$Q^\pi(s, a) = r(s, a) + \mathbb{E}_{s' \sim P}[V^\pi(s')]$$
$$V^\pi(s) = \mathbb{E}_{s' \sim P, a \sim \pi}[r(s, a) + Q^\pi(s, a)]$$

Dynamic Programming

As noted in the prior section, once an optimal Q-function is obtained it is a simple process to construct an optimal policy, by selecting actions that maximize the Q-value. This leads to an indirect method for finding the optimal policy: we can first solve for the optimal Q-values, and from there the optimal policy.

The process of solving for the optimal values is typically done in an iterative, dynamic programming fashion by iterating the Bellman backup operator (denoted as $\mathcal{T}$) is defined as the following:

$$\mathcal{T}Q(s, a) = r(s, a) + \mathbb{E}_{s' \sim P}[V(s')]$$

where

$$V(s) = \max_a Q(s, a)$$

For shorthand, we denote application of the Bellman operator $\mathcal{T}$ on a value function V or Q as $\mathcal{T}V$ and $\mathcal{T}Q$, respectively. Q-iteration is a dynamic programming algorithm that iterates the Bellman backup: $Q^{t+1} \leftarrow \mathcal{T}Q^t$. Because the Bellman backup is a γ-contraction in the ℓ_∞ norm, and Q^* is its fixed point, Q-iteration can be shown to converge to Q^* [Sutton and Barto, 2018]. A similar result can be shown for value iteration, when iterating $V^{t+1} \leftarrow \mathcal{T}V^t$. The values of the optimal policy, Q^* and V^* satisfy Bellman consistency, in the sense that $Q^* = TQ^*$ and $V^* = TV^*$. Other values do not satisfy this with equality, and the difference, $|Q - TQ|$ or $|V - TV|$ is often known as the Bellman error or Bellman residual.

Additionally, we have the similar *policy evaluation* backup operator $\mathcal{T}^\pi$ operator given by:

$$\mathcal{T}^\pi Q(s, a) = r(s, a) + \mathbb{E}_{s'}[V(s')]$$

where

$$V(s) = \mathbb{E}_{a \sim \pi}[Q(s, a)]$$

This operation can similarily be written in vector notation as $\mathcal{T}^\pi Q$ or $\mathcal{T}^\pi V$. The values of the policy being evaluated, Q^π and V^π satisfy Bellman consistency for the evaluation operator, in the sense that $Q^\pi = \mathcal{T}^\pi Q^\pi$ and $V^\pi = \mathcal{T}^\pi V^\pi$.

When state spaces are large, function approximation is needed to represent the Q-values. This corresponds to *fitted Q-iteration* (FQI) [Ernst et al., 2005], a form of approximate dynamic programming (ADP), which forms the basis of modern deep RL methods such as DQN [Mnih et al., 2015]. FQI projects the values of the Bellman backup onto a family of Q-function approximators $\mathcal{Q}$: $Q^{t+1} \leftarrow \Pi_\mu(\mathcal{T}Q^t)$. Π_μ denotes a μ-weighted ℓ_2 projection, which minimizes the *Bellman error*:

$$\Pi_\mu(Q) \stackrel{\text{def}}{=} \operatorname*{argmin}_{Q' \in \mathcal{Q}} \mathbb{E}_{s,a\sim\mu}[(Q'(s,a) - Q(s,a))^2]. \tag{2.1}$$

The values produced by the Bellman backup, $(\mathcal{T}Q^t)(s,a)$ are commonly referred to as *target values*, and when neural networks are used for function approximation, the previous Q-function $Q^t(s,a)$ is referred to as the *target network*. Convergence guarantees for Q-iteration do not cleanly translate to FQI. Π_μ is an ℓ_2 projection, but $\mathcal{T}$ is a contraction in the ℓ_∞ norm – this norm mismatch means the composition of the backup and projection is no longer guaranteed to be a contraction under any norm [Bertsekas and Tsitsiklis, 1996], and hence convergence is not guaranteed.

In large action spaces (e.g., continuous), the maximization $\max_{a'} Q(s',a')$ is generally intractable. Actor-critic methods Sutton and Barto [2018], Fujimoto et al. [2018], Haarnoja et al. [2018] address this by additionally learning a policy π_θ that maximizes the Q-function.

Offline Reinforcement Learning

In episodic RL, the algorithm is given access to the MDP via trajectory samples for arbitrary π of the algorithm's choosing. Off-policy methods may use experience replay [Lin, 1992] to store these trajectories in a *replay buffer* $\mathcal{D}$ of transitions (s_t, a_t, s_{t+1}, r_t), and use an off-policy algorithm such as Q-learning [Watkins and Dayan, 1992] to optimize π. However, these methods still iteratively collect additional data, and omitting this collection step can produce poor results. For example, running state-of-the-art off-policy RL algorithms on trajectories collected from an expert policy can result in diverging Q-values [Kumar et al., 2019].

In *offline RL*, the algorithm no longer has access to the MDP, and is instead presented with a fixed *dataset* of transitions $\mathcal{D} = \{(s, a, s', r(s,a))\}$. The (unknown) policy that generated this data is referred to as a *behavior policy* π_B. Effective offline RL algorithms must handle distribution shift, as well as data collected via processes that may not be representable by the chosen policy class. Levine et al. [2020] provide a comprehensive discussion of the problems affecting offline RL.

2.2 Model-based Optimization

In reinforcement learning, we considered problems with sequential structure. However, many real-world problems do not necessarily conform to such structure. Instead, we can adopt the

more generic **model-based optimization** (MBO) framework which resembles a generic optimization problem. In MBO, we assume the existence of a stochastic ground truth function $f(y|\boldsymbol{x})$, and are given a dataset $\mathcal{D}$ of inputs $\boldsymbol{x}$ along with outputs y sampled from $f(y|\boldsymbol{x})$. Like in standard optimization problems, the goal of MBO is to find the input value that maximizes the true function:

$$\boldsymbol{x}^* = \text{argmax}_{\boldsymbol{x}} \mathbb{E}_{y \sim f(y|\boldsymbol{x})}[y]. \tag{2.2}$$

Much like how we considered learning purely from offline data in reinforcement learning, in **offline MBO**, the algorithm is not allowed to query the true function $f(y|\boldsymbol{x})$, and must find the best possible point $\boldsymbol{x}^*$ using only the guidance of a fixed dataset $\mathcal{D} = \{\boldsymbol{x}_{1:N}, y_{1:N}\}$. One approach to solving this problem is to introduce a separate proxy function $\hat{f}_\theta(y|\boldsymbol{x}) \approx f(y|\boldsymbol{x})$, which is learned from $\mathcal{D}$ as an estimate of the true function. From here, standard optimization algorithms such as gradient descent can be used to find the optimum of the proxy function, $\hat{\boldsymbol{x}}^* = \text{argmax}_{\boldsymbol{x}} \mathbb{E}_{y \sim \hat{f}_\theta(y|\boldsymbol{x})}[y]$. Alternatively, a trivial algorithm could be to select the highest-performing point in the dataset. While adversarial ground truth functions can easily be constructed where this is the best one can do (e.g., if $f(x) = -\infty$ on any $x \notin \mathcal{D}$), in many reasonable domains it should be possible to perform better than the best point in the dataset.

Chapter 3

Related Work

The work presented in this thesis is primarily concerned with offline decision making and associated challenges such as building algorithms that are robust to distribution shift. Offline learning using large, previously-collect datasets has been successfully applied to real-world systems such as in robotics [Cabi et al., 2019], recommender systems [Li et al., 2010, Strehl et al., 2010, Thomas et al., 2017], and dialogue systems [Henderson et al., 2008, Pietquin et al., 2011, Jaques et al., 2019]. Moreover, significant efforts have been made to incorporate large-scale datasets into off-policy RL [Kalashnikov et al., 2018, Mo et al., 2018, Gu et al., 2017], but these works use large numbers of robots to collect online interaction during training.

Chapter 4 considers offline learning in the case of model-free Q-learning algorithm, where we identify difficulties arising from error propagation and motivate the development of the BEAR (Bootstrapping Error Accumulation Reduction) algorithm. Errors arising from inadequate sampling, distributional shift, and function approximation have been rigorously studied as "error propagation" in approximate dynamic programming (ADP) [Bertsekas and Tsitsiklis, 1996, Munos, 2003, Farahmand et al., 2010, Scherrer et al., 2015]. These works study how Bellman errors accumulate and propagate to nearby states via bootstrapping. We build upon tools from this analysis to show that performing Bellman backups on static datasets leads to error accumulation due to out-of-distribution values. Our approach is motivated as reducing the rate of propagation of error propagation between states.

Algorithmically, most closely related to this work is batch-constrained Q-learning (BCQ) [Fujimoto et al., 2019b] and SPIBB [Laroche et al., 2019], which also discuss instability arising from previously unseen actions. Fujimoto et al. [2019b] show convergence properties of an action-constrained Bellman backup operator in tabular, error-free settings. We prove stronger results under approximation errors and provide a bound on the *suboptimality* of the solution. As a consequence, although we experimentally find that [Fujimoto et al., 2019b] outperforms standard Q-learning methods when the off-policy data is collected by an expert, BEAR outperforms Fujimoto et al. [2019b] when the off-policy data is collected by a suboptimal policy.

In Chapter 5 we explore the issues of distribution shift and robustness within model-

based reinforcement learning. Model-based reinforcement learning methods are promising candidates for real-world sequential decision-making problems due to their data efficiency [Kaelbling et al., 1996]. Gaussian processes and time-varying linear dynamical systems provide excellent performance in the low-data regime [Deisenroth and Rasmussen, 2011, Levine and Koltun, 2013, Kumar et al., 2016]. Neural network predictive models [Draeger et al., 1995, Gal et al., 2016, Depeweg et al., 2016, Nagabandi et al., 2018], are appealing because they allow for algorithms that combine the sample efficiency of a model-based approach with the asymptotic performance of high-capacity function approximators, even in domains with high-dimensional observations [Oh et al., 2015, Ebert et al., 2018, Kaiser et al., 2020]. Our work uses an ensemble of probabilistic networks, as in Chua et al. [2018], although our model is employed to learn a policy rather than in the context of a receding-horizon planning routine.

Theoretical analysis of error accumulation in model-based reinforcement learning algorithms has been considered by Sun et al. [2018] and Luo et al. [2019], who bound the discrepancy between returns under a model and those in the real environment of interest. Their approaches enforce a trust region around a reference policy, whereas we do not constrain the policy but instead consider rollout length based on estimated model generalization capacity. Alternate analyses have been carried out by incorporating the structure of the value function into the model learning [Farahmand et al., 2017] or by regularizing the model by controlling its Lipschitz constant [Asadi et al., 2018]. Prior work has also constructed complexity bounds for model-based approaches in the tabular setting [Szita and Szepesvari, 2010] and for the linear quadratic regulator [Dean et al., 2020].

Chapter 6 explores decision making within the offline model-based optimization (MBO) framework. MBO has been applied to problems such as designing DNA [Killoran et al., 2017], drugs [Popova et al., 2018], or materials [Hautier et al., 2010]. The estimation of distribution algorithm [Bengoetxea et al., 2001] alternates between searching in the input space and model space using a maximum likelihood objective. Kumar and Levine [2020] propose to learn an inverse mapping from output values to input values, and optimize over the output values which produce consistent input values. Brookes et al. [2019] propose CbAS, which uses a trust-region to limit exploitation of the model. Fannjiang and Listgarten [2020] casts the MBO problem as a minimax game based on the *oracle gap*, or the value between the ground truth function and the estimated function. There are also several related areas that could arguably be viewed as special cases of MBO. One is in contextual bandits under the batch learning from bandit feedback setting, where learning is often done on logged experience [Swaminathan and Joachims, 2015b, Joachims et al., 2018], or offline reinforcement learning [Levine et al., 2020], where model-based methods construct estimates of the MDP parameters [Kidambi et al., 2020, Yu et al., 2020].

In Chapter 7, we present D4RL (Datasets for Deep Data-Driven Reinforcement Learning), a benchmark for offline reinforcement learning and evaluate the performance of state-of-the-art algorithms. Most recent work in offline RL has primarily used datasets generated by a previously trained behavior policy, ranging from a random initial policy to a near-expert online-trained policy. This approach has been used for continuous control for

robotics [Fujimoto et al., 2019b, Kumar et al., 2019, Wu et al., 2019, Gulcehre et al., 2020], navigation [Laroche et al., 2019], industrial control [Hein et al., 2017], and Atari video games [Agarwal et al., 2020b]. To standardize the community around common datasets, several recent works have proposed benchmarks for offline RL algorithms. Agarwal et al. [2020b], Fujimoto et al. [2019a] propose benchmarking based on the discrete Atari domain. Concurrently to our work, Gulcehre et al. [2020] proposed a benchmark based on locomotion and manipulation tasks with perceptually challenging input and partial observability.

In Chapter 8, we consider benchmarking and evaluation in the related area of off-policy evaluation by introducing the DOPE (Benchmarks for Deep Off-Policy Evaluation) benchmark. Off-policy evaluation (OPE) has been studied extensively across a range of different domains, from healthcare [Thapa et al., 2005, Raghu et al., 2018, Nie et al., 2020], to recommender systems [Li et al., 2010, Dudík et al., 2014, Theocharous et al., 2015], and robotics [Kalashnikov et al., 2018]. Broadly speaking we can categories OPE methods into groups based the use of importance sampling [Precup et al., 2000], value functions [Sutton et al., 2009, Migliavacca et al., 2010, Sutton et al., 2016, Yang et al., 2020], and learned transition models [Paduraru, 2007], though a number of methods combine two or more of these components [Jiang and Li, 2016, Thomas and Brunskill, 2016, Munos et al., 2016]. A significant body of work in OPE is also concerned with providing statistical guarantees [Thomas et al., 2015]. Current benchmarks and evaluation of OPE methods is based around several metrics, including error in predicting the true return of the evaluated policy [Voloshin et al., 2019], correlation between the evaluation output and actual returns [Irpan et al., 2019], and ranking and model selection metrics [Doroudi et al., 2018]. As there is no single accepted metric used by the entire community, we provide a set of candidate metrics along with the benchmark.

Part I

Robust Algorithms

Chapter 4

Bootstrapping Error in Model-Free Reinforcement Learning

Model-free Q-learning algorithms have been one of the most successful classes of RL methods, and have seen recent results such as being able to solve Atari video games entirely from images [Mnih et al., 2015] or complex strategy games such as Go [Silver et al., 2016]. One appeal of Q-learning is that it can learn off-policy, meaning it can learn the optimal policy even while observing another policy act [Sutton and Barto, 2018]. This allows Q-learning to leverage tools such as experience replay [Lin, 1992], or learning from an agent's past experience, to greatly accelerate learning. In principle, this means that off-policy methods can leverage offline datasets and still extract performant or optimal policies. In practice, however, most off-policy methods are limited in their ability to learn entirely from off-policy data. Recent off-policy RL methods (e.g., [Haarnoja et al., 2018, Munos et al., 2016, Kalashnikov et al., 2018, Espeholt et al., 2018]) have demonstrated sample-efficient performance on complex tasks in robotics [Kalashnikov et al., 2018] and simulated environments [Todorov et al., 2012], but these methods can still fail to learn when presented with arbitrary off-policy data without the opportunity to collect more experience from the environment. This issue persists even when the off-policy data comes from effective expert policies, which in principle should address any exploration challenge [De Bruin et al., 2015, Fujimoto et al., 2019b, Fu et al., 2019]. This sensitivity to the training data distribution is a limitation of practical off-policy RL algorithms, and one would hope that an off-policy algorithm should be able to learn reasonable policies through training on static datasets before being deployed in the real world.

In this chapter, we motivate an off-policy, value-based RL method that can learn from static datasets. As we show, a crucial challenge in applying value-based methods to off-policy scenarios arises in the bootstrapping process employed when Q-functions are evaluated on out of *out-of-distribution* action inputs for computing the backup when training from off-policy data. This may introduce errors in the Q-function and the algorithm is unable to collect new data in order to remedy those errors, making training unstable and potentially diverging.

First, we formalize and analyze the reasons for instability and poor performance when learning from off-policy data. We show that, through careful action selection, error propagation through the Q-function can be mitigated. We then propose a principled algorithm called *bootstrapping error accumulation reduction* (BEAR) to control bootstrapping error in practice, which uses the notion of *support-set matching* to prevent error accumulation. Through systematic experiments, we show the effectiveness of our method on continuous-control MuJoCo tasks, with a variety of off-policy datasets: generated by a random, suboptimal, or optimal policies. BEAR is consistently robust to the training dataset, matching or exceeding the state-of-the-art in all cases, whereas existing algorithms only perform well for specific datasets.

4.1 Relation to Prior Work

Our approach constrains actor updates so that the actions remain in the support of the training dataset distribution. Several works have explored similar ideas in the context of off-policy learning learning in online settings. Kakade and Langford [2002] shows that large policy updates can be destructive, and propose a conservative policy iteration scheme which constrains actor updates to be small for provably convergent learning. Grau-Moya et al. [2018] use a learned prior over actions in the maximum entropy RL framework [Levine, 2018] and justify it as a regularizer based on mutual information. However, none of these methods use static datasets. Importance Sampling based distribution re-weighting Munos et al. [2016], Gelada and Bellemare [2019], Precup et al. [2001], Mahmood et al. [2015] has also been explored primarily in the context of off-policy policy evaluation.

As discussed in Chapter 3, our algorithm is also closely related to batch-constrained Q-learning (BCQ) [Fujimoto et al., 2019b] and SPIBB [Laroche et al., 2019]. Our work differs in that we prove stronger results under approximation errors and provide a bound on the *suboptimality* of the solution. This is crucial as it drives the design choices for a practical algorithm. Empirically, we find BEAR achieves stronger and more consistent results than BCQ across a wide variety of datasets and environments. As we explain below, the BCQ constraint is too aggressive; BCQ generally fails to substantially improve over the behavior policy, while our method actually improves when the data collection policy is suboptimal or random. SPIBB [Laroche et al., 2019], like BEAR, is an algorithm based on constraining the learned policy to the support of a behavior policy. However, the authors do not extend safe performance guarantees from the batch-constrained case to the relaxed support-constrained case, and do not evaluate on high-dimensional control tasks. REM [Agarwal et al., 2020b] is a concurrent work that uses a random convex combination of an ensemble of Q-networks to perform offline reinforcement learning from a static dataset consisting of interaction data generated while training a DQN agent.

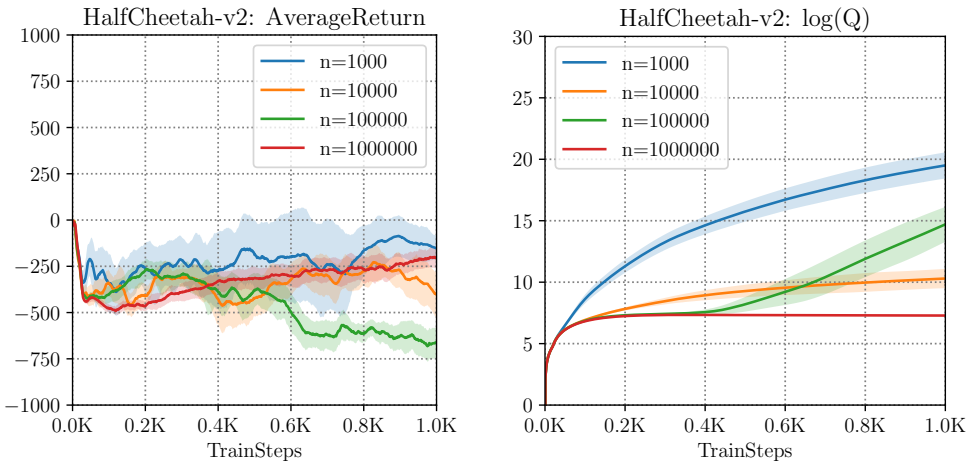


Figure 4.1: Performance of SAC on HalfCheetah-v2 (return (left) and log Q-values (right)) with off-policy expert data w.r.t. number of training samples (n). Note the large discrepancy between returns (which are negative) and log Q-values (which have large positive values), which is not solved with additional samples.

4.2 Out-of-Distribution Actions in Q-Learning

Q-learning methods often fail to learn on static, off-policy data, as shown in Figure 4.1. At first glance, this resembles overfitting, but increasing the size of the static dataset does not rectify the problem, suggesting the issue is more complex. We can understand the source of this instability by examining the form of the Bellman backup. Although minimizing the mean squared Bellman error corresponds to a supervised regression problem, the targets for this regression are themselves derived from the current Q-function estimate. The targets are calculated by maximizing the learned Q-values with respect to the action at the next state. However, the Q-function estimator is only reliable on inputs from the same distribution as its training set. As a result, naïvely maximizing the value may evaluate the $\hat{Q}$ estimator on actions that lie far outside of the training distribution, resulting in pathological values that incur large error. We refer to these actions as out-of-distribution (OOD) actions.

Formally, let $\zeta_k(s,a) = |Q_k(s,a) - Q^*(s,a)|$ denote the total error at iteration k of Q-learning, and let $\delta_k(s,a) = |Q_k(s,a) - \mathcal{T}Q_{k-1}(s,a)|$ denote the current Bellman error. Then, we have $\zeta_k(s,a) \leq \delta_k(s,a) + \gamma \max_{a'} \mathbb{E}_{s'}[\zeta_{k-1}(s',a')]$. In other words, errors from (s',a') are discounted, then accumulated with new errors $\delta_k(s,a)$ from the current iteration. We expect $\delta_k(s,a)$ to be high on OOD states and actions, as errors at these state-actions are never directly minimized while training.

To mitigate bootstrapping error, we can restrict the policy to ensure that it output actions that lie in the support of the training distribution. This is distinct from previous work (e.g., BCQ [Fujimoto et al., 2019b]) which implicitly constrains the *distribution* of the learned

policy to be close to the behavior policy, similarly to behavioral cloning Schaal [1999]. While this is sufficient to ensure that actions lie in the training set with high probability, it is overly restrictive. For example, if the behavior policy is close to uniform, the learned policy will behave randomly, resulting in poor performance, even when the data is sufficient to learn a strong policy (see Figure 4.2 for an illustration). Formally, this means that a learned policy $\pi(a|s)$ has positive density *only where* the density of the behaviour policy $\beta(a|s)$ is more than a threshold (i.e., $\forall a, \beta(a|s) \leq \varepsilon \implies \pi(a|s) = 0$), instead of a closeness constraint on the value of the density $\pi(a|s)$ and $\beta(a|s)$. Our analysis instead reveals a tradeoff between staying within the data distribution and finding a suboptimal solution when the constraint is too restrictive. Our analysis motivates us to restrict the support of the learned policy, but not the probabilities of the actions lying within the support. This avoids evaluating the Q-function estimator on OOD actions, but remains flexible in order to find a performant policy. Our proposed algorithm leverages this insight.

Distribution-Constrained Backups

In this section, we define and analyze a backup operator that restricts the set of policies used in the maximization of the Q-function, and we derive performance bounds which depend on the restricted set. This provides motivation for constraining policy support to the data distribution. We begin with the definition of a distribution-constrained operator:

Definition 4.2.1 (Distribution-constrained operators)**.** *Given a set of policies* Π *, the distribution-constrained backup operator is defined as:*

$$\mathcal{T}^{\Pi}Q(s,a) \stackrel{\text{def}}{=} \mathbb{E}\big[R(s,a) + \gamma \max_{\pi \in \Pi} \mathbb{E}_{P(s'|s,a)}\left[V(s')\right]\big] \qquad V(s) \stackrel{\text{def}}{=} \max_{\pi \in \Pi} \mathbb{E}_{\pi}[Q(s,a)] \quad .$$

This backup operator satisfies properties of the standard Bellman backup, such as convergence to a fixed point, as discussed in Appendix A.1. To analyze the (sub)optimality of performing this backup under approximation error, we first quantify two sources of error. The first is a *suboptimality bias*. The optimal policy may lie outside the policy constraint set, and thus a suboptimal solution will be found. The second arises from distribution shift between the training distribution and the policies used for backups. This formalizes the notion of OOD actions. To capture suboptimality in the final solution, we define a *suboptimality constant*, which measures how far π^* is from Π.

Definition 4.2.2 (Suboptimality constant)**.** *The suboptimality constant is defined as:*

$$\alpha(\Pi) = \max_{s,a} |\mathcal{T}^{\Pi}Q^*(s,a) - \mathcal{T}Q^*(s,a)|.$$

Next, we define a concentrability coefficient [Munos, 2005], which quantifies how far the visitation distribution generated by policies from Π is from the training data distribution. This constant captures the degree to which states and actions are out of distribution.

Assumption 4.2.1 (Concentrability)**.** *Let ρ_0 denote the initial state distribution, and $\mu(s,a)$ denote the distribution of the training data over $\mathcal{S} \times \mathcal{A}$, with marginal $\mu(s)$ over $\mathcal{S}$. Suppose there exist coefficients $c(k)$ such that for any $\pi_1, ...\pi_k \in \Pi$ and $s \in \mathcal{S}$:*

$$\rho_0 P^{\pi_1} P^{\pi_2} ... P^{\pi_k}(s) \leq c(k)\mu(s),$$

where P^{π_i} is the transition operator on states induced by π_i. Then, define the concentrability coefficient $C(\Pi)$ as

$$C(\Pi) \stackrel{\text{def}}{=} (1-\gamma)^2 \sum_{k=1}^{\infty} k\gamma^{k-1} c(k).$$

To provide some intuition for $C(\Pi)$, if μ was generated by a single policy π, and $\Pi = \{\pi\}$ was a singleton set, then we would have $C(\Pi) = 1$, which is the smallest possible value. However, if Π contained policies far from π, the value could be large, potentially infinite if the support of Π is not contained in π. Now, we bound the performance of approximate distribution-constrained Q-iteration:

Theorem 4.2.1. *Suppose we run approximate distribution-constrained value iteration with a set constrained backup $\mathcal{T}^{\Pi}$. Assume that $\delta(s,a) \geq \max_k |Q_k(s,a) - \mathcal{T}^{\Pi} Q_{k-1}(s,a)|$ bounds the Bellman error. Then,*

$$\lim_{k\to\infty} \mathbb{E}_{\rho_0}[|V^{\pi_k}(s) - V^*(s)|] \leq \frac{\gamma}{(1-\gamma)^2} \left[C(\Pi) \mathbb{E}_{\mu}[\max_{\pi \in \Pi} \mathbb{E}_{\pi}[\delta(s,a)]] + \frac{1-\gamma}{\gamma} \alpha(\Pi) \right]$$

Proof. See Appendix A.2, Theorem A.2.1 □

This bound formalizes the tradeoff between keeping policies chosen during backups close to the data (captured by $C(\Pi)$) and keeping the set Π large enough to capture well-performing policies (captured by $\alpha(\Pi)$). When we expand the set of policies Π, we are increasing $C(\Pi)$ but decreasing $\alpha(\Pi)$. An example of this tradeoff, and how a careful choice of Π can yield superior results, is given in a tabular gridworld example in Fig. 4.2, where we visualize errors accumulated during distribution-constrained Q-iteration for different choices of Π.

Finally, we motivate the use of support sets to construct Π. We are interested in the case where $\Pi_\epsilon = \{\pi \mid \pi(a|s) = 0 \text{ whenever } \beta(a|s) < \epsilon\}$, where β is the behavior policy (i.e., Π is the set of policies that have support in the probable regions of the behavior policy). Defining Π_ϵ in this way allows us to bound the concentrability coefficient:

Theorem 4.2.2. *Assume the data distribution μ is generated by a behavior policy β. Let $\mu(s)$ be the marginal state distribution under the data distribution. Define $\Pi_\epsilon = \{\pi \mid \pi(a|s) = 0$ whenever $\beta(a|s) < \epsilon\}$ and let μ_{Π_ϵ} be the highest discounted marginal state distribution*

starting from the initial state distribution ρ and following policies $\pi \in \Pi_\epsilon$ at each time step thereafter. Then, there exists a concentrability coefficient $C(\Pi_\epsilon)$ which is bounded:

$$C(\Pi_\epsilon) \leq C(\beta) \cdot \left(1 + \frac{\gamma}{(1-\gamma)f(\epsilon)}(1-\epsilon)\right)$$

where $f(\epsilon) \stackrel{\text{def}}{=} \min_{s \in \mathcal{S}, \mu_{\Pi_\epsilon}(s) > 0}[\mu(s)] > 0$.

Proof. See Appendix A.2, Theorem A.2.2 □

Qualitatively, $f(\epsilon)$ is the minimum discounted visitation marginal of a state under the behaviour policy if only actions which are more than ϵ likely are executed in the environment. Thus, using support sets gives us a single lever, ϵ, which simultaneously trades off the value of $C(\Pi)$ and $\alpha(\Pi)$. Not only can we provide theoretical guarantees, we will see in our experiments (Sec. 4.4) that constructing Π in this way provides a simple and effective method for implementing distribution-constrained algorithms.

Intuitively, this means we can prevent an increase in overall error in the Q-estimate by selecting policies supported on the support of the training action distribution, which would ensure roughly bounded projection error $\delta_k(s,a)$ while reducing the suboptimality bias, potentially by a large amount. Bounded error $\delta_k(s,a)$ on the support set of the training distribution is a reasonable assumption when using highly expressive function approximators, such as deep networks, especially if we are willing to reweight the transition set Schaul et al. [2016], Fu et al. [2019]. We further elaborate on this point in Appendix A.3.

4.3 BEAR: Bootstrapping Error Accumulation Reduction

We now propose a practical actor-critic algorithm (built on the framework of TD3 Fujimoto et al. [2018] or SAC Haarnoja et al. [2018]) that uses distribution-constrained backups to reduce accumulation of bootstrapping error. The key insight is that we can search for a policy with the same support as the training distribution, while preventing accidental error accumulation. Our algorithm has two main components. Analogous to BCQ [Fujimoto et al., 2018], we use K Q-functions and use the minimum Q-value for policy improvement, and design a constraint which will be used for searching over the set of policies Π_ϵ, which share the same support as the behaviour policy. Both of these components will appear as modifications of the policy improvement step in actor-critic style algorithms. We also note that policy improvement can be performed with the mean of the K Q-functions, and we found that this scheme works as good in our experiments.

We denote the set of Q-functions as: $\hat{Q}_1, \cdots, \hat{Q}_K$. Then, the policy is updated to maximize the conservative estimate of the Q-values within Π_ϵ:

$$\pi_\phi(s) := \max_{\pi \in \Pi_\epsilon} \mathbb{E}_{a \sim \pi(\cdot|s)} \left[\min_{j=1,\ldots,K} \hat{Q}_j(s,a) \right]$$

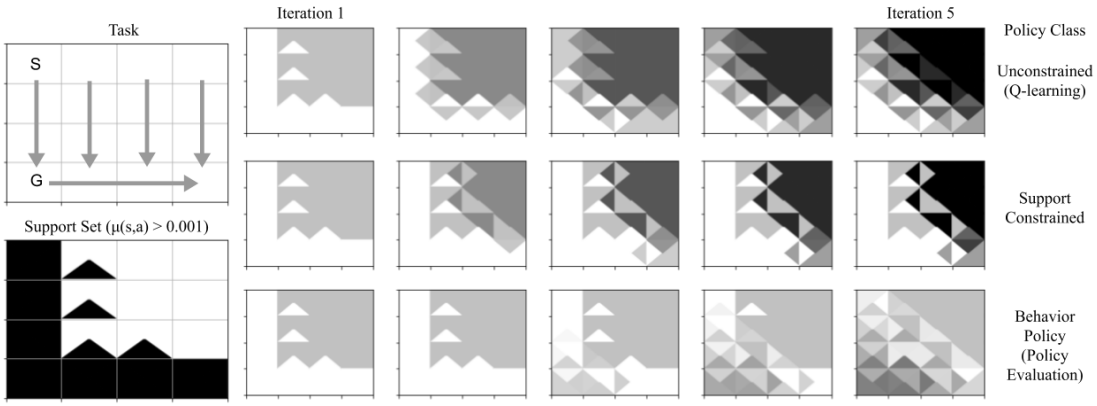


Figure 4.2: Visualized error propagation in Q-learning for various choices of the constraint set Π: unconstrained (top row) distribution-constrained (middle), and constrained to the behaviour policy (policy-evaluation, bottom). Triangles represent Q-values for actions that move in different directions. The task (left) is to reach the bottom-left corner (G) from the top-left (S), but the behaviour policy (visualized as arrows in the task image, support state-action pairs are shown in black on the support set image) travels to the bottom-right with a small amount of ϵ-greedy exploration. Dark values indicate high error, and light values indicate low error. Standard backups propagate large errors from the low-support regions into the high-support regions, leading to high error. Policy evaluation reduces error propagation from low-support regions, but introduces significant suboptimality bias, as the data policy is not optimal. A carefully chosen distribution-constrained backup strikes a balance between these two extremes, by confining error propagation in the low-support region while introducing minimal suboptimality bias.

In practice, the behavior policy β is unknown, so we need an approximate way to constrain π to to the set Π. To this end, We define a differentiable constraint that approximately constrains π to Π, and then approximately solve the constrained optimization problem via dual gradient descent. We use the sampled version of maximum mean discrepancy (MMD) [Gretton et al., 2012] between the unknown behaviour policy β and the actor π because it can be estimated based solely on samples from the distributions. Given samples $x_1, \cdots, x_n \sim P$ and $y_1, \cdots, y_m \sim Q$, the sampled MMD between P and Q is given by:

$$\text{MMD}^2(\{x_1, \cdots, x_n\}, \{y_1, \cdots, y_m\}) = \frac{1}{n^2} \sum_{i,i'} k(x_i, x_{i'}) - \frac{2}{nm} \sum_{i,j} k(x_i, y_j) + \frac{1}{m^2} \sum_{j,j'} k(y_j, y_{j'}).$$

Here, $k(\cdot, \cdot)$ is any universal kernel. In our experiments, we find both Laplacian and Gaussian kernels work well. The expression for MMD does not involve the density of either distribution and it can be optimized directly through samples. Empirically we find that, in the low-intermediate sample regime, the sampled MMD between P and Q is similar to the MMD between a uniform distribution over P's support and Q, which makes MMD roughly suited

for constraining distributions to a given support set. (See Appendix A.3 for numerical simulations justifying this approach).

Putting everything together, the optimization problem in the policy improvement step is

$$\pi_\phi := \max_{\pi \in \Delta_{|S|}} \mathbb{E}_{s\sim\mathcal{D}} \mathbb{E}_{a\sim\pi(\cdot|s)} \left[\min_{j=1,..,K} \hat{Q}_j(s,a) \right] \text{ s.t. } \mathbb{E}_{s\sim\mathcal{D}}[\text{MMD}(\mathcal{D}(s), \pi(\cdot|s))] \leq \varepsilon \quad (4.1)$$

where ε is an approximately chosen threshold. We choose a threshold of $\varepsilon = 0.05$ in our experiments. The algorithm is summarized in Algorithm 1.

How does BEAR connect with distribution-constrained backups described in Section 4.1? Step 5 of the algorithm restricts π_ϕ to lie in the support of β. This insight is formally justified in Theorems 4.1 & 4.2 ($C(\Pi_\varepsilon)$ is bounded). Computing distribution-constrained backup exactly by maximizing over $\pi \in \Pi_\varepsilon$ is intractable in practice. As an approximation, we sample Dirac policies in the support of β (Alg 1, Line 5) and perform empirical maximization to compute the backup. As the maximization is performed over a *narrower* set of Dirac policies ($\{\delta_{a_i}\} \subseteq \Pi_\varepsilon$), the bound in Theorem 4.1 still holds. Empirically, we show in Section 4.4 that this approximation is sufficient to outperform previous methods.

Algorithm 1 BEAR Q-Learning (BEAR-QL)

input : Dataset $\mathcal{D}$, target network update rate τ, mini-batch size N, sampled actions for MMD n, minimum λ.

1: Initialize Q-ensemble $\{Q_{\theta_i}\}_{i=1}^K$, actor π_ϕ, Lagrange multiplier α, target networks $\{Q_{\theta'_i}\}_{i=1}^K$, and a target actor $\pi_{\phi'}$, with $\phi' \leftarrow \phi, \theta'_i \leftarrow \theta_i$
2: **for** t in {1, . . . , N} **do**
3: Sample mini-batch of transitions $(s, a, r, s') \sim \mathcal{D}$
Q-update:
4: Sample p action samples, $\{a_i \sim \pi_{\phi'}(\cdot|s')\}_{i=1}^p$
5: Define $y(s,a) := \max_{a_i}[\lambda \min_{j=1,..,K} Q_{\theta'_j}(s', a_i) + (1-\lambda) \max_{j=1,..,K} Q_{\theta'_j}(s', a_i)]$
6: $\forall i, \theta_i \leftarrow \arg\min_{\theta_i}(Q_{\theta_i}(s,a) - (r + \gamma y(s,a)))^2$
Policy-update:
7: Sample actions $\{\hat{a}_i \sim \pi_\phi(\cdot|s)\}_{i=1}^m$ and $\{a_j \sim \mathcal{D}(s)\}_{j=1}^n$, n preferably an intermediate integer(1-10)
8: Update ϕ, α by minimizing Equation 4.1 by using dual gradient descent with Lagrange multiplier α
9: **Update Target Networks:** $\theta'_i \leftarrow \tau\theta_i + (1-\tau)\theta'_i$; $\phi' \leftarrow \tau\phi + (1-\tau)\phi'$
10: **end for**

In summary, the actor is updated towards maximizing the Q-function while still being constrained to remain in the valid search space defined by Π_ϵ. The Q-function uses actions sampled from the actor to then perform distribution-constrained Q-learning, over a reduced set of policies. At test time, we sample p actions from $\pi_\phi(s)$ and the Q-value maximizing action out of these is executed in the environment. Implementation and other details are present in Appendix A.4.

4.4 Experiments

In our experiments, we study how BEAR performs when learning from static off-policy data on a variety of continuous control benchmark tasks. We evaluate our algorithm in three settings: when the dataset $\mathcal{D}$ is generated by **(1)** a completely random behaviour policy, **(2)** a partially trained, medium scoring policy, and **(3)** an optimal policy. Condition **(2)** is of particular interest, as it captures many common use-cases in practice, such as learning from imperfect demonstration data (e.g., of the sort that are commonly available for autonomous driving Gao et al. [2018]), or reusing previously collected experience during off-policy RL. We compare our method to several prior methods: a baseline actor-critic algorithm (TD3), the BCQ algorithm [Fujimoto et al., 2019b], which aims to address a similar problem, as discussed in Section 4.2, KL-control [Jaques et al., 2019] (which solves a KL-penalized RL problem similarly to maximum entropy RL), a static version of DQfD [Hester et al., 2018] (where a constraint to upweight Q-values of state-action pairs observed in the dataset is added as an auxiliary loss on top a regular actor-critic algorithm), and a behaviour cloning (BC) baseline, which simply imitates the data distribution. This serves to measure whether each method actually performs effective RL, or simply copies the data. We report the average evaluation return over 5 seeds of the policy given by the learned algorithm, in the form of a learning curve as a function of number of gradient steps taken by the algorithm. These samples are only collected for evaluation, and are not used for training.

Performance on Medium-Quality Data

We first discuss the evaluation of condition with "mediocre" data **(2)**, as this condition resembles the settings where we expect training on offline data to be most useful. We collected one million transitions from a partially trained policy, so as to simulate imperfect demonstration data or data from a mediocre prior policy. In this scenario, we found that BEAR-QL consistently outperforms both BCQ Fujimoto et al. [2019b] and a naïve off-policy RL baseline (TD3) by large margins, as shown in Figure 4.3. This scenario is the most relevant from an application point of view, as access to optimal data may not be feasible, and random data might have inadequate exploration to efficient learn a good policy. We also evaluate the accuracy with which the learned Q-functions predict actual policy returns. These trends are provided in Appendix A.5. Note that the performance of BCQ often tracks the performance of the BC baseline, suggesting that BCQ primarily imitates the data. Our KL-control baseline uses automatic temperature tuning [Haarnoja et al., 2018]. We find that KL-control usually performs similar or worse to BC, whereas DQfD tends to diverge often due to cumulative error due to OOD actions and often exhibits a huge variance across different runs (for example, HalfCheetah-v2 environment).

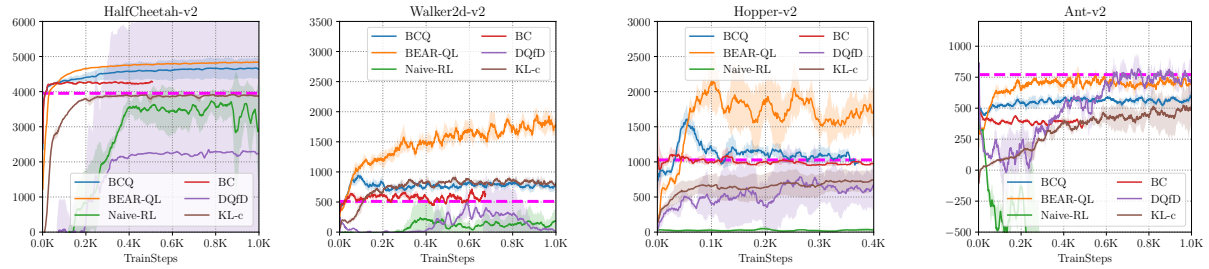


Figure 4.3: Average performance of BEAR-QL, BCQ, Naïve RL and BC on medium-quality data averaged over 5 seeds. BEAR-QL outperforms both BCQ and Naïve RL. Average return over the training data is indicated by the magenta line. One step on the x-axis corresponds to 1000 gradient steps.

Performance on Random and Optimal Datasets

In Figure 4.5, we show the performance of each method when trained on data from a random policy (top) and a near-optimal policy (bottom). In both cases, our method BEAR achieves good results, consistently exceeding the average dataset return on random data, and matching the optimal policy return on optimal data. Naïve RL also often does well on random data. For a random data policy, all actions are in-distribution, since they all have equal probability. This is consistent with our hypothesis that OOD actions are one of the main sources of error in off-policy learning on static datasets. The prior BCQ method Fujimoto et al. [2019b] performs well on optimal data but performs poorly on random data, where the constraint is too strict. These results show that BEAR-QL is robust to the dataset composition, and can learn consistently in a variety of settings. We find that KL-control and DQfD can be unstable in these settings.

Finally, in Figure 4.4, we show that BEAR outperforms other considered prior methods in the challenging Humanoid-v2 environment as well, in two cases – Medium-quality data and random data.

Analysis of BEAR-QL

In this section, we aim to analyze different components of our method via an ablation study. Our first ablation studies the support constraint discussed in Section 4.3, which uses MMD to measure support. We replace it with a more standard KL-divergence distribution constraint, which measures similarity in density. Our hypothesis is that this should provide a more conservative constraint, since matching distributions is not necessary for matching support. KL-divergence performs well in some cases, such as with optimal data, but as shown in Figure 4.6, it performs worse than MMD on medium-quality data. Even when KL-divergence is hand tuned fully, so as to prevent instability issues it still performs worse than a not-well tuned MMD constraint. We provide the results for this setting in the Appendix.

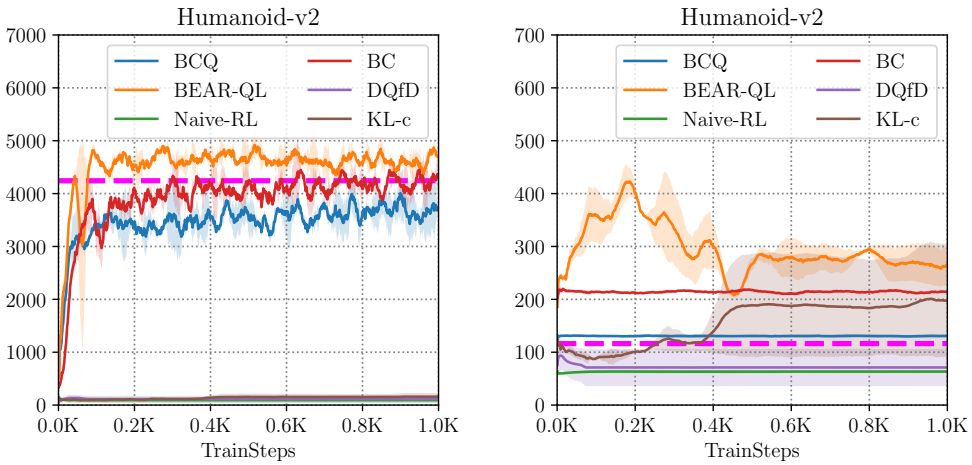


Figure 4.4: Performance of BEAR-QL, BCQ, Naïve RL and BC on medium-quality (left) and random (right) data in the Humanoid-v2 environment. Note that BEAR-QL outperforms prior methods.

We also vary the number of samples n that are used to compute the MMD constraint. We find that smaller n ($\approx$ 4 or 5) gives better performance. Although the difference is not large, consistently better performance with 4 samples leans in favour of our hypothesis that an intermediate number of samples works well for support matching, and hence is less restrictive.

4.5 Discussion

The goal in this chapter was to study off-policy reinforcement learning with static datasets. We theoretically and empirically analyze how error propagates in off-policy RL due to the use of out-of-distribution actions for computing the target values in the Bellman backup. Our experiments suggest that this source of error is one of the primary issues afflicting off-policy RL: increasing the number of samples does not appear to mitigate the degradation issue (Figure 4.1), and training with naïve RL on data from a random policy, where there are no out-of-distribution actions, shows much less degradation than training on data from more focused policies (Figure 4.5). Armed with this insight, we develop a method for mitigating the effect of out-of-distribution actions, which we call BEAR-QL. BEAR-QL constrains the backup to use actions that have non-negligible support under the data distribution, but without being overly conservative in constraining the learned policy. We observe experimentally that BEAR-QL achieves good performance across a range of tasks, and across a range of dataset compositions, learning well on random, medium-quality, and expert data.

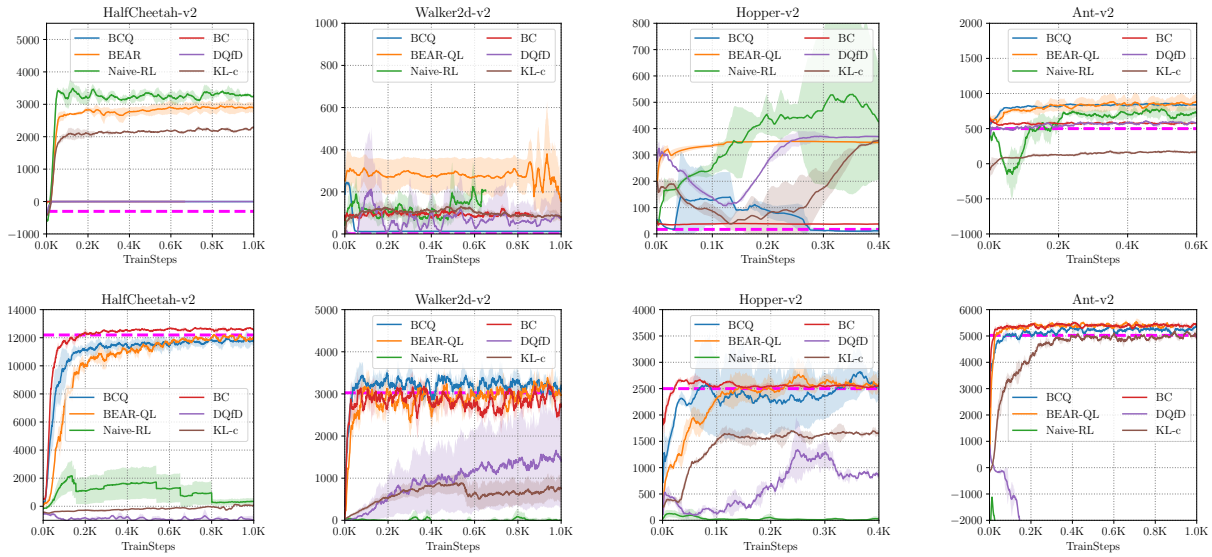


Figure 4.5: Average performance of BEAR-QL, BCQ, Naïve RL and BC on random data (top row) and optimal data (bottom row) over 5 seeds. BEAR-QL is the only algorithm capable of learning in both scenarios. Naïve RL cannot handle optimal data, since it does not illustrate mistakes, and BCQ favors a behavioral cloning strategy (performs quite close to behaviour cloning in most cases), causing it to fail on random data. Average return over the training dataset is indicated by the dashed magenta line.

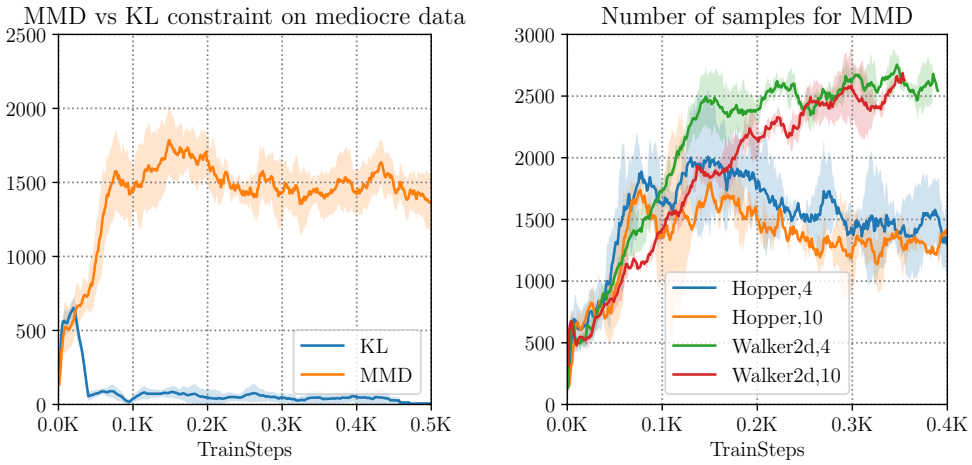


Figure 4.6: Average return (averaged Hopper-v2 and Walker2d-v2) as a function of train steps for ablation studies from Section 4.4. (a) MMD constrained optimization is more stable and leads to better returns, (b) 4 sample MMD is more performant than 10.

Chapter 5

Model-Based Policy Optimization

In Chapter 4, we explored the effect of distribution shift during policy evaluation in the *model-free* setting. We now move on the analyze the effect of these errors on performance in the model-based RL setting, where we will find that errors can likewise be categorized into those originating from in-distribution generalization errors (such as Bellman errors) and distribution shift errors (such as those captured by the concentrability coefficient). While we experimentally evaluate these ideas in an online model-based setting, similar ideas can be applied to the offline setting, which has been explored in works such as MOReL and MOPO [Kidambi et al., 2020, Yu et al., 2020].

While model-free methods have shown promise as a general-purpose tool for learning complex policies from raw state inputs [Mnih et al., 2015, Lillicrap et al., 2016, Haarnoja et al., 2018], but their generality comes at the cost of efficiency. When dealing with real-world physical systems, for which data collection can be an arduous process, model-based approaches are appealing due to their comparative sample complexity. Conceptually, model-based methods are also simple: one can use an offline dataset to construct and estimate of the dynamics of the system, and then use any policy optimization or planning algorithm within the learned dynamics to obtain a policy.

In this chapter, we study how to most effectively use a predictive model for policy optimization. We first formulate and analyze a class of model-based reinforcement learning algorithms with improvement guarantees. Although there has been recent interest in monotonic improvement of model-based reinforcement learning algorithms [Sun et al., 2018, Luo et al., 2019], most commonly used model-based approaches lack the improvement guarantees that underpin many model-free methods [Schulman et al., 2015]. While it is possible to apply analogous techniques to the study of model-based methods to achieve similar guarantees, it is more difficult to use such analysis to justify model usage in the first place due to pessimistic bounds on model error. However, we show that more realistic model error rates derived empirically allow us to modify this analysis to provide a more reasonable tradeoff on model usage.

We build on these insights to propose MBPO (model-based policy optimization), which makes limited use of a predictive model to achieve pronounced improvements in perfor-

mance compared to other model-based approaches. More specifically, we disentangle the task horizon and model horizon by querying the model only for short rollouts. We empirically demonstrate that a large amount of these short model-generated rollouts can allow a policy optimization algorithm to learn substantially faster than recent model-based alternatives while retaining the asymptotic performance of the most competitive model-free algorithms. We also show that MBPO does not suffer from the same pitfalls as prior model-based approaches, avoiding model exploitation and failure on long-horizon tasks. Finally, we empirically investigate different strategies for model usage, supporting the conclusion that careful use of short model-based rollouts provides the most benefit to a reinforcement learning algorithm.

5.1 Relation to Prior Work

The primary contribution this work is a theoretical analysis of error accumulation in model-based reinforcement learning due to generalization and distribution shift, as well as a practical algorithm combining an ensemble-based dynamics model (as in [Chua et al., 2018]) with a model-free planner.

Learned models may be incorporated into otherwise model-free methods for improvements in data efficiency. For example, a model-free policy can be used as an action proposal distribution within a model-based planner [Piché et al., 2019]. Conversely, model rollouts may be used to provide extra training examples for a Q-function [Sutton, 1990], to improve the target value estimates of existing data points [Feinberg et al., 2018], or to provide additional context to a policy [Du and Narasimhan, 2019]. However, the performance of such approaches rapidly degrades with increasing model error [Gu et al., 2016], motivating work that interpolates between different rollout lengths [Buckman et al., 2018], tunes the ratio of real to model-generated data [Kalweit and Boedecker, 2017], or does not rely on model predictions [Heess et al., 2015]. Our approach similarly tunes model usage during policy optimization, but we show that justifying non-negligible model usage during most points in training requires consideration of the model's ability to generalize outside of its training distribution.

Prior methods have also explored incorporating computation that resembles model-based planning but without constraining the intermediate predictions of the planner to match plausible environment observations [Tamar et al., 2016, Racanière et al., 2017, Oh et al., 2017, Silver et al., 2017]. While such methods can reach asymptotic performance on par with model-free approaches, they may not benefit from the sample efficiency of model-based methods as they forgo the extra supervision used in standard model-based methods.

The bottleneck in scaling model-based approaches to complex tasks often lies in learning reliable predictive models of high-dimensional dynamics [Atkeson and Schaal, 1997]. While ground-truth models are most effective when queried for long horizons [Holland et al., 2018], inaccuracies in learned models tend to make long rollouts unreliable. Ensembles have shown to be effective in preventing a policy or planning procedure from exploiting such inaccuracies

Algorithm 2 Monotonic Model-Based Policy Optimization

1: Initialize policy $\pi(a|s)$, predictive model $p_\theta(s', r|s, a)$, empty dataset $\mathcal{D}$.
2: **for** N epochs **do**
3: Collect data with π in real environment: $\mathcal{D} = \mathcal{D} \cup \{(s_i, a_i, s'_i, r_i)\}_i$
4: Train model p_θ on dataset $\mathcal{D}$ via maximum likelihood: $\theta \leftarrow \text{argmax}_\theta \mathbb{E}_\mathcal{D}[\log p_\theta(s', r|s, a)]$
5: Optimize policy under predictive model: $\pi \leftarrow \text{argmax}_{\pi'} \hat{J}(\pi') - C(\epsilon_m, \epsilon_\pi)$
6: **end for**

[Rajeswaran et al., 2017, Kurutach et al., 2018, Clavera et al., 2018, Chua et al., 2018]. Alternatively, a model may also be trained on its own outputs to avoid compounding error from multi-step predictions [Talvitie, 2014, 2016] or predict many timesteps into the future [Whitney and Fergus, 2019]. We demonstrate that a combination of model ensembles with short model rollouts is sufficient to prevent model exploitation.

5.2 Monotonic Improvement with Model Bias

In this section, we first lay out a general recipe for MBPO with monotonic improvement. This general recipe resembles or subsumes several prior algorithms and provides us with a concrete framework that is amenable to theoretical analysis. Described generically in Algorithm 2, MBPO optimizes a policy under a learned model, collects data under the updated policy, and uses that data to train a new model. While conceptually simple, the performance of MBPO can be difficult to understand; errors in the model can be exploited during policy optimization, resulting in large discrepancies between the predicted returns of the policy under the model and under the true dynamics.

Monotonic model-based improvement

Our goal is to outline a principled framework in which we can provide performance guarantees for model-based algorithms. To show monotonic improvement for a model-based method, we wish to construct a bound of the following form:

$$J(\pi) \geq \hat{J}(\pi) - C.$$

$J(\pi)$ denotes the returns of the policy in the true MDP, whereas $\hat{J}(\pi)$ denotes the returns of the policy under our model. Such a statement guarantees that, as long as we improve by at least C under the model, we can guarantee improvement on the true MDP.

The gap between true returns and model returns, C, can be expressed in terms of two error quantities of the model: generalization error due to sampling, and distribution shift due to the updated policy encountering states not seen during model training. As the model is trained with supervised learning, sample error can be quantified by standard PAC generalization

bounds, which bound the difference in expected loss and empirical loss by a constant with high probability [Shalev-Shwartz and Ben-David, 2014]. We denote this generalization error by $\epsilon_m = \max_t \mathbb{E}_{s\sim\pi_{D,t}}[D_{TV}(p(s',r|s,a)||p_\theta(s',r|s,a))]$, which can be estimated in practice by measuring the validation loss of the model on the time-dependent state distribution of the data-collecting policy π_D. For our analysis, we denote distribution shift by the maximum total-variation distance, $\max_s D_{TV}(\pi||\pi_D) \leq \epsilon_\pi$, of the policy between iterations. In practice, we measure the KL divergence between policies, which we can relate to ϵ_π by Pinsker's inequality. With these two sources of error controlled (generalization by ϵ_m, and distribution shift by ϵ_π), we now present our bound:

Theorem 5.2.1. *Let the expected TV-distance between two transition distributions be bounded at each timestep by ϵ_m and the policy divergence be bounded by ϵ_π. Let $R_{\max}$ denote the maximum possible reward. Then the true returns and model returns of the policy are bounded as:*

$$J(\pi) \geq \hat{J}(\pi) - \underbrace{\left[\frac{2\gamma R_{\max}(\epsilon_m + 2\epsilon_\pi)}{(1-\gamma)^2} + \frac{4R_{\max}\epsilon_\pi}{(1-\gamma)}\right]}_{C(\epsilon_m,\epsilon_\pi)} \tag{5.1}$$

Proof. See Section B.1, Theorem B.1.1. □

This bound implies that as long as we improve the returns under the model $\hat{J}(\pi)$ by more than $C(\epsilon_m, \epsilon_\pi)$, we can guarantee improvement under the true returns.

Interpolating Model-Based and Model-Free Updates

Theorem 5.2.1 provides a useful relationship between model returns and true returns. However, it contains several issues regarding cases when the model error ϵ_m is high. First, there may not exist a policy such that $\hat{J}(\pi) - J(\pi) > C(\epsilon_m, \epsilon_\pi)$, in which case improvement is not guaranteed. Second, the analysis relies on running full rollouts through the model, allowing model errors to compound. This is reflected in the bound by a factor scaling quadratically with the effective horizon, $1/(1-\gamma)$. In such cases, we can improve the algorithm by choosing to rely less on the model and instead more on real data collected from the true dynamics when the model is inaccurate.

In order to allow for dynamic adjustment between model-based and model-free rollouts, we introduce the notion of a *branched rollout*, in which we begin a rollout from a state under the previous policy's state distribution $d_{\pi_D}(s)$ and run k steps according to π under the learned model p_θ. This branched rollout structure resembles the scheme proposed in the original Dyna algorithm [Sutton, 1990], which can be viewed as a special case of a length 1 branched rollouts. Formally, we can view this as executing a nonstationary policy which begins a rollout by sampling actions from the previous policy π_D. Then, at some specified time, we switch to unrolling the trajectory under the model p and current policy π for k steps. For simplicity, we present theorems for a version of branching (described in Appendix B.1) which unrolls each state along a trajectory by k steps under the model,

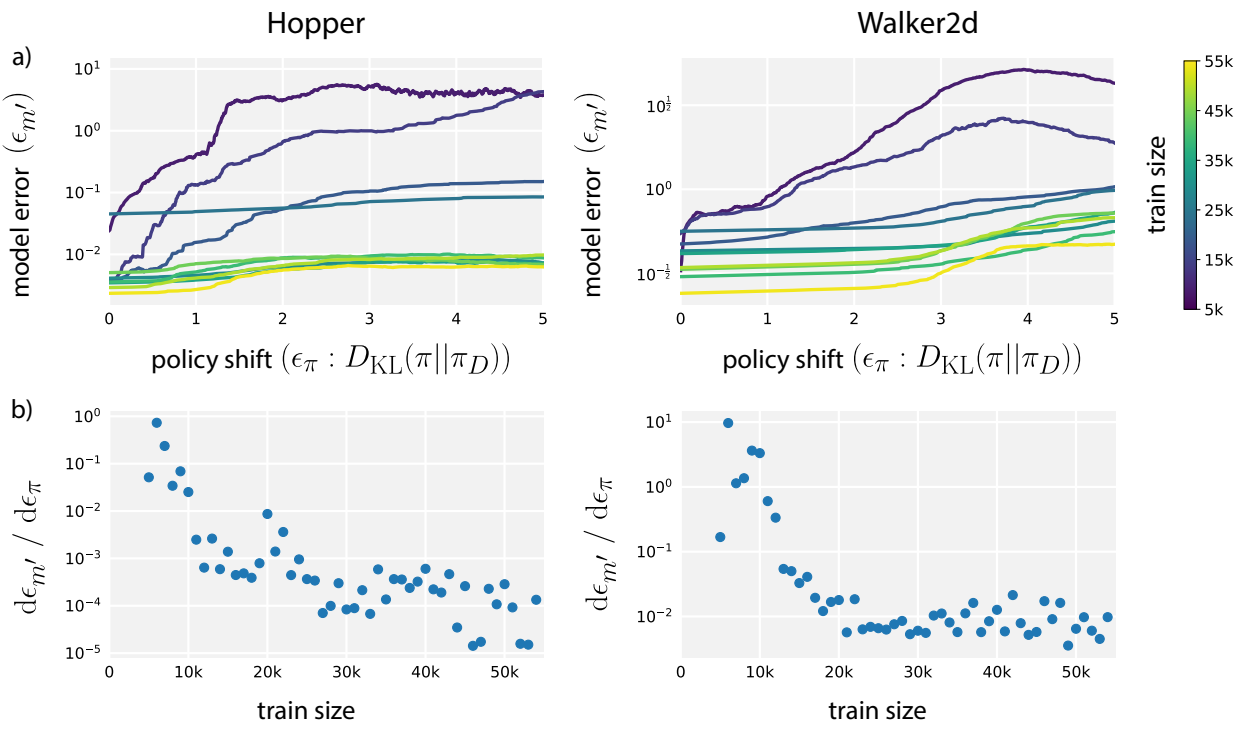


Figure 5.1: (a) We train a predictive model on the state distribution of π_D and evaluate it on policies π of varying KL-divergence from π_D without retraining. The color of each curve denotes the amount of data from π_D used to train the model corresponding to that curve. The offsets of the curves depict the expected trend of increasing training data leading to decreasing model error on the training distribution. However, we also see a decreasing influence of state distribution shift on model error with increasing training data, signifying that the model is generalizing better. (b) We measure the local change in model error versus KL-divergence of the policies at $\epsilon_\pi = 0$ as a proxy to model generalization.

but discards intermediate states generated b the model. For completeness, we also include the more realistic scenario where all model-generated states are included in Appendix B.1, although the general conclusions remain the same. Under such a branching scheme, the returns can be bounded as follows:

Theorem 5.2.2. *Given returns $J^{\text{branch}}(\pi)$ from the k-branched rollout method,*

$$J(\pi) \geq J^{\text{branch}}(\pi) - 2R_{\max}\left[\frac{\gamma^{k+1}\epsilon_\pi}{(1-\gamma)^2} + \frac{\gamma^k + 2}{(1-\gamma)}\epsilon_\pi + \frac{k}{1-\gamma}(\epsilon_m + 2\epsilon_\pi)\right]. \tag{5.2}$$

Proof. See Section B.1, Theorem B.1.3. □

Model Generalization in Practice

Theorem 5.2.2 would be most useful for guiding algorithm design if it could be used to determine an optimal model rollout length k. While this bound does include two competing factors, one exponentially decreasing in k and another scaling linearly with k, the values of the associated constants prevent an actual tradeoff; taken literally, this lower bound is maximized when $k = 0$, corresponding to not using the model at all. One limitation of the analysis is pessimistic scaling of model error ϵ_m with respect to policy shift ϵ_π, as we do not make any assumptions about the generalization capacity or smoothness properties of the model [Asadi et al., 2018].

To better determine how well we can expect our model to generalize in practice, we empirically measure how the model error under new policies increases with policy change ϵ_π. We train a model on the state distribution of a data-collecting policy π_D and then continue policy optimization while measuring the model's loss on all intermediate policies π during this optimization. Figure 5.1a shows that, as expected, the model error increases with the divergence between the current policy π and the data-collecting policy π_D. However, the rate of this increase depends on the amount of data collected by π_D. We plot the local change in model error over policy change, $\frac{\mathrm{d}\epsilon_{m'}}{\mathrm{d}\epsilon_\pi}$, in Figure 5.1b. The decreasing dependence on policy shift shows that not only do models trained with more data perform better on their training distribution, but they also generalize better to nearby distributions.

The clear trend in model error growth rate suggests a way to modify the pessimistic bounds. In the previous analysis, we assumed access to only model error ϵ_m on the distribution of the most recent data-collecting policy π_D and approximated the error on the current distribution as $\epsilon_m + 2\epsilon_\pi$. If we can instead approximate the model error on the distribution of the current policy π, which we denote as $\epsilon_{m'}$, we may use this directly. For example, approximating $\epsilon_{m'}$ with a linear function of the policy divergence yields:

$$\hat{\epsilon}_{m'}(\epsilon_\pi) \approx \epsilon_m + \epsilon_\pi \frac{\mathrm{d}\epsilon_{m'}}{\mathrm{d}\epsilon_\pi}$$

where $\frac{\mathrm{d}\epsilon_{m'}}{\mathrm{d}\epsilon_\pi}$ is empirically estimated as in Figure 5.1. Equipped with an approximation of $\epsilon_{m'}$, the model's error on the distribution of the current policy π, we arrive at the following bound:

Theorem 5.2.3. *Under the k-branched rollout method, using model error under the updated policy* $\epsilon_{m'} \geq \max_t \mathbb{E}_{s\sim\pi_{D,t}}[D_{TV}(p(s'|s,a)||\hat{p}(s'|s,a))]$*, we have*

$$J(\pi) \geq J^{\text{branch}}(\pi) - 2R_{\max}\left[\frac{\gamma^{k+1}\epsilon_\pi}{(1-\gamma)^2} + \frac{\gamma^k \epsilon_\pi}{(1-\gamma)} + \frac{k}{1-\gamma}(\epsilon_{m'})\right]. \tag{5.3}$$

Proof. See Section B.1, Theorem B.1.2. □

While this bound appears similar to Theorem 5.2.2, the important difference is that this version actually motivates model usage. More specifically,

$$k^* = \operatorname*{argmin}_k \left[\frac{\gamma^{k+1}\epsilon_\pi}{(1-\gamma)^2} + \frac{\gamma^k \epsilon_\pi}{(1-\gamma)} + \frac{k}{1-\gamma}(\epsilon_{m'}) \right] > 0$$

for sufficiently low $\epsilon_{m'}$. While this insight does not immediately suggest an algorithm design by itself, we can build on this idea to develop a method that makes limited use of truncated, but nonzero-length, model rollouts.

5.3 Model-Based Policy Optimization with Deep Reinforcement Learning

We now present a practical model-based reinforcement learning algorithm based on the derivation in the previous section. Instantiating Algorithm 2 amounts to specifying three design decisions: (1) the parametrization of the model p_θ, (2) how the policy π is optimized given model samples, and (3) how to query the model for samples for policy optimization.

Predictive model. In our work, we use a bootstrapped ensemble of dynamics models $\{p_\theta^1, ..., p_\theta^B\}$. Each member of the ensemble is a probabilistic neural network whose outputs parametrize a Gaussian distribution with diagonal covariance: $p_\theta^i(s_{t+1}, r|s_t, a_t) = \mathcal{N}(\mu_\theta^i(s_t, a_t), \Sigma_\theta^i(s_t, a_t)))$. Individual probabilistic models capture aleatoric uncertainty, or the noise in the outputs with respect to the inputs. The bootstrapping procedure accounts for epistemic uncertainty, or uncertainty in the model parameters, which is crucial in regions when data is scarce and the model can by exploited by policy optimization. Chua et al. [2018] demonstrate that a proper handling of both of these uncertainties allows for asymptotically competitive model-based learning. To generate a prediction from the ensemble, we simply select a model uniformly at random, allowing for different transitions along a single model rollout to be sampled from different dynamics models.

Policy optimization. We adopt soft-actor critic (SAC) [Haarnoja et al., 2018] as our policy optimization algorithm. SAC alternates between a policy evaluation step, which estimates $Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t=0}^\infty \gamma^t r(s_t, a_t)|s_0 = s, a_0 = a\right]$ using the Bellman backup operator, and a policy improvement step, which trains an actor π by minimizing the expected KL-divergence $J^{\text{SAC}}(\phi, \mathcal{D}) = \mathbb{E}_{s_t \sim \mathcal{D}}[D_{KL}(\pi || \exp\{Q^\pi - V^\pi\})]$.

Model usage. Many recent model-based algorithms have focused on the setting in which model rollouts begin from the initial state distribution [Kurutach et al., 2018, Clavera et al., 2018]. While this may be a more faithful interpretation of Algorithm 2, as it is optimizing a policy purely under the state distribution of the model, this approach entangles the model rollout length with the task horizon. Because compounding model errors make extended

Algorithm 3 Model-Based Policy Optimization with Deep Reinforcement Learning

Initialize policy π_ϕ, predictive model p_θ, environment dataset $\mathcal{D}_{\text{env}}$, model dataset $\mathcal{D}_{\text{model}}$
for N epochs **do**
  Train model p_θ on $\mathcal{D}_{\text{env}}$ via maximum likelihood
  for E steps **do**
    Take action in environment according to π_ϕ; add to $\mathcal{D}_{\text{env}}$
    for M model rollouts **do**
      Sample s_t uniformly from $\mathcal{D}_{\text{env}}$
      Perform k-step model rollout starting from s_t using policy π_ϕ; add to $\mathcal{D}_{\text{model}}$
    end for
    for G gradient updates **do**
      Update policy parameters on model data: $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J^{\text{SAC}}(\phi, \mathcal{D}_{\text{model}})$
    end for
  end for
end for

rollouts difficult, these works evaluate on truncated versions of benchmarks. The branching strategy described in Section 5.2, in which model rollouts begin from the state distribution of a different policy under the true environment dynamics, effectively relieves this limitation. In practice, branching replaces few long rollouts from the initial state distribution with many short rollouts starting from replay buffer states.

A practical implementation of MBPO is described in Algorithm 3.[1] The primary differences from the general formulation in Algorithm 2 are k-length rollouts from replay buffer states in the place of optimization under the model's state distribution and a fixed number of policy update steps in the place of an intractable argmax. Even when the horizon length k is short, we can perform many such short rollouts to yield a large set of model samples for policy optimization. This large set allows us to take many more policy gradient steps per environment sample (between 20 and 40) than is typically stable in model-free algorithms. A full listing of the hyperparameters included in Algorithm 3 for all evaluation environments is given in Section B.3.

5.4 Experiments

Our experimental evaluation aims to study two primary questions: (1) How well does MBPO perform on benchmark reinforcement learning tasks, compared to state-of-the-art model-based and model-free algorithms? (2) What conclusions can we draw about appropriate model usage?

[1] When SAC is used as the policy optimization algorithm, we must also perform gradient updates on the parameters of the Q-functions, but we omit these updates for clarity.

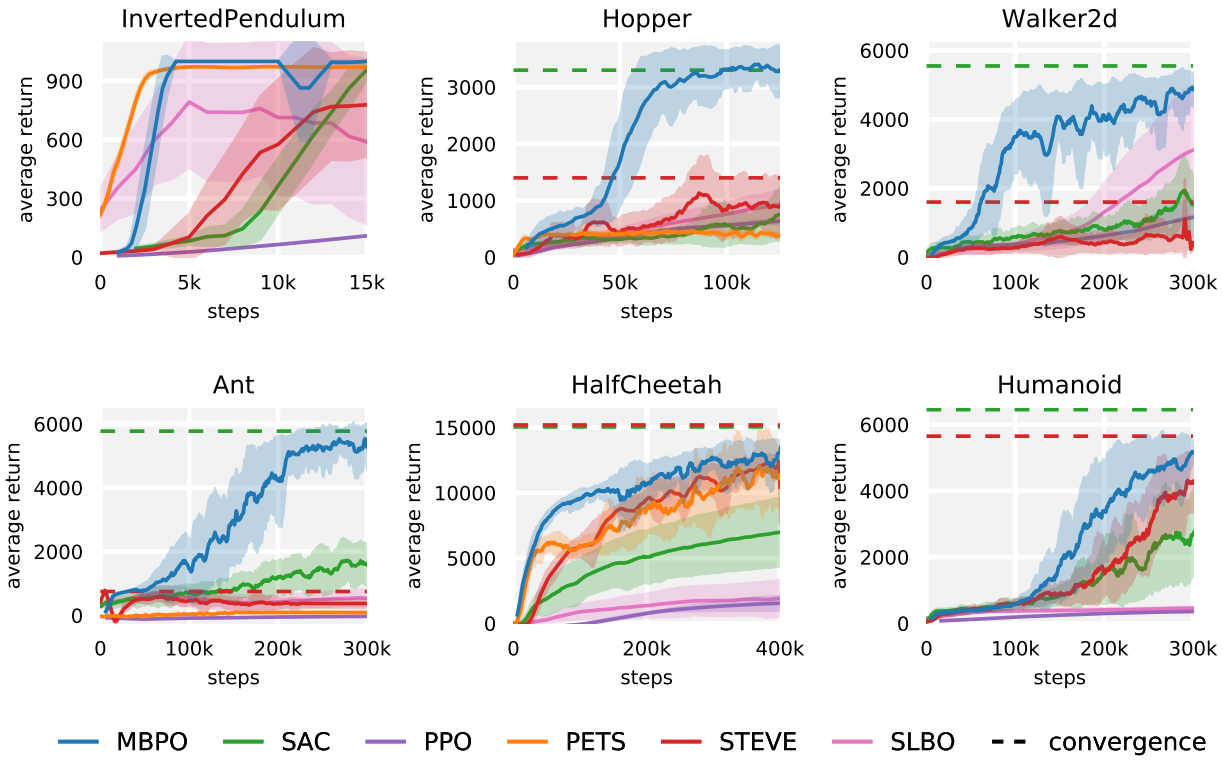


Figure 5.2: Training curves of MBPO and five baselines on continuous control benchmarks. Solid curves depict the mean of five trials and shaded regions correspond to standard deviation among trials. MBPO has asymptotic performance similar to the best model-free algorithms while being faster than the model-based baselines. For example, MBPO's performance on the Ant task at 300 thousand steps matches that of SAC at 3 million steps. We evaluated all algorithms on the standard 1000-step versions of the benchmarks.

Comparative Evaluation

In our comparisons, we aim to understand both how well our method compares to state-of-the-art model-based and model-free methods and how our design choices affect performance. We compare to two state-of-the-art model-free methods, SAC [Haarnoja et al., 2018] and PPO [Schulman et al., 2017], both to establish a baseline and, in the case of SAC, measure the benefit of incorporating a model, as our model-based method uses SAC for policy learning as well. For model-based methods, we compare to PETS [Chua et al., 2018], which does not perform explicit policy learning, but directly uses the model for planning; STEVE [Buckman et al., 2018], which also uses short-horizon model-based rollouts, but incorporates data from these rollouts into value estimation rather than policy learning; and SLBO [Luo et al., 2019], a model-based algorithm with performance guarantees that performs model rollouts from the

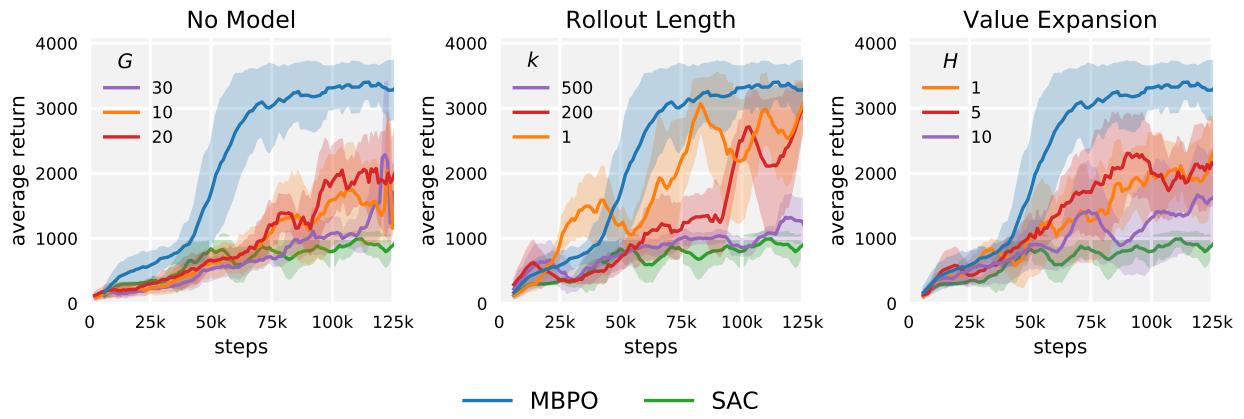


Figure 5.3: **No model:** SAC run without model data but with the same range of gradient updates per environment step (G) as MBPO on the Hopper task. **Rollout length:** While we find that increasing rollout length k over time yields the best performance for MBPO (Appendix B.3), single-step rollouts provide a baseline that is difficult to beat. **Value expansion:** We implement H-step model value expansion from Feinberg et al. [2018] on top of SAC for a more informative comparison. We also find in the context of value expansion that single-step model rollouts are surprisingly competitive.

initial state distribution. These comparisons represent the state-of-the-art in both model-free and model-based reinforcement learning.

We evaluate MBPO and these baselines on a set of MuJoCo continuous control tasks [Todorov et al., 2012] commonly used to evaluate model-free algorithms. Note that some recent works in model-based reinforcement learning have used modified versions of these benchmarks, where the task horizon is chosen to be shorter so as to simplify the modeling problem [Kurutach et al., 2018, Clavera et al., 2018]. We use the standard full-length version of these tasks. MBPO also does not assume access to privileged information in the form of fully observable states or the reward function for offline evaluation.

Figure 5.2 shows the learning curves for all methods, along with asymptotic performance of algorithms which do not converge in the region shown. These results show that MBPO learns substantially faster, an order of magnitude faster on some tasks, than prior model-free methods, while attaining comparable final performance. For example, MBPO's performance on the Ant task at 300 thousand steps is the same as that of SAC at 3 million steps. On Hopper and Walker2d, MBPO requires the equivalent of 14 and 40 minutes, respectively, of simulation time if the simulator were running in real time. More crucially, MBPO learns on some of the higher-dimensional tasks, such as Ant, which pose problems for purely model-based approaches such as PETS.

Design Evaluation

We next make ablations and modifications to our method to better understand why MBPO outperforms prior approaches. Results for the following experiments are shown in Figure 5.3.

No model. The ratio between the number of gradient updates and environment samples, G, is much higher in MBPO than in comparable model-free algorithms because the model-generated data reduces the risk of overfitting. We run standard SAC with similarly high ratios, but without model data, to ensure that the model is actually helpful. While increasing the number of gradient updates per sample taken in SAC does marginally speed up learning, we cannot match the sample-efficiency of our method without using the model. For hyperparameter settings of MBPO, see Section B.3.

Rollout horizon. While the best-performing rollout length schedule on the Hopper task linearly increases from $k = 1$ to 15 (Section B.3), we find that fixing the rollout length at 1 for the duration of training retains much of the benefit of our model-based method. We also find that our model is accurate enough for 200-step rollouts, although this performs worse than shorter values when used for policy optimization. 500-step rollouts are too inaccurate for effective learning. While more precise fine-tuning is always possible, augmenting policy training data with single-step model rollouts provides a baseline that is surprisingly difficult to beat and outperforms recent methods which perform longer rollouts from the initial state distribution. This result agrees with our theoretical analysis which prescribes short model-based rollouts to mitigate compounding modeling errors.

Value expansion. An alternative to using model rollouts for direct training of a policy is to improve the quality of target values of samples collected from the real environment. This technique is used in model-based value expansion (MVE) [Feinberg et al., 2018] and STEVE [Buckman et al., 2018]. While MBPO outperforms both of these approaches, there are other confounding factors making a head-to-head comparison difficult, such as the choice of policy learning algorithm. To better determine the relationship between training on model-generated data and using model predictions to improve target values, we augment SAC with the H-step Q-target objective:

$$\frac{1}{H}\sum_{t=-1}^{H-1}(Q(\hat{s}_t, \hat{a}_t - (\sum_{k=t}^{H-1}\gamma^{k-t}\hat{r}_k + \gamma^H Q(\hat{s}_H, \hat{a}_H))^2$$

in which $\hat{s}_t$ and $\hat{r}_t$ are model predictions and $\hat{a}_t \sim \pi(a_t|\hat{s}_t)$. We refer the reader to Feinberg et al. [2018] for further discussion of this approach. We verify that SAC also benefits from improved target values, and similar to our conclusions from MBPO, single-step model rollouts ($H = 1$) provide a surprisingly effective baseline. While model-generated data augmentation and value expansion are in principle complementary approaches, preliminary experiments did not show improvements to MBPO by using improved target value estimates.

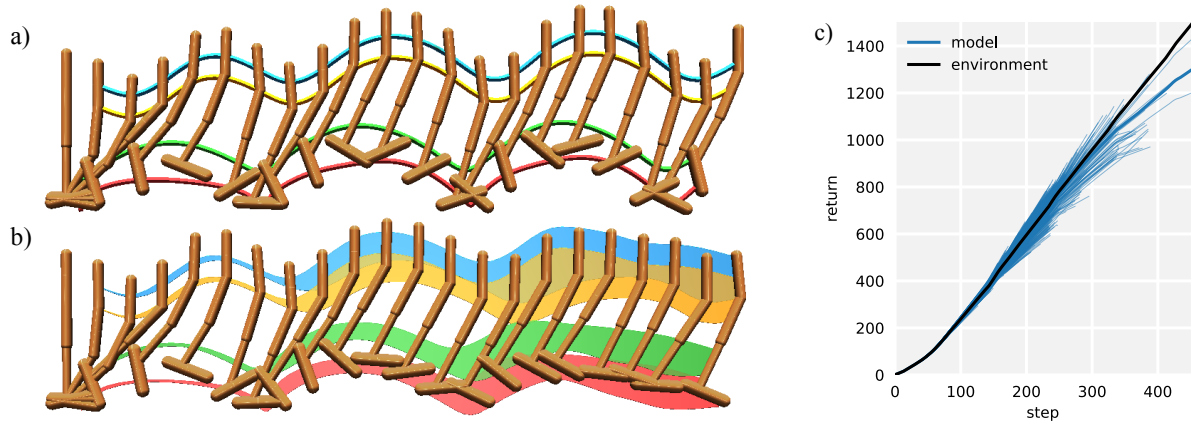


Figure 5.4: a) A 450-step hopping sequence performed in the real environment, with the trajectory of the body's joints traced through space. b) The same action sequence rolled out under the model 1000 times, with shaded regions corresponding to one standard deviation away from the mean prediction. The growing uncertainty and deterioration of a recognizable sinusoidal motion underscore accumulation of model errors. c) Cumulative returns of the same policy under the model and actual environment dynamics reveal that the policy is not exploiting the learned model. Thin blue lines reflect individual model rollouts and the thick blue line is their mean.

Model exploitation. We analyze the problem of "model exploitation," which a number of recent works have raised as a primary challenge in model-based reinforcement learning [Rajeswaran et al., 2017, Clavera et al., 2018, Kurutach et al., 2018]. We plot empirical returns of a trained policy on the Hopper task under both the real environment and the model in Figure 5.4 (c) and find, surprisingly, that they are highly correlated, indicating that a policy trained on model-predicted transitions may not exploit the model at all if the rollouts are sufficiently short. This is likely because short rollouts are more likely to reflect the real dynamics (Figure 5.4 a-b), reducing the opportunities for policies to rely on inaccuracies of model predictions. While the models for other environments are not necessarily as accurate as that for Hopper, we find across the board that model returns tend to *underestimate* real environment returns in MBPO.

5.5 Discussion

We have investigated the role of model usage in policy optimization procedures through both a theoretical and empirical lens. We have shown that, while it is possible to formulate model-based reinforcement learning algorithms with monotonic improvement guarantees, such an analysis cannot necessarily be used to motivate using a model in the first place.

However, an empirical study of model generalization shows that predictive models can indeed perform well outside of their training distribution. Incorporating a linear approximation of model generalization into the analysis gives rise to a more reasonable tradeoff that does in fact justify using the model for truncated rollouts. The algorithm stemming from this insight, MBPO, has asymptotic performance rivaling the best model-free algorithms, learns substantially faster than prior model-free or model-based methods, and scales to long horizon tasks that often cause model-based methods to fail. We experimentally investigate the tradeoffs associated with our design decisions, and find that model rollouts as short as a single step can provide pronounced benefits to policy optimization procedures.

Chapter 6

Offline Model-Based Optimization

Up to this chapter, we have primarily concerned ourselves with offline learning in the context of sequential decision making within the offline reinforcement learning framework. However, many real-world decision making problems often lack temporal structure, making it difficult to apply reinforcement learning. Examples of such problems include the design of materials [Mansouri Tehrani et al., 2018], proteins [Brookes et al., 2019, Kumar and Levine, 2020], neural network architectures [Zoph and Le, 2017], or vehicles [Hoburg and Abbeel, 2014]. These problems are structured as a more traditional optimization problem, where we must select a single input (such as the architecture of a neural network) that maximizes some objective function. We frame offline decision making in this more general setting as the *offline model-based optimization* (MBO) problem, where a static dataset of input-output pairs is available but function queries are not allowed.

A straightforward method to solving offline MBO problems would be to estimate a proxy of the ground truth function $\hat{f}_\theta$ using supervised learning, and to optimize the input $\boldsymbol{x}$ with respect to this proxy. However, this approach is brittle and prone to failure, because the model-fitting process often has little control over the values of the proxy function on inputs outside of the training set. An algorithm that directly optimizes $\hat{f}_\theta$ could easily exploit the proxy to produce adversarial inputs that nevertheless are scored highly under $\hat{f}_\theta$ [Kumar and Levine, 2020, Fannjiang and Listgarten, 2020].

In order to counteract the effects of model exploitation, we propose to use the normalized maximum likelihood framework (NML) [Barron et al., 1998]. The NML estimator produces the distribution closest to the MLE assuming an *adversarial* output label, and has been shown to be effective for resisting adversarial attacks [Bibas et al., 2019]. Moreover, NML provides a principled approach to generating uncertainty estimates which allows it to reason about out-of-distribution queries. However, because NML is typically intractable except for a handful of special cases [Roos et al., 2008], we show in this work how we can circumvent intractability issues with NML in order to construct a reliable and robust method for MBO. Because of its general formulation, the NML distribution provides a flexible approach to constructing conservative and robust estimators using high-dimensional models such as neural networks.

In this chapter we develop an offline MBO algorithm that utilizes a novel approximation to the NML distribution to obtain an uncertainty-aware forward model for optimization, which we call NEMO (**N**ormalized maximum likelihood **E**stimation for **M**odel-based **O**ptimization). The basic premise of NEMO is to construct a conditional NML distribution that maps inputs to a distribution over outputs. While constructing the NML distribution is intractable in general, we discuss novel methods to amortize the computational cost of NML, which allows us the scale our method to practical problems with high dimensional inputs using neural networks. A separate optimization algorithm can then be used to optimize over the output to any desired confidence level. Theoretically, we provide insight into why NML is useful for the MBO setting by showing a regret bound for modeling the ground truth function. Empirically, we evaluate our method on a selection of tasks from the Design Benchmark [Trabucco et al., 2021], where we show that our method performs competitively with state-of-the-art baselines. Additionally, we provide a qualitative analysis of the uncertainty estimates produced by NEMO, showing that it provides reasonable uncertainty estimates, while commonly used methods such as ensembles can produce erroneous estimates that are both confident and wrong in low-data regimes.

6.1 Relation to Prior Work

This chapter presents methods for model-based optimization algorithms based on building uncertainty aware models using normalized maximum likelihood. In contrast to prior works on MBO such as MINs [Kumar and Levine, 2020], CbAS [Brookes et al., 2019], or autofocused oracles [Fannjiang and Listgarten, 2020], our approach explicitly reasons about uncertainty to construct robust models of the problem.

A related line of work is derivative-free optimization, which is typically used in settings where only function evaluations are available. This includes methods such as REINFORCE [Williams, 1992] and reward-weighted regression [Peters and Schaal, 2007] in reinforcement learning, the cross-entropy method [Rubinstein, 1999], latent variable models [Garnelo et al., 2018, Kim et al., 2018], and Bayesian optimization [Snoek et al., 2012, Shahriari et al., 2015]. Of these approaches, Bayesian optimization is the most often used when function evaluations are expensive and limited. However, all of the aforementioned methods focus on the active or online setting, whereas in this work, we are concerned with the offline setting where additional function evaluations are not available.

Our method is utilizes normalized maximum likelihood estimation, which is an information-theoretic framework based on the minimum description length principle [Rissanen, 1978]. While the standard NML formulation is purely generative, the conditional or predictive NML setting can be used Rissanen and Roos [2007], Fogel and Feder [2018] for supervised learning and prediction problems. Bibas et al. [2019] apply this framework for prediction using deep neural networks, but require an expensive finetuning process for every input. The goal of our work is to provide a scalable and tractable method to approximate the CNML distribution, and we apply this framework to offline optimization problems.

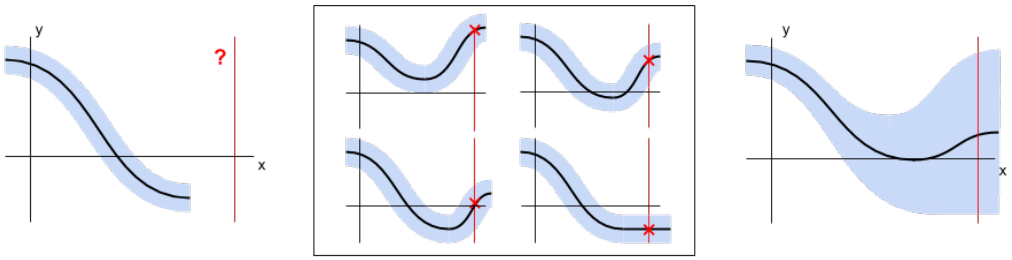


Figure 6.1: An illustrative example of the construction of the CNML distribution. **Left** We wish to estimate the $p(y|x)$ at some query point x, marked by the vertical red line. **Middle** We compute the MLE assuming we knew the true label y, for every possible value of y. **Right** Finally, we normalize the predictions across all MLE models to produce p_{NML}. The final prediction will likely exhibit large amounts of uncertainty on queries x far from the dataset, because it is easier for the individual MLE estimates to overfit to these outliers.

Like CNML, **conformal prediction** [Shafer and Vovk, 2008] is concerned with predicting the value of a query point $\hat{y}_{t+1}$ given a prior dataset, and provides per-instance confidence intervals, based on how consistent the new input is with the rest of the dataset. Our work instead relies on the NML framework, where the NML regret serves a similar purpose for measuring how close a new query point is to existing, known data.

6.2 Normalized Maximum Likelihood

Recall from Section 2.2 that the goal of the offline model-based optimization problem is to solve an optimization problem:

$$\boldsymbol{x}^* = \text{argmax}_{\boldsymbol{x}} \mathbb{E}_{y \sim f(y|\boldsymbol{x})}[y]. \tag{6.1}$$

We can solve this problem by first constructing an estimate of the objective function $f(y|\boldsymbol{x})$, and then optimizing it's expectation with respect to $\boldsymbol{x}$ using a method such as gradient ascent.

In order to produce a conditional distribution we can use for estimating the ground truth function that is robust to out-of-distribution inputs, we leverage the conditional or predictive NML (CNML) framework [Rissanen and Roos, 2007, Fogel and Feder, 2018, Bibas et al., 2019]. Intuitively, the CNML distribution is the distribution closest to the MLE assuming the test label y is chosen adversarially. This is useful for the MBO setting since we do not know the ground truth value y at points we are querying during optimization, and the CNML distribution gives us conservative estimates that help mitigate model exploitation (see Fig. 6.1). Formally, the CNML estimator is the minimax solution to a notion of regret, called the *individual regret* defined as $\text{Regret}_{\text{ind}}(h, y) = \log p(y|\boldsymbol{x}, \hat{\theta}_{\mathcal{D} \cup (\boldsymbol{x}, y)}) - \log h(y|\boldsymbol{x})$, and $p_{\text{NML}}(y|\boldsymbol{x}) = \arg\min_h \max_{y'} \text{Regret}_{\text{ind}}(h, y')$ [Fogel and Feder, 2018]. The notation $\mathcal{D} \cup (\boldsymbol{x}, y)$

refers to an augmented dataset by appending a query point and label $(\boldsymbol{x}, y)$, to a fixed offline dataset $\mathcal{D}$, and $\hat{\theta}_{\mathcal{D}\cup(\boldsymbol{x},y)}$ denotes the MLE estimate for this augmented dataset. The query point $(\boldsymbol{x}, y)$ serves to represent the test point we are interested in modeling. The solution to the minimax problem can be expressed as [Fogel and Feder, 2018]:

$$p_{\text{NML}}(y|\boldsymbol{x}) = \frac{p(y|\boldsymbol{x}, \hat{\theta}_{\mathcal{D}\cup(\boldsymbol{x},y)})}{\int_{y'} p(y'|\boldsymbol{x}, \hat{\theta}_{\mathcal{D}\cup(\boldsymbol{x},y')})dy'}, \tag{6.2}$$

where $\hat{\theta}_{\mathcal{D}\cup(\boldsymbol{x},y)} = \arg\max_\theta \frac{1}{N+1} \sum_{(\boldsymbol{x},y)\in\mathcal{D}\cup(\boldsymbol{x},y)} \log p(y|\boldsymbol{x}, \theta)$ is the maximum likelihood estimate for p using the dataset $\mathcal{D}$ augmented with $(\boldsymbol{x}, y)$.

The NML family of estimators has connections to Bayesian methods, and has shown to be asymptotically equivalent to Bayesian inference under the uninformative Jeffreys prior [Rissanen, 1996]. NML and Bayesian modeling both suffer from intractability, albeit for different reasons. Bayesian modeling is generally intractable outside of special choices of the prior and model class Θ where conjugacy can be exploited. On the other hand, NML is intractable because the denominator requires integrating and training a MLE estimator for every possible y.

6.3 NEMO: Normalized Maximum Likelihood Estimation for Model-Based Optimization

We now present NEMO, our proposed algorithm for high-dimensional offline MBO. NEMO is a tractable scheme for estimating and optimizing the estimated expected value of the target function under the CNML distribution. As mentioned above, the CNML estimator (Eqn. 6.2) is difficult to compute directly, because it requires **a)** obtaining the MLE for each value of y, and **b)** integrating these estimates over y. In this section, we describe how to address these two issues, using amortization and quantization. We outline the high-level pseudocode in Algorithm 4, and presented a more detailed implementation in Appendix C.1.

An Iterative Algorithm for Model-Based Optimization

We first describe the overall structure of our algorithm, which addresses issue **a)**, the intractability of computing an MLE estimate for every point we wish to query. In this section we assume that the domain of y is discrete, and describe in the following section how we utilize a quantization scheme to approximate a continuous y with a discrete one.

Recall from Section 2.2 that we wish to construct a proxy for the ground truth, which we will then optimize with gradient ascent. The most straightforward way to integrate NML and MBO would be to fully compute the NML distribution described by Eqn. 6.2 at each optimization step, conditioned on the current optimization iterate $\boldsymbol{x}_t$. This would produce a conditional distribution $p_{\text{NML}}(y|\boldsymbol{x})$ over output values, and we can optimize $\boldsymbol{x}_t$ with respect to some function of this distribution, such as the mean. While this method is tractable to

Algorithm 4 NEMO: Normalized Maximum Likelihood for Model-Based Optimization

Input Model class $\{f_\theta : \theta \in \Theta\}$, Dataset $\mathcal{D} = (\boldsymbol{x}_{1:N}, y_{1:N})$, number of bins K, evaluation function $g(y)$, learning rates α_θ, α_x.
Initialize K models $\theta_0^{1:K}$, optimization iterate $\boldsymbol{x}_0$
Quantize $y_{1:N}$ into K bins, denoted as $\lfloor \mathcal{Y} \rfloor = \{\lfloor y_1 \rfloor, \cdots \lfloor y_k \rfloor\}$.
for iteration t in $1 \dots T$ **do**
  for k in $1 \dots K$ **do**
    construct augmented dataset: $\mathcal{D}' \leftarrow \mathcal{D} \cup (\boldsymbol{x}_t, \lfloor y_k \rfloor)$.
    update model: $\theta_{t+1}^k \leftarrow \theta_t^k + \alpha_\theta \nabla_{\theta_t^k} \text{LogLikelihood}(\theta_t^k, \mathcal{D}')$
  end for
  estimate CNML distribution: $\hat{p}_{\text{NML}}(y|x_t) \propto p(y|\boldsymbol{x}_t, \theta_t^y) / \sum_k p(\lfloor y_k \rfloor | \boldsymbol{x}_t, \theta_t^k)$
  Update $\boldsymbol{x}$: $\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t + \alpha_x \nabla_x E_{y \sim \hat{p}_{\text{NML}}(y|\boldsymbol{x})}[g(y)]$
end for

implement for small problems, it will still be significantly slower than standard optimization methods, because it requires finding the MLE estimate for every y value per iteration of the algorithm. This can easily become prohibitively expensive when using large neural networks on high-dimensional problems.

To remedy this problem, we propose to amortize the learning process by incrementally learning the NML distribution while optimizing the iterate $\boldsymbol{x}_t$. In order to do this, we maintain one model per value of y, $\hat{\theta}_k$, each corresponding to one element in the normalizing constant of the NML distribution. During each step of the algorithm, we sample a batch of datapoints, and train each model by appending the current iterate $\boldsymbol{x}_t$ as well as a label $y_{t,k}$ to the batch with a weight w (which is typically set to $w = 1/N$). We then perform a number of gradient step on each model, and use the resulting models to form an estimate of the NML distribution $p_{\text{NML}}(y_t|\boldsymbol{x}_t)$. We then compute a score from the NML distribution, such as the mean, and perform one step of gradient ascent on $\boldsymbol{x}_t$.

While the incremental algorithm produces only an approximation to the true NML distribution, it brings the computational complexity of the resulting algorithm to just $O(K)$ gradient steps per iteration, rather than solving entire inner-loop optimization problems. This brings the computational cost to be comparable to other baseline methods we evaluated for MBO.

Quantization and Architecture

The next challenge towards developing a practical NML method is addressing issue **b)**, the intractability of integrating over a continuous y. We propose to tackle this issue with quantization and a specialized architecture for modeling the ground truth function.

Quantization. One situation in which the denominator is tractable is when the domain of y is discrete and finite. In such a scenario, we could train K models, where K is the size

of the domain, and directly sum over the likelihood estimates to compute the normalizing factor.

In order to turn the NML distribution into a tractable, discrete problem, we quantize all outputs in the dataset by flooring each y value to the nearest bin $\lfloor y_k \rfloor$, with the size of each interval defined as $B = (y_{\max} - y_{\min})/K$. While quantization has potential to induce additional rounding errors to the optimization process, we find in our experiments in Section 6.5 that using moderate value such as $K = 20$ or $K = 40$ provides both a reasonably accurate solution while not being excessively demanding on computation.

This scheme of quantization can be interpreted as a rectangular quadrature method, where the integral over y is approximated as:

$$\int_y p(y|x, \hat{\theta}_{\mathcal{D}\cup(\boldsymbol{x},y)})dy \approx B \sum_{k=1}^{K} p(\lfloor y_k \rfloor | x, \hat{\theta}_{\mathcal{D}\cup(\boldsymbol{x},\lfloor y_k \rfloor)})$$

Discretized logistic architecture. Quantization introduces unique difficulties into the optimization process for MBO. In particular, quantization results in flat regions in the optimization landscape, making using gradient-based algorithms to optimize both inputs $\boldsymbol{x}$ and models $p(y|x, \theta)$ challenging. In order to alleviate these issues, we propose to model the output using a discretized logistic architecture, depicted in Fig. 6.2. The discretized logistic architecture transforms and input x into the mean parameter of a logistic distribution $\mu(\boldsymbol{x})$, and outputs one minus the CDF of a logistic distribution queried at regular intervals of $1/K$ (recall that the CDF of a logistic distribution is itself the logistic or sigmoid function). Therefore, the final output is a vector o of length K, where element k is equal to $\sigma(\mu(\boldsymbol{x}) + k/K)$. We note that similar architectures have been used elsewhere, such as for modeling the output over pixel intensity values in images [Salimans et al., 2017].

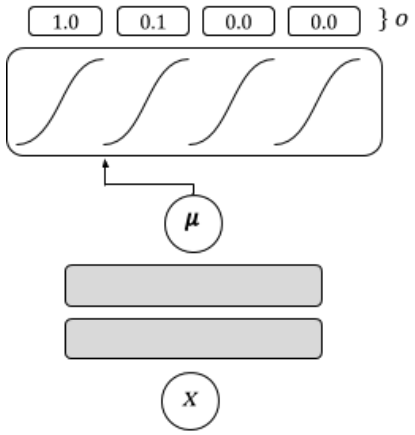


Figure 6.2: A diagram of the discretized logistic architecture. The mean μ is denoted by an arrow, which is then passed through offset sigmoid functions to produce o.

We train this model by first encoding a label y as a vector y_{disc}, where $y_{disc}[k \leq \text{bin}(y)] = 1$ and elements $y_{disc}[k > \text{bin}(y)] = 0$. $\text{bin}(y)$ denotes the index of the quantization bin that y falls under. The model is then trained using a standard binary cross entropy loss, applied per-element across the entire output vector. Because the output represents the one minus the CDF, the expected value of the discretized logistic architecture can be computed as $y_{\text{mean}}(\boldsymbol{x}) = \mathbb{E}_{y\sim p(y|x)}[g(y)] = \sum_k [g(k) - g(k-1)]o[k]$. If we assume that g normalizes all output values to $[0, 1]$ in uniform bins after quantization, the mean can easily be computed as a sum over the entire output vector, $y_{\text{mean}} = \frac{1}{K}\sum_k o[k]$

Optimization The benefit of using such an architecture is that when optimizing for $\boldsymbol{x}$, rather than optimizing the predicted output directly, we can compute gradients with respect

to the logistic parameter μ. Because μ is a single scalar output of a feedforward network, it is less susceptible to flat gradients introduced by the quantization procedure. Optimizing with respect to μ is sensible as it shares the same global optimum as y_{mean}, and gradients with respect to μ and y_{mean} share a positive angle, as shown by the following theorem:

Proposition 6.3.1 (Discretized Logistic Gradients)**.** *Let $\mu(\boldsymbol{x})$ denote the mean of the discretized logistic architecture for input $\boldsymbol{x}$, and $y_{mean}(\boldsymbol{x})$ denote the predicted mean. Then,*

1. *If $x \in \arg\max_{\boldsymbol{x}} \mu(\boldsymbol{x})$, then $\boldsymbol{x} \in \arg\max_{\boldsymbol{x}} y_{mean}(\boldsymbol{x})$.*
2. *For any $\boldsymbol{x}$, $\langle \nabla_{\boldsymbol{x}} \mu(\boldsymbol{x}), \nabla_{\boldsymbol{x}} y_{mean}(\boldsymbol{x}) \rangle \geq 0$.*

Proof. This statement directly from monotonicity. Because y_{mean} is a sum of monotonic functions in μ, y_{mean} must also be a monotonic function of μ. This implies that the global maxima are the same, and that gradients must point in the same direction since $\langle \nabla_{\boldsymbol{x}} \mu, \nabla_{\boldsymbol{x}} y_{\text{mean}} \rangle = \left\langle \nabla_{\boldsymbol{x}} \mu, \frac{dy_{\text{mean}}}{d\mu} \nabla_{\boldsymbol{x}} \mu \right\rangle = \frac{dy_{\text{mean}}}{d\mu} ||\nabla_{\boldsymbol{x}} \mu||_2^2 \geq 0$. □

6.4 Theoretical Results

We now highlight some theoretical motivation for using CNML in the MBO setting, and show that estimating the true function with the CNML distribution is close to an expert even if the test label is chosen adversarially, which makes it difficult for an optimizer to exploit the model. As discussed earlier, the CNML distribution minimizes a notion of regret based on the log-loss with respect to an adversarial test distribution. This construction leads the CNML distribution to be very conservative for out-of-distribution inputs. However, the notion of regret in conventional CNML does not easily translate into a statement about the outputs of the function we are optimizing. In order to reconcile these differences, we introduce a new notion of regret, the *functional regret*, which measures the difference between the output estimated under some model against an expert within some function class Θ.

Definition 6.4.1 (Functional Regret)**.** *Let $q(y|\boldsymbol{x})$ be an estimated conditional distribution, $\boldsymbol{x}$ represent a query input, and y^* represent a label for $\boldsymbol{x}$. We define the functional regret of a distribution q as:*

$$Regret_f(q, \mathcal{D}, \boldsymbol{x}, y^*) = |\mathbb{E}_{y \sim q(y|\boldsymbol{x})}[g(y)] - \mathbb{E}_{y \sim p(y|\boldsymbol{x}, \hat{\theta})}[g(y)]|$$

Where $\hat{\theta}$ is MLE estimator for the augmented dataset $\mathcal{D} \cup (\boldsymbol{x}, y^*)$ formed by appending $(\boldsymbol{x}, y^*)$ to $\mathcal{D}$.

A straightforward choice for the evaluation function g is the identity function $g(y) = y$, in which case the functional regret controls the difference in expected values between q and the MLE estimate $p(y|x, \hat{\theta})$. We now show that the functional regret is bounded for the CNML distribution:

Theorem 6.4.1. *Let p_{NML} be the conditional NML distribution defined in Eqn. 6.2. Then,*

$$\forall_x \max_{y^*} Regret_f(p_{NML}, \mathcal{D}, x, y^*) \leq 2g_{max}\sqrt{\Gamma(\mathcal{D}, x)/2}.$$

$\Gamma(\mathcal{D}, x) = \log\{\sum_y p(y|x, \hat{\theta}_{\mathcal{D}\cup(\boldsymbol{x},y)})\}$ is the minimax individual regret, and $g_{max} = \max_{y\in\mathcal{Y}} g(y)$.

Proof. There are two lemmas we will use in our proof. First, the difference in expected value between two distributions $p(x)$ and $q(x)$ can be bounded by the total variation distance $TV(p, q)$ and the maximum function value $f_{\max} = \max_x f(x)$:

$$\begin{aligned}
|E_{p(x)}[f(x)] - E_{q(x)}[f(x)]| &= |\sum_x [p(x) - q(x)]f(x)| \\
&\leq f_{\max}|\sum_x [p(x) - q(x)]| \\
&= f_{\max} 2TV(p(x), q(x))
\end{aligned}$$

Second, Fogel and Feder [2018] show that the NML distribution obtains the best possible minimax individual regret of

$$\max_y \text{Regret}_{\text{ind}}(p_{\text{NML}}, \mathcal{D}, x, y) = \log\{\sum_y p(y|x, \hat{\theta}_{\mathcal{D}\cup(\boldsymbol{x},y)})\} \stackrel{\text{def}}{=} \Gamma(\mathcal{D}, x)$$

Using these two facts, we can show:

$$\begin{aligned}
\text{Regret}_f(p_{\text{NML}}, \mathcal{D}, x, y^*) &= |\mathbb{E}_{y\sim p_{\text{NML}}(y|x)}[g(y)] - \mathbb{E}_{y\sim p(y|x,\hat{\theta}_{\mathcal{D}\cup(\boldsymbol{x},y^*)})}[g(y)]| \\
&\leq 2g_{\max} TV(p(y|x, \hat{\theta}_{\mathcal{D}\cup(\boldsymbol{x},y^*)}), p_{\text{NML}}(y|x)) \\
&\leq 2g_{\max}\sqrt{\frac{1}{2} KL(p(y|x, \hat{\theta}_{\mathcal{D}\cup(\boldsymbol{x},y^*)}), p_{\text{NML}}(y|x))} \\
&\leq 2g_{\max}\sqrt{\frac{1}{2}\max_q \mathbb{E}_{y\sim q(y|x)}[\log p(y|x, \hat{\theta}_{\mathcal{D}\cup(\boldsymbol{x},y)}) - \log p_{\text{NML}}(y|x)]} \\
&= 2g_{\max}\sqrt{\Gamma(\mathcal{D}, x)/2}
\end{aligned}$$

Where we apply the total variation distance lemma from lines 1 to 2. From lines 2 to 3, we used Pinsker's inequality to bound total variation with KL, and from lines 3 to 4 we used the fact that the maximum regret is always greater than the KL, i.e.

$$\begin{aligned}
KL(p(y|x, \hat{\theta}_{\mathcal{D}\cup(\boldsymbol{x},y^*)}), p_{\text{NML}}(y|x)) &= \mathbb{E}_{y\sim p(y|x,\hat{\theta})}[\log p(y|x, \hat{\theta}_{\mathcal{D}\cup(\boldsymbol{x},y^*)}) - \log p_{\text{NML}}(y|x)] \\
&\leq \mathbb{E}_{y\sim p(y|x,\hat{\theta})}[\log p(y|x, \hat{\theta}_{\mathcal{D}\cup(\boldsymbol{x},y)}) - \log p_{\text{NML}}(y|x)] \\
&\leq \max_q \mathbb{E}_{y\sim q(y|x)}[\log p(y|x, \hat{\theta}_{\mathcal{D}\cup(\boldsymbol{x},y)}) - \log p_{\text{NML}}(y|x)]
\end{aligned}$$

On the final step, we substituted the definition of $\Gamma(\mathcal{D}, x)$ as the individual regret of the NML distribution p_{NML}. □

This statement states that, for any test input $\boldsymbol{x}$, the CNML estimator is close to the best possible expert if the test label y is chosen adversarially. Importantly, the expert is allowed to see the label of the test point, but the CNML estimator is not, which means that if the true function lies within the model class, this statement effectively controls the discrepancy in performance between the true function and p_{NML}. The amount of slack is controlled by the minimax individual regret Γ [Fogel and Feder, 2018], which can be interpreted as a measure of uncertainty in the model. For large model classes Θ and data points $\boldsymbol{x}$ far away from the data, the individual regret is naturally larger as the NML estimator becomes more uncertain, but for data points $\boldsymbol{x}$ close to Θ the regret becomes very small. This behavior can be easily seen in Fig. 6.3, where the CNML distribution is very focused in regions close to the data but outputs large uncertainty estimates in out-of-distribution regions.

6.5 Experiments

In our experimental evaluation, we aim to **1)** evaluate how well the proposed quantized NML estimator estimates uncertainty in an offline setting, and **2)** compare the performance of NEMO to a number of recently proposed offline MBO algorithms on high-dimensional offline MBO benchmark problems. Our code is available at `https://sites.google.com/view/nemo-mbo`

Modeling with Quantized NML

We begin with an illustrative example of modeling a function with quantized NML. We compared a learned a quantized NML distribution with a bootstrapped ensemble method [Breiman, 1996] on a simple 1-dimensional problem, shown in Fig. 6.3. The ensemble method is implemented by training 32 neural networks using the same model class as NML, but with resampled datasets and randomized initializations. Both methods are trained on a discretized output, using a softmax cross-entropy loss function. We see that in areas within the support of the data, the NML distribution is both confident and relatively accurate. However, in regions outside of the support, the quantized NML outputs a highly uncertain estimate. In contrast, the ensemble method, even with bootstrapping and random initializations, tends to produce an ensemble of models that all output similar values. Therefore, in regimes outside of the data, the ensemble still outputs highly confident estimates, even though they may be wrong.

High-Dimensional Model-Based Optimization

We evaluated NEMO on a set of high-dimensional MBO problems. The details for the tasks, baselines, and experimental setup are as follows, and hyperparameter choices with additional implementation details can be found in Appendix C.1.

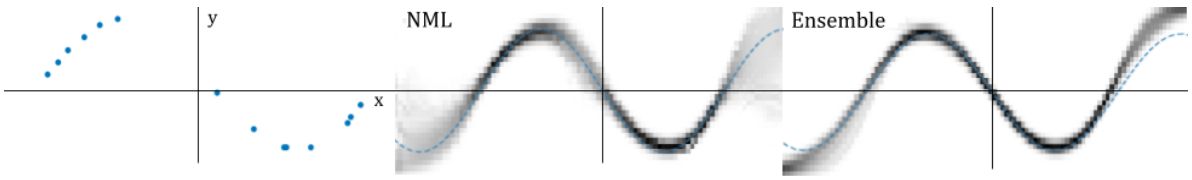


Figure 6.3: A comparison of uncertainty estimates between quantized NML and bootstrapped ensembles. **Left**: A small training dataset collected using the ground truth function $\sin(x)$ and quantized into 32 bins. Each dot represents a data point. **Middle**: Predictions from quantized CNML, where darker regions indicate outputs with high estimated probability. The ground truth is marked by a dotted blue line. Note that in regions where there is little data, the NML distribution tends to correctly output a very diffuse distribution with uncertain outputs. **Right**: Predictions from a bootstrap ensemble of 32 models. In out-of-support regions far from the data, the bootstrap ensemble tends to underestimate uncertainty and produce overconfident predictions.

Tasks

We evaluated on 6 tasks from the Design-bench [Trabucco et al., 2021], modeled after real-world design problems for problems in materials engineering [Hamidieh, 2018], biology [Sarkisyan et al., 2016], and chemistry [Gaulton et al., 2012], and simulated robotics. Because we do not have access to a real physical process for evaluating the material and molecule design tasks, Design-bench follows experimental protocol used in prior work [Brookes et al., 2019, Fannjiang and Listgarten, 2020] which obtains a ground truth evaluation function by training a separate regressor model to evaluate the performance of designs. For the robotics tasks, designs are evaluated using the MuJoCo physics simulator [Todorov et al., 2012].

Superconductor. The Superconductor task involves designing a superconducting material that has a high critical temperature. The input is space is an 81-dimensional vector, representing properties such as atomic radius and valence of elements which make up the material. This dataset contains a total of 21,263 superconductoring materials proposed by Hamidieh [2018].

GFP. The goal of the green fluorescent protein (GFP) task is to design a protein with high fluorescence, based on work proposed by Sarkisyan et al. [2016]. This task requires optimizing a 238-dimensional sequence of discrete variables, with each dimension representing one amino acid in the protein and taking on one of 20 values. We parameterize the input space as logits in order to make this discrete problem amenable to continuous optimization. In total, the dataset consists of 5000 such proteins annotated with fluorescence values.

MoleculeActivity. The MoleculeActivity task involves designing the substructure of a molecule that exhibits high activity when tested against a target assay [Gaulton et al., 2012]. The input space is represented by 1024 binary variables parameterized by logits, which corresponds to the Morgan radius 2 substructure fingerprints. This dataset contains a total of 4216 data points.

The final 3 tasks, **HopperController**, **AntMorphology**, and **DKittyMorphology**, involve designing robotic agents. HopperController involves learning the parameters of a 5126-parameter neural network controller for the Hopper-v2 task from OpenAI Gym [Brockman et al., 2016]. The Ant and DKitty morphology tasks involve optimizing robot parameters such as size, orientation, and joint positions. AntMorphology has 60 parameters, and DKittyMorphology has 56 parameters.

Baselines

In addition to NEMO, we evaluate several baseline methods. A logical alternative to NEMO is a forward ensemble method, since both NEMO and ensemble methods maintain a list of multiple models in order to approximate a distribution over the function value, and ensembles are often used to obtain uncertainty-aware models. We implement an ensemble baseline by training K networks on the task dataset with random initializations and bootstrapping, and then optimizing the mean value of the ensemble with gradient ascent. In our results in Table 6.1, we label the ensemble as "Ensemble" and a single forward model as "Forward". Additionally, we implement a Bayesian optimization baseline wth Gaussian processes (GP-BO) for the Superconductor, GFP, and MoleculeActivity tasks, where we fit the parameters of a kernel and then optimize the expected improvement according to the posterior. We use an RBF kernel for the Superconductor task, and an inner product kernel for the GFP and MoleculeActivity tasks since they have large, discrete input spaces. Note that the GP baseline has no variance between runs since the resulting method is completely deterministic.

We evaluate 3 state-of-the-art methods for offline MBO: model inversion networks (MINs) [Kumar and Levine, 2020], conditioning by adaptive sampling (CbAS) [Brookes et al., 2019], and autofocused oracles [Fannjiang and Listgarten, 2020]. MINs train an inverse mapping from outputs y to inputs $\boldsymbol{x}$, and generate candidate inputs by searching over high values of y and evaluating these on the inverse model. CbAS uses a generative model of $p(x)$ as a trust region to prevent model exploitation, and autofocused oracles expands upon CbAS by iteratively updating the learned proxy function and iterates within a minimax game based on a quantity known as the oracle gap.

6.6 Results and Discussion

Our results are shown in Table 6.1. We follow an evaluation protocol used in prior work for design problems [Brookes et al., 2019, Fannjiang and Listgarten, 2020], where the algorithm proposes a set of candidate designs, and the 100th and 50th percentile of scores are reported. This mimics a real-world scenario in which a batch of designs can be synthesized in parallel, and the highest performing designs are selected for use.

For each experiment, we produced a batch of 128 candidate designs. MINs, CbAS, and autofocused oracles all learn a generative model to produce candidate designs, so we sampled this batch from the corresponding model. Ensemble methods and NEMO do not maintain

generative models, so we instead optimized a batch of 128 particles. We report results averaged of 16 random seeds.

NEMO outperforms all methods on the Superconductor task by a very large margin, under both the 100th and 50th percentile metrics, and in the HopperController task under the 100th percentile metric. For the remaining tasks (GFP, MoleculeActivity, AntMorphology, and HopperMorphology), NEMO also produces competitive results in line with the best performing algorithm for each task. These results are promising in that NEMO performs consistently well across all 6 domains evaluated, and indicates a significant number of designs found in the GFP and Superconductor task were *better* than the best performing design in the dataset. In Section 6.6, we present learning curves for NEMO, as well as an ablation study demonstrating the the beneficial effect of NML compared to direct optimization on a proxy function. Note that unlike the prior methods (MINs, CbAS, Autofocused), NEMO *does not* require training a generative model on the data, only a collection of forward models.

Ablation Studies

In this section, we present 3 ablation studies. The first is on the effect of NML training, by comparing NEMO to optimizing a pretrained baseline neural network. The second ablation study investigates the architecture choice, comparing the discretized logistic architecture to a standard feedforward neural network. The final ablation study investigates the ratio of model optimization steps to input optimization steps. Each logging iteration in these figures corresponds to 50 loop iterations as depicted in Algorithm 6.

Effect of NML

In this section, we present learning curves for NML, as well as an ablation study comparing NML (orange curve) against a forward optimization algorithm without NML (blue curve), labeled "No NML". The "No NML" algorithm is identical to the NML algorithm detailed in Alg. 6, except the NML learning rate α_θ is set to 0.0. This means that the only model training that happens is done during the pretraining step. For illustrative purposes, we initialize the iterates from the worst-performing scores in the dataset to better visualize improvement, rather than initializing from the best scores which we used in our final reported numbers.

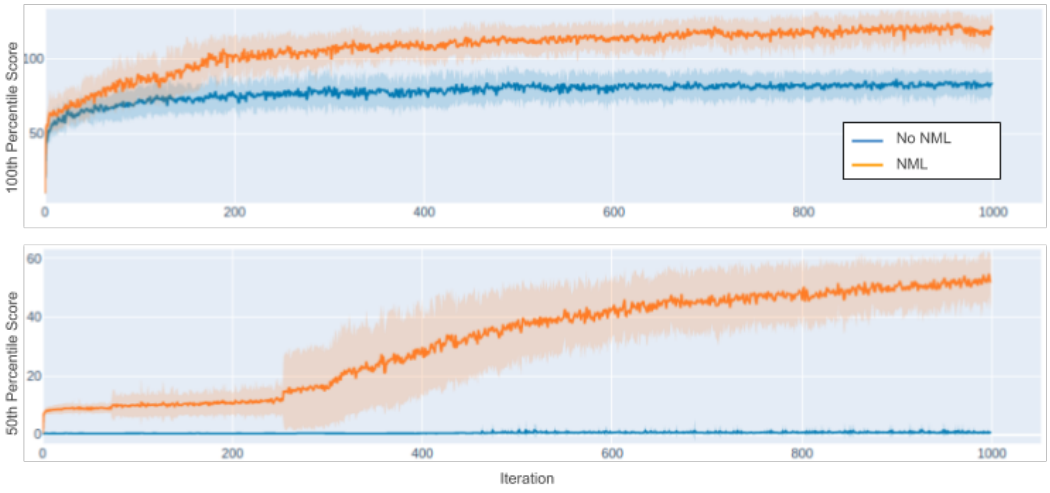
The scores on the Superconductor task are shown in the following figure. Removing NML training makes it very difficult for training on most designs, as shown by the poor performance on the 50th percentile metric.

100th Perc.	Superconductor	GFP	MoleculeActivity
NEMO **(ours)**	**127.0 ± 7.292**	3.359 ± 0.036	**6.682 ± 0.209**
Ensemble	88.01 ± 10.43	2.924 ± 0.039	6.525 ± 0.1159
Forward	89.64 ± 9.201	2.894 ± 0.001	6.636 ± 0.066
MINs	80.23 ± 10.67	3.315 ± 0.033	6.508 ± 0.236
CbAS	72.17 ± 8.652	**3.408 ± 0.029**	6.301 ± 0.131
Autofoc.	77.07 ± 11.11	**3.365 ± 0.023**	6.345 ± 0.141
GP-BO	89.72 ± 0.000	2.894 ± 0.000	**6.745 ± 0.000**
Dataset Max	73.90	3.152	6.558
	HopperController	AntMorphology	DKittyMorphology
NEMO **(ours)**	**2130.1 ± 506.9**	**393.7 ± 6.135**	**431.6 ± 47.79**
Ensemble	1877.0 ± 704.2	-	-
Forward	1050.8 ± 284.5	**399.9 ± 4.941**	390.7 ± 49.24
MINs	746.1 ± 636.8	388.5 ± 9.085	352.9 ± 38.65
CbAS	547.1 ± 423.9	**393.0 ± 3.750**	369.1 ± 60.65
Autofoc.	443.8 ± 142.9	386.9 ± 10.58	376.3 ± 47.47
Dataset Max	1361.6	108.5	215.9

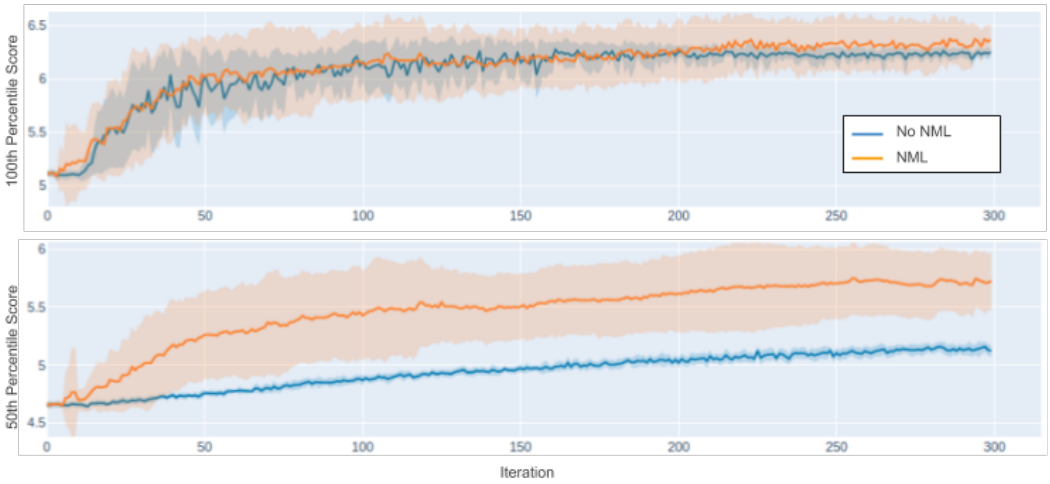
Table 6.1: 100th percentile ground truth scores and standard deviations over a batch of 128 designs for each task, averaged across 16 trials.

50th Perc.	Superconductor	GFP	MoleculeActivity
NEMO **(ours)**	66.41 ± 4.618	**3.219 ± 0.039**	5.814 ± 0.092
Ensemble	48.72 ± 2.637	2.910 ± 0.020	**6.412 ± 0.123**
Forward	54.06 ± 5.060	2.894 ± 0.000	**6.401 ± 0.186**
MINs	37.32 ± 10.50	3.135 ± 0.019	5.806 ± 0.078
CbAS	32.21 ± 7.255	**3.269 ± 0.018**	5.742 ± 0.123
Autofoc.	31.57 ± 7.457	**3.216 ± 0.029**	5.759 ± 0.158
GP-BO	**72.42 ± 0.000**	2.894 ± 0.000	6.373 ± 0.000
	HopperController	AntMorphology	DKittyMorphology
NEMO **(ours)**	390.2 ± 43.37	**326.9 ± 5.229**	180.8 ± 34.94
Ensemble	362.5 ± 80.09	-	-
Forward	185.0 ± 72.88	318.0 ± 12.05	**255.3 ± 6.379**
MINs	**520.4 ± 301.5**	184.8 ± 29.52	211.6 ± 13.67
CbAS	132.5 ± 23.88	267.3 ± 16.55	203.2 ± 3.580
Autofoc.	116.4 ± 18.66	176.7 ± 59.94	199.3 ± 8.909

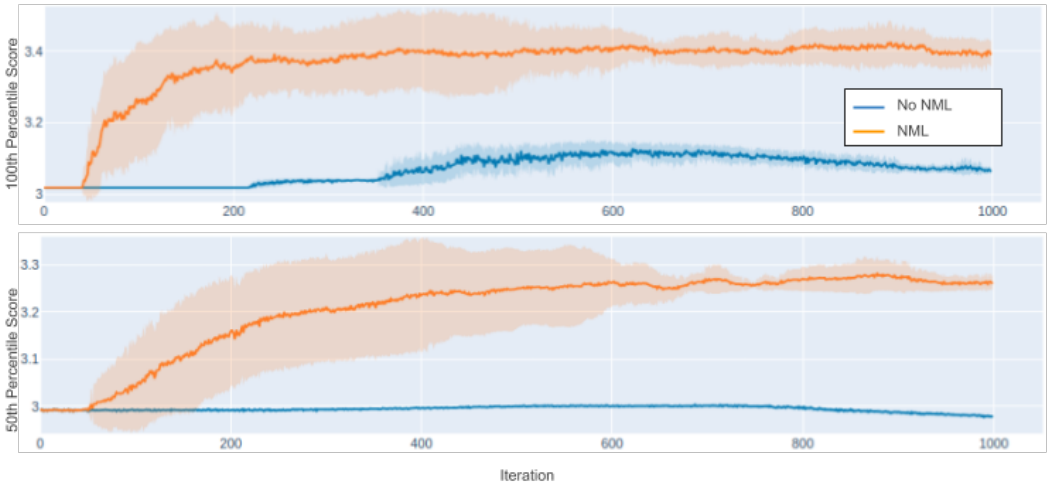
Table 6.2: 50th percentile ground truth scores and standard deviations over a batch of 128 designs for each task, averaged across 16 trials.



The scores on the MoleculeActivity task follow a similar trend.



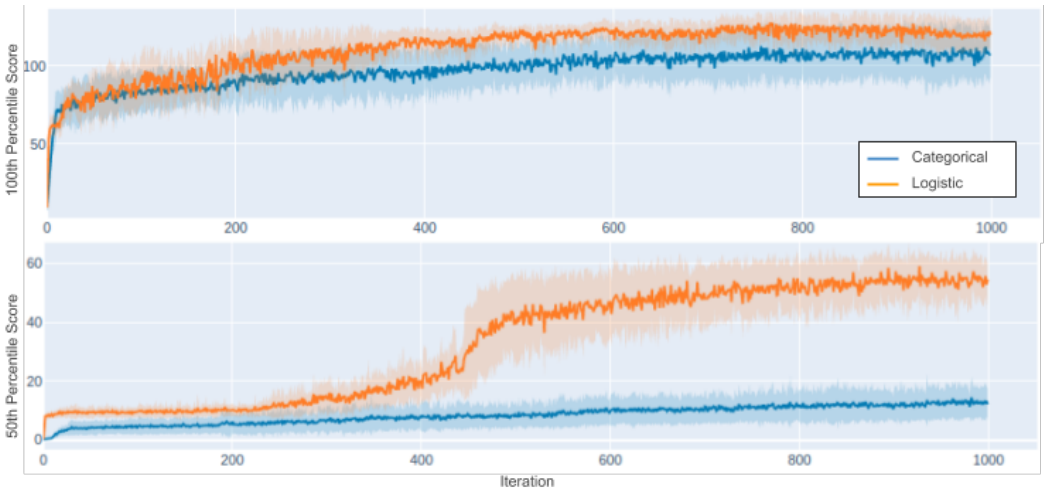
And finally, the scores on the GFP task also display the same trend.



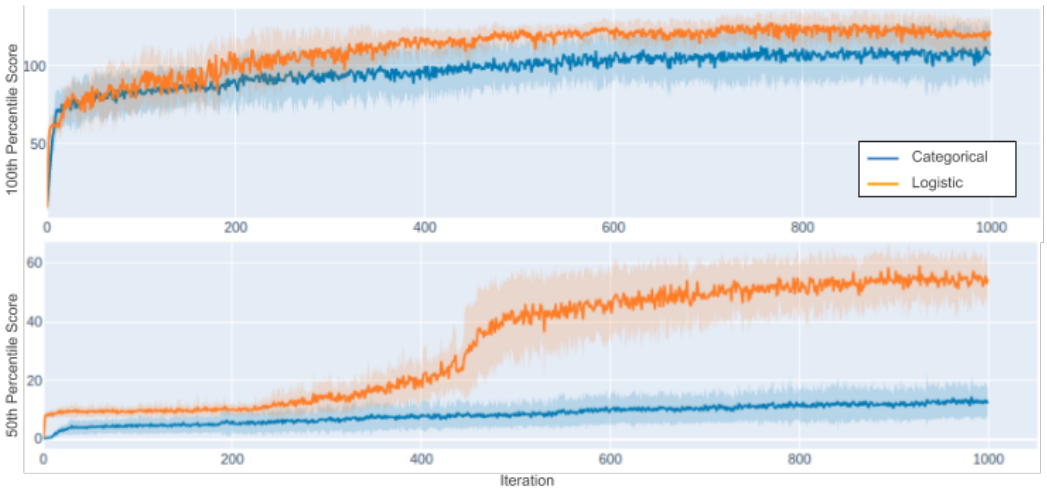
Architecture Choice

In this ablation study, we investigate the efficacy of the discretized logistic architecture. As a baseline, we compared against a standard feedforward network, trained with a softmax cross-entropy loss to predict the discretized output y. We label this network as "Categorical", because the output of the network is a categorical distribution. All other hyperparameters, including network layer sizes, remain unchanged from those reported in Appendix. C.1.

On the Superconductor task, both the discretized logistic and categorical networks score well on the 100th percentile metric, but the Categorical architecture displays less consistency in optimizing designs, as given by poor performance on the 50th percentile metric.



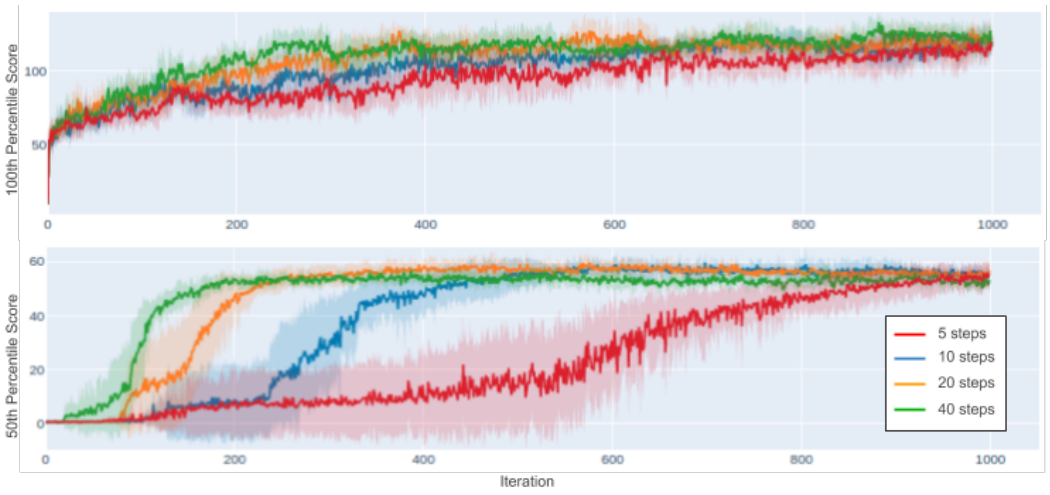
On the MoleculeActivity task, the Categorical network performs comparatively better, but still underperforms the discretized logistic architecture.



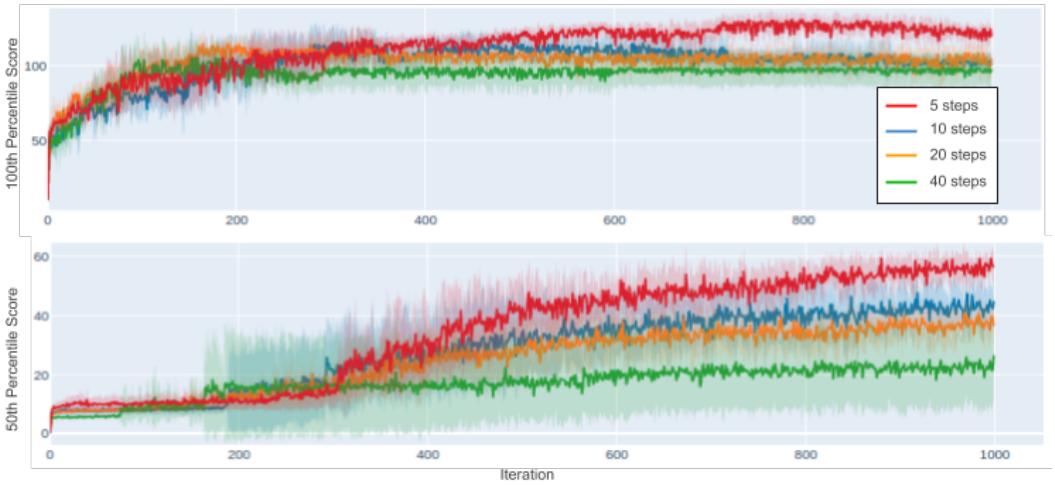
Ratio of Optimization Steps

In this ablation study, we investigate the effect of the ratio of model optimization steps to input optimization step. For this experiment, we fix the learning rate α_x to the hyperparameter values in Appendix. C.1, fix the input optimization steps to 1, and vary the number of model optimization steps we take.

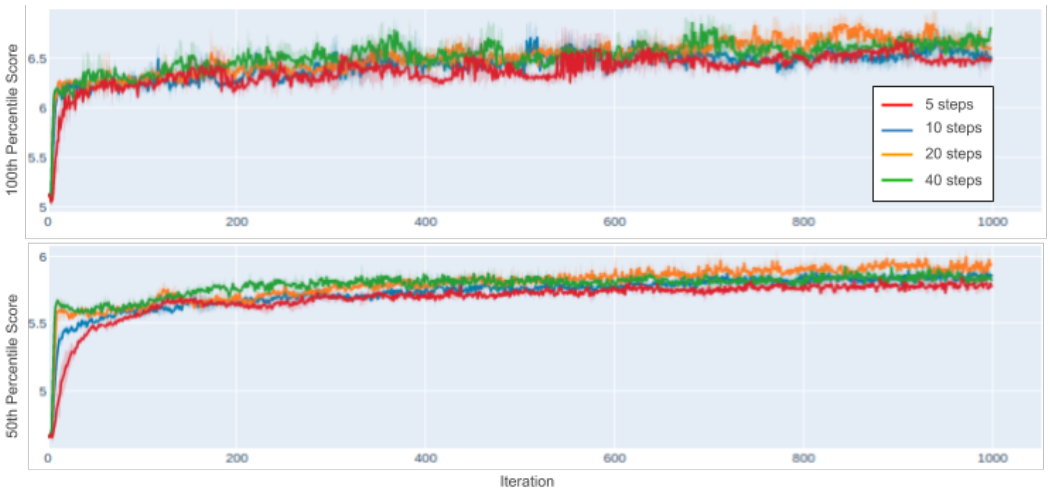
We first investigate the Superconductor task, using a small model learning rate $\alpha_\theta = 0.0005$. In this setting, we see a clear trend that additional model optimization steps are helpful and increase convergence speed.



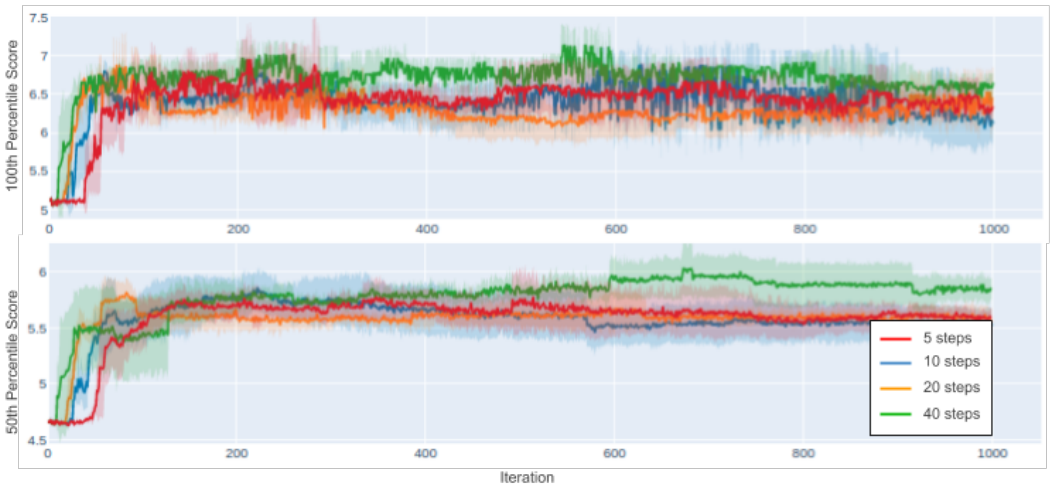
Using a higher learning rate of $\alpha_\theta = 0.05$, a smaller amount of steps works better, which suggests that it is easy to destabilize the learning process using a higher learning rate.



A similar trend holds true in the MoleculeActivity task, albeit less pronounced. The following figure uses a learning rate of $\alpha_\theta = 0.0005$, and we once again see that more model optimization steps leads to increased performance.



And using a learning rate of $\alpha_\theta = 0.05$, the advantage becomes less clear.



Overall, while we performed a grid search over the learning rates to achieve the highest performance, using a large number of model optimization steps with a small learning rate α_θ appears to be a consistent strategy which performs well.

6.7 Discussion

We have presented NEMO (Normalized Maximum Likelihood Estimation for Model-Based Optimization), an algorithm that mitigates model exploitation on MBO problems by constructing a conservative model of the true function. NEMO generates a tractable approximation to the NML distribution to provide conservative objective value estimates for out-of-distribution inputs. Our theoretical analysis also suggests that this approach is an effective

way to estimate unknown objective functions outside the training distribution. We evaluated NEMO on a number of design problems in materials science, robotics, biology, and chemistry, where we show that it attains very large improvements on two tasks, while performing competitively with respect to prior methods on the other four.

Part II

Benchmarks and Evaluation

Chapter 7

Benchmarking Offline Reinforcement Learning

While offline reinforcement learning (RL) holds the promise leveraging large, previously-collected datasets in the context of sequential decision making, current methods have not yet fulfilled the promise of utilizing these algorithms in a meaningful way to solve real-world application domains such as robotics, autonomous driving, and healthcare. While recent work has investigated technical reasons for this [Fujimoto et al., 2019b, Kumar et al., 2019, Wu et al., 2019], a major challenge in addressing these issues has been the lack of standard evaluation benchmarks to guide and measure progress. Ideally, such a benchmark should: **a)** be composed of tasks that reflect challenges in real-world applications of data-driven RL, **b)** be widely accessible for researchers and define clear evaluation protocols for reproducibility, and **c)** contain a range of difficulty to differentiate between algorithms, especially challenges particular to the offline RL setting.

Most recent works [Fujimoto et al., 2018, Wu et al., 2019, Kumar et al., 2019, Peng et al., 2019, Agarwal et al., 2020b] use existing online RL benchmark domains and data collected from training runs of online RL methods. However, these benchmarks were not designed with offline RL in mind and such datasets do not reflect the heterogenous nature of data collected in practice. Wu et al. [2019] find that existing benchmark datasets are not sufficient to differentiate between simple baseline approaches and recently proposed algorithms. Furthermore, the aforementioned works do not propose a standard evaluation protocol, which makes comparing methods challenging.

In this chapter, we present Datasets for Deep Data-Driven Reinforcement Learning (D4RL), a benchmark targeted towards policy optimization algorithms which attempt to find the best policy given a dataset of interaction. We focus our design around tasks and data collection strategies that exercise dimensions of the offline RL problem likely to occur in practical applications, such as partial observability, passively logged data, or human demonstrations. To serve as a reference, we benchmark state-of-the-art offline RL algorithms [Haarnoja et al., 2018, Kumar et al., 2019, Wu et al., 2019, Agarwal et al., 2020b, Fujimoto et al., 2019b, Nachum et al., 2019, Peng et al., 2019, Kumar et al., 2020] and pro-

vide reference implementations as a starting point for future work. While previous studies (e.g., [Wu et al., 2019]) found that all methods including simple baselines performed well on the limited set of tasks used in prior work, we find that most algorithms struggle to perform well on tasks with properties crucial to real-world applications such as passively logged data, narrow data distributions, and limited human demonstrations. By moving beyond simple benchmark tasks and data collected by partially-trained RL agents, we reveal important and unappreciated deficiencies of existing algorithms.

7.1 Relation to Prior Work

The main points of difference between D4RL and other related work, such as RL Unplugged [Gulcehre et al., 2020] and Fujimoto et al. [2019a] lies in the selection of tasks and datasets. RL Unplugged is a benchmark mainly focused on locomotion and manipulation tasks with perceptually challenging input and partial observability. Fujimoto et al. [2019a] also proposes a visually challenging benchmark based on the Atari domain. While these are important contributions, both benchmarks suffer from the same shortcomings as prior evaluation protocols: they rely on data collected from online RL training runs. In contrast with these benchmarks, in addition to collecting data from online RL training runs, D4RL focuses on a range of dataset collection procedures inspired by real-world applications, such as human demonstrations, exploratory agents, and hand-coded controllers. As alluded to by Wu et al. [2019] and as we show in our experiments, the performance of current methods depends strongly on the data collection procedure, demonstrating the importance of modeling realistic data collection procedures in a benchmark.

7.2 Task Design Factors

In order to design a benchmark that provides a meaningful measure of progress towards realistic applications of offline RL, we choose datasets and tasks to cover a range of properties designed to challenge existing RL algorithms. We discuss these properties as follows:

Narrow and biased data distributions, such as those from deterministic policies, are problematic for offline RL algorithms and may cause divergence both empirically [Fujimoto et al., 2019b, Kumar et al., 2019] and theoretically [Munos, 2003, Farahmand et al., 2010, Kumar et al., 2019, Agarwal et al., 2020a, Du et al., 2020]. Narrow datasets may arise in human demonstrations, or when using hand-crafted policies. An important challenge in offline RL is to be able to gracefully handle diverse data distributions without algorithms diverging or producing performance worse than the provided behavior. A common approach for dealing with such data distributions is to adopt a conservative approach which tries to keep the behavior close to the data distribution [Fujimoto et al., 2019b, Kumar et al., 2019, Wu et al., 2019].

Undirected and multitask data naturally arises when data is passively logged, such as recording user interactions on the internet or recording videos of a car for autonomous driving. This data may not necessarily be directed towards the specific task one is trying to accomplish. However, pieces of trajectories can still provide useful information to learn from. For example, one may be able to combine sub-trajectories to accomplish a task. In the figure to the upper-right, if an agent is given trajectories from A-B and B-C in a dataset (left image), it can form a trajectory from A-C by combining the corresponding halves of the original trajectories. We refer to this property as *stitching*, since the agent can use portions of existing trajectories in order to solve a task, rather than relying on generalization outside of the dataset.

Sparse rewards. Sparse reward problems pose challenges to traditional RL methods due to the difficulty of credit assignment and exploration. Because offline RL considers fixed datasets without exploration, sparse reward problems provide an unique opportunity to isolate the ability of algorithms to perform credit assignment decoupled from exploration.

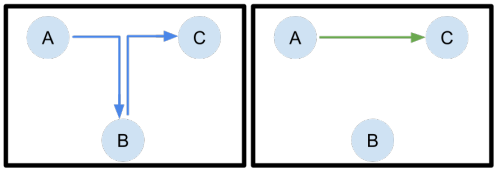


Figure 7.1: An example of stitching together subtrajectories to solve a task.

Suboptimal data. For tasks with a clear objective, the datasets may not contain behaviors from optimal agents. This represents a challenge for approaches such as imitation learning, which generally require expert demonstrations. We note prior work (e.g., [Fujimoto et al., 2019b, Kumar et al., 2019, Wu et al., 2019]) predominantly uses data with this property.

Non-representable behavior policies, non-Markovian behavior policies, and partial observability. When the dataset is generated from a partially-trained agent, we ensure that the behavior policy can be realized within our model class. However, real-life behavior may not originate from a policy within our model class, which can introduce additional representational errors. For example, data generated from human demonstrations or handcrafted controllers may fall outside of the model class. More generally, **non-Markovian** policies and tasks with **partial observability** can introduce additional modeling errors when we estimate action probabilities under the assumption that the data was generated from a Markovian policy. These errors can cause additional bias for offline RL algorithms, especially in methods that assume access to action probabilities from a Markovian policy such as importance weighting [Precup et al., 2000].

Realistic domains. As we discussed previously, real-world evaluation is the ideal setting for benchmarking offline RL, however, it is at odds with a widely-accessible and reproducible benchmark. To strike a balance, we opted for simulated environments which have been previously studied and are broadly accepted by the research community. These simulation packages (such as MuJoCo, Flow, and CARLA) have been widely used to benchmark *online* RL methods and are known to fit well into that role. Moreover, on several domains we utilize human demonstrations or mathematical models of human behavior in order to provide datasets generated from realistic processes. However, this did have the effect of restricting

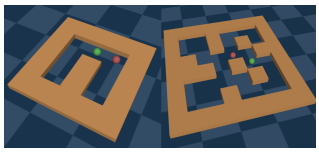
our choice of tasks. While recent progress has been made on simulators for recommender systems (e.g., [Ie et al., 2019]), they use "stylized" user models and they have not been thoroughly evaluated by the community yet. In the future as simulators mature, we hope to include additional tasks.

In addition, we include a variety of qualitatively different tasks to provide broad coverage of the types of domains where offline RL could be used. We include locomotion, traffic management, autonomous driving, and robotics tasks. We also provide tasks with a wide range in difficulty, from tasks current algorithms can already solve to harder problems that are currently out of reach. Finally, for consistency with prior works, we also include the OpenAI Gym robotic locomotion tasks and similar datasets used by Fujimoto et al. [2019b], Kumar et al. [2019], Wu et al. [2019].

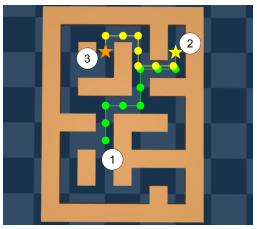
7.3 Tasks and Datasets

Given the properties outlined in Section 7.2, we assembled the following tasks and datasets. All tasks consist of an offline *dataset* (typically 10^6 steps) of trajectory samples for training, and a *simulator* for evaluation. The mapping is not one-to-one – several tasks use the same simulator with different datasets. Appendix D.3 lists domains and dataset types along with their sources and Appendix D.2 contains a more comprehensive table of statistics such as size.

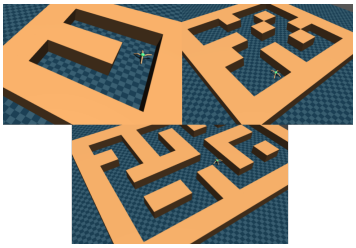
Maze2D. *(Non-markovian policies, undirected and multitask data)* The Maze2D domain is a navigation task requiring a 2D agent to reach a fixed goal location. The tasks are designed to provide a simple test of the ability of offline RL algorithms to stitch together previously collected subtrajectories to find the shortest path to the evaluation goal. Three maze layouts are provided. The "umaze" and "medium" mazes are shown to the right, and the "large" maze is shown below.



The data is generated by selecting goal locations at random and then using a planner that generates sequences of waypoints that are followed using a PD controller. In the figure on the left, the waypoints, represented by circles, are planned from the starting location (1) along the path to a goal (2). Upon reaching a threshold distance to a waypoint, the controller updates its internal state to track the next waypoint along the path to the goal. Once a goal is reached, a new goal is selected (3) and the process continues. The trajectories in the dataset are visualized in Appendix D.7. Because the controllers memorize the reached waypoints, the data collection policy is non-Markovian.

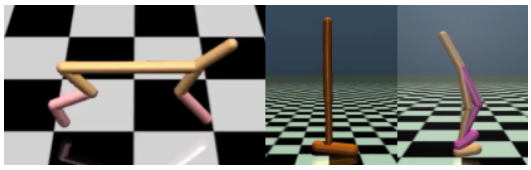


AntMaze. *(Non-markovian policies, sparse rewards, undirected and multitask data)* The AntMaze domain is a navigation domain that replaces the 2D ball from Maze2D with the more complex 8-DoF "Ant" quadraped robot. We introduce this domain to test the stitching challenge using a morphologically complex robot that could mimic real-world robotic navigation tasks. Additionally, for this task we use a sparse 0-1 reward which is activated upon reaching the goal.



The data is generated by training a goal reaching policy and using it in conjunction with the same high-level waypoint generator from **Maze2D** to provide subgoals that guide the agent to the goal. The same 3 maze layouts are used: "umaze", "medium", and "large". We introduce three flavors of datasets: 1) the ant is commanded to reach a specific goal from a fixed start location (`antmaze-umaze-v0`), 2) in the "diverse" datasets, the ant is commanded to a random goal from a random start location, 3) in the "play" datasets, the ant is commanded to specific hand-picked locations in the maze (which are not necessarily the goal at evaluation), starting from a different set of hand-picked start locations. As in Maze2D, the controllers for this task are non-Markovian as they rely on tracking visited waypoints. Trajectories in the dataset are visualized in Appendix D.7.

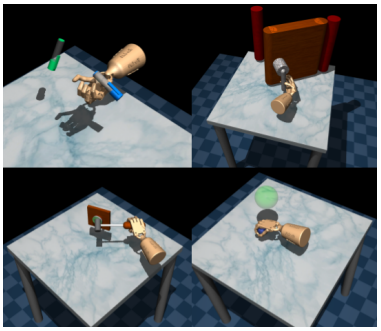
Gym-MuJoCo. *(Suboptimal agents, narrow data distributions)*



The Gym-MuJoCo tasks (Hopper, HalfCheetah, Walker2d) are popular benchmarks used in prior work in offline deep RL [Fujimoto et al., 2019b, Kumar et al., 2019, Wu et al., 2019]. For consistency, we provide standardized datasets similar to previous work, and additionally propose mixing datasets to test the impact of heterogenous policy mixtures. We expect that methods that rely on regularizing to the behavior policy may fail when the data contains poorly performing trajectories.

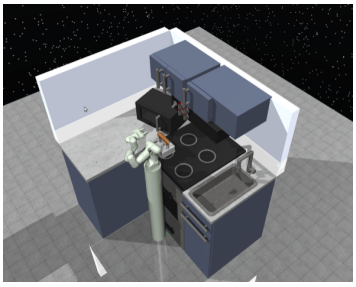
The "medium" dataset is generated by first training a policy online using Soft Actor-Critic [Haarnoja et al., 2018], early-stopping the training, and collecting 1M samples from this partially-trained policy. The "random" datasets are generated by unrolling a randomly initialized policy on these three domains. The "medium-replay" dataset consists of recording all samples in the replay buffer observed during training until the policy reaches the "medium" level of performance. Datasets similar to these three have been used in prior work, but in order to evaluate algorithms on mixtures of policies, we further introduce a "medium-expert" dataset by mixing equal amounts of expert demonstrations and suboptimal data, generated via a partially trained policy or by unrolling a uniform-at-random policy.

Adroit. *(Non-representable policies, narrow data distributions, sparse rewards, realistic)* The Adroit domain [Rajeswaran et al., 2018] (pictured left) involves controlling a 24-DoF simulated Shadow Hand robot tasked with hammering a nail, opening a door, twirling a pen, or picking up and moving a ball. This domain was selected to measure the effect of a narrow expert data distributions and human demonstrations on a sparse reward, high-dimensional robotic manipulation task.



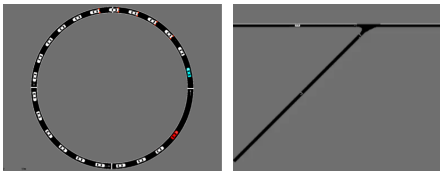
While Rajeswaran et al. [2018] propose utilizing human demonstrations, in conjunction with online RL fine-tuning, our benchmark adapts these tasks for evaluating the fully offline RL setting. We include three types of datasets for each task, two of which are included from the original paper: a small amount of demonstration data from a human ("human") (25 trajectories per task) and a large amount of expert data from a fine-tuned RL policy ("expert"). To mimic the use-case where a practitioner collects a small amount of additional data from a policy trained on the demonstrations, we introduce a third dataset generated by training an imitation policy on the demonstrations, running the policy, and mixing data at a 50-50 ratio with the demonstrations, referred to as "cloned." The Adroit domain has several unique properties that make it qualitatively different from the Gym MuJoCo tasks. First, the data is collected in from human demonstrators. Second, each task is difficult to solve with online RL, due to sparse rewards and exploration challenges, which make cloning and online RL alone insufficient. Lastly, the tasks are high dimensional, presenting a representation learning challenge.

FrankaKitchen. *(Undirected and multitask data, realistic)* The Franka Kitchen domain, first proposed by Gupta et al. [2020], involves controlling a 9-DoF Franka robot in a kitchen environment containing several common household items: a microwave, a kettle, an overhead light, cabinets, and an oven. The goal of each task is to interact with the items in order to reach a desired goal configuration. For example, one such state is to have the microwave and sliding cabinet door open with the kettle on the top burner and the overhead light on. This domain benchmarks the effect of multitask behavior on a realistic, non-navigation environment in which the "stitching" challenge is non-trivial because the collected trajectories are complex paths through the state space. As a result, algorithms must rely on generalization to unseen states in order to solve the task, rather than relying purely on trajectories seen during training.



In order to study the effect of "stitching" and generalization, we use 3 datasets of human demonstrations, originally proposed by Gupta et al. [2020]. The "complete" dataset consists of the robot performing all of the desired tasks in order. This provides data that is easy for an imitation learning method to solve. The "partial" and "mixed" datasets consist of

undirected data, where the robot performs subtasks that are not necessarily related to the goal configuration. In the "partial" dataset, a subset of the dataset is guaranteed to solve the task, meaning an imitation learning agent may learn by selectively choosing the right subsets of the data. The "mixed" dataset contains no trajectories which solve the task completely, and the RL agent must learn to assemble the relevant sub-trajectories. This dataset requires the highest degree of generalization in order to succeed.



Flow. *(Non-representable policies, realistic)* The Flow benchmark [Vinitsky et al., 2018] is a framework for studying traffic control using deep reinforcement learning. We use two tasks in the Flow benchmark which involve controlling autonomous vehicles to maximize the flow of traffic through a ring or merge road configuration (left).

We use the Flow domain in order to provide a task that simulates real-world traffic dynamics. A large challenge in autonomous driving is to be able to directly learn from human behavior. Thus, we include "human" data from agents controlled by the intelligent driver model (IDM) [Treiber et al., 2000], a hand-designed model of human driving behavior. In order to provide data with a wider distribution as a reference, we also include "random" data generated from an agent that commands random vehicle accelerations.

Offline CARLA. *(Partial observability, non-representable policies, undirected and multitask data, realistic)* CARLA [Dosovitskiy et al., 2017] is a high-fidelity autonomous driving simulator that has previously been used with imitation learning approaches [Rhinehart et al., 2019, Codevilla et al., 2018] from large, static datasets. The agent controls the throttle (gas pedal), the steering, and the break pedal for the car, and receives 48x48 RGB images from the driver's perspective as observations. We propose two tasks for offline RL: lane following within a figure eight path (shown to the right, top picture), and navigation within a small town (bottom picture). The principle challenge of the CARLA domain is partial observability and visual complexity, as all observations are provided as first-person RGB images.

The datasets in both tasks are generated via hand-designed controllers meant to emulate human driving - the lane-following task uses simple heuristics to avoid cars and keep the car within lane boundaries, whereas the navigation task layers an additional high-level controller on top that takes turns randomly at intersections. Similarly to the Maze2D and AntMaze domains, this dataset consists of undirected navigation data in order to test the "stitching" property, however, it is in a more perceptually challenging domain.

Evaluation protocol. Previous work tunes hyperparameters with online evaluation inside the simulator, and as Wu et al. [2019] show, the hyperparameters have a large impact

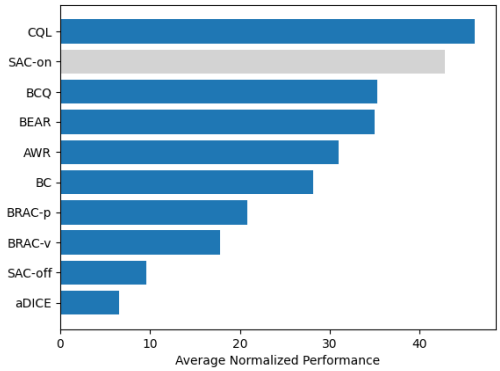
on performance. Unfortunately, extensive online evaluation is not practical in real-world applications and this leads to over optimistic performance expectations when the system is deployed in a truly offline setting. To rectify this problem, we designate a subset of tasks in each domain as "training" tasks, where hyperparameter tuning is allowed, and another subset as "evaluation" tasks on which final performance is measured (See Appendix D.4 Table D.3).

To facilitate comparison across tasks, we normalize scores for each environment roughly to the range between 0 and 100, by computing

$$\texttt{normalized score} = 100 * \frac{\texttt{score} - \texttt{random score}}{\texttt{expert score} - \texttt{random score}}.$$

A normalized score of 0 corresponds to the average returns (over 100 episodes) of an agent taking actions uniformly at random across the action space. A score of 100 corresponds to the average returns of a domain-specific expert. For Maze2D, and Flow domains, this corresponds to the performance of the hand-designed controller used to collect data. For CARLA, AntMaze, and FrankaKitchen, we used an estimate of the maximum score possible. For Adroit, this corresponds to a policy trained with behavioral cloning on human-demonstrations and fine-tuned with RL. For Gym-MuJoCo, this corresponds to a soft-actor critic [Haarnoja et al., 2018] agent.

7.4 Evaluation



We evaluated recently proposed offline RL algorithms and several baselines on our offline RL benchmarks. This evaluation **(1)** provides baselines as a reference for future work, and **(2)** identifies shortcomings in existing offline RL algorithms in order to guide future research. The average normalized performance for all tasks is plotted in the figure to the right. It is clear that as we move beyond simple tasks and data collection strategies, differences between algorithms are exacerbated and deficiencies in all algorithms are revealed. See Table 7.1 for normalized results for all tasks, Table 7.2 for unnormalized scores, and Appendix D.5 for experimental details.

We evaluated behavioral cloning (BC), online and offline soft actor-critic (SAC) [Haarnoja et al., 2018], bootstrapping error reduction (BEAR) [Kumar et al., 2019], and behavior-regularized actor-critic (BRAC) [Wu et al., 2019], advantage-weighted regression (AWR) [Peng et al., 2019], batch-constrained Q-learning (BCQ) [Fujimoto et al., 2019b], and AlgaeDICE [Nachum et al., 2019]. We note that REM was originally designed for discrete action spaces, and the continuous action version has not been developed extensively. In

most domains, we expect online SAC to outperform offline algorithms when given the same amount of data because this baseline is able to collect on-policy data. There are a few exceptions, such as for environments where exploration challenges make it difficult to find high-reward states, such as the Adroit and maze domains.

Overall, the benchmarked algorithms obtained the most success on datasets generated from an RL-trained policy, such as in the Adroit and Gym-MuJoCo domains. In these domains, offline RL algorithms are able to match the behavior policy when given expert data, and outperform when given suboptimal data. This positive result is expected, as it is the predominant setting in which these prior algorithms have been benchmarked in past work.

Another positive result comes in the form of sparse reward tasks. In particular, many methods were able to outperform the baseline online SAC method on the Adroit and AntMaze domains. This indicates that offline RL is a promising paradigm for overcoming exploration challenges, which has also been confirmed in recent work [Nair et al., 2020]. We also find that conservative methods that constrain the policy to the dataset, such as BEAR, AWR, CQL, and BCQ, are able to handle biased and narrow data distributions well on domains such as Flow and Gym-MuJoCo.

Tasks with undirected data, such as the Maze2D, FrankaKitchen, CARLA and AntMaze domains, are challenging for existing methods. Even in the simpler Maze2D domain, the large maze provides a surprising challenge for most methods. However, the smaller instances of Maze2D and AntMaze are very much within reach of current algorithms. Mixture distributions (a form of non-representable policies) were also challenging for all algorithms we evaluated. For MuJoCo, even though the medium-expert data contains expert data, the algorithms performed roughly on-par with medium datasets, except for hopper. The same pattern was found in the cloned datasets for Adroit, where the algorithms mostly performed on-par with the limited demonstration dataset, even though they had access to additional data.

We find that many algorithms were able to succeed to some extent on tasks with controller-generated data, such as Flow and Carla-lane. We also note that tasks with limited data, such as human demonstrations in Adroit and FrankaKitchen, remain challenging. This potentially points to the need for more sample-efficient methods, as big datasets may not be always be available.

7.5 Discussion

In this chapter we introduced the D4RL (Datasets for Deep Data-driven Reinforcement learning) benchmark, which focuses on benchmarking and evaluating algorithms for policy optimization in offline reinforcement learning. For each benchmark, we motivated our task selection by properties reflected in real-world tasks, such as narrow data distribution, undirected behavior, or temporally-extended control. Existing benchmarks have largely concentrated on robotic control using data produced by policies trained with RL [Fujimoto et al.,

2019b, Kumar et al., 2019, Wu et al., 2019, Gulcehre et al., 2020, Dulac-Arnold et al., 2021]. This can give a misleading sense of progress, as we have shown in our experiments that many of the more challenging properties that we expect real-world datasets to have appear to result in a substantial challenge for existing methods.

	Task Name	SAC	BC	SAC-off	BEAR	BRAC-p	BRAC-v	AWR	BCQ	aDICE	CQL
Maze 2D	maze2d-umaze	62.7	3.8	88.2	3.4	4.7	-16.0	1.0	12.8	-15.7	5.7
	maze2d-medium	21.3	30.3	26.1	29.0	32.4	33.8	7.6	8.3	10.0	5.0
	maze2d-large	2.7	5.0	-1.9	4.6	10.4	40.6	23.7	6.2	-0.1	12.5
AntMaze	antmaze-umaze	0.0	65.0	0.0	73.0	50.0	70.0	56.0	78.9	0.0	74.0
	antmaze-umaze-diverse	0.0	55.0	0.0	61.0	40.0	70.0	70.3	55.0	0.0	84.0
	antmaze-medium-play	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	61.2
	antmaze-medium-diverse	0.0	0.0	0.0	8.0	0.0	0.0	0.0	0.0	0.0	53.7
	antmaze-large-play	0.0	0.0	0.0	0.0	0.0	0.0	0.0	6.7	0.0	15.8
	antmaze-large-diverse	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.2	0.0	14.9
Gym	halfcheetah-random	100.0	2.1	30.5	25.1	24.1	31.2	2.5	2.2	-0.3	35.4
	walker2d-random	100.0	1.6	4.1	7.3	-0.2	1.9	1.5	4.9	0.5	7.0
	hopper-random	100.0	9.8	11.3	11.4	11.0	12.2	10.2	10.6	0.9	10.8
	halfcheetah-medium	100.0	36.1	-4.3	41.7	43.8	46.3	37.4	40.7	-2.2	44.4
	walker2d-medium	100.0	6.6	0.9	59.1	77.5	81.1	17.4	53.1	0.3	79.2
	hopper-medium	100.0	29.0	0.8	52.1	32.7	31.1	35.9	54.5	1.2	58.0
	halfcheetah-medium-replay	100.0	38.4	-2.4	38.6	45.4	47.7	40.3	38.2	-2.1	46.2
	walker2d-medium-replay	100.0	11.3	1.9	19.2	-0.3	0.9	15.5	15.0	0.6	26.7
	hopper-medium-replay	100.0	11.8	3.5	33.7	0.6	0.6	28.4	33.1	1.1	48.6
	halfcheetah-medium-expert	100.0	35.8	1.8	53.4	44.2	41.9	52.7	64.7	-0.8	62.4
	walker2d-medium-expert	100.0	6.4	-0.1	40.1	76.9	81.6	53.8	57.5	0.4	111.0
	hopper-medium-expert	100.0	111.9	1.6	96.3	1.9	0.8	27.1	110.9	1.1	98.7
Adroit	pen-human	21.6	34.4	6.3	-1.0	8.1	0.6	12.3	68.9	-3.3	37.5
	hammer-human	0.2	1.5	0.5	0.3	0.3	0.2	1.2	0.5	0.3	4.4
	door-human	-0.2	0.5	3.9	-0.3	-0.3	-0.3	0.4	-0.0	-0.0	9.9
	relocate-human	-0.2	0.0	0.0	-0.3	-0.3	-0.3	-0.0	-0.1	-0.1	0.2
	pen-cloned	21.6	56.9	23.5	26.5	1.6	-2.5	28.0	44.0	-2.9	39.2
	hammer-cloned	0.2	0.8	0.2	0.3	0.3	0.3	0.4	0.4	0.3	2.1
	door-cloned	-0.2	-0.1	0.0	-0.1	-0.1	-0.1	0.0	0.0	0.0	0.4
	relocate-cloned	-0.2	-0.1	-0.2	-0.3	-0.3	-0.3	-0.2	-0.3	-0.3	-0.1
	pen-expert	21.6	85.1	6.1	105.9	-3.5	-3.0	111.0	114.9	-3.5	107.0
	hammer-expert	0.2	125.6	25.2	127.3	0.3	0.3	39.0	107.2	0.3	86.7
	door-expert	-0.2	34.9	7.5	103.4	-0.3	-0.3	102.9	99.0	0.0	101.5
	relocate-expert	-0.2	101.3	-0.3	98.6	-0.3	-0.4	91.5	41.6	-0.1	95.0
Flow	flow-ring-controller	100.7	-57.0	9.2	62.7	-12.3	-91.2	75.2	76.2	15.2	52.0
	flow-ring-random	100.7	94.9	70.0	103.5	95.7	78.6	80.4	94.6	83.6	87.9
	flow-merge-controller	121.5	114.1	111.6	150.4	129.8	143.9	152.7	114.8	196.4	157.2
	flow-merge-random	121.5	-17.1	-40.1	-20.6	146.2	27.3	99.6	28.2	4.7	40.6
Franka Kitchen	kitchen-complete	0.0	33.8	15.0	0.0	0.0	0.0	0.0	8.1	0.0	43.8
	kitchen-partial	0.6	33.8	0.0	13.1	0.0	0.0	15.4	18.9	0.0	49.8
	kitchen-mixed	0.0	47.5	2.5	47.2	0.0	0.0	10.6	8.1	2.5	51.0
CARLA	carla-lane	-0.8	31.8	0.1	-0.2	18.2	19.6	-0.4	-0.1	-1.2	20.9
	carla-town	1.4	-1.8	-1.8	-2.7	-4.6	-2.6	1.9	1.9	-11.2	-2.6

Table 7.1: Normalized results comparing online & offline SAC (SAC, SAC-off), bootstrapping error reduction (BEAR), behavior-regularized actor critic with policy (BRAC-p) or value (BRAC-v) regularization, behavioral cloning (BC), advantage-weighted regression (AWR), batch-constrained Q-learning (BCQ), continuous random ensemble mixtures (cREM), and AlgaeDICE (aDICE). Average results are reported over 3 seeds, and normalized to a score between 0 (random) and 100 (expert).

Domain	Task Name	SAC	BC	SAC-off	BEAR	BRAC-p	BRAC-v	AWR	BCQ	aDICE	CQL
Maze2D	maze2d-umaze	110.4	29.0	145.6	28.6	30.4	1.7	25.2	41.5	2.2	31.7
	maze2d-medium	69.5	93.2	82.0	89.8	98.8	102.4	33.2	35.0	39.6	26.4
	maze2d-large	14.1	20.1	1.5	19.0	34.5	115.2	70.1	23.2	6.5	40.0
AntMaze	antmaze-umaze	0.0	0.7	0.0	0.7	0.5	0.7	0.6	0.8	0.0	0.7
	antmaze-umaze-diverse	0.0	0.6	0.0	0.6	0.4	0.7	0.7	0.6	0.0	0.8
	antmaze-medium-play	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.6
	antmaze-medium-diverse	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.5
	antmaze-large-play	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.2
	antmaze-large-diverse	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1
Gym	halfcheetah-random	12135.0	-17.9	3502.0	2831.4	2713.6	3590.1	36.3	-1.3	-318.0	4114.8
	walker2d-random	4592.3	73.0	192.0	336.3	-7.2	87.4	71.5	228.0	26.5	322.9
	hopper-random	3234.3	299.4	347.7	349.9	337.5	376.3	312.4	323.9	10.0	331.2
	halfcheetah-medium	12135.0	4196.4	-808.6	4897.0	5158.8	5473.8	4366.1	4767.9	-551.6	5232.1
	walker2d-medium	4592.3	304.8	44.2	2717.0	3559.9	3725.8	800.7	2441.0	15.5	2664.2
	hopper-medium	3234.3	923.5	5.7	1674.5	1044.0	990.4	1149.5	1752.4	17.5	2557.3
	halfcheetah-medium-replay	12135.0	4492.1	-581.3	4517.9	5350.8	5640.6	4727.4	4463.9	-540.8	5455.6
	walker2d-medium-replay	4592.3	518.6	87.8	883.8	-11.5	44.5	712.5	688.7	31.0	1227.3
	hopper-medium-replay	3234.3	364.4	93.3	1076.8	0.5	-0.8	904.0	1057.8	14.0	1227.3
	halfcheetah-medium-expert	12135.0	4169.4	-55.7	6349.6	5208.1	4926.6	6267.3	7750.8	-377.6	7466.9
	walker2d-medium-expert	4592.3	297.0	-5.1	1842.7	3533.1	3747.5	2469.7	2640.3	19.7	5097.3
	hopper-medium-expert	3234.3	3621.2	32.9	3113.5	42.6	5.1	862.0	3588.5	15.5	3192.0
Adroit	pen-human	739.3	1121.9	284.8	66.3	339.0	114.7	463.1	2149.0	-0.7	1214.0
	hammer-human	-248.7	-82.4	-214.2	-242.0	-239.7	-243.8	-115.3	-210.5	-234.8	300.2
	door-human	-61.8	-41.7	57.2	-66.4	-66.5	-66.4	-44.4	-56.6	-56.5	234.3
	relocate-human	-13.7	-5.6	-4.5	-18.9	-19.7	-19.7	-7.2	-8.6	-10.8	2.0
	pen-cloned	739.3	1791.8	797.6	885.4	143.4	22.2	931.3	1407.8	8.6	1264.6
	hammer-cloned	-248.7	-175.1	-244.1	-241.1	-236.7	-236.9	-226.9	-224.4	-233.1	-0.41
	door-cloned	-61.8	-60.7	-56.3	-60.9	-58.7	-59.0	-56.1	-56.3	-56.4	-44.76
	relocate-cloned	-13.7	-10.1	-16.1	-17.6	-19.8	-19.4	-16.6	-17.5	-18.8	-10.66
	pen-expert	739.3	2633.7	277.4	3254.1	-7.8	6.4	3406.0	3521.3	-6.9	3286.2
	hammer-expert	-248.7	16140.8	3019.5	16359.7	-241.4	-241.1	4822.9	13731.5	-235.2	11062.4
	door-expert	-61.8	969.4	163.8	2980.1	-66.4	-66.6	2964.5	2850.7	-56.5	2926.8
	relocate-expert	-13.7	4289.3	-18.2	4173.8	-20.6	-21.4	3875.5	1759.6	-8.7	4019.9
Flow	flow-ring-controller	25.8	-273.3	-147.7	-46.3	-188.5	-338.2	-22.6	-20.7	-136.3	-66.5
	flow-ring-random	25.8	14.7	-32.4	31.0	16.2	-16.2	-12.7	14.2	-6.8	15.1
	flow-merge-controller	375.4	359.8	354.6	436.5	392.9	422.9	441.4	361.4	533.9	450.9
	flow-merge-random	375.4	82.6	33.9	75.1	427.6	176.3	329.3	178.3	128.7	204.6
FrankaKitchen	kitchen-complete	0.0	1.4	0.6	0.0	0.0	0.0	0.0	0.3	0.0	1.8
	kitchen-partial	0.0	1.4	0.0	0.5	0.0	0.0	0.6	0.8	0.0	1.9
	kitchen-mixed	0.0	1.9	0.1	1.9	0.0	0.0	0.4	0.3	0.1	2.0
Offline CARLA	carla-lane	-8.6	324.7	-0.3	-3.0	186.1	199.6	-4.5	-1.4	-13.4	213.2
	carla-town	-79.7	-161.5	-159.9	-182.5	-231.6	-181.5	-65.8	-66.6	-400.2	-180.379

Table 7.2: The raw, un-normalized scores for each task and algorithm are reported in the table below. These scores represent the undiscounted return obtained from executing a policy in the simulator, averaged over 3 random seeds.

Chapter 8

Benchmarking Off-Policy Evaluation

In chapter 7, we presented a platform for benchmarking algorithms focused on policy optimization for offline reinforcement learning. In this chapter, we turn our focus to the related and equally important problems of policy evaluation and policy selection. Similar to the motivation for policy optimization, there are many real-world scenarios in which we need to measure the performance of a policy (a predominant application example is in recommender systems [Li et al., 2010]) or to estimate the best performing policy out of a set (such as in offline A/B testing). Moreover, off-policy evaluation is potentially a crucial component in evaluating the performance of offline RL algorithms on real-world tasks, since it can be difficult to evaluate such systems online.

Although considerable theoretical [Thomas and Brunskill, 2016, Swaminathan and Joachims, 2015a, Jiang and Li, 2016, Wang et al., 2017, Yang et al., 2020] and practical progress [Gilotte et al., 2018, Nie et al., 2020, Kalashnikov et al., 2018] on OPE algorithms has been made in a range of different domains, there are few broadly accepted evaluation tasks that combine complex, high-dimensional problems commonly explored by modern deep reinforcement learning algorithms [Bellemare et al., 2013, Brockman et al., 2016] with standardized evaluation protocols and metrics. Our goal is to provide a set of tasks with a range of difficulty, excercise a variety of design properties, and provide policies with different behavioral patterns in order to establish a standardized framework for comparing OPE algorithms. We put particular emphasis on large datasets, long-horizon tasks, and task complexity to facilitate the development of scalable algorithms that can solve high-dimensional problems.

In this chapter, we present Benchmarks for Deep Off-Policy Evaluation (DOPE), which is a benchmark targeted towards the related goals of policy evaluation and selection. These two problem settings (evaluation and selection) are primarily useful in situations when one is given a set of policies and must either estimate or rank their performance, such as in digital advertising. Like D4RL presented in chapter 7, DOPE is designed to measure the performance of OPE methods by **1)** evaluating on challenging control tasks with properties known to be difficult for OPE methods, but which occur in real-world scenarios, **2)** evaluating across a range of policies with different values, to directly measure performance on policy evaluation, ranking and selection, and **3)** evaluating in ideal and adversarial settings in terms

of dataset coverage and support. These factors are independent of task difficulty, but are known to have a large impact on OPE performance. To achieve 1, we selected tasks on a set of design principles outlined in Section 8.3. To achieve 2, for each task we include 10 to 96 policies for evaluation and devise an evaluation protocol that measures policy evaluation, ranking, and selection as outlined in Section 8.2. To achieve 3, we provide two domains with differing dataset coverage and support properties described in Section 8.4.

8.1 Relation to Prior Work

Our work is closely related to [Paine et al., 2020] which studies OPE in a similar setting, however in our work we present a benchmark for the community and compare a range of OPE methods. Voloshin et al. [2019] have also recently proposed benchmarking for OPE methods on a variety of tasks ranging from tabular problems to image-based tasks in Atari. Our work differs in several key aspects. Voloshin et al. [2019] is composed entirely of discrete action tasks, whereas out benchmark focuses on continuous action tasks. Voloshin et al. [2019] assumes full support for the evaluation policy under the behavior policy data, whereas we designed our datasets and policies to ensure that different cases of dataset and policy distributions could be studied. Finally, all evaluations in Voloshin et al. [2019] are performed using the MSE metric, and they do not provide standardized datasets. In our work, we provide a variety of policies for each problem which enables one to evaluate metrics such as ranking for policy selection, and a wide range of standardized datasets for reproducibility.

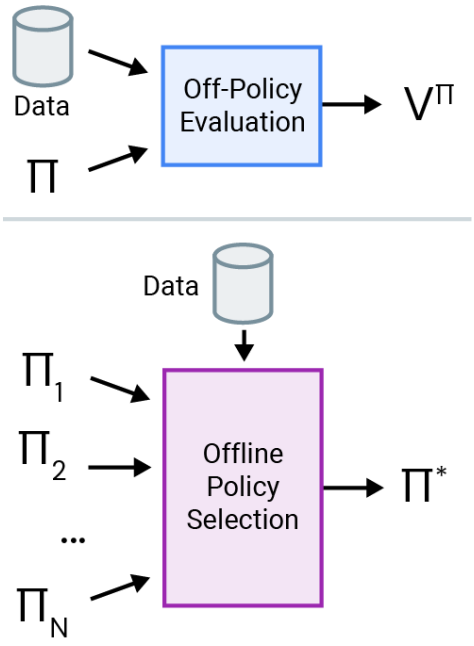


Figure 8.1: In Off-Policy Evaluation (top) the goal is to estimate the value of a single policy given only data. Offline Policy Selection (bottom) is a closely related problem: given a set of N policies, attempt to pick the best given only data.

8.2 Policy Evaluation Metrics

We consider policy evaluation in the purely offline setting. The typical setup for this problem formulation is that we are provided with a discount γ, a dataset of trajectories collected from a behavior policy $\mathcal{D} = \{(s_0, a_0, r_0, s_1, \ldots)\}$, and optionally the action probabilities for the behavior policy $\pi_B(a_t|s_t)$. In many practical applications, logging action propensities is not possible, for example, when the behavior policy is a mix of ML and hard-coded business logic. For this reason, we focus on the setting without

propensities to encourage future work on behavior-agnostic OPE methods. For the methods that require propensities, we estimate the propensities with behavior cloning.

The objective can take multiple flavors, as shown in Fig. 8.1. A common task in OPE is to estimate the performance, or value, of a policy π (which may not be the same as π_B) so that the estimated value is as close as possible to V^π under a metric such as MSE or absolute error. A second task is to perform policy selection, where the goal is to select the best policy or set of policies out of a group of candidates. This setup corresponds to how OPE is commonly used in practice, which is to find the best performing strategy out of a pool when online evaluation is too expensive to be feasible.

Evaluation Protocol

The goal of DOPE to provide metrics for policy ranking, evaluation and selection. Many existing OPE methods have only been evaluated on point estimates of value such as MSE, but policy selection is an important, practical use-case of OPE. In order to explicitly measure the quality of using OPE for policy selection, we provide a set of policies with varying value, and devise two metrics that measure how well OPE methods can rank policies.

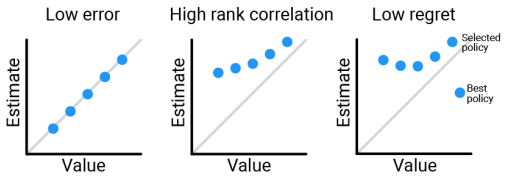


Figure 8.2: Error is a natural measure for off-policy evaluation. However for policy selection, it is sufficient to (i) rank the policies as measured by rank correlation, or (ii) select a policy with the lowest regret.

For each task we include a dataset of logged experiences $\mathcal{D}$, and a set of policies $\{\pi_1, \pi_2, ..., \pi_N\}$ with varying values. For each policy, OPE algorithms must use $\mathcal{D}$ to produce an estimate of the policy's value. For evaluation of these estimates, we provide "ground truth values" $\{V^{\pi_1}, V^{\pi_2}, ..., V^{\pi_N}\}$ that are computed by running the policy for $M \geq 1000$ episodes, where the exact value of M is given by the number of episodes needed to lower the error bar on the ground truth values to 0.666. The estimated values are then compared to these ground truth values using three different metrics encompassing both policy evaluation and selection (illustrated in Figure 8.2; see Appendix E.1 for mathematical definitions).

Absolute Error This metric measures estimate accuracy instead of its usefulness for ranking. Error is the most commonly used metric to assess performance of OPE algorithms. We opted to use absolute error instead of MSE to be robust to outliers.

Regret@k This metric measures how much worse the best policies identified by the estimates are than the best policy in the entire set. It is computed by identifying the top-k policies according to the estimated returns. Regret@k is the difference between the actual expected return of the best policy in the entire set, and the actual value of the best policy in the top-k set.

Rank correlation This metric directly measures how well estimated values rank policies, by computing the correlation between ordinal rankings according by the OPE estimates and

ordinal rankings according to the ground truth values.

8.3 Task Design Factors

We describe our motivating properties for selecting tasks for the benchmark as follows:

High Dimensional Spaces (H) High-dimensionality is a key-feature in many real-world domains where it is difficult to perform feature engineering, such as in robotics, autonomous driving, and more. In these problems, it becomes challenging to accurately estimate quantities such as the value function without the use of high-capacity models such a neural networks and large datasets with wide state coverage. Our benchmark contains complex continuous-space tasks which exercise these challenges.

Long Time-Horizon (L) Long time horizon tasks are known to present difficult challenges for OPE algorithms. Some algorithms have difficulty doing credit assignment for these tasks. This can be made worse as the state dimension or action dimension increases.

Sparse Rewards (R) Sparse reward tasks increase the difficulty of credit assignment and add exploration challenges, which may interact with data coverage in the offline setting. We include a range robotics and navigation tasks which are difficult to solve due to reward sparsity.

Temporally extended control (T) The ability to make decisions hierarchically is major challenge in many reinforcement learning applications. We include two navigation tasks which require high-level planning in addition to low-level control in order to simulate the difficulty in such problems.

8.4 Tasks

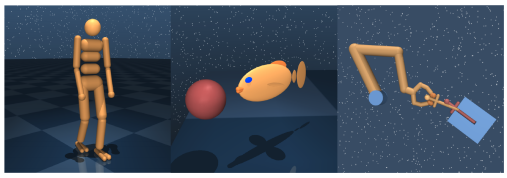
DOPE contains two domains designed to provide a more comprehensive picture of how well OPE methods perform in different settings. These two domains are constructed using two benchmarks previously proposed for offline reinforcement learning: RL Unplugged [Gulcehre et al., 2020] and D4RL (presented in chapter 7), and reflect the challenges found within them.

The **DOPE RL Unplugged** domain is constrained in two important ways: 1) the data is always generated using online RL training, ensuring there is adequate coverage of the state-action space, and 2) the policies are generated by applying offline RL algorithms to the same dataset we use for evaluation, ensuring that the behavior policy and evaluation policies induce similar state-action distributions. Using it, we hope to understand how OPE methods work as task complexity increases from simple Cartpole tasks to controlling a Humanoid body while controlling for ideal data.

On the other hand, the **DOPE D4RL** domain has: 1) data from various sources (including random exploration, human teleoperation, and RL-trained policies with limited exploration), which results in varying levels of coverage of the state-action space, and 2) policies

that are generated using online RL algorithms, making it less likely that the behavior and evaluation policies share similar induced state-action distributions. Both of these result in distribution shift which is known to be challenging for OPE methods, even in simple tasks. So, using it we hope to measure how well OPE methods work in more practical data settings.

DOPE RL Unplugged



DeepMind Control Suite (Tassa et al., 2018) is a set of control tasks implemented in MuJoCo (Todorov et al., 2012). We consider the subset included in RL Unplugged. This subset includes tasks that cover a range of difficulties. From Cartpole swingup, a simple task with a single degree of freedom, to Humanoid run which involves control of a complex bodies with 21 degrees of freedom. All tasks use the default feature representation of the system state, including proprioceptive information such as joint positions and velocity, and additional sensor information and target position where appropriate. The observation dimension ranges from 5 to 67.

Datasets and policies We train four offline RL algorithms (D4PG [Barth-Maron et al., 2018], ABM [Siegel et al., 2020], CRR [Wang et al., 2020] and behavior cloning), varying their hyperparameters. For each algorithm-task-hyperparameter combination, we train an agent with 3 random seeds on the DM Control Suite dataset from RL Unplugged and record policy snapshots at exponentially increasing intervals (after 25k learner steps, 50k, 100K, 200K, etc). Following Gulcehre et al. [2020], we consider a deterministic policy for D4PG and stochastic policies for BC, ABM and CRR. The datasets are taken from the RL Unplugged benchmark, where they were created by training multiple (online) RL agents and collecting both successful and unsuccessful episodes throughout training. All offline RL algorithms are implemented using the Acme framework [Hoffman et al., 2020].

DOPE D4RL

As part of the DOPE benchmark, we use a subset of the D4RL tasks presented in chapter 7.

Gym-MuJoCo tasks. Gym-MuJoCo consists of several continuous control tasks implemented within the MuJoCo simulator [Todorov et al., 2012] and provided in the OpenAI Gym [Brockman et al., 2016] benchmark for online RL. We include the HalfCheetah, Hopper, Walker2D, and Ant tasks. We include this domain primarily for comparison with past works, as a vast array of popular RL methods have been evaluated and developed on these tasks [Schulman et al., 2015, Lillicrap et al., 2016, Schulman et al., 2017, Fujimoto et al., 2018, Haarnoja et al., 2018].

Gym-MuJoCo datasets and policies. For each task, in order to explore the effect of varying distributions, we include 5 datasets originally proposed by Fu et al. [2020]. 3 correspond to different performance levels of the agent – "random", "medium", and "expert". We

Statistics	cartpole swingup	cheetah run	finger turn hard	fish swim	humanoid run	walker stand	walker walk	manipulator insert ball	manipulator insert peg
Dataset size	40K	300K	500K	200K	3M	200K	200K	1.5M	1.5M
State dim.	5	17	12	24	67	24	24	44	44
Action dim.	1	6	2	5	21	6	6	5	5
Properties	-	H, L	H, L	H, L	H, L	H, L	H, L	H, L, T	H, L,T

Statistics	maze2d	antmaze	halfcheetah	hopper	walker	ant	hammer	door	relocate	pen
Dataset size	1/2/4M	1M	1M	1M	1M	1M	11K/1M	7K/1M	10K/1M	5K/500K
# datasets	1	1	5	5	5	5	3	3	3	3
State dim.	4	29	17	11	17	111	46	39	39	45
Action dim.	2	8	6	3	6	8	26	28	30	24
Properties	T	T, R	H	H	H	H	H, R	H, R	H, R	H, R

Table 8.1: Task statistics for RLUnplugged tasks (top) and D4RL tasks (bottom). Dataset size is the number of (s, a, r, s') tuples. For each dataset, we note the properties it possesses: high dimensional spaces (**H**), long time-horizon (**L**), sparse rewards (**R**), temporally extended control (**T**).

additionally include a mixture of medium and expert dataset, labeled "medium-expert", and data collected from a replay buffer until the policy reaches the medium level of performance, labeled "medium-replay". For policies, we selected 11 policies collected from evenly-spaced snapshots of training a Soft Actor-Critic agent [Haarnoja et al., 2018], which covers a range of performance between random and expert.

Maze2D and AntMaze tasks. Maze2D and AntMaze are two maze navigation tasks originally proposed in D4RL [Fu et al., 2020]. The domain consists of 3 mazes ranging from easy to hard ("umaze", "medium", "large"), and two morphologies: a 2D ball in Maze2D and the "Ant" robot of the Gym benchmark in AntMaze. For Maze2D, we provide a less challenging reward computed base on distance to a fixed goal. For the AntMaze environment reward is given only upon reaching the fixed goal.

Maze2D and AntMaze datasets and policies. Datasets for both morphologies consists of undirect data navigating randomly to different goal locations. The datasets for Maze2D are collected by using a high-level planner to command waypoints to a low-level PID controller in order to reach randomly selected goals. The dataset in AntMaze is generated using the same high-level planner, but the low-level planner is replaced with a goal-conditioned policy trained to reach arbitrary waypoints. Both of these datasets are generated from non-Markovian policies, as the high-level controller maintains a history of waypoints reached in order to construct a plan to the goal. We provide policies for all environments except "antmaze-large" by taking training snapshots obtained while running the DAPG algorithm [Rajeswaran et al., 2018]. Because obtaining high-performing policies

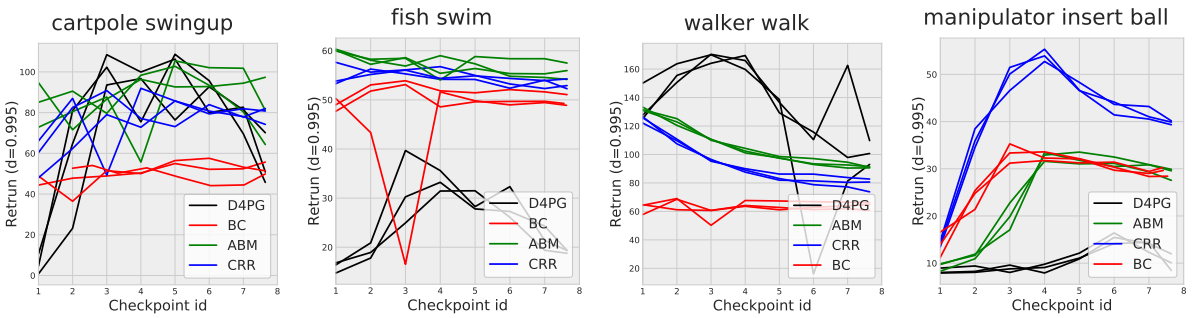


Figure 8.3: Online evaluation of policy checkpoints for 4 Offline RL algorithms with 3 random seeds. We observe a large degree of variability between the behavior of algorithms on different tasks. Without online evaluation, tuning the hyperparameters (e.g., choice of Offline RL algorithm and policy checkpoint) is challenging. This highlights the practical importance of Offline policy selection when online evaluation is not feasible. See Figure E.7 for additional tasks.

for "antmaze-large" was challenging, we instead used imitation learning on a large amount of expert data to generate evaluation policies. This expert data is obtained by collecting additional trajectories that reach the goal using a high-level waypoint planner in conjunction with a low-level goal-conditioned policy (this is the same method as was used to generate the dataset, Sec. 5 [Fu et al., 2020]).

Adroit tasks. The Adroit domain is a realistic simulation based on the Shadow Hand robot, first proposed by Rajeswaran et al. [2018]. There are 4 tasks in this domain: opening a door ("door"), pen twirling ("pen"), moving a ball to a target location ("relocate"), and hitting a nail with a hammer ("hammer"). These tasks all contain sparse rewards and are difficult to learn without demonstrations.

Adroit datasets and policies. We include 3 datasets for each task. The "human" dataset consists of a small amount of human demonstrations performing the task. The "expert" dataset consists of data collected from an expert trained via DAPG [Rajeswaran et al., 2018]. Finally, the "cloned" dataset contains a mixture of human demonstrations and data collected from an imitation learning algorithm trained on the demonstrations. For policies, we include 11 policies collected from snapshots while running the DAPG algorithm, which range from random performance to expert performance.

8.5 Evaluation

The goal of our evaluation is two-fold. First, we wish to measure the performance of a variety of existing algorithms to provide baselines and reference numbers for future research. Second, we wish to identify shortcomings in these approaches to reveal promising directions for future research.

Baselines

We selected six methods to evaluate, which cover a variety of approaches that have been explored for the OPE problem.

Fitted Q-Evaluation (FQE) As in Le et al. [2019], we train a neural network to estimate the value of the evaluation policy π by bootstrapping from $Q(s', \pi(s'))$. We tried two different implementations, one from Kostrikov and Nachum [2020][1] and another from Paine et al. [2020] labeled FQE-L2 and FQE-D respectively to reflect different choices in loss function and parameterization.

Model-Based (MB) Similar to Paduraru [2007], we train dynamics and reward models on transitions from the offline dataset $\mathcal{D}$. Our models are deep neural networks trained to maximize the log likelihood of the next state and reward given the current state and action, similar to models from successful model-based RL algorithms [Chua et al., 2018, Janner et al., 2019]. We follow the setup detailed in Zhang et al. [2021]. We include both the feed-forward and auto-regressive models labeled MB-FF and MB-AR respectively. To evaluate a policy, we compute the return using simulated trajectories generated by the policy under the learned dynamics model.

Importance Sampling (IS) We perform importance sampling with a learned behavior policy. We use the implementation from Kostrikov and Nachum [2020][3], which uses self-normalized (also known as weighted) step-wise importance sampling [Precup et al., 2000]. Since the behavior policy is not known explicitly, we learn an estimate of it via a max-likelihood objective over the dataset $\mathcal{D}$, as advocated by Xie et al. [2019], Hanna et al. [2019]. In order to be able to compute log-probabilities when the target policy is deterministic, we add artificial Gaussian noise with standard deviation 0.01 for all deterministic target policies.

Doubly-Robust (DR) We perform weighted doubly-robust policy evaluation Thomas and Brunskill [2016] using the implementation of Kostrikov and Nachum [2020][3]. Specifically, this method combines the IS technique above with a value estimator for variance reduction. The value estimator is learned using deep FQE with an L2 loss function. More advanced approaches that trade variance for bias exist (e.g., MAGIC [Thomas and Brunskill, 2016]), but we leave implementing them to future work.

DICE This method uses a saddle-point objective to estimate marginalized importance weights $d^\pi(s,a)/d^{\pi_B}(s,a)$; these weights are then used to compute a weighted average of reward over the offline dataset, and this serves as an estimate of the policy's value in the MDP. We use the implementation from Yang et al. [2020] corresponding to the algorithm *BestDICE*.[2]

Variational Power Method (VPM) This method runs a variational power iteration algorithm to estimate the importance weights $d^\pi(s,a)/d^{\pi_B}(s,a)$ without the knowledge of the behavior policy. It then estimates the target policy value using weighted average of rewards similar to the DICE method. Our implementation is based on the same network

[1] Code available at `https://github.com/google-research/google-research/tree/master/policy_eval`.

[2] Code available at `https://github.com/google-research/dice_rl`.

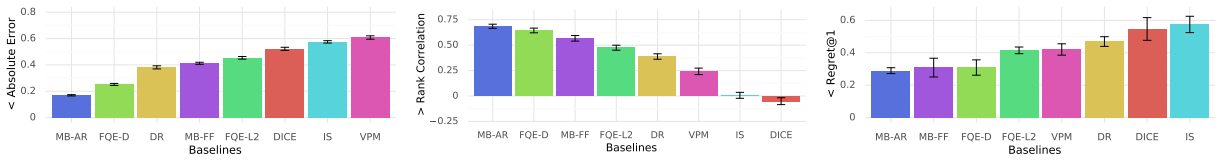


Figure 8.4: **DOPE RL Unplugged** Mean overall performance of baselines.

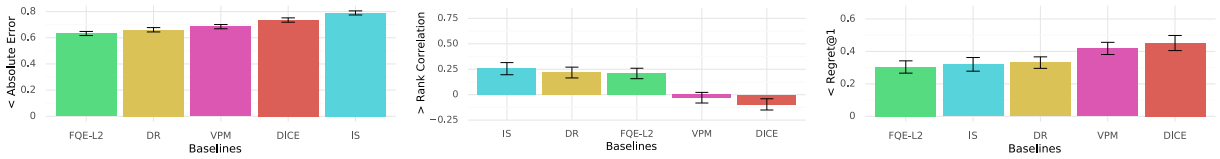


Figure 8.5: **DOPE D4RL** Mean overall performance of baselines.

and hyperparameters for OPE setting as in Wen et al. [2020]. We further tune the hyperparameters including the regularization parameter λ, learning rates α_θ and α_v, and number of iterations on the Cartpole swingup task using ground-truth policy value, and then fix them for all other tasks.

Results

To facilitate aggregate metrics and comparisons between tasks and between DOPE RL Unplugged and DOPE D4RL, we normalize the returns and estimated returns to range between 0 and 1. For each set of policies we compute the worst value $V_{worst} = min\{V^{\pi_1}, V^{\pi_2}, ..., V^{\pi_N}\}$ and best value $V_{best} = max\{V^{\pi_1}, V^{\pi_2}, ..., V^{\pi_N}\}$ and normalize the returns and estimated returns according to $x' = (x - V_{worst})/(V_{best} - V_{worst})$.

We present results averaged across DOPE RL Unplugged in Fig. 8.4, and results for DOPE D4RL in Fig. 8.5. Overall, no evaluated algorithm attains near-oracle performance under any metric (absolute error, regret, or rank correlation). Because the dataset is finite, we do not expect that achieving oracle performance is possible. Nevertheless, based on recent progress on this benchmark (e.g., Zhang et al. [2021]), we hypothesize that the benchmark has room for improvement, making it suitable for driving further improvements on OPE methods and facilitating the development of OPE algorithms that can provide reliable estimates on the types of high-dimensional problems that we consider.

While all algorithms achieve sub-optimal performance, some perform better than others. We find that on the DOPE RL Unplugged tasks model based (MB-AR, MB-FF) and direct value based methods (FQE-D, FQE-L2) significantly outperform importance sampling methods (VPM, DICE, IS) across all metrics. This is somewhat surprising as DICE and VPM have shown promising results in other settings. We hypothesize that this is due to the relationship between the behavior data and evaluation policies, which is different from

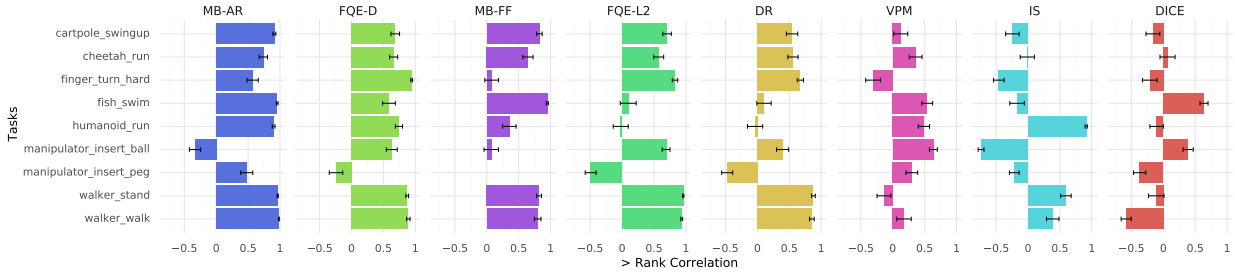


Figure 8.6: Rank correlation for each baseline algorithm for each RL Unplugged task considered.

standard OPE settings. Recall that in DOPE RL Unplugged the behavior data is collected from an online RL algorithm and the evaluation policies are learned via offline RL from the behavior data. In our experience all methods work better when the behavior policy is a noisy/perturbed version of the evaluation policy. Moreover, MB and FQE-based methods may implicitly benefit from the architectural and optimization advancements made in policy optimization settings, which focus on similar environments and where these methods are more popular than importance sampling approaches. Note that within the MB and FQE methods, design details can create a significant difference in performance. For example model architecture (MB-AR vs MB-FF) and implementation differences (FQE-D vs FQE-L2) show differing performance on certain tasks.

On DOPE D4RL, direct value based methods still do well, with FQE-L2 performing best on the Absolute Error and Regret@1 metrics. However, there are cases where other methods outperform FQE. Notably, IS and DR outperform FQE-L2 under the rank correlation metric. As expected, there is a clear performance gap between DOPE RL Unplugged and DOPE D4RL. While both domains have challenging tasks, algorithms perform better under the more ideal conditions of DOPE RL Unplugged than under the challenging conditions of DOPE D4RL (0.69 vs 0.25 rank correlation respectively).

In Fig. 8.6 we show the rank correlation for each task in DOPE RL Unplugged. Most tasks follow the overall trends, but we will highlight a few exceptions. 1) Importance sampling is among the best methods for the *humanoid run* task, significantly outperforming direct value-based methods. 2) while MB-AR and FQE-D are similar overall, there are a few tasks where the difference is large, for example FQE-D outperfroms MB-AR on *finger turn hard*, and *manipulator insert ball*, where as MB-AR outperforms FQE-D on *cartpole swingup*, *fish swim*, *humanoid run*, and *manipulator insert peg*. We show the scatter plots for MB-AR and FQE-D on these tasks in Fig 8.7 which highlights different failure modes: when MB-AR performs worse, it assigns similar values for all policies; when FQE-D performs worse, it severely over-estimates the values of poor policies.

We present more detailed results, separated by task, in Appendix E.2. Note in particular how in Table E.4, which shows the regret@1 metric for different D4RL tasks, the particular

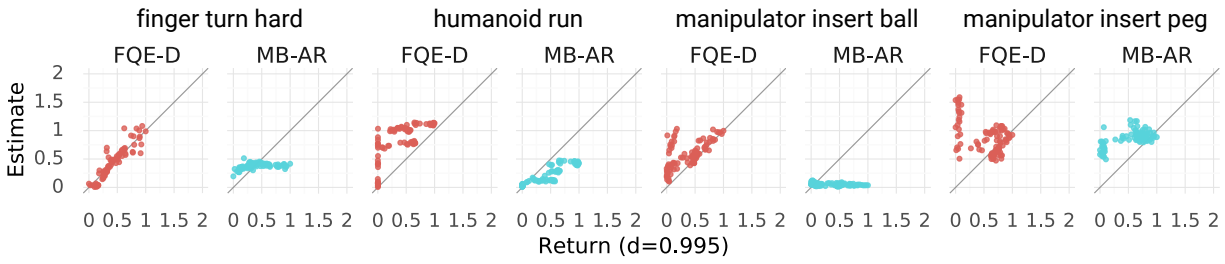


Figure 8.7: Scatter plots of estimate vs ground truth return for MB-AR and FQE-D on selected tasks.

choice of dataset for the Gym-MuJoCo, Adroit, and AntMaze domains causes a significant difference in the performance of OPE methods. This indicates the importance of evaluating multiple distinct datasets, with different data distribution properties (e.g., more narrow datasets, such as expert data, vs. broader datasets, such as random data), as no tested method is reliably robust to the effects of dataset variation.

High-dimensional tasks requiring temporally extended control were also challenging, as highlighted by the performance on the AntMaze domain. No algorithm was able to achieve a good absolute error value on such tasks, and importance sampling was the only method able to achieve a correlation consistently above zero, suggesting that these more complex tasks are a particularly important area for future methods to focus on.

8.6 Discussion

In this chapter we presented the Deep Off-Policy Evaluation (DOPE) benchmark, which aims to provide a platform for studying policy evaluation and selection across a wide range of challenging tasks and datasets. In contrast to prior benchmarks, DOPE provides multiple datasets and policies, allowing researchers to study how data distributions affect performance and to evaluate a wide variety of metrics, including those that are relevant for offline policy selection. In comparing existing OPE methods, we find that no existing algorithms consistently perform well across all of the tasks, which further reinforces the importance of standardized and challenging OPE benchmarks. Moreover, algorithms that perform poorly under one metric, such as absolute error, may perform better on other metrics, such as correlation, which provides insight into what algorithms to use depending on the use case (e.g., policy evaluation vs. policy selection).

Chapter 9

Conclusion

In this thesis, we explored the challenge of learning intelligent decision making systems from offline data. This represents a fundamental paradigm shift from many commonly used frameworks for decision making such as reinforcement learning which require online interaction. We explored offline learning in two settings - reinforcement learning for sequential decision making problems, and model-based optimization for problems with more generic structure. In reinforcement learning, we explored algorithms in two popular settings: model-free Q-learning methods and model-based methods. In both of these settings, we theoretically analyzed the effect of distribution shift due to policy optimization, and proposed practical algorithms to mitigate model exploitation. In the model-based optimization setting, we turned to uncertainty-aware models and utilized the normalized maximum likelihood framework, showing how it could devise novel designs in application areas ranging from drug design to robotics.

However, there are limitations to the proposed methods. All of the proposed algorithms fall under the umbrella of enforcing robustness. We assume that learned models are inherently vulnerable to distribution shift, and construct our algorithms in a way to mitigate out-of-distribution inputs. While this general approach can be effective from a safety standpoint, it can also be very limiting and result in overly pessimistic algorithms and analysis. An alternative to building robust algorithms is to build robustness into the models themselves. For example, we can leverage underlying structure in the problems to identify when models can extrapolate accurately, or train models using modern techniques that limit the effectiveness of adversarial attacks.

A natural trend in machine learning has been that larger and larger datasets have been collected to tackle more challenging problems. We believe that offline learning is the key to making the leap from solving prediction problems to decision making problems, and hope that the ideas presented in this thesis can provide a path towards building robust methods that can be applied safely and inexpensively on real-world challenges.

Bibliography

A. Agarwal, S. M. Kakade, J. D. Lee, and G. Mahajan. Optimality and approximation with policy gradient methods in markov decision processes. In *Conference on Learning Theory*, 2020a.

R. Agarwal, D. Schuurmans, and M. Norouzi. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, 2020b.

A. Antos, C. Szepesvari, and R. Munos. Value-iteration based fitted policy iteration: Learning with a single trajectory. In *IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, 2007.

A. Antos, C. Szepesvári, and R. Munos. Fitted q-iteration in continuous action-space mdps. In *Advances in Neural Information Processing Systems*, 2008.

K. Asadi, D. Misra, and M. Littman. Lipschitz continuity in model-based reinforcement learning. In *International Conference on Machine Learning*, 2018.

C. G. Atkeson and S. Schaal. Learning tasks from a single demonstration. In *International Conference on Robotics and Automation*, 1997.

A. Barron, J. Rissanen, and B. Yu. The minimum description length principle in coding and modeling. *IEEE Transactions on Information Theory*, 44(6):2743–2760, 1998.

G. Barth-Maron, M. W. Hoffman, D. Budden, W. Dabney, D. Horgan, D. TB, A. Muldal, N. Heess, and T. Lillicrap. Distributional policy gradients. In *International Conference on Learning Representations*, 2018.

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47: 253–279, 2013.

E. Bengoetxea, P. Larrañaga, I. Bloch, and A. Perchant. Estimation of distribution algorithms: A new evolutionary computation approach for graph matching problems. In *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 454–469. Springer, 2001.

D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-dynamic programming*. Athena Scientific, 1996.

K. Bibas, Y. Fogel, and M. Feder. Deep pnml: Predictive normalized maximum likelihood for deep neural networks. *arXiv preprint arXiv:1904.12286*, 2019.

L. Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

D. Brookes, H. Park, and J. Listgarten. Conditioning by adaptive sampling for robust design. In *International Conference on Machine Learning*, 2019.

T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

J. Buckman, D. Hafner, G. Tucker, E. Brevdo, and H. Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. In *Advances in Neural Information Processing Systems*, 2018.

J. Byrd and Z. Lipton. What is the effect of importance weighting in deep learning? In *International Conference on Machine Learning*, 2019.

S. Cabi, S. G. Colmenarejo, A. Novikov, K. Konyushkova, S. Reed, R. Jeong, K. Żołna, Y. Aytar, D. Budden, M. Vecerik, et al. A framework for data-driven robotics. *arXiv preprint arXiv:1909.12200*, 2019.

N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko. End-to-end object detection with transformers. In *European Conference on Computer Vision*, pages 213–229. Springer, 2020.

K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, 2018.

I. Clavera, J. Rothfuss, J. Schulman, Y. Fujita, T. Asfour, and P. Abbeel. Model-based reinforcement learning via meta-policy optimization. In *Conference on Robot Learning*, 2018.

F. Codevilla, M. Miiller, A. López, V. Koltun, and A. Dosovitskiy. End-to-end driving via conditional imitation learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–9. IEEE, 2018.

T. De Bruin, J. Kober, K. Tuyls, and R. Babuška. The importance of experience replay database composition in deep reinforcement learning. In *Deep reinforcement learning workshop at Advances in Neural Information Processing Systems*, 2015.

S. Dean, H. Mania, N. Matni, B. Recht, and S. Tu. On the sample complexity of the linear quadratic regulator. *Foundations of Computational Mathematics*, 20(4):633–679, 2020.

M. Deisenroth and C. E. Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *International Conference on Machine Learning*, 2011.

S. Depeweg, J. M. Hernández-Lobato, F. Doshi-Velez, and S. Udluft. Learning and policy search in stochastic dynamical systems with bayesian neural networks. In *International Conference on Learning Representations*, 2016.

S. Doroudi, P. S. Thomas, and E. Brunskill. Importance sampling for fair policy selection. In *International Joint Conference on Artificial Intelligence*, 2018.

A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. CARLA: An open urban driving simulator. In *Conference on Robot Learning*, 2017.

A. Draeger, S. Engell, and H. Ranke. Model predictive control using neural networks. *IEEE Control Systems Magazine*, 1995.

S. S. Du, S. M. Kakade, R. Wang, and L. F. Yang. Is a good representation sufficient for sample efficient reinforcement learning? In *International Conference on Learning Representations*, 2020.

Y. Du and K. Narasimhan. Task-agnostic dynamics priors for deep reinforcement learning. In *International Conference on Machine Learning*, 2019.

M. Dudík, D. Erhan, J. Langford, L. Li, et al. Doubly robust policy evaluation and optimization. *Statistical Science*, 29(4):485–511, 2014.

G. Dulac-Arnold, N. Levine, D. J. Mankowitz, J. Li, C. Paduraru, S. Gowal, and T. Hester. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, pages 1–50, 2021.

F. Ebert, C. Finn, S. Dasari, A. Xie, A. X. Lee, and S. Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018.

D. Ernst, P. Geurts, and L. Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(Apr):503–556, 2005.

L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, 2018.

C. Fannjiang and J. Listgarten. Autofocused oracles for model-based design. *Advances in Neural Information Processing Systems*, 2020.

A.-m. Farahmand, C. Szepesvári, and R. Munos. Error propagation for approximate policy and value iteration. In *Advances in Neural Information Processing Systems*, 2010.

A.-M. Farahmand, A. Barreto, and D. Nikovski. Value-aware loss function for model-based reinforcement learning. In *International Conference on Artificial Intelligence and Statistics*, 2017.

V. Feinberg, A. Wan, I. Stoica, M. I. Jordan, J. E. Gonzalez, and S. Levine. Model-based value estimation for efficient model-free reinforcement learning. In *International Conference on Machine Learning*, 2018.

Y. Fogel and M. Feder. Universal supervised learning for individual data. *arXiv preprint arXiv:1812.09520*, 2018.

J. Fu, A. Kumar, M. Soh, and S. Levine. Diagnosing bottlenecks in deep q-learning algorithms. In *International Conference on Machine Learning*, 2019.

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

S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, 2018.

S. Fujimoto, E. Conti, M. Ghavamzadeh, and J. Pineau. Benchmarking batch deep reinforcement learning algorithms. *arXiv preprint arXiv:1910.01708*, 2019a.

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

Y. Gal, R. McAllister, and C. E. Rasmussen. Improving PILCO with Bayesian neural network dynamics models. In *ICML Workshop on Data-Efficient Machine Learning Workshop*, 2016.

Y. Gao, H. Xu, J. Lin, F. Yu, S. Levine, and T. Darrell. Reinforcement learning from imperfect demonstrations. *arXiv preprint arXiv:1802.05313*, 2018.

M. Garnelo, D. Rosenbaum, C. Maddison, T. Ramalho, D. Saxton, M. Shanahan, Y. W. Teh, D. Rezende, and S. A. Eslami. Conditional neural processes. In *International Conference on Machine Learning*, 2018.

A. Gaulton, L. J. Bellis, A. P. Bento, J. Chambers, M. Davies, A. Hersey, Y. Light, S. McGlinchey, D. Michalovich, B. Al-Lazikani, and J. P. Overington. ChEMBL: a large-scale bioactivity database for drug discovery. *Nucleic acids research*, 40:D1100–D1107, Jan 2012. ISSN 1362-4962.

C. Gelada and M. G. Bellemare. Off-policy deep reinforcement learning by bootstrapping the covariate shift. In *AAAI Conference on Artificial Intelligence*, 2019.

A. Gilotte, C. Calauzènes, T. Nedelec, A. Abraham, and S. Dollé. Offline a/b testing for recommender systems. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 198–206, 2018.

J. Grau-Moya, F. Leibfried, and P. Vrancx. Soft q-learning with mutual-information regularization. In *International Conference on Learning Representations*, 2018.

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

S. Gu, T. Lillicrap, I. Sutskever, and S. Levine. Continuous deep Q-learning with model-based acceleration. In *International Conference on Machine Learning*, 2016.

S. Gu, E. Holly, T. Lillicrap, and S. Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *IEEE International Conference on Robotics and Automation*, 2017.

C. Gulcehre, Z. Wang, A. Novikov, T. L. Paine, S. G. Colmenarejo, K. Zolna, R. Agarwal, J. Merel, D. Mankowitz, C. Paduraru, G. Dulac-Arnold, J. Li, M. Norouzi, M. Hoffman, O. Nachum, G. Tucker, N. Heess, and N. de Freitas. Rl unplugged: Benchmarks for offline reinforcement learning, 2020.

A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. In *Conference on Robot Learning*, 2020.

T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, 2018.

K. Hamidieh. A data-driven statistical model for predicting the critical temperature of a superconductor. *Computational Materials Science*, 154:346 – 354, 2018.

J. Hanna, S. Niekum, and P. Stone. Importance sampling policy evaluation with an estimated behavior policy. In *International Conference on Machine Learning*, 2019.

G. Hautier, C. C. Fischer, A. Jain, T. Mueller, and G. Ceder. Finding nature's missing ternary oxide compounds using machine learning and density functional theory. *Chemistry of Materials*, 22(12):3762–3767, 2010.

K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask R-CNN. In *International Conference on Computer Vision*, 2017.

N. Heess, G. Wayne, D. Silver, T. Lillicrap, Y. Tassa, and T. Erez. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, 2015.

D. Hein, S. Udluft, M. Tokic, A. Hentschel, T. A. Runkler, and V. Sterzing. Batch reinforcement learning on the industrial benchmark: First experiences. In *International Joint Conference on Neural Networks*, 2017.

J. Henderson, O. Lemon, and K. Georgila. Hybrid reinforcement/supervised learning of dialogue policies from fixed data sets. *Computational Linguistics*, 34(4):487–511, 2008.

T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband, et al. Deep q-learning from demonstrations. In *AAAI Conference on Artificial Intelligence*, 2018.

W. Hoburg and P. Abbeel. Geometric programming for aircraft design optimization. *AIAA Journal*, 52(11):2414–2426, 2014.

M. Hoffman, B. Shahriari, J. Aslanides, G. Barth-Maron, F. Behbahani, T. Norman, A. Abdolmaleki, A. Cassirer, F. Yang, K. Baumli, et al. Acme: A research framework for distributed reinforcement learning. *arXiv preprint arXiv:2006.00979*, 2020.

G. Z. Holland, E. J. Talvitie, and M. Bowling. The effect of planning shape on dyna-style planning in high-dimensional state spaces. *arXiv preprint arXiv:1806.01825*, 2018.

E. Ie, C.-w. Hsu, M. Mladenov, V. Jain, S. Narvekar, J. Wang, R. Wu, and C. Boutilier. Recsim: A configurable simulation platform for recommender systems. *arXiv preprint arXiv:1909.04847*, 2019.

A. Irpan, K. Rao, K. Bousmalis, C. Harris, J. Ibarz, and S. Levine. Off-policy evaluation via off-policy classification. In *Advances in Neural Information Processing Systems*, 2019.

M. Janner, J. Fu, M. Zhang, and S. Levine. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, 2019.

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

N. Jiang and L. Li. Doubly robust off-policy value evaluation for reinforcement learning. In *International Conference on Machine Learning*, 2016.

T. Joachims, A. Swaminathan, and M. de Rijke. Deep learning with logged bandit feedback. In *International Conference on Learning Representations*, 2018.

L. P. Kaelbling, M. L. Littman, and A. P. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowsi, S. Levine, R. Sepassi, G. Tucker, and H. Michalewski. Model-based reinforcement learning for Atari. In *International Conference on Learning Representations*, 2020.

S. Kakade and J. Langford. Approximately optimal approximate reinforcement learning. In *International Conference on Machine Learning*, 2002.

D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, et al. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning*, 2018.

G. Kalweit and J. Boedecker. Uncertainty-driven imagination for continuous deep reinforcement learning. In *Conference on Robot Learning*, 2017.

S. Karita, N. Chen, T. Hayashi, T. Hori, H. Inaguma, Z. Jiang, M. Someki, N. E. Y. Soplin, R. Yamamoto, X. Wang, et al. A comparative study on transformer vs rnn in speech applications. In *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 449–456. IEEE, 2019.

R. Kidambi, A. Rajeswaran, P. Netrapalli, and T. Joachims. Morel: Model-based offline reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020.

N. Killoran, L. J. Lee, A. Delong, D. Duvenaud, and B. J. Frey. Generating and designing dna with deep generative models. *arXiv preprint arXiv:1712.06148*, 2017.

H. Kim, A. Mnih, J. Schwarz, M. Garnelo, A. Eslami, D. Rosenbaum, O. Vinyals, and Y. W. Teh. Attentive neural processes. In *International Conference on Learning Representations*, 2018.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

I. Kostrikov and O. Nachum. Statistical bootstrapping for uncertainty estimation in off-policy evaluation. *arXiv preprint arXiv:2007.13609*, 2020.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012.

A. Kumar and S. Levine. Model inversion networks for model-based optimization. *Advances in Neural Information Processing Systems*, 2020.

A. Kumar, J. Fu, G. Tucker, and S. Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. In *Advances in Neural Information Processing Systems*, 2019.

A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative q-learning for offline reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020.

V. Kumar, E. Todorov, and S. Levine. Optimal control with learned local models: Application to dexterous manipulation. In *International Conference on Robotics and Automation*, 2016.

T. Kurutach, I. Clavera, Y. Duan, A. Tamar, and P. Abbeel. Model-ensemble trust-region policy optimization. In *International Conference on Learning Representations*, 2018.

R. Laroche, P. Trichelair, and R. T. Des Combes. Safe policy improvement with baseline bootstrapping. In *International Conference on Machine Learning*, 2019.

H. Le, C. Voloshin, and Y. Yue. Batch policy learning under constraints. In *International Conference on Machine Learning*, 2019.

S. Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.

S. Levine and V. Koltun. Guided policy search. In *International Conference on Machine Learning*, 2013.

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

L. Li, W. Chu, J. Langford, and R. E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *International Conference on World Wide Web*, pages 661–670, 2010.

T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations*, 2016.

L.-J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

Y. Luo, H. Xu, Y. Li, Y. Tian, T. Darrell, and T. Ma. Algorithmic framework for model-based deep reinforcement learning with theoretical guarantees. In *International Conference on Learning Representations*, 2019.

A. R. Mahmood, H. Yu, M. White, and R. S. Sutton. Emphatic temporal-difference learning. *arXiv preprint arXiv:1507.01569*, 2015.

A. Mansouri Tehrani, A. O. Oliynyk, M. Parry, Z. Rizvi, S. Couper, F. Lin, L. Miyagi, T. D. Sparks, and J. Brgoch. Machine learning directed search for ultraincompressible, superhard materials. *Journal of the American Chemical Society*, 140(31):9844–9853, 2018.

M. Migliavacca, A. Pecorino, M. Pirotta, M. Restelli, and A. Bonarini. Fitted policy search: Direct policy search using a batch reinforcement learning approach. In *3rd International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems (ERLARS 2010)*, page 35. Citeseer, 2010.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2015.

K. Mo, H. Li, Z. Lin, and J.-Y. Lee. The adobeindoornav dataset: Towards deep reinforcement learning based real-world indoor robot visual navigation. *arXiv preprint arXiv:1802.08824*, 2018.

R. Munos. Error bounds for approximate policy iteration. In *International Conference on Machine Learning*, 2003.

R. Munos. Error bounds for approximate value iteration. In *National Conference on Artificial Intelligence*, 2005.

R. Munos, T. Stepleton, A. Harutyunyan, and M. Bellemare. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, 2016.

O. Nachum, B. Dai, I. Kostrikov, Y. Chow, L. Li, and D. Schuurmans. Algaedice: Policy gradient from arbitrary experience. *arXiv preprint arXiv:1912.02074*, 2019.

A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *International Conference on Robotics and Automation*, 2018.

A. Nair, M. Dalal, A. Gupta, and S. Levine. Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

X. Nie, E. Brunskill, and S. Wager. Learning when-to-treat policies. *Journal of the American Statistical Association*, pages 1–18, 2020.

J. Oh, X. Guo, H. Lee, R. Lewis, and S. Singh. Action-conditional video prediction using deep networks in Atari games. In *Advances in Neural Information Processing Systems*, 2015.

J. Oh, S. Singh, and H. Lee. Value prediction network. In *Advances in Neural Information Processing Systems*, 2017.

C. Paduraru. Planning with approximate and learned models of markov decision processes. *MsC Thesis, University of Alberta*, 2007.

T. L. Paine, C. Paduraru, A. Michi, C. Gulcehre, K. Zolna, A. Novikov, Z. Wang, and N. de Freitas. Hyperparameter selection for offline reinforcement learning. *arXiv preprint arXiv:2007.09055*, 2020.

X. B. Peng, A. Kumar, G. Zhang, and S. Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.

J. Peters and S. Schaal. Policy gradient methods for robotics. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006.

J. Peters and S. Schaal. Reinforcement learning by reward-weighted regression for operational space control. In *International Conference on Machine learning*, 2007.

A. Piché, V. Thomas, C. Ibrahim, Y. Bengio, and C. Pal. Probabilistic planning with sequential Monte Carlo methods. In *International Conference on Learning Representations*, 2019.

O. Pietquin, M. Geist, S. Chandramohan, and H. Frezza-Buet. Sample-efficient batch reinforcement learning for dialogue management optimization. *ACM Transactions on Speech and Language Processing (TSLP)*, 7(3):1–21, 2011.

M. Popova, O. Isayev, and A. Tropsha. Deep reinforcement learning for de novo drug design. *Science advances*, 4(7):eaap7885, 2018.

D. Precup, R. S. Sutton, and S. Singh. Eligibility traces for off-policy policy evaluation. In *International Conference on Machine Learning*, 2000.

D. Precup, R. S. Sutton, and S. Dasgupta. Off-policy temporal-difference learning with function approximation. In *International Conference on Machine Learning*, 2001.

S. Racanière, T. Weber, D. Reichert, L. Buesing, A. Guez, D. Jimenez Rezende, A. Puigdomènech Badia, O. Vinyals, N. Heess, Y. Li, R. Pascanu, P. Battaglia, D. Hassabis, D. Silver, and D. Wierstra. Imagination-augmented agents for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, 2017.

C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21:1–67, 2020.

A. Raghu, O. Gottesman, Y. Liu, M. Komorowski, A. Faisal, F. Doshi-Velez, and E. Brunskill. Behaviour policy estimation in off-policy policy evaluation: Calibration matters. *arXiv preprint arXiv:1807.01066*, 2018.

A. Rajeswaran, S. Ghotra, S. Levine, and B. Ravindran. EPOpt: Learning robust neural network policies using model ensembles. In *International Conference on Learning Representations*, 2017.

A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. In *Robotics: Science and Systems*, 2018.

P. Rajpurkar, J. Irvin, K. Zhu, B. Yang, H. Mehta, T. Duan, D. Ding, A. Bagul, C. Langlotz, K. Shpanskaya, et al. Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning. *arXiv preprint arXiv:1711.05225*, 2017.

J. Redmon and A. Farhadi. Yolo9000: better, faster, stronger. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

N. Rhinehart, R. McAllister, and S. Levine. Deep imitative models for flexible inference, planning, and control. In *International Conference on Learning Representations*, 2019.

J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

J. Rissanen and T. Roos. Conditional nml universal models. In *2007 Information Theory and Applications Workshop*, pages 337–341. IEEE, 2007.

J. J. Rissanen. Fisher information and stochastic complexity. *IEEE transactions on information theory*, 42(1):40–47, 1996.

T. Roos, T. Silander, P. Kontkanen, and P. Myllymaki. Bayesian network structure learning using factorized nml universal models. In *2008 Information Theory and Applications Workshop*, pages 272–276. IEEE, 2008.

R. Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and computing in applied probability*, 1(2):127–190, 1999.

T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *arXiv preprint arXiv:1701.05517*, 2017.

K. S. Sarkisyan, D. A. Bolotin, M. V. Meer, D. R. Usmanova, A. S. Mishin, G. V. Sharonov, D. N. Ivankov, N. G. Bozhanova, M. S. Baranov, O. Soylemez, N. S. Bogatyreva, P. K. Vlasov, E. S. Egorov, M. D. Logacheva, A. S. Kondrashov, D. M. Chudakov, E. V. Putintseva, I. Z. Mamedov, D. S. Tawfik, K. A. Lukyanov, and F. A. Kondrashov. Local fitness landscape of the green fluorescent protein. *Nature*, 533(7603):397–401, May 2016. ISSN 1476-4687. doi: 10.1038/nature17995.

S. Schaal. Is imitation learning the route to humanoid robots?, 1999.

T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *CoRR*, abs/1511.05952, 2016.

B. Scherrer, M. Ghavamzadeh, V. Gabillon, B. Lesner, and M. Geist. Approximate modified policy iteration and its application to the game of tetris. *Journal of Machine Learning Research*, 16:1629–1676, 2015. URL `http://jmlr.org/papers/v16/scherrer15a.html`.

J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, 2015.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

G. Shafer and V. Vovk. A tutorial on conformal prediction. *Journal of Machine Learning Research*, 9(Mar):371–421, 2008.

B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2015.

S. Shalev-Shwartz and S. Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

V. Shankar, A. Dave, R. Roelofs, D. Ramanan, B. Recht, and L. Schmidt. Do image classifiers generalize across time? *arXiv preprint arXiv:1906.02168*, 2019.

N. Y. Siegel, J. T. Springenberg, F. Berkenkamp, A. Abdolmaleki, M. Neunert, T. Lampe, R. Hafner, and M. Riedmiller. Keep doing what worked: Behavioral modelling priors for offline reinforcement learning. In *International Conference on Learning Representations*, 2020.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

D. Silver, H. van Hasselt, M. Hessel, T. Schaul, A. Guez, T. Harley, G. Dulac-Arnold, D. Reichert, N. Rabinowitz, A. Barreto, and T. Degris. The predictron: End-to-end learning and planning. In *International Conference on Machine Learning*, 2017.

J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, 2012.

A. Strehl, J. Langford, L. Li, and S. M. Kakade. Learning from logged implicit exploration data. In *Advances in Neural Information Processing Systems*, 2010.

W. Sun, G. J. Gordon, B. Boots, and J. Bagnell. Dual policy iteration. In *Advances in Neural Information Processing Systems*, 2018.

I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, 2014.

R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *International Conference on Machine Learning*, 1990.

R. S. Sutton and A. G. Barto. *c.* MIT Press, second edition, 2018.

R. S. Sutton, H. R. Maei, D. Precup, S. Bhatnagar, D. Silver, C. Szepesvári, and E. Wiewiora. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *International Conference on Machine Learning*, 2009.

R. S. Sutton, A. R. Mahmood, and M. White. An emphatic approach to the problem of off-policy temporal-difference learning. *The Journal of Machine Learning Research*, 17(1): 2603–2631, 2016.

A. Swaminathan and T. Joachims. Counterfactual risk minimization: Learning from logged bandit feedback. In *International Conference on Machine Learning*, pages 814–823, 2015a.

A. Swaminathan and T. Joachims. The self-normalized estimator for counterfactual learning. In *Advances in Neural Information Processing Systems*, 2015b.

C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.

I. Szita and C. Szepesvari. Model-based reinforcement learning with nearly tight exploration complexity bounds. In *International Conference on Machine Learning*, 2010.

E. Talvitie. Model regularization for stable sample rollouts. In *Conference on Uncertainty in Artificial Intelligence*, 2014.

E. Talvitie. Self-correcting models for model-based reinforcement learning. In *AAAI Conference on Artificial Intelligence*, 2016.

A. Tamar, Y. WU, G. Thomas, S. Levine, and P. Abbeel. Value iteration networks. In *Advances in Neural Information Processing Systems*, 2016.

D. Thapa, I.-S. Jung, and G.-N. Wang. Agent based decision support system using reinforcement learning under emergency circumstances. In *International Conference on Natural Computation*, pages 888–892. Springer, 2005.

G. Theocharous, P. S. Thomas, and M. Ghavamzadeh. Personalized ad recommendation systems for life-time value optimization with guarantees. In *International Joint Conference on Artificial Intelligence*, 2015.

P. Thomas and E. Brunskill. Data-efficient off-policy policy evaluation for reinforcement learning. In *International Conference on Machine Learning*, 2016.

P. S. Thomas, G. Theocharous, and M. Ghavamzadeh. High-confidence off-policy evaluation. In *AAAI Conference on Artificial Intelligence*, 2015.

P. S. Thomas, G. Theocharous, M. Ghavamzadeh, I. Durugkar, and E. Brunskill. Predictive off-policy policy evaluation for nonstationary decision problems, with applications to digital marketing. In *AAAI Conference on Artificial Intelligence*, 2017.

E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

B. Trabucco, X. Geng, A. Kumar, and S. Levine. Design-bench: Benchmarks for data-driven offline model-based optimization. 2021.

M. Treiber, A. Hennecke, and D. Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical review E*, 62(2):1805, 2000.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

E. Vinitsky, A. Kreidieh, L. Le Flem, N. Kheterpal, K. Jang, C. Wu, F. Wu, R. Liaw, E. Liang, and A. M. Bayen. Benchmarks for reinforcement learning in mixed-autonomy traffic. In *Conference on Robot Learning*, 2018.

C. Voloshin, H. M. Le, N. Jiang, and Y. Yue. Empirical study of off-policy policy evaluation for reinforcement learning. *arXiv preprint arXiv:1911.06854*, 2019.

Y.-X. Wang, A. Agarwal, and M. Dudık. Optimal and adaptive off-policy evaluation in contextual bandits. In *International Conference on Machine Learning*, pages 3589–3597. PMLR, 2017.

Z. Wang, A. Novikov, K. Żołna, J. T. Springenberg, S. Reed, B. Shahriari, N. Siegel, J. Merel, C. Gulcehre, N. Heess, and N. de Freitas. Critic regularized regression. In *Advances in Neural Information Processing Systems*, 2020.

C. J. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

J. Wen, B. Dai, L. Li, and D. Schuurmans. Batch stationary distribution estimation. In *International Conference on Machine Learning*, 2020.

W. Whitney and R. Fergus. Understanding the asymptotic performance of model-based RL methods, 2019. URL `https://openreview.net/forum?id=B1g29oAqtm`.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

C. Wu, A. Kreidieh, K. Parvate, E. Vinitsky, and A. M. Bayen. Flow: Architecture and benchmarking for reinforcement learning in traffic control. *arXiv preprint arXiv:1710.05465*, page 10, 2017.

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

Y. Xie, Q. Liu, Y. Zhou, B. Liu, Z. Wang, and J. Peng. Off-policy evaluation and learning from logged bandit feedback: Error reduction via surrogate policy. In *International Conference on Learning Representations*, 2019.

M. Yang, O. Nachum, B. Dai, L. Li, and D. Schuurmans. Off-policy evaluation via the regularized lagrangian. *arXiv preprint arXiv:2007.03438*, 2020.

T. Yu, G. Thomas, L. Yu, S. Ermon, J. Zou, S. Levine, C. Finn, and T. Ma. Mopo: Model-based offline policy optimization. In *Advances in Neural Information Processing Systems*, 2020.

M. R. Zhang, T. Paine, O. Nachum, C. Paduraru, G. Tucker, ziyu wang, and M. Norouzi. Autoregressive dynamics models for offline policy evaluation and optimization. In *International Conference on Learning Representations*, 2021.

S. Zhang, L. Yao, A. Sun, and Y. Tay. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)*, 52(1):1–38, 2019.

K. Zhou and J. C. Doyle. *Essentials of robust control*, volume 104. Prentice hall Upper Saddle River, NJ, 1998.

B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.

Appendix A

Bootstrapping Error Accumulation Reduction

A.1 Distribution-Constrained Backup Operator

In this section, we analyze properties of the constrained Bellman backup operator, defined as:

$$\mathcal{T}^{\Pi}Q(s,a) \stackrel{\text{def}}{=} \mathbb{E}\big[R(s,a) + \gamma \max_{\pi \in \Pi} \mathbb{E}_{P(s'|s,a)}\left[V(s')\right]\big]$$

where

$$V(s) \stackrel{\text{def}}{=} \max_{\pi \in \Pi} \mathbb{E}_{\pi}[Q(s,a)].$$

Such an operator can be reduced to a standard Bellman backup in a modified MDP. We can construct an MDP M' from the original MDP M as follows:

- The state space, discount, and initial state distributions remain unchanged from M.
- We define a new action set $\mathcal{A}' = \Pi$ to be the choice of policy π to execute.
- We define the new transition distribution p' as taking one step under the chosen policy π to execute and one step under the original dynamics p: $p'(s'|s,\pi) = E_{\pi}[p(s'|s,a)]$.
- Q-values in this new MDP, $Q^{\Pi}(s,\pi)$ would, in words, correspond to executing policy π for one step and executing the policy which maximizes the future discounted value function in the original MDP M thereafter.

Under this redefinition, the Bellman operator $\mathcal{T}^{\Pi}$ is mathematically the same operation as the Bellman operator under M'. Thus, standard results from MDP theory carry over - i.e. the existence of a fixed point and convergence of repeated application of $\mathcal{T}^{\Pi}$ to said fixed point.

A.2 Error Propagation Proofs

In this section, we provide proofs for Theorem 4.2.1 and Theorem 4.2.2.

Theorem A.2.1. *Suppose we run approximate distribution-constrained value iteration with a set constrained backup $\mathcal{T}^{\Pi}$. Assume that $\delta(s,a) \geq \max_k |Q_k(s,a) - \mathcal{T}^{\Pi}Q_{k-1}(s,a)|$ bounds the Bellman error. Then,*

$$\lim_{k\to\infty} \mathbb{E}_{\rho_0}[|V_k(s) - V^*(s)|] \leq \frac{\gamma}{(1-\gamma)^2}\left[C(\Pi)\mathbb{E}_{\mu}[\max_{\pi\in\Pi}\mathbb{E}_{\pi}[\delta(s,a)]] + \frac{1-\gamma}{\gamma}\alpha(\Pi)\right]$$

Proof. We first begin by introducing V^{Π}, the fixed point of $\mathcal{T}^{\Pi}$. By the triangle inequality, we have:

$$\begin{aligned}\mathbb{E}_{\rho_0}[|V_k(s) - V^*(s)|] &= \mathbb{E}_{\rho_0}[|V_k(s,a) - V^{\Pi}(s) + V^{\Pi}(s) - V^*(s)|] \\ &\leq \underbrace{\mathbb{E}_{\rho_0}[|V_k(s) - V^{\Pi}(s)|]}_{L_1} + \underbrace{\mathbb{E}_{\rho_0}[|V^{\Pi}(s) - V^*(s)|]}_{L_2}\end{aligned}$$

First, we note that $\max_{\pi}\mathbb{E}_{\pi}[\delta(s,a)]$ provides an upper bound on the value error:

$$\begin{aligned}|V_k(s) - \mathcal{T}^{\Pi}V_{k-1}(s)| &= |\max_{\pi}\mathbb{E}_{\pi}[Q_k(s,a)] - \max_{\pi}\mathbb{E}_{\pi}[\mathcal{T}^{\pi}Q_{k-1}(s,a)]| \\ &\leq \max_{\pi}\mathbb{E}_{\pi}[|Q_k(s,a) - \mathcal{T}^{\pi}Q_{k-1}(s,a)|] \\ &\leq \max_{\pi}\mathbb{E}_{\pi}[\delta(s,a)]\end{aligned}$$

We can bound L_1 with

$$L_1 \leq \frac{2\gamma}{(1-\gamma)^2}[C(\Pi)]\mathbb{E}_{\mu}[\max_{\pi\in\Pi}\mathbb{E}_{\pi}[\delta(s,a)]]$$

by direct modification of the proof of Theorem 3 of Farahmand et al. [2010] or Theorem 1 of Munos [2005] with $k=1$ ($p=1$), but replacing V^* with V^{Π} and $\mathcal{T}$ with $\mathcal{T}^{\Pi}$, as $\mathcal{T}^{\Pi}$ is a contraction and V^{Π} is its fixed point. An alternative proof involves viewing $\mathcal{T}^{\Pi}$ as a backup under a modified MDP (see Appendix A.1), and directly apply Theorem 1 of Munos [2005] under this modified MDP. A similar bound also holds true for value iteration with the $\mathcal{T}^{\Pi}$ operator which can be analysed on similar lines as the above proof and Munos [2005].

To bound L_2, we provide a simple ℓ_{∞}-norm bound, although we could in principle apply techniques used to bound L_1 to get a tighter distribution-based bound.

$$\begin{aligned}\left\|V^{\Pi} - V^*\right\|_{\infty} &= \left\|\mathcal{T}^{\Pi}V^{\Pi} - \mathcal{T}V^*\right\|_{\infty} \\ &\leq \left\|\mathcal{T}^{\Pi}V^{\Pi} - \mathcal{T}^{\Pi}V^*\right\|_{\infty} + \left\|\mathcal{T}^{\Pi}V^{\Pi} - \mathcal{T}V^*\right\|_{\infty} \\ &\leq \gamma\left\|V^{\Pi} - V^*\right\|_{\infty} + \alpha(\Pi)\end{aligned}$$

Thus, we have $\left\|V^{\Pi} - V^*\right\|_{\infty} \leq \frac{\alpha}{1-\gamma}$. Because the maximum is greater than the expectation, $L_2 = \mathbb{E}_{\rho_0,\pi}[|V^{\Pi}(s) - V^*(s)|] \leq \left\|V^{\Pi} - V^*\right\|_{\infty}$.

Adding L_1 and L_2 completes the proof. □

Theorem A.2.2. *Assume the data distribution μ is generated by a behavior policy β, such that $\mu(s,a) = \mu_\beta(s,a)$. Let $\mu(s)$ be the marginal state distribution under the data distribution. Let us define $\Pi_\epsilon = \{\pi \mid \pi(a|s) = 0$ whenever $\beta(a|s) < \epsilon\}$. Then, there exists a concentrability coefficient $C(\Pi_\epsilon)$ is bounded as:*

$$C(\Pi_\epsilon) \leq C(\beta) \cdot \left(1 + \frac{\gamma}{(1-\gamma)f(\epsilon)}(1-\epsilon)\right)$$

where $f(\epsilon) \stackrel{\text{def}}{=} \min_{s \in \mathcal{S}, \mu_\Pi(s) > 0}[\mu(s)]$.

Proof. For notational clarity, we refer to Π_ϵ as Π in this proof. The term μ_Π is the highest discounted marginal state distribution starting from the initial state distribution ρ and following policies $\pi \in \Pi$. Formally, it is defined as:

$$\mu_\Pi \stackrel{\text{def}}{=} \max_{\{\pi_i\}_i:\ \forall\ i,\ \pi_i \in \Pi} (1-\gamma) \sum_{m=1}^{\infty} m\gamma^{m-1} \rho_0 P^{\pi_1} \cdots P^{\pi_m}$$

Now, we begin the proof of the theorem. We first note, from the definition of Π, $\forall\ s \in \mathcal{S}\ \forall\ \pi \in \Pi, \pi(a|s) > 0 \implies \beta(a|s) > \epsilon$. This suggests a bound on the total variation distance between β and any $\pi \in \Pi$ for all $s \in \mathcal{S}$, $D_{TV}(\beta(\cdot|s)||\pi(\cdot|s)) \leq 1 - \epsilon$. This means that the marginal state distributions of β and Π, are bounded in total variation distance by: $D_{TV}(\mu_\beta||\mu_\Pi) \leq \frac{\gamma}{1-\gamma}(1-\epsilon)$, where μ_Π is the marginal state distribution as defined above. This can be derived from Schulman et al. [2015], Appendix B, which bounds the difference in returns of two policies by showing the state marginals between two policies are bounded if their total variation distance is bounded.

Further, the definition of the set of policies Π implies that $\forall\ s \in \mathcal{S}, \mu_\Pi(s) > 0 \implies \mu_\beta(s) \geq f(\epsilon)$, where $f(\epsilon) > 0$ is a constant that depends on ϵ and captures the minimum visitation probability of a state $s \in \mathcal{S}$ when rollouts are executed from the initial state distribution ρ while executing the behaviour policy $\beta(a|s)$, under the constraint that only actions with $\beta(a|s) \geq \epsilon$ are selected for execution in the environment. Combining it with the total variation divergence bound, $\max_s ||\mu_\beta(s) - \mu_\Pi(s)|| \leq \frac{\gamma}{1-\gamma}(1-\epsilon)$, we get that

$$\sup_{s \in \mathcal{S}} \frac{\mu_\Pi(s)}{\mu_\beta(s)} \leq 1 + \frac{\gamma}{(1-\gamma)f(\epsilon)}(1-\epsilon)$$

We know that, $C(\Pi) \stackrel{\text{def}}{=} (1-\gamma)^2 \sum_{k=1}^{\infty} k\gamma^{k-1} c(k)$ is the ratio of the marginal state visitation distribution under the policy iterates when performing backups using the distribution-constrained operator and the data distribution $\mu = \mu_\beta$. Therefore,

$$\frac{C(\Pi_\epsilon)}{C(\beta)} \stackrel{\text{def}}{=} \sup_{s \in \mathcal{S}} \frac{\mu_\Pi(s)}{\mu_\beta(s)} \leq 1 + \frac{\gamma}{(1-\gamma)f(\epsilon)}(1-\epsilon)$$

□

A.3 Additional Details Regarding BEAR-QL

In this appendix, we address several remaining points regarding the support matching formulation of BEAR-QL, and further discuss its connections to prior work.

Why can we choose actions from Π_ϵ, the support of the training distribution, and need not restrict action selection to the policy distribution?

In Section 4.2, we designed a new distribution-constrained backup and analyzed its properties from an error propagation perspective. Theorems 4.2.1 and 4.2.2 tell us that, if the maximum projection error on all actions within the support of the train distribution is bounded, then the worst-case error incurred is also bounded. That is, we have a bound on $\max_{\pi \in \Pi_\epsilon} \mathbb{E}_\pi[\delta_k(s,a)]$. In this section, we provide an intuitive explanation for why action distributions that are very different from the training policy distributions, but still lie in the support of the train distribution, can be chosen without incurring large error. In practice, we use powerful function approximators for Q-learning, such as deep neural networks. That is, $\delta_k(s,a)$ is the Bellman error for one iteration of Q-iteration/Q-learning, which can essentially be viewed as a supervised regression problem with a very expressive function class. In this scenario, we expect a bounded error on the entire support of the training distribution, and we therefore expect approximation error to depend less on the specific density of a datapoint under the data distribution, and more on whether or not that datapoint is within the support of the data distribution. I.e., any point that is within the support would have a comparatively low error, due to the expressivity of the function approximator.

Another justification is that, a different version of the Bellman error objective renormalizes the action-distributions to the uniform distribution by applying an inverse behavior policy density weighting. For example, Antos et al. [2008], Antos et al. [2007] use this variant of Bellman error:

$$Q_{k+1} = \text{argmin}_Q \sum_{i=1, a_i \sim \beta(\cdot|s_i)}^{N} \frac{1}{\beta\left(a_i|s_i\right)} \left(Q\left(s_i, a_i\right) - \left[R(s,a) + \gamma \max_{a' \in \mathcal{A}} Q_k\left(s_{i+1}, a'\right) \right] \right)^2$$

This implies that this form of Bellman error mainly depends upon the support of the behaviour policy β (i.e. the set of action samples sampled from β with a high-enough probability which we formally refer to as $\beta(a|s) \geq \epsilon$ in the main text). In a scenario when this form of Bellman error is being minimized, $\delta_k(s,a)$ is defined as

$$\delta_k(s,a) = \frac{1}{\beta(a|s)} \left| Q_k(s,a) - \mathcal{T} Q_{k-1}(s,a) \right|$$

The overall error, hence, incurred due to error propagation is expected to be insensitive to distribution change, provided the support of the distribution doesn't change. Therefore, all

policies $\pi \in \Pi_\epsilon$ incur the same amount of propagated error ($|V_k - V_\Pi|$) whereas different amount of suoptimality biases – suggesting the existence of a different policy in Π_ϵ which propagates the same amount of error while having a lower suboptimality bias. However, in practice, it has been observed that using the inverse density weighting under the behaviour policy doesn't lead to substantially better performance for vanilla RL (not in the setting with purely off-policy, static datasets), so we use the unmodified Bellman error objective.

Both of these justifications indicate that bounded $\delta_k(s, a)$ is reasonable to expect under in-support action distributions.

Details on connection between BEAR-QL and distribution-constrained backups

Distribution-constrained backups perform maximization over a set of policies Π_ϵ which is defined as the set of policies that share the support with the behaviour policy. In the BEAR-QL algorithm, π_ϕ is maximized towards maximizing the expected Q-value for each state under the action distribution defined by it, while staying in-support (through the MMD constraint). The maximization step biases π_ϕ towards the in-support actions which maximize the current Q-value. By sampling multiple Dirac-delta action distributions - δ_{a_i} - and then performing an explicit maximum over them for computing the target is a stochastic approximation to the distribution-constrained operator. What is the importance of training the actor to maximize the expected Q-value? We found empirically that this step is important as without this maximization step and high-dimensional action spaces, it is likely to require many more samples (exponentially more, due to curse of dimensionality) to get the correct action that maximizes the target value while being in-support. This is hard and unlikely, and in some experiments we tried with this variant, we found it to lead to suboptimal solutions. At evaluation time, we use the Q-function as the actor. The same process is followed. Dirac-delta action distribution candidates δ_{a_i} are sampled, and then the action a_i that is gives the empirical maximum over the Q-function values is the action that is executed in the environment.

How effective is the MMD constraint in constraining supports of distributions?

In Section 4.3, we argued in favour of the usage of the sampled MMD distance between distributions to search for a policy that is supported on the same support as the train distribution. Revisiting the argument, in this section, we argue, via numerical simulations, the effectiveness of the MMD distance between two probability distributions in constraining the support of the distribution being learned, without constraining the distribution density function too much. While, MMD distance computed exactly between two distribution functions will match distributions exactly and that explains its applicability in 2-sample tests, however, with a limited number of samples, we empirically find that the values of the MMD distance

computed using samples from two d-dimensional Gaussian distributions with diagonal covariance matrices: $P \stackrel{\text{def}}{=} \mathcal{N}(\mu_P, \Sigma_P)$ and $Q \stackrel{\text{def}}{=} \mathcal{N}(\mu_Q, \Sigma_Q)$ is roughly equal to the MMD distance computed using samples from $\mathcal{U}_\alpha(P) \stackrel{\text{def}}{=} [\text{ Uniform}(\mu_P^1 \pm \alpha\Sigma_P^{1,1})] \times \cdots \times [\text{ Uniform}(\mu_P^d \pm \alpha\Sigma_P^{d,d})]$ and Q. This means that when minimizing the MMD distance to train distribution Q, the gradient signal would push Q towards a uniform distribution supported on P's support as this solution exhibits a lower MMD value – which is the objective we are optimizing.

Figure A.1 shows an empirical comparison of $\text{MMD}(P, Q)$ when $Q = P$, computed by sampling n-samples from P, and $\text{MMD}(\mathcal{U}_\alpha(P), Q)$ (also when $Q = P$) computed by sampling n-samples from $\mathcal{U}_\alpha(P)$. We observe that MMD distance computed using limited samples can, in fact, be higher between a distribution and itself as compared to a uniform distribution over a distribution's support and itself. In Figure A.1, note that for smaller values of n and appropriately chosen α (mentioned against each figure, the support of the uniform distribution), the estimator for $\text{MMD}(\mathcal{U}_\alpha(P), P)$ can provide lower estimates than the value of the estimator for $\text{MMD}(P, P)$. This observation suggests that when the number of samples is not enough to sample infer the distribution shape, density-agnostic distances like MMD can be used as optimization objectives to push distributions to match supports. Subfigures (c) and (d) shows the increase in MMD distance as the support of the uniform distribution is expanded.

In order to provide a theoretical example, we refer to Example 1 in Gretton et al. [2012], and extend it. First, note that the example argues that a fixed sample size of samples drawn from a distribution P, there exists another discrete distribution Q supported on m^2 samples from the support set of P, such that there atleast is a probability $\begin{pmatrix} m^2 \\ m \end{pmatrix} \frac{m!}{m^{2m}} > 1 - e^{-1} > 0.63$ that a sample from Q is indeed a sample from P as well. So, with a smaller value of m, *no* 2-sample test will be able to distinguish between P and Q. We would also note that this example is exactly the argument that our algorithm build upon. We further extend this example by noting that if Q were rather not completely supported on the support of P, then there exists atleast a probability of ϵ that a sample from Q lies outside the support of P. This gives us a lower bound on the value of the MMD estimator, indicating that the MMD 2-sample test will be able to detect this distribution due to an irreducible difference of $\epsilon\sqrt{\min_{y \in \text{Extremal(P)}} \mathbb{E}_{x \sim P}[k(x, y)]}$ (where y is an "extremal point" in P's support) in the MMD estimate.

Algorithm 5 Importance-Sampled BEAR-QL

input : Dataset $\mathcal{D}$, target network update rate τ, mini-batch size N, sampled actions for MMD n, minimum λ, policy gradient clipping constants $\beta_1, \beta_2; \beta_1 \leq \beta_2$, MMD threshold constant ε

Initialize Q-ensemble $\{Q_{\theta_i}\}_{i=1}^{M}$, actor π_ϕ, set-determining policy π_{set}, Lagrange multiplier α, target networks $\{Q_{\theta'_i}\}_{i=1}^{M}$, and a target actor $\pi_{\phi'}$, with $\phi' \leftarrow \phi, \theta'_i \leftarrow \theta_i$

for t in {1, . . . , N} **do**

Sample mini-batch of transitions $(s, a, r, s') \sim \mathcal{D}$

Q-update:

Sample m action samples, $\{a_i \sim \pi_{\phi'}(\cdot|s')\}_{i=1}^{n}$

Define $y = \frac{1}{m}\sum_{a_i}[\lambda \min_{j=1,..,M} Q_{\theta'_j}(s', a_i) + (1-\lambda)\max_{j=1,..,M} Q_{\theta'_j}(s', a_i)]$

$\forall i, \theta_i \leftarrow \arg\min_{\theta_i}(Q_{\theta_i}(s, a) - (r + \gamma y))^2$

Set-update and Actor-update:

Sample actions $A_1(s) \equiv \{\hat{a}_i \sim \pi_{set}(\cdot|s)\}_{i=1}^{m}$ and $A_2(s) \equiv \{a_j \sim \mathcal{D}(s)\}_{j=1}^{n}$, $n << m$

Update π_{set}, α:

$$\pi_{set}, \alpha \leftarrow \arg\min_{\pi_{set}} \max_{\alpha \geq 0} \sqrt{\frac{(1-\delta)\,\mathrm{var_k}\, E_{a\sim\pi_{set}(\cdot|s)}[\hat{Q}_k(s,a)]}{\delta}} + \alpha \mathbb{E}_{s\sim\mathcal{D}}([\mathrm{MMD}(A_1, A_2)] - \varepsilon)$$

Update ϕ using Importance Sampled Policy Gradient:

$$\pi_\phi \leftarrow \max_{\pi_\phi} \mathbb{E}_{s\sim\mathcal{D}} \mathbb{E}_{a\sim\pi_{set}(\cdot|s)} \Big(\Big[\frac{\pi_\phi(a|s)}{\pi_{set}(a|s)} \Big]_{\beta_1}^{\beta_2} Q(s,a) \Big)$$

Update Target Networks: $\theta'_i \leftarrow \tau\theta_i + (1-\tau)\theta'_i$; $\phi' \leftarrow \tau\phi + (1-\tau)\phi'$

end for

A.4 Additional Experimental Details

Data collection We trained behaviour policies using the Soft Actor-Critic algorithm Haarnoja et al. [2018]. In all cases, random data was generated by running a uniform at random policy in the environment. Optimal data was generated by training SAC agents in all 4 domains until convergence to the returns mentioned in Figure 4.5. Mediocre data was generated by training a policy until the return value marked in each of the plots in Figure 4.3. Each of our datasets contained 1e6 samples. We used the same datasets for evaluating different algorithms to maintain uniformity across results.

Choice of kernels In our experiments, we found that the choice of the kernel is an important design decision that needs to be made. In general, we found that a Laplacian kernel $k(x, y) = \exp(\frac{-||x-y||}{\sigma})$ worked well in all cases. Gaussian kernel $k(x, y) = \exp(\frac{-||x-y||^2}{2\sigma^2})$ worked quite well in the case of optimal dataset. For the Laplacian kernel, we chose $\sigma = 10.0$ for Cheetah, Ant and Hopper, and $\sigma = 20.0$ for Walker. However, we found that $\sigma = 20.0$ worked well for all environments in all settings. For the Gaussian kernel, we chose $\sigma = 20.0$

for all settings. Kernels often tend to not provide relevant measurements of distance especially in high-dimensional spaces, so one direction for future work is to design right kernels. We further experimented with a mixture of Laplacian kernel with different bandwidth parameters σ $(1.0, 10.0, 50.0)$ on Hopper-v2 and Walker2d-v2 where we found that it performs comparably and sometimes is better than a simple Laplacian kernel, probably because it is able to track supports upto different levels of thresholds due to multiple kernels.

More details about the algorithm At evaluation time, we find that using the greedy maximum of the Q-function over the support set of the behaviour policy (which can be approximated by sampling multiple Dirac-delta policies δ_{a_i} from the policy π_ϕ and performing a greedy maximization of the Q-values over these Dirac-delta policies) works best, better than unrolling the learned actor π_ϕ in the environment. This was also found useful in Fujimoto et al. [2019b]. Another detail about the algorithm is deciding which samples to use for computing the MMD objective. We train a parameteric model π_{data} which fits a tanh-Gaussian distribution to a given the states s, $\pi_{data}(\cdot|s) = \tanh \mathcal{N}(\mu(\cdot|s), \sigma(\cdot|s))$ and then use this to sample a candidate n actions for computing the MMD-distance, meaning that MMD is computed between $a_1, \cdots, a_N \sim \pi_{data}$ and π_ϕ. We find the latter to work better in practice. Also, computing the MMD distance between actions before applying the tanh transformation work better, and leads to a constraint, that perhaps provides stronger gradient signal – because tanh saturates very quickly, after which gradients almost vanish.

Other hyperparameters Other hyperparameters include the following – (1) The variance of the Gaussian σ^2 /(standard deviation of) Laplacian kernel σ: We tried a variance of 10, 20, and 40. We found that 10 and 20 worked well across Cheetah, Hopper and Ant, and 20 worked well for Walker2d; (2) The learning rate for the Lagrange multiplier was chosen to be 1e-3, and the log of the Lagrange multiplier was clipped between $[-5, 10]$ to prevent instabilities; (3) For the policy improvement step, we found using average Q works better than min Q for Walker2d. For the baselines, we used BCQ code from the official implementation accompanying Fujimoto et al. [2019b], TD3 code from the official implementation accompanying Fujimoto et al. [2018] and the BC baseline was the VAE-based behaviour cloning baseline also used in Fujimoto et al. [2019b]. We evaluated on 10 evaluation episodes (which were separate from the train distribution) after every 1000 iterations and used the average score and the variance for the plots.

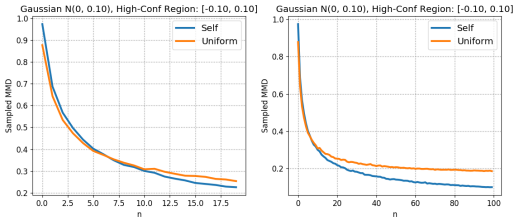
A.5 Additional Experimental Results

In this section, we provide some extra plots for some extra experiments. In Figure A.2 we provide the difference between learned Q-values and Monte carlo returns of the policy in the environment. In Figure A.3 we provide the trends of comparisons of Q-values learned by BEAR-QL and BCQ in three environments. In Figure A.4 we compare the performance when using the MMD constraint vs using the KL constraint in the case of three environments.

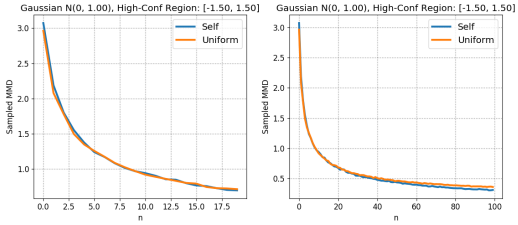
In order to be fair at comparing to MMD, we train a model for the behaviour policy and constrain the KL-divergence to this behaviour policy. (For MMD, we compute MMD using samples from the model of the behaviour policy.) Note that in the case of Half Cheetah with medium-quality data, KL divergence constraint works pretty well, but it fails drastically in the case of Hopper and Walker2d and the Q-values tend to diverge. Figure A.4 summarizes the trends for 3 environments.

We further study the performance of the KL-divergence in the setting when the KL-divergence is stable. In this setting we needed to perform extensive hyperparameter tuning to find the optimal Lagrange multiplier for the KL-constraint and plain and simple dual descent always gave us an unstable solution with the KL-constraint. Even in this case tuned hyperparameter case, we find that using a KL-constraint is worse than using a MMD-constraint. Trends are summarized in Figure A.5.

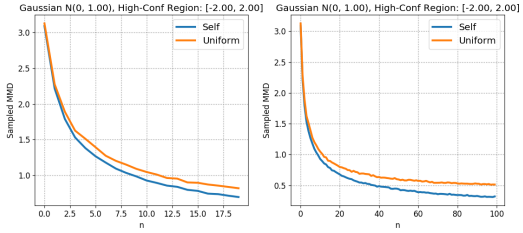
As described in Section A.3, we can achieve a reduced overall error $||V_k(s) - V^*(s)||$, if we use the MMD support-matching constraint alongside importance sampling, i.e. when we multiply the Bellman error with the inverse of the behaviour policy density. Empirically, we tried reweighting the Bellman error by inverse of the fitted behavior policy density, alongside the BEAR-QL algorithm. The trends for two environments and medium-quality data are summarized in Figure A.6. We found that reweighting the Bellman error wasn't that useful, although in theory, it provides an absolute error reduction as described by Theorem 4.1. We hypothesize that this could be due to the possible reason that when optimizing neural nets using stochastic gradient procedures, importance sampling isn't that beneficial [Byrd and Lipton, 2019].



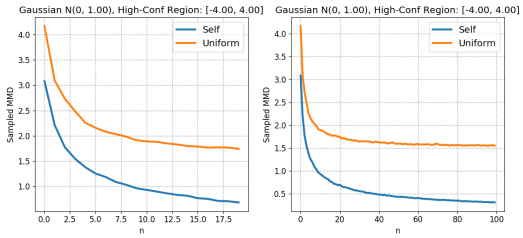
(a) $\mathcal{N}(0, 0.1), \mathcal{U}(-0.1, 0.1)$



(b) $\mathcal{N}(0, 1.0), \mathcal{U}(-1.5, 1.5)$



(c) $\mathcal{N}(0, 1.0), \mathcal{U}(-2.0, 2.0)$



(d) $\mathcal{N}(0, 1.0), \mathcal{U}(-4.0, 4.0)$

Figure A.1: Comparing MMD distance between a 1-d Gaussian distribution (P) and itself (P), and a uniform distribution over support set of the P and the distribution P. The parameters of the Gaussian distribution (P) and the uniform distribution being considered are mentioned against each plot. ('Self' refers to MMD(P, P) and 'Uniform' refers to MMD$(P, \mathcal{U}(P))$.) Note that for small values of $n \approx 1 - 10$, the MMD with the Uniform distribution is slightly lower in magnitude than the MMD between the distribution P and itself (sub-figures (a), (b) and (c)). For (d), as the support of this uniform distribution is enlarged, this leads to an increase in the value of MMD in the uniform approximation case – which suggests that a near-local minimizer for the MMD distance can be obtained by making sure that the distribution which is being trained in this process shares the same support as the other given distribution.

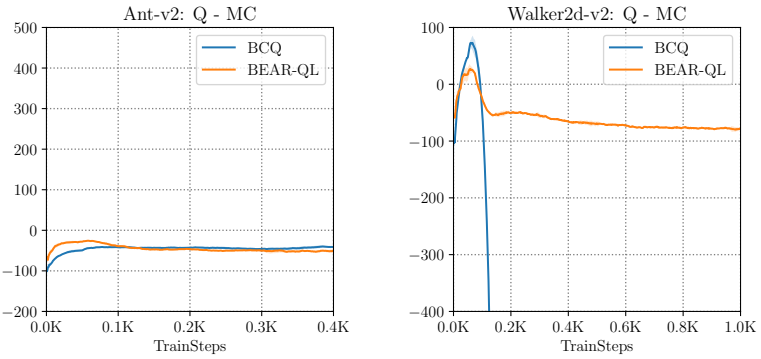


Figure A.2: The trend of the difference between the Q-values and Monte-Carlo returns: $Q - MC$ returns for 2 environments. Note that a high value of $Q - MC$ corresponds to more overestimation. In these plots, BEAR-QL is more well behaved than BCQ. In Walker2d-v2, BCQ tends to diverge in the negative direction. In the case of Ant-v2, although roughly the same, the difference between Q values and Monte-carlo returns is slightly lower in the case of BEAR-QL suggestion no risk of overestimation. (This corresponds to medium-quality data.)

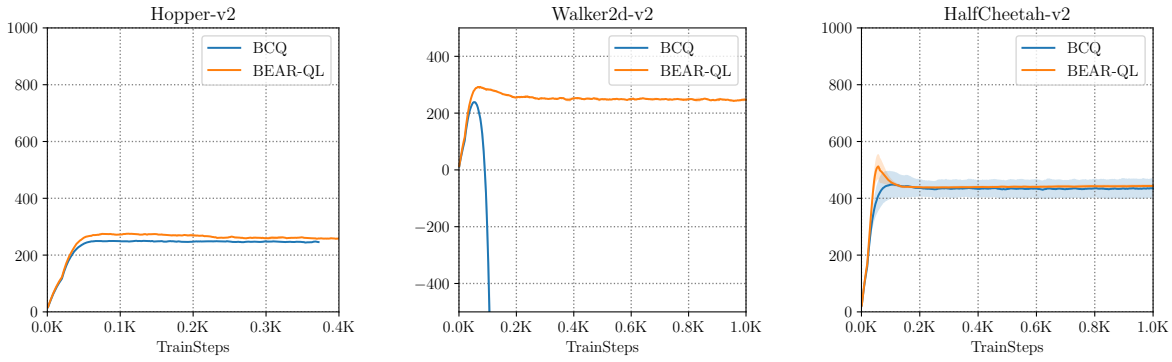


Figure A.3: The trends of Q-values as a function of number of gradient steps taken in case of 3 environments. BCQs Q-values tend to be more unstable (especially in the case of Walker2d, where they diverge in the negative direction) as compared to BEAR-QL. This corresponds to medium-quality data.

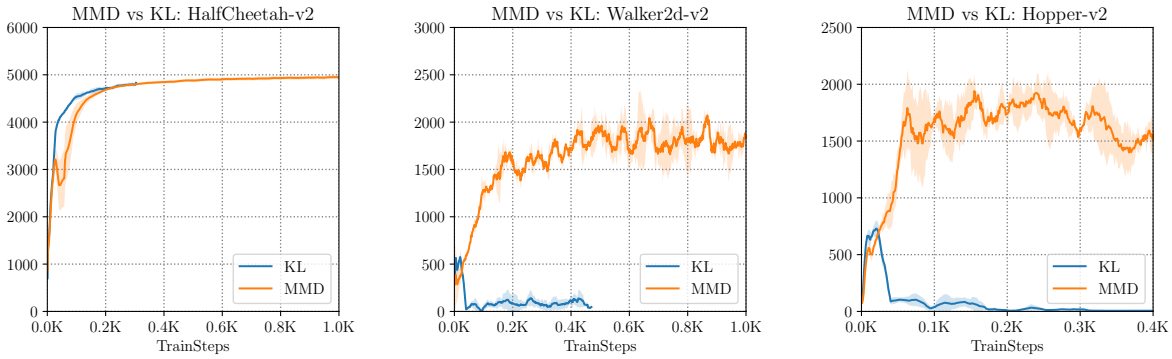


Figure A.4: Performance Trends (measured in AverageReturn) for Hopper-v2, HalfCheetah-v2 and Walker2d-v2 environments with BEAR-QL algorthm but varying kind of constraint. In general we find that using the KL constraint leads to worse performance. However, in some rare cases (for example, HalfCheetah-v2), the KL constraint learns faster. In general, we find that the KL-constraint often leads to diverging Q-values. This experiment corresponds to medium-quality data.

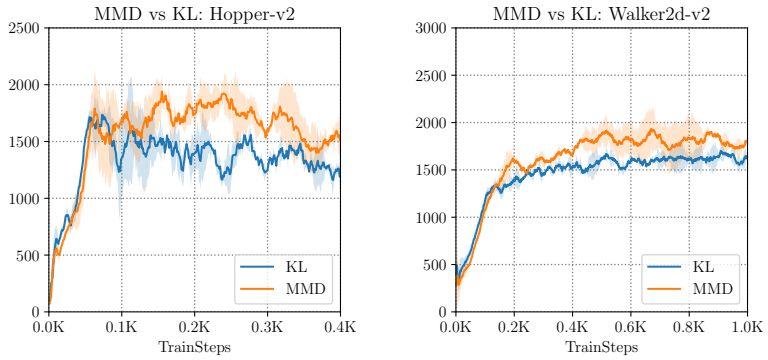


Figure A.5: Performance Trends (measured in Average Returns) for Hopper-v2 and Walker2d-v2 environments with BEAR-QL algorithm with an extensively tuned KL-constraint and the MMD-constraint from. Note that the MMD-constraint still outperforms the KL-constraint.

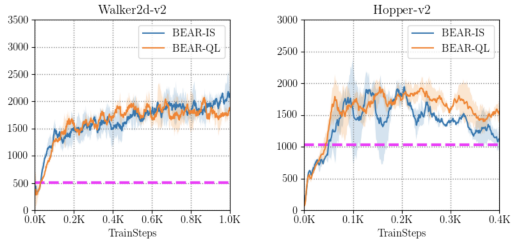


Figure A.6: BEAR with importance sampled Bellman error minimization. We find that importance sampling isn't that beneficial in practice.

Appendix B

Model-Based Policy Optimization

B.1 Model-Based Policy Optimization with Performance Guarantees

In this appendix, we provide proofs for bounds presented in Chapter 5.

We begin with a standard bound on model-based policy optimization, with bounded policy change ϵ_π and model error ϵ_m.

Theorem B.1.1 (MBPO performance bound)**.** *Let the expected total variation between two transition distributions be bounded at each timestep by* $\max_t \mathbb{E}_{s\sim\pi_{D,t}}[D_{TV}(p(s'|s,a)||\hat{p}(s'|s,a))] \leq \epsilon_m$*, and the policy divergences are bounded as* $\max_s D_{TV}(\pi_D(a|s)||\pi(a|s)) \leq \epsilon_\pi$*. Then the returns are bounded as:*

$$J(\pi) \geq \hat{J}(\pi) - \frac{2\gamma R_{\max}(\epsilon_m + 2\epsilon_\pi)}{(1-\gamma)^2} - \frac{4R_{\max}\epsilon_\pi}{(1-\gamma)}$$

Proof. Let π_D denote the data collecting policy. As-is we can use Lemma B.2.3 to bound the returns, but it will require bounded model error under the new policy π. Thus, we need to introduce π_D by adding and subtracting $J(\pi_D)$, to get:

$$J(\pi) - \hat{J}(\pi) = \underbrace{J(\pi) - J(\pi_D)}_{L_1} + \underbrace{J(\pi_D) - \hat{J}(\pi)}_{L_2}$$

We can bound L_1 and L_2 both using Lemma B.2.3.

For L_1, we apply Lemma B.2.3 using $\delta = \epsilon_\pi$ (no model error because both terms are under the true model), and obtain:

$$L_1 \geq -\frac{2R_{\max}\gamma\epsilon_\pi}{(1-\gamma)^2} - \frac{2R_{\max}\epsilon_\pi}{1-\gamma}$$

For L_1, we apply Lemma B.2.3 using $\delta = \epsilon_\pi + \epsilon_m$ and obtain:

$$L_2 \geq -\frac{2R_{\max}\gamma(\epsilon_m + \epsilon_\pi)}{(1-\gamma)^2} - \frac{2R_{\max}\epsilon_\pi}{1-\gamma}$$

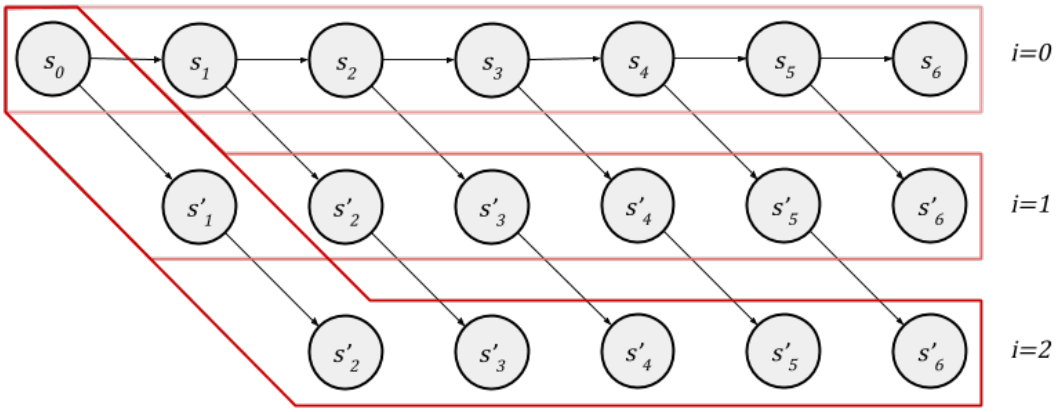


Figure B.1: The conceptual illustration of the branched rollout generating process, with $k = 2$. The top row ($i = 0$) represents the original trajectory, and the horizontal arrows represent transitions under the *pre-branch* policy and dynamics. Each state is then advanced forward by one step (represented by diagonal arrows) under the *post-branch* policy and dynamics to generate a new trajectory for $i = 1$, with the initial state appended to the beginning of the trajectory. We repeat this process K times, and average the occupancy measures of each individual rollout (associated with i) to obtain the occupancy measure of the final averaged K-branched rollout. The standard K-branched rollout only uses the trajectory corresponding to $i = K$.

Adding these two bounds together yields the desired result. □

Next, we describe bounds for branched rollouts. We define a branched rollout as a rollout which begins under some policy and dynamics (either true or learned), and at some point in time switches to rolling out under a new policy and dynamics for k steps. Our definition is motivated by our practical implementation based on experience replay in a Dyna-like algorithm, and provide a conceptual illustration in Fig. B.1. The unbranched trajectory ($i = 0$) typically represents data collected in the environment under our old policy. We then simulate each state forward up to K steps under the model and new policy, and add these experiences to the replay memory. We precisely define the mechanics of how the occupancy measure of the rollout in two schemes: the K-branched rollout, and the averaged K-branched rollout. The standard K-branched rollout only uses $i = k$, and discards all states for $i < k$. The averaged K-branched rollout instead computes the mean over all $i = 0$ to $i = k$. THe standard branched rollout has simpler bounds (which display the same trends as the averaged rollout), but the averaged branch rollout is closer to our practical algorithm, which adds all model-generated states to the replay memory with equal weight.

Definition B.1.1 (Standard K-Branched Rollout)**.** *The occupancy measure of the standard*

K-branched rollout is defined as:

$$\rho^{branch,k}(s,a) = \sum_{t=0}^{\infty} \gamma^t p_{k,t}(s_t = s, a_t = a)$$

Where $p_{i,t}(s,a)$ denotes the state-action marginal of the pre-branch dynamics advanced under the post-branch dynamics $p^{post}(s'|s,a)$ and policy $\pi^{post}(a|s)$. It is defined as follows:

$$p_{i,t}(s_t, a_t) = \sum_{s_{t-1}, a_{t-1}} \pi^{post}(a_t|s_t) p^{post}(s_t|s_{t-1}, a_{t-1}) p_{i-1,t-1}(s_{t-1}, a_{t-1})$$

and $p_{i=0,t}(s,a)$ represents the state-action marginal under only the pre-branch dynamics, and $p_{i,t\leq 0}(s) = \rho_0(s)$ is fixed to the initial state distribution.

Definition B.1.2 (Averaged K-Branched Rollout)**.** *The occupancy measure of the averaged K-branched rollout is defined as the average of the standard branched rollout:*

$$\bar{\rho}^{branch,k}(s,a) = \frac{1}{k+1} \sum_{i=0}^{k} \rho^{branch,i}(s,a)$$

We first present the simpler bound where the model error is bounded under the new policy, which we label as $\epsilon_{m'}$. This bound is difficult to apply in practice as supervised learning will typically control model error under the dataset collected by the previous policy.

Theorem B.1.2. *Let the expected total variation between two the learned model is bounded at each timestep under the expectation of π by $\max_t \mathbb{E}_{s\sim\pi_t}[D_{TV}(p(s'|s,a)||\hat{p}(s'|s,a))] \leq \epsilon_{m'}$, and the policy divergences are bounded as $\max_s D_{TV}(\pi_D(a|s)||\pi(a|s)) \leq \epsilon_\pi$. Then under the standard branched rollouts scheme with a branch length of k, the returns are bounded as:*

$$J(\pi) \geq J^{\text{branch}}(\pi) - 2R_{\max}\left[\frac{\gamma^{k+1}\epsilon_\pi}{(1-\gamma)^2} + \frac{\gamma^k \epsilon_\pi}{(1-\gamma)} + \frac{k}{1-\gamma}(\epsilon_{m'})\right]$$

Proof. As in the proof for Theorem B.1.1, the proof for this theorem requires adding and subtracting the correct reference quantity and applying the corresponding returns bound (Lemma B.2.4).

The choice of reference quantity is a branched rollout which executes the old policy π_D under the true dynamics until the branch point, then executes the new policy π under the true dynamics for k steps. We denote the returns under this scheme as $J^{\pi_D,\pi}$. We can split the returns as follows:

$$J(\pi) - J^{\text{branch}} = \underbrace{J(\pi) - J^{\pi_D,\pi}}_{L_1} + \underbrace{J^{\pi_D,\pi} - J^{\text{branch}}}_{L_2}$$

We can bound both terms L_1 and L_2 using Lemma B.2.4.

L_1 accounts for the error from executing the old policy instead of the current policy. As there is no difference between $J(\pi)$ and $J^{\pi,\pi}$, we can compare $J^{\pi,\pi}$ with $J^{\pi_D,\pi}$. This term only suffers from error *before* the branch begins, and we can use Lemma B.2.4 $\epsilon_\pi^{\text{pre}} \leq \epsilon_\pi$ and all other errors set to 0. This implies:

$$|J(\pi) - J^{\pi_D,\pi}| \leq 2R_{\max}\left[\frac{\gamma^{k+1}}{(1-\gamma)^2}\epsilon_\pi + \frac{\gamma^k}{1-\gamma}\epsilon_\pi\right]$$

L_2 incorporates model error *under the new policy* incurred after the branch. Again we use Lemma B.2.4, setting $\epsilon_m^{\text{post}} \leq \epsilon_{m'}$ and all other errors set to 0. This implies:

$$|J(\pi) - J^{\pi_D,\pi}| \leq 2R_{\max}\left[\frac{k}{1-\gamma}\epsilon_{m'}\right]$$

Adding L_1 and L_2 together completes the proof.

□

Corollary B.1.2.1 (Theorem B.1.2 for averaged K-branched rollouts)**.** *Let the expected total variation between two the learned model is bounded at each timestep under the expectation of π by $\max_t \mathbb{E}_{s\sim\pi_t}[D_{TV}(p(s'|s,a)||\hat{p}(s'|s,a))] \leq \epsilon_{m'}$, and the policy divergences are bounded as $\max_s D_{TV}(\pi_D(a|s)||\pi(a|s)) \leq \epsilon_\pi$. Then under the averaged branched rollouts scheme with a branch length of k, the returns are bounded as:*

$$J(\pi) \geq J^{\text{branch}}(\pi) - \frac{2R_{\max}}{1-\gamma}\left[\frac{\gamma\epsilon_\pi}{(k+1)(1-\gamma)^2} + \frac{\epsilon_\pi}{(k+1)(1-\gamma)} + \frac{k+2}{2}(\epsilon_{m'})\right]$$

The next bound is an analogue of Theorem B.1.2 except using model errors under the previous policy π_D rather than the new policy π.

Theorem B.1.3. *Let the expected total variation between two the learned model is bounded at each timestep under the expectation of π by $\max_t \mathbb{E}_{s\sim\pi_{D,t}}[D_{TV}(p(s'|s,a)||\hat{p}(s'|s,a))] \leq \epsilon_m$, and the policy divergences are bounded as $\max_s D_{TV}(\pi_D(a|s)||\pi(a|s)) \leq \epsilon_\pi$. Then under standard branched rollouts scheme with a branch length of k, the returns are bounded as:*

$$J(\pi) \geq J^{\text{branch}}(\pi) - 2R_{\max}\left[\frac{\gamma^{k+1}\epsilon_\pi}{(1-\gamma)^2} + \frac{\gamma^k+2}{(1-\gamma)}\epsilon_\pi + \frac{k}{1-\gamma}(\epsilon_m + 2\epsilon_\pi)\right]$$

Proof. This proof is a short extension of the proof for Theorem B.1.2. The only modification is that we need to bound L_2 in terms of the model error under the π_D rather than π.

Once again, we design a new reference rollout. We use a rollout that executes the old policy π_D under the true dynamics until the branch point, then executes the *old* policy π_D under the model for k steps. We denote the returns under this scheme as $J^{\pi_D,\hat{\pi_D}}$. We can split L_2 as follows:

$$J^{\pi_D,\pi} - J^{\text{branch}} = \underbrace{J^{\pi_D,\pi} - J^{\pi_D,\hat{\pi_D}}}_{L_3} + \underbrace{J^{\pi_D,\hat{\pi_D}} - J^{\text{branch}}}_{L_4}$$

Once again, we bound both terms L_3 and L_4 using Lemma B.2.4.

The rollouts in L_3 differ in both model and policy after the branch. This can be bound using Lemma B.2.4 by setting $\epsilon_\pi^{\text{post}} = \epsilon_\pi$ and $\epsilon_m^{\text{post}} = \epsilon_m$. This results in:

$$|J^{\pi_D,\pi} - J^{\pi_D,\hat{\pi_D}}| \leq 2R_{\max}\left[\frac{k}{1-\gamma}(\epsilon_m + \epsilon_\pi) + \frac{1}{1-\gamma}\epsilon_\pi\right]$$

The rollouts in L_4 differ only in the policy after the branch (as they both rollout under the model). This can be bound using Lemma B.2.4 by setting $\epsilon_\pi^{\text{post}} = \epsilon_\pi$ and $\epsilon_m^{\text{post}} = 0$. This results in:

$$|J^{\pi_D,\hat{\pi_D}} - J^{\text{branch}}| \leq 2R_{\max}\left[\frac{k}{1-\gamma}(\epsilon_\pi) + \frac{1}{1-\gamma}\epsilon_\pi\right]$$

Adding L_1 from Theorem B.1.2 and L_3, L_4 above completes the proof.

□

Corollary B.1.3.1 (Theorem B.1.3 for averaged K-branched rollouts)**.** *Let the expected total variation between two the learned model is bounded at each timestep under the expectation of π by $\max_t \mathbb{E}_{s\sim\pi_{D,t}}[D_{TV}(p(s'|s,a)||\hat{p}(s'|s,a))] \leq \epsilon_m$, and the policy divergences are bounded as $\max_s D_{TV}(\pi_D(a|s)||\pi(a|s)) \leq \epsilon_\pi$. Then under the averaged branched rollouts scheme with a branch length of k, the returns are bounded as:*

$$J(\pi) \geq J^{\text{branch}}(\pi) - \frac{2R_{\max}}{1-\gamma}\left[\frac{\gamma\epsilon_\pi}{(k+1)(1-\gamma)^2} + \frac{1}{(k+1)(1-\gamma)}\epsilon_\pi + \frac{k}{2}(\epsilon_m + 2\epsilon_\pi) + 2\epsilon_\pi\right]$$

B.2 Useful Lemmas

In this section, we provide proofs for various lemmas used in our bounds.

Lemma B.2.1 (TVD of Joint Distributions)**.** *Suppose we have two distributions $p_1(x,y) = p_1(x)p_1(y|x)$ and $p_2(x,y) = p_2(x)p_2(y|x)$. We can bound the total variation distance of the joint as:*

$$D_{TV}(p_1(x,y)||p_2(x,y)) \leq D_{TV}(p_1(x)||p_2(x)) + \max_x D_{TV}(p_1(y|x)||p_2(y|x))$$

Alternatively, we have a tighter bound in terms of the expected TVD of the conditional:

$$D_{TV}(p_1(x,y)||p_2(x,y)) \leq D_{TV}(p_1(x)||p_2(x)) + \mathbb{E}_{x\sim p_1}[D_{TV}(p_1(y|x)||p_2(y|x))]$$

Proof.

$$\begin{aligned}
D_{TV}(p_1(x,y)||p_2(x,y)) &= \frac{1}{2}\sum_{x,y}|p_1(x,y) - p_2(x,y)| \\
&= \frac{1}{2}\sum_{x,y}|p_1(x)p_1(y|x) - p_2(x)p_2(y|x)| \\
&= \frac{1}{2}\sum_{x,y}|p_1(x)p_1(y|x) - p_1(x)p_2(y|x) + (p_1(x) - p_2(x))p_2(y|x)| \\
&\leq \frac{1}{2}\sum_{x,y}p_1(x)|p_1(y|x) - p_2(y|x)| + |p_1(x) - p_2(x)|p_2(y|x) \\
&= \frac{1}{2}\sum_{x,y}p_1(x)|p_1(y|x) - p_2(y|x)| + \frac{1}{2}\sum_{x}|p_1(x) - p_2(x)| \\
&= \mathbb{E}_{x\sim p_1}[D_{TV}(p_1(y|x)||p_2(y|x))] + D_{TV}(p_1(x)||p_2(x)) \\
&\leq \max_x D_{TV}(p_1(y|x)||p_2(y|x)) + D_{TV}(p_1(x)||p_2(x))
\end{aligned}$$

□

Lemma B.2.2 (Markov chain TVD bound, time-varying)**.** *Suppose the expected KL-divergence between two transition distributions is bounded as* $\max_t \mathbb{E}_{s\sim p_1^t(s)} D_{KL}(p_1(s'|s)||p_2(s'|s)) \leq \delta$*, and the initial state distributions are the same –* $p_1^{t=0}(s) = p_2^{t=0}(s)$*. Then the distance in the state marginal is bounded as:*

$$D_{TV}(p_1^t(s)||p_2^t(s)) \leq t\delta$$

Proof. We begin by bounding the TVD in state-visitation at time t, which is denoted as $\epsilon_t = D_{TV}(p_1^t(s)||p_2^t(s))$.

$$\begin{aligned}
|p_1^t(s) - p_2^t(s)| &= |\sum_{s'} p_1(s_t = s|s')p_1^{t-1}(s') - p_2(s_t = s|s')p_2^{t-1}(s')| \\
&\leq \sum_{s'} |p_1(s_t = s|s')p_1^{t-1}(s') - p_2(s_t = s|s')p_2^{t-1}(s')| \\
&= \sum_{s'} |p_1(s|s')p_1^{t-1}(s') - p_2(s|s')p_1^{t-1}(s') + p_2(s|s')p_1^{t-1}(s') - p_2(s|s')p_2^{t-1}(s')| \\
&\leq \sum_{s'} p_1^{t-1}(s')|p_1(s|s') - p_2(s|s')| + p_2(s|s')|p_1^{t-1}(s') - p_2^{t-1}(s')| \\
&= \mathbb{E}_{s'\sim p_1^{t-1}}[|p_1(s|s') - p_2(s|s')|] + \sum_{s'} p(s|s')|p_1^{t-1}(s') - p_2^{t-1}(s')|
\end{aligned}$$

$$\begin{aligned}
\epsilon_t = D_{TV}(p_1^t(s)||p_2^t(s)) &= \frac{1}{2}\sum_s |p_1^t(s) - p_2^t(s)| \\
&= \frac{1}{2}\sum_s \left(\mathbb{E}_{s' \sim p_1^{t-1}}[|p_1(s|s') - p_2(s|s')|] + \sum_{s'} p(s|s')|p_1^{t-1}(s') - p_2^{t-1}(s')| \right) \\
&= \frac{1}{2}\mathbb{E}_{s' \sim p_1^{t-1}}[\sum_s |p_1(s|s') - p_2(s|s')|] + D_{TV}(p_1^{t-1}(s')||p_2^{t-1}(s')) \\
&= \delta_t + \epsilon_{t-1} \\
&= \epsilon_0 + \sum_{i=0}^{t} \delta_t \\
&= \sum_{i=0}^{t} \delta_t = t\delta
\end{aligned}$$

Where we have defined $\delta_t = \frac{1}{2}\mathbb{E}_{s' \sim p_1^{t-1}}[\sum_s |p_1(s|s') - p_2(s|s')]$, which we assume is upper bounded by δ. Assuming we are not modeling the initial state distribution, we can set $\epsilon_0 = 0$. □

Lemma B.2.3 (Returns bound)**.** *Suppose the expected KL-divergence between two dynamics distributions is bounded as* $\max_t \mathbb{E}_{s \sim p_1^t(s)} D_{KL}(p_1(s', a|s)||p_2(s', a|s)) \leq \epsilon_m$*, and* $\max_s D_{TV}(\pi_1(a|s)||\pi_2(a|s)) \leq \epsilon_\pi$*. Then the returns are bounded as:*

$$|J_1 - J_2| \leq \frac{2R\gamma(\epsilon_\pi + \epsilon_m)}{(1-\gamma)^2} + \frac{2R\epsilon_\pi}{1-\gamma}$$

Proof. Here, J_1 denotes returns of π_1 under dynamics $p_1(s'|s, a)$, and J_2 denotes returns of π_2 under dynamics $p_2(s'|s, a)$.

$$\begin{aligned}
|J_1 - J_2| &= |\sum_{s,a}(p_1(s, a) - p_2(s, a))r(s, a)| \\
&= |\sum_{s,a}(\sum_t \gamma^t p_1^t(s, a) - p_2^t(s, a))r(s, a)| \\
&= |\sum_t \sum_{s,a} \gamma^t (p_1^t(s, a) - p_2^t(s, a))r(s, a)| \\
&\leq \sum_t \sum_{s,a} \gamma^t |p_1^t(s, a) - p_2^t(s, a)|r(s, a) \\
&\leq R_{\max} \sum_t \sum_{s,a} \gamma^t |p_1^t(s, a) - p_2^t(s, a)|
\end{aligned}$$

We now apply Lemma B.2.2, using $\delta = \epsilon_m + \epsilon_\pi$ (via Lemma B.2.1) to get:

$$D_{TV}(p_1^t(s)||p_2^t(s)) \leq t(\epsilon_m + \epsilon_\pi)$$

And since we assume $\max_s D_{TV}(\pi_1(a|s)||\pi_2(a|s)) \leq \epsilon_\pi$, we get

$$D_{TV}(p_1^t(s,a)||p_2^t(s,a)) \leq t(\epsilon_m + \epsilon_\pi) + \epsilon_\pi$$

Thus, plugging this back in we get:

$$\begin{aligned}
|J_1 - J_2| &\leq R_{\max} \sum_t \sum_{s,a} \gamma^t |p_1^t(s,a) - p_2^t(s,a)| \\
&\leq 2R_{\max} \sum_t \gamma^t t(\epsilon_m + \epsilon_\pi) + \epsilon_\pi \\
&\leq 2R_{\max}(\frac{\gamma(\epsilon_\pi + \epsilon_m)}{(1-\gamma)^2} + \frac{\epsilon_\pi}{1-\gamma})
\end{aligned}$$

□

Lemma B.2.4 (Returns bound, branched rollout)**.** *Assume we run a branched rollout of length k. Before the branch ("pre" branch), we assume that the dynamics distributions are bounded as $\max_t \mathbb{E}_{s\sim p_1^t(s)} D_{KL}(p_1^{\text{pre}}(s'|s,a)||p_2^{\text{pre}}(s'|s,a)) \leq \epsilon_m^{\text{pre}}$ and after the branch as $\max_t \mathbb{E}_{s\sim p_1^t(s)} D_{KL}(p_1^{\text{post}}(s'|s,a)||p_2^{\text{post}}(s'|s,a)) \leq \epsilon_m^{\text{post}}$. Likewise, the policy divergence is bounded pre- and post- branch by $\epsilon_\pi^{\text{pre}}$ and $\epsilon_\pi^{\text{post}}$, respectively. Then the standard K-step returns are bounded as:*

$$|J_1 - J_2| \leq 2R_{\max}\left[\frac{\gamma^{k+1}}{(1-\gamma)^2}(\epsilon_m^{\text{pre}} + \epsilon_\pi^{\text{pre}}) + \frac{k}{1-\gamma}(\epsilon_m^{\text{post}} + \epsilon_\pi^{\text{post}}) + \frac{\gamma^k}{1-\gamma}\epsilon_\pi^{\text{pre}} + \frac{1}{1-\gamma}\epsilon_\pi^{\text{post}}\right]$$

And the averaged K-step returns are bounded as:

$$|J_1 - J_2| \leq \frac{2R_{\max}}{1-\gamma}\left[\frac{k}{2}(\epsilon_m^{\text{post}} + \epsilon_\pi^{\text{post}}) + \frac{\gamma}{(k+1)(1-\gamma)^2}(\epsilon_m^{\text{pre}} + \epsilon_\pi^{\text{pre}}) + \frac{1}{(k+1)(1-\gamma)}\epsilon_\pi^{\text{pre}} + \epsilon_\pi^{\text{post}}\right]$$

Proof. We first analyze the divergence for a particular i, and then we will average this quantity from $i = 1$ to $i = k$ to obtain the final bound.

We begin by bounding state marginals at each timestep, similar to Lemma B.2.3. Recall that Lemma B.2.2 implies that state marginal error at each timestep can be bounded by the state marginal error at the previous timestep, plus the divergence at the current timestep. Thus, letting $d_1(s,a)$ and $d_2(s,a)$ denote the state-action marginals, we can write:

For $t \leq i$:

$$D_{TV}(d_1^t(s,a)||d_2^t(s,a)) \leq t(\epsilon_m^{\text{post}} + \epsilon_\pi^{\text{post}}) + \epsilon_\pi^{\text{post}} \leq i(\epsilon_m^{\text{post}} + \epsilon_\pi^{\text{post}}) + \epsilon_\pi^{\text{post}}$$

and for $t \geq i$:

$$D_{TV}(d_1^t(s,a)||d_2^t(s,a)) \leq (t-i)(\epsilon_m^{\text{pre}} + \epsilon_\pi^{\text{pre}}) + i(\epsilon_m^{\text{post}} + \epsilon_\pi^{\text{post}}) + \epsilon_\pi^{\text{pre}} + \epsilon_\pi^{\text{post}}$$

We can now bound the difference in occupancy measures by averaging the state marginal error over time, weighted by the discount:

$$\begin{aligned}
D_{TV}(d_1(s,a)||d_2(s,a)) &\leq (1-\gamma)\sum_{t=0}^{\infty}\gamma^t D_{TV}(d_1^t(s,a)||d_2^t(s,a)) \\
&\leq (1-\gamma)\sum_{t=0}^{i}\gamma^t(i(\epsilon_m^{\text{post}}+\epsilon_\pi^{\text{post}})+\epsilon_\pi^{\text{post}}) \\
&\quad + (1-\gamma)\sum_{t=i}^{\infty}\gamma^t[(t-i)(\epsilon_m^{\text{pre}}+\epsilon_\pi^{\text{pre}})+i(\epsilon_m^{\text{post}}+\epsilon_\pi^{\text{post}})+\epsilon_\pi^{\text{pre}}+\epsilon_\pi^{\text{post}}] \\
&\leq i(\epsilon_m^{\text{post}}+\epsilon_\pi^{\text{post}})+\epsilon_\pi^{\text{post}}+\frac{\gamma^{i+1}}{1-\gamma}(\epsilon_m^{\text{pre}}+\epsilon_\pi^{\text{pre}})+\gamma^i\epsilon_\pi^{\text{pre}}
\end{aligned}$$

Next, to obtain a bound for the averaged branched rollout, we average this bound over k:

$$\begin{aligned}
&\frac{1}{k+1}\sum_{i=0}^{k} i(\epsilon_m^{\text{post}}+\epsilon_\pi^{\text{post}})+\epsilon_\pi^{\text{post}}+\frac{\gamma^{i+1}}{1-\gamma}(\epsilon_m^{\text{pre}}+\epsilon_\pi^{\text{pre}})+\gamma^i\epsilon_\pi^{\text{pre}} \\
&= \frac{1}{k+1}[\frac{k(k+1)}{2}(\epsilon_m^{\text{post}}+\epsilon_\pi^{\text{post}})+\frac{\gamma}{1-\gamma}\frac{1-\gamma^{k+1}}{1-\gamma}(\epsilon_m^{\text{pre}}+\epsilon_\pi^{\text{pre}})+\frac{1-\gamma^{k+1}}{1-\gamma}\epsilon_\pi^{\text{pre}}]+\epsilon_\pi^{\text{post}} \\
&\leq \frac{k}{2}(\epsilon_m^{\text{post}}+\epsilon_\pi^{\text{post}})+\frac{\gamma}{(k+1)(1-\gamma)^2}(\epsilon_m^{\text{pre}}+\epsilon_\pi^{\text{pre}})+\frac{1}{(k+1)(1-\gamma)}\epsilon_\pi^{\text{pre}}+\epsilon_\pi^{\text{post}}
\end{aligned}$$

Multiplying this bound by $\frac{2R_{\max}}{1-\gamma}$ to convert the state-marginal bound into a returns bound completes the proof.

□

B.3 Hyperparameter Settings

		Half Cheetah	Inverted Pendulum	Walker2d	Ant	Hopper	Humanoid
N	epochs	400	15	300		125	300
E	environment steps per epoch	1000					
M	model rollouts per environment step	400					
B	ensemble size	7					
	network architecture	MLP with 4 hidden layers of size 200					MLP with 4 hidden layers of size 400
G	policy updates per environment step	40	20				
k	model horizon	1			$1 \rightarrow 25$ over epochs $20 \rightarrow 100$	$1 \rightarrow 15$ over epochs $20 \rightarrow 100$	$1 \rightarrow 25$ over epochs $20 \rightarrow 300$

Table B.1: Hyperparameter settings for MBPO results shown in Figure 5.2. $x \rightarrow y$ over epochs $a \rightarrow b$ denotes a thresholded linear function, *i.e.* at epoch e, $f(e) = \min(\max(x + \frac{e-a}{b-a} \cdot (y - x), x), y)$

Appendix C

Model-Based Optimization

C.1 Experimental Details

Detailed Pseudocode

There are a number of additional details we implemented in order to improve the performance of NEMO for high-dimensional tasks. These include:

- Using the Adam optimizer [Kingma and Ba, 2015] rather than stochastic gradient descent.
- Pretraining the models θ in order to initialize the procedure with an accurate initial model.
- Optimizing over a batch of M designs, in order to follow previous evaluation protocols.
- Optimizing with x with respect to the internal scores μ instead of $E_{p_{\text{NML}}}[g(y)]$.
- Using target networks for the NML model, originally proposed in reinforcement learning algorithms, to improve the stability of the method.

We present pseudocode for the practical implementation of NEMO below:

Hyperparameters

The following table lists the hyperparameter settings we used for each task. We obtained our hyperparameter settings by performing a grid search across different settings of α_θ, α_x, and τ. We used 2-layer neural networks with softplus activations for all experiments. We used a smaller networks for GFP, Hopper, Ant, and DKitty (64-dimensional layers) and a lower discretization K for computational performance reasons, but we did not tune over these parameters.

For baseline methods, please refer to Trabucco et al. [2021] for hyperparameter settings.

Algorithm 6 NEMO – Practical Instantiation

Input Model class Θ, Dataset $\mathcal{D} = (\boldsymbol{x}_{1:N}, y_{1:N})$, number of bins K, batch size M, learning rates α_θ, α_x, target update rate τ
Initialize K models $\theta_0^{1:K}$
Initialize batch of optimization iterates $\mathcal{B}_0 = \boldsymbol{x}_0^{1:M}$ from the **best** performing $\boldsymbol{x}$ in $\mathcal{D}$.
Pretrain $\theta_0^{1:K}$ using supervised learning on $\mathcal{D}$.
Initialize target networks $\bar{\theta}_0^{1:K} \leftarrow \theta_0^{1:K}$.
Quantize $y_{1:N}$ into K bins, denoted as $\lfloor \mathcal{Y} \rfloor = \{\lfloor y_1 \rfloor, \cdots \lfloor y_k \rfloor\}$.
for iteration t in $1 \dots T$ **do**
  for k in $1 \dots K$ **do**
    Sample $\boldsymbol{x}'_t$ from batch $\mathcal{B}_t$.
    Construct Augmented dataset: $\mathcal{D}' \leftarrow \mathcal{D} \cup (\boldsymbol{x}'_t, \lfloor y_k \rfloor)$
    Compute gradient $g_\theta \leftarrow \nabla_{\theta_t^y} \text{LogLikelihood}(\theta_t^y, \mathcal{D}')$
    Update model θ_t^y using Adam and g_θ with learning rate α_θ.
  end for
  Update target networks $\bar{\theta}_{t+1}^y \leftarrow \tau \theta_{t+1}^y + (1-\tau)\bar{\theta}_t^y$
  for m in $1 \dots M$ **do**
    Compute internal values $\mu^k(\boldsymbol{x}_t^m)$ from target networks $\bar{\theta}_t^y$ for all $k \in 1, \cdots, K$
    Compute gradient g_x for $\boldsymbol{x}_t^m$: $\nabla_x \frac{1}{K} \sum_k \mu^k(\boldsymbol{x}_t^m)$
    Update $\boldsymbol{x}_t^m$ using Adam and gradient g_x with learning rate α_x.
  end for
end for

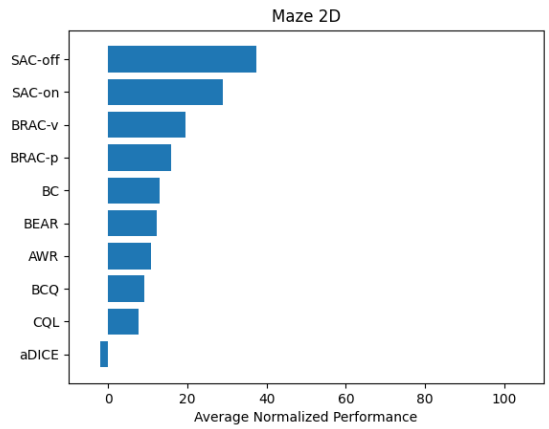
	Superconductor	MoleculeActivity	GFP	Hopper	Ant	DKitty
Learning rate α_θ	0.05		0.005			
Learning rate α_x	0.1		0.01	0.001		
Network Size	256,256		64,64			
Discretization K	40		20			
Batch size M	128					
Target update rate τ	0.05					

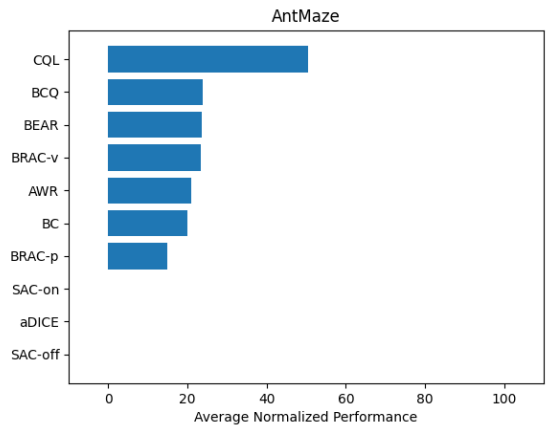
Appendix D

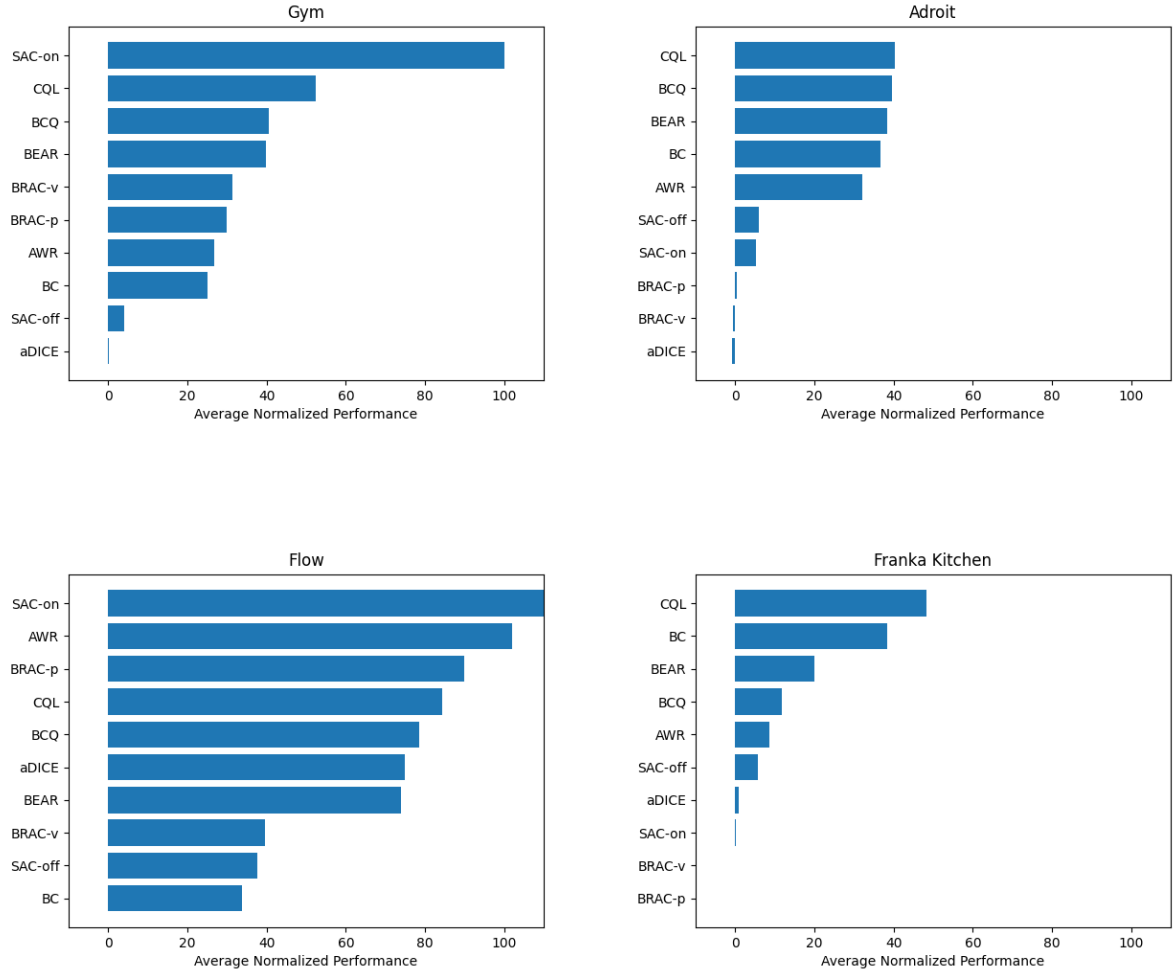
Benchmarking Offline Reinforcement Learning

D.1 Results by Domain

The following tables summarize performance for each domain (excluding CARLA, due to all algorithms performing poorly), sorted by the best performing algorithm to the worst.







D.2 Task Properties

The following is a full list of task properties and dataset statistics for all tasks in the benchmark. Note that the full dataset for "carla-town" requires over 30GB of memory to store, so we also provide a subsampled version of the dataset which we used in our experiments.

Domain	Task Name	Controller Type	# Samples
Maze2D	maze2d-umaze	Planner	10^6
	maze2d-medium	Planner	$2 * 10^6$
	maze2d-large	Planner	$4 * 10^6$
AntMaze	antmaze-umaze	Planner	10^6
	antmaze-umaze-diverse	Planner	10^6
	antmaze-medium-play	Planner	10^6
	antmaze-medium-diverse	Planner	10^6
	antmaze-large-play	Planner	10^6
	antmaze-large-diverse	Planner	10^6
Gym-MuJoCo	hopper-random	Policy	10^6
	hopper-medium	Policy	10^6
	hopper-medium-replay	Policy	200920
	hopper-medium-expert	Policy	2×10^6
	halfcheetah-random	Policy	10^6
	halfcheetah-medium	Policy	10^6
	halfcheetah-medium-replay	Policy	101000
	halfcheetah-medium-expert	Policy	2×10^6
	walker2d-random	Policy	10^6
	walker2d-medium	Policy	10^6
	walker2d-medium-replay	Policy	100930
	walker2d-medium-expert	Policy	2×10^6
Adroit	pen-human	Human	5000
	pen-cloned	Policy	$5 * 10^5$
	pen-expert	Policy	$5 * 10^5$
	hammer-human	Human	11310
	hammer-cloned	Policy	10^6
	hammer-expert	Policy	10^6
	door-human	Human	6729
	door-cloned	Policy	10^6
	door-expert	Policy	10^6
	relocate-human	Human	9942
	relocate-cloned	Policy	10^6
	relocate-expert	Policy	10^6
Flow	flow-ring-random	Policy	10^6
	flow-ring-controller	Policy	10^6
	flow-merge-random	Policy	10^6
	flow-merge-controller	Policy	10^6
FrankaKitchen	kitchen-complete	Human	3680
	kitchen-partial	Human	136950
	kitchen-mixed	Human	136950
CARLA	carla-lane	Planner	10^5
	carla-town	Planner	$2 * 10^6$ full 10^5 subsampled

Table D.1: Statistics for each task in the benchmark. For the controller type, "planner" refers to a hand-designed navigation planner, "human" refers to human demonstrations, and "policy" refers to random or neural network policies.

D.3 Task and Datasets

The following table lists the tasks and dataset types included in the benchmark, including sources for each.

Domain	**Source**	**Dataset Types**
Maze2D	N/A	UMaze, Medium, Large
AntMaze	N/A	UMaze, Diverse, Play
Gym-MuJoCo	Brockman et al. [2016] Todorov et al. [2012]	Expert, Random, Medium (Various) Medium-Expert, Medium-Replay
Adroit	Rajeswaran et al. [2018]	Human, Expert [Rajeswaran et al., 2018] Cloned
Flow	Wu et al. [2017]	Random, Controller
Franka Kitchen	Gupta et al. [2020]	Complete, Partial, Mixed [Gupta et al., 2020]
CARLA	Dosovitskiy et al. [2017]	Controller

Table D.2: Domains and dataset types contained within our benchmark. Maze2D and AntMaze are new domains we propose. For each dataset, we also include references to the source if originally proposed in another work. Datasets borrowed from prior work include MuJoCo (Expert, Random, Medium), Adroit (Human, Expert), and FrankaKitchen (Complete, Partial, Mixed). All other datasets are datasets proposed by this work.

D.4 Training and Evaluation Task Split

The following table lists our recommended protocol for hyperparameter tuning. Hyperparameters should be tuned on the tasks listed on the left in the "Training" column, and algorithms should be evaluated without tuning on the tasks in the right column labeled "Evaluation".

Domain	Training	Evaluation
Maze2D	maze2d-umaze	maze2d-eval-umaze
	maze2d-medium	maze2d-eval-medium
	maze2d-large	maze2d-eval-large
AntMaze	ant-umaze	ant-eval-umaze
	ant-umaze-diverse	ant-eval-umaze-diverse
	ant-medium-play	ant-eval-medium-play
	ant-medium-diverse	ant-eval-medium-diverse
	ant-large-play	ant-eval-large-play
	ant-large-diverse	ant-eval-large-diverse
Adroit	pen-human	hammer-human
	pen-cloned	hammer-cloned
	pen-expert	hammer-expert
	door-human	relocate-human
	door-cloned	relocate-cloned
	door-expert	relocate-expert
Gym	halfcheetah-random	hopper-random
	halfcheetah-medium	hopper-medium
	halfcheetah-mixed	hopper-mixed
	halfcheetah-medium-expert	hopper-medium-expert
	walker2d-random	ant-random
	walker2d-medium	ant-medium
	walker2d-mixed	ant-mixed
	walker2d-medium-expert	and-medium-expert

Table D.3: Our recommended partition of tasks into "training" tasks where hyperparameter tuning is allowed, and "evaluation" tasks where final algorithm performance should be reported.

D.5 Experiment Details

For all experiments, we used default hyperparameter settings and minimal modifications to public implementations wherever possible, using 500K training iterations or gradient steps. The code bases we used for evaluation are listed below. We ran our experiments using Google cloud platform (GCP) on `n1-standard-4` machines.

- BRAC and AlgaeDICE: `https://github.com/google-research/google-research`
- AWR: `https://github.com/xbpeng/awr`
- SAC: `https://github.com/vitchyr/rlkit`
- BEAR: `https://github.com/aviralkumar2907/BEAR`

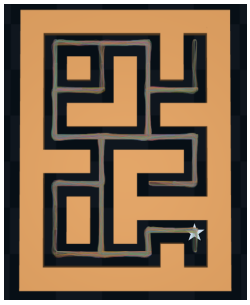
- BCQ: `https://github.com/sfujim/BCQ`
- CQL: `https://github.com/aviralkumar2907/CQL`

D.6 Assessing the Feasibility of Hard Tasks

Few prior methods were able to successfully solve carla-town or the larger AntMaze tasks. While including tasks that present a challenge for current methods is important to ensure that our benchmark has room for improvement, it is also important to provide some confidence that the tasks are actually solvable. We verified this in two ways. First, we ensured that the trajectories observed in these tasks have adequate coverage of the state space. An illustration of the trajectories in the CARLA and AntMaze tasks are shown below, where trajectories are shown as different colored lines and the goal state is marked with a star. We see that in carla-town and AntMaze, each lane or corridor is traversed multiple times by the agent.

(a) carla-town

(b) antmaze-large



(c) maze2d-large

Second, the data in AntMaze was generated by having the ant follow the same high-level planner in the maze as in Maze2D. Because Maze2D is solvable by most methods, we would expect this to mean that AntMaze is potentially solvable as well. This While the dynamics of the ant itself are much more complex, its walking gait is a relatively regular periodic motion, and since the high-level waypoints are similar, we would expect the AntMaze data to provide similar coverage as in the 2D mazes, as shown in the figures on the right. While the Ant has more erratic motion, both datasets cover the the majority of the maze thoroughly. A comparison of the state coverage between Maze2D and AntMaze on all tasks is shown in the following Appendix section D.7.

D.7 Maze Domain Trajectories

In this section, we visualized trajectories for the datasets in the Maze2D and AntMaze domains. Each image plots the states visited along each trajectory as a different colored line, overlaid on top of the maze. The goal state is marked as a white star.

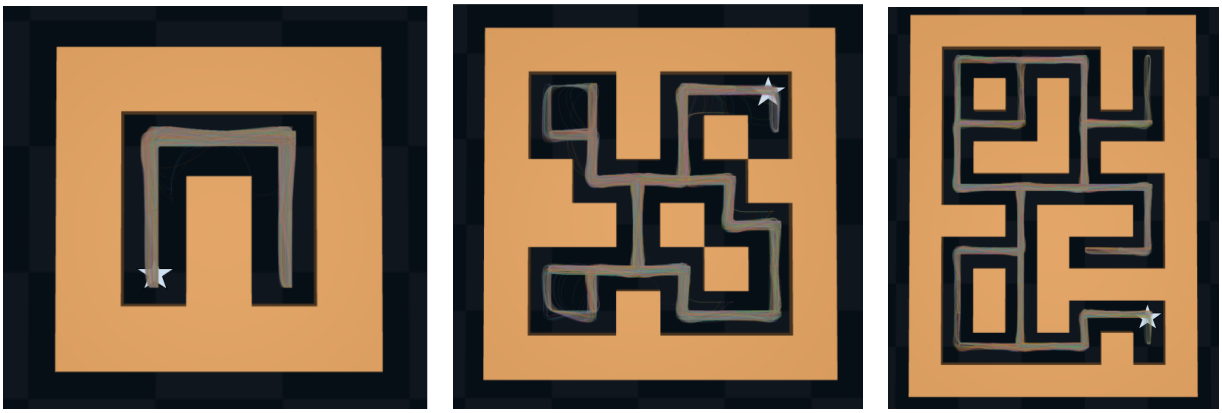


Figure D.2: Trajectories visited in the Maze2D domain. From left-to-right: maze2d-umaze, maze2d-medium, and maze2d-large.

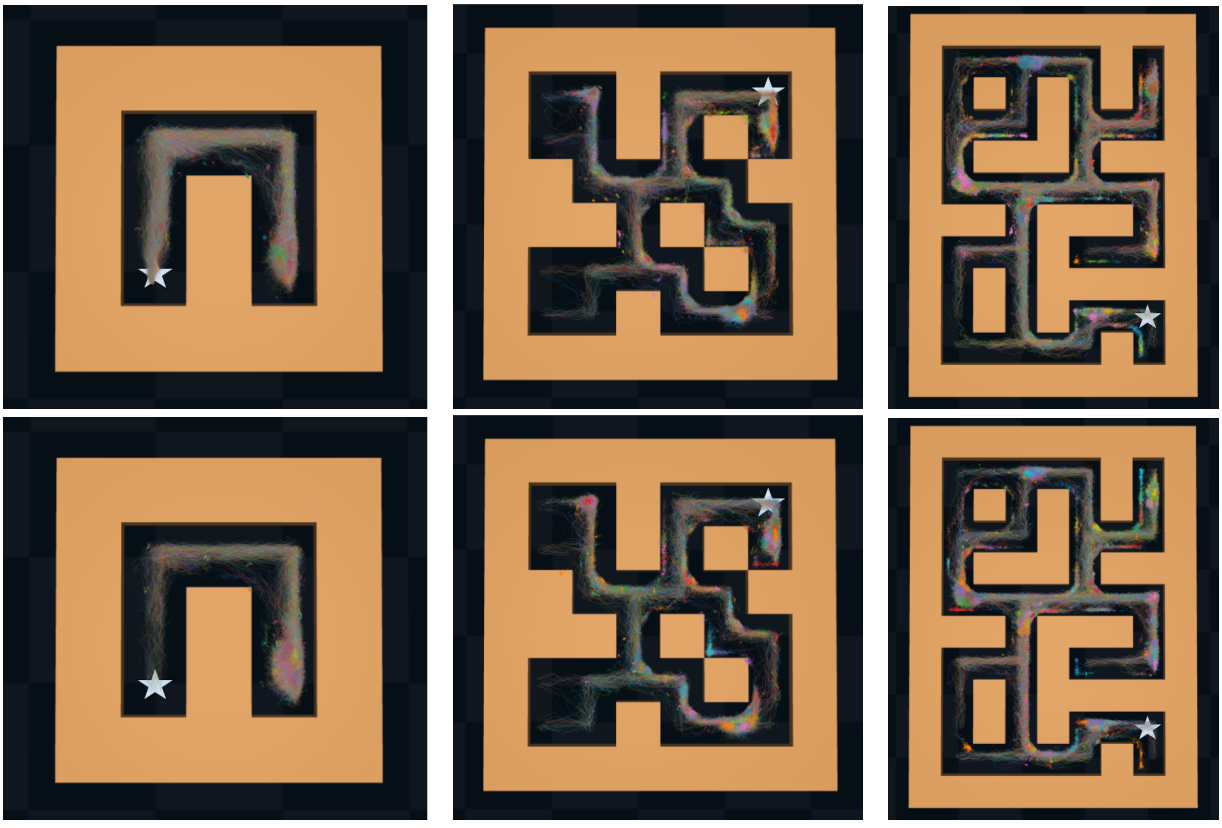


Figure D.3: Trajectories visited in the AntMaze domain. Top row, from left-to-right: antmaze-umaze, antmaze-medium-play, and antmaze-large-play. Bottom row, from left-to-right: antmaze-umaze-diverse, antmaze-medium-diverse, and antmaze-large-diverse.

Appendix E

Benchmarking Off-Policy Evaluation

E.1 Off-policy Evaluation Metrics

The metrics we use are defined as follows:

Absolute Error We evaluate policies using absolute error in order to be robust to outliers. The absolute error is defined as the difference between the value and estimated value of a policy:

$$\text{AbsErr} = |V^\pi - \hat{V}^\pi| \tag{E.1}$$

Where V^π is the true value of the policy, and $\hat{V}^\pi$ is the estimated value of the policy.

Regret@k Regret@k is the difference between the value of the best policy in the entire set, and the value of the best policy in the top-k set (where the top-k set is chosen by estimated values). It can be defined as:

$$\text{Regret @ k} = \max_{i \in 1:N} V_i^\pi - \max_{j \in \text{topk}(1:N)} V_j^\pi \tag{E.2}$$

Where topk$(1:N)$ denotes the indices of the top K policies as measured by estimated values $\hat{V}^\pi$.

Rank correlation Rank correlation (also Spearman's ρ) measures the correlation between the ordinal rankings of the value estimates and the true values. It can be written as:

$$\text{RankCorr} = \frac{\text{Cov}(V_{1:N}^\pi, \hat{V}_{1:N}^\pi)}{\sigma(V_{1:N}^\pi)\sigma(\hat{V}_{1:N}^\pi)} \tag{E.3}$$

E.2 Detailed Off-policy Evaluation Results

Detailed results figures and tables are presented here. We show results by task in both tabular and chart form, as well as scatter plots which compare the estimated returns against

the ground truth returns for every policy.

Chart Results

First we show the normalized results for each algorithm and task.

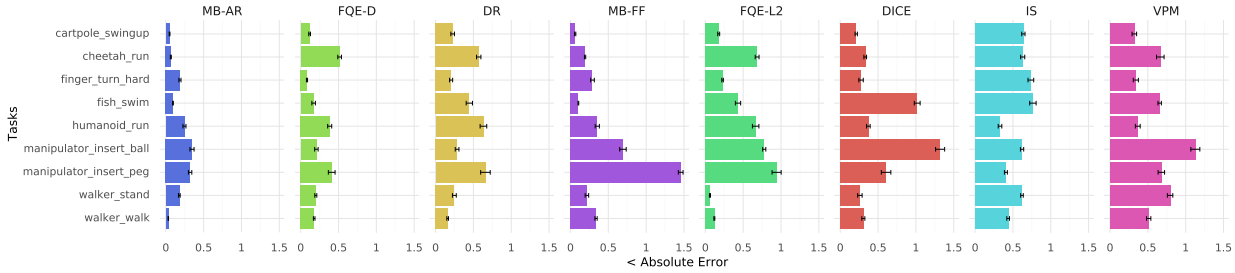


Figure E.1: Absolute error for each baseline algorithm for each RL Unplugged task considered.

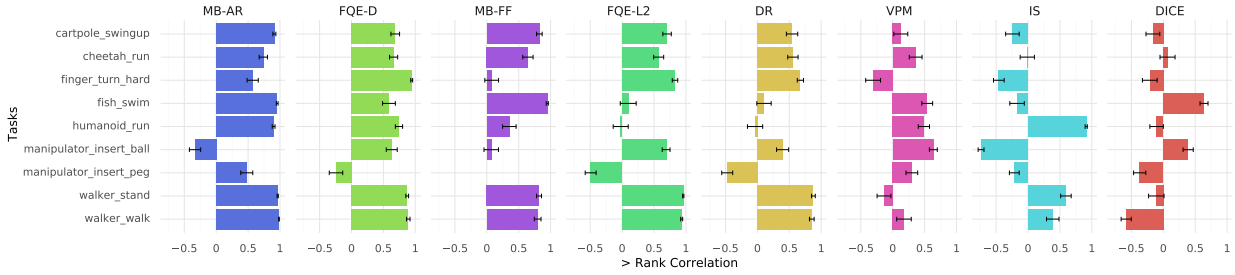


Figure E.2: Rank correlation for each baseline algorithm for each RL Unplugged task considered.

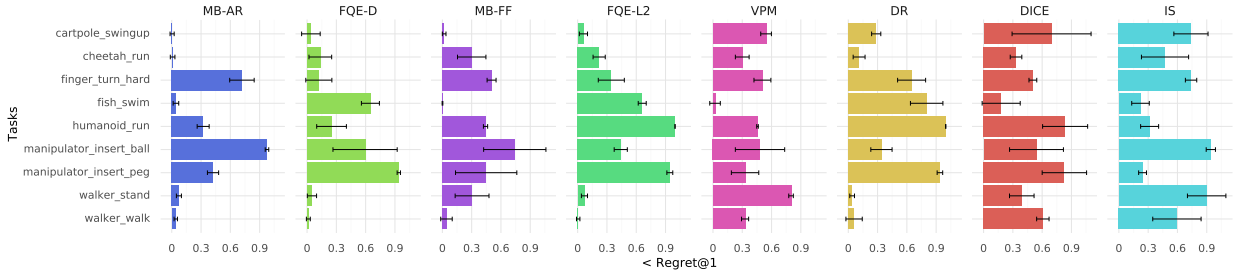


Figure E.3: Regret@1 for each baseline algorithm for each RL Unplugged task considered.

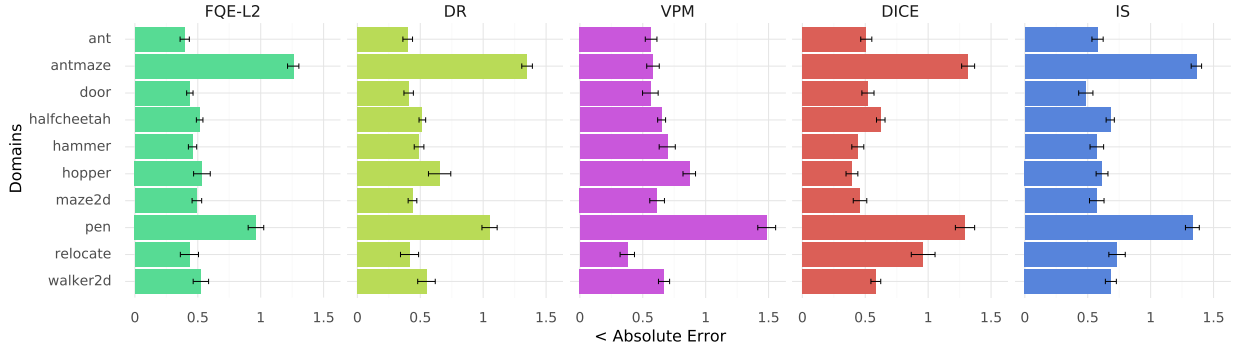


Figure E.4: Absolute error for each baseline algorithm for each D4RL task domain considered.

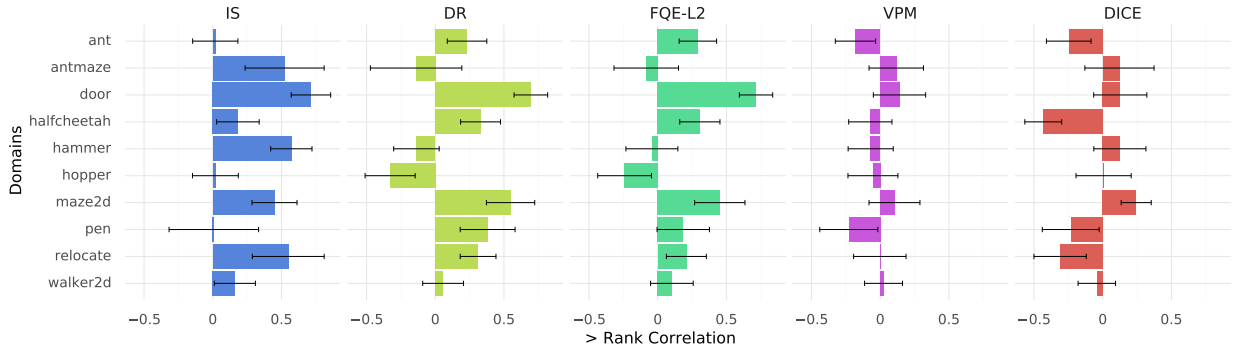


Figure E.5: Rank correlation for each baseline algorithm for each D4RL task domain considered.

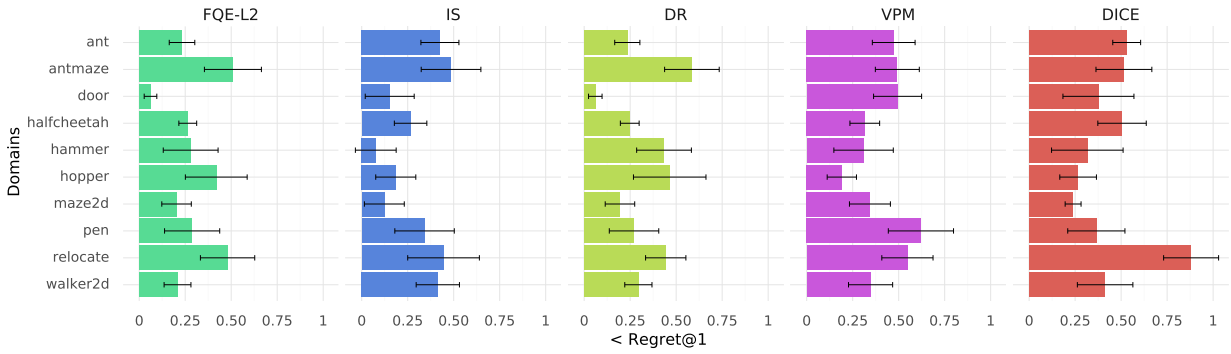


Figure E.6: Regret@1 for each baseline algorithm for each D4RL task domain considered.

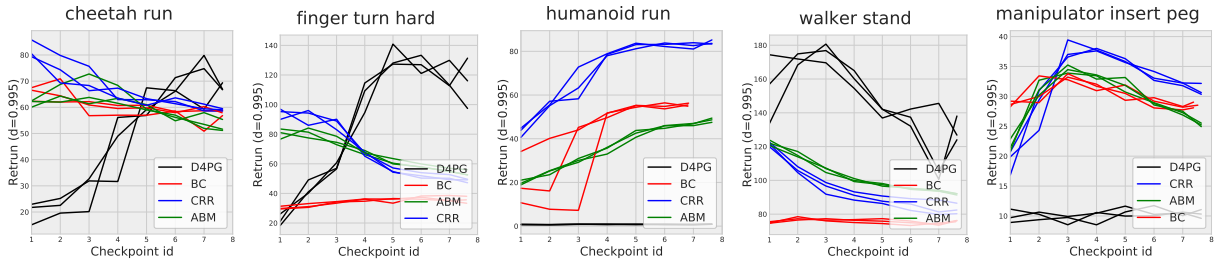


Figure E.7: Online evaluation of policy checkpoints for 4 Offline RL algorithms with 3 random seeds. We observe a large degree of variability between the behavior of algorithms on different tasks.

Tabular Results

Next, we present the results for each task and algorithm in tabular form, with means and standard deviations reported across 3 seeds.

		Cartpole swingup	Cheetah run	Finger turn hard	Fish swim	Humanoid run
Absolute Error btw. OPE and ground truth	Variational power method	37.53 ±3.50	61.89 ±4.25	46.22 ±3.93	31.27 ±0.99	35.29 ±3.03
	Importance Sampling	68.75 ±2.39	44.29 ±1.91	90.10 ±4.68	34.82 ±1.93	27.89 ±1.98
	Best DICE	22.73 ±1.65	23.35 ±1.32	33.52 ±3.48	59.48 ±2.47	31.42 ±2.04
	Model based - FF	**6.80 ±0.85**	13.64 ±0.59	35.99 ±3.00	**4.75 ±0.23**	30.12 ±2.40
	FQE (L2)	19.02 ±1.34	48.26 ±1.78	27.91 ±1.18	19.82 ±1.57	56.28 ±3.52
	Doubly Robust (IS, FQE)	24.38 ±2.51	40.27 ±2.05	25.26 ±2.48	20.28 ±1.90	53.64 ±3.68
	FQE (distributional)	12.63 ±1.21	36.50 ±1.62	**10.23 ±0.93**	7.76 ±0.95	32.36 ±2.27
	Model based - AR	**5.32 ±0.54**	**4.64 ±0.46**	22.93 ±1.72	**4.31 ±0.22**	**20.95 ±1.61**

		Walker stand	Walker walk	Manipulator insert ball	Manipulator insert peg	Median ↓
Absolute Error btw. OPE and ground truth	Variational power method	96.76 ±3.59	87.24 ±4.25	79.25 ±6.19	21.95 ±1.17	46.22
	Importance Sampling	66.50 ±1.90	67.24 ±2.70	29.93 ±1.10	12.78 ±0.66	44.29
	Best DICE	27.58 ±3.01	47.28 ±3.13	103.45 ±5.21	22.75 ±3.00	31.42
	Model based - FF	23.34 ±2.41	52.23 ±2.34	34.30 ±2.55	121.12 ±1.58	30.12
	FQE (L2)	**6.51 ±0.71**	18.34 ±0.95	36.32 ±1.07	31.12 ±2.37	27.91
	Doubly Robust (IS, FQE)	26.82 ±2.66	24.63 ±1.69	13.33 ±1.16	22.28 ±2.34	24.63
	FQE (distributional)	21.49 ±1.41	27.57 ±1.54	**9.75 ±1.10**	12.66 ±1.39	12.66
	Model based - AR	19.12 ±1.23	**5.14 ±0.49**	17.13 ±1.34	**9.71 ±0.70**	9.71

Table E.1: Average absolute error between OPE metrics and ground truth values at a discount factor of 0.995 In each column, absolute error values that are not significantly different from the best ($p > 0.05$) are bold faced. Methods are ordered by median.

		Cartpole swingup	Cheetah run	Finger turn hard	Fish swim	Humanoid run
Rank Correlation btw. OPE and ground truth	Importance Sampling	−0.23 ±0.11	−0.01 ±0.12	−0.45 ±0.08	−0.17 ±0.11	**0.91 ±0.02**
	Best DICE	−0.16 ±0.11	0.07 ±0.11	−0.22 ±0.11	0.44 ±0.09	−0.10 ±0.10
	Variational power method	0.01 ±0.11	0.01 ±0.12	−0.25 ±0.11	0.56 ±0.08	0.36 ±0.09
	Doubly Robust (IS, FQE)	0.55 ±0.09	0.56 ±0.08	0.67 ±0.05	0.11 ±0.12	−0.03 ±0.12
	Model based - FF	0.83 ±0.05	**0.64 ±0.08**	0.08 ±0.11	**0.95 ±0.02**	0.35 ±0.10
	FQE (distributional)	0.69 ±0.07	**0.67 ±0.06**	**0.94 ±0.01**	0.59 ±0.10	0.74 ±0.06
	FQE (L2)	0.70 ±0.07	0.56 ±0.08	0.83 ±0.04	0.10 ±0.12	−0.02 ±0.12
	Model based - AR	**0.91 ±0.02**	**0.74 ±0.07**	0.57 ±0.09	**0.96 ±0.01**	**0.90 ±0.02**

		Walker stand	Walker walk	Manipulator insert ball	Manipulator insert peg	Median ↑
Rank Correlation btw. OPE and ground truth	Importance Sampling	0.59 ±0.08	0.38 ±0.10	−0.72 ±0.05	−0.25 ±0.08	−0.17
	Best DICE	−0.11 ±0.12	−0.58 ±0.08	0.19 ±0.11	−0.35 ±0.10	−0.11
	Variational power method	−0.35 ±0.10	−0.10 ±0.11	**0.61 ±0.08**	**0.41 ±0.09**	0.01
	Doubly Robust (IS, FQE)	0.88 ±0.03	0.85 ±0.04	0.42 ±0.10	−0.47 ±0.09	0.55
	Model based - FF	0.82 ±0.04	0.80 ±0.05	0.06 ±0.10	−0.56 ±0.08	0.64
	FQE (distributional)	0.87 ±0.02	0.89 ±0.03	**0.63 ±0.08**	−0.23 ±0.10	0.69
	FQE (L2)	**0.96 ±0.01**	0.94 ±0.02	**0.70 ±0.07**	−0.48 ±0.08	0.70
	Model Based - AR	**0.96 ±0.01**	**0.98 ±0.00**	−0.33 ±0.09	**0.47 ±0.09**	0.90

Table E.2: Spearman's rank correlation (ρ) coefficient (bootstrap mean ± standard deviation) between different OPE metrics and ground truth values at a discount factor of 0.995. In each column, rank correlation coefficients that are not significantly different from the best ($p > 0.05$) are bold faced. Methods are ordered by median. Also see Table E.3 and Table E.1 for Normalized Regret@5 and Average Absolute Error results.

		Cartpole swingup	Cheetah run	Finger turn hard	Fish swim	Humanoid run
Regret@5 for OPE *vs.* ground truth	Importance Sampling	0.73 ±0.16	0.40 ±0.21	0.64 ±0.05	0.12 ±0.05	**0.31 ±0.09**
	Best DICE	0.68 ±0.41	0.27 ±0.05	0.44 ±0.04	**0.35 ±0.24**	0.84 ±0.22
	Variational power method	0.50 ±0.13	0.37 ±0.04	0.45 ±0.13	**0.02 ±0.02**	0.56 ±0.08
	Doubly Robust (IS, FQE)	0.28 ±0.05	**0.09 ±0.05**	0.56 ±0.12	0.61 ±0.12	0.99 ±0.00
	FQE (L2)	0.06 ±0.04	0.17 ±0.05	**0.30 ±0.11**	0.50 ±0.03	0.99 ±0.00
	Model based - FF	**0.02 ±0.02**	0.24 ±0.12	0.43 ±0.04	**0.00 ±0.00**	0.44 ±0.02
	FQE (distributional)	**0.03 ±0.09**	**0.11 ±0.09**	**0.10 ±0.12**	0.49 ±0.06	**0.24 ±0.15**
	Model based - AR	**0.00 ±0.02**	**0.01 ±0.02**	0.63 ±0.11	**0.03 ±0.02**	**0.32 ±0.06**

		Walker stand	Walker walk	Manipulator insert ball	Manipulator insert peg	Median ↓
Regret@5 for OPE *vs.* ground truth	Importance Sampling	0.54 ±0.11	0.54 ±0.23	0.83 ±0.05	**0.22 ±0.03**	0.54
	Best DICE	0.24 ±0.07	0.55 ±0.06	**0.44 ±0.07**	0.75 ±0.04	0.44
	Variational power method	0.41 ±0.02	0.39 ±0.02	**0.52 ±0.20**	0.32 ±0.02	0.41
	Doubly Robust (IS, FQE)	**0.02 ±0.01**	**0.05 ±0.07**	**0.30 ±0.10**	0.73 ±0.01	0.30
	FQE (L2)	**0.04 ±0.02**	**0.00 ±0.02**	**0.37 ±0.07**	0.74 ±0.01	0.30
	Model based - FF	**0.18 ±0.10**	**0.03 ±0.05**	0.83 ±0.06	0.74 ±0.01	0.24
	FQE (distributional)	**0.03 ±0.03**	**0.01 ±0.02**	**0.50 ±0.30**	0.73 ±0.01	0.11
	Model based - AR	**0.04 ±0.02**	**0.04 ±0.02**	0.85 ±0.02	**0.30 ±0.04**	0.04

Table E.3: Normalized Regret@5 (bootstrap mean ± standard deviation) for OPE methods *vs.* ground truth values at a discount factor of 0.995. In each column, normalized regret values that are not significantly different from the best ($p > 0.05$) are bold faced. Methods are ordered by median.

		Halfcheetah expert	Halfcheetah medium	Halfcheetah medium-expert	Halfcheetah medium-replay	Halfcheetah random
Abs. Error	IS	1404 ±152	**1217 ±123**	1400 ±146	1409 ±154	1405 ±155
	VPM	**945 ±164**	**1374 ±153**	1427 ±111	1384 ±148	1411 ±154
	Best DICE	**944 ±161**	**1382 ±130**	**1078 ±132**	1440 ±158	1446 ±156
	Doubly Robust	**1025 ±95**	**1222 ±134**	**1015 ±103**	**1001 ±129**	**949 ±126**
	FQE (L2)	**1031 ±95**	**1211 ±130**	**1014 ±101**	**1003 ±132**	**938 ±125**

		Antmaze large-diverse	Antmaze large-play	Antmaze medium-diverse	Antmaze medium-play	Antmaze umaze
Abs. Error	IS	0.62 ±0.01	0.85 ±0.00	0.55 ±0.01	0.81 ±0.00	0.62 ±0.04
	VPM	**0.02 ±0.02**	**0.26 ±0.24**	**0.07 ±0.05**	**0.11 ±0.06**	**0.12 ±0.03**
	Best DICE	5.55 ±0.36	19.62 ±1.28	2.42 ±1.56	19.47 ±2.15	14.97 ±1.93
	Doubly Robust	0.99 ±0.01	1.59 ±0.01	0.61 ±0.03	1.47 ±0.01	0.87 ±0.04
	FQE (L2)	0.53 ±0.01	0.78 ±0.00	0.29 ±0.01	0.71 ±0.01	0.39 ±0.03

		Antmaze umaze-diverse	Door cloned	Door expert	Door human	Hammer cloned
Abs. Error	IS	**0.14 ±0.02**	891 ±188	**648 ±122**	870 ±173	7403 ±1126
	VPM	**0.12 ±0.03**	1040 ±188	**879 ±182**	862 ±163	7459 ±1114
	Best DICE	**0.17 ±0.04**	697 ±79	**856 ±134**	1108 ±199	**4169 ±839**
	Doubly Robust	**0.11 ±0.02**	**424 ±73**	1353 ±218	**379 ±65**	6101 ±679
	FQE (L2)	**0.11 ±0.03**	**438 ±81**	1343 ±84	**389 ±60**	**5415 ±558**

		Hammer expert	Hammer human	Maze2d large	Maze2d medium	Maze2d umaze
Abs. Error	IS	**3052 ±608**	**7352 ±1118**	45.61 ±10.43	61.29 ±7.78	50.20 ±9.16
	VPM	7312 ±1117	**7105 ±1107**	44.10 ±10.69	60.30 ±8.37	62.81 ±8.40
	Best DICE	**3963 ±758**	**5677 ±936**	**42.46 ±9.66**	58.97 ±9.57	**21.95 ±4.69**
	Doubly Robust	**3485 ±590**	**5768 ±751**	**22.94 ±6.82**	**23.64 ±4.96**	76.93 ±4.42
	FQE (L2)	**2950 ±728**	**6000 ±612**	**24.31 ±6.56**	**35.11 ±6.33**	79.67 ±4.93

		Pen cloned	Pen expert	Pen human	Relocate cloned	Relocate expert
Abs. Error	IS	1707 ±128	4547 ±222	3926 ±128	**632 ±215**	2731 ±147
	VPM	2324 ±129	2325 ±136	**1569 ±215**	**586 ±135**	**620 ±214**
	Best DICE	**1454 ±219**	2963 ±279	4193 ±244	1347 ±485	**1095 ±221**
	Doubly Robust	**1323 ±98**	**2013 ±564**	2846 ±200	**412 ±124**	**1193 ±350**
	FQE (L2)	**1232 ±105**	**1057 ±281**	2872 ±170	**439 ±125**	1351 ±393

		Relocate human	Ant expert	Ant medium	Ant medium-expert	Ant medium-replay
Abs. Error	IS	**638 ±217**	**605 ±104**	594 ±104	604 ±102	**603 ±101**
	VPM	**806 ±166**	**607 ±108**	570 ±109	604 ±106	**612 ±105**
	Best DICE	4526 ±474	**558 ±108**	**495 ±90**	**471 ±100**	**583 ±110**
	Doubly Robust	**606 ±116**	**584 ±114**	**345 ±66**	**326 ±66**	**421 ±72**
	FQE (L2)	**593 ±113**	**583 ±122**	**345 ±64**	**319 ±67**	**410 ±79**

		Ant random	Hopper expert	Hopper medium	Hopper random	Walker2d expert
Abs. Error	IS	**606 ±103**	**106 ±29**	405 ±48	412 ±45	**405 ±62**
	VPM	**570 ±99**	442 ±43	433 ±44	438 ±44	**367 ±68**
	Best DICE	**530 ±92**	259 ±54	**215 ±41**	**122 ±16**	**437 ±60**
	Doubly Robust	**404 ±106**	426 ±99	**307 ±85**	289 ±50	**519 ±179**
	FQE (L2)	**398 ±111**	282 ±76	**283 ±73**	261 ±42	**453 ±142**

		Walker2d medium	Walker2d medium-expert	Walker2d medium-replay	Walker2d random	Median
Abs. Error	IS	428 ±60	436 ±62	**427 ±60**	**430 ±61**	603.82
	VPM	426 ±60	425 ±61	**424 ±64**	**440 ±58**	585.53
	Best DICE	**273 ±31**	**322 ±60**	**374 ±51**	**419 ±57**	530.43
	Doubly Robust	**368 ±74**	**217 ±46**	**296 ±54**	**347 ±74**	411.99
	FQE (L2)	**350 ±79**	**233 ±42**	**313 ±73**	**354 ±73**	398.37

		Halfcheetah expert	Halfcheetah medium-expert	Halfcheetah medium-replay	Halfcheetah random	Door cloned
Rank Corr.	Best DICE	−0.44 ±0.30	**−0.08 ±0.35**	**−0.15 ±0.41**	−0.70 ±0.22	**0.18 ±0.31**
	VPM	**0.18 ±0.35**	−0.47 ±0.29	**−0.07 ±0.36**	**0.27 ±0.36**	−0.29 ±0.36
	FQE (L2)	**0.78 ±0.15**	**0.62 ±0.27**	**0.26 ±0.37**	**−0.11 ±0.41**	**0.55 ±0.27**
	IS	0.01 ±0.35	**−0.06 ±0.37**	**0.59 ±0.26**	**−0.24 ±0.36**	**0.66 ±0.22**
	Doubly Robust	**0.77 ±0.17**	**0.62 ±0.27**	**0.32 ±0.37**	**−0.02 ±0.38**	**0.60 ±0.28**

		Door expert	Hammer cloned	Hammer expert	Maze2d large	Maze2d medium
Rank Corr.	Best DICE	−0.06 ±0.32	**0.35 ±0.38**	−0.42 ±0.31	**0.56 ±0.21**	−0.64 ±0.23
	VPM	**0.65 ±0.23**	−0.77 ±0.22	**0.39 ±0.31**	−0.26 ±0.33	**−0.05 ±0.39**
	FQE (L2)	**0.89 ±0.09**	**−0.15 ±0.33**	**0.29 ±0.34**	**0.30 ±0.36**	**0.16 ±0.38**
	IS	**0.76 ±0.17**	**0.58 ±0.27**	**0.64 ±0.24**	**0.63 ±0.19**	**0.44 ±0.25**
	Doubly Robust	**0.76 ±0.13**	−0.70 ±0.20	**0.49 ±0.31**	**0.31 ±0.36**	**0.41 ±0.35**

		Pen expert	Relocate expert	Ant expert	Ant medium	Ant medium-expert
Rank Corr.	Best DICE	−0.53 ±0.30	−0.27 ±0.34	**−0.13 ±0.37**	−0.36 ±0.28	**−0.33 ±0.40**
	VPM	**0.08 ±0.33**	**0.39 ±0.31**	**−0.42 ±0.38**	−0.20 ±0.31	**−0.28 ±0.28**
	FQE (L2)	**−0.01 ±0.33**	−0.57 ±0.28	**−0.13 ±0.32**	**0.65 ±0.25**	**0.37 ±0.35**
	IS	−0.45 ±0.31	**0.52 ±0.23**	**0.14 ±0.41**	−0.17 ±0.32	**−0.21 ±0.35**
	Doubly Robust	**0.52 ±0.28**	−0.40 ±0.24	**−0.28 ±0.32**	**0.66 ±0.26**	**0.35 ±0.35**

		Ant medium-replay	Ant random	Hopper expert	Hopper medium	Hopper random
Rank Corr.	Best DICE	−0.24 ±0.39	**−0.21 ±0.35**	**−0.08 ±0.32**	**0.19 ±0.33**	**−0.13 ±0.39**
	VPM	−0.26 ±0.29	**0.24 ±0.31**	**0.21 ±0.32**	**0.13 ±0.37**	−0.46 ±0.20
	FQE (L2)	**0.57 ±0.28**	**0.04 ±0.33**	**−0.33 ±0.30**	**−0.29 ±0.33**	**−0.11 ±0.36**
	IS	**0.07 ±0.39**	**0.26 ±0.34**	**0.37 ±0.27**	−0.55 ±0.26	**0.23 ±0.34**
	Doubly Robust	**0.45 ±0.32**	**0.01 ±0.33**	−0.41 ±0.27	**−0.31 ±0.34**	**−0.19 ±0.36**

		Walker2d expert	Walker2d medium	Walker2d medium-expert	Walker2d medium-replay	Walker2d random
Rank Corr.	Best DICE	**−0.37 ±0.27**	**0.12 ±0.38**	**−0.34 ±0.34**	**0.55 ±0.23**	**−0.19 ±0.36**
	VPM	**0.17 ±0.32**	**0.44 ±0.21**	**0.49 ±0.37**	−0.52 ±0.25	**−0.42 ±0.34**
	FQE (L2)	**0.35 ±0.33**	**−0.09 ±0.36**	**0.25 ±0.32**	−0.19 ±0.36	**0.21 ±0.31**
	IS	**0.22 ±0.37**	**−0.25 ±0.35**	**0.24 ±0.33**	**0.65 ±0.24**	**−0.05 ±0.38**
	Doubly Robust	**0.26 ±0.34**	**0.02 ±0.37**	**0.19 ±0.33**	−0.37 ±0.39	**0.16 ±0.29**

		Median
Rank Corr.	Best DICE	−0.19
	VPM	−0.05
	FQE (L2)	0.21
	IS	0.23
	Doubly Robust	0.26

		Halfcheetah expert	Halfcheetah medium	Halfcheetah medium-expert	Halfcheetah medium-replay	Halfcheetah random
Regret@1	Best DICE	**0.32** ±0.40	0.82 ±0.29	**0.38** ±0.37	**0.30** ±0.07	0.81 ±0.30
	VPM	**0.14** ±0.09	**0.33** ±0.19	0.80 ±0.34	**0.25** ±0.09	**0.12** ±0.07
	Doubly Robust	**0.11** ±0.08	0.37 ±0.15	**0.14** ±0.07	**0.33** ±0.18	**0.31** ±0.10
	FQE (L2)	**0.12** ±0.07	0.38 ±0.13	**0.14** ±0.07	**0.36** ±0.16	0.37 ±0.08
	IS	**0.15** ±0.08	**0.05** ±0.05	**0.73** ±0.42	**0.13** ±0.10	**0.31** ±0.11

		Antmaze large-diverse	Antmaze large-play	Antmaze medium-diverse	Antmaze medium-play	Antmaze umaze
Regret@1	Best DICE	**0.54** ±0.34	**0.96** ±0.13	**0.04** ±0.11	**0.09** ±0.10	**0.69** ±0.39
	VPM	**0.88** ±0.27	**0.45** ±0.30	**0.14** ±0.10	**0.03** ±0.08	**0.62** ±0.32
	Doubly Robust	**0.83** ±0.30	**0.93** ±0.21	**0.05** ±0.07	**0.17** ±0.31	**0.42** ±0.36
	FQE (L2)	**0.93** ±0.25	**1.00** ±0.03	**0.16** ±0.10	**0.05** ±0.19	**0.41** ±0.35
	IS	**0.39** ±0.26	**0.71** ±0.20	**0.14** ±0.09	**0.18** ±0.06	**0.86** ±0.06

		Antmaze umaze-diverse	Door cloned	Door expert	Door human	Hammer cloned
Regret@1	Best DICE	**0.42** ±0.28	**0.65** ±0.45	0.37 ±0.27	**0.10** ±0.27	**0.67** ±0.48
	VPM	**0.63** ±0.32	**0.81** ±0.33	**0.03** ±0.03	0.69 ±0.24	0.72 ±0.39
	Doubly Robust	**0.79** ±0.14	**0.11** ±0.08	**0.05** ±0.07	**0.05** ±0.09	0.78 ±0.38
	FQE (L2)	**0.64** ±0.37	**0.11** ±0.06	**0.03** ±0.03	**0.05** ±0.08	**0.36** ±0.39
	IS	**0.22** ±0.36	**0.02** ±0.07	**0.01** ±0.04	**0.45** ±0.40	**0.03** ±0.15

		Hammer expert	Hammer human	Maze2d large	Maze2d medium	Maze2d umaze
Regret@1	Best DICE	**0.24** ±0.34	**0.04** ±0.08	**0.15** ±0.08	**0.44** ±0.05	**0.03** ±0.07
	VPM	**0.04** ±0.07	**0.18** ±0.29	0.66 ±0.10	**0.24** ±0.24	**0.06** ±0.12
	Doubly Robust	**0.09** ±0.09	0.46 ±0.23	**0.21** ±0.16	**0.27** ±0.14	**0.03** ±0.07
	FQE (L2)	0.05 ±0.04	0.46 ±0.23	**0.20** ±0.14	**0.31** ±0.14	**0.03** ±0.07
	IS	**0.01** ±0.04	**0.19** ±0.30	**0.16** ±0.23	**0.15** ±0.15	**0.02** ±0.12

		Pen cloned	Pen expert	Pen human	Relocate cloned	Relocate expert
Regret@1	Best DICE	**0.12** ±0.08	**0.33** ±0.20	**0.04** ±0.09	**0.96** ±0.18	0.97 ±0.07
	VPM	**0.36** ±0.18	**0.25** ±0.13	**0.28** ±0.12	**0.11** ±0.29	**0.76** ±0.23
	Doubly Robust	**0.13** ±0.06	**0.05** ±0.07	**0.09** ±0.08	**0.18** ±0.27	0.98 ±0.08
	FQE (L2)	**0.12** ±0.07	**0.11** ±0.14	**0.07** ±0.05	**0.29** ±0.42	1.00 ±0.06
	IS	**0.14** ±0.09	**0.31** ±0.10	**0.17** ±0.15	**0.63** ±0.41	**0.18** ±0.14

		Relocate human	Ant expert	Ant medium	Ant medium-expert	Ant medium-replay
Regret@1	Best DICE	0.97 ±0.11	**0.62** ±0.15	**0.43** ±0.10	**0.60** ±0.16	0.64 ±0.13
	VPM	0.77 ±0.18	**0.88** ±0.22	**0.40** ±0.21	**0.32** ±0.24	0.72 ±0.43
	Doubly Robust	**0.17** ±0.15	**0.43** ±0.22	**0.12** ±0.18	**0.37** ±0.13	**0.05** ±0.09
	FQE (L2)	**0.17** ±0.14	**0.43** ±0.22	**0.12** ±0.18	**0.36** ±0.14	**0.05** ±0.09
	IS	**0.63** ±0.41	**0.47** ±0.32	0.61 ±0.18	**0.46** ±0.18	**0.16** ±0.23

		Ant random	Hopper expert	Hopper medium	Hopper random	Walker2d expert
Regret@1	Best DICE	**0.50** ±0.29	0.20 ±0.08	**0.18** ±0.19	**0.30** ±0.15	**0.35** ±0.36
	VPM	**0.15** ±0.24	**0.13** ±0.10	**0.10** ±0.14	0.26 ±0.10	**0.09** ±0.19
	Doubly Robust	**0.28** ±0.15	**0.34** ±0.35	**0.32** ±0.32	**0.41** ±0.17	**0.06** ±0.07
	FQE (L2)	**0.28** ±0.15	0.41 ±0.20	**0.32** ±0.32	**0.36** ±0.22	**0.06** ±0.07
	IS	**0.56** ±0.22	**0.06** ±0.03	**0.38** ±0.28	**0.05** ±0.05	**0.43** ±0.26

		Walker2d medium	Walker2d medium-expert	Walker2d medium-replay	Walker2d random	Median
Regret@1	Best DICE	**0.27** ±0.43	0.78 ±0.27	**0.18** ±0.12	**0.39** ±0.33	0.38
	VPM	**0.08** ±0.06	**0.24** ±0.42	0.46 ±0.31	0.88 ±0.20	0.28
	Doubly Robust	**0.25** ±0.09	**0.30** ±0.12	0.68 ±0.23	**0.15** ±0.20	0.25
	FQE (L2)	0.31 ±0.10	**0.22** ±0.14	**0.24** ±0.20	**0.15** ±0.21	0.24
	IS	**0.70** ±0.39	**0.13** ±0.07	**0.02** ±0.05	**0.74** ±0.33	0.18

Table E.4: Regret for D4RL

Scatter Plots

Finally, we present scatter plots plotting the true returns of each policy against the estimated returns. Each point on the plot represents one evaluated policy.

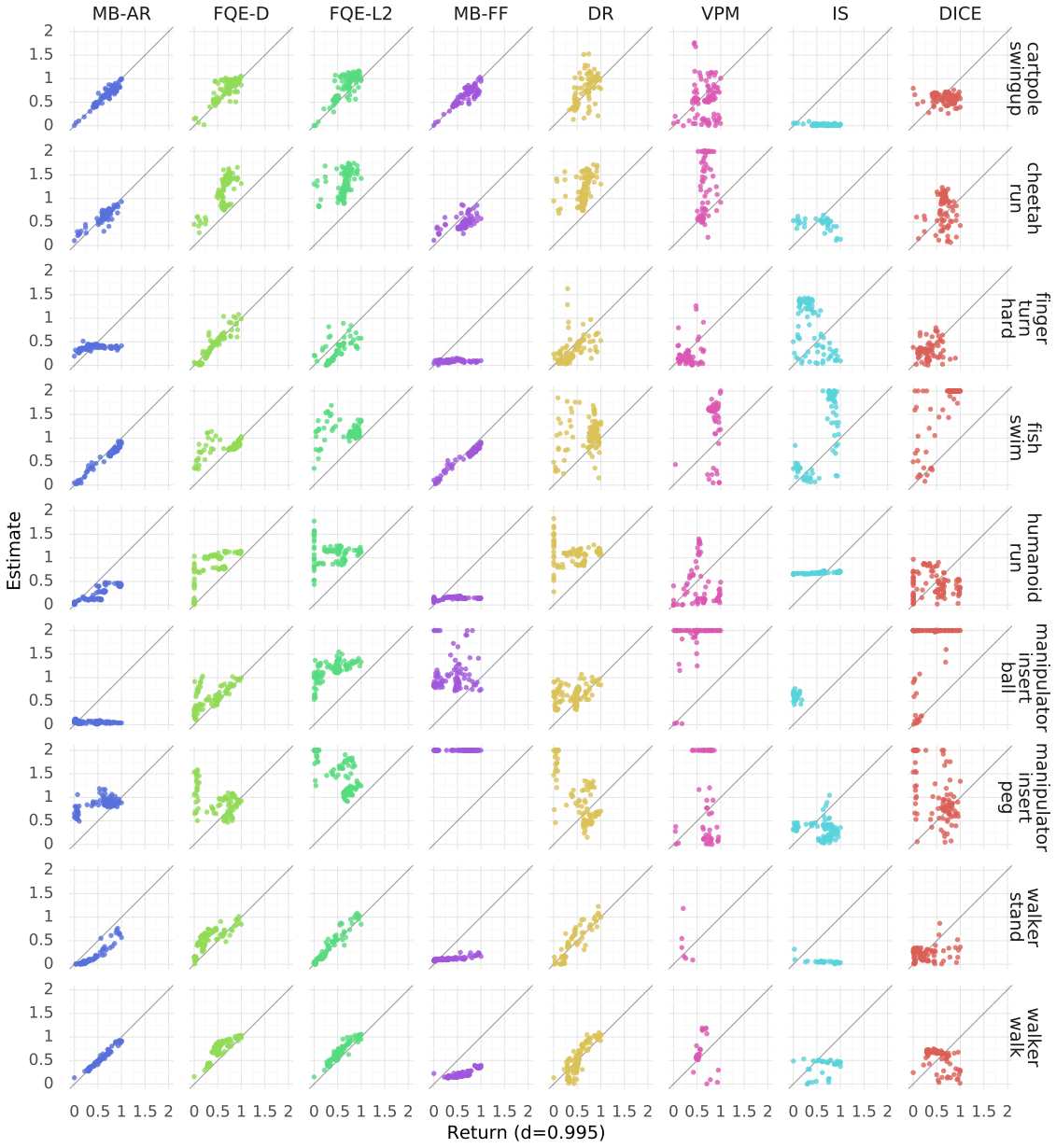


Figure E.8: Scatter plots of estimate vs ground truth return for each baseline on each task in DOPE RL Unplugged.

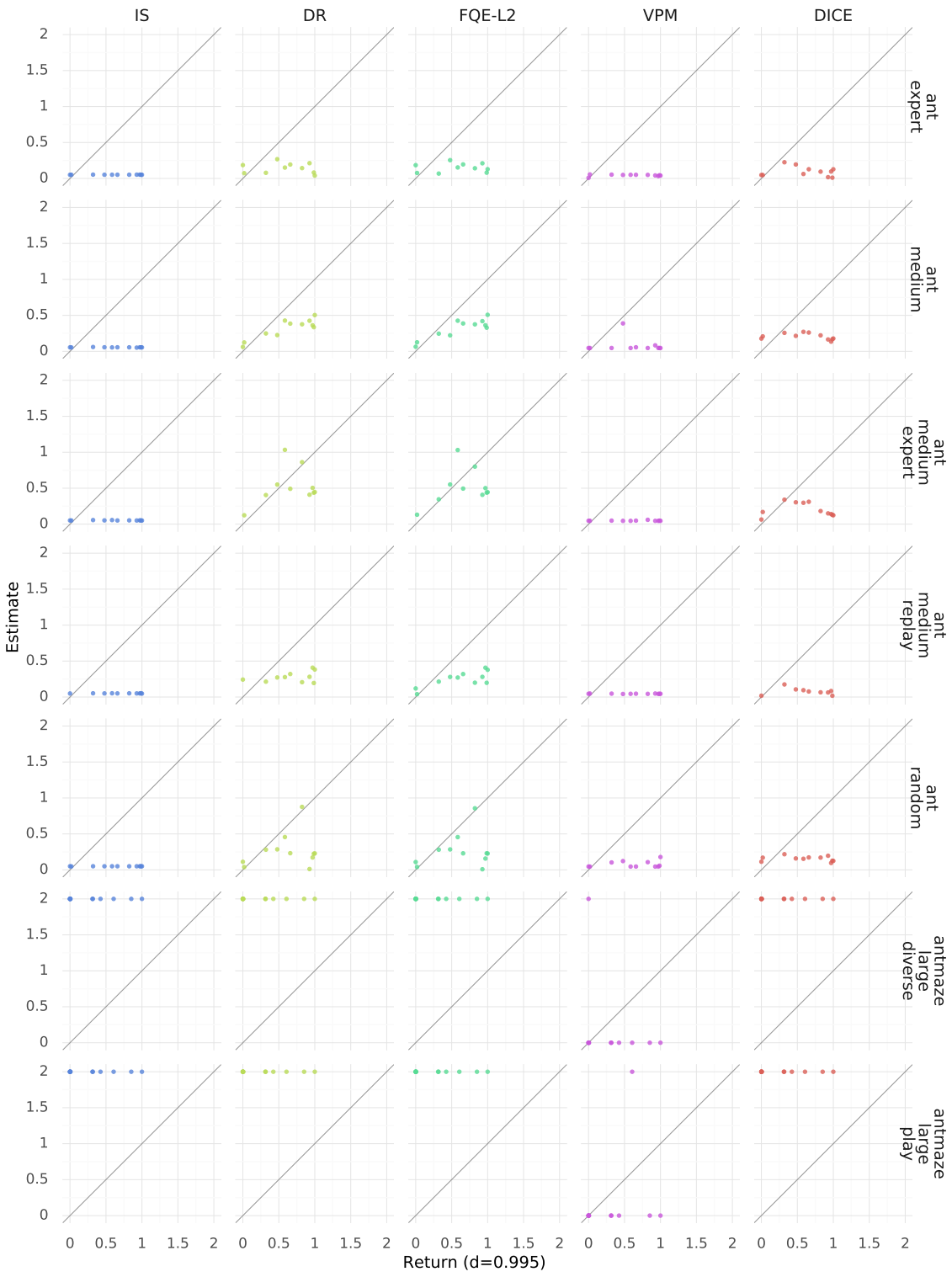


Figure E.9: Scatter plots of estimate vs ground truth return for each baseline on each task in DOPE D4RL (part 1).

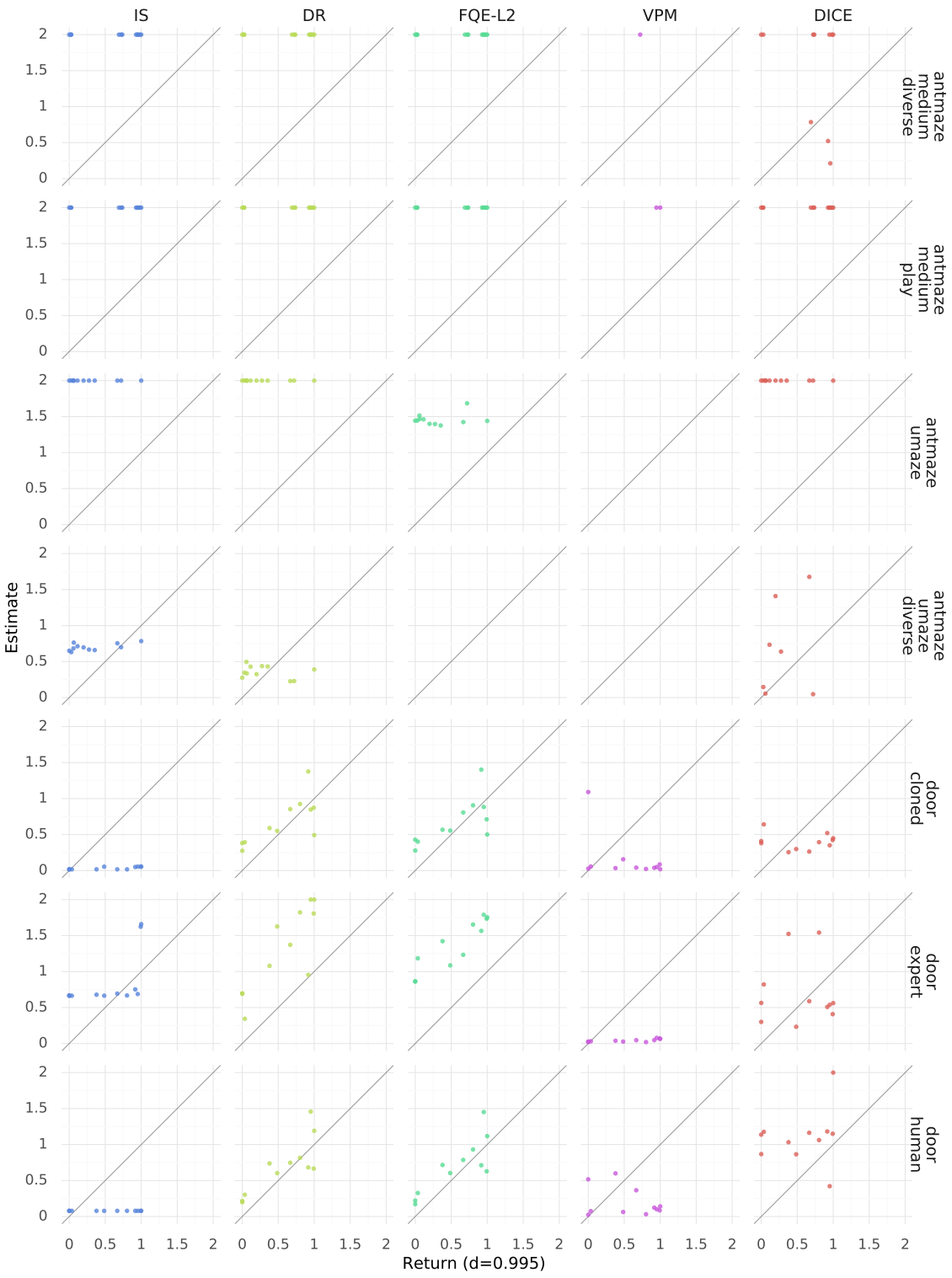


Figure E.10: Scatter plots of estimate vs ground truth return for each baseline on each task in DOPE D4RL (part 2).

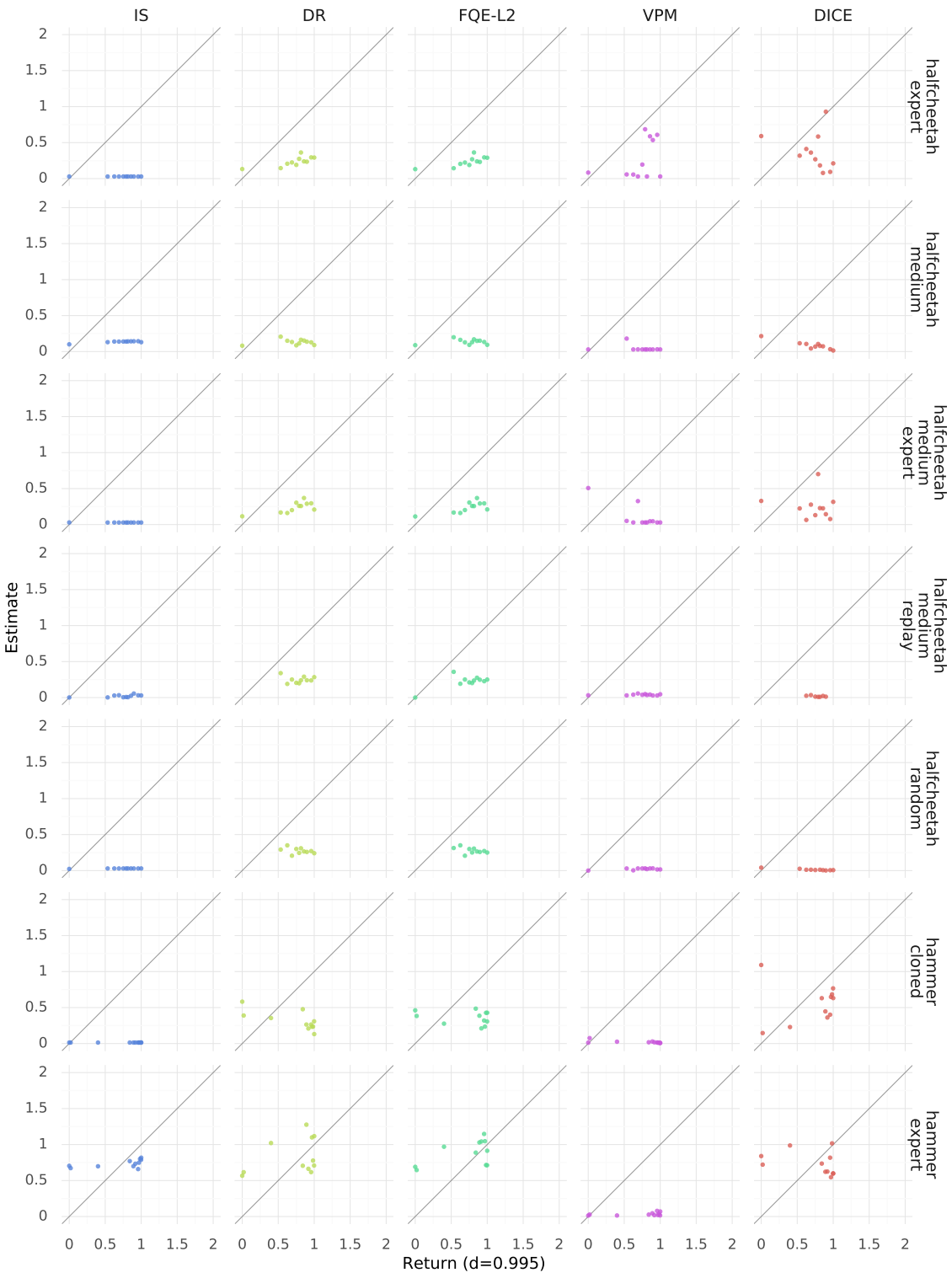


Figure E.11: Scatter plots of estimate vs ground truth return for each baseline on each task in DOPE D4RL (part 3).

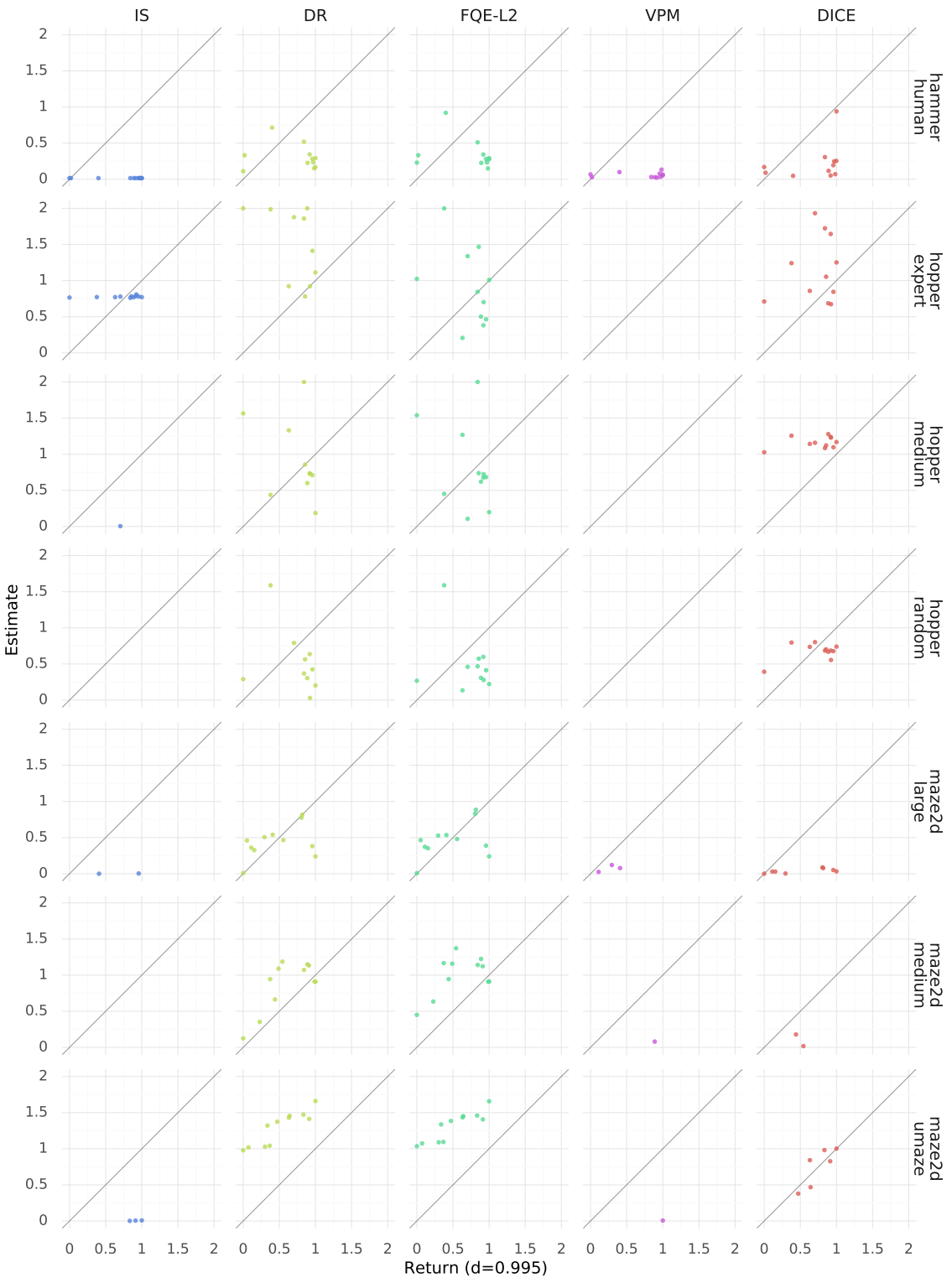


Figure E.12: Scatter plots of estimate vs ground truth return for each baseline on each task in DOPE D4RL (part 4).

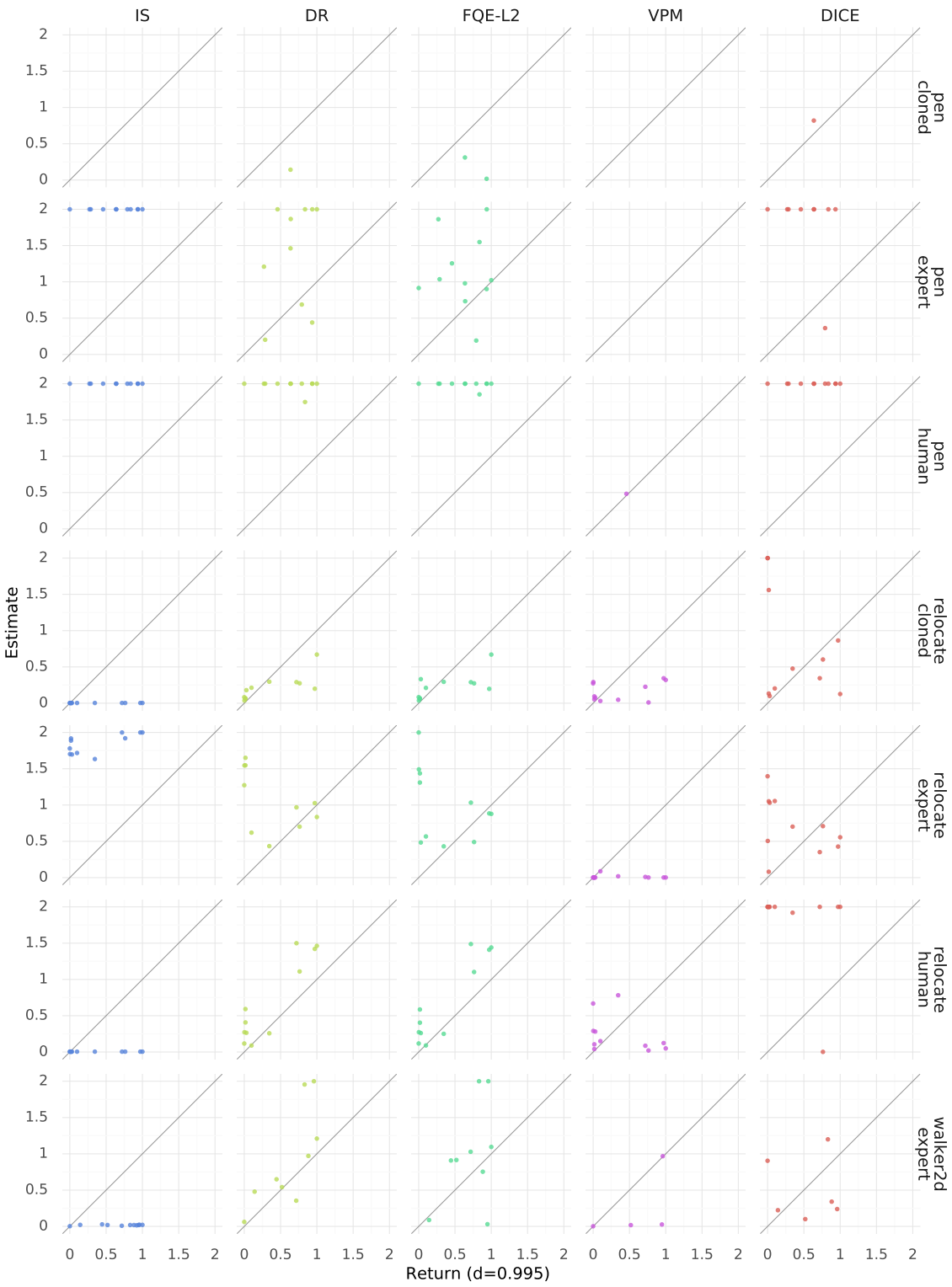


Figure E.13: Scatter plots of estimate vs ground truth return for each baseline on each task in DOPE D4RL (part 5).

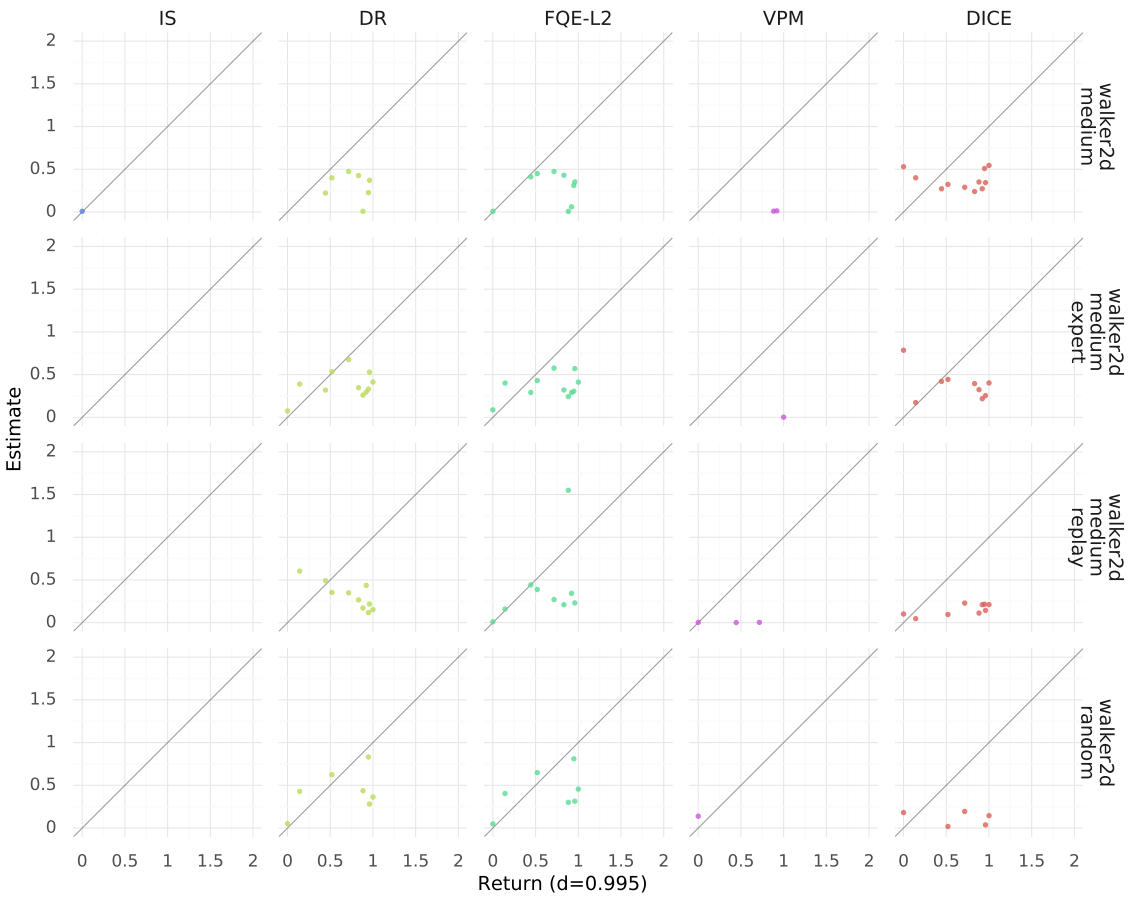


Figure E.14: Scatter plots of estimate vs ground truth return for each baseline on each task in DOPE D4RL (part 6).