remove showlabels
remove offset
set nocomments

# Techniques for Sample-Efficient Reinforcement Learning

by

William Fairclough Whitney

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

September, 2021

---

Professor Kyunghyun Cho

To my dog Weierstraß, with affection.

add dedication

# Acknowledgements


- all my collaborators
- housemates

# Abstract

# Contents

# List of Figures

# List of Tables

# List of Appendices

# 1 Introduction

{sec:introduction}

Reinforcement learning (RL) provides a framework for systems which learn to make decisions under uncertainty about their environment. Such a system must simultaneously determine which of all possible environments it is interacting with and solve for an optimal strategy in that environment. This uncertainty is the core challenge of reinforcement learning, and one of the most fundamental questions of the field is how much experience a learning system requires to resolve its uncertainty and perform well. We call this the sample complexity of reinforcement learning.

Classically reinforcement learning was restricted to environments with small, countable state and action spaces.[1] In this setting a variety of algorithms were developed with robust guarantees on their performance relative to the ideal policy and on the amount of data required to approach perfect behavior. Bandit algorithms for environments without temporal dependence and dynamic programming for those with non-trivial dynamics both yielded practical success. However, the specter of exponentially-increasing sample complexity limited these approaches to low-dimensional settings.

The combination of deep learning with reinforcement learning in the last decade has given rise to the new subfield of "deep reinforcement learning", which leverages function approximation to scale reinforcement learning to tasks with large or uncountable state or action spaces. Deep reinforcement learning has led to dramatic results across a range of domains, from board games like Go to complex multiplayer computer games like StarCraft, from controlling automated balloons to manipulating Rubik's cubes, and even to abstract tasks like chip design.

A key unifying feature of these most impressive deep RL results is the availability of simulators. Each of these domains admits the construction of a simulation of sufficient fidelity that a policy may be trained in simulation and then deployed on the real task with little to no fine tuning. Furthermore, these simulations are inexpensive relative to collecting "real" data, reducing the cost of training a policy by orders of magnitude. By turning data collection into computation, simulation freed deep RL from paying attention to sample efficiency, as it was more important how computationally fast an algorithm was than how many hours or years of (virtual) experience it consumed.[2]

## 1.1 The Cost of Free Data

Our reliance on simulation comes with hidden costs. Simulation-based RL has generated spectacular results, and simulation should be a primary tool in any effort to solve a real-world task. However, the availability of cheap simulation has shaped what the deep RL community studies by reinforcing work on simulatable domains and the large-data regime. This has resulted in slower progress on questions in the sample-limited regime. Sample-efficient reinforcement learning is important due to the fundamental scientific importance of understanding sample complexity and the intractibility of simulating every domain of interest.

Sample complexity is foundational. The study of sample complexity in reinforcement learning addresses fundamental questions about what information is needed to reliably solve a problem. The observation that realistic, high-dimensional tasks are solvable with small sample sizes may be surprising, given that there are theoretical lower bounds requiring a number of samples which is linear in the number of states or exponential in the horizon [Du et al. 2020]. This disagreement requires explanation, and it suggests that real-world problems contain a significant amount of structure which makes them easier to solve. Understanding this structure in more

detail, as well as how it interacts with the inductive biases of the function approximators we use, could lead to significant breakthroughs.

Simulators are expensive. While many simulators already exist that are computationally inexpensive, many important systems of interest are computationally difficult to simulate with sufficient precision for transfer to the real world. Simulations of fluid dynamics, deformable materials, and large numbers of contacts (to name a few) can be significantly slower than real-time. Furthermore, the development of new realistic simulators is *financially* expensive not only because of the engineering of the simulator itself, but additionally due to the need to create and maintain a close correspondence between the simulation and the physical system of interest. These costs inhibit the use of simulators for domains which are too complex or too niche.

This thesis consists of work done in the last several years which makes deep RL somewhat more capable in the sample-limited regime. With this line of work I hope to unlock the wide range of environments which currently lack high-fidelity simulators. Beyond simply enabling RL in more complex environments, improvements to sample efficiency has the potential to allow agents to adapt on the fly to the specifics of the environment or objectives they interact with. A home robot might come to know the best way to pick up *your* cups, or an automated factory could produce more rapidly over the course of a production run as the networked assembly arms pool experience about manipulating each component part. In these settings learning more efficiently can be directly translated into consumer experience and dollars saved.

## 1.2 Elements of Sample-Efficient Learning

Reinforcement learning can be thought of as comprising two distinct processes: data collection and policy optimization. Data collection, also known as *exploration*, is when the agent interacts with the environment in order to gain more information about the optimal policy. Policy opti-

mization is the process of using data collected from the environment to produce a policy that achieves as much reward as possible. In order that an RL algorithm overall be sample efficient, both exploration and policy optimization must themselves be efficient.

Efficient exploration means rapidly acquiring information about the optimal policy. This means collecting data that is diverse, such that valuable states and actions will be quickly uncovered, but also biasing data collection towards states and actions which are more likely to be optimal. If done well exploration spans the environment and then collects evidence to allow the agent to discard poor policies and differentiate between actions which are optimal and those which are merely good.

Efficient policy optimization means squeezing as much performance as possible out of the data which is currently available. While this is often discussed in the narrow frame of model-free RL, this process might include building models, learning representations, or even meta-learning. In principle one might hope to feed some prior beliefs along with whatever data has been collected from the environment and get back the best policy which is supported by that data. Recent work in off-policy RL and the offline RL setting have made progress towards this vision, but it remains some way off.

## 1.3 Overview

This thesis consists of work on improving the sample efficiency of reinforcement learning through three main directions: (1) collecting more diverse data without confounding policy learning; (2) learning representations which capture structure in the environment; and (3) studying policy optimization in the batch setting to develop policy improvement operators that are robust to limited data. The motivating challenge through much of this work is robotic manipulation using low-dimensional positions or high-dimensional images as observations and with continuous-valued action spaces. However, the findings described here are applicable more widely across reinforce-

ment learning.

The rest of the thesis is organized as follows:

- Chapter 2 introduces background on the problem of sample-efficient RL and how it relates to the several settings under which RL has been studied.
- Part I describes the role of exploration in sample-efficient RL, with Chapter 3 illustrating how exploration techniques adapted from deep RL to the sample-limited robotic setting can improve sample efficiency and performance.
- Part II discusses the role of representation in reinforcement learning, with connections to representation learning more generally in Chapter 4 and work on representations for sample-efficient RL in Chapter 5.
- Part III studies policy optimization in the offline setting, first describing the impact of over-paraterized models in offline bandit problems in Chapter 6, then studying the repeated application of policy improvement operators in the full offline RL problem in Chapter 7.

# Notes

1. Or more accurately, environments admitting assumptions that allow them to be treated as such. A 2-D continuous state space can be treated as a discrete grid of states with arbitrarily little waste under the assumption that the environment changes sufficiently slowly, for example.

2. Of course, not everyone in the deep RL community ignores sample efficiency. Notably many people working on robotics have maintained remarkable discipline in only running simulated benchmarks for physically plausible amounts of time, but there is also a small but vibrant community working on reaching human Atari performance with human-like amounts of experience.

# 2 | The Many Settings of Reinforcement Learning

{sec:rl-settings}

## 2.1 Notation

A Markov decision process (MDP) $\mathcal{M}$ consists of a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma, s_0)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $P$ is the transition function mapping $\mathcal{S} \times \mathcal{A}$ to distributions on $\mathcal{S}$, $R$ is the scalar reward function on $\mathcal{S} \times \mathcal{A}$, $\gamma$ is the discount factor, and $s_0$ is the starting state. Note that a single start state can be converted to a start distribution by letting $P(s_0, \cdot)$ be independent of the action taken. An *agent* interacts with an MDP by producing a policy $\pi$ at each timestep and observing the transitions it visits.

The *value* of a state $s$ for a policy $\pi$ is the (discounted) sum of future rewards obtained by running that policy starting from the state $s$:

$$V^\pi(s) = \mathop{\mathbb{E}}_{\substack{a_t \sim \pi(\cdot|s_t) \\ s_{t+1} \sim P(s_t, a_t)}} \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \,\Big|\, s_0 = s \tag{2.1}$$

Similarly the value of a state-action pair $(s, a)$ is written as

$$Q^\pi(s, a) = \mathop{\mathbb{E}}_{\substack{a_t \sim \pi(\cdot|s_t) \\ s_{t+1} \sim P(s_t, a_t)}} \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \,\Big|\, s_0 = s, a_0 = a \tag{2.2}$$

Any MDP admits a deterministic optimal policy $\pi^*$ with corresponding value functions $V^*$ and $Q^*$ such that $V^*(s) \geqslant V^\pi(s)$ for all $s$ and $\pi$ [Sutton and Barto 2018].

## 2.2 Discounting and Resets

When $\gamma < 1$ we say that the setting is *discounted*, and for $\gamma = 1$ we say that it is *undiscounted*. An environment may have a time limit $T$ such that after every $T$ steps, the state is reset to $s_0$. In this case we call it *episodic*. To satisfy the Markov property, episodic environments should include the current timestep $t$ in the state.

For episodic environments, it is natural to define a policy's quality in terms of the total reward earned (on average) in a single episode, with discounted environments preferring rewards obtained earlier in the episode. For non-episodic environments comparisons between policies are less clear-cut; see Section 2 of Strehl and Littman [2008] for a discussion.

Common practice in deep reinforcement learning is to train agents with discounting and resets, but not include the timestep in the state observations and simply ignore the transition from $s_T$ to $s_0$. Policies are typically evaluated by the total undiscounted reward in an episode, despite the conflict with the training setup. In most cases the rewards are truncated to reflect the limited episode duration [Schulman et al. 2017; Fujimoto et al. 2018a; Haarnoja et al. 2018b]. Since the agent is unable to tell when an episode will end, this effectively introduces noise into value prediction targets, and this noise varies by state depending on how often the agent has ended an episode on that state. In other cases value targets may be bootstrapped from the state $s_T$ as if the environment were not episodic. This has two issues: (1) it introduces bias by treating an estimate of $V(s_T)$ as the true value, when in some states and environments this estimate may never be updated; and (2) by pretending the environment has no resets, it introduces a mismatch between the training and test objectives.[3] However, it does not introduce noise.

More fundamentally, practically all discounted policy gradient algorithms drop the discount-

ing term from the state distribution. Nota and Thomas [2020] show that this results in following a direction which is not the gradient of any function, and which is not guaranteed to converge to a good solution with respect to the discounted or undiscounted objectives. While this is deeply worrying, these methods frequently work well in practice. This may be due to their usage with overparameterized neural network models, which are largely invariant to a reweighting of the data [Byrd and Lipton 2018; Brandfonbrener et al. 2021].

## 2.3 Defining Sample Complexity

We say that a policy $\pi$ is $\varepsilon$-optimal if $V^*(s_0) \leqslant V^{\pi}(s_0) + \varepsilon$. Define $\boldsymbol{\pi} = A(\mathcal{M}, i)$ to be the policy obtained by running the agent (i.e. algorithm) $A$ in the environment $\mathcal{M}$ for $i$ timesteps, being careful to note that the policy $\boldsymbol{\pi}$ is itself a random variable due to the randomness in the experience collected in those $i$ steps. Let the *sample complexity* of learning a $\varepsilon$-optimal policy on $\mathcal{M}$ with $A$ be the expected number of steps (indexed as $i$) such that

$$V^{\pi_i}(s_0) < V^*(s_0) \leqslant -\varepsilon, \qquad \text{where } \pi_i = A(\mathcal{M}, i). \tag{2.3}$$

This definition is related to those proposed by Fiechter [1994] and Strehl and Littman [2008] for PAC learning. Sometimes it is also useful to consider the "anytime" performance of an algorithm $A$ on an MDP $\mathcal{M}$. An algorithm $A_1$ would dominate $A_2$ if $\forall i$, $V^{A_1(\mathcal{M},i)}(s_0) \geqslant V^{A_2(\mathcal{M},i)}(s_0)$.

## 2.4 Online and Deployment Settings

{sec:regret-deploymen

While the overall MDP framework is largely shared in the community, several different objectives for a learning agent are commonly studied. The online and offline settings are perhaps the most studied in the theory community, but the "learn-and-deploy" setting has the most relevance for present applications of RL.

THE ONLINE SETTING. Here an RL agent learns by continually interacting with the environment with the goal of maximizing the total reward earned across all time. This gives rise to the explore-exploit tradeoff when acting: at each moment, the agent may choose to take an action which is uninformative but leads to greater short-term reward, or one which will yield more information at the cost of lower reward. The objective for this setting is to minimize the rate of accumulation of *regret*, which measures the difference between the total reward obtained by an optimal policy and the agent:

talk about sample complexity

$$L(A, \mathcal{M}, T) = \mathbb{E}\left[\sum_{i=1}^{T} R(s_i^*, a_i^*) - R(s_i, a_i)\right]. \tag{2.4}$$

This setting is appropriate when an agent is being trained "on the job," where mistakes early in learning have just as much deleterious effect as those made later.

talk about sample complexity

THE LEARN-AND-DEPLOY SETTING. This setting consists of distinct learning and deployment phases. In the learning phase, the agent is not required to perform well and may collect whatever data is most informative. In the deployment phase, the policy is fixed and should be as close to optimal as possible. Note that the policy produced at the end of the training procedure need bear no resemblance to those used to collect data. For episodic environments, with a training period consisting of $N$ steps we can write the this objective as

$$L(A, \mathcal{M}, N) = V^*(s_0) - V^{\pi_N}(s_0), \qquad \text{where } \pi_N = A(\mathcal{M}, N) \tag{2.5}$$

Historically this setting has not been much discussed, though it is analagous to the task of best-arm identification in bandits [Russo 2016; Kaufmann et al. 2016]. It is also related to the iterative technique of fitted Q iteration [Ernst et al. 2005; Riedmiller 2005].

Crucially, this setting is the one used in practice in nearly every application of reinforcement learning. RL algorithms are not yet safe and reliable enough to allow them to update a policy

on the fly, especially with a physical system which may be damaged or cause injury. Furthermore, most works studying RL implicitly provide results in this setting by evaluating according to a different policy than the one used for training, for example showing learning curves with a deterministic policy [Mnih et al. 2015b; Lillicrap et al. 2016; Fujimoto et al. 2018a; Haarnoja et al. 2018b]. Comparisons of final, large-data performance similarly reflect this setting [Silver et al. 2016; Vinyals et al. 2019; OpenAI et al. 2019a,b].

talk about sample complexity

The offline setting. Also known as the *batch* setting, this consists only of pure policy optimization given a fixed dataset of environment interactions collected by an extrinsic behavior policy. After learning from this data in whatever way it sees fit, an algorithm produces a fixed policy with the objective of earning as much reward as possible. Though described as a reinforcement learning setting, it does not include any actual reinforcement as the agent never learns from its own interactions with the environment. However, this makes the offline RL setting uniquely valuable for isolating how much can be learned from particular data. This setting is also appealing as it would in principle allow an agent to be trained in a risk-free way by using data collected from a safe policy, and for free (in terms of samples) if data from one experiment can be repurposed as training data for another.[4]

Do I want a section on simulated versus physical envs?

# Notes

3. This mismatch is decreased if the divergence between the distributions $P(s_T, \pi(\cdot \mid s_T))$ and $s_0$ is smaller. Like most problems in RL, it also becomes smaller if smaller discount factors are used. However in general there are actions which would provide more short-term reward, and thus perform better toward the end of an episode, than the infinite-horizon optimal actions.

4. While there are doubtless some settings where this cross-task data reuse is possible, it has quite clear limits. For a policy to be trained to solve task B using data collected from task A, the policy used for task A would have had to actually also solve task B. It could have been done piecewise rather than in a single good trajectory, but unless the tasks are extremely similar it is vanishingly unlikely. Perhaps a more practical application would be to start with a safe but poor policy for solving a task, then incrementally collect new data, refine the policy using offline RL, validate that the new policy is also safe, and then collect data once more.

# Part I

# Exploration and Sample Efficiency

{sec:exploration}

Introducing exploration for sample-efficient control.

# 3 Decoupled Exploration and Exploitation Policies

{sec:deep}

## 3.1 Introduction

Recent progress in reinforcement learning (RL) for continuous control has led to significant improvements in sample complexity and performance. While earlier on-policy algorithms required hundreds of millions of environment steps to learn, recent off-policy algorithms have brought the sample complexity of model-free RL in range of solving tasks on real robots [Haarnoja et al. 2018c].

In parallel, a rich literature has been developed for directed exploration in deep reinforcement learning, inspired in part by the theoretical impact of exploration on sample complexity. The bulk of these methods fall into the family of bonus-based exploration (BBE) methods, in which a policy receives a bonus for visiting states deemed to be interesting or novel. BBE algorithms have enabled RL to solve a variety of long-horizon, sparse-reward tasks, most notably the game Montezuma's Revenge from the Arcade Learning Environment (ALE) [Bellemare et al. 2015].

These two subfields both aim to minimize the sample complexity of model-free RL, and their methods are in principle perfectly complementary. Off-policy algorithms extract improved policies from data collected by (notionally) arbitrary behavior, and their performance is limited only by the coverage of the data; meanwhile exploration generates data with improved coverage. How-

ever, to date the impact of directed exploration techniques on sample-efficient control has been minimal, with state of the art algorithms using undirected exploration such as maximum-entropy objectives. In this paper we investigate the missing synergy between off-policy continuous control and directed exploration.

We find that BBE is poorly suited to the few-sample regime due to slowly-decaying bias in the learned policy and slow adaptation to the non-stationary exploration bonus. Bias due to optimizing a reward function other than the task reward leads a policy trained with BBE to exhibit poor performance until the bonus decays toward zero. Meanwhile, the non-stationary (continually decreasing) exploration bonus cannot necessarily be optimized by a fixed policy, violating one of the core assumptions of RL. This leads to slow exploration as the policy adapts only gradually, especially in the off-policy case where replay buffers will contain stale rewards. These observations underline research by Taiga et al. [2020] showing that across the ALE, no BBE algorithm outperforms undirected $\varepsilon$-greedy exploration.

We demonstrate that bias and slow coverage are the culprits of BBE's lackluster performance by proposing a new exploration algorithm, Decoupled Exploration and Exploitation Policies (DEEP), which addresses these limitations. DEEP decouples the learning of a *task policy*, which is trained to maximize the true task reward, and an *exploration policy*, which maximizes only the exploration bonus. Both policies are trained off-policy using data collected according to the product of the two policy distributions. Unlike the policy learned by BBE, DEEP's task policy is always unbiased in the sense that it reflects the current belief about the optimal action in each state. Furthermore, this decoupling allows DEEP to aggressively update the exploration policy without affecting the convergence of the task policy, thereby adapting more rapidly to the changing exploration bonus.

We perform experiments using policies based on Q-learning [Sutton and Barto 2018; Mnih et al. 2015a] on toy tasks and soft actor-critic (SAC) [Haarnoja et al. 2018c] on larger-scale tasks from the DeepMind Control Suite [Tassa et al. 2018]. Our results show that on tasks with dense

rewards and uniform resets, BBE often performs worse than the underlying policy-learning algorithm while DEEP incurs no cost for exploring. On tasks with more natural resets and sparse rewards, DEEP covers the state space more rapidly than BBE and reaches peak performance in a fraction of the samples required with undirected exploration. In total, DEEP strictly outperforms undirected exploration while solving many sparse environments just as fast as dense ones.

## 3.2 Background

### 3.2.1 Notation

A Markov decision process (MDP) $\mathcal{M}$ consists of a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $P$ is the transition function mapping $\mathcal{S} \times \mathcal{A}$ to distributions on $\mathcal{S}$, $R$ is the scalar reward function on $\mathcal{S} \times \mathcal{A}$, and $\gamma$ is the discount factor. We use lower-case $(s, a, r)$ to refer to concrete realizations of states, actions, and rewards. We use $\mathcal{M}_f$ to denote the MDP $\mathcal{M}$ with the original reward function $R$ replaced by another function $f$. For convenience we assume exploration rewards are within $[0, 1]$, and we define $\bar{r} = 1/1-\gamma$, which is the maximum discounted value possible.

### 3.2.2 Bonus-based exploration: a recipe for exploration in deep RL

{sec:intrinsic_reward

Bonus-based exploration has emerged as the standard framework for exploration in the deep reinforcement learning community. In this framework, an agent learns in a sequence of MDPs $\widetilde{\mathcal{M}} = \{\mathcal{M}_{\widetilde{R}_n}\}_{n=1}^{N}$ where the reward function $\widetilde{R}_n$ changes as a function of each transition. A typical choice is $\widetilde{R}_n = R + R_n^+$, where $R_n^+$ is an exploration bonus which measures the "novelty" of a transition $(s, a, s')$ given the history of all transitions up to $n$. After taking each transition $(s, a, s')$, the reward $\widetilde{r} = \widetilde{R}_n(s, a, s')$ is calculated and the tuple $(s, a, s', \widetilde{r})$ is added to a replay dataset $D$. The agent optimizes its reward in this (non-stationary) MDP $\widetilde{\mathcal{M}}$ via some model-free RL algorithm

operating on the replay dataset. The realization of a particular algorithm in this family amounts to defining a novelty function and picking a model-free RL algorithm [Stadie et al. 2015; Houthooft et al. 2016; Bellemare et al. 2016; Pathak et al. 2017; Tang et al. 2017; Burda et al. 2018; Machado et al. 2020]. We illustrate this recipe in Algorithm 1.

Pseudo-counts. Building upon theoretically-motivated exploration methods for discrete environments [Strehl and Littman 2008], Bellemare et al. [2016] proposed to give exploration bonuses based on a *pseudo-count* function $\hat{N}$. A pseudo-count has two key properties. Like a true count, a pseudo-count increases by 1 each time a state (or state-action pair) is visited. Unlike a true count, a pseudo-count generalizes across states and actions; that is, when a state $s$ is visited, the pseudo-count for nearby states $s + \varepsilon$ may increase as well.

## 3.3 Limitations of bonus-based exploration

The bonus-based exploration algorithm, illustrated in Algorithm 1, has two weaknesses which limit its usefulness for sample-efficient policy learning.

Bias with finite samples. Because they estimate the optimal policy on the modified MDP $\mathcal{M}'$, bonus-based exploration algorithms learn biased policies as long as the exploration bonus is nonzero. According to theory, the exploration bonus should be scaled larger than is done in practice [Strehl and Littman 2008] and decay slower than $1/N(s)$ [Kolter and Ng 2009] in order to guarantee convergence to the optimal policy. This behavior, shown in Figure 3.2, can result in slow convergence to the optimal policy and substantially biased policies after a practically feasible number of samples.

Slow adaptation to changing rewards. Algorithms in this family update the policy according to the schedule of the underlying model-free RL algorithm – for example at the end of each

**Algorithm 1** Bonus-based exploration {alg:bbe}

**Require:** replay dataset $D$, policy $\pi$, bonus $R_n^+$

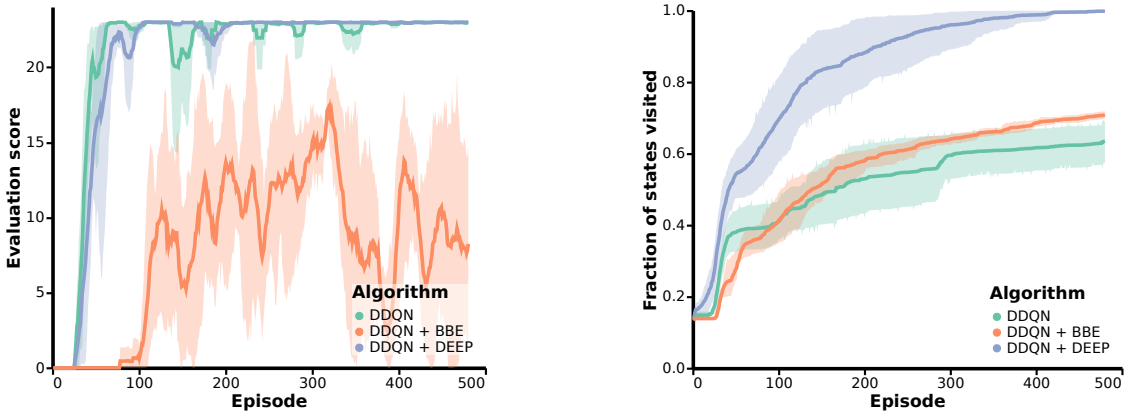
$n \leftarrow 0$
**repeat**
  **for** one episode **do**
    Collect $(s, a, s', r) \sim P(s, \pi(s))$
    $\widetilde{r} \leftarrow r + R_n^+(s, a, s')$
    $D \leftarrow D \cup (s, a, s', \widetilde{r})$
    $R_{n+1}^+ \leftarrow \text{Update}(R_n^+, (s, a, s'))$
    $n \leftarrow n + 1$
  Train $\pi$ with samples from $D$
**until** $n = N$

**Algorithm 2** Decoupled Exploration and Exploitation Policies {alg:deep}

**Require:** replay dataset $D$, temperature $\tau$, task policy $\pi_{\text{task}}$, exploration policy $\pi_{\text{explore}}$, bonus $R_n^+$

$n \leftarrow 0$
**repeat**
  **for** one episode **do**
    Update $\pi_{\text{explore}}$ on $\mathcal{M}_{R_n^+}$
    Set $\beta(a|s) \propto \pi_{\text{task}}(a|s) \cdot \pi_{\text{explore}}(a|s)$
    Collect $(s, a, s', r) \sim P(s, \beta(s))$
    $D \leftarrow D \cup (s, a, s', r)$
    $R_{n+1}^+ \leftarrow \text{Update}(R_n^+, (s, a, s'))$
    $n \leftarrow n + 1$
  Train $\pi_{\text{task}}$ with samples from $D$
**until** $n = N$

**Figure 3.1:** Comparison of classic bonus-based exploration (BBE) with our method (DEEP). BBE computes exploration bonuses at the time of visiting a transition, adds them to the real rewards, and uses a replay buffer of experience to learn a policy. DEEP separates the exploration policy $\pi_{\text{explore}}$ from the task policy $\pi_{\text{task}}$, allowing $\pi_{\text{task}}$ to be an unbiased estimate of the optimal policy throughout training. It always uses the *current* exploration reward function $R_n^+$ when updating the exploration value function, and is fast-adapting to deal with the non-stationary bonus MDP.




**(a)** Reward **(b)** Coverage

**Figure 3.2:** Experiments in a $40 \times 40$ grid-world environment with one goal state, where learning algorithms were warm-started with 20 episodes of data from a skilled policy. **(a)** With enough signal to find the goal, DDQN (Double DQN, Hasselt et al. [2016]) alone rapidly converges to the optimal policy. BBE introduces bias, causing the policy to continually explore. Our method, DEEP, learns the task policy just as rapidly as DDQN alone. **(b)** Though it performs well, DDQN simply goes to the goal during each train episode and does not explore other options. BBE continues to seek out new states at a slow but steady rate. DEEP explores far more than BBE during data collection despite simultaneously performing just as well as DDQN at evaluation time.

{fig:gridworld_warmst

episode. This works well for the stationary MDPs that these algorithms were developed for, but the modified MDP $\mathcal{M}'$ which represents the exploration problem is non-stationary. This leads to an agent which determines the most novel state and then stays there for an entire episode. This degenerate behavior leads to potentially exploring only a single state per episode instead of visiting a sequence of new states as the reward function evolves.[5] The use of replay buffers compounds this effect, since algorithms in this family compute exploration rewards at the time the transition is collected, rather than when it is used. An algorithm which is unaware of the non-stationary nature of the MDP will maximize the return on this mixture of reward functions rather than the reward that incorporates the current bonus, resulting in slow coverage of the environment. Figure 3.3 shows uniform random actions, BBE, and BBE with the fast adaptation scheme we propose in Section 3.4.3 all exploring in a $40 \times 40$ grid-world without rewards. While BBE covers the state space much faster than undirected exploration, it is unnecessarily slow. See Appendix 3.A for visualizations.

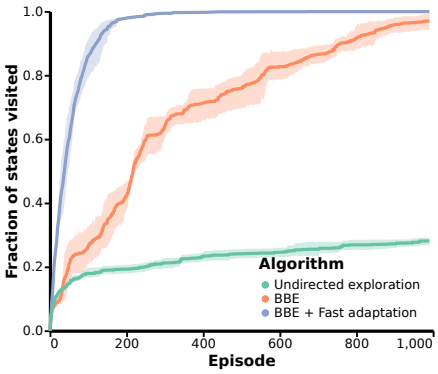



**Figure 3.3:** Pure exploration

{fig:gridworld_visits

## 3.4 Decoupled exploration for sample-efficient control

In this section, we describe a new algorithm called Decoupled Exploration and Exploitation Policies (DEEP) which addresses the limitations of BBE. The core insight is that by leveraging off-policy RL algorithms, DEEP can learn two policies from the same replay: a task policy $\pi_{\text{task}}$, which maximizes the reward on the original MDP $\mathcal{M}_R$, and an exploration policy $\pi_{\text{explore}}$, which maximizes only the reward on the bonus MDP $\mathcal{M}_{R_n^+}$. This decoupling serves two purposes. First, it enables good performance even before exploration is complete by using $\pi_{\text{task}}$ at test time. Second, it allows $\pi_{\text{explore}}$ to be updated aggressively in order to more closely match the non-stationary bonus MDP; unlike $\pi_{\text{task}}$, it is not important that $\pi_{\text{explore}}$ converge exactly to an optimal policy.

Like BBE, DEEP is a family of algorithms related by their structure; a particular algorithm in this family consists of a choice of an exploration reward function and an off-policy RL algorithm for learning each policy. Throughout this work, we use a pseudo-count based exploration reward. For discrete tasks we use Double DQN (DDQN, Hasselt et al. [2016]) and Boltzmann policies. For tasks with continuous actions we use soft actor-critic (SAC) for $\pi_{\text{task}}$ and a DDQN policy for $\pi_{\text{explore}}$.

### 3.4.1 Pseudo-count estimation

{sec:kernel_counts}

Following Bellemare et al. [2016], we use an exploration reward derived from pseudo-counts. Instead of the high-dimensional pixel observations of the ALE, Control Suite has low-dimensional (<100-D) observations corresponding to joint and object locations and velocities. This lower dimensionality renders extracting pseudo-counts from a density estimator unnecessary, and in our experiments we specify a pseudo-count function based on kernels. For a real-valued state-action pair $x = [s, a]$ and a set of previous observations $\{x_i\}_{i=1}^n$, define the pseudo-count of $x$ and the exploration reward as

$$\hat{N}_n(x) = \sum_{i=1}^{n} k(x_i, x) \qquad\qquad R_n^+(s,a) = \hat{N}_n\big([s,a]\big)^{-1/2} \tag{3.1}$$

where $k$ is a kernel function scaled to have a global maximum $k(x, x) = 1$. This satisfies the key requirement for a pseudo-count function, namely that a visit to a state $x$ increases $\hat{N}(x)$ by 1 and $\hat{N}(x')$ by at most 1 for any other state $x'$. In our experiments we use a Gaussian kernel (scaled to have a maximum value of 1) with diagonal covariance. For implementation details see Appendix 3.B.

### 3.4.2 Separating task and exploration policies

DEEP uses two separate policies. Each is trained off-policy using transitions sampled from the replay buffer; $\pi_{\text{task}}$ is updated using the rewards from $R$ logged in the replay, while $\pi_{\text{explore}}$ is updated using rewards from $R_n^+$. Since $\pi_{\text{task}}$ is trained only on the rewards for the true task, it is unbiased in the sense that it reflects the current best estimate of the optimal policy. This stands in contrast to BBE policies, which optimize the sum of task and exploration rewards and thus represent a biased estimate of the optimal task policy until the exploration rewards go to zero. Our method is agnostic to the choice of algorithm and policy parameterization. However, it will

be most effective with policy learning algorithms that work well when trained far off-policy and produce high-entropy policies (e.g. policies which cover all optimal actions). For these reasons we use the state-of-the-art maximum-entropy algorithm SAC to learn $\pi_{\text{task}}$ in environments with continuous actions. For experiments with discrete action spaces we forego explicitly learning the task policy $\pi_{\text{task}}$; instead we learn the task Q-function via DDQN [Hasselt et al. 2016] and define the task policy as $\pi_{\text{task}}(a|s;Q) \propto \exp(Q(s,a)/\tau)$, where $\tau > 0$ becomes a hyperparameter – we refer to the supplementary material for details.

### 3.4.3 Fast-adapting exploration policy

{sec:fast_adaptation}

The non-stationary nature of the exploration reward function poses a challenge to typical model-free RL algorithms, which assume a fixed reward function. BBE methods update a single policy using a replay buffer which, at a step $n$, contains rewards from a mixture of bonus reward functions $\{R_1^+, \ldots, R_n^+\}$, computed using different past novelty or count estimates.[6] This results in slow adaptation to the non-stationary objective of exploration. DEEP makes two changes to mitigate the impact of the non-stationarity in the exploration reward function.

First, DEEP leverages access to $R_n^+$ to compute exploration rewards when they are needed to update $\pi_{\text{explore}}$ rather than when the transition is collected. We choose to use Q-learning rather than a more sophisticated algorithm in order to learn $\pi_{\text{explore}}$ as rapidly as possible; with changing rewards, using a policy to amortize the maximization of a value function as in SAC or DDPG would slow down learning. We represent $\pi_{\text{explore}}$ directly as a Boltzmann policy of this exploration Q-function $Q_{\text{explore}}$:

$$\pi_{\text{explore}}(a \mid s) \propto \exp\left\{\frac{Q_{\text{explore}}(s,a)}{\tau_{\text{explore}}}\right\}, \tag{3.2}$$

where $\tau_{\text{explore}}$ is a temperature hyperparameter.

Second, by decoupling $\pi_{\text{explore}}$ from $\pi_{\text{task}}$, DEEP unlocks the ability to update $\pi_{\text{explore}}$ more

aggressively without affecting $\pi_{\text{task}}$'s convergence to the optimal policy. This enables the exploration policy to more rapidly adapt to the non-stationary exploration reward. In our experiments we achieve this by using a larger learning rate and more updates per environment step than is usually done; future work might investigate more sophisticated schemes such as prioritized sweeping [Moore and Atkeson 1993] or prioritized experience replay [Schaul et al. 2016]. To improve stability we use DDQN updates and clip Q targets at $\bar{r}$, the maximum discounted exploration value.

{sec:optimistic}

Optimism. Further adapting $\pi_{\text{explore}}$ to the unique properties of the exploration reward function, we propose to leverage optimism in its updates and actions. We propose to make $Q_{\text{explore}}$ optimistic by leveraging the pseudo-count function in a manner similar to that proposed by Rashid et al. [2020]. We assume that the value function is trustworthy for transitions with very large counts, and very untrustworthy for transitions with near-zero counts. When the count is zero we impose an optimistic prior which assumes the transition will lead to a whole episode of novel transitions; as the count increases we interpolate between this prior and the learned value function using a weighting function:

$$Q^{+}_{\text{explore}}(s,a) = w(s,a)\cdot Q_{\text{explore}}(s,a) + (1-w(s,a))\cdot \bar{r}, \qquad w(s,a) = \frac{\sqrt{N(s,a)}}{\sqrt{N(s,a)+c}} \tag{3.3}$$

{{eq:optimism}}

where $\bar{r} = 1/1-\gamma$, is the maximum discounted return in the bonus MDP and $c$ is a small constant representing how many counts' worth of confidence to ascribe to the optimistic prior. We use this optimistic $Q^{+}_{\text{explore}}$ for computing targets for Bellman updates and for computing $\pi_{\text{explore}}$ (Eq. 3.4). For details of the implementation of the fast-adapting $\pi_{\text{explore}}$, see Appendix 3.C.

### 3.4.4 Product distribution behavior policy

A good behavior policy should attempt to explore all of the transitions which are relevant for learning the optimal policy. This entails a trade-off between taking actions which are more novel and ones which are more likely to be relevant to a high-performing policy. DEEP encodes this by representing the behavior policy as a product of the task policy $\pi_{\text{task}}$ and the pure-exploration policy $\pi_{\text{explore}}$:

$$\beta(a \mid s) \propto \pi_{\text{task}}(a \mid s)\, \pi_{\text{explore}}(a \mid s). \tag{3.4}$$ {{eq:behavior_policy}}

The choice to parameterize $\beta$ as a factored policy was made for its simplicity and ease of off-policy learning. Alternative formulations for making this trade-off while preserving the unbiased task policy are possible, and we view the form of our proposed behavior policy as just one option among many. One alternative would be interleaving the behavior of multiple policies within one episode, akin to e.g. Scheduled Auxiliary Control [Riedmiller et al. 2018].

In order to approximately sample from this behavior policy, we use self-normalized importance sampling with $\pi_{\text{task}}$ as the proposal distribution:

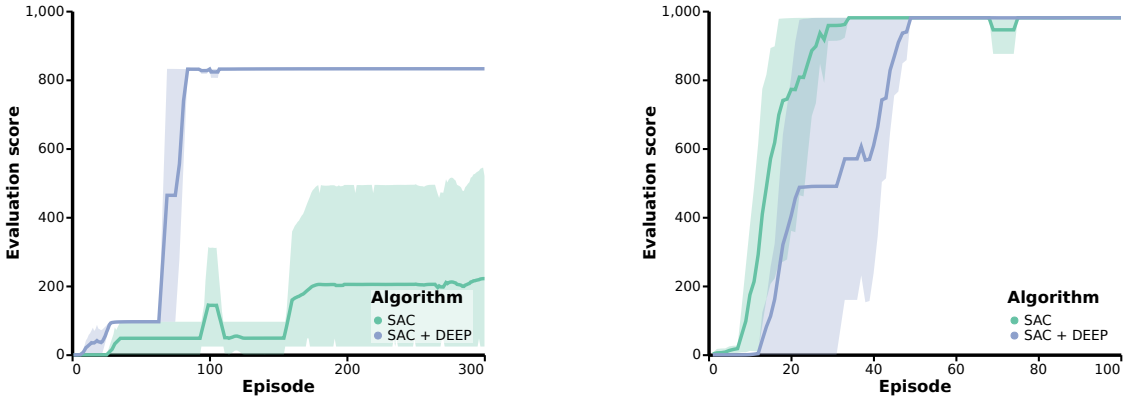
1. Draw $k$ samples $a_1, \ldots, a_k$ from $\pi_{\text{task}}$
2. Evaluate $\pi_{\text{explore}}(a_i \mid s)$ for each $i \in 1 \ldots k$
3. Draw a sample from the discrete distribution $p(a_i) = \pi_{\text{explore}}(a_i \mid s) / \sum_{i'} \pi_{\text{explore}}(a_{i'} \mid s)$.

Note that since the proposal distribution is $\pi_{\text{task}}$, the $\pi_{\text{task}}$ terms in computing weights cancel and only the $\pi_{\text{explore}}$ terms remain. Importance weighting is consistent in the limit of $k \to \infty$ but introduces bias towards $\pi_{\text{task}}$ [Vehtari et al. 2015]. With small $k$, this bias makes it unlikely that $\beta$ will select actions that are very unlikely under $\pi_{\text{task}}$; roughly speaking, this procedure selects the "most exploratory" action in the support of the task policy. This may act as a backstop to prevent

the behavior from going too far outside the task policy to be useful. DEEP works best when $\pi_{\text{task}}$ is trained in a way that preserves variance in the policy (e.g. SAC's target entropy), enabling the behavior policy to select exploratory actions. In discrete action spaces we additionally use self-normalized importance sampling – using a uniform proposal over actions – to obtain samples from $\pi_{\text{task}}$ in step 1.

## 3.5 Experiments

In this section we perform experiments to give insight into the behavior of undirected exploration, BBE, and DEEP. First we perform a set of investigative experiments to probe how DEEP interacts with environments with different reward structures. Then we perform experiments on pairs of benchmark continuous control tasks with easy and hard exploration requirements.




**(a)** Local optimum environment **(b)** Adversarial environment

{sub:hallway_two_local_optimum} {sub:fingertip_dist

**Figure 3.4:** Two environments illustrating different reward structures. **(a)** An environment with a locally-optimal goal (reward 0.1) near the start state. SAC finds this nearby goal, but doesn't explore far enough to find the real goal (reward 1.0). When trained with DEEP, it finds the distractor goal but moves on to the real goal. **(b)** An adversarial environment for DEEP which has a very small goal state close to the start state, making it easy to find with random actions but hard with directed exploration.

{fig:hallway}

### 3.5.1 Investigative experiments

We construct a simple MuJoCo [Todorov et al. 2012] environment called Hallway to look more closely at how exploration interacts with reward structure. This environment consists of a long narrow 2D room with the agent controlling the velocity of a small sphere, which starts each episode at one end of this hallway. The following two experiments share dynamics and differ only in their rewards.

Local optima. A valuable role for exploration is enabling an agent to escape from locally optimal behavior. To test this, we add two goal states with shaped rewards to the Hallway environment. The first is close to the start state but only provides reward at most 0.1, while the second is at the far end of the hallway but provides reward 1.0. Figure 3.4(a) shows that exploration using DEEP allows the agent to quickly find its way to the faraway optimal reward while SAC gets stuck in the local optimum.

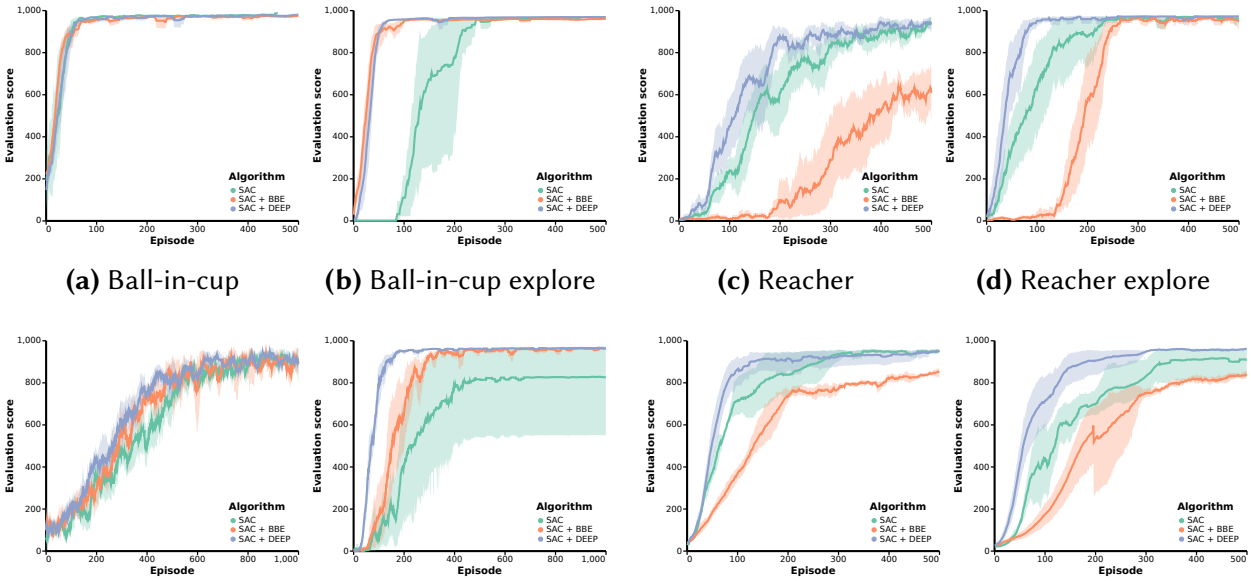
Limitations. DEEP covers states quickly, but there is no such thing as a universally optimal exploration strategy. For example, there exist environments for which the random walk dynamics of undirected exploration find the optimal strategy faster than uniform state coverage. Figure 3.4(b) provides one such example: a Hallway environment with a very small goal state close to the start state. SAC discovers this goal faster than SAC + DEEP, though DEEP does eventually find it as well.

### 3.5.2 Benchmark experiments

Next we provide experiments based on DeepMind Control Suite [Tassa et al. 2018], a standard benchmark for continuous control RL algorithms. We introduce versions of several environments which are modified to remove the accommodations that make them solvable without exploration,

then provide results of SAC, SAC + BBE, and SAC + DEEP on the original and modified environments.

#### 3.5.2.1 Results




**(a)** Ball-in-cup **(b)** Ball-in-cup explore **(c)** Reacher **(d)** Reacher explore

**(e)** Finger **(f)** Finger explore **(g)** Walker **(h)** Walker explore

**Figure 3.5:** Results on original Control Suite environments (left in each pair) and modified versions without exploratory resets and rewards (right). Across the original environments, SAC + DEEP performs as well or better than SAC, while SAC + BBE performs much worse on some environments. On the exploration environments, DEEP + SAC learns much faster than SAC. BBE sometimes provides significant gains over SAC but sometimes performs worse even on exploration environments. {fig:control_suite}

#### 3.5.2.2 Environments for evaluating exploration

While Control Suite has driven great progress in policy learning, it was not designed to evaluate an agent's exploration capabilities; in fact, the included environments were selected to be solvable by algorithms with only undirected exploration. From that work:

> We ran variety of learning agents (e.g. Lillicrap et al. 2015; Mnih et al. 2016) against all tasks, and iterated on each task's design until we were satisfied that [...] the task is solved correctly by at least one agent. [Tassa et al. 2018]

Control Suite avoids the need for directed exploration via two mechanisms. First, in many environments the start state distribution is sufficiently wide (e.g. uniform over reachable states) to guarantee that any policy will see high-value states.[7] Second, some environments have rewards shaped to guide the agent towards better performance (e.g. a linearly increasing reward for forward walking speed).

To construct a benchmark for continuous control with exploration, we selected four environments with different objectives (manipulation and locomotion, single-objective or goal-conditional) and observation dimensions (6-24). We then created "exploration" versions of these environments with restricted start state distributions and sparse rewards. The original environments and their exploration versions together form a benchmark which measures an algorithm's exploration ability and policy convergence. Environment details are in Appendix 3.D and their implementation is in the supplement.

#### 3.5.2.3 Algorithms

We include experiments on these eight benchmark tasks with three algorithms: SAC [Haarnoja et al. 2018c] with no additional exploration; BBE with SAC for the policy learner; and DEEP with SAC for $\pi_{\text{task}}$ and DDQN for $\pi_{\text{explore}}$. BBE and DEEP use the pseudo-count reward described in Section 3.4.1 and the SAC implementation is that of Yarats and Kostrikov [2020] with no hyperparameter changes.

The kernel-based exploration bonus used for BBE and DEEP requires a scaling law to set the kernel variance as a function of the observation dimension. We adapt the scaling relationship from [Henderson and Parmeter 2012] (see Appendix 3.B). BBE has an additional hyperparameter for the scale of the bonus. We performed a sweep with values in $\{10^{-2}, 10^{-1}, 1, 10\}$ and found that 1 performed best overall. This setting, which makes the maximum bonus equal to the maximum environment reward, ensures that visiting a new state remains the best option until the true goal state is discovered. Further implementation details are available in Appendix 3.E. We additionally

performed an ablation which, like DEEP, learns separate Q functions for the two rewards, but which learns one policy to maximizes the sum of their Q values. In our experiments (available in Appendix 3.F) this ablation never outperforms BBE, so for clarity we exclude it here.

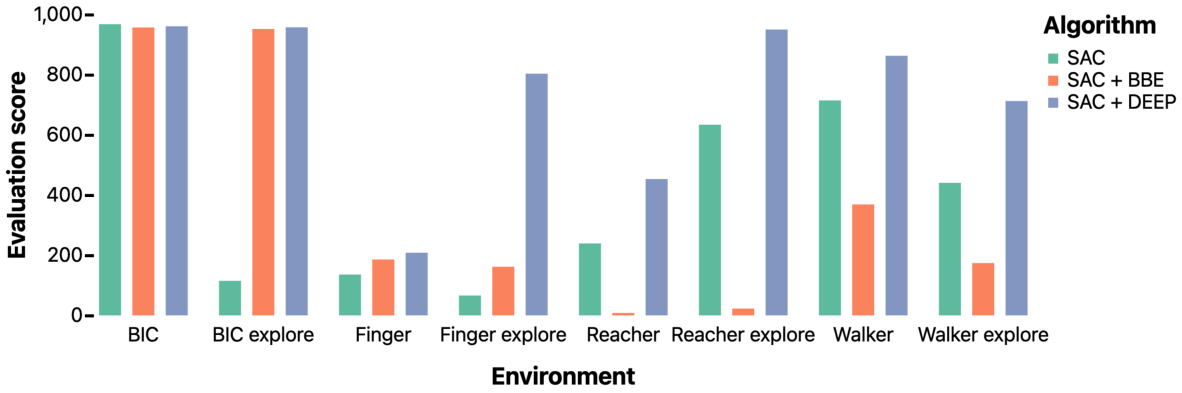



**Figure 3.6:** Results after 100 episodes. In this extremely sample-limited regime, exploration speed and fast policy convergence are both essential. In every environment, SAC with DEEP (blue, right column in each set of three) performs comparably to or better than SAC alone or SAC with BBE.

{fig:control_suite_su

We present experiments on the original versions of four Control Suite tasks and their exploration counterparts. The results are shown in Figure 3.5 with the means and 95% confidence intervals over 8 seeds. We find that across the original environments, DEEP gives similar or slightly better performance to SAC, while BBE significantly impairs SAC on two of the four environments and matches SAC on the other two. Across the exploration environments, DEEP gives the best performance and sample efficiency. BBE performs better than SAC alone on two exploration environments and worse than SAC on the other two. Figure 3.6 shows the performance of each algorithm after only 100 episodes, highlighting the substantial benefits from using DEEP in the few-sample regime.

Overall, SAC + DEEP never performs worse than SAC alone, while yielding substantial improvements in environments where rewarding states are harder to discover. BBE's more mixed performance provides a possible explanation for the limited influence that methods of that family have had on sample-efficient continuous control, and perhaps more generally on sample-efficient RL. Given that in this setting the addition of BBE is as likely to harm as to help, its lack of adoption

is unsurprising.

## 3.6 Related work

Sample-efficient continuous control. Our method leverages progress on sample efficient off-policy RL, as it can be combined with any off-policy algorithm. A strong line of work has brought the sample complexity of model-free control within range of solving tasks on real robots [Popov et al. 2017; Kalashnikov et al. 2018; Haarnoja et al. 2018b; Fujimoto et al. 2018c; Haarnoja et al. 2018c; Abdolmaleki et al. 2018].

Bonus-based exploration. There have been many bonuses proposed in the BBE framework. Several works [Stadie et al. 2015; Pathak et al. 2017; Burda et al. 2018] propose to use prediction error of a learned model to measure a transition's novelty, with the key differences being the state representation used for making predictions. Houthooft et al. [2016] propose a bonus based on the information gain of the policy. Bellemare et al. [2016] and others [Ostrovski et al. 2017; Tang et al. 2017] use continuous count analogues to calculate the count-based bonuses of Strehl and Littman [2008]. Machado et al. [2020] use the norm of learned successor features as a bonus, and show that it implicitly counts visits. Unlike previous work, our paper focuses on the updates and representation of the behavior policy, and DEEP can be used in conjunction with any of these bonuses. Never Give Up [Badia et al. 2020] uses an episodic exploration bonus and trains policies with different bonus scales including a task policy. However, it is designed to maximize asymptotic performance rather than sample efficiency and does not learn faster than a baseline early in training.

Optimism. Classic exploration methods [Kearns and Singh 1998; Brafman and Tennenholtz 2002; Strehl and Littman 2008; Jaksch et al. 2008], depend on an optimistically-defined model. Model-free methods with theoretical guarantees [Strehl et al. 2006; Jin et al. 2018] use Q functions

initialized optimistically. Similar to our Eq. (3.3), Rashid et al. [2020] propose a method for ensuring optimism in Q learning with function approximation by using a count function. However, DEEP leaves the task policy unbiased in the few-sample regime by separating the exploration policy from the task policy.

Temporally-extended actions. A variety of work proposes to speed up $\epsilon$-greedy exploration via temporally-extended actions which reduce dithering. Some methods [Schoknecht and Riedmiller 2003; Neunert et al. 2020] propose to bias policies towards repeating primitive actions, resulting in faster exploration without limiting expressivity. Dabney et al. [2020] describe a temporally-extended version of $\epsilon$-greedy exploration which samples a random action and a random *duration* for that action. Whitney et al. [2020] use a learned temporally-extended action space representing the reachable states within a fixed number of steps. While these methods improve over single-step $\epsilon$-greedy, they are unable to perform directed exploration or discover faraway states.

Randomized value functions. Modern works [Osband et al. 2016, 2019] extend Thompson sampling [Thompson 1933] to neural networks and the full RL setting. Relatedly, [Fortunato et al. 2018; Plappert et al. 2018] learn noisy parameters and sample policies from them for exploration.

## 3.7 Discussion

In this paper we have investigated the potential for directed exploration to improve the sample efficiency of RL in continuous control. We found that BBE suffers from bias and slow state coverage, leading to performance which is often worse than undirected exploration. We introduced Decoupled Exploration and Exploitation Policies, which separately learns an unbiased task policy and an exploration policy and combines them to select actions at training time. DEEP pays no performance penalty even on dense-reward tasks and explores faster than BBE. In our ex-

periments, DEEP combined with SAC provides strictly better performance and sample efficiency than SAC alone. We believe that with its combination of reliable and efficient policy learning across dense and sparse environments, SAC + DEEP provides a compelling default algorithm for practitioners.

## Appendix 3.A Grid-world visualizations

{sec:gridworld-vis}

The grid-world environment we use consists of a 40x40 environment with actions up, down, left, and right, with a single start state in the lower left and a single goal state in the upper right. In Figure 3.7 we show the state of each algorithm after training on the grid-world for 100 episodes. While DDQN and BBE have only visited a small fraction of the states, DEEP has covered most of the environment and will soon solve it. As they continue to run BBE will eventually find the goal while DDQN will not.

A video version of Figure 3.7, which shows its evolution over time, is available in the supplement. We find that watching the qualitative behavior of each algorithm can be enlightening. Note that as the figures and videos are rendered by sampling states and actions, some noise in the form of missing states may appear.

## Appendix 3.B Pseudo-count implementation

{sec:count_implementa

Define the Gaussian kernel with dimension $d$ as

$$k_{\text{Gauss}}(x, x_i) = (2\pi)^{-\frac{d}{2}} \det(\Sigma)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(x - x_i)^\intercal \Sigma^{-1}(x - x_i)\right\}. \tag{3.5}$$

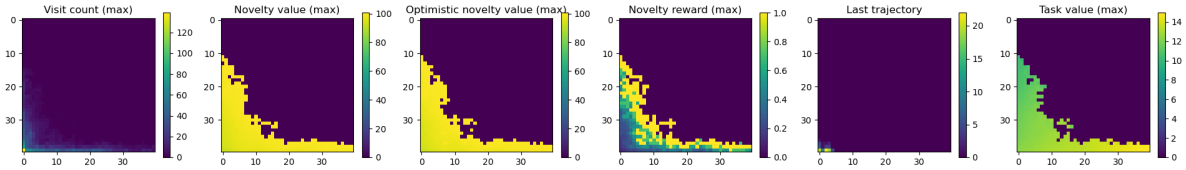
We can normalize this function to have a maximum at $k(x, x) = 1$ simply by removing the normalizer (everything outside the exponential) by noting that $e^0 = 1$. This gives the kernel we use:

$$k(x, x_i) = \exp\left\{-\frac{1}{2}(x - x_i)^\intercal \Sigma^{-1}(x - x_i)\right\}, \tag{3.6}$$

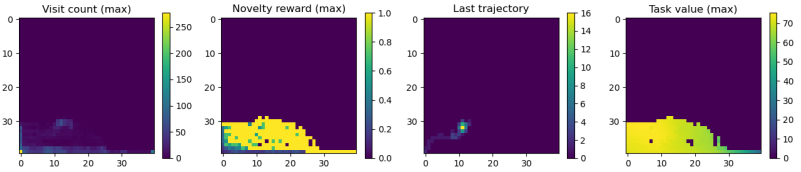
{{eq:count_kernel}}

where the covariance is a diagonal matrix $\Sigma = \text{diag}(\sigma_1^2, \ldots, \sigma_d^2)$.

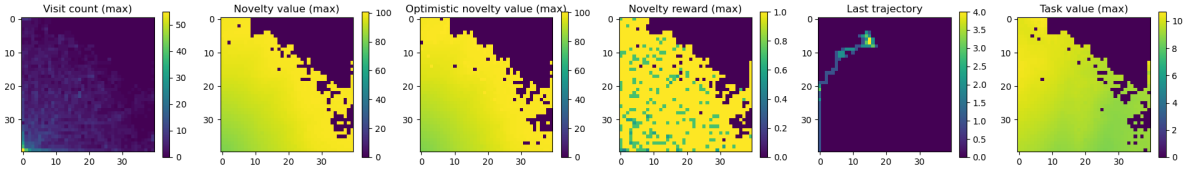
To compute $\hat{N}_n(s, a)$ using this kernel, we perform the following steps:




**(a)** DDQN




**(b)** BBE




**(c)** DEEP

**Figure 3.7:** The state of each algorithm after 100 episodes in the grid-world environment. Note that the novelty reward and novelty value shown for DDQN are for visualization purposes only, as the algorithm does not use them. The "Task value" shown for BBE is the sum of the task and novelty rewards, which BBE treats as its objective. Un-visited states are marked as zero in each plot. The annotation (max) indicates that the value shown is the maximum value for any action at that state. "Last trajectory" shows the states visited in the last training episode. Figures generated by sampling.

{fig:gridworld_algori

1. Normalize $s$ and $a$:

$$\bar{s} = \frac{s - \mathcal{S}_{\min}}{\mathcal{S}_{\max} - \mathcal{S}_{\min}} \qquad \bar{a} = \frac{a - \mathcal{A}_{\min}}{\mathcal{A}_{\max} - \mathcal{A}_{\min}} \tag{3.7}$$

2. Define $x = [\bar{s}, \bar{a}]$ as the concatenation of the normalized state and action.

3. Compute the kernel from Eq. (3.6) and sum across all of the previous normalized observations $x_i$:

$$\hat{N}_n(s, a) = \hat{N}_n(x) = \sum_{i=1}^{n} k(x, x_i) \tag{3.8}$$

For this final step, we leverage the MIT-licensed Kernel Operations library (KeOps, Charlier et al. [2021]), a high-performance GPU-accelerated library for efficient computation of reductions on symbolically-defined matrices. This substantially outperforms implementations in other frameworks, including fully-JITted JAX [Bradbury et al. 2018] version, especially as the dimension of the data grows.

To avoid having to tune the covariance for each environment, we adapt the scaling rule of [Henderson and Parmeter 2012]. This rule of thumb requires assumptions on the data (notably, that it comes i.i.d. from a Normal) which are violated in the exploration problem. However, we find this scaling to be useful in practice. The rule of thumb bandwidth for dimension $j$ of the data for a multivariate Gaussian kernel is

$$h_j^{ROT} = \left(\frac{4}{2+d}\right)^{\frac{1}{4+d}} \hat{\sigma}_j n^{-\frac{1}{4+d}}, \tag{3.9}$$

where $\hat{\sigma}_j$ is the empirical variance of dimension $j$ of the data. Making the assumption that our states are normalized to be in $[-1, 1]$ and distributed uniformly, we can set $\hat{\sigma}_j \approx 0.3$. As such, in every experiment with continuous-valued states, we set the kernel variance for dimension $j$ as

$$\sigma_j = 0.3 \left(\frac{4}{2+d}\right)^{\frac{1}{4+d}} n^{-\frac{1}{4+d}}. \tag{3.10}$$

In experiments we find that changes to this scale of less than an order of magnitude make little difference.

Throughout we use 1 for the kernel variance on the action dimensions.

### 3.B.1 Updating the kernel estimator

Updates to the kernel count estimator consist of appending new normalized observations to the set $\{x_i\}$. However, we find that computing this kernel becomes prohibitively slow beyond roughly

100K entries, and our experiments run for up to 1M steps. We take two steps to avoid this slowdown. Both rely on nonuniform weighting of entries in the observation table when computing the count, leading us to maintain an additional array of weights in addition to the table of observations.

Avoiding duplicate entries. If a new observation $x$ has $k(x, x_i) > 0.95$ with some set $M$ of existing indices, we do not add $x$ to the table, and instead add $1/|M|$ to each entry $i \in M$ of the weights table. In essence, if there is an exact duplicate for $x$ in the table already, we simply count that existing entry twice. While this is helpful, the probability of observing exact matches decreases rapidly in the dimension of the observations, so this step plays a limited role.

Evicting previous entries. Once the length of the observations table reaches some maximum ($2^{15} = 32768$ in our experiments), we evict an existing entry in the table uniformly at random when we make a new entry, thus maintaining the table at that maximum size. This introduces risk that the exploration bonus would not go to zero in the limit of many environment steps, which we avoid by re-distributing the weight of the evicted observations among those still in the table. We do this redistribution uniformly; that is, if we evict the entry at location $i$, with weight $w_i$, we add $w_i/(n-1)$ to the weight of each of the $(n-1)$ other entries. Our reweighting procedure maintains the same *total* amount of count when evicting observations and ensures that bonuses go to zero in the limit of data. In experiments we find that the exploration rewards earned when using a very small observation table (and thus, many evictions) were practically indistinguishable from using an observation table of unlimited size.

### 3.B.2 Tabular environments

For the grid-world environment used in Figures 3.2 and 3.3, we use a tabular visit count rather than pseudo-counts.

## Appendix 3.C Details on rapid Q updates

{sec:fast_updates_app

We aim to rapidly update $Q_{\text{explore}}$ to maximize reward on the non-stationary exploration MDP $\mathcal{M}_{R_n^+}$ and thus explore rapidly. This has three components: (1) updating using the current reward function $R_n^+$ rather than logged rewards, (2) performing many updates to $Q_{\text{explore}}$ at every timestep using a large learning rate, and (3) using an optimistic version of $Q_{\text{explore}}$ which is aware of the high value of taking actions that have not yet been explored. However, aggressively updating $Q_{\text{explore}}$ poses its own problems; most significantly, Q-learning with function approximation has a tendency to diverge if updated too aggressively with too few new samples. We use three modifications to the typical Bellman update with target networks [Mnih et al. 2015a] to mitigate this issue while incorporating optimism.

- **Soft DoubleDQN update.** The DoubleDQN [Hasselt et al. 2016] update reduces overestimation in Q-learning by selecting and evaluating actions using different parameters. We use a soft version of the DoubleDQN update by replacing the max operator with an exponential-Q policy over uniform random actions using a low temperature.

- **Value clipping.** To further mitigate the problem of Q-learning overestimation and divergence, we clip the Bellman targets to be within the range of possible Q values for $\mathcal{M}_{R_n^+}$. Given that the rewards $r^+$ are scaled to be in $[0, 1]$, any policy would have a value $Q_{\text{explore}}(s, a) \in [0, \bar{r}]$, where $\bar{r} = 1/1-\gamma$.

- **Optimistic targets.** We use the optimistic adjustment in Eq. (3.3) when computing targets.

Define the softmax-Q policy for some Q function $Q$ as

$$\pi(a \mid s; Q) = \frac{\exp\left\{Q(s,a)/\tau\right\}}{\int_{\mathcal{A}} \exp\left\{Q(s,a')/\tau \, da'\right\}} \tag{3.11}$$

which we approximate using self-normalized importance sampling with a uniform proposal distribution. The target for updating $Q_{\text{explore}}$ is

$$y(s, a, s') = \text{clip}\left(R_n^+(s, a) + \gamma \mathop{\mathbb{E}}_{a' \sim \pi(\cdot|s'; Q^+_{\text{explore}})}\left[Q^+_{\text{explore}}(s', a'; \theta^-)\right], 0, \bar{r}\right). \tag{3.12}$$ {{eq:update_target}}

where $R_n^+(s, a)$ is the current exploration bonus, which we recompute at update time; $Q^+_{\text{explore}}(s', a'; \theta^-)$ is the target network for the exploration value function, with optimism applied. We then minimize the squared error between $y(s, a, s')$ and $Q_{\text{explore}}(s, a)$.

## Appendix 3.D Environments for exploration

{sec:environments_app

To enable benchmarking the performance of exploration methods on continuous control, we constructed a new set of environments. Our motivation comes from the challenges of performing resets and defining shaped rewards in real-world robotics, where it is not possible to measure and set states exactly. Unlike in simulation, it may be difficult or impossible to implement uniform resets of the robot and the objects in the scene; states with a walking robot standing upright or a block in midair require significant expertise to reach. Similarly many shaped rewards in simulation rely on precise knowledge of the locations of objects in a scene to provide rewards corresponding to e.g. an objects distance from a goal. We make modifications which capture the spirit of these real-world constraints, though these exact environments might still be difficult to construct in the real world:

- Small reset distributions. Instead of resetting every object in the scene uniformly in the space, we randomize each object's configuration over a smaller set of starting states. This reflects some properties of real environments, such as walking robots starting on the ground instead of midair, or the object in a manipulation task not starting in its goal receptacle.

- Sparse rewards. While dense rewards are difficult to construct without real-time monitor-

ing of object positions, sparse rewards are often simpler. A picking task, for example, can provide a sparse reward simply by checking whether the desired object is inside a receptacle.

As the base environments for our benchmark, we select four tasks from DeepMind Control Suite [Tassa et al. 2018], an Apache-licensed standard set of benchmarks implemented using the commercial MuJoCo physics simulator [Todorov et al. 2012]. Denoting the state (observation) and action dimensions of an environment as $\dim(\mathcal{S}) \to \dim(\mathcal{A})$, these environments are:

- **Ball-in-cup catch** (manipulation, $8d \to 2d$).
- **Reacher hard** (goal-directed, $6d \to 2d$).
- **Finger turn_hard** (manipulation, goal-directed, $12d \to 2d$).
- **Walker walk** (locomotion, $24d \to 6d$).

We modify each environment to remove the accommodations of wide reset distributions and sparse rewards which make them easy to solve without directed exploration. The new environments and their changes are as follows:

- **Ball-in-cup explore** (manipulation, $8d \to 2d$). The original task resets the ball uniformly in the reachable space, including already in the cup (the goal state). We modify the environment to only reset the ball in a region below the cup, as if the ball was hanging and gently swinging. The original task already has sparse rewards.
- **Reacher explore** (goal-directed, $6d \to 2d$). The original task samples arm positions and goal positions uniformly, resulting in the arm being reset very near the goal. We modify the reset distribution to only include states with the arm mostly extended to the right and targets opposite it on the left in a cone with angle $\pi/2$. Note that this task is somewhat easier than the original since the policy only needs to navigate between smaller regions

of the space, but is harder due to the resets not providing exploration. The original task already has sparse rewards.

- **Finger explore** (manipulation, goal-directed, $12d \rightarrow 2d$). The original task resets the finger uniformly, the angle of the spinner uniformly, and the goal location uniformly on the circle reachable by the tip of the spinner. We modify the environment to reset the finger joints each in the quadrant pointing away from the spinner, the spinner pointing in the downward quadrant, and the target in the upward quadrant. Similarly to Reacher explore, this task is simpler than the original but harder to explore in. The original task already has sparse rewards.

- **Walker explore** (locomotion, $24d \rightarrow 6d$). The original environment resets the walker just above the ground with random joint angles, leading to it frequently starting upright and in position to begin walking. We modify the environment by allowing time to progress for 200 steps in the underlying physics (20 environment steps), which is enough time for the walker to fall to the floor. This simulates the walker starting in a random lying-down configuration. The original rewards include a sparse reward for being upright and above a certain height and a linear component for forward velocity. We replace the forward velocity reward with a sparse version which provides reward only when the agent is moving at or above the target speed.

The code for these environments is included in the supplement.

## Appendix 3.E Experimental implementation details

{sec:appendix_benchma

### 3.E.1 Computing infrastructure

We implemented DEEP using JAX [Bradbury et al. 2018] and the neural network library Flax [Heek et al. 2020] which is built on it, both of which are Apache-licensed libraries released by

Google. The SAC implementation we use is from Yarats and Kostrikov [2020] (MIT-licensed), built on Pytorch [Paszke et al. 2019a] (custom BSD-style license). The experiments in this paper take about a week to run using 16 late-model NVidia GPUs by running 2-4 seeds of each experiment at once on each GPU.

### 3.E.2 Network architectures and training

For the Q networks used as $Q_{\text{explore}}$ in continuous environments and as the task policy in the grid-world experiments, we use fully-connected networks with two hidden layers of 512 units each and ReLU activations. These networks flatten and concatenate the state and action together to use as input and produce a 1d value prediction. This enables us to use the same networks and training for discrete and continuous actions rather than using the usual discrete-action trick of simultaneously producing a Q-value estimate for every action given the state.

These Q networks are trained using the Adam optimizer [Kingma and Ba 2014] with learning rate $10^{-3}$. $Q_{\text{explore}}$ is updated with two Bellman updates with batch size 128 per environment step. We update the target network after every environment step for $Q_{\text{explore}}$ to allow very rapid information propagation about changing rewards. For the Q network defining $\pi_{\text{task}}$ in the grid-world we use a learning rate of $10^{-4}$ and update the target network after every 50 Bellman updates.

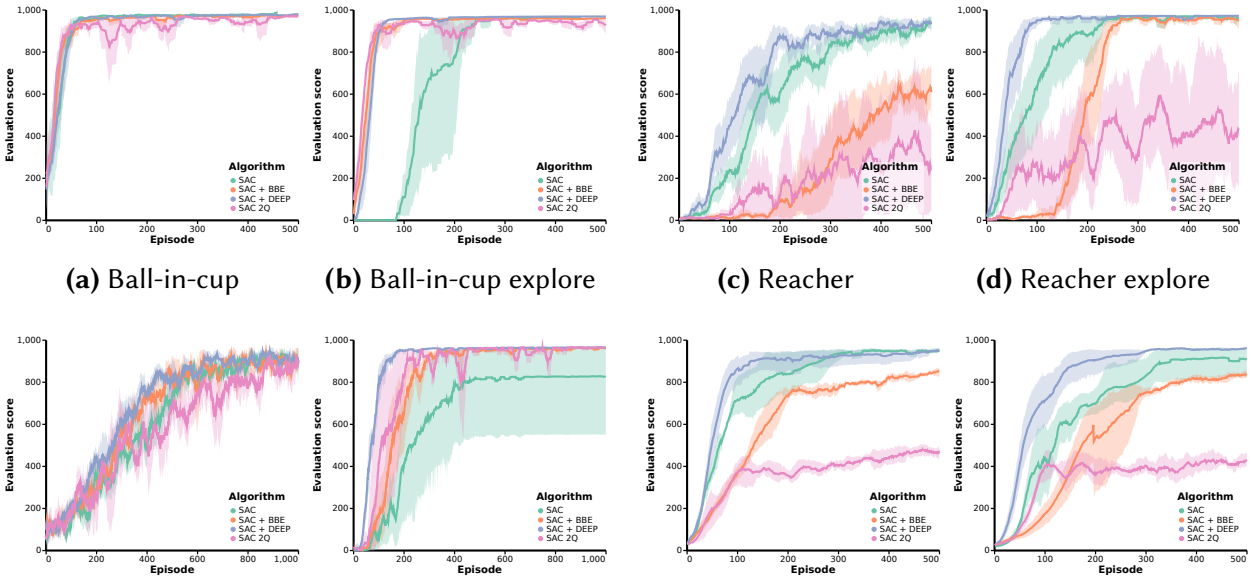
### 3.E.3 Other hyperparameters

We draw 64 samples from $\pi_{\text{task}}$ when computing the behavior policy and 64 samples from a uniform distribution over actions when updating $Q_{\text{explore}}$ as described in Appendix Section 3.C. We set the temperature for Boltzmann sampling from all Q-network policies as $\tau = 0.1$. $Q_{\text{explore}}$ uses a discount $\gamma = 0.99$.

# Appendix 3.F Additional benchmark results

{sec:benchmark_result

We performed an experiment to check whether it was simply the separation of learning two separate Q functions which enabled DEEP's performance. To do this, we modified SAC + BBE to learn one Q function for the task reward function and one Q function for the exploration reward function. The policy was then trained to maximize the sum of those two Q functions. This baseline, which we call SAC 2Q, performed uniformly worse than SAC + BBE, but we include its results here for completeness.




**(a)** Ball-in-cup **(b)** Ball-in-cup explore **(c)** Reacher **(d)** Reacher explore

**(e)** Finger **(f)** Finger explore **(g)** Walker **(h)** Walker explore

**Figure 3.8:** SAC 2Q performs uniformly worse than SAC + BBE.

# Notes

5. Some implementations of bonus-based exploration may update the policy within an episode, for example via a single gradient step per environment step on transitions sampled i.i.d. from a replay. However, such a small update is typically not enough to change the qualitative behavior of the agent and adapting to the changing MDP has not been an emphasis in prior work.

6. See e.g. the code from Machado et al. [2020]: https://github.com/mcmachado/count_based_exploration_sr/blob/master/function_approximation/exp_eig_sr/train.py#L204

7. Some environments, such as Manipulator, additionally start a fraction of episodes at the goal state.

# Part II

# Representations and Auxiliary Tasks

{sec:representation}

Introduction to representation for RL.

# 4 Evaluating Learned Representations

{sec:representation-e

## 4.1 Introduction

One of the first steps in building a machine learning system is selecting a representation of data. Whereas classical machine learning pipelines often begin with feature engineering, the advent of deep learning has led many to argue for pure end-to-end learning where the deep network constructs the features [LeCun et al. 2015]. However, huge strides in unsupervised learning [Hénaff et al. 2020; Chen et al. 2020a; He et al. 2019; van den Oord et al. 2018; Bachman et al. 2019; Devlin et al. 2019; Liu et al. 2019b; Raffel et al. 2019; Brown et al. 2020] have led to a reversal of this trend in the past two years, with common wisdom now recommending that the design of most systems start from a pretrained representation. With this boom in representation learning techniques, practitioners and representation researchers alike have the question: Which representation is best for my task?


**Figure 4.1:** The representation learning pipeline.

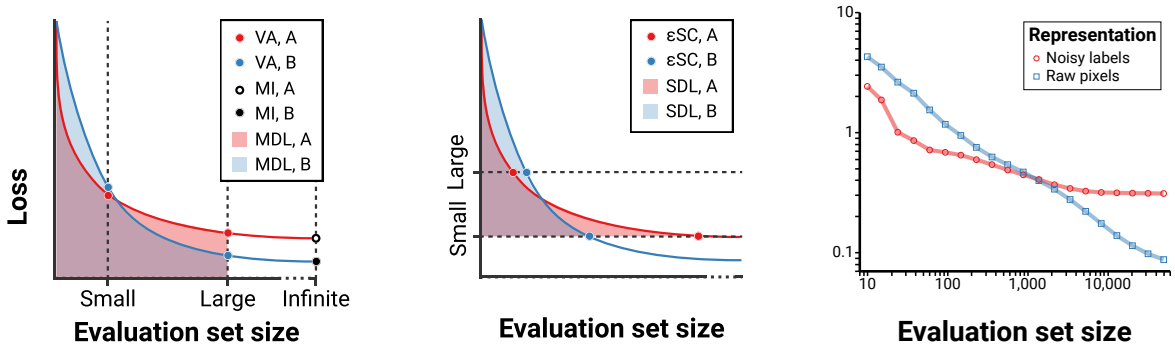
{fig:representation_p

This question exists as the middle step of the representation learning pipeline shown in Figure 4.1. The first step is representation learning, which consists of training a representation

function on a training set using a pretext objective, which may be supervised or unsupervised. The second step, which this paper considers, is representation evaluation. In this step, one uses a measure of representation quality and a labeled *evaluation dataset* to see how well the representation performs. The final step is deployment, in which the practitioner or researcher puts the learned representation to use. Deployment could involve using the representation on a stream of user-provided data to solve a variety of end tasks [LeCun 2015], or simply releasing the trained weights of the representation function for general use. In the same way that BERT [Devlin et al. 2019] representations have been applied to a whole host of problems, the task or amount of data available in deployment might differ from the evaluation phase.

We take the position that the best representation is the one which allows for the most *efficient* learning of a predictor to solve the task. We will measure efficiency in terms of either number of samples or information about the optimal predictor contained in the samples. This position is motivated by practical concerns; the more labels that are needed to solve a task in the deployment phase, the more expensive to use and the less widely applicable a representation will be. To date the field has lacked clearly defined and motivated tools for analyzing the complexity of learning with a given representation. This work seeks to provide those tools for the representation learning community.

We build on a substantial and growing body of literature that attempts to answer the question of which representation is best. Simple, traditional means of evaluating representations, such as the validation accuracy of linear probes [Ettinger et al. 2016; Shi et al. 2016; Alain and Bengio 2016], have been widely criticized [Hénaff et al. 2020; Resnick et al. 2019]. Instead, researchers have taken up a variety of alternatives such as the validation accuracy (VA) of nonlinear probes [Conneau et al. 2018; Hénaff et al. 2020], mutual information (MI) between representations and labels [Bachman et al. 2019; Pimentel et al. 2020], and minimum description length (MDL) of the labels conditioned on the representations [Blier and Ollivier 2018; Yogatama et al. 2019; Voita and Titov 2020].




**(a)** Existing measures **(b)** Proposed measures **(c)** Illustrative experiment

**Figure 4.2:** Each measure for evaluating representation quality is a simple function of the "loss-data" curve shown here, which plots validation loss of a probe against evaluation dataset size. **Left:** Validation accuracy (VA), mutual information (MI), and minimum description length (MDL) measure properties of a given evaluation dataset, with VA measuring the loss at a finite amount of evaluation data, MI measuring it at infinity, and MDL integrating it from zero to $n$. This dependence on evaluation dataset size can lead to misleading conclusions as the amount of available data changes. **Middle:** Our proposed methods instead measure the complexity of learning a predictor with a particular loss tolerance. $\varepsilon$ sample complexity ($\varepsilon$SC) measures the number of samples required to reach that loss tolerance, while surplus description length (SDL) integrates the surplus loss incurred above that tolerance. Neither depends on the evaluation dataset size. **Right:** A simple example task which illustrates the issue. One representation, which consists of noisy labels, allows quick learning, while the other supports low loss in the limit of data. Evaluating either representation at a particular evaluation dataset size risks drawing the wrong conclusion.

We find that these methods all have clear limitations. As can be seen in Figure 4.2, VA and MDL are liable to choose different representations for the same task when given evaluation datasets of different sizes. Instead we want an evaluation measure which depends on the data *distribution*, not a particular evaluation dataset sample or evaluation dataset size. Furthermore, VA and MDL lack a predefined notion of success in solving a task. In combination with small evaluation datasets, these measures may lead to premature evaluation by producing a judgement even when there is not enough data to solve the task or meaningfully distinguish one representation from another. Meanwhile, MI measures the lowest loss achievable by any predictor irrespective of the number of samples required to learn it or the computational cost to compute it. None of these existing techniques measure the improved data efficiency that a good representation can yield, despite this being one of the primary applications for representation learning.

To eliminate these issues, we propose two measures of representation quality. In both of

our measures, the user specifies a tolerance $\varepsilon$ so that a population loss of less than $\varepsilon$ qualifies as solving the task. Then the measure computes the cost of learning a predictor which achieves that loss. The first measure is the *surplus description length* (SDL) which modifies the MDL to measure the complexity of learning an $\varepsilon$-loss predictor rather than computing the complexity of the labels in the evaluation dataset. The second is the *$\varepsilon$-sample complexity* ($\varepsilon$SC) which measures the sample complexity of learning an $\varepsilon$-loss predictor. These measures resolve the issues with prior work and provide tools for researchers and practitioners to evaluate the extent to which a learned representation can improve data efficiency. Furthermore, they formalize existing research challenges for learning representations which allow state of the art performance while using as few labels as possible (e.g. Hénaff et al. [2020]).

To facilitate our analysis, we also propose a framework called the *loss-data framework*, illustrated in Figure 4.2, that plots the validation loss against the evaluation dataset size [Talmor et al. 2019; Yogatama et al. 2019; Voita and Titov 2020]. This framework simplifies comparisons between measures. Prior work measures integrals (MDL) and slices (VA and MI) along the data axis. Our work proposes instead measuring integrals (SDL) and slices ($\varepsilon$SC) along the loss axis. This illustrates how prior work makes tacit choices about the function to learn based on the choice of evaluation dataset size. Our work instead makes an explicit, interpretable choice of what function to learn via the threshold $\varepsilon$ and measures the complexity of learning such a function. We experimentally investigate the behavior of these methods, illustrating the sensitivity of VA and MDL, and the robustness of SDL and $\varepsilon$SC, to evaluation dataset size.

Efficient implementation. To enable reproducible representation evaluation for representation researchers, we have developed a highly optimized open source Python package at `https://github.com/willwhitney/reprieve`. This package enables construction of loss-data curves with arbitrary representations and datasets and is library-agnostic, supporting representations and learning algorithms implemented in any Python ML library. By leveraging the JAX library

[Bradbury et al. 2018] to parallelize the training of probes on a single accelerator, our package constructs loss-data curves in around two minutes on one GPU.

## 4.2 The loss-data framework for representation evaluation

In this section we formally present the representation evaluation problem, define our loss-data framework, and show how prior work fits into the framework.

Notation. We use bold letters to denote random variables. A supervised learning problem is defined by a joint distribution $\mathcal{D}$ over observations and labels $(\mathbf{X}, \mathbf{Y})$ in the sample space $\mathcal{X} \times \mathcal{Y}$ with density denoted by $p$. Let the random variable $\mathbf{D}^n$ be a sample of $n$ i.i.d. $(\mathbf{X}, \mathbf{Y})$ pairs, realized by $D^n = (X^n, Y^n) = \{(x_i, y_i)\}_{i=1}^n$. This is the evaluation dataset. Let $\mathcal{R}$ denote a representation space and $\phi : \mathcal{X} \to \mathcal{R}$ a representation function. The methods we consider all use parametric probes, which are neural networks $\hat{p}_\theta : \mathcal{R} \to P(\mathcal{Y})$ parameterized by $\theta \in \mathbb{R}^d$ that are trained on $D^n$ to estimate the conditional distribution $p(y \mid x)$. We often abstract away the details of learning the probe by simply referring to an algorithm $\mathcal{A}$ which returns a predictor: $\hat{p} = \mathcal{A}(\phi(D^n))$. Abusing notation, we denote the composition of $\mathcal{A}$ with $\phi$ by $\mathcal{A}_\phi$. Define the population loss and the expected population loss for $\hat{p} = \mathcal{A}_\phi(D^n)$, respectively as

$$L(\mathcal{A}_\phi, D^n) = \mathop{\mathbb{E}}_{(\mathbf{X},\mathbf{Y})} -\log \hat{p}(\mathbf{Y} \mid \mathbf{X}) \qquad L(\mathcal{A}_\phi, n) = \mathop{\mathbb{E}}_{\mathbf{D}^n} L(\mathcal{A}_\phi, \mathbf{D}^n). \tag{4.1}$$ {{eq:loss_data}}

The expected population loss averages over evaluation dataset samples, removing the variance that comes from using some particular evaluation dataset to train a probe. In this section we will focus on population quantities, but note that any algorithmic implementation must replace these by their empirical counterparts.

The representation evaluation problem. The representation evaluation problem asks us to define a real-valued measurement of the quality of a representation $\phi$ for solving solving the task defined by $(\mathbf{X}, \mathbf{Y})$. Explicitly, each method defines a real-valued function $m(\phi, \mathcal{D}, \mathcal{A}, \Psi)$ of a representation $\phi$, data distribution $\mathcal{D}$, probing algorithm $\mathcal{A}$, and some method-specific set of hyperparameters $\Psi$. By convention, smaller values of the measure $m$ correspond to better representations. Defining such a measurement allows us to compare different representations.

### 4.2.1 Defining the loss-data framework.

The loss-data framework is a lens through which we contrast different measures of representation quality. The key idea, demonstrated in Figure 4.2, is to plot the loss $L(\mathcal{A}_\phi, n)$ against the evaluation dataset size $n$. Explicitly, at each $n$, we train a probing algorithm $\mathcal{A}$ using a representation $\phi$ to produce a predictor $\hat{p}$, and then plot the loss of $\hat{p}$ against $n$. Similar analysis has appeared in Voita and Titov [2020]; Yogatama et al. [2019]; Talmor et al. [2019]. We can represent each of the prior measures as points on the curve at fixed $x$ (VA, MI) or integrals of the curve along the $x$-axis (MDL). Our measures correspond to evaluating points at fixed $y$ ($\varepsilon$SC) and integrals along the $y$-axis (SDL).

### 4.2.2 Existing methods in the loss-data framework

{sec:existing_methods

Nonlinear probes with limited data. A simple strategy for evaluating representations is to choose a probe architecture and train it on a limited amount of data from the task and representation of interest [Hénaff et al. 2020; Zhang and Bowman 2018]. Each representation is typically scored by its validation accuracy, leading us to call this the validation accuracy (VA) measure. This method can be interpreted in our framework by replacing the validation accuracy with the validation loss and taking an expectation over draws of evaluation datasets of size $n$. On the

loss-data curve, this measure corresponds to evaluation at $x = n$, so that

$$m_{\text{VA}}(\phi, \mathcal{D}, \mathcal{A}, n) = L(\mathcal{A}_\phi, n). \tag{4.2}$$

Mutual information. Mutual information (MI) between a representation $\phi(\mathbf{X})$ and targets $\mathbf{Y}$ is another often-proposed metric for learning and evaluating representations [Pimentel et al. 2020; Bachman et al. 2019]. In terms of entropy, mutual information is equivalent to the information gain about $\mathbf{Y}$ from knowing $\phi(\mathbf{X})$:

$$I(\phi(\mathbf{X}); \mathbf{Y}) = H(\mathbf{Y}) - H(\mathbf{Y} \mid \phi(\mathbf{X})). \tag{4.3}$$

In general mutual information is intractable to estimate for high-dimensional or continuous-valued variables [McAllester and Stratos 2020], and a common approach is to use a very expressive model for $\hat{p}$ and maximize a variational lower bound:

$$I(\phi(\mathbf{X}); \mathbf{Y}) \geqslant H(\mathbf{Y}) + \mathop{\mathbb{E}}_{(\mathbf{X},\mathbf{Y})} \log \hat{p}(\mathbf{Y} \mid \phi(\mathbf{X})). \tag{4.4}$$

Since $H(\mathbf{Y})$ is not a function of the parameters, maximizing the lower bound is equivalent to minimizing the negative log-likelihood. Moreover, if we assume that $\hat{p}$ is expressive enough to represent $p$ and take $n \to \infty$, this inequality becomes tight. As such, MI estimation can be seen a special case of nonlinear probes as described above, where instead of choosing some particular setting of $n$ we push it to infinity. We formally define the mutual information measure of a representation as

$$m_{\text{MI}}(\phi, \mathcal{D}, \mathcal{A}) = \lim_{n \to \infty} L(\mathcal{A}_\phi, n). \tag{4.5}$$

A decrease in this measure reflects an increase in the mutual information. On the loss-data curve, this corresponds to evaluation at $x = \infty$.

Minimum description length. Recent studies [Yogatama et al. 2019; Voita and Titov 2020] propose using the Minimum Description Length (MDL) principle [Rissanen 1978; Grünwald 2004] to evaluate representations. These works use an online or prequential code [Blier and Ollivier 2018] to encode the labels given the representations. The codelength $\ell$ of $Y^n$ given $\phi(X^n)$ is then defined as

$$\ell(Y^n \mid \phi(X^n)) = -\sum_{i=1}^{n} \log \hat{p}_i(y_i \mid \phi(x_i)), \tag{4.6}$$

where $\hat{p}_i$ is the output of running a pre-specified algorithm $\mathcal{A}$ on the evaluation dataset up to element $i$: $\hat{p}_i = \mathcal{A}_\phi(X^n_{1:i}, Y^n_{1:i})$. This measure can exhibit large variance on small evaluation datasets, especially since it is sensitive to the (random) order in which the examples are presented. We remove this variance by taking an expectation over the sampled evaluation datasets for each $i$ and define a population variant of the MDL measure [Voita and Titov 2020] as

$$m_{\text{MDL}}(\phi, \mathcal{D}, \mathcal{A}, n) = \mathbb{E}\left[\ell(\mathbf{Y}^n \mid \phi(\mathbf{X}^n))\right] = \sum_{i=1}^{n} L(\mathcal{A}, i). \tag{4.7}$$ {{eq:mdl_expected}}

Thus, $m_{\text{MDL}}$ measures the area under the loss-data curve on the interval $x \in [0, n]$.

## 4.3 Limitations of existing methods

Each of the prior methods, VA, MDL, and MI, have limitations that we attempt to solve with our methods. In this section we present these limitations.

### 4.3.1 Sensitivity to evaluation set size in VA and MDL

As seen in Section 4.2.2, the representation quality measures of VA and MDL both depend on $n$, the size of the evaluation dataset. Because of this dependence, the ranking of representations given by these evaluation metrics can change as $n$ increases. Choosing to deploy one representation rather than another by comparing these metrics at arbitrary $n$ may lead to premature decisions in the machine learning pipeline since a larger evaluation dataset could give a different ordering.

{sec:counter_ex}

A theoretical example. Let $s \in \{0,1\}^d$ be a fixed binary vector and consider a data generation process where the $\{0,1\}$ label of a data point is given by the parity on $s$, i.e., $y_i = \langle x_i, s \rangle \bmod 2$ where $y_i \in \{0,1\}$ and $x_i \in \{0,1\}^d$. Let $Y^n = \{y_i\}_{i=1}^n$ be the given labels and consider the following two representations: (1) Noisy label: $z_i = \langle x_i, s \rangle + e_i \bmod 2$, where $e_i \in \{0,1\}$ is a random bit with bias $\alpha < 1/2$, and (2) Raw data: $x_i$.

For the noisy label representation, guessing $y_i = z_i$ achieves validation accuracy of $1 - \alpha$ for any $n$, which, is information-theoretically optimal. On the other hand, the raw data representation will achieve perfect validation accuracy once the evaluation dataset contains $d$ linearly independent $x_i$'s. In this case, Gaussian elimination will exactly recover $s$. The probability that a set of $n > d$ random vectors in $\{0,1\}^d$ does not contain $d$ linearly independent vectors decreases exponentially in $n - d$. Hence, the expected validation accuracy for $n$ sufficiently larger than $d$ will be exponentially close to 1. As a result, the representation ranking given by validation accuracy and description length favors the noisy label representation when $n \ll d$, but the raw data representation will be much better in these metrics when $n \gg d$. This can be misleading. Although this is a concocted example for illustration purposes, our experiments in Section 4.5 validate that dependence of representation rankings on $n$ does occur in practice.

### 4.3.2 Insensitivity to representation quality & computational complexity in MI

{sec:mutual_info}

MI considers the lowest validation loss achievable with the given representation and ignores any concerns about statistical or computational complexity of achieving such accuracy. This leads to some counterintuitive properties which make MI an undesirable metric:

1. MI is insensitive to statistical complexity. Two random variables which are perfectly predictive of one another have maximal MI, though their relationship may be sufficiently complex that it requires exponentially many samples to verify [McAllester and Stratos 2020].

2. MI is insensitive to computational complexity. For example, the mutual information between an intercepted encrypted message and the enemy's plan is high [Shannon 1948; Xu et al. 2020], despite the extreme computational cost required to break the encryption.

3. MI is insensitive to representation. By the data processing inequality [Cover and Thomas 2006], *any* $\phi$ applied to $\mathbf{X}$ can only decrease its mutual information with $\mathbf{Y}$; no matter the query, MI always reports that the raw data is at least as good as the best representation.

### 4.3.3 Lack of a predefined notion of success

All three prior methods lack a predefined notion of successfully solving a task and will always return some ordering of representations. When the evaluation dataset is too small or all of the representations are poor, it may be that no representation can yet solve the task (i.e. achieve a useful accuracy). Since the order of representations can change as more data is added, any judgement would be premature. Indeed, there is often an implicit minimum requirement for the loss a representation should achieve to be considered meaningful. As we show in the next section, our methods makes this requirement explicit.

## 4.4 Surplus description length & $\varepsilon$ sample complexity

The methods discussed above measure a property of the data, such as the attainable accuracy on $n$ points, by learning an unspecified function. Instead, we propose to precisely define the function of interest and measure its complexity using data. Fundamentally, we shift from making a statement about the inputs of an algorithm, like VA and MDL do, to a statement about the outputs.

### 4.4.1 Surplus description length (SDL)

Imagine trying to efficiently encode a large number of samples of a random variable $\mathbf{e}$ which takes values in $\{1 \dots K\}$ with probability $p(\mathbf{e})$. An optimal code for these events has expected length[8] $\mathbb{E}[\ell(\mathbf{e})] = \mathbb{E}_{\mathbf{e}}[-\log p(\mathbf{e})] = H(\mathbf{e})$. If this data is instead encoded using a probability distribution $\hat{p}$, the expected length becomes $H(\mathbf{e}) + D_{\mathrm{KL}}\big(p \mid\mid \hat{p}\big)$. We call $D_{\mathrm{KL}}\big(p \mid\mid \hat{p}\big)$ the *surplus description length* (SDL) from encoding according to $\hat{p}$ instead of $p$:

$$D_{\mathrm{KL}}\big(p \mid\mid \hat{p}\big) = \mathop{\mathbb{E}}_{\mathbf{e}\sim p}\left[\log p(\mathbf{e}) - \log \hat{p}(\mathbf{e})\right]. \tag{4.8}$$

When the true distribution $p$ is a delta, the entire length of a code under $\hat{p}$ is surplus since $\log 1 = 0$.

Recall that the prequential code for estimating MDL computes the description length of the labels given observations in an evaluation dataset by iteratively creating tighter approximations $\hat{p}_1 \dots \hat{p}_n$ and integrating the area under the curve. Examining Equation (4.7), we see that

$$m_{\mathrm{MDL}}(\phi, \mathcal{D}, \mathcal{A}, n) = \sum_{i=1}^{n} L(\mathcal{A}_\phi, i) \geq \sum_{i=1}^{n} H(\mathbf{Y} \mid \phi(\mathbf{X})). \tag{4.9}$$

If $H(\mathbf{Y} \mid \phi(\mathbf{X})) > 0$, MDL grows without bound as the size of the evaluation dataset $n$ increases.

Instead, we propose to measure the complexity of a learned predictor $p(\mathbf{Y} \mid \phi(\mathbf{X}))$ by computing the surplus description length of encoding an infinite stream of data according to the online code instead of the true conditional distribution.

{def:sdl_entropy}

**Definition 4.1** (Surplus description length of online codes)**.** Given random variables $\mathbf{X}, \mathbf{Y} \sim \mathcal{D}$, a representation function $\phi$, and a learning algorithm $\mathcal{A}$, define

$$m_{\mathrm{SDL}}(\phi, \mathcal{D}, \mathcal{A}) = \sum_{i=1}^{\infty} \Big[ L(\mathcal{A}_\phi, i) - H(\mathbf{Y} \mid \mathbf{X}) \Big]. \tag{4.10}$$

This surplus description length beyond the optimal code improves on MDL by being bounded. However, as discussed in Section 4.3.2 entropy is intractible to estimate, and in practice it is more relevant to measure the cost of learning a good enough predictor rather than a theoretically perfect one.

We generalize this definition to measure the complexity of learning an approximating conditional distribution with loss $\varepsilon$. This corresponds to the additional description length incurred by encoding data with the learning algorithm $\mathcal{A}$ rather than using a fixed predictor with loss $\varepsilon$.

{def:sdl}

**Definition 4.2** (Surplus description length of online codes with a specified baseline)**.** Take random variables $\mathbf{X}, \mathbf{Y} \sim \mathcal{D}$, a representation function $\phi$, a learning algorithm $\mathcal{A}$, and a loss tolerance $\varepsilon \geqslant H(\mathbf{Y} \mid \mathbf{X})$. Let $[c]_+$ denote $\max(0, c)$ and then we define

$$m_{\mathrm{SDL}}(\phi, \mathcal{D}, \mathcal{A}, \varepsilon) = \sum_{i=1}^{\infty} \Big[ L(\mathcal{A}_\phi, i) - \varepsilon \Big]_+. \tag{4.11}$$

One interpretation of this measure is that it gives the cost (in terms of information) for recreating an $\varepsilon$-loss predictor when using the representation $\phi$.

In our framework, the surplus description length corresponds to computing the area between the loss-data curve and a baseline set by $y = \varepsilon$. Whereas MDL measures the complexity of a sample of $n$ points, SDL measures the complexity of a function which solves the task to $\varepsilon$ tolerance.

Estimating the SDL. Naively computing SDL would require unbounded data and the estimation of $L(\mathcal{A}_\phi, i)$ for every $i$. However, any reasonable learning algorithm obtains a better-generalizing predictor when given more i.i.d. data from the target distribution [Kaplan et al. 2020]. If we assume that algorithms are monotonically improving so that $L(\mathcal{A}, i+1) \leqslant L(\mathcal{A}, i)$, SDL only depends on $i$ up to the first point where $L(\mathcal{A}, n) \leqslant \varepsilon$. Approximating this integral can be done efficiently by taking a log-uniform partition of the evaluation dataset size and computing the Riemann sum as in Voita and Titov [2020]. Note that evaluating a representation only requires training probes, not the large representation functions themselves, and thus has modest computational requirements. Crucially, if the tolerance $\varepsilon$ is set unrealizeably low or the amount of available data is insufficient, an implementation is able to report that the given complexity estimate is only a lower bound.

### 4.4.2 $\varepsilon$ sample complexity ($\varepsilon$SC)

In addition to surplus description length we introduce a second, conceptually simpler measure of representation quality: $\varepsilon$ sample complexity.

{def:sample_complexit

**Definition 4.3** (Sample complexity of an $\varepsilon$-loss predictor)**.** Given random variables $\mathbf{X}, \mathbf{Y} \sim \mathcal{D}$, a representation function $\phi$, a learning algorithm $\mathcal{A}$, and a loss tolerance $\varepsilon \geqslant H(\mathbf{Y} \mid \phi(\mathbf{X}))$, define

$$m_{\varepsilon\text{SC}}(\phi, \mathcal{D}, \mathcal{A}, \varepsilon) = \min \left\{ n \in \mathbb{N} : L(\mathcal{A}_\phi, n) \leq \varepsilon \right\}. \tag{4.12}$$

The $\varepsilon$ sample complexity measures the complexity of learning an $\varepsilon$-loss predictor by the number of samples it takes to find it. This measure allows the comparison of two representations by first picking a target function to learn (via a setting of $\varepsilon$), then measuring which representation enables learning that function with less data.

In our framework, sample complexity corresponds to taking a horizontal slice of the loss-data curve at $y = \varepsilon$, analogous to VA's slice at $y = n$. VA makes a statement about the data (by setting

$n$) and reports the accuracy of some function given that data. In contrast, $\varepsilon$ sample complexity specifies the desired function and determines its complexity by how many samples are needed to learn it.

Estimating the $\varepsilon$SC. Given an assumption that algorithms are monotonically improving such that $L(\mathcal{A}, n+1) \leqslant L(\mathcal{A}, n)$, $\varepsilon$SC can be estimated efficiently. With $n$ finite samples in the evaluation dataset, an algorithm may estimate $\varepsilon$SC by splitting the data into $k$ uniform-sized bins and estimating $L(\mathcal{A}, {}^{ik}/_{n})$ for $i \in \{1 \dots k\}$. By recursively performing this search on the interval which contains the transition from $L > \varepsilon$ to $L < \varepsilon$, we can rapidly reach a precise estimate or report that $m_{\varepsilon\text{SC}}(\phi, \mathcal{D}, \mathcal{A}, \varepsilon) > n$.

Using objectives other than negative log-likelihood. Our exposition of $\varepsilon$SC uses negative log-likelihood for consistency with other methods, such as MDL, which require it. However, it is straightforward to extend $\varepsilon$SC to work with whatever objective function is desired under the assumption that said objective is monotone with increasing data when using algorithm $\mathcal{A}$. A natural choice in many cases would be prediction accuracy, where a practitioner might target e.g. a 95% accurate predictor.

### 4.4.3 Setting $\varepsilon$

A value for the threshold $\varepsilon$ corresponds to the set of $\varepsilon$-loss predictors that a representation should make easy to learn. Choices of $\varepsilon \geqslant H(\mathbf{Y} \mid \mathbf{X})$ represent attainable functions, while selecting $\varepsilon < H(\mathbf{Y} \mid \mathbf{X})$ leads to unbounded SDL and $\varepsilon$SC for any choice of the algorithm $\mathcal{A}$.

For evaluating representation learning methods in the research community, we recommend using SDL and establishing benchmarks which specify (1) a downstream task, in the form of an evaluation dataset; (2) a criterion for success, in the form of a setting of $\varepsilon$; (3) a standard probing algorithm $\mathcal{A}$. The setting of $\varepsilon$ can be done by training a large model on the raw representation of

the full evaluation dataset and using its validation loss as $\varepsilon$ when evaluating other representations. This guarantees that $\varepsilon \geqslant H(\mathbf{Y} \mid \mathbf{X})$ and the task is feasible with any representation at least as good as the raw data. In turn, this ensures that SDL is bounded.

In practical applications, $\varepsilon$ should be a part of the design specification for a system. As an example, a practitioner might know that an object detection system with 80% per-frame accuracy is sufficient and labels are expensive. For this task, the best representation would be one which enables the most sample efficient learning of a predictor with error $\varepsilon = 0.2$ using a $0-1$ loss.

## 4.5 Experiments

{sec:experiments}

We empirically show the behavior of VA, MDL, SDL, and $\varepsilon$SC with two sets of experiments on real data. These experiments have the following goals:

1. Test whether the theoretical issue of sensitivity to evaluation dataset size for VA and MDL occurs in practice.

2. Demonstrate that SDL and $\varepsilon$SC produce lower bounds when insufficient data is available and concrete quantities otherwise.

3. Evaluate whether the computation of SDL and $\varepsilon$SC scales to large-scale tasks.

### 4.5.1 Tasks and representations

For the first experiment, shown in Figure 4.3 and Table 4.1, we use the small-scale task of MNIST classification. We evaluate three representations: (1) the last hidden layer of a small convolutional network pretrained on CIFAR-10; (2) the raw pixels; and (3) the bottleneck of a variational autoencoder (VAE) [Kingma and Welling 2014; Rezende et al. 2014a] trained on MNIST.

For the second experiment, shown in Figure 4.4 and Table 4.2, we compare the representations given by different layers of a pretrained ELMo model [Peters et al. 2018]. We use the part-of-

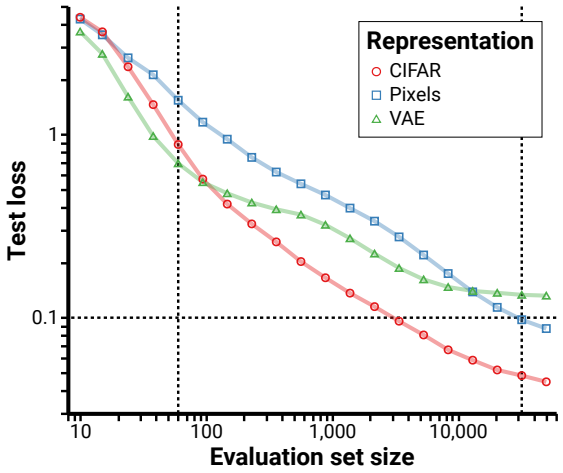



**Figure 4.3:** Results using three representations on MNIST. The intersections between curves indicate evaluation dataset sizes where VA would change its ranking of these representations. Curves are estimated using eight bootstrap-sampled evaluation datasets and initializations at each point to ensure the measured quantities are close to the expectation.

{fig:multi_mnist}

speech task introduced by Hewitt and Liang [2019] and implemented by Voita and Titov [2020] with the same probe architecture and other hyperparameters as those works. This leads to a large-scale representation evaluation task, with 4096-dimensional representation vectors and an output space of size $48^k$ for a sentence of $k$ words.

In each set of experiments we compute loss-data curves by estimating the expected population loss at each evaluation dataset size using a bootstrapped sample from the full evaluation dataset, reducing the variance of the results. Note that in each experiment we omit MI as for a finite evaluation dataset, the MI measure is the same as validation loss. Details of the experiments, including representation training, probe architectures, and hyperparameters, are available in Appendix 4.A.

### 4.5.2 Results

These experiments demonstrate that the issue of sensitivity to evaluation dataset size in fact occurs in practice, both on small problems (Table 4.1) and at scale (Table 4.2): VA and MDL choose different representations when given evaluation sets of different sizes. Because these measures

| n | Representation | CIFAR | Pixels | VAE |
|---|---|---|---|---|
| 60 | VA | 0.88 | 1.54 | **0.70** |
| | MDL | 122.75 | 147.34 | **93.8** |
| | SDL, $\varepsilon$=0.1 | > 116.75 | > 141.34 | > 87.8 |
| | $\varepsilon$SC, $\varepsilon$=0.1 | > 60.0 | > 60.0 | > 60.0 |
| 31936 | VA | **0.05** | 0.10 | 0.13 |
| | MDL | **2165.1** | 5001.57 | 4898.37 |
| | SDL, $\varepsilon$=0.1 | **260.6** | 1837.08 | > 1704.77 |
| | $\varepsilon$SC, $\varepsilon$=0.1 | **3395** | 31936 | > 31936.0 |

**Table 4.1:** Estimated measures of representation quality on MNIST. At small evaluation dataset sizes, VA and MDL state that the VAE representation is the best, even though every representation yields poor prediction quality with that amount of data. Since SDL and $\varepsilon$SC have a target for prediction quality, they are able to report when the evaluation dataset is insufficient to achieve the desired performance. {tab:multi_mnist}

are a function of the evaluation dataset size, making a decision about which representation to use with a small evaluation dataset would be premature.

By contrast, SDL and $\varepsilon$SC are functions only of the data *distribution*, not a finite sample. Once they measure the complexity of learning an $\varepsilon$-loss function, that measure is invariant to the size of the evaluation dataset. Crucially, since these measures contain a notion of success in solving a task, they are able to avoid the issue of premature evaluation and notify the user if there is insufficient data to evaluate and return a lower bound instead.

The part of speech experiment in Figure 4.4 and Table 4.2 demonstrates that SDL and $\varepsilon$SC can scale to tasks of a practically relevant size. This experiment is of a similar size to the widespread use of BERT [Devlin et al. 2019] or SimCLR [Chen et al. 2020a], and evaluating our measures to high precision took about an hour on one GPU.

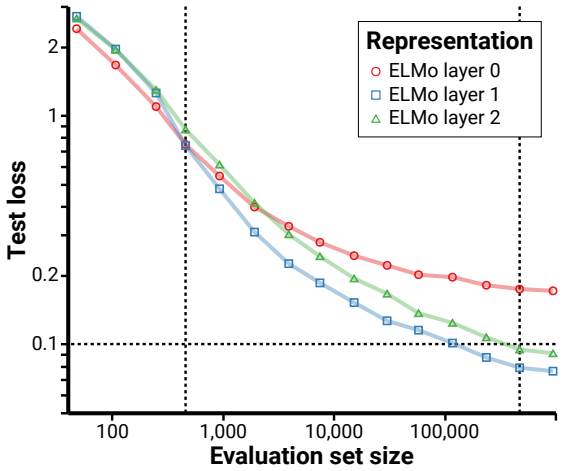



**Figure 4.4:** Results using three representations on the part of speech classification task. Loss-data curves are estimated using four bootstrap-sampled evaluation datasets and network initializations at each point. {fig:elmo_layers}

## 4.6 Related work

Representation evaluation methods. Until recently, the standard technique for evaluating representation quality was the use of linear probes [Kiros et al. 2015; Hill et al. 2016; van den Oord et al. 2018; Chen et al. 2020a]. However, Hénaff et al. [2020] find that evaluation with linear probes is largely uncorrelated with the more practically relevant objective of low-data accuracy, and Resnick et al. [2019] show that linear probe performance does not predict performance for transfer across tasks. Beyond linear probes, Zhang and Bowman [2018] and Hewitt and Liang [2019] show that restrictions on model capacity or evaluation dataset size are necessary to separate the performance of randomly- and linguistically-pretrained representations. Voita and Titov [2020] propose using the MDL framework, which measures the description length of the labels given the observations. An earlier work by Yogatama et al. [2019] also uses prequential codes to evaluate representations for linguistic tasks. Talmor et al. [2019] look at the loss-data curve (called "learning curve" in their work) and use a weighted average of the validation loss at various training set sizes to evaluate representations.

| n | ELMo layer | 0 | 1 | 2 |
|---|---|---|---|---|
| 461 | VA | 0.75 | **0.74** | 0.87 |
| | MDL | **884.54** | 1009.26 | 1017.72 |
| | SDL, $\varepsilon$=0.1 | > 478.67 | > 528.51 | > 561.7 |
| | $\varepsilon$SC, $\varepsilon$=0.1 | > 461 | > 461 | > 461 |
| 474838 | VA | 0.17 | **0.08** | 0.09 |
| | MDL | 92403.41 | **52648.50** | 65468.54 |
| | SDL, $\varepsilon$=0.1 | > 40882.72 | **2765.11** | 7069.56 |
| | $\varepsilon$SC, $\varepsilon$=0.1 | > 474838 | **237967** | 474838 |

**Table 4.2:** Estimated measures of representation quality on the part of speech classification task. With small evaluation datasets, MDL finds that the lowest ELMo layer gives the best results, but when the evaluation dataset grows the outcome changes. {tab:elmo_layers}

Foundational work. A fundamental paper by Blier and Ollivier [2018] introduces prequential codes as a measure of the complexity of a deep learning model. Xu et al. [2020] introduce predictive $\mathcal{V}$-information, a theoretical generalization of mutual information which takes into account computational constraints, and is essentially the mutual information lower bound often reported in practice. Work by Dubois et al. [2020] describe representations which, in combination with a specified family of predictive functions, have guarantees on their generalization performance.

## 4.7 Discussion

In this work, we have introduced the loss-data framework for comparing representation evaluation measures and used it to diagnose the issue of sensitivity to evaluation dataset size in the validation accuracy and minimum description length measures. We proposed two measures, surplus description length and $\varepsilon$ sample complexity, which eliminate this issue by measuring the complexity of learning a predictor which solves the task of interest to $\varepsilon$ tolerance. Empirically, we showed that sensitivity to evaluation dataset size occurs in practice for VA and MDL, while SDL and $\varepsilon$SC are robust to the amount of available data and are able to report when it is insuffi-

cient to make a judgment.

Each of these measures depends on a choice of algorithm $\mathcal{A}$, including hyperparameters such as probe architecture, which could make the evaluation procedure less robust. To alleviate this, future work might consider a set of algorithms $A = \{\mathcal{A}_i\}_{i=1}^{K}$ and a method of combining them, such as the model switching technique of Blier and Ollivier [2018]; Erven et al. [2012] or a Bayesian prior.

Finally, while existing measures such as VA, MI, and MDL do not measure our notion of the best representation for a task, under other settings they may be the correct choice. For example, if only a fixed set of data will ever be available, selecting representations using VA might be a reasonable choice; and if unbounded data is available for free, perhaps MI is the most appropriate measure. However, in many cases the robustness and interpretability offered by SDL and $\varepsilon$SC make them a practical choice for practitioners and representation researchers alike.

# Appendix 4.A Experimental details

{sec:experiment_detai

In each experiment we first estimate the loss-data curve using a fixed number of dataset sizes $n$ and multiple random seeds, then compute each measure from that curve. Reported values of SDL correspond to the estimated area between the loss-data curve and the line $y = \varepsilon$ using Riemann sums with the values taken from the left edge of the interval. This is the same as the chunking procedure of Voita and Titov [2020] and is equivalent to the code length of transmitting each chunk of data using a fixed model and switching models between intervals. Reported values of $\varepsilon$SC correspond to the first measured $n$ at which the loss is less than $\varepsilon$.

All of the experiments were performed on a single server with 4 NVidia Titan X GPUs, and on this hardware no experiment took longer than an hour. All of the code for our experiments, as well as that used to generate our plots and tables, is included in the supplement.

## 4.A.1 MNIST experiments

For our experiments on MNIST, we implement a highly-performant vectorized library in JAX to construct loss-data curves. With this implementation it takes about one minute to estimate the loss-data curve with one sample at each of 20 settings of $n$. We approximate the loss-data curves at 20 settings of $n$ log-uniformly spaced on the interval $[10, 50000]$ and evaluate loss on the test set to approximate the population loss. At each dataset size $n$ we perform the same number of updates to the model; we experimented with early stopping for smaller $n$ but found that it made no difference on this dataset. In order to obtain lower-variance estimates of the expected risk at each $n$, we run 8 random seeds for each representation at each dataset size, where each random seed corresponds to a random initialization of the probe network and a random subsample of the evaluation dataset.

Probes consist of two-hidden-layer MLPs with hidden dimension 512 and ReLU activations. All probes and representations are trained with the Adam optimizer [Kingma and Ba 2015] with

learning rate $10^{-4}$.

Each representation is normalized to have zero mean and unit variance before probing to ensure that differences in scaling and centering do not disrupt learning. The representations of the data we evaluate are implemented as follows.

Raw pixels. The raw MNIST pixels are provided by the Pytorch `datasets` library [Paszke et al. 2019a]. It has dimension $28 \times 28 = 784$.

CIFAR. The CIFAR representation is given by the last hidden layer of a convolutional neural network trained on the CIFAR-10 dataset. This representation has dimension 784 to match the size of the raw pixels. The network architecture is as follows:

```
nn.Conv2d(1, 32, 3, 1),
nn.ReLU(),
nn.MaxPool2d(2),
nn.Conv2d(32, 64, 3, 1),
nn.ReLU(),
nn.MaxPool2d(2),
nn.Flatten(),
nn.Linear(1600, 784)
nn.ReLU()
nn.Linear(784, 10)
nn.LogSoftmax()
```

VAE. The VAE (variational autoencoder; Kingma and Welling [2014]; Rezende et al. [2014a]) representation is given by a variational autoencoder trained to generate the MNIST digits. This VAE's latent variable has dimension 8. We use the mean output of the encoder as the representation of the data. The network architecture is as follows:

```
self.encoder_layers = nn.Sequential(
    nn.Linear(784, 400),
    nn.ReLU(),
    nn.Linear(400, 400),
    nn.ReLU(),
    nn.Linear(400, 400),
    nn.ReLU(),
)
self.mean = nn.Linear(400, 8)
self.variance = nn.Linear(400, 8)

self.decoder_layers = nn.Sequential(
    nn.Linear(8, 400),
    nn.ReLU(),
    nn.Linear(400, 400),
    nn.ReLU(),
    nn.Linear(400, 784),
)
```

### 4.A.2 Part of speech experiments

We follow the methodology and use the official code[9] of Voita and Titov [2020] for our part of speech experiments using ELMo [Peters et al. 2018] pretrained representations. In order to obtain lower-variance estimates of the expected risk at each $n$, we run 4 random seeds for each representation at each dataset size, where each random seed corresponds to a random initialization of the probe network and a random subsample of the evaluation dataset. We approximate the loss-data curves at 10 settings of $n$ log-uniformly spaced on the range of the available data $n \in [10, 10^6]$.

To more precisely estimate $\varepsilon$SC, we perform one recursive grid search step: we space 10 settings over the range which in the first round saw $L(\mathcal{A}_\phi, n)$ transition from above to below $\varepsilon$.

Probes consist of the MLP-2 model of Hewitt and Liang [2019]; Voita and Titov [2020] and all training parameters are the same as in those works.

# Notes

8. Code length in nats due to the base $e$.

9. Code available at https://github.com/lena-voita/description-length-probing.

# 5 | Dynamics-Aware Embeddings

{sec:dyne}

## 5.1 Introduction

In recent years, there has been a lot of excitement around end-to-end model-free reinforcement learning for control, both in simulation [Lillicrap et al. 2015; Andrychowicz et al. 2018; Haarnoja et al. 2018b; Fujimoto et al. 2018c] and on real hardware [Kalashnikov et al. 2018; Haarnoja et al. 2018c]. In this paradigm, we simultaneously learn intermediate representations and policies by maximizing rewards provided by environment. End-to-end learning has one indisputable advantage: since every component of the system is optimized for the end objective, there are no sub-optimal modules that limit best-case performance by losing task-relevant information.

Learning only from the target task is however a double-edged sword. When the end objective provides only weak signal for learning, a policy with a poor representation may require many samples to learn a better one. By contrast, a policy with a good representation may be able to rapidly fit a simple function of that representation even with weak signal.

Consider the environment shown in Figure 5.1, and two representations of its state: coordinates and pixels. As a function of the agent's $x$ coordinate, the value function is simple and smooth. The coordinate representation has structure which is useful for learning about the task; namely, points which are close in $L^2$ distance have similar values. By contrast, a pixel representation of the agent's state (below, blue) is practically a one-hot vector. Two states whose $x$ coordinates differ by one unit have pixels exactly as different as states which differ by 100 units.

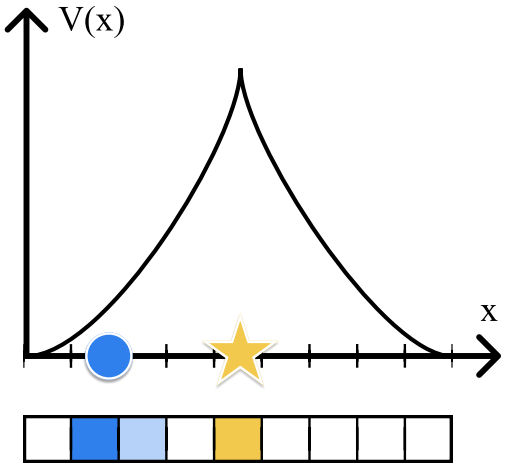



**Figure 5.1:** A 1D environment. The agent (blue dot) can move continuously left and right to reach the goal (gold star).

{fig:pixel_illustrati

This illustrates the importance of good representations and the potential of representation learning to aid RL.

We propose a self-supervised objective for learning embeddings of states and action sequences such that a pair of states or action sequences will be close together if they have similar outcomes. This objective simultaneously trains a smooth embedding space for states and a temporally abstract action space for control which is task-independent and generalizes across goals and objects.

We demonstrate the effectiveness of our representation learning objective by training the twin delayed deep deterministic policy gradient algorithm (TD3) [Fujimoto et al. 2018c] with learned action and state spaces. With a learned representation of temporally abstract actions, our method exhibits improved sample efficiency compared to state-of-the-art RL methods on control tasks, with larger gains on more complex environments. When additionally combined with our learned state representation, our method allows TD3 to scale to pixel observations. We demonstrate good performance on a simple family of goal-conditioned 2D control tasks within a few million environment steps without adjusting any TD3 hyperparameters. This stands in contrast to end-to-end model-free RL from pixels, which requires extensive tuning [Lillicrap et al. 2015] and on the order of 100 million environment steps[10] [Barth-Maron et al. 2018].

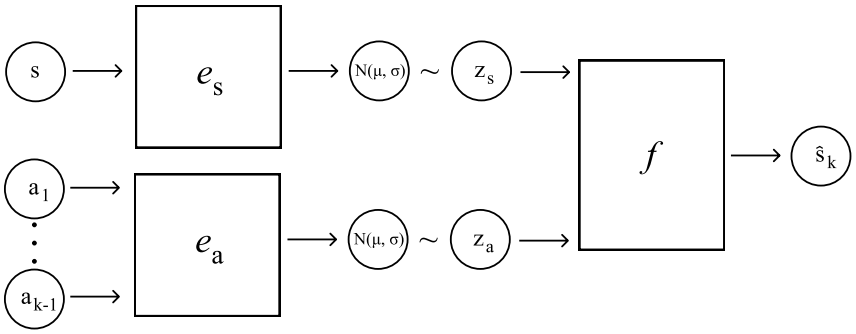



**Figure 5.2:** Computational architecture for training the DynE encoders $e_a$ and $e_s$. The encoders are trained to minimize the information content of the learned embeddings while still allowing the predictor $f$ to make accurate predictions.

{fig:model}

## 5.2 Dynamics-aware embeddings

### 5.2.1 Notation

We consider the framework of reinforcement learning in Markov decision processes (MDPs).[11] We denote the state of an environment (e.g. joint angles of a robot or pixels) by $s \in \mathcal{S}$, and we assume that the states given by the environment satisfy the Markov property. We refer to a sequence of actions $\{a_1, \ldots, a_k\} \in \mathcal{A}^k$ using the shorthand $\boldsymbol{a}^k$. We use $s' \sim \mathbf{T}(s, a)$ to refer to the environment's (stochastic) transition function, and overload it to accept sequences of actions: $s_{t+k} \sim \mathbf{T}(s_t, \boldsymbol{a}_t^k)$.

### 5.2.2 Model and learning objective

We propose that a good representation for reinforcement learning should represent states or actions close together if they have similar outcomes (resulting trajectories). This allows the agent to generalize from a small number of samples since each sample accurately reflects the value of all the states or actions in its neighborhood. In a Markov decision process the outcome of taking an action $a$ in a state $s$ is summarized by the distribution of resulting states $p(s'|s, a) = \mathbf{T}(s, a)$.

Therefore we construct a method which embeds states and actions such that nearby embeddings have similar distributions of next states.

Our method, which we call Dynamics-aware Embedding (DynE), learns encoders $e_s$ and $e_a$ which embed a state and action sequence into latent spaces $z_s \in \mathcal{Z}_s$ and $z_a \in \mathcal{Z}_a$ respectively. These encodings are optimized to form a maximally compressed representation of the sufficient statistics of $p(s'|s, \boldsymbol{a}^k)$ such that $p(s'|s, \boldsymbol{a}^k) \approx p(s'|z_s, z_a)$. We approximate this by maximizing the following objective:

$$\mathcal{L}(\phi_s, \phi_a, \theta) = \mathop{\mathbb{E}}_{s, \boldsymbol{a}^k, s' \sim \rho^\pi} \Big[ - \log p(s'|z_s, z_a; \theta) \qquad \text{predict } s' \qquad (5.1) \quad \texttt{\{\{eq:logprob\}\}}$$

$$+ \beta D_{\mathrm{KL}}\big(e_s(s; \phi_s) \;||\; \mathcal{N}(0, \boldsymbol{I})\big) \qquad \text{compress } s \qquad (5.2) \quad \texttt{\{\{eq:skl\}\}}$$

$$+ \gamma D_{\mathrm{KL}}\big(e_a(\boldsymbol{a}^k; \phi_a) \;||\; \mathcal{N}(0, \boldsymbol{I})\big) \Big] \qquad \text{compress } \boldsymbol{a}^k \qquad (5.3) \quad \texttt{\{\{eq:akl\}\}}$$

where $z_s \sim e_s(s)$, $z_a \sim e_a(\boldsymbol{a}^k)$, and $\rho^\pi$ is the distribution of transitions under a behavior policy $\pi$.

The DynE objective is similar to a $\beta$-VAE [Higgins et al. 2017a] for $s'$ but with a different variational family; like a $\beta$-VAE, it forms a variational lower bound on $p(s')$ when $\beta = \gamma = 1$. Where a variational autoencoder [Kingma and Welling 2013; Rezende et al. 2014b] or $\beta$-VAE chooses the variational family to be $\mathcal{Q} = \{q(z|s')\}$, we use a factored latent space $\{z_s, z_a\}$ and independent posterior approximations given the previous state and the action: $\mathcal{Q} = \{(q(z_s|s), q(z_a|\boldsymbol{a}^k))\}$. This factorization yields separate encoders for states and actions where the state encoder's output is valid for any action and vice versa.

The DynE objective can also be interpreted in the information bottleneck (IB) framework [Tishby et al. 2000]. In the IB framework term (5.1) is the prediction objective and terms (5.2) and (5.3) regularize the latent representation to remove all extraneous information. Our construction is nearly identical to the approximate information bottleneck proposed by Alemi et al. [2016], with the main difference being the factorization of the representation into separate state and action components.

In our experiments we use an isotropic Normal distribution for $p(s'|z_s, z_a; \theta)$ such that term (5.1) reduces to $\|f(z_s, z_a; \theta) - s'\|_2^2$ where $f$ computes the mean. We use diagonal-covariance Normal distributions for $e_s$ and $e_a$ such that $\{\mu_s, \sigma_s^2\} = e_s(s)$, $\{\mu_a, \sigma_a^2\} = e_a(\boldsymbol{a}^k)$, $z_s \sim \mathcal{N}(\mu_s, \sigma_s^2)$, and $z_a \sim \mathcal{N}(\mu_a, \sigma_a^2)$. The behavior policy we use for data collection is $\pi = Unif(\mathcal{A})$.

## 5.3 Using learned embeddings for reinforcement learning

### 5.3.1 Decoding to raw actions

In order to be useful for RL, the abstract action space produced by the encoder must be decodeable to raw actions in the environment. Since the mapping from action sequences to high-level actions is many-to-one, inverting it is nontrivial. We simplify this ill-posed problem by defining an objective with a single optimum.

Once the action encoder $e_a$ is fully trained, we hold it fixed and train an action decoder $d_a$ to minimize

$$\mathcal{L}(d_a) = \mathop{\mathbb{E}}_{z_a \sim \mathcal{N}(0, I)} \left[ ||e_a(d_a(z_a)) - z_a||_2^2 + \lambda ||d_a(z_a)||_2^2 \right] \tag{5.4}$$ {{eq:decoder}}

The first term of this objective ensures that the action decoder $d$ is a one-sided inverse of $e_a$; that is, $e_a(d_a(z_a)) = z_a$ but $d_a(e_a(a_1, \ldots, a_k)) \neq a_1, \ldots, a_k$. The second term of the loss ensures that $d_a$ is in particular the minimum-norm one-sided inverse of $e_a$ and gives the objective for the output of $d_a$ a single minimum. Out of all the action sequences which have the same outcome, the minimum-norm sequence is desireable as it leads to trajectories which are smooth and consume less energy. We choose $\lambda$ to be small (e.g. $10^{-2}$) to ensure that the reconstruction criterion dominates the optimization.

### 5.3.2 Efficient RL with temporal abstraction

Once equipped with a decoder which maps from high-level actions to sequences of raw actions, we train a high-level policy that solves a task by selecting high-level actions. In this section we extend the deterministic policy gradient [Silver et al. 2014] family of algorithms to work with temporally-extended actions while maintaining off-policy updates and learning from every environment step. This allows our method to achieve superior sample efficiency when working with high-level actions. In particular, we extend the twin delayed deep deterministic policy gradient (TD3) algorithm [Fujimoto et al. 2018c] to work with the DynE representation of actions to form an algorithm we call DynE-TD3.

We first describe why DPG requires modifications to accommodate temporally-abstracted actions. One simple approach to combining DynE with DPG would be to incorporate the $k$-step DynE action space into the environment to form a new MDP. This MDP allows the use of DPG without modification; however, it only emits observations once every $k$ timesteps. As a result, after $N$ steps in the original environment, the deterministic policy $\mu$ and critic function $Q$ can only be trained on $N/k$ observations. This has a substantial impact on sample efficiency when measured in the original environment.

Instead we require an algorithm which can perform updates to the policy $\mu$ and critic $Q$ for every environment step. To do this, we train both $\mu$ and $Q$ in the abstract action space with minor changes to their updates. We distinguish these functions which use DynE actions from their raw equivalents by adding a superscript DynE, i.e. $\mu^{\text{DynE}}$ and $Q^{\text{DynE}}$. We augment the critic function with an additional input, $i$, which represents the number of steps $0 \leqslant i < k$ of the current embedded action $z$ that have already been executed. This forms the DynE-TD3 critic:

$$Q^{\text{DynE}}(e_s(s_t), z_t, i) = \sum_{j=0}^{k-i-1} \left(\gamma^j r_{t+j}\right) + \gamma^{k-i} Q^{\text{DynE}}\Big(e_s(s_{t+k-i}), \mu^{\text{DynE}}(e_s(s_{t+k-i})), 0\Big) \qquad (5.5) \quad \texttt{\{\{eq:critic\}\}}$$

In plain language, the value of being on step $i$ of abstract action $e_t$ is the value of finishing the remaining $(k - i)$ steps of $z_t$ and then continuing on following the policy. This is similar to the idea of $k$-step returns [Sutton and Barto 2018], but with a variable $k$ which depends on the step within the current plan. Whereas $k$-step returns would typically require an off-policy correction such as Retrace [Munos et al. 2016], conditioning on $z_t$ and $i$ determines all $k - i$ actions in the return. In effect, they remain a single action, making the update valid off policy. The DynE critic is trained by minimizing the Bellman error implied by Eq. (5.5).

To update the policy we follow the standard DPG technique of using the gradient of the critic. We modify the algorithm to take into account that $i = 0$ at the time of issuing a new high-level action. The gradient of the return with respect to the policy parameters is then

$$\nabla_\theta J_\pi(\mu_\theta^{\text{DynE}}) \approx \mathop{\mathbb{E}}_{s\sim\rho^\pi}\left[\nabla_\theta \mu_\theta^{\text{DynE}}(e_s(s))\ \nabla_z Q^{\text{DynE}}(e_s(s), z, 0)|_{z=\mu_\theta^{\text{DynE}}(e_s(s))}\right] \tag{5.6}$$

given that data was collected according to a behavior policy $\pi$.

## 5.4 Related work

Successor representations, an inspiration for this work, represent a state by the expected rate of future visits to other states [Dayan 1993; Kulkarni et al. 2016b; Barreto et al. 2017]. Successor representations have been demonstrated to be an effective model of animal and human learning [Momennejad et al. 2017; Stachenfeld et al. 2017]. They are also one of the earliest realizations of the idea of representing each state by its future. Whereas successor representations learn future occupancy maps for a particular policy, we learn an embedding space where states are close together if they have similar outcomes for any policy.

Several papers have proposed using (variational) auto-encoders to learn embeddings for ob-

servations [Lange and Riedmiller 2010; Van Hoof et al. 2016; Higgins et al. 2017b; Caselles-Dupré et al. 2018]; unlike our work, these models operate on a single observation at a time and do not depend on the environment dynamics. Forward prediction has also been used as an auxiliary task to speed RL training [Jaderberg et al. 2016], and Jonschkowski et al. [2017] learn representations which adhere to physical constraints. Ghosh et al. [2018] propose to learn state embeddings using the action distribution of a goal-conditioned policy; however, their technique depends on already having a successful policy. Other work has proposed to use mutual information maximization to learn embeddings which facilitate exploration via intrinsic motivation [Kim et al. 2018].

Similarly to this work, hierarchical reinforcement learning seeks to learn temporal abstractions. These abstractions are variously defined as skills [Florensa et al. 2017; Hausman et al. 2018], options [Sutton et al. 1999; Bacon et al. 2017], or goal-directed sub-policies [Kulkarni et al. 2016a; Vezhnevets et al. 2017]. Most closely related are SeCTAR [Co-Reyes et al. 2018] and HIRO [Nachum et al. 2018]. SeCTAR simultaneously learns a generative model of future states and a low-level policy which can reach those states. HIRO learns a representation of goals such that a high-level policy can induce any action in a low-level policy. Unlike this work, both SeCTAR and HIRO learn state-dependent low-level policies, not action representations. Furthermore SeCTAR assumes the reward function is given ahead of time, and HIRO's off-policy performance depends on an approximate re-labeling of action sequences to train the high-level policy.

Also related are methods which attempt to learn embeddings of single actions to enable efficient learning in very large action spaces [Dulac-Arnold et al. 2015; Chandak et al. 2019]. In particular, Chandak et al. [2019] learns a latent space of actions based on the effects of an action on the environment. However, their latent spaces are for a single action and they do not consider learned state representations. Another related direction is learning embeddings of one or more actions from demonstrations [Tennenholtz and Mannor 2019]; this embedded action space builds in prior knowledge from the demonstrator and can allow faster learning.

## 5.5 Representation Experiments

In this section we empirically investigate how the learned DynE representations reshape the problem of reinforcement learning. First we make a connection between temporal abstraction and exploration, revealing that DynE actions result in better state coverage. Then we probe the relationship between DynE state embeddings and the task value function.

### 5.5.1 Temporal abstraction and exploration

When embedding an action sequence, the DynE objective seeks to preserve information about the outcome of that action sequence (i.e. the change in state), but minimize information about the original action sequence. As shown in Appendix 5.D, this leads to a representation where all action sequences which have similar outcomes embed close together. We propose that this temporally abstract action space, where actions correspond to multi-step outcomes, allows random actions to explore the environment more efficiently.

We empirically validate the exploration benefits of the temporally abstract DynE actions. Figure 5.3 shows that uniformly sampling a DynE action results in a nearly uniform distribution over the states reachable within $k$ steps. Over the course of an entire episode, selecting DynE actions uniformly at random reaches faraway states more often than random exploration with raw actions. Appendix 5.F shows the qualitative difference between random trajectories in the raw and DynE action spaces, and Appendix 5.C studies the impact of varying $k$ on the performance of a learned policy.

### 5.5.2 State representations

The DynE objective compresses states while preserving information about the outcome of taking any action in that state. If this compression is successful, states which have similar outcomes

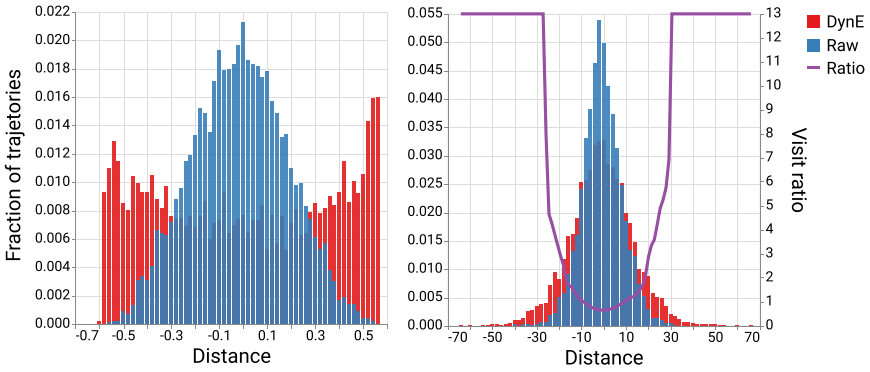



**Figure 5.3:** The distribution of state distances reached by uniform random exploration using DynE actions ($k = 4$) or raw actions in Reacher Vertical. **Left:** Randomly selecting a 4-step DynE action reaches a state uniformly sampled from those reachable in 4 environment timesteps. **Right:** Over the length of an episode (100 steps), random exploration with DynE actions reaches faraway states very much more often than exploration with raw actions. The visit ratio shows how frequently DynE exploration reaches a certain distance compared to raw exploration.

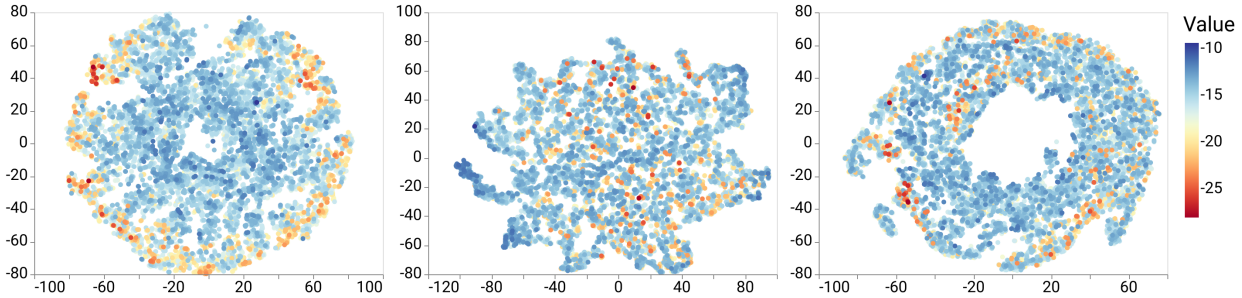
{fig:exploration_hist

will be close together in embedding space. In an MDP, two states which have identical successor states have values which differ by at most the range of the reward function $r_{\max} - r_{\min}$. While in general states which lead to merely similar successors may have arbitrarily different value, we suggest that in many tasks of interest, similar successors may entail similar value.

We investigate whether the DynE state embedding leads to neighborhoods with similar value in the Reacher Vertical environment. We collect 10K states from a random policy in the environment and perform dimensionality reduction on three representations of those states: the DynE embedding of state images, low-dimensional joint states, and pixels. Figure 5.4 shows the results of this dimensionality reduction, in which every point is colored by its value under a fully-trained TD3 policy on the low-d states. DynE embeddings have neighborhoods with more similar values than states or pixels.

## 5.6 Reinforcement learning experiments

In this section we assess the effectiveness of the DynE representations for deep RL, individually analyzing the contributions of the action and state representations before combining them. First




(a) DynE (pixels) (b) States (c) Pixels

**Figure 5.4:** The relationship between state representations and task value. Each plot shows the t-SNE dimensionality reduction of a state representation, where each point is colored by its value under a near-optimal policy. (a) The DynE embedding from pixels places states with similar values close together. (b) The low-dimensional states, which consist of joint angles, relative positions, and velocities, have some neighborhoods of similar value, but also many regions of mixed value. (c) The relationship between the pixel representation and the task value is very complex. {fig:state_tsne}

we evaluate the DynE action space on a set of six tasks with low-dimensional state observations, testing its usefulness across a set of tasks and object interactions. Then, we test the DynE state space on a set of three tasks with pixel observations. Finally, we combine DynE actions with DynE observations, verifying that the two learned representations are complementary.

Appendix 5.B provides a full description of hyperparameters and model architectures, and all of the code for DynE is available on GitHub at https://github.com/dyne-submission/dynamics-aware-embeddings.

Environments We use six continuous control tasks from two families implemented in the MuJoCo simulator [Todorov et al. 2012] to evaluate our method. Within each family, the task and observation space change but the robot being controlled stays roughly the same, allowing us to test the transferrability of the DynE action space between tasks. The Reacher family consists of three of tasks which involve controlling a 2D, 2DoF arm to interact with various objects. The 7DoF family of tasks from OpenAI Gym [Brockman et al. 2016a] is quite difficult, featuring three tasks in which a 3D, 7DoF arm must use different end effectors to push or throw various objects to randomly-generated goal positions. Images and detailed descriptions of both families of tasks

are available in Appendix 5.A.

### 5.6.1 Low-dimensional states

{sec:action_experimen

For training the DynE action representation we use 100K steps with a uniformly random behavior policy in the simplest environment in each family with no reward or other supervisory signal. As this DynE pretraining is unsupervised and only occurs once for each family of environments, the $x$ axis on these training curves refers only to the samples used to train the policy.[12] We then transfer this action representation to all three environments in the family. When training DynE-TD3 we use all of the default hyperparameters from the TD3 implementation across all environments.

We directly test the impact of switching from raw to DynE actions by comparing TD3 to DynE-TD3. For completeness we compare with two additional state-of-the-art model-free methods: soft actor-critic (SAC) [Haarnoja et al. 2018b,c] and proximal policy optimization (PPO) [Schulman et al. 2017]. We also compare with soft actor-critic with latent space policies (SAC-LSP) [Haarnoja et al. 2018a], an innovative hierarchical method which transforms a low-level action space into an abstract one by training an invertible low-level policy. In all cases we use the official implementations[13][14][15] and the MuJoCo hyperparameters used by the authors. We also attempted to compare with the hierarchical method by Nachum et al. [2018], but after several emails with the authors and dozens of experiments we were unable to get it to converge on tasks other than those in their paper.

Results Figure 5.5 shows the results of these experiments. Most significantly, they show that switching from the raw action space (TD3 curve) to the DynE action space results in faster training and allows TD3 to solve the difficult 7DoF suite of tasks. We see that the DynE action space generalizes across several tasks with the same robot, even when interacting with objects unseen during training. It is especially worth noting that the gains from DynE increase as the tasks be-

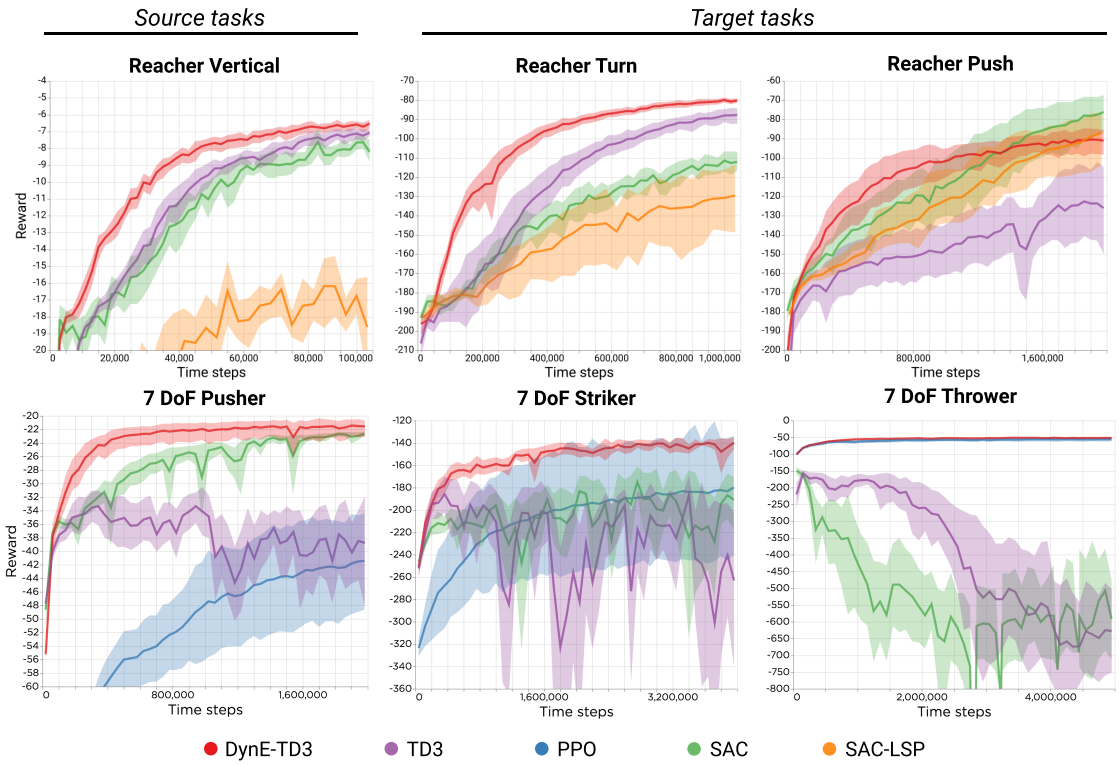



**Figure 5.5:** Performance of DynE-TD3 and baselines on two families of environments with low-dimensional observations. Dark lines are mean reward over 8 seeds and shaded areas are bootstrapped 95% confidence intervals. Across all the environments, TD3 learns faster with the DynE action space than with the raw actions. Within each family of environments, the DynE action space was trained only on the simplest task (left).

{fig:lowd_results}

come harder, maintaining convergence, stability, and low variance in the face of high-dimensional control with difficult exploration. Since SAC-LSP [Haarnoja et al. 2018a] performs similarly but worse than SAC we test it only on the simpler Reacher family of tasks; meanwhile, the PPO curves do not enter the frame on the Reacher family of tasks due to its poor sample efficiency.

### 5.6.2 Pixels

Using the Reacher family of environments we evaluate several state representations by their effectiveness for policy learning with TD3.

We evaluate two established methods for learning representations from single images. "DARLA" is the Disentangled Representation Learning Agent proposed by Higgins et al. [2017b] with the

denoising autoencoder loss, which is referred to in that work as $\beta$-VAE$_{DAE}$. "VAE" is a standard variational autoencoder [Kingma and Welling 2013; Rezende et al. 2014b], which has previously been found to learn effective representations for control [Van Hoof et al. 2016]; it is equivalent to DARLA with the pixel-space loss and $\beta = 1$. Since these representations operate on a single frame at a time, we apply them to the most recent four frames independently and then concatenate the embeddings before feeding them to the policy. These representations have compressed latent spaces, but they encode no knowledge of the environment's dynamics, allowing us to evaluate the importance of incorporating the dynamics into our embeddings.

Next we evaluate representation learning methods whose objectives incorporate the dynamics. "S-DynE," for State DynE, is the DynE state embedding $e_s$, and "SA-DynE" combines the DynE state and action representations. "S-Deterministic" and "SA-Deterministic" are ablations of the corresponding DynE methods which have the same forward-prediction objective but no KL or noise on the latent representations. Comparing the DynE methods to their respective ablations reveals the contribution of explicitly introducing a compression objective to the latent space.

For training all of the learned representations we use a dataset of 100K steps in each environment from a uniformly random policy. In every case we train TD3 with the learned representations using all of the default hyperparameters from the official TD3 implementation.

We compare these representation learning methods with TD3 trained from pixels. As there are no experiments on pixels in the TD3 paper, we performed extensive search over network architectures and hyperparameters. We included in our search the configurations used in the pixel experiments of DDPG [Lillicrap et al. 2015] as well as those used in successful discrete-action RL works from pixels [Schulman et al. 2017; Kostrikov 2018; Espeholt et al. 2018].

Results Figure 5.6 shows the results of these experiments. We find that the single-image methods are unable to solve any of the three tasks from pixels; TD3 from pixels diverges in all cases, while VAE and DARLA learn gradually at best. If simply reducing the dimension of the states

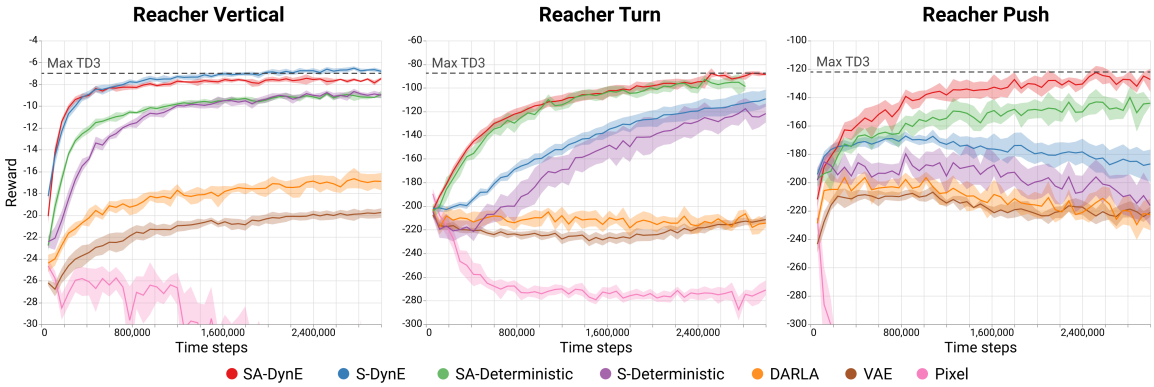



**Figure 5.6:** Performance of TD3 trained with various representations. Learned representations for state which incorporate the dynamics make a dramatic difference. SA-DynE converges stably and rapidly and achieves performance from pixels that nearly equals TD3's performance from states. Dark lines are mean reward over 8 seeds and shaded areas are bootstrapped 95% confidence intervals. {fig:pixel_results}

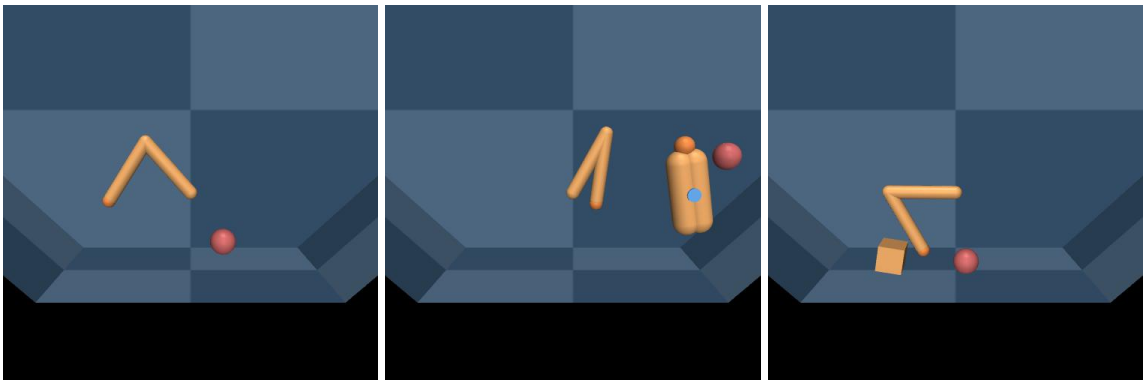
were sufficient to enable effective policy training, we would expect good performance from these methods. S-DynE and S-Deterministic, which incorporate the dynamics into their representation learning objectives, perform far better. The minimality imposed by the DynE objective allows S-DynE and SA-Dyne to outperform their deterministic ablations. SA-DynE learns rapidly and reliably, finding behaviors which qualitatively solve all three tasks. The improvement of SA-DynE over S-DynE shows that the state and action representations are complementary.

## 5.7 Discussion

In this work we proposed a method, Dynamics-aware Embedding (DynE), that jointly learns embedded representations of states and actions for reinforcement learning. Our experiments reveal that DynE action embeddings lead to more efficient exploration, resulting in more sample efficient learning on complex tasks, while DynE state embeddings allow unmodified model-free RL algorithms to scale to pixel observations. When combined, the DynE state and action embeddings result in stable, sample-efficient learning of high-quality policies from pixels.

# Appendix 5.A Environment description

{sec:task-renders}



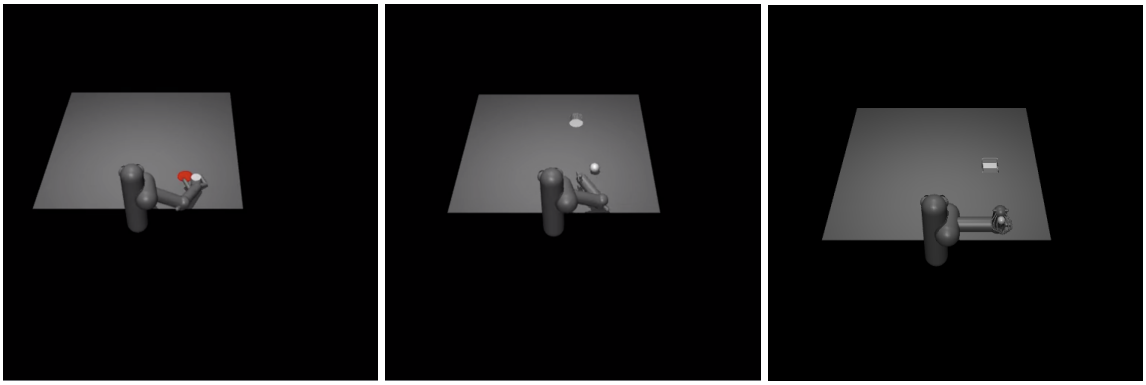
**(a)** ReacherVertical **(b)** ReacherTurn **(c)** ReacherPush

**Figure 5.7:** The Reacher family of environments. ReacherVertical requires the agent to move the tip of the arm to the red dot. ReacherTurn requires the agent to turn a rotating spinner (dark red) so that the tip of the spinner (gray) is close to the target point (red). ReacherPush requires the agent to push the brown box onto the red target point. The initial state of the simulator and the target point are randomized for each episode. In each environment the rewards are dense and there is a penalty on the norm of the actions. The robot's kinematics are the same in each environment but the state spaces are different.

{fig:reacher_family}

The first task family, pictured in Figure 5.7, is the "Reacher family", based on the Reacher-v2 MuJoCo [Todorov et al. 2012] task from OpenAI Gym [Brockman et al. 2016a]. These tasks form a simple new benchmark for multitask robot learning. The first task, which we use as the "source" task for training the DynE space, is ReacherVertical, a standard reach to a location task. The other two tasks are inspired by the DeepMind Control Suite's Finger Turn and Stacker environments, respectively [Tassa et al. 2018]. In ReacherTurn, the same 2-link Reacher robot must turn a spinner to the specified random location. In ReacherPush, the Reacher must push a block to the correct random location.

The second task family is the "7DoF family", which comprises Pusher-v2, Striker-v2, and Thrower-v2 from OpenAI Gym [Brockman et al. 2016a]. We use Pusher-v2 as the source task. These tasks use similar (though not identical) robot models, making them a feasible family of tasks for transfer. They are shown in Figure 5.8.



**(a)** Pusher-v2 **(b)** Striker-v2 **(c)** Thrower-v2

**Figure 5.8:** The 7DoF family of environments. Pusher-v2 requires the agent to use a C-shaped end effector to push a puck across the table onto a red circle. Striker-v2 requires the agent to use a flat end effector to hit a ball so that it rolls across the table and reaches the goal. Thrower-v2 requires the agent to throw a ball to a target using a small scoop. As with the Reacher family, the dynamics of the robot are the same within the 7DoF family of tasks. However, the morphology of the robot, as well as the object it interacts with, is different.

{fig:7dof_family}

### 5.A.1 Pixels

We use full-color images rendered at 256x256 and resized to 64x64 pixels. In order to allow the agents to perceive motion, we stack the current frame with the three most recent frames, resulting in an observation of dimension 12x64x64.

## Appendix 5.B Hyperparameters and DynE training

{sec:hyperparams}

For DynE-TD3 we use all of the default hyperparameters from the TD3 code[16] across all tasks. For all experiments we choose the dimension of the DynE action space to be equal to the dimension of a single action in the environment. We set the number of actions in the DynE space to be $k = 4$ for all experiments except Thrower-v2, for which we use $k = 8$. We use the Adam optimizer [Kingma and Ba 2014] with learning rate $10^{-4}$. All our experiments used recent-model NVidia GPUs.

Training on states When computing log-likelihoods we divide by the number of dimensions in the state in an attempt to make the correct settings of $\gamma$ invariant to the observation dimension; the same result could be achieved by multiplying the values of $\gamma$ that we report by the state dimension and changing the learning rate. With that scaling we set we set our hyperparameters $\gamma = \lambda = 10^{-2}$ across all environments. We concatenate all the joint angles and velocities to use as the states during representation learning. We preprocess the $s, s'$ pairs by first taking the difference $\Delta s = s' - s$ and then whitening so that $\Delta s$ has zero mean and unit variance in each dimension. This preprocessing encourages the encoder to represent both position and velocity in the latent space; the scales of these two components are quite different.

We use fully-connected networks for the action encoder $e_a$ and the conditional state predictor $f$. Each function has two hidden layers of 400 units. Training this model should take 5-10 minutes on GPU.

Training on pixels We train a DynE model for each environment, taking in a stack of frames and a sequence of $k = 4$ actions and predicting future states. To speed training we predict only the two latest frames of the future state (i.e. the picture of the world at time $t + k$ and $t + k - 1$) instead of all four. When doing RL we take the state encoder $e_s$ from this model and use it to preprocess all states from the environment.

We set the dimension of the state embedding $z_s$ to 100. We did not try other options, and given the sensitivity of RL to state dimension a smaller setting would very likely yield faster learning. We set $\beta = \gamma = 1$, at which setting DynE is optimizing a variational lower bound on $p(s_{t+k}|s_t, \boldsymbol{a}^k)$. We recommend ensuring that the predictions (not generations) from the model are correctly rendering all the task-relevant objects; if $\beta$ and $\gamma$ are too high, the model may incur lower loss by ignoring details in the image. We use cyclic KL annealing [Liu et al. 2019a] to improve convergence over a wide range of settings.

We use the DCGAN architecture [Radford et al. 2015] for the image encoder $e_s$ and the pre-

dictor $f$. The action encoder $e_a$ is fully connected with two hidden layers of 400 units. Training this model takes 1-2 hours on GPU.

## Appendix 5.C Varying levels of temporal abstraction

{sec:varying_k}

We study the impact of varying $k$, the level of temporal abstraction in the DynE action space. We find that increasing $k$ improves performance and learning speed up to a point; beyond this point, performance degrades. The optimal setting of $k$ will depend on the environment dynamics. We expect that environments with very slow dynamics will benefit from a greater degree of temporal abstraction.

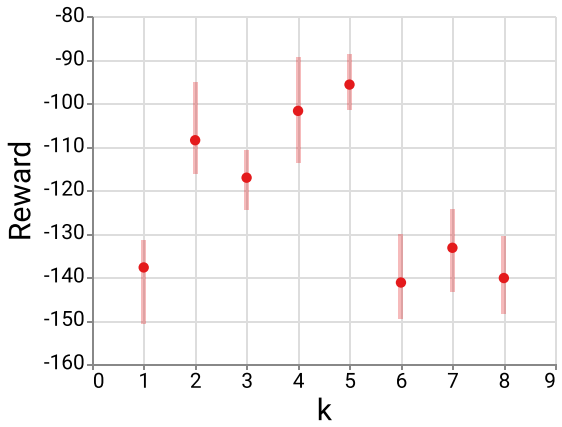



**Figure 5.9:** DynE-TD3 results on Reacher Push with varying $k$. We find that increased temporal abstraction improves performance up to a point, beyond which the action space is no longer able to represent the optimal policy and performace degrades. Solid points are the mean reward obtained after training for 1M environment steps. Shaded bars represent the min and max performance over 4 seeds.

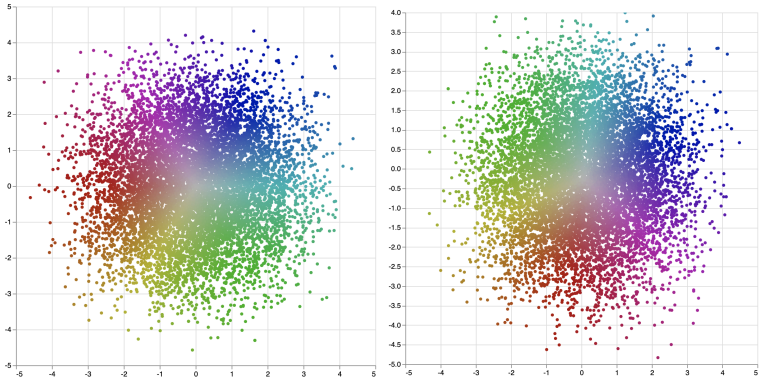
{fig:varying_k}

## Appendix 5.D Visualizing the DynE action space

{sec:latent-vis}

To better understand the structure in the DynE action embedding space, we visualize the relationship between the outcome of a sequence of actions and the DynE embedding of those actions. When embedding an action sequence, the DynE objective seeks to preserve information about

the outcome of that action sequence (i.e. the change in state), but minimize information about the original action sequence. Therefore we should see that all action sequences which have similar outcomes embed close together, regardless of the actions along the way. Figure 5.10 investigates this in a simple Point environment with an easy-to-visualize 2D $(x, y)$ state. For this simple problem, we see that all pairs of action sequences $\boldsymbol{a}_1^k$ and $\boldsymbol{a}_2^k$ with similar outcomes are close together in the embedding space. The correspondence between the two spaces appears to remain strong for high-dimensional and nonlinear environments, but is much harder to render in two dimensions.



**(a)** Outcome space **(b)** DynE action space

**Figure 5.10:** The mapping between the outcomes and embeddings of action sequences. We sample 10K random sequences of four actions and evaluate their outcomes in the environment dynamics, measured by $(\Delta x, \Delta y) = s_{t+4} - s_t$. **(a)** We plot the outcome $(\Delta x, \Delta y)$ of each action sequence and color each point according to its location in the plot. **(b)** We use DynE to embed each action sequence into two dimensions; each point in this plot corresponds to a point in (a) and takes its color from that corresponding point. The similarity of the two plots and the smooth color gradient in (b) indicate that DynE is embedding action sequences according to their outcomes.

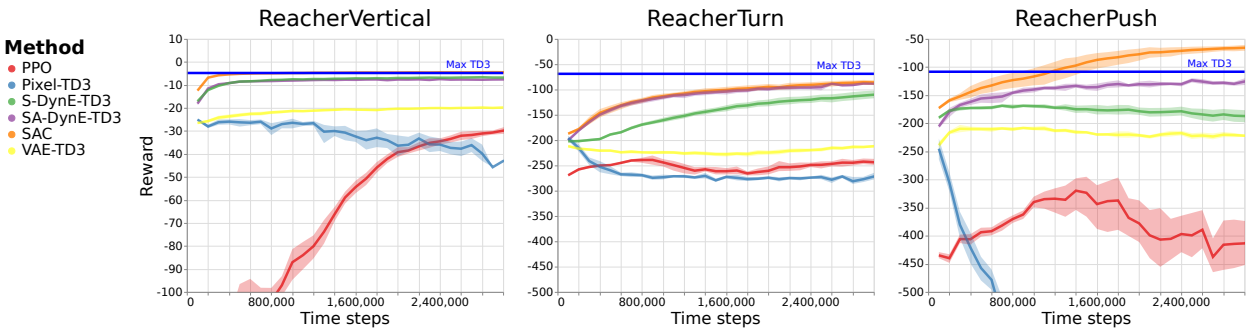
{fig:spaces}

## Appendix 5.E Extended results

{sec:extended_results

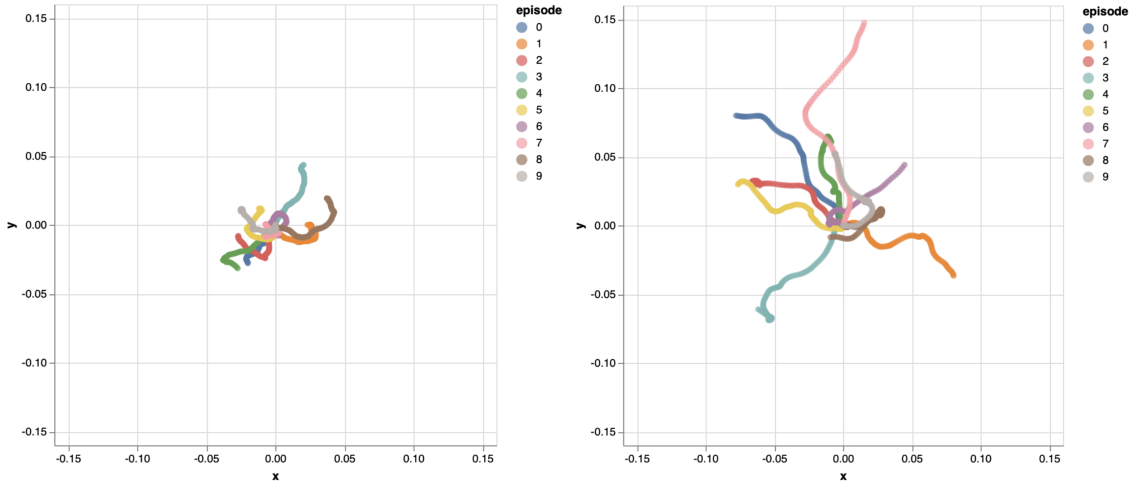
## Appendix 5.F Exploration with raw and DynE action spaces

{sec:exploration_traj




{fig:extended_pixel_r

**Figure 5.11:** These plots allow for direct comparison between the methods from pixels (Pixel-TD3, VAE-TD3, S-DynE-TD3, and SA-DynE-TD3) and our baselines from low-dimensional states (PPO and SAC). The DynE methods from pixels perform competitively with some baselines from states.




**(a)** Random exploration with raw actions **(b)** Random exploration with DynE

{fig:exploration}

**Figure 5.12:** These figures illustrate the way the DynE action space enables more efficient exploration. Each figure is generated by running a uniform random policy for ten episodes on a `PointMass` environment. Since the environment has only two position dimensions, we can plot the actual 2D position of the mass over the course of each episode. **Left:** A policy which selects actions at each environment timestep uniformly at random explores a very small region of the state space. **Right:** A policy which randomly selects DynE actions once every $k$ timesteps explores much more widely.

# Notes

10. Number of steps required to train D4PG taken from Hafner et al. [2018], as Barth-Maron et al. [2018] does not include this information.

11. In the interest of space we omit the usual recap of Markov decision processes and reinforcement learning. We refer the reader to Section 2 of Silver et al. [2014] for notation and background on MDPs.

12. On all environments except the simplest (Reacher Vertical) shifting the DynE-TD3 plot by 100K steps does not affect the ordering of the results.

13. TD3: https://github.com/sfujim/TD3/

14. SAC and SAC-LSP: https://github.com/haarnoja/sac

15. PPO: https://github.com/openai/baselines/tree/master/baselines/ppo2

16. https://github.com/sfujim/TD3

# Part III

# Improving Performance with Batched Data

{sec:offline}

Introduction to offline RL.

# 6 Offline Contextual Bandits with Overparameterized Models

{sec:offline-bandits}

## 6.1 Introduction

The offline contextual bandit problem can be used to model decision making from logged data in domains as diverse as recommender systems [Li et al. 2010; Bottou et al. 2013], healthcare [Prasad et al. 2017; Raghu et al. 2017], and robotics [Pinto and Gupta 2016]. Prior work on the problem has primarily focused on underparameterized models with finite and small VC dimensions. This work has come from the bandit literature [Strehl et al. 2010; Swaminathan and Joachims 2015a,b], the reinforcement learning literature [Munos and Szepesvári 2008; Chen and Jiang 2019], and the causal inference literature [Bottou et al. 2013; Athey and Wager 2017; Kallus 2018; Zhou et al. 2018].

In contrast, the best performance in modern supervised learning is often achieved by massively overparameterized models that are capable of fitting random labels [Zhang et al. 2016]. Use of such large models renders vacuous the bounds that require a small model class. But, the massive capacity of popular neural network models is now often viewed as a feature rather than a bug. Large models reduce approximation error and allow for easier optimization [Du et al. 2018] while still being able to generalize in regression and classification problems [Belkin et al. 2018, 2019]. In this paper, we investigate whether the strong performance of overparameterized

models in supervised learning translates to the offline contextual bandit setting. The main prior work that considers this setup is [Joachims et al. 2018], which we discuss in detail in Section 6.7.

To formalize the differences between the supervised learning and contextual bandit settings, we introduce a novel regret decomposition. This decomposition shares the approximation and estimation terms from classic work in supervised learning [Vapnik 1982; Bottou and Bousquet 2008], but adds a term for "bandit" error which captures the excess risk due to only receiving partial feedback.

We use this framework to address the question: can we use overparameterized models for offline contextual bandits? Or is the bandit error a fundamental problem when we use large models? We find mixed results. Value-based algorithms benefit from the same generalization behavior as overparameterized supervised learning, but policy-based algorithms do not. We show that this difference is explained by a property of their objectives called *action-stability*. An objective is action-stable if there exists a single prediction which is simultaneously optimal for any observed action (where a "prediction" is a vector of state-action values for a value-based objective or an action distribution for a policy-based objective). Action-stable objectives perform well when combined with overparameterized models since the random actions taken by the behavior policy do not change the optimal prediction. However, interpolating an unstable objective results in learning a different function for every sample of actions, even though the true optimal policy remains unchanged.

On the theory side, we prove that overparameterized value-based algorithms are action stable and have small bandit error via reduction to overparameterized regression. Meanwhile we prove that policy-based algorithms are not action-stable which allows us to prove lower bounds on the "in-sample" regret and lower bounds on the regret for simple nonparametric models.

Empirically, we demonstrate the gap in both action stability and bandit error between policy-based and value-based algorithms when using large neural network models on synthetic and image-based datasets.

In summary, our main contributions are:

- We introduce the concept of bandit error, which separates contextual bandits from supervised learning.
- We introduce action-stability and show that a lack of action-stability causes bandit error.
- We show a gap between policy-based and value-based algorithms based on action-stability and bandit error both in theory and experiments.

## 6.2 Setup

{sec:setup}

### 6.2.1 Offline contextual bandit problem

First we will define the contextual bandit problem [Langford and Zhang 2008]. Let the context space $\mathcal{X}$ be infinite and the action space $\mathcal{A}$ be finite with $|\mathcal{A}| = K < \infty$. At each round, a context $x \in \mathcal{X}$ and a full feedback reward vector $r \in [r_{\min}, r_{\max}]^K$ are drawn from a joint distribution $\mathcal{D}$. Note that $r$ can depend on $x$ since they are jointly distributed. A policy $\pi : \mathcal{X} \to \mathcal{P}(\mathcal{A})$ maps contexts to distributions over actions. An action $a$ is sampled according to $\pi(a|x)$ and the reward is $r(a)$, the component of the vector $r$ corresponding to $a$. We use "bandit feedback" to refer to only observing $r(a)$. This contrasts with the "full feedback" problem where at each round the full vector of rewards $r$ is revealed, independent of the action.

In the offline setting there is a finite dataset of $N$ rounds with a fixed behavior policy $\beta$. Then we denote the dataset as $S = \{x_i, r_i, a_i, p_i\}_{i=1}^N$ where $p_i$ is the observed propensity $p_i = \beta(a_i|x_i)$. The tuples in the datasets lie in $\mathcal{X} \times [r_{min}, r_{max}]^K \times \mathcal{A} \times [0, 1]$ and are drawn i.i.d from the joint distribution induced by $\mathcal{D}$ and $\beta$. From $S$ we define the datasets $S_B$ for bandit feedback and $S_F$ for

full feedback:

$$S_B = \{(x_i, r_i(a_i), a_i, p_i)\}_{i=1}^N, \quad S_F = \{(x_i, r_i)\}_{i=1}^N.$$

Note that we are assuming access to the behavior probabilities $p_i = \beta(a_i|x_i)$, so the issues that we raise do not have to do with estimating propensities. We will further make the following assumption about the behavior.

{ass:positivity}

**Assumption 1** (Strict positivity)**.** *We have strict positivity of $\tau$ if $\beta(a|x) \geq \tau > 0$ for all $a, x$. Thus, in any dataset we will have $p_i = \beta(a_i|x_i) \geq \tau > 0$.*

There is important work that focuses learning without strict positivity by making algorithmic modifications like clipping [Bottou et al. 2013; Strehl et al. 2010; Swaminathan and Joachims 2015a] and behavior constraints [Fujimoto et al. 2018b; Laroche et al. 2019]. However, these issues are orthogonal to the main contribution of our paper, so we focus on the setting with strict positivity.

The goal of an offline contextual bandit algorithm is to take in a dataset and produce a policy $\pi$ so as to maximize the value $V(\pi)$ defined as

$$V(\pi) := \mathop{\mathbb{E}}_{x,r\sim\mathcal{D}} \mathop{\mathbb{E}}_{a\sim\pi(\cdot|x)} [r(a)].$$

We will use $\pi^*$ to denote the deterministic policy that maximizes $V$. Finally, define the $Q$ function at a particular context, action pair as

$$Q(x, a) := \mathop{\mathbb{E}}_{r|x} [r(a)].$$

### 6.2.2 Model classes

The novelty of our setting comes from the use of overparameterized model classes that are capable of interpolating the training objective. To define this more formally, all of the algorithms we consider take a model class of either policies $\Pi$ or $Q$ functions $\mathcal{Q}$ and optimize some objective over the data with respect to the model class. Following the empirical work of [Zhang et al. 2016] and theoretical work of [Belkin et al. 2018] we will call a model class "overparameterized" or "interpolating" if the model class contains a model that exactly optimizes the training objective. Formally, if we have data $\{x_i\}_{i=1}^N$ and a pointwise loss function $\ell(x, y)$, then a model class $\Pi$ can interpolate the data if

$$\inf_{\pi \in \Pi} \sum_{i=1}^{N} \ell(x_i, \pi(x_i)) = \sum_{i=1}^{N} \inf_{y} \ell(x_i, y).$$

This contrasts with traditional statistical learning settings where we assume that the model class is finite or has low complexity as measured by something like VC dimension [Strehl et al. 2010; Swaminathan and Joachims 2015a].

### 6.2.3 Algorithms

{sec:basic-algs}

Now that we have defined the problem setting, we can define the algorithms that we will analyze. This is not meant to be a comprehensive account of all algorithms, but a broad picture of the "vanilla" versions of the main families of algorithms. Since we are focusing on statistical issues we do not consider how the objectives are optimized.

Supervised learning with full feedback. In a full feedback problem, empirical value maximization (the analog to standard empirical risk minimization) is defined by maximizing the em-

pirical value $\hat{V}_F$:

$$\hat{V}_F(\pi; S_F) := \frac{1}{N}\sum_{i=1}^{N}\langle r_i, \pi(\cdot|x_i)\rangle \tag{6.1}$$

$$\pi_F := \arg\max_{\pi\in\Pi} \hat{V}_F(\pi; S_F). \tag{6.2}$$

Policy-based learning. Importance weighted or "inverse propensity weighted" policy optimization directly optimizes the policy to maximize an estimate of its value. Since we only observe the rewards of the behavior policy, we use importance weighting to get an unbiased value estimate to maximize. Explicitly:

$$\hat{V}_B(\pi; S_B) := \frac{1}{N}\sum_{i=1}^{N} r_i(a_i)\frac{\pi(a_i|x_i)}{p_i} \tag{6.3}$$

$$\pi_B := \arg\max_{\pi\in\Pi} \hat{V}_B(\pi; S_B). \tag{6.4}$$ {{eq:pi}}

Note that this is the "vanilla" version of the policy-based algorithm and modifications like regularizers, baselines/control variates, clipped importance weights, and self-normalized importance weights have been proposed [Bottou et al. 2013; Joachims et al. 2018; Strehl et al. 2010; Swaminathan and Joachims 2015a,b]. For our purposes considering this vanilla version is sufficient since as we show in Section 6.4, any objective that takes the form $\pi(a_i|x_i)f(x_i, a_i, r_i, p_i)$ at each datapoint will have the same sort of problem with action-stability.

It is important to note that with underparameterized model classes, this algorithm is guaranteed to return nearly the best policy in the class. Explicilty, Strehl et al. [2010] prove that for a finite policy class $\Pi$, with high probability the regret of the learned policy $\pi_B$ is bounded as $O(\frac{1}{\tau}\sqrt{\frac{\log|\Pi|}{N}})$. This is elaborated in Appendix 6.F. However, these guarantees no longer hold in our overparameterized setting.

VALUE-BASED LEARNING. Another simple algorithm is to first learn the $Q$ function and then use a greedy policy with respect to this estimated $Q$ function. Explicitly:

$$\hat{Q}_{S_B} := \arg\min_{f \in \mathcal{Q}} \sum_{i=1}^{N} (f(x_i, a_i) - r_i(a_i))^2 \tag{6.5}$$ {{eqn:hatQ}}

$$\pi_{\hat{Q}_{S_B}}(a|x) := \mathbb{1}\left[a = \arg\max_{a'} \hat{Q}_{S_B}(x, a')\right]. \tag{6.6}$$

This algorithm is sometimes called the "direct method" [Dudík et al. 2011]. The RL literature also often defines a class of model-based algorithms, but in the contextual bandit problem there are no state transitions so model-based algorithms are equivalent to value-based algorithms.

This algorithm also has a guarantee with small model classes. Explicilty, Chen and Jiang [2019] prove that for a finite *and well specified* model class $\mathcal{Q}$, with high probability the regret of the learned policy $\pi_{\hat{Q}_{S_B}}$ is bounded as $O(\frac{1}{\sqrt{\tau}}\sqrt{\frac{\log|\mathcal{Q}|}{N}})$. This is elaborated in Appendix 6.F. Again, these guarantees no longer hold in our overparameterized setting.

DOUBLY ROBUST POLICY OPTIMIZATION. The class of doubly robust algorithms [Dudík et al. 2011] does not fall cleanly into the value-based or policy-based bins since it requires first learning a value function and using that to optimize a policy. However, with overparamterized models, doubly robust learning becomes exactly equivalent to our vanilla value-based algorithm unless we use crossfitting since the estimated Q values will coincide with the rewards. We prove this formally in Appendix 6.D where we also show some issues that the doubly robust policy objective can have with overparameterized models and highly stochastic rewards. For our purposes, we will only consider the policy-based and value-based approaches since the doubly robust approach collapses to the value-based approach with overparameterized models.

## 6.3 Bandit error

{sec:decomp}

In supervised learning, the standard decomposition of the excess risk separates the approximation and estimation error [Bottou and Bousquet 2008]. The approximation error is due to the limited function class and the estimation error is due to minimizing the empirical risk rather than the true risk. Since the full feedback policy learning problem is equivalent to supervised learning, the same decomposition applies. Formally, consider a full feedback algorithm $\mathcal{A}_F$ which takes the dataset $S_F$ and produces a policy $\pi_F$. Then

$$\underbrace{\mathop{\mathbb{E}}_{S}[V(\pi^*) - V(\pi_F)]}_{\text{regret}} = \underbrace{V(\pi^*) - \sup_{\pi \in \Pi} V(\pi)}_{\text{approximation error}} + \underbrace{\mathop{\mathbb{E}}_{S}[\sup_{\pi \in \Pi} V(\pi) - V(\pi_F)]}_{\text{estimation error}}.$$

We can instead consider a bandit feedback algorithm $\mathcal{A}_B$ which takes the dataset $S_B$ and produces a policy $\pi_B$. To extend the above decomposition to the bandit problem we add a new term, the bandit error, that results from having access to $S_B$ rather than $S_F$. Now we have:

$$\underbrace{\mathop{\mathbb{E}}_{S}[V(\pi^*) - V(\pi_B)]}_{\text{regret}} = \underbrace{V(\pi^*) - \sup_{\pi \in \Pi} V(\pi)}_{\text{approximation error}} + \underbrace{\mathop{\mathbb{E}}_{S}[\sup_{\pi \in \Pi} V(\pi) - V(\pi_F)]}_{\text{estimation error}} + \underbrace{\mathop{\mathbb{E}}_{S}[V(\pi_F) - V(\pi_B)]}_{\text{bandit error}}.$$

Disentangling sources of error. The approximation error is the same quantity that we encounter in the supervised learning problem, measuring how well our function class can do. The estimation error measures the error due to overfitting on finite contexts and noisy rewards. The bandit error accounts for the error due to only observing the actions chosen by the behavior policy. This is not quite analogous to overfitting to noise in the rewards since stochasticity in the actions is actually required to have the coverage of context-action pairs needed to learn a policy. While the standard approximation-estimation decomposition could be directly extended to the bandit problem, our approximation-estimation-bandit decomposition is more conceptually useful

since it disentangles these two types of error.

Can bandit error be negative? Usually, we think about an error decomposition as a sum of positive terms. This is not necessarily the case with our decomposition, but we view this as a feature rather than a bug. Intuitively, the bandit error term captures the contribution of the actions selected by the behavior policy. If the behavior policy is nearly optimal and the rewards are highly stochastic, there may be more signal in the actions selected by the behavior policy than the observed rewards. Thus overfitting the actions chosen by behavior policy can sometimes be beneficial, causing the bandit error to be negative. The two terms disentangle the approximation error (due to reward noise) from bandit error (due to behavior actions).

## 6.4 Action-stable objective functions

{sec:stable}

Consider a simple thought experiment. We collect a contextual bandit dataset $S_B$ from a two-action environment using a uniformly random behavior policy. Then we construct a second dataset $\widetilde{S}_B$ by evaluating the outcome of taking the opposite action at each observed context. Since nothing about the environment has changed, we know that the optimal policy remains the same. Therefore we desire the following property from a bandit objective: there exists a single model which is optimal (with respect to that objective) on both $S_B$ and $\widetilde{S}_B$. We say that such an objective is *action-stable* because it has an optimal policy which is stable to re-sampling of the actions in the dataset.

More formally, we define action stability pointwise at a datapoint $z = (x, r, p)$ where $r \in [r_{\min}, r_{\max}]^K$ and $p \in \Delta^K$ is the behavior probability vector in the $K$-dimensional simplex (recall that $K$ is the number of the actions). Let $z(a)$ denote the datapoint when action $a$ is sampled so that $z(a) = (x, r(a), p(a), a)$. The objectives for both policy and value-based algorithms decompose into sums over the data of some loss $\ell(z(a), \pi(a|x))$ or $\ell(z(a), Q(x, a))$.

Note that the output space of a policy is the simplex so that $\pi(\cdot|x) \in \Delta^K$, while the output of a Q function[17] is $Q(x, \cdot) \in \mathbb{R}^K$. To allow for this difference in our definition, we will define a generic $K$-dimensional output space $\mathcal{Y}^K$ and its corresponding restriction to one dimension as $\mathcal{Y}$. So for a policy-based algorithm $\mathcal{Y}^K = \Delta^K$ and $\mathcal{Y} = [0, 1]$, while for a value-based algorithm $\mathcal{Y}^K = \mathbb{R}^K$ and $\mathcal{Y} = \mathbb{R}$. Now we can define action-stability.

**Definition 6.1** (Action-stable objective)**.** An objective function $\ell$ is action-stable at a datapoint $z$ if there exists $y^* \in \mathcal{Y}^K$ such that for all $a \in \mathcal{A}$:

$$\ell(z(a), y^*(a)) = \min_{y \in \mathcal{Y}} \ell(z(a), y).$$

If an objective is not action-stable, a function which minimizes that objective exactly at every datapoint $(x, r(a), p(a), a)$ does *not* minimize it for a different choice of $a$. As a direct consequence, interpolating an unstable objective results in learning a different function for every sample of actions, even though the true optimal policy remains unchanged.

We find that policy-based objectives are not action-stable, while value-based objectives are. In the next section we will use the instability of policy-based objectives to show that policy-based algorithms exhibit large bandit error when used with overparameterized models. Our stability results are stated in the following two Lemmas, whose proofs can be found in Appendix 6.A.

{thmt@@vbstable@data}
{thmt@@vbstable}

**Lemma 6.2** (Value-based stability)**.** *Value-based objectives are action stable since we can let $y^* = r$ and this minimizes the square loss at every action.*

{thmt@@pbstable@data}
{thmt@@pbstable}

**Lemma 6.3** (Policy-based instability)**.** *All policy-based objectives which take the form $\ell(z(a), \pi(a|x)) = f(z(a))\pi(a|x)$ are not action-stable at $z$ unless $f(z(a)) > 0$ for exactly one action $a$.*

These Lemmas tell us that the stochasticity of the behavior policy can cause instability for policy-based objectives. This is worrisome since one would hope that more stochastic behavior

policies give us more information about all the actions and should thus yield better policies. Indeed, this is the case for value-based algorithms as we will see in the next section. But for policy-based algorithms, stochastic behavior can itself be a cause of overfitting due to the instability of the objective function.

Stabilizing policy-based algorithms. To avoid this problem in a policy-based algorithm, the sign of the function $f(z(a))$ must indicate the optimal action. This essentially requires having access to a *baseline* function $b(s)$ that separates the optimal action from all the others so that $r(a) > b(s)$ if and only if $a$ is the optimal action. And then $f(z(a)) = \frac{r(a)-b(s)}{\beta(a|s)}$ yields an action-stable algorithm. This is in general as difficult as learning the full value function $Q$. One notable special case is when the bandit problem is induced by an underlying classification problem, so that only one action has reward 1 and all others have 0. In this case, any constant baseline between 0 and 1 will lead to action stability. This case has often been considered in the literature, e.g. by Joachims et al. [2018] as we discuss in Section 6.7.

Now that we have built up an understanding of the problem we can prove some formal results that show how value-based algorithms more effectively leverage overparameterized models by being action-stable.

## 6.5 Regret bounds

Recall that as explained in Section 6.2, both policy-based and value-based algorithms have regret guarantees when we use small model classes [Strehl et al. 2010; Chen and Jiang 2019]. But, when we move to the overparameterized setting, this is no longer the case. In this section we prove regret upper bounds for value-based learning by using recent results in overparameterized regression. Then we prove lower bounds on the regret of policy-based algorithms due to their action-instability.

### 6.5.1 Value-based learning

In this section we show that value-based algorithms can provably compete with the optimal policy. The key insight is to reduce the problem to regression and then leverage the guarantees on overparameterized regression from the supervised learning literature. This is formalized by the following theorem.

{thmt@@reduction@data
{thmt@@reduction}

**Theorem 6.4** (Reduction to regression)**.** *By Assumption 1 we have $\beta(a|x) \geq \tau$ for all $x, a$. Then with $\hat{Q}_{S_B}$ as defined in (6.5) we have*

$$V(\pi^*) - V(\pi_{\hat{Q}_{S_B}}) \leq \frac{2}{\sqrt{\tau}} \sqrt{\mathop{\mathbb{E}}_{x,a\sim\beta}[(Q(x,a) - \hat{Q}_{S_B}(x,a))^2]}.$$

A proof can be found in Appendix 6.B. Similar results are presented as intermediate results in Chen and Jiang [2019]; Munos and Szepesvári [2008]. The implication of this result that we want to emphasize is that any generalization guarantees for overparameterized regression immediately become guarantees for value-based learning in offline contextual bandits. Essentially, Theorem 6.4 gives us a regret bound in any problem where overparameterized regression works. The following results from the overparameterized regression literature demonstrate a few of these guarantees, which all require some sort of regularity assumption on the true $Q$ function to bound the regression error:

- The results of [Bartlett et al. 2020] give finite sample rates for overparameterized linear regression by the minimum norm interpolator depending on the covariance matrix of the data and assuming that the true function is realizable.
- The results of [Belkin et al. 2019] imply that under smoothness assumptions on $Q$, a particular singular kernel will interpolate the data and have optimal non-parametric rates. After applying our reduction, the rates are no longer optimal for the policy learning problem due to the square root.

- The results of [Bach 2017] show how choosing the minimum norm infinite width neural network in a particular function space can yield adaptive finite sample guarantees for many types of underlying structure in the $Q$ function.
- The results of [Cover 1968] imply the consistency of a one nearest neighbor regressor when the rewards are noiseless and $Q$ is piecewise continuous. This will contrast nicely with Theorem 6.6 below.

Each of these guarantees implies a corresponding corollary to Theorem 6.4 resulting in a regret bound for that particular combination of model and assumptions on $Q$.

### 6.5.2 Policy-based learning

Now we will show how the policy-based learning algorithms can provably fail because they lack action-stability. We will do this in a few ways. First, we will show that on the contexts in the dataset an action-unstable algorithm must suffer regret. This means that we cannot even learn the optimal policy at the contexts seen during training. Then to deal with generalization beyond the dataset we will prove a regret lower bound for a specific overparameterized model, namely one nearest neighbor. Finally, we discuss a conjecture that such lower bounds can be extended to richer model classes like neural networks.

Since we are proving lower bounds, making any more simplifying assumptions only makes the bound stronger. As such, all of our problem instances that create the lower bounds have only two actions ($K = 2$).

Regret on the observed contexts. Before considering how a policy generalizes off of the data, it is useful to consider what happens at the contexts in the dataset. This is especially true for overparameterized models which can exactly optimize the objective on the dataset. To do this, we will define the value of a policy $\pi$ on the contexts in a dataset $S$ (which we will call the

"in-sample" value) by

$$V(\pi; S) := \frac{1}{N} \sum_{i=1}^{N} \mathop{\mathbb{E}}_{r|x_i} \mathop{\mathbb{E}}_{a \sim \pi(\cdot|x_i)} [r(a)]. \tag{6.7}$$

Then the following Theorem shows that the policy $\pi_B$ learned by the simple policy-based algorithm in Equation (6.4) must suffer substantial regret on $S$.

{thmt@@vsthm@data}
{thmt@@vsthm}

**Theorem 6.5** (In-sample regret lower bound)**.** *Let $K = 2$ and the policy class be overparameterized. Define $\Delta_r(x) = \left|\mathbb{E}_{r|x}[r(1) - r(2)]\right|$ as the absolute expected gap in rewards at $x$. Define $p_u(x)$ to be the probability that the policy-based objective is action-unstable at $x$. Recall that $\beta(a|x) \geq \tau$ by Assumption 1. Then*

$$\mathop{\mathbb{E}}_{S}[V(\pi^*; S) - V(\pi_B; S)] \geq \tau \mathop{\mathbb{E}}_{x} \left[p_u(x)\Delta_r(x)\right].$$

The full proof is in Appendix 6.C. This Theorem tells us that as often as the objective is action-unstable, we can suffer substantial regret even *on the contexts in the dataset.* We now offer some brief intuition of the proof. When we have two actions and an algorithm is not action-stable at $x$, then action chosen by the learned policy $\pi_B$ at $x_i$ is directly dependent on the observed action $a_i$. Since the behavior $\beta$ will choose each action with probability at least $\tau$ by Assumption 1, the learned policy $\pi_B$ must choose the suboptimal action with probability at least $\tau$ at $x_i$. This causes unavoidable regret for unstable algorithms, as formally stated in Theorem 6.5.

Note that Theorem 6.5 essentially says that action-unstable algorithms convert noise in the behavior into regret. This is the essential problem with unstable algorithms. Rather than using stochasticity in the behavior policy to get estimates of counterfactual actions, an action-unstable algorithm is sensitive to this stochasticity like it is label noise in supervised learning.

In Appendix 6.C.2 we present a result that makes this connection between behavior policy noise and action-instability more direct. Specifically we show a reduction that takes any classifi-

cation problem and turns it into a bandit problem such that optimizing the policy-based objective is equivalent to solving a noisy variant of the classification problem. On the contrary, optimizing the full-feedback objective is equivalent to the noiseless classification problem.

The limitation of this result is that it only applies in-sample and does not rule out that the model class could leverage its inductive bias to perform well away from the training data. Next we convert this in-sample regret lower bound into a standard regret lower bound for a particular simple interpolating model class, the nearest neighbor policy.

Regret with generalization: nearest neighbor models. The above result shows what happens at the contexts *in the dataset* $S$. It seems that only pathological combinations of model class and problem instance could perform poorly on $S$ but recover strong performance off of the data. However, it cannot be ruled out a priori if the model class has strong inductive biases to generalize well. In this section we will show that at least for a very simple overparameterized model class, the generalization of the model does not improve performance.

The following theorem shows that the simplest interpolating model class, a one nearest neighbor rule, fails to recover the optimal policy even in the limit of infinite data.

{thmt@@nn@data}
{thmt@@nn}

**Theorem 6.6** (Regret lower bound for one nearest neighbor)**.** *Let* $\Delta_r = r_{\max} - r_{\min}$. *Then there exist problem instances with noiseless rewards where*

$$\limsup_{N\to\infty} \mathop{\mathbb{E}}_{S}[V(\pi_F) - V(\pi_B)] = \frac{\Delta_r}{2},$$

*but*

$$\limsup_{N\to\infty} \mathop{\mathbb{E}}_{S}[V(\pi^*) - V(\pi_F)] = 0.$$

The proof is in Appendix 6.C. This result shows that using a nearest neighbor scheme to generalize based on the signal provided by the policy-based objective is not sufficient to learn

an optimal policy. Importantly, note that since the rewards are noiseless, a nearest neighbor policy does recover the optimal policy with full feedback and Theorem 6.4 shows that value-based algorithms also recover the optimal policy in this setting. So, the model class is capable of solving the problem, it is the action-unstable algorithm that is causing irreducible error.

More complicated model classes. The above result for nearest neighbor models is illustrative, but does not apply to richer model classes like neural networks. While we were not able to construct such lower bounds, we conjecture that they do exist and hope that future work can prove them. We have two reasons to believe that such lower bounds exist. First, Theorem 6.5 is agnostic to model class. For a policy to perform well despite poor performance in-sample would require strong inductive biases from the model class. Proving lower bounds requires ruling out such inductive biases as we have shown for nearest neighbor rules. Second, our empirical results presented in the next section show that policy-based algorithms have action-instability and high bandit error with neural networks. The inductive biases are not enough to overcome the poor in-sample performance.

## 6.6 Experiments

{sec:exp}

In this section we experimentally verify that the phenomena described by the theory above extend to practical settings that go slightly beyond the assumptions of the theory.[18] Specifically we want to verify the following with overparameterized neural network models:

1. Policy-based algorithms are action-unstable while value-based algorithms are action-stable.
2. This causes high bandit error for policy-based algorithms, but not value-based algorithms.

We will break the section into two parts. First we consider a synthetic problem using simple feed-forward neural nets and then we show similar phenomena when the contexts are high-dimensional images and the models are Resnets [He et al. 2016].

### 6.6.1 Synthetic data

First, we will clearly demonstrate action-stability and bandit error in a synthetic problem with a linear reward function. Specifically, we sample some hidden reward matrix $\theta$ and then sample contexts and rewards from isotropic Gaussians:

$$\theta \sim U([0,1]^{K\times d}), \quad x \sim \mathcal{N}(0, I_d), \quad r \sim \mathcal{N}(\theta x, \epsilon I_d).$$

Actions are sampled according to a uniform behavior:

$$a \sim \beta(\cdot|x) = U(\{1, \ldots, K\}).$$

For these experiments we set $K = 2, d = 10, \epsilon = 0.1$. We take $N = 100$ training points and sample an independent test set of 500 points. As our models we use MLPs with one hidden layer of width 512. In our experience, the findings are robust to all these hyperparameters of the problem so long as the model is overparameterized. Full details about the training procedure along with learning curves and further results are in Appendix 6.E.

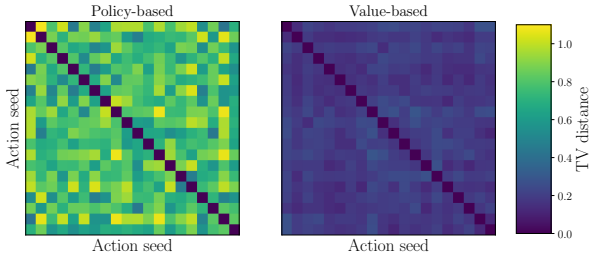



**Figure 6.1:** We test action-stability by resampling the actions 20 times for a single dataset of contexts. Each pixel corresponds to the pair of action seeds $i, j$ and the color shows the TV distance between $\pi_i(\cdot|x)$ and $\pi_j(\cdot|x)$ on a held-out test set sampled from the data generating distribution. The policy-based algorithms are highly sensitive to the randomly sampled actions. {fig:toy_stability}

To confirm (1) and (2) listed above we conduct two experiments. First, to test the action-stability of an algorithm with a neural network model, we train 20 different policies on the same

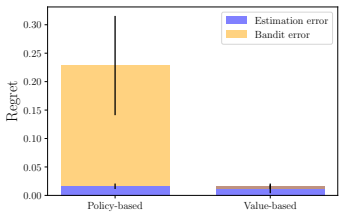



**Figure 6.2:** Estimated bandit error by averaging the values calculated on the held-out test sets for 50 independently sampled datasets. Error bars show one standard deviation. While policy-based learning has high bandit error, value-based learning has essentially zero bandit error.

{fig:toy_regret}

dataset of contexts and rewards, but with resampled actions. Formally, we sample $\{x_i, r_i\}_{i=1}^N$ from the Gaussian distributions described above and then sample $a_i \sim \beta(\cdot|x_i)$ with 20 different seeds. We then train each algorithm to convergence and compare the resulting policies by total variation (TV) distance. Results are shown in Figure 6.1. We find that our results from Section 6.4 are confirmed: policy-based algorithms are unstable leading to high TV distance between policies trained on different seeds while value-based algorithms are stable.

Second, we estimate the bandit error of each algorithm. To do this we train policies to convergence for the policy-based, value-based, and full-feedback objectives 50 independently sampled datasets (where now we also resample the contexts and rewards). For this estimate, we assume perfect optimization and no approximation error. Each estimate is calculated on a held out test set. Explicitly, let $\pi_B^j, \pi_Q^j, \pi_F^j$ are the policy-based, value-based, and full-feedback policies trained on dataset $S^j$ with seed $j$ and corresponding test set $T^j$. Then we estimate bandit error as $\frac{1}{J}\sum_{j=1}^J V(\pi_F^j; T^j) - V(\pi_B^j; T^j)$. Similarly, since we know $\theta$ we can compute $\pi^*$ and use this to estimate the estimation error. The results shown in Figure 6.2 demonstrate that the policy-based algorithm suffers from substantially more bandit error and thus more regret.

### 6.6.2 Classification data

Most prior work on offline contextual bandits conducts experiments on classification datasets that are transformed into bandit problems [Beygelzimer and Langford 2009; Dudík et al. 2011;

Swaminathan and Joachims 2015a,b; Joachims et al. 2018; Chen et al. 2019]. This methodology obscures issues of action-stability because the very particular reward function used (namely 1 for a correct label and 0 for incorrect) makes the policy-based objective action-stable. However, even minor changes to this reward function can dramatically change the performance of policy-based algorithms by rendering the objective action-unstable.

To make a clear comparison to prior work that uses deep neural networks for offline contextual bandits like Joachims et al. [2018], we will consider the same image based bandit problem that they do in their work. Namely, we will use the a bandit version of CIFAR-10 [Krizhevsky 2009].

To turn CIFAR into an offline bandit problem we view each possible label as an action and assign reward of 1 for a correct label/action and 0 for an incorrect label/action. We use two different behavior policies to generate training data: (1) a uniformly random behavior policy and (2) the hand-crafted policy used in [Joachims et al. 2018]. We train Resnet-18 [He et al. 2016] models using Pytorch [Paszke et al. 2019b]. Again full details about the training procedure are in Appendix 6.E.

As explained in Section 6.4, the policy-based objective is stable if and only if the sign of the reward minus baseline is an indicator of the optimal action. To test this insight we consider two variants of policy-based learning: "unstable" where we use a baseline of -0.1 so that the effective rewards are 1.1 for a correct label and 0.1 for incorrect and "stable" where we use a baseline of 0.1 so that the effective rewards are of 0.9 and -0.1 to make the objective stable[19]. Note that this "stable" variant of the algorithm *only* exists because we are considering a classification problem. In settings with more rich structure in the rewards, defining such an algorithm is not possible and only versions of the unstable algorithm would exist.

We again estimate the regret decomposition as we did with the synthetic data. The difference is that this time we only use one seed since we only have one CIFAR-10 dataset. The results in Figure 6.3 confirm the results from the synthetic data. The standard (unstable) policy-based

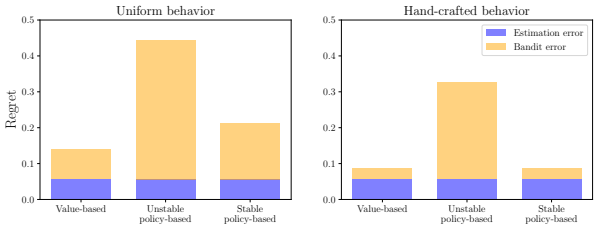



**Figure 6.3:** Estimated regret decomposition on CIFAR with uniform behavior (left) and the hand-crafted behavior of Joachims et al. [2018] (right). We see that the value-based learning has lowest bandit error and unstable policy-based learning the most. On the hand-crafted dataset the stable policy-based algorithm performs as well as value-based learning. {fig:cifar_regret}

algorithm suffers from large bandit error. The value-based algorithm has the best performance across both datasets although the "stable" policy-based algorithm performs about as well for the hand-crafted behavior policy.

## 6.7 Related work

{sec:related}

Now that we have presented our results, we will contrast them with some related work to clarify our contribution.

### 6.7.1 Relation to propensity overfitting

Swaminathan and Joachims [2015b] introduce what they call "propensity overfitting". By providing an example, they show that policy-based algorithms overfit to maximize the sum of propensities ($\sum_i \frac{\pi(a_i|x_i)}{p_i}$) rather than the value when the rewards are strictly positive. They provide a motivating example, but no formal definition of propensity overfitting and argue that it helps to describe the gap between supervised learning and bandit learning. In contrast, we introduce and formally define bandit error, which makes this gap between supervised learning and bandit learning precise and does not rely on the specific algorithm being used. Then we introduce and formally define action-instability, which explains an important cause of bandit error for policy-based algorithms when using large models. By mathematically formalizing these ideas we provide

a more rigorous foundation for future work.

### 6.7.2 Relation to [Joachims et al. 2018]

The main related work that considers offline contextual bandits with large neural network models is Joachims et al. [2018]. Specifically, that paper proposes a policy-based algorithm with an objective of the form: $\frac{r_i(a_i)-\lambda}{\beta(a_i|x_i)}\pi(a_i|x_i)$ for some constant baseline $\lambda$ determined by a hyperparameter sweep, but motivated by a connection to self-normalized importance sampling.

Our work contrasts with this prior work in two key ways. First, we show that the algorithm proposed in Joachims et al. [2018] is action-unstable. Specifically, our Lemma 6.3 shows that any policy-based algorithm with an objective of the form $\sum_i f(z_i(a_i))\pi(a_i|x_i)$ cannot be action-stable unless the sign of $f(z(a))$ is the indicator of the optimal action. However, since that paper only tests the algorithm on classification problems where the rewards are in $\{0, 1\}$, any setting of $\lambda$ between 0 and 1 causes the sign of $f$ to indicate the optimal action. The action-stability analysis shows how this algorithm will struggle beyond the classification setting.

Second, we show that value-based methods provably and empirically work in the overparameterized setting. In contrast, Joachims et al. [2018] does not consider value-based methods. We show that value-based methods are not affected by action-stability issues (Lemma 6.2) and have vanishing bandit error (Theorem 6.4). We empirically test this conclusion on the same CIFAR-10 bandit problem as Joachims et al. [2018] and find that a value-based approach outperforms the policy-based approach proposed in that paper (Figure 6.3).

### 6.7.3 Variance of importance weighting

The importance weighted value estimates used by policy-based algorithms suffer from high variance due to low probability actions that have very large importance weights. Much prior work focuses on reducing this variance [Strehl et al. 2010; Bottou et al. 2013; Swaminathan and Joachims

2015a]. In contrast, the issue we consider, action-instability in the overparameterized setting, is distinct from this variance issue. When the policy class is flexible enough to optimize the objective at each datapoint, the optimal predictor in that class does not depend on the importance weights. Meanwhile action-unstable objectives translate stochasticity in the behavior policy into noise in the objective, causing the overfitting issues that we see in policy-based algorithms. In fact, our Theorem 6.5 suggests that regret will be worse for more uniform behavior policies when using an action-unstable objective, even though these may be beneficial in terms of variance. This is born out in our experiments where the behavior is usually *uniform* and *known*, which is a favorable setup in terms of the variance of the value estimates, but an unfavorable setup for action-unstable policy learning algorithms.

## 6.8 Discussion

We have examined the offline contextual bandit problem with overparameterized models. We introduced a new regret decomposition to separate the effects of estimation error and bandit error. We showed that policy-based algorithms are not action-stable and thus suffer from high bandit error with stochastic behavior policies. This is borne out both in the theory and experiments.

It is important to emphasize that our results may not apply beyond the setting we consider in this paper. When there is no strict positivity, there is unobserved confounding, there are continuous actions, or the model classes are small and misspecified then policy-based learning may have lower regret and lower bandit error than value-based learning.

In future work we hope to extend the ideas from the bandit setting to the full RL problem with longer horizon that requires temporal credit assignment. We predict that action-stability and bandit error remain significant issues there. We also hope to investigate action-stable algorithms beyond the simple value-based algorithms we consider here.

## Appendix 6.A Action-stability

{app:stable}

**Lemma 6.2** (Value-based stability)**.** *Value-based objectives are action stable since we can let $y^* = r$ and this minimizes the square loss at every action.*

*Proof.* Consider a datapoint $z = (x, r)$ which becomes $z(a) = (x, a, r(a))$ when we sample action $a$ from the behavior. At this datapoint, the value-based objective for an estimated Q function $\hat{Q}$ is

$$\ell(z(a), \hat{Q}(x, a)) = (r(a) - \hat{Q}(x, a))^2 \tag{6.8}$$

This is minimized at all $a$ by $\hat{Q}(x, a) = r(a)$. So setting $y^* = \hat{Q}(x, \cdot) = r$, we can exactly minimize $\ell$ at $z$. Since such a $y^*$ exists, the objective is by definition action-stable. □

**Lemma 6.3** (Policy-based instability)**.** *All policy-based objectives which take the form $\ell(z(a), \pi(a|x)) = f(z(a))\pi(a|x)$ are not action-stable at $z$ unless $f(z(a)) > 0$ for exactly one action $a$.*

*Proof.* Consider a datapoint $z = (x, r)$ which becomes $z(a) = (x, a, r(a), p(a))$ when we sample action $a$ from the behavior with probability $p(a)$. At this datapoint, a generic policy-based objective evaluated on a policy $\hat{\pi}$ takes the form

$$\ell(z(a), \hat{\pi}(a|x)) = f(z(a))\hat{\pi}(a|x) \tag{6.9}$$

As special examples of the function $f$ we have the generic policy-based objective from Equation (6.4) when $f(z(a)) = \frac{r(a)}{p(a)}$. Moreover we can incorporate any baseline function $b(x)$ so that $f(z(a)) = \frac{r(a)-b(x)}{p(a)}$. This algorithm covers the one presented by Joachims et al. [2018].

Now to prove the claim, we have three cases: (1) $f(z(a)) < 0$ for all $a$, (2) $f(z(a)) > 0$ for at least two actions $a_1, a_2$, and (3) $f(z(a)) > 0$ at exactly one action $a_1$. We will show that in cases

1 and 2 the objective is action-unstable, but in case 3 it is action-stable.

Case 1. Assume that $f(z(a)) < 0$ for all $a$. Now for any given $a$ to maximize the objective $f(z(a))\hat{\pi}(a|x)$ while ensuring that $\hat{\pi}(a|x)$ is a valid probability we must set $\hat{\pi}(a|x) = 0$. But, if we set $\hat{\pi}(a|x) = 0$ for all $a$, we no longer have a valid probability distribution, since $0 \notin \Delta^K$. Thus, we cannot find $y^* \in \Delta^K$ that optimizes the loss at $z$ across all actions, so the objective is action-unstable.

Case 2. Assume that $f(z(a)) > 0$ for at least two actions $a_1, a_2$. Now at $a_1, a_2$ the objective $f(z(a))\hat{\pi}(a|x)$ is maximized by setting $\pi(a|x) = 1$. However, there is no valid element $y$ of $\Delta^K$ such that $y(a_1) = 1$ and $y(a_2) = 1$. Thus, we cannot find $y^* \in \Delta^K$ that optimizes the loss at $z$ across all actions, so the objective is action-unstable.

Case 3. Assume $f(z(a)) > 0$ at exactly one action $a_1$. Then at action $a_1$ we can maximize $f(z(a_1))\hat{\pi}(a_1|x)$ by setting $\hat{\pi}(a_1|x) = 1$. And since $f(z(a)) \leq 0$ for all other actions $a \neq a_1$, we can maximize $f(z(a))\hat{\pi}(a|x)$ by setting $\hat{\pi}(a|x) = 0$. Now since $\mathbb{1}[a = a_1] \in \Delta^K$, there does exist a vector $y^* \in \mathcal{Y}$ which exactly optimizes $\ell$ regardless of which action is sampled. So, the objective is action-stable if and only if we are in this case. □

## Appendix 6.B Value-based learning

{app:value}

**Theorem 6.4** (Reduction to regression)**.** *By Assumption 1 we have $\beta(a|x) \geq \tau$ for all $x, a$. Then with $\hat{Q}_{S_B}$ as defined in (6.5) we have*

$$V(\pi^*) - V(\pi_{\hat{Q}_{S_B}}) \leq \frac{2}{\sqrt{\tau}} \sqrt{\mathop{\mathbb{E}}_{x,a\sim\beta}[(Q(x,a) - \hat{Q}_{S_B}(x,a))^2]}.$$

*Proof.* The proof follows directly from linking the subsequent lemmas with $\hat{\pi} = \pi_{\hat{Q}_{S_B}}$ and $\Pi$ be the set of all policies in Lemma 6.7. □

{thmt@@mismatch@data}
{thmt@@mismatch}

**Lemma 6.7** (Mismatch: from MSE to Regret)**.** *Assume strict positivity. Let $\hat{\pi}$ be the greedy policy with respect to some $\hat{Q}$ and let $\Pi$ be any class of policies to compete against, which contains $\hat{\pi}$. Then*

$$\sup_{\pi \in \Pi} V(\pi) - V(\hat{\pi}) \leq 2\sqrt{\sup_{\pi \in \Pi} \mathop{\mathbb{E}}_{x,a \sim \mathcal{D},\pi}[(Q(x,a) - \hat{Q}(x,a))^2]} \tag{6.10}$$

*Proof.* We can expand the definition of regret and then add and subtract and apply a few inequalities. Let $\bar{\pi}$ be the policy in $\Pi$ which maximizes $V$. Then

$$\begin{aligned}
\sup_{\pi \in \Pi} V(\pi) - V(\hat{\pi}) &= \mathop{\mathbb{E}}_{x}\left[\mathop{\mathbb{E}}_{a \sim \bar{\pi}|x}[Q(x,a)] - \mathop{\mathbb{E}}_{a \sim \hat{\pi}|x}[Q(x,a)]\right] && (6.11)\\
&= \mathop{\mathbb{E}}_{x}\left[\mathop{\mathbb{E}}_{a \sim \bar{\pi}|x}[Q(x,a)] - \mathop{\mathbb{E}}_{a \sim \hat{\pi}|x}[\hat{Q}(x,a)] + \mathop{\mathbb{E}}_{a \sim \hat{\pi}|x}[\hat{Q}(x,a)] - \mathop{\mathbb{E}}_{a \sim \hat{\pi}|x}[Q(x,a)]\right] && (6.12)\\
&\leq \mathop{\mathbb{E}}_{x}\left[\mathop{\mathbb{E}}_{a \sim \bar{\pi}|x}[|Q(x,a) - \hat{Q}(x,a)|] + \mathop{\mathbb{E}}_{a \sim \hat{\pi}|x}[|Q(x,a) - \hat{Q}(x,a)|]\right] && (6.13)\\
&\leq \sqrt{\mathop{\mathbb{E}}_{x}\mathop{\mathbb{E}}_{a \sim \bar{\pi}|x}[(Q(x,a) - \hat{Q}(x,a))^2]} + \sqrt{\mathop{\mathbb{E}}_{x}\mathop{\mathbb{E}}_{a \sim \hat{\pi}|x}[(Q(x,a) - \hat{Q}(x,a))^2]} && (6.14)\\
&\leq 2\sqrt{\sup_{\pi \in \Pi} \mathop{\mathbb{E}}_{x}\left[\mathop{\mathbb{E}}_{a \sim \pi|x}[(Q(x,a) - \hat{Q}(x,a))^2]\right]} && (6.15)
\end{aligned}$$

The first inequality holds since $\hat{\pi}$ maximizes $\hat{Q}$ and by using the definition of absolute value, the second by Jensen, and the third by introducing the supremum. □

{thmt@@transfer@data}
{thmt@@transfer}

**Lemma 6.8** (Transfer: from $\beta$ to $\pi$)**.** *Assume strict positivity and take any Q-function $\hat{Q}$ and any policy $\pi$, then*

$$\mathop{\mathbb{E}}_{x,a \sim \mathcal{D},\pi}[Q(x,a) - \hat{Q}(x,a))^2] < \frac{1}{\tau}\left(\mathop{\mathbb{E}}_{x,a \sim \mathcal{D},\beta}[(Q(x,a) - \hat{Q}(x,a))^2]\right). \tag{6.16}$$

*Proof.* Let $\pi$ be any policy. Then

$$\mathop{\mathbb{E}}_{x} \mathop{\mathbb{E}}_{a\sim\pi|x}[(Q(x,a)-\hat{Q}(x,a))^2] = \int_x p(x)\sum_a \pi(a|x)(Q(x,a)-\hat{Q}(x,a))^2 dx \tag{6.17}$$

$$= \int_x \sum_a \pi(a|x)\frac{\beta(a|x)}{\beta(a|x)}p(x)(Q(x,a)-\hat{Q}(x,a))^2 dx \tag{6.18}$$

$$< \frac{1}{\tau}\int_x \sum_a \beta(a|x)p(x)(Q(x,a)-\hat{Q}(x,a))^2 dx \tag{6.19}$$

$$= \frac{1}{\tau}\mathop{\mathbb{E}}_{x,a\sim\mathcal{D},\beta}[(Q(x,a)-\hat{Q}(x,a))^2] \tag{6.20}$$

where we use a multiply and divide trick and apply the definition of strict positivity to ensure that $\frac{\pi(a|x)}{\beta(a|x)} < \frac{1}{\tau}$. □

# Appendix 6.C Policy-based learning

{app:pb}

## 6.C.1 In-sample regret

{lem:policy_decomp}

**Lemma 6.9.** *Let $\Pi$ be an interpolating class and $K = 2$. Then there exists a $\pi_B$ as defined in Equation (6.4) such that*

1. *the behavior of $\pi_B$ at each datapoint $x_i \in S$ only depends on $a_i, r_i(a_i)$, and $p_i$*
2. *$\pi_B(\cdot|x_i) \in \{(0,1),(1,0)\}$.*

*Proof.* We will begin by proving part 2. Note that the objective that $\pi_B$ optimizes takes the form $\frac{r_i(a_i)}{p_i}\pi(a_i|x_i)$ at each datapoint. Since probabilities are constrained to $[0,1]$ this is optimized by $\pi(a_i|x_i) = 0$ if $\frac{r_i(a_i)}{p_i} < 0$ and $\pi(a_i|x_i) = 1$ if $\frac{r_i(a_i)}{p_i} > 0$. Since we have an overparameterized model class, we know that $\Pi$ contains a $\pi_B$ that can exactly choose the optimizer at each datapoint. Since $K = 2$, once we know $\pi(a_i|x_i)$ we immediately have $\pi(\bar{a}_i|x_i) = 1 - \pi(a_i|x_i)$ (where $\hat{a}_i$ is the action that is not equal to $a_i$). Thus $\pi_B(\cdot|x_i) \in \{(0,1),(1,0)\}$.

Now part 1 follows directly since the above reasoning showed that $\pi_B(\cdot|x_i)$ is defined precisely by the sign of $\frac{r_i(a_i)}{p_i}$ and the identity of $a_i$. □

**Theorem 6.5** (In-sample regret lower bound)**.** *Let $K = 2$ and the policy class be overparameterized. Define $\Delta_r(x) = \left|\mathbb{E}_{r|x}[r(1) - r(2)]\right|$ as the absolute expected gap in rewards at $x$. Define $p_u(x)$ to be the probability that the policy-based objective is action-unstable at $x$. Recall that $\beta(a|x) \geq \tau$ by Assumption 1. Then*

$$\mathop{\mathbb{E}}_{S}[V(\pi^*; S) - V(\pi_B; S)] \geq \tau \mathop{\mathbb{E}}_{x}\left[p_u(x)\Delta_r(x)\right].$$

*Proof.* By part 1 of Lemma 6.9 and linearity of expectation we can decompose the expected in-sample value as

$$\mathop{\mathbb{E}}_{S}[V(\pi^*; S) - V(\pi_B; S)] = \frac{1}{N}\sum_{i=1}^{N} \mathop{\mathbb{E}}_{x_i,r_i,a_i}\left[\mathop{\mathbb{E}}_{a\sim\pi^*}\mathop{\mathbb{E}}_{r|x_i}[r(a)] - \mathop{\mathbb{E}}_{a\sim\pi_B}\mathop{\mathbb{E}}_{r|x_i}[r(a)]\right].$$

Since the data are iid we further have that

$$\mathop{\mathbb{E}}_{S}[V(\pi^*; S) - V(\pi_B; S)] = \mathop{\mathbb{E}}_{x_i,r_i,a_i}\left[\mathop{\mathbb{E}}_{a\sim\pi^*}\mathop{\mathbb{E}}_{r|x_i}[r(a)] - \mathop{\mathbb{E}}_{a\sim\pi_B}\mathop{\mathbb{E}}_{r|x_i}[r(a)]\right].$$

Define the event $U_{x,r}$ to be the event that the policy-based objective is action-unstable at $x, r$. So $p_u(x) = \mathbb{E}_{r|x}[\mathbb{1}[U_{x,r}]]$. We can split this expectation up into stable and unstable parts by conditioning on either $\overline{U}_{x_i,r_i}$ or $U_{x_i,r_i}$, and lower bound the regret on the stable datapoints by 0:

$$\begin{aligned}
\mathop{\mathbb{E}}_{S}[V(\pi^*; S) - V(\pi_B; S)] = &\mathop{\mathbb{E}}_{x_i,r_i|\overline{U}_{x_i,r_i}}\mathop{\mathbb{E}}_{a_i|x_i}\left[\mathop{\mathbb{E}}_{a\sim\pi^*}\mathop{\mathbb{E}}_{r|x_i}[r(a)] - \mathop{\mathbb{E}}_{a\sim\pi_B}\mathop{\mathbb{E}}_{r|x_i}[r(a)]\right] \\
&+ \mathop{\mathbb{E}}_{x_i,r_i|U_{x_i,r_i}}\mathop{\mathbb{E}}_{a_i|x_i}\left[\mathop{\mathbb{E}}_{a\sim\pi^*}\mathop{\mathbb{E}}_{r|x_i}[r(a)] - \mathop{\mathbb{E}}_{a\sim\pi_B}\mathop{\mathbb{E}}_{r|x_i}[r(a)]\right] \\
\geq &\mathop{\mathbb{E}}_{x_i,r_i|U_{x_i,r_i}}\mathop{\mathbb{E}}_{a_i|x_i}\left[\mathop{\mathbb{E}}_{a\sim\pi^*}\mathop{\mathbb{E}}_{r|x_i}[r(a)] - \mathop{\mathbb{E}}_{a\sim\pi_B}\mathop{\mathbb{E}}_{r|x_i}[r(a)]\right].
\end{aligned}$$

By part 2 of Lemma 6.9 we know that $\pi_B(\cdot|x_i)$ is either $(1,0)$ or $(0,1)$. Conditioned on the objective being unstable at $x_i$ and using the fact that there are only two actions, we know that $\pi_B(x_i)$ must be different depending on whether $a_i = 1$ or $a_i = 2$. Define $a^1_{i,B}$ to be the action that $\pi_B$ selects at $x_i$ when $a_i = 1$ and $a^2_{i,B}$ the action when $a_i = 2$. Let $a^*_i$ be the action chosen by the deterministic optimal policy $\pi^*$ at $x_i$. Thus we can split the expectation over $a_i$ in the above expression and then plug in definitions to get:

$$\mathop{\mathbb{E}}_{S}[V(\pi^*;S) - V(\pi_B;S)] \geq \mathop{\mathbb{E}}_{x_i,r_i|U_{x_i,r_i}}\left[\beta(a_i = 1|x_i)\mathop{\mathbb{E}}_{r|x_i}[r(a^*_i) - r(a^1_{i,B})] + \beta(a_i = 2|x_i)\mathop{\mathbb{E}}_{r|x_i}[r(a^*_i) - r(a^2_{i,B})]\right].$$

Since we assumed that $\beta(a|x_i) \geq \tau$ for all $a$ we can lower bound the above by

$$\mathop{\mathbb{E}}_{S}[V(\pi^*;S) - V(\pi_B;S)] \geq \tau \mathop{\mathbb{E}}_{x_i,r_i}\left[\mathbb{1}[U_{x_i,r_i}]\left(\mathop{\mathbb{E}}_{r|x_i}[r(a^*_i) - r(a^1_{i,B})] + \mathop{\mathbb{E}}_{r|x_i}[r(a^*_i) - r(a^2_{i,B})]\right)\right].$$

Finally, we note that since $a^1_{i,B} \neq a^2_{i,B}$ and there are only 2 actions that the above is precisely

$$\begin{aligned}
\mathop{\mathbb{E}}_{S}[V(\pi^*;S) - V(\pi_B;S)] &\geq \tau \mathop{\mathbb{E}}_{x_i,r_i}\left[\mathbb{1}[\overline{E}_{x_i,r_i}]\mathop{\mathbb{E}}_{r|x_i}[r(a^*_i) - r(a \neq a^*_i)]\right] \\
&= \tau \mathop{\mathbb{E}}_{x_i,r_i}[\mathbb{1}[\overline{E}_{x_i,r_i}]\Delta_r(x_i)] \\
&= \tau \mathop{\mathbb{E}}_{x_i}[\mathop{\mathbb{E}}_{r_i|x_i}[\mathbb{1}[U_{x_i,r_i}]]\Delta_r(x_i)] \\
&= \tau \mathop{\mathbb{E}}_{x_i}[p_u(x_i)\Delta_r(x_i)] \\
&= \tau \mathop{\mathbb{E}}_{x}[p_u(x)\Delta_r(x)].
\end{aligned}$$

□

### 6.C.2 Connection to noisy classification

{app:noisy}

This section states and proves the Theorem referenced in the main text connecting action-unstable policy-based learning with noisy classification.

{thmt@@noisy@data}
{thmt@@noisy}

**Theorem 6.10** (Noisy classification reduction)**.** *Take any noise level $\eta < 1/2$ and any binary classification problem $C$ consisting of a distribution $\mathcal{D}_C$ over $\mathcal{X}$ and a labeling function $y_C : \mathcal{X} \to \{-1, 1\}$. There exists an offline contextual bandit problem $\mathcal{B}$ with noiseless rewards such that*

1. *Maximizing $\hat{V}_B$ in $\mathcal{B}$ is equivalent to minimizing the 0/1 loss on a training set drawn from $C$ where labels are flipped with probability $\eta$.*
2. *Maximizing $\hat{V}_F$ in $\mathcal{B}$ is equivalent to minimizing the 0/1 loss on a training set drawn from $C$ with noiseless training labels.*

*Proof.* First we will construct the bandit problem $\mathcal{B}$ with two actions corresponding to the classification problem $C$. For any constant $c_r > 0$ we define $\mathcal{B}$ by

$$x \sim \mathcal{D}_C, \qquad r|x = \begin{cases} c_r(1-\eta, \eta) & y_C(x) = 1 \\ c_r(\eta, 1-\eta) & y_C(x) = -1 \end{cases}, \qquad \beta(1|x) = \begin{cases} 1-\eta & y_C(x) = 1 \\ \eta & y_C(x) = -1 \end{cases} \tag{6.21}$$

Now we will show that in this problem, $\hat{V}_B$ is equivalent to the 0/1 loss for $C$ with noisy labels. To do this first note that by construction, for $x$ with $y_C(x) = 1$ we have $\frac{r(1)|x}{\beta(1|x)} = \frac{c_r(1-\eta)}{1-\eta} = c_r$ and $\frac{r(2)|x}{\beta(2|x)} = \frac{c_r\eta}{\eta} = c_r$, and similarly for $x$ with $y_C(x) = -1$ we have $\frac{r(1)|x}{\beta(1|x)} = \frac{c_r\eta}{\eta} = c_r$ and $\frac{r(2)|x}{\beta(2|x)} = \frac{c_r(1-\eta)}{1-\eta} = c_r$.

$$\hat{V}_B(\pi) = \frac{1}{N}\sum_{i=1}^{N} r_i(a_i)\frac{\pi(a_i|x_i)}{\beta(a_i|x_i)} = \frac{1}{N}\sum_{i=1}^{N}\frac{r_i(a_i)}{\beta(a_i|x_i)}\pi(a_i|x_i) \tag{6.22}$$

$$= \frac{c_r}{N}\sum_{i=1}^{N}\pi(a_i|x_i) \tag{6.23}$$

This is equivalent to 0/1 loss with noisy labels since $\beta$ generates $a_i$ according to $y_C$ where the label is flipped with probability $\eta$.

Now we will show that $\hat{V}_F$ is equivalent to the 0/1 loss for $C$ with clean labels. Note that by construction $r(a)|x = c_r\eta + \pi^*(a|x)c_r(1-2\eta)$. So,

$$\hat{V}_F(\pi) = \frac{1}{N}\sum_{i=1}^{N}\langle r_i, \pi(\cdot|x_i)\rangle = \frac{c_r}{N}\sum_{i=1}^{N}\langle \eta\mathbf{1} + (1-2\eta)\pi^*(\cdot|x_i), \pi(\cdot|x_i)\rangle \tag{6.24}$$

$$= \frac{c_r\eta}{N} + \frac{c_r(1-2\eta)}{N}\sum_{i=1}^{N}\langle \pi^*(\cdot|x_i), \pi(\cdot|x_i)\rangle \tag{6.25}$$

This is equivalent to 0/1 loss with noisy labels since $\pi^*$ exactly corresponds to $y_C$. □

### 6.C.3 Nearest Neighbor

**Theorem 6.6** (Regret lower bound for one nearest neighbor)**.** *Let $\Delta_r = r_{\max} - r_{\min}$. Then there exist problem instances with noiseless rewards where*

$$\limsup_{N\to\infty} \mathop{\mathbb{E}}_{S}[V(\pi_F) - V(\pi_B)] = \frac{\Delta_r}{2},$$

*but*

$$\limsup_{N\to\infty} \mathop{\mathbb{E}}_{S}[V(\pi^*) - V(\pi_F)] = 0.$$

*Proof.* First we need to formally define the nearest neighbor rules that interpolate the objectives $\hat{V}_B$ and $\hat{V}_F$. These are simple in the case of two actions. Let $i(x)$ be the index of the nearest

neighbor to $x$ in the dataset. Then

$$\pi_B(a|x) = \begin{cases} 1 & \big(a = a_{i(x)} \text{ AND } r_{i(x)}(a_{i(x)}) > 0\big) \text{OR} \big(a \neq a_{i(x)} \text{ AND } r_{i(x)}(a_{i(x)}) \leq 0\big) \\ 0 & otherwise. \end{cases} \tag{6.26}$$

This is saying that $\pi_B$ chooses the same action as the observed nearest neighbor if that reward was positive, and the opposite action if that was negative. And for the full feedback we just choose the best action from the nearest datapoint.

$$\pi_F(a|x) = \begin{cases} 1 & a = \arg\max_{a'} r_{i(x)}(a') \\ 0 & otherwise. \end{cases} \tag{6.27}$$

Now we can construct the problem instances needed for the Theorem. To construct the example, take a bandit problem with two actions (called 1 and 2):

$$x \sim U([-1, 1]), \quad r|x = (1, 1 + \Delta_r), \quad \beta(1|x) = \beta(2|x) = 1/2 \; \forall \; x, a$$

The true optimal policy has $\pi^*(2|x) = 1$ for all $x$ and $V(\pi^*) = 1+\Delta_r$. The policy with full feedback $\pi_F$ is to always choose action 2, since every observation will show that action 2 is better.

Now, we will show that in the limit of infinite data, $\pi_F$ has no regret. Since the rewards are noiseless, the maximum observed reward at a context is exactly the optimal action at that context. Thus, we precisely have a classification problem with noiseless labels so that the Bayes risk is 0. Since we $\pi^*$ is continuous, the class conditional densities (determined by the indicator of the argmax of $Q$) are piecewise continuous. This allows us to apply the classic result of [Cover and Hart 1967] that a nearest neighbor rule has asymptotic risk less twice the Bayes risk, which in this case is zero. This means that asymptotically $P(\pi_F(a|x) \neq \pi^*(a|x)) = 0$ which immediately

gives the second desired result of zero regret in the limit of infinite data under full feedback.

Now we note that since rewards are always positive, we can simplify the definition of $\pi_B$ as

$$\pi_B(a|x) = \mathbb{1}[a = a_{i(x)}]. \tag{6.28}$$

Then we have that

$$V(\pi_F) - V(\pi_B) = \mathop{\mathbb{E}}_{x}[\mathop{\mathbb{E}}_{a\sim\pi_F|x}[Q(x,a)] - \mathop{\mathbb{E}}_{a\sim\pi_B|x}[Q(x,a)]] \tag{6.29}$$

$$= \mathop{\mathbb{E}}_{x}[\Delta_r + 1 - (\pi_B(1|x) + \pi_B(2|x)(\Delta_r + 1))]] \tag{6.30}$$

$$= \Delta_r + 1 - \mathop{\mathbb{E}}_{x}[\mathbb{1}[a_{i(x)} = 1] + (\Delta_r + 1)\mathbb{1}[a_{i(x)} = 2]] \tag{6.31}$$

Taking expectation over $S$ we get

$$\mathop{\mathbb{E}}_{S}[V(\pi_F) - V(\pi_B)] = \mathop{\mathbb{E}}_{S}[\Delta_r + 1 - \mathop{\mathbb{E}}_{x}[\mathbb{1}[a_{i(x)} = 1] + (\Delta_r + 1)\mathbb{1}[a_{i(x)} = 2]]] \tag{6.32}$$

$$= \Delta_r + 1 - \mathop{\mathbb{E}}_{x}[P_S(a_{i(x)} = 1) + (\Delta_r + 1)P_S(a_{i(x)} = 2)]] \tag{6.33}$$

$$= \Delta_r + 1 - \mathop{\mathbb{E}}_{x}[\frac{1}{2} + (\Delta_r + 1)\frac{1}{2}]] \tag{6.34}$$

$$= \frac{\Delta_r}{2} \tag{6.35}$$

This construction did not depend on the size of the dataset, so it is even true as the number of datapoints tends to infinity. □

## Appendix 6.D Discussion of doubly robust algorithms

{app:dr}

Before going into the comparison, we will define the doubly robust algorithm [Dudík et al. 2011] in our notation. Specifically,

$$\widehat{V}_{DR}(\pi) := \sum_{i=1}^{N} \left[ \sum_{a} \pi(a|x_i)\hat{Q}(x_i, a) + \frac{\pi(a_i|x_i)}{\beta(a_i|x_i)}(r_i(a_i) - \hat{Q}(x_i, a_i)) \right], \qquad \hat{\pi}_{DR} = \arg\max_{\pi \in \Pi} \widehat{V}_{DR}(\pi) \tag{6.36}$$

{{eq:dr}}

As stated in the main text, when we use overparameterized models and train $\hat{Q}$ on the same data that we use to optimize the policy, then doubly robust methods are equivalent to the vanilla value-based algorithm. This is formalized in Lemma 6.11 below.

This equivalence can be avoided by using crossfitting so that $\hat{Q}$ is not trained on the same data as $\pi$. However, then it is possible that the doubly robust policy objective becomes action-unstable. This is true *even with* access to the true $Q$ function, but requires stochastic rewards. To construct such an example we leverage the stochastic rewards so that instability only occurs at datapoints where certain reward vectors are sampled. This is shown in Lemma 6.12 below.

One final point is to consider the motivation for doubly robust methods. Usually it is motivated by concerns about consistency of the value function estimation or estimation of behavior policy [Dudík et al. 2011]. However, in our setting we have (1) an overparamterized model class which is large enough to contain the true value function, and (2) exact access to the behavior probabilities. So it is not clear why doubly robust methods would be motivated in our setting.

{lem:dr_equiv}

**Lemma 6.11** (Equivalence of DR and vanilla VB)**.** *When we use overparameterized models and do not use crossfitting, doubly robust learning from Equation (6.36) is equivalent to vanilla value-based learning from Equation (6.5).*

*Proof.* When the model for $\hat{Q}$ is overparameterized and trained on the full dataset, we know that

$\hat{Q}(x_i, a_i) = r_i(a_i)$. Thus we get that

$$\widehat{V}_{DR}(\pi) = \sum_{i=1}^{N} \left[ \sum_a \pi(a|x_i)\hat{Q}(x_i, a) + \frac{\pi(a_i|x_i)}{\beta(a_i|x_i)}(r_i(a_i) - \hat{Q}(x_i, a_i)) \right] \tag{6.37}$$

$$= \sum_{i=1}^{N} \left[ \sum_a \pi(a|x_i)\hat{Q}(x_i, a) + \frac{\pi(a_i|x_i)}{\beta(a_i|x_i)}(0) \right] \tag{6.38}$$

$$= \sum_{i=1}^{N} \sum_a \pi(a|x_i)\hat{Q}(x_i, a) \tag{6.39}$$

With an overparameterized policy class, we can exactly recover the greedy policy relative to $\hat{Q}$ to optimize this objective. □

{lem:dr_unstable}

**Lemma 6.12** (Instability of DR)**.** *There exist problems with stochastic rewards where even with access to the exact Q function, the doubly robust policy objective is action-unstable with probability 1/2.*

*Proof.* We need only consider one datapoint since the action-stability property is defined on a per datapoint basis. To make this construction we will consider only two actions.

$$r|x = \begin{cases} (0, 1) & w.p.\ 1/2 \\ (0, -2) & otherwise \end{cases}, \qquad \beta(\cdot|x) = (1/2, 1/2) \tag{6.40}$$

So, we know that

$$Q(\cdot|x) = (0, -0.5) \tag{6.41}$$

Now we claim that when the sampled datapoint has $r = (0, 1)$ the doubly robust objective is action-unstable (and this happens with probability 1/2 by construction). We can explicitly expand

the DR objective for the policy $\pi$ at $x$ when action $a$ is sampled

$$\ell_{DR}(\pi, x, a, r) = \pi(1|x) \cdot 0 + \pi(2|x) \cdot (-0.5) + \frac{\pi(a|x)}{1/2}(r(a) - Q(x, a)) \tag{6.42}$$

So when $a = 1$ we have $r(a) = 0$ and $Q(x, a) = 0$ so that

$$\ell_{DR}(\pi, x, a, r) = \pi(2|x) \cdot (-0.5) + 2 \cdot \pi(1|x)(0 - 0) = \pi(2|x) \cdot (-0.5) \tag{6.43}$$

And when $a = 2$ we have $r(a) = 1$ (because that was the sampled reward) and $Q(x, a) = 0$ so that

$$\ell_{DR}(\pi, x, a, r) = \pi(2|x) \cdot (-0.5) + 2 \cdot \pi(2|x)(1 - (-0.5)) = \pi(2|x) \cdot (2.5) \tag{6.44}$$

Now, this is clearly action-unstable since the optimizer when $a = 1$ is sampled is $\pi(\cdot|x) = (1, 0)$ while when $a = 2$ is sampled we get $\pi(\cdot|x) = (0, 1)$. □

## Appendix 6.E Details of Bandit Experiments

{app:experiments}

### 6.E.1 Synthetic data

Data. As described in the main text we sample some hidden reward matrix $\theta$ and then sample contexts and rewards from isotropic Gaussians:

$$\theta \sim U([0, 1]^{K \times d}), \quad x \sim \mathcal{N}(0, I_d), \quad r \sim \mathcal{N}(\theta x, \epsilon I_d).$$

Actions are sampled according to a uniform behavior:

$$a \sim \beta(\cdot|x) = U(\{1, \ldots, K\}).$$

We set $K = 2, d = 10, \epsilon = 0.1$. For each random seed we take $N = 100$ training points and sample an independent test set of 500 points. For experiment 1 we sample $\theta$ and one dataset of $x, r$ tuples, then we sample 20 independent sets of actions. For experiment 2 we sample all parameters separately to construct each of the 50 datasets.

Model. For policies and Q functions we use a multilayer perceptron with one hidden layer of width 512 and ReLU activations. The only difference between policy and Q architecture is that the policy has a softmax layer on the output so that the output is a probability distribution.

Learning. We train using SGD with momentum. Learning rate is 0.01, momentum is 0.9, batch size is 10, and weight decay is 0.0001. We train every model for 1000 epochs decreasing the learning rate by a factor of 10 after 200 epochs. This trains well past the point of convergence in our experience.

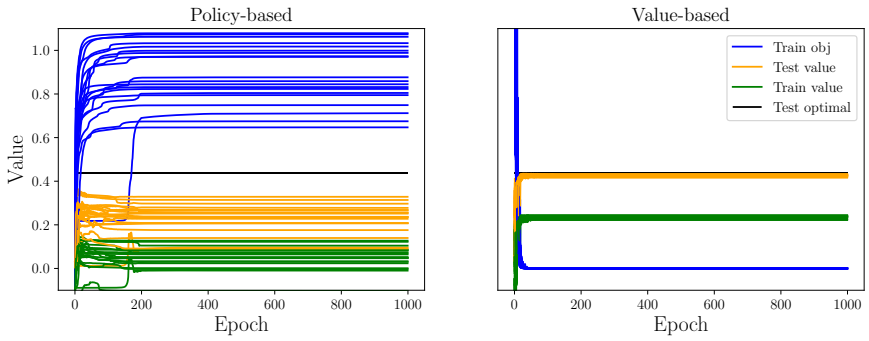



**Figure 6.4:** We show learning curves across each of the twenty different action resampled datasets. {fig:toy_learning}

Extended results. Figure 6.4 shows learning curves for each of the twenty different action datasets from experiment 1. We use "train obj" to refer to the training objective which is squared error for value-based learning and $\hat{V}_B$ for policy-based learning. We use "train value" and "test value" to refer to $V(\pi; S)$ for $S$ being the train and test sets respectively. We can evaluate the true value at each datapoint since we know the full reward vector at each datapoint.

We see that the policy-based objective is dramatically higher than the highest achievable value due to overfitting of the noise in the actions. The gap between train and test value is mot likely explained by noise in the contexts sampled in those respective datasets (by chance the test set has higher value contexts).

### 6.E.2 CIFAR-10

DATA. We use a bandit version of the CIFAR-10 dataset [Krizhevsky 2009]. We split the train set into a train set of the first 45000 examples and validation set of the last 5000. We normalize the images and use data augmentation of random flips and crops of size 32. Each of the 10 labels becomes an action. We define rewards to be 1 for a correct prediction and 0 for an incorrect prediction. We use two different behavior policies. One is a uniform behavior that selects each action with probability 0.1 and the other is the hand-crafted behavior policy from [Joachims et al. 2018].

MODEL. We use a ResNet-18 [He et al. 2016] from PyTorch [Paszke et al. 2019b] for both the policy and the Q function. The only modification we make to accommodate for the smaller images in CIFAR is to remove the first max-pooling layer.

LEARNING. We train using SGD with momentum 0.9,a batch size 128, and weight decay of 0.0001 for 1000 epochs. Training takes about 20 hours for each run on an NVIDIA RTX 2080 Ti GPU. We use a learning rate of 0.1 for the first 200 epochs, 0.01 for the next 200, and 0.001 for the last 600. To improve stability we use gradient clipping and reduce the learning rate in the very first epoch to 0.01.

EXTENDED RESULTS. Figures 6.5 and 6.6 show learning curves for each of the three algorithms we consider across each dataset. The labels refer to the same quantities as they did on the synthetic problem.

One interesting phenomena is that the unstable policy-based algorithm displays a clear overfitting phenomena as we would predict due to the noise in the actions being transferred into noise in the objective. Since we have strictly positive rewards here, this is also an instance of "propensity overfitting" [Swaminathan and Joachims 2015b]. As a result, limiting the capacity of the model class by early stopping could improve performance somewhat. But by limiting capacity in this way we are exiting the overparameterized/interpolating regime described by Zhang et al. [2016].

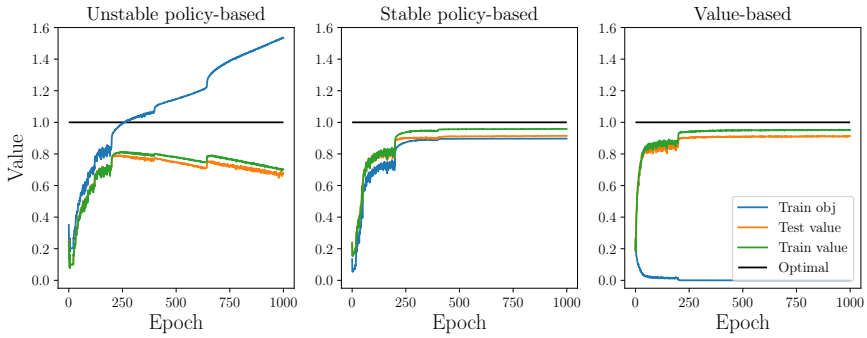



**Figure 6.5:** Learning curves on the hand-crafted action dataset. {fig:cifar_learning_b

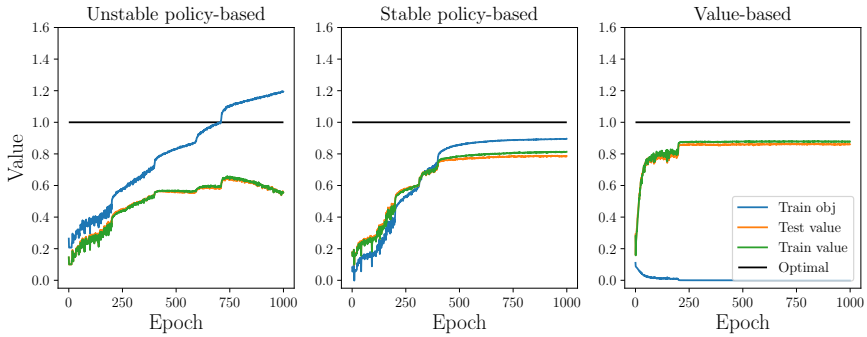



**Figure 6.6:** Learning curves on the uniform action dataset. {fig:cifar_learning_u

## Appendix 6.F Small model classes

{app:small}

In this section we state and prove theorems that bound each term of our regret decomposition for each algorithm we consider when we use finite model classes. Similar results can be shown for

other classical notions of model class complexity. We include these results for completeness, but the main focus of our paper is the overparameterized regime where such bounds are vacuous.

{thm:pol-small}

**Theorem 6.13** (Policy-based learning with a small model class)**.** *Assume strict positivity and a finite policy class* $\Pi$*. Let* $\varepsilon_\Pi = V(\pi^*) - \sup_{\pi\in\Pi} V(\pi)$*. Denote* $\Delta_r = r_{max} - r_{min}$*. Then we have that for any* $\delta > 0$ *with probability* $1-\delta$ *each of the following holds:*

$$\begin{aligned}
\textit{Approximation Error} &= V(\pi^*) - \sup_{\pi\in\Pi} V(\pi) \leq \varepsilon_\Pi \\
\textit{Estimation Error} &= \sup_{\pi\in\Pi} V(\pi) - V(\pi_F) \leq 2\Delta_r\sqrt{\frac{\log(2|\Pi|/\delta)}{2N}} \\
\textit{Bandit Error} &= V(\pi_F) - V(\pi_B) \leq \frac{2\Delta_r}{\tau}\sqrt{\frac{\log(2|\Pi|/\delta)}{2N}}
\end{aligned}$$

*Proof.* The bound on approximation error follows directly from the definition of $\varepsilon_\Pi$. The bound on the estimation error follows from a standard application of a Hoeffding bound on the random variables $X_i = \langle r_i, \pi(\cdot|x_i)\rangle$ which are bounded by $\Delta_r$ and a union bound over the policy class.

The bound on bandit error essentially follows Theorem 3.2 of [Strehl et al. 2010], we include a proof for completeness:

$$\begin{aligned}
V(\pi_F) - V(\pi_B) &= V(\pi_F) - \hat{V}_B(\pi_B) + \hat{V}_B(\pi_B) - V(\pi_B) \\
&\leq V(\pi_F) - \hat{V}_B(\pi_F) + \hat{V}_B(\pi_B) - V(\pi_B) \\
&\leq 2\sup_{\pi\in\Pi} |V(\pi) - \hat{V}_B(\pi)| \\
&\leq \frac{2\Delta_r}{\tau}\sqrt{\frac{\log(2|\Pi|/\delta)}{2N}}
\end{aligned}$$

The first inequality comes from the definition of $\pi_B$. The second comes since both $\pi_F, \pi_B \in \Pi$. And the last inequality follows from an application of a Hoeffding bound on the random variables $X_i = r_i(a_i)\frac{\pi(a_i|x_i)}{p_i}$ which are bounded by $\frac{\Delta_r}{\tau}$ and a union bound over the policy class. □

{thm:val-small}

**Theorem 6.14** (Value-based learning with a small model class)**.** *Assume strict positivity and a finite function class $\mathcal{Q}$ which induces a finite class of greedy policies $\Pi_{\mathcal{Q}}$. Let $\varepsilon_{\mathcal{Q}} = \inf_{\widehat{Q}\in\mathcal{Q}} \mathbb{E}_{x,a\sim\mathcal{D},\beta}[(Q(x,a)-\widehat{Q}(x,a))^2]$. Denote $\Delta_r = r_{max} - r_{min}$. Then we have that for any $\delta > 0$ with probability $1-\delta$ each of the following holds:*

$$\textit{Approximation Error} = V(\pi^*) - \sup_{\pi\in\Pi_{\mathcal{Q}}} V(\pi) \leq 2\sqrt{\varepsilon_{\mathcal{Q}}/\tau} \tag{6.45}$$

$$\textit{Estimation Error} = \sup_{\pi\in\Pi_{\mathcal{Q}}} V(\pi) - V(\pi_F) \leq 2\Delta_r\sqrt{\frac{\log(|\mathcal{Q}|/\delta)}{2N}} \tag{6.46}$$

$$\textit{Bandit Error} = V(\pi_F) - V(\pi_{\widehat{Q}}) \leq \frac{10\Delta_r}{\sqrt{\tau}}\sqrt{\frac{\log(|\mathcal{Q}|/\delta)}{N}} + 6\sqrt{\Delta_r}\left(\frac{\log(|\mathcal{Q}|/\delta)}{\tau N}\varepsilon_{\mathcal{Q}}\right)^{1/4} + 2\sqrt{\varepsilon_{\mathcal{Q}}/\tau} \tag{6.47}$$

*Proof.* To bound the approximation error, we can let $\hat{\pi}$ be the greedy policy associated with a Q-function $\widehat{Q}$ and apply Lemmas 6.7 and 6.8. This gives us

$$V(\pi^*) - \sup_{\hat{\pi}\in\Pi_{\mathcal{Q}}} V(\hat{\pi}) = \inf_{\widehat{Q}\in\mathcal{Q}} [V(\pi^*) - V(\hat{\pi})] \leq \inf_{\widehat{Q}\in\mathcal{Q}} \frac{2}{\sqrt{\tau}}\sqrt{\mathbb{E}_{x,a\sim\mathcal{D},\beta}[(Q(x,a)-\widehat{Q}(x,a))^2]} = 2\sqrt{\varepsilon_{\mathcal{Q}}/\tau}. \tag{6.48}$$

The bound on the estimation error follows the same as before from standard uniform convergence arguments.

The bound on the bandit error follows by again applying Lemmas 6.7 and 6.8 and then making the concentration argument from Lemma 16 of [Chen and Jiang 2019]. Explicitly, our Lemmas give us

$$V(\pi_F) - V(\pi_{\widehat{Q}}) \leq V(\pi^*) - V(\pi_{\widehat{Q}}) \leq \frac{2}{\sqrt{\tau}}\sqrt{\mathbb{E}_{x,a\sim\mathcal{D},\beta}[(Q(x,a)-\widehat{Q}(x,a))^2]}. \tag{6.49}$$

Then, to bound the squared error term, we can add and subtract:

$$\mathop{\mathbb{E}}_{x,a\sim\mathcal{D},\beta}[(Q(x,a)-\widehat{Q}(x,a))^2] = \mathop{\mathbb{E}}_{x,a\sim\mathcal{D},\beta}[(Q(x,a)-\widehat{Q}(x,a))^2] - \inf_{\bar{Q}\in\boldsymbol{Q}} \mathop{\mathbb{E}}_{x,a\sim\mathcal{D},\beta}[(Q(x,a)-\bar{Q}(x,a))^2] \tag{6.50}$$

$$+ \inf_{\bar{Q}\in\boldsymbol{Q}} \mathop{\mathbb{E}}_{x,a\sim\mathcal{D},\beta}[(Q(x,a)-\bar{Q}(x,a))^2] \tag{6.51}$$

$$\leq \mathop{\mathbb{E}}_{x,a\sim\mathcal{D},\beta}[(Q(x,a)-\widehat{Q}(x,a))^2] - \inf_{\bar{Q}\in\boldsymbol{Q}} \mathop{\mathbb{E}}_{x,a\sim\mathcal{D},\beta}[(Q(x,a)-\bar{Q}(x,a))^2] \tag{6.52}$$

$$+ \varepsilon_{\boldsymbol{Q}}. \tag{6.53}$$

Now we want to show that the difference in squared error terms concentrates for large $N$. This is precisely what Lemma 16 from [Chen and Jiang 2019] does using a one-sided Bernstein inequality. This gives us for any $\delta > 0$ an upper bound with probability $1-\delta$ of

$$\frac{56\Delta_r^2 \log(|\boldsymbol{Q}|/\delta)}{3N} + \sqrt{\varepsilon_{\boldsymbol{Q}} \frac{32\Delta_r^2 \log(|\boldsymbol{Q}|/\delta)}{N}} \tag{6.54}$$

Plugging this in and simplifying the constants gives the result. □

# Notes

17. When using neural networks Q is usually implemented as a function of $x$ with $K$ outputs [Mnih et al. 2015a].

18. Code can be found at `https://github.com/davidbrandfonbrener/deep-offline-bandits`.

19. This corresponds to the optimal value of $\lambda$ in the experiments of Joachims et al. [2018]. Our "stable" model slightly outperforms theirs, likely due to a slightly better implementation.

# 7 | Offline RL Without Off-Policy Evaluation

{sec:offline-rl}

## 7.1 Introduction

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

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

We then dive deeper to understand why such a simple baseline is effective. First, we examine

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

Finally, we recognize that while the the one-step algorithm is a strong baseline, it is not always the best choice. In the final section we provide some guidance about when iterative algorithms can perform better than the simple one-step baseline. Namely, when the dataset is large and behavior policy has good coverage of the state-action space, then off-policy evaluation can succeed and iterative algorithms can be effective. In contrast, if the behavior policy is already fairly good, but as a result does not have full coverage, then one-step algorithms are often preferable.

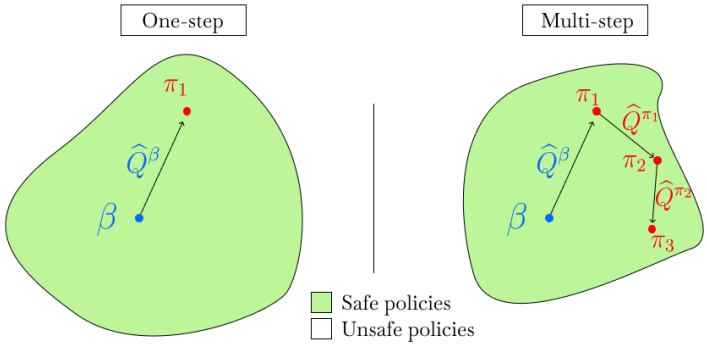



**Figure 7.1:** A cartoon illustration of the difference between one-step and multi-step methods. All algorithms constrain themselves to a neighborhood of "safe" policies around $\beta$. A one-step approach (left) only uses the on-policy $\widehat{Q}^\beta$, while a multi-step approach (right) repeatedly uses off-policy $\widehat{Q}^{\pi_i}$. {fig:cartoon}

Our main contributions are:

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

- A description of when one-step algorithms are likely to outperform iterative approaches.

## 7.2 Setting and notation

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

$$J(\pi) := \mathop{\mathbb{E}}_{\rho,P,\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right] = \mathop{\mathbb{E}}_{\substack{s\sim\rho \\ a\sim\pi|s}}[Q^\pi(s,a)]. \tag{7.1}$$

Following Fu et al. [2020] and others in this line of work, we allow access to the environment to tune a small (< 10) set of hyperparameters. See Paine et al. [2020] for a discussion of the active area of research on hyperparameter tuning for offline RL. We also discuss this further in Appendix 7.C.

## 7.3 Related work

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

Q regularization:

1. The majority of prior work acts directly on the policy. Some authors have proposed explicit constraints on the learned policy to only select actions where $(s, a)$ has sufficient support under the data generating distribution [Fujimoto et al. 2018b, 2019; Laroche et al. 2019]. Another proposal is to regularize the learned policy towards the behavior policy [Wu et al. 2019] usually either with a KL divergence [Jaques et al. 2019] or MMD [Kumar et al. 2019]. This is a very straighforward way to stay close to the behavior with a hyperparameter that determines just how close. All of these algorithms are iterative and rely on off-policy evaluation.

2. [Siegel et al. 2020; Wang et al. 2020b; Chen et al. 2020b] all use algorithms that filter out datapoints with low Q values and then perform imitation learning. [Wang et al. 2018; Peng et al. 2019] use a weighted imitation learning algorithm where the weights are determined by exponentiated Q values. These algorithms are iterative.

3. Another way to prevent the learned policy from choosing unknown actions is to incorporate some form of regularization to encourage staying near the behavior and being pessimistic about unknown state, action pairs [Wu et al. 2019; Nachum et al. 2019; Kumar et al. 2020; Kostrikov et al. 2021]. However, properly being able to quantify uncertainty about unknown states is notoriously difficult when dealing with neural network value functions [Buckman et al. 2020]. Again all of these algorithms are iterative.

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

There are also important connections between the one-step algorithm and the literature on conservative policy improvement [Kakade and Langford 2002; Schulman et al. 2015; Achiam et al. 2017], which we discuss in more detail in Appendix 7.B.

## 7.4 Defining the algorithms

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

### 7.4.1 Algorithmic template

We consider a generic offline approximate modified policy iteration (OAMPI) scheme, shown in Algorithm 3. Essentially the algorithm alternates between two steps. First, there is a policy evaluation step where we estimate the Q function of the current policy $\pi_{k-1}$ by $\widehat{Q}^{\pi_{k-1}}$ using only the dataset $D_N$. Implementations also often use the prior Q estimate $\widehat{Q}^{\pi_{k-2}}$ to warm-start the approximation process. Second, there is a policy improvement step. This step takes in the estimated Q function $\widehat{Q}^{\pi_{k-1}}$, the estimated behavior $\hat{\beta}$, and the dataset $D_N$ and produces a new policy $\pi_k$. Again an algorithm may use $\pi_{k-1}$ to warm-start the optimization. Moreover, we expect this improvement step to be regularized or constrained to ensure that $\pi_k$ remains in the support of $\beta$ and $D_N$. Choices for this regularization/constraint are discussed below. Now we discuss a few ways to instantiate the template.

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

**Algorithm 3** OAMPI

{alg:oapi}

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

1: Set $\pi_0 = \hat{\beta}$. Initialize $\widehat{Q}^{\pi_{-1}}$ randomly.
2: **for** k = 1, …, K **do**
3: Policy evaluation: $\widehat{Q}^{\pi_{k-1}} = \boldsymbol{Q}(\pi_{k-1}, D_N, \widehat{Q}^{\pi_{k-2}})$
4: Policy improvement: $\pi_k = \mathcal{I}(\widehat{Q}^{\pi_{k-1}}, \hat{\beta}, D_N, \pi_{k-1})$

to find the resulting $\pi_1$.

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

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

### 7.4.2 Policy evaluation operators

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

algorithms looks like

$$Q(\pi_{k-1}, D_N, \widehat{Q}^{\pi_{k-2}}) = \arg\min_Q \sum_{i=1}^{N} \left( r(s_i, a_i) + \gamma \mathop{\mathbb{E}}_{a' \sim \pi_{k-1}|s'_i} Q(s'_i, a') - Q(s_i, a_i) \right)^2, \tag{7.2}$$

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

### 7.4.3 Policy improvement operators

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

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

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

$$\hat{\pi}_k^M(a|s) = \mathbb{1}[a = \arg\max_{a_j} \{\widehat{Q}^{\pi_{k-1}}(s, a_j) : a_j \sim \pi_{k-1}(\cdot|s),\ 1 \le j \le M\}]. \tag{7.3}$$

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

$$\hat{\pi}_k^{\alpha} = \arg\max_{\pi} \sum_i \mathop{\mathbb{E}}_{a\sim\pi|s} [\widehat{Q}^{\pi_{k-1}}(s_i, a)] - \alpha D(\hat{\beta}(\cdot|s_i), \pi(\cdot|s_i)) \tag{7.4}$$

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

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

$$\hat{\pi}_k^{\tau} = \arg\max_{\pi} \sum_i \exp(\tau(\widehat{Q}^{\pi_{k-1}}(s_i, a_i) - \widehat{V}(s_i))) \log \pi(a_i|s_i). \tag{7.5}$$

## 7.5 Benchmark Results

{sec:bench}

Our main empirical finding is that one step of policy improvement is sufficient to beat state of the art results on much of the D4RL benchmark suite [Fu et al. 2020]. This is striking since prior work focuses on iteratively estimating the Q function of the current policy iterate, but we only use one-step derived from $\widehat{Q}^{\beta}$. Results are shown in Table 7.1. Full experimental details are in Appendix 7.C.

As we can see in the table, all of the one-step algorithms usually outperform the best itera-

**Table 7.1:** Results of one-step algorithms on the D4RL benchmark. The first column gives the best results across several iterative algorithms considered in [Fu et al. 2020]. We run 3 seeds and each algorithm is tuned over 6 values of their respective hyperparameter. We report the mean and standard deviation over seeds on 100 evaluation episodes per seed. We **bold** the best result on each dataset and blue any result where a one-step algorithm beat the best reported iterative result from [Fu et al. 2020]. We use m for medium, m-e for medium-expert, m-re for medium-replay, r for random, and c for cloned.

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

{tab:d4rl}

tive algorithms tested by Fu et al. [2020]. The one notable exception is the case of random data (especially on halfcheetah), where iterative algorithms have a clear advantage. We will discuss potential causes of this further in Section 7.7.

To give a more direct comparison that controls for any potential implementation details, we use our implementation of reverse KL regularization to create multi-step and iterative algorithms. We are not using algorithmic modifications like Q ensembles, regularized Q values, or early stopping that have been used in prior work. But, our iterative algorithm recovers similar performance to prior regularized actor-critic approaches. These results are shown in Table 7.2.

**Table 7.2:** Results of reverse KL regularization on the D4RL benchmark across one-step, multi-step, and iterative algorithms. Again we run 3 seeds and 6 hyperparameters and report the mean and standard deviation across seeds using 100 evaluation episodes.

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

{tab:multi}

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

## 7.6 What goes wrong for iterative algorithms?

{sec:why}

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

Distribution shift causes evaluation error by reducing the effective sample size in the fixed dataset for evaluating the current policy and has been extensively considered in prior work as discussed below. Iterative error exploitation occurs when we repeatedly optimize policies against our Q estimates and exploit their errors. This introduces a bias towards overestimation at each

step (much like the training error in supervised learning is biased to be lower than the test error). Moreover, by iteratively re-using the data and using prior Q estimates to warmstart training at each step, the errors from one step are amplified at the next. This type of error is particular to multi-step and iterative algorithms.

### 7.6.1 Learning curves and hyperparameter sensitivity

To begin to understand why iterative and multi-step algorithms can fail it is instructive to look at the learning curves. As shown in Figure 7.2, we often observe that the iterative algorithm will begin to learn and then crash. Regularization can help to prevent this crash since strong enough regularization towards the behavior policy ensures that the evaluation is nearly on-policy.

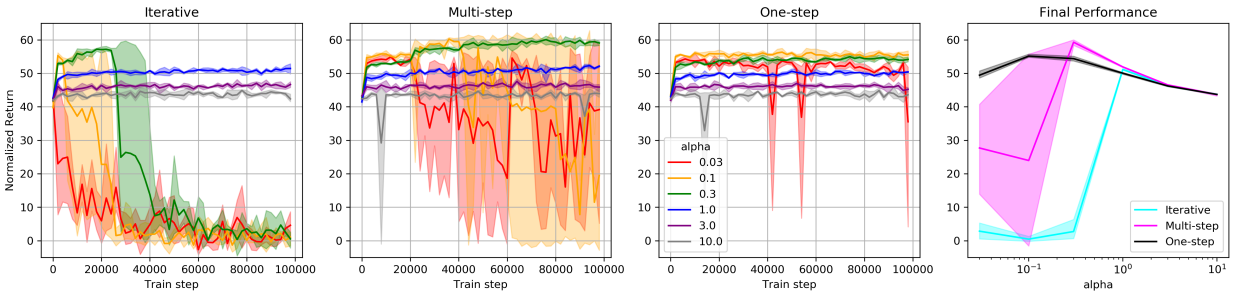



**Figure 7.2:** Learning curves and final performance on halfcheetah-medium across different algorithms and regularization hyperparameters. Error bars show min and max over 3 seeds. Similar figures for other datasets from D4RL can be found in Appendix 7.D.

{fig:learning_curves}

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

### 7.6.2 Distribution shift

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

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

Empirical evidence. Both the multi-step and iterative algorithms in our experiments rely on off-policy evaluation as a key subroutine. We examine how easy it is to evaluate the policies encountered along the learning trajectory. To control for issues of iterative error exploitation (discussed in the next subsection), we train Q estimators from scratch on a heldout evaluation

dataset sampled from the behavior policy. We then evaluate these trained Q function on rollouts from 1000 datapoints sampled from the replay buffer. Results are shown in Figure 7.3.

The results show a correlation betweed KL and MSE. Moreover, we see that the MSE generally increases over training. One way to mitigate this, as seen in the figure, is to use a large value of $\alpha$. We just cannot take a very large step before running into problems with distribution shift. But, when we take such a small step, the information from the on-policy $\widehat{Q}^{\beta}$ is about as useful as the newly estimated $\widehat{Q}^{\pi}$. This is seen, for example, in Figure 7.2 where we get very similar performance across algorithms at high levels of regularization.

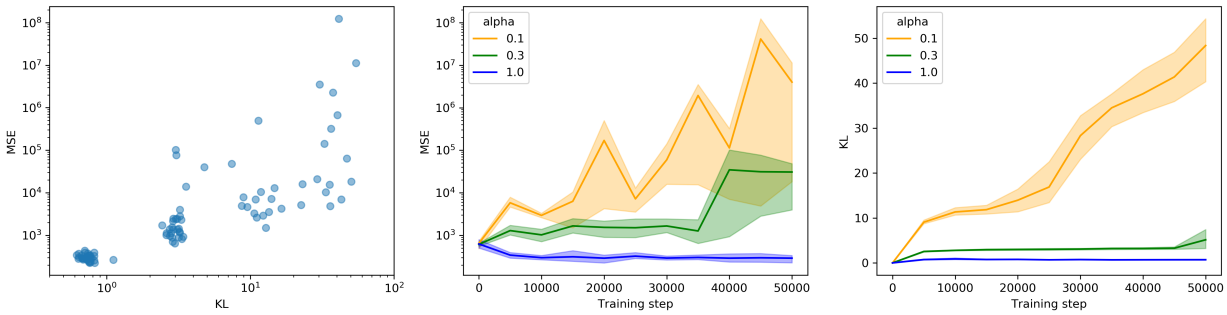



**Figure 7.3:** Results of running the iterative algorithm on halfcheetah-medium. Each checkpointed policy is evaluated by a Q function trained from scratch on heldout data. MSE refers to $\mathbb{E}_{s,a\sim\beta}[\hat{Q}^{\pi_i}(s,a) - Q^{\pi_i}(s,a)]$ and KL refers to $\mathbb{E}_{s\sim\beta}[KL(\pi(\cdot|s)\|\beta(\cdot|s)]$. Left: 90 policies taken from various points in training with various hyperaparmeters and random seeds. Center: MSE learning curves. Right: KL learning curves. Error bars show min and max over 3 random seeds. {fig:mse}

### 7.6.3 Iterative error exploitation

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

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

$$\widehat{Q}^\pi(s,a) = r(s,a) + \gamma \mathop{\mathbb{E}}_{\substack{s'|s,a \\ a'\sim\pi|s'}} [\widehat{Q}^\pi(s',a')] + \varepsilon_\beta^\pi(s,a). \tag{7.6}$$

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

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

$$\widehat{Q}^\pi(s,a) = Q^\pi(s,a) + \widetilde{Q}_\beta^\pi(s,a), \qquad \widetilde{Q}_\beta^\pi(s,a) := \mathop{\mathbb{E}}_{\pi|s_0,a_0=s,a} \left[ \sum_{t=0}^{\infty} \gamma^t \varepsilon_\beta(s_t,a_t) \right]. \tag{7.7}$$

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

Now we can explain the problem. Recall that under our assumption the $\varepsilon_\beta$ are fixed once we have a dataset and likely to have larger magnitude the further we go from the support of the dataset. So, with each step $\pi_i$ is able to better maximize $\varepsilon_\beta$, thus moving further from $\beta$ and increasing the magnitude of $\widetilde{Q}_\beta^{\pi_i}$ relative to $Q^{\pi_i}$. Even though $Q^{\pi_i}$ may provide better signal than $Q^\beta$, it can easily be drowned out by $\widetilde{Q}_\beta^{\pi_i}$. In contrast, $\widetilde{Q}_\beta^\beta$ has small magnitude, so the one-step algorithm is robust to errors[20].

An example. Now we consider a simple gridworld example to illustrate iterative error exploitation. This example fits exactly into the setup outlined above since all errors are due to reward estimation so the $\varepsilon_\beta$ is indeed constant over all $\pi$. The gridworld we consider has one deterministic good state with reward 1 and many stochastic bad states that have rewards distributed as $\mathcal{N}(-0.5, 1)$. We collect a dataset of 100 trajectories, each of length 100. One run of the multi-step offline regularized policy iteration algorithm is illustrated in Figure 7.4.

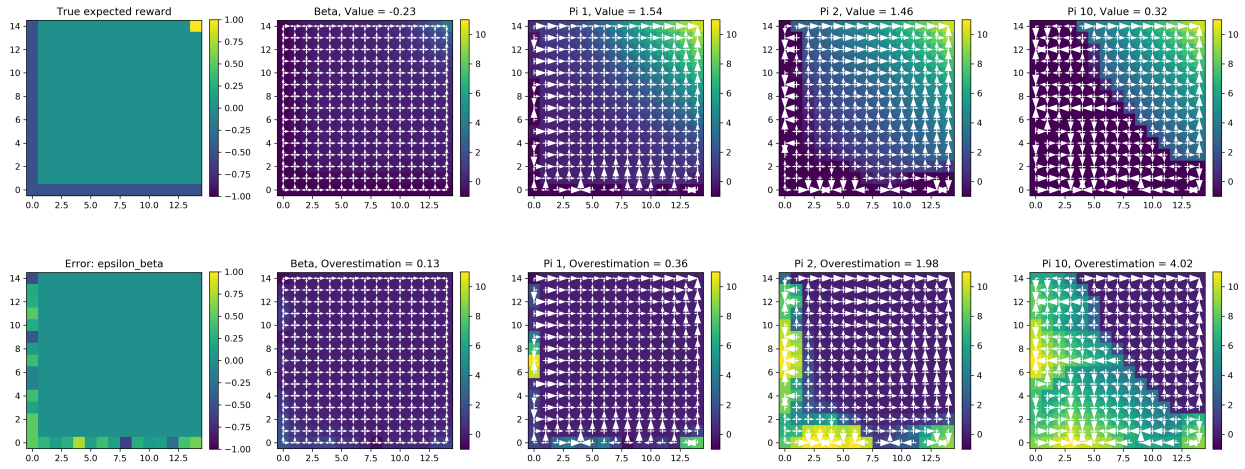



**Figure 7.4:** An illustration of multi-step offline regularized policy iteration. The leftmost panel in each row shows the true reward (top) or error $\varepsilon_\beta$ (bottom). Then each subsequent panel plots $\pi_i$ (with arrow size proportional to $\pi_i(a|s)$) over either $Q^{\pi_i}$ (top) or $\widetilde{Q}^{\pi}_{\beta}$ (bottom), averaged over actions at each state. The one-step policy ($\pi_1$) has the highest value. The behavior policy here is a mixture of optimal $\pi^*$ and uniform $u$ with coefficient 0.2 so that $\beta = 0.2 \cdot \pi^* + 0.8 \cdot u$. We set $\alpha = 0.1$ as the regularization parameter for reverse KL regularization. {fig:gridworld}

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

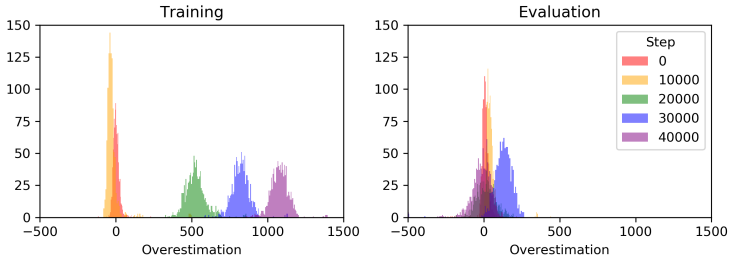



**Figure 7.5:** Histograms of overestimation error ($\widehat{Q}^{\pi_i}(s,a) - Q^{\pi_i}(s,a)$) on halfcheetah-medium with the iterative algorithm. Left: errors from the training Q function. Right: errors from an independently trained Q function. {fig:over}

eas where we have less data. So even though the errors in the gridworld are caused by noise in the rewards, while errors in D4RL are caused by function approximation, we think this is a useful mental model of the problem.

Empirical evidence. In practice we cannot easily visualize the progression of errors. However, the dependence between steps still arises as overestimation of the Q values. We can track the overestimation of the Q values over training as a way to measure how much bias is being induced by optimizing against our dependent Q estimators. As a control we can also train Q estimators from scratch on independently sampled evaluation data. These independently trained Q functions do not have the same overestimation bias even though the squared error does tend to increase as the policy moves further from the behavior (as seen in Figure 7.3). Explicitly, we track 1000 state, action pairs from the replay buffer over training. For each checkpointed policy we perform 3 rollouts at each state to get an estimate of the true Q value and compare this to the estimated Q value. Results are shown in Figure 7.5.

## 7.7 When are multiple steps useful? {sec:when}

So far we have focused on why the one-step algorithm often works better than the multi-step and iterative algorithms. However, we do not want to give the impression that one-step is always better. Indeed, our own experiments in Section 7.5 show a clear advantage for the multi-step and

iterative approaches when we have randomly collected data. While we cannot offer a precise delineation of when one-step will outperform multi-step, in this section we offer some intuition as to when we can expect to see benefits from multiple steps of policy improvement.

As seen in Section 7.6, multi-step and iterative algorithms have problems when they propagate estimation errors. This is especially problematic in noisy and/or high dimensional environments. While the multi-step algorithms propagate this noise more widely than the one-step algorithm, they also propagate the signal. So, when we have sufficient coverage to reduce the magnitude of the noise, this increased propagation of signal can be beneficial. The D4RL experiments suggest that we are usually on the side of the tradeoff where the errors are large enough to make one-step preferable.

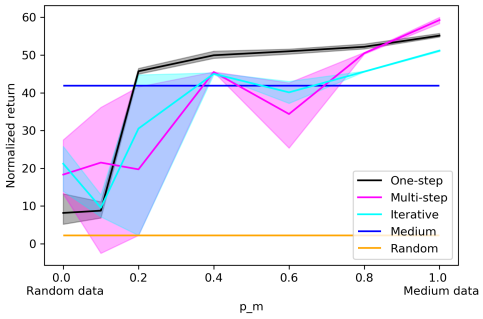



**Figure 7.6:** Performance of all three algorithms with reverse KL regularization across mixtures between halfcheetah-random and halfcheetah-medium. Error bars indicate min and max over 3 seeds. {fig:interp}

In Appendix 7.A we illustrate a simple gridworld example where a slight modification of the behavior policy from Figure 7.4 makes multi-step dramatically outperform one-step. This modified behavior policy (1) has better coverage of the noisy states (which reduces error, helping multi-step), and (2) does a worse job propagating the reward from the good state (hurting one-step).

We can also test empirically how the behavior policy effects the tradeoff between error and signal propagation. To do this we construct a simple experiment where we mix data from the random behavior policy with data from the medium behavior policy. Explicitly we construct a dataset $D$ out of the datasets $D_r$ for random and $D_m$ for medium such that each trajectory in $D$

comes from the medium dataset with probability $p_m$. So for $p_m = 0$ we have the random dataset and $p_m = 1$ we have the medium dataset, and in between we have various mixtures. Results are shown in Figure 7.6. It takes surprisingly little data from the medium policy for one-step to outperform the iterative algorithm.

## 7.8 Discussion, limitations, and future work

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

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

# Appendix 7.A Gridworld example where multi-step outperforms one-step

{sec:app_grid}

As explained in the main text, this section presents an example that is only a slight modification of the one in Figure 7.4, but where a multi-step approach is clearly preferred over just one step. The data-generating and learning processes are exactly the same (100 trajectories of length 100, discount 0.9, $\alpha = 0.1$ for reverse KL regularization). The only difference is that rather than using a behavior that is a mixture of optimal and uniform, we use a behavior that is a mixture of maximally suboptimal and uniform. If we call the suboptimal policy $\pi^-$ (which always goes down and left in our gridworld), then the behavior for the modified example is $\beta = 0.2 \cdot \pi^- + 0.8 \cdot u$, where $u$ is uniform. Results are shown in Figure 7.7.

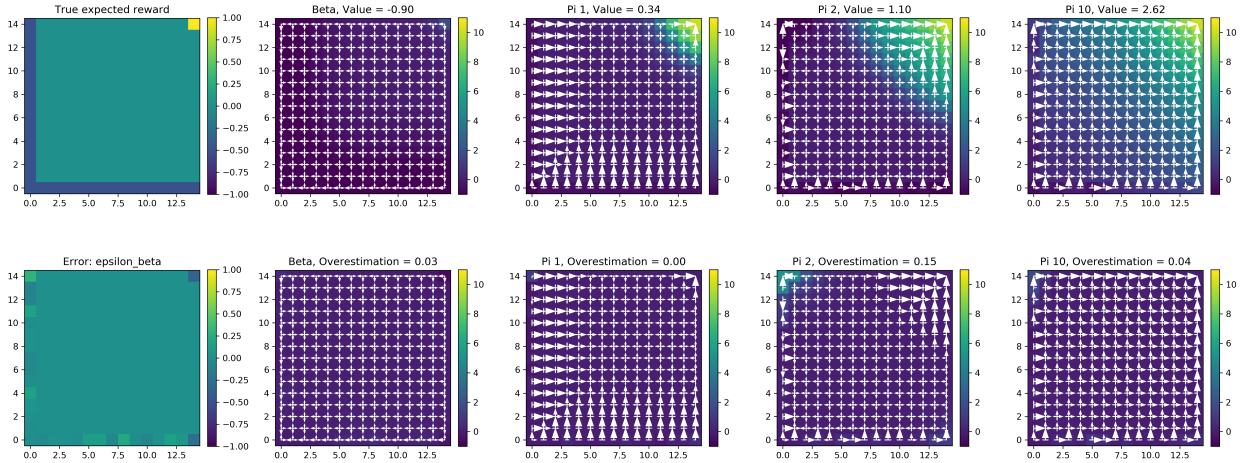



**Figure 7.7:** A gridworld example with modified behavior where multi-step is much better than one-step.

{fig:multi_gridworld}

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

one-step method.

## Appendix 7.B Connection to policy improvement guarantees

{sec:app_improvement}

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

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

**Lemma 7.1** (Performance difference, [Kakade and Langford 2002])**.** *For any two policies $\pi$ and $\beta$,*

$$J(\pi) - J(\beta) = \frac{1}{1-\gamma}\mathop{\mathbb{E}}_{s\sim d^\pi}[Q^\beta(s,\pi) - Q^\beta(s,\beta)]. \tag{7.8}$$

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

**Lemma 7.2** (Conservative Policy Improvement, [Achiam et al. 2017])**.** *For any two policies $\pi$ and*

$\beta$, *let* $\|A_\pi^\beta\|_\infty = \sup_s |Q^\beta(s,\pi) - Q^\beta(s,\beta)|$. *Then,*

$$J(\pi) - J(\beta) \geq \frac{1}{1-\gamma} \mathop{\mathbb{E}}_{s\sim d^\beta} \left[ \left( Q^\beta(s,\pi) - Q^\beta(s,\beta) \right) - \frac{2\gamma \|A_\pi^\beta\|_\infty}{1-\gamma} D_{TV}(\pi(\cdot|s) \| \beta(\cdot|s)) \right] \tag{7.9}$$

*where* $D_{TV}$ *denotes the total variation distance.*

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

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

## Appendix 7.C Experimental setup

{sec:app_exp_setup}

### 7.C.1 Benchmark experiments (Tables 7.1 and 7.2, Figure 7.2)

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

Hyperparameter tuning. We follow the practice of [Fu et al. 2020] and tune a small set of hyperparameters by interacting with the simulator to estimate the value of the policies learned under each hyperparameter setting. The hyperparam-

**Table 7.3:** Hyperparameter sweeps for each algorithm.

| Algorithm | Hyperparameter set |
|---|---|
| Reverse KL ($\alpha$) | {0.03, 0.1, 0.3, 1.0, 3.0, 10.0} |
| Easy BCQ ($M$) | {2, 5, 10, 20, 50, 100} |
| Exponentially weighted ($\tau$) | {0.1, 0.3, 1.0, 3.0, 10.0, 30.0} |

{tab:hyperparams}

eter sets for each algorithm can be
seen in Table 7.3.

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

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

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

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

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

We use three random seeds.

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

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

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

### 7.C.2 MSE experiment (Figure 3)

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

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

We train for 1 million steps.

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

### 7.C.3 Gridworld experiment (Figure 4)

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

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

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

### 7.C.4 Overestimation experiment (Figure 5)

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

### 7.C.5 Mixed data experiment (Figure 6)

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

## Appendix 7.D Learning curves

{sec:app_extra_exp}

In this section we reproduce the learning curves and hyperparameter plots across the one-step, multi-step, and iterative algorithms with reverse KL regularization, as in Figure 7.2.

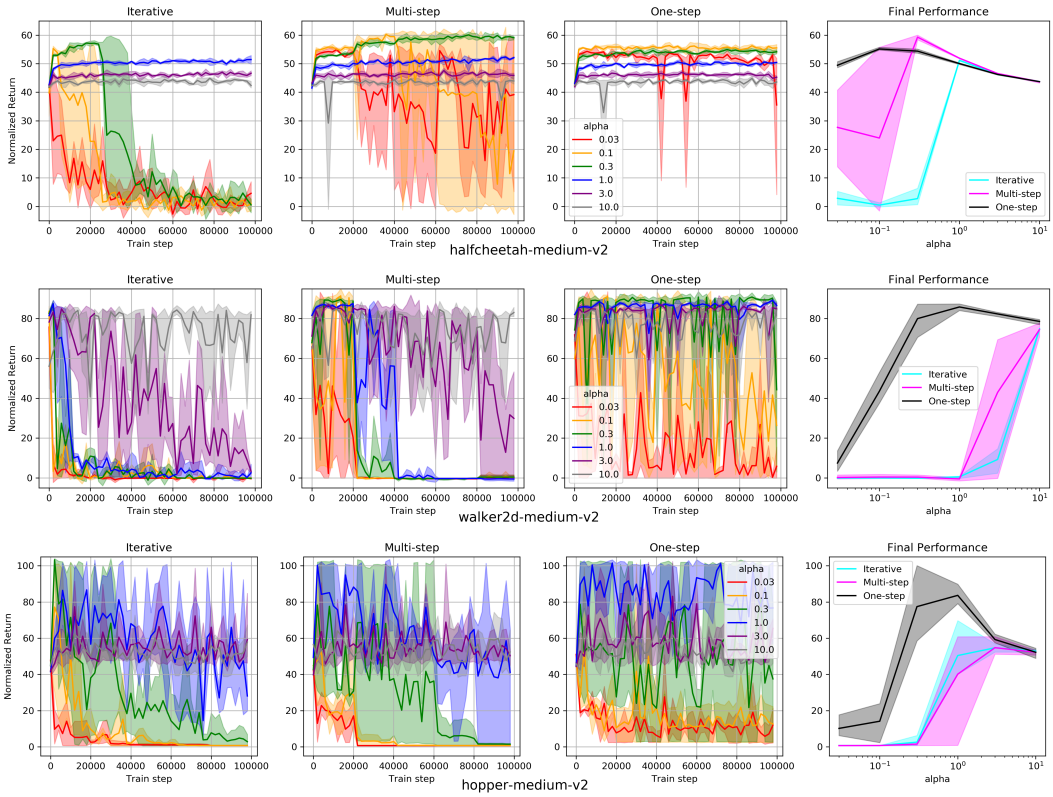



**Figure 7.8:** Learning curves on the medium datasets.

{fig:app_lc_medium}

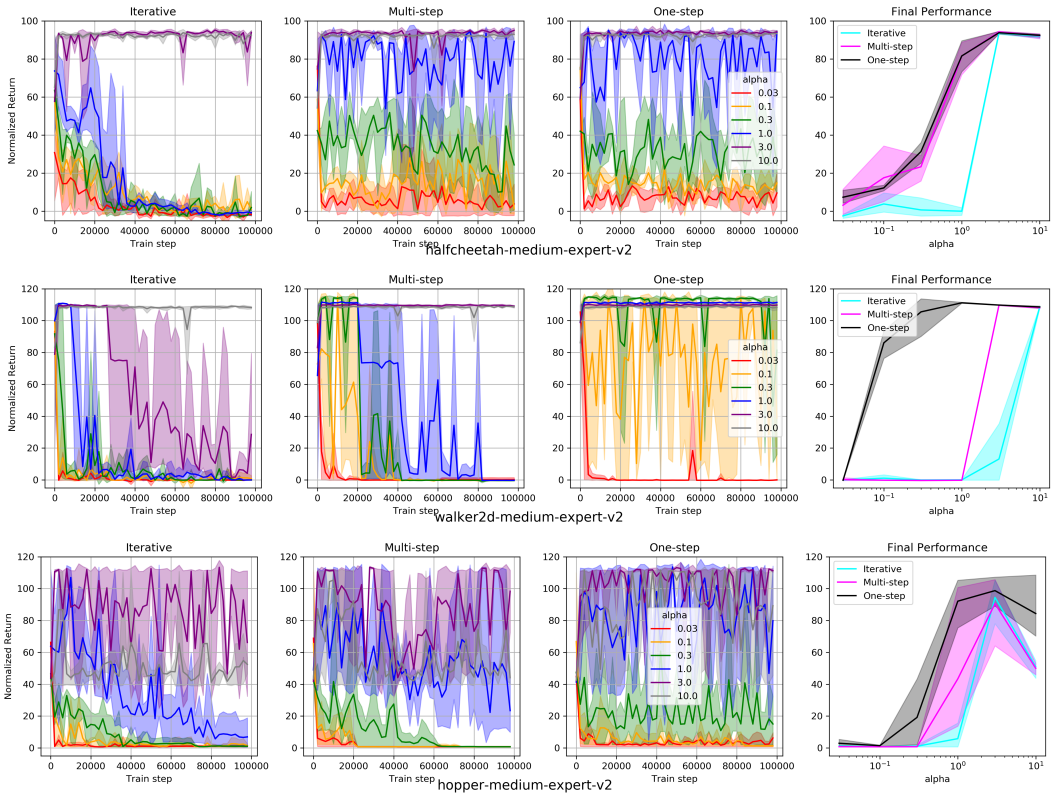



**Figure 7.9:** Learning curves on the medium-expert datasets.

{fig:app_lc_medium-ex

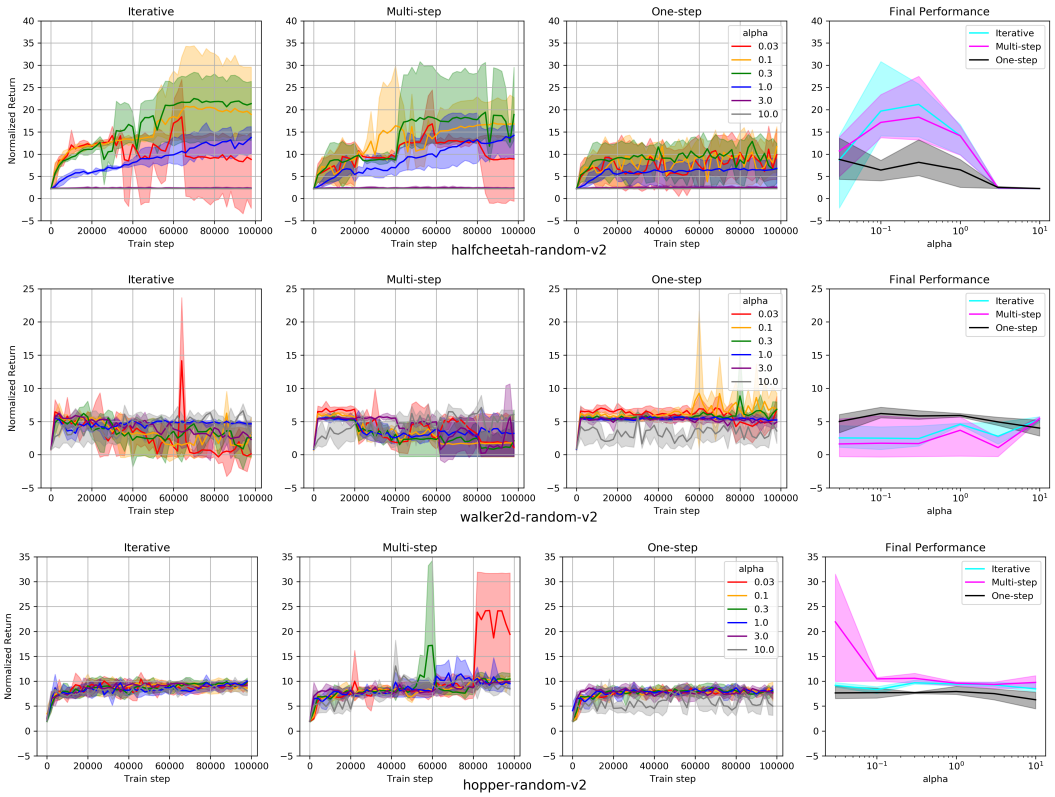



**Figure 7.10:** Learning curves on the random datasets.

{fig:app_lc_random}

# Notes

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

# 8 Conclusion

{sec:conclusion}

# Bibliography


Abdolmaleki, A., Springenberg, J. T., Tassa, Y., Munos, R., Heess, N. M. O., and Riedmiller, M. A. (2018). Maximum a posteriori policy optimisation. *ArXiv*, abs/1806.06920.

Achiam, J., Held, D., Tamar, A., and Abbeel, P. (2017). Constrained policy optimization. In *International Conference on Machine Learning*, pages 22–31. PMLR.

Agarwal, A., Jiang, N., and Kakade, S. (2019). Reinforcement learning: Theory and algorithms.

Alain, G. and Bengio, Y. (2016). Understanding intermediate layers using linear classifier probes. *International Conference on Learning Representations*.

Alemi, A. A., Fischer, I., Dillon, J. V., and Murphy, K. (2016). Deep variational information bottleneck. *arXiv preprint arXiv:1612.00410*.

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

Andrychowicz, M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., et al. (2018). Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177*.

Athey, S. and Wager, S. (2017). Efficient policy learning. *arXiv preprint arXiv:1702.02896*.

Bach, F. (2017). Breaking the curse of dimensionality with convex neural networks. *The Journal of Machine Learning Research*, 18(1):629–681.

Bachman, P., Hjelm, R. D., and Buchwalter, W. (2019). Learning representations by maximizing mutual information across views. In *Advances in Neural Information Processing Systems*.

Bacon, P.-L., Harb, J., and Precup, D. (2017). The option-critic architecture. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Badia, A. P., Sprechmann, P., Vitvitskyi, A., Guo, D., Piot, B., Kapturowski, S., Tieleman, O., Arjovsky, M., Pritzel, A., Bolt, A., and Blundell, C. (2020). Never give up: Learning directed exploration strategies. *ArXiv*, abs/2002.06038.

Barreto, A., Dabney, W., Munos, R., Hunt, J. J., Schaul, T., van Hasselt, H. P., and Silver, D. (2017). Successor features for transfer in reinforcement learning. In *Advances in neural information processing systems*, pages 4055–4065.

Barth-Maron, G., Hoffman, M. W., Budden, D., Dabney, W., Horgan, D., Muldal, A., Heess, N., and Lillicrap, T. (2018). Distributed distributional deterministic policy gradients. *arXiv preprint arXiv:1804.08617*.

Bartlett, P. L., Long, P. M., Lugosi, G., and Tsigler, A. (2020). Benign overfitting in linear regression. *Proceedings of the National Academy of Sciences*.

Belkin, M., Hsu, D. J., and Mitra, P. (2018). Overfitting or perfect fitting? risk bounds for classification and regression rules that interpolate. In *Advances in neural information processing systems*, pages 2300–2311.

Belkin, M., Rakhlin, A., and Tsybakov, A. B. (2019). Does data interpolation contradict statistical optimality? In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1611–1619. PMLR.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. (2016). Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 1471–1479.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2015). The arcade learning environment: An evaluation platform for general agents (extended abstract). In *IJCAI*.

Beygelzimer, A. and Langford, J. (2009). The offset tree for learning with partial labels. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 129–138.

Blier, L. and Ollivier, Y. (2018). The description length of deep learning models. In *Advances in Neural Information Processing Systems*.

Bottou, L. and Bousquet, O. (2008). The tradeoffs of large scale learning. In *Advances in neural information processing systems*, pages 161–168.

Bottou, L., Peters, J., Quiñonero-Candela, J., Charles, D. X., Chickering, D. M., Portugaly, E., Ray, D., Simard, P., and Snelson, E. (2013). Counterfactual reasoning and learning systems: The example of computational advertising. *The Journal of Machine Learning Research*, 14(1):3207–3260.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. (2018). JAX: composable transformations of Python+NumPy programs.

Brafman, R. and Tennenholtz, M. (2002). R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *J. Mach. Learn. Res.*, 3:213–231.

Brandfonbrener, D., Whitney, W. F., Ranganath, R., and Bruna, J. (2021). Offline contextual bandits with overparameterized models.

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

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016b). Openai gym. *CoRR*, abs/1606.01540.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCand lish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language Models are Few-Shot Learners. *arXiv preprint arXiv:2005.14165*.

Buckman, J., Gelada, C., and Bellemare, M. G. (2020). The importance of pessimism in fixed-dataset policy optimization.

Burda, Y., Edwards, H., Storkey, A., and Klimov, O. (2018). Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*.

Burkhardt, J. (2014). The truncated normal distribution.

Byrd, J. and Lipton, Z. C. (2018). What is the effect of importance weighting in deep learning? *arXiv preprint arXiv:1812.03372*.

Caselles-Dupré, H., Garcia-Ortiz, M., and Filliat, D. (2018). Continual state representation learning for reinforcement learning using generative replay. *arXiv preprint arXiv:1810.03880*.

Chandak, Y., Theocharous, G., Kostas, J., Jordan, S., and Thomas, P. S. (2019). Learning action representations for reinforcement learning. *arXiv preprint arXiv:1902.00183*.

Charlier, B., Feydy, J., Glaunès, J. A., Collin, F.-D., and Durif, G. (2021). Kernel operations on the gpu, with autodiff, without memory overflows. *Journal of Machine Learning Research*, 22(74):1–6.

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

Chen, M., Gummadi, R., Harris, C., and Schuurmans, D. (2019). Surrogate objectives for batch policy optimization in one-step decision making. In *Advances in Neural Information Processing Systems*, pages 8825–8835.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. E. (2020a). A simple framework for contrastive learning of visual representations. *arXiv preprint arXiv:2002.05709*.

Chen, X., Zhou, Z., Wang, Z., Wang, C., Wu, Y., and Ross, K. (2020b). Bail: Best-action imitation learning for batch deep reinforcement learning. *Advances in Neural Information Processing Systems*, 33.

Co-Reyes, J. D., Liu, Y., Gupta, A., Eysenbach, B., Abbeel, P., and Levine, S. (2018). Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings. In *ICML*.

Conneau, A., Kruszewski, G., Lample, G., Barrault, L., and Baroni, M. (2018). What you can cram into a single $&!#* vector: Probing sentence embeddings for linguistic properties. In *Annual Meeting of the Association for Computational Linguistics*, pages 2126–2136.

Cover, T. (1968). Estimation by the nearest neighbor rule. *IEEE Transactions on Information Theory*, 14(1):50–55.

Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27.

Cover, T. M. and Thomas, J. A. (2006). *Elements of information theory*. Wiley.

Dabney, W., Ostrovski, G., and Barreto, A. (2020). Temporally-extended $\epsilon$-greedy exploration. *arXiv: Learning*.

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

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *North American Chapter of the Association for Computational Linguistics*, pages 4171–4186.

Du, S., Kakade, S., Wang, R., and Yang, L. F. (2020). Is a good representation sufficient for sample efficient reinforcement learning? *ArXiv*, abs/1910.03016.

Du, S. S., Zhai, X., Poczos, B., and Singh, A. (2018). Gradient descent provably optimizes over-parameterized neural networks. *arXiv preprint arXiv:1810.02054*.

Duan, Y., Jia, Z., and Wang, M. (2020). Minimax-optimal off-policy evaluation with linear function approximation. In *International Conference on Machine Learning*, pages 2701–2709. PMLR.

Dubois, Y., Kiela, D., Schwab, D. J., and Vedantam, R. (2020). Learning optimal representations with the decodable information bottleneck. *ArXiv*, abs/2009.12789.

Dudík, M., Langford, J., and Li, L. (2011). Doubly robust policy evaluation and learning. *arXiv preprint arXiv:1103.4601*.

Dulac-Arnold, G., Evans, R., van Hasselt, H., Sunehag, P., Lillicrap, T., Hunt, J., Mann, T., Weber, T., Degris, T., and Coppin, B. (2015). Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*.

Ernst, D., Geurts, P., and Wehenkel, L. (2005). Tree-based batch mode reinforcement learning. *J. Mach. Learn. Res.*, 6:503–556.

Erven, T., Grünwald, P., and Rooij, S. (2012). Catching up faster by switching sooner: A predictive approach to adaptive stimation with an application to the aic-bic dilemma. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 74.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., et al. (2018). Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*.

Ettinger, A., Elgohary, A., and Resnik, P. (2016). Probing for semantic evidence of composition by means of simple classification tasks. In *Workshop on Evaluating Vector-Space Representations for NLP*, pages 134–139.

Fiechter, C. (1994). Efficient reinforcement learning. In *COLT '94*.

Florensa, C., Duan, Y., and Abbeel, P. (2017). Stochastic neural networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1704.03012*.

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., Blundell, C., and Legg, S. (2018). Noisy networks for exploration. *ArXiv*, abs/1706.10295.

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

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

Fujimoto, S., Hoof, H. V., and Meger, D. (2018a). Addressing function approximation error in actor-critic methods. *ArXiv*, abs/1802.09477.

Fujimoto, S., Meger, D., and Precup, D. (2018b). Off-policy deep reinforcement learning without exploration. *arXiv preprint arXiv:1812.02900*.

Fujimoto, S., van Hoof, H., and Meger, D. (2018c). Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*.

Ghosh, D., Gupta, A., and Levine, S. (2018). Learning actionable representations with goal-conditioned policies. *arXiv preprint arXiv:1811.07819*.

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

Grünwald, P. (2004). A tutorial introduction to the minimum description length principle. *arXiv preprint math:0406077*.

Gulcehre, C., Wang, Z., Novikov, A., Paine, T. L., Colmenarejo, S. G., Zolna, K., Agarwal, R., Merel, J., Mankowitz, D., Paduraru, C., et al. (2020). Rl unplugged: Benchmarks for offline reinforcement learning. *arXiv preprint arXiv:2006.13888*.

Haarnoja, T., Hartikainen, K., Abbeel, P., and Levine, S. (2018a). Latent space policies for hierarchical reinforcement learning. *arXiv preprint arXiv:1804.02808*.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018b). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*.

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

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. (2018). Learning latent dynamics for planning from pixels. *arXiv preprint arXiv:1811.04551*.

Hasselt, H. V., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *AAAI*.

Hausman, K., Springenberg, J. T., Wang, Z., Heess, N., and Riedmiller, M. (2018). Learning an embedding space for transferable robot skills. In *International Conference on Learning Representations*.

He, K., Fan, H., Wu, Y., Xie, S., and Girshick, R. (2019). Momentum contrast for unsupervised visual representation learning. *arXiv preprint arXiv:1911.05722*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Heek, J., Levskaya, A., Oliver, A., Ritter, M., Rondepierre, B., Steiner, A., and van Zee, M. (2020). Flax: A neural network library and ecosystem for JAX.

Hénaff, O. J., Srinivas, A., Fauw, J., Razavi, A., Doersch, C., Eslami, S., and Oord, A. (2020). Data-efficient image recognition with contrastive predictive coding. In *ICML*.

Henderson, D. and Parmeter, C. F. (2012). Normal reference bandwidths for the general order, multivariate kernel density derivative estimator. *Statistics & Probability Letters*, 82:2198–2205.

Hewitt, J. and Liang, P. (2019). Designing and interpreting probes with control tasks. In *Empirical Methods in Natural Language Processing and International Joint Conference on Natural Language Processing*, pages 2733–2743.

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., and Lerchner, A. (2017a). beta-vae: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*.

Higgins, I., Pal, A., Rusu, A., Matthey, L., Burgess, C., Pritzel, A., Botvinick, M., Blundell, C., and Lerchner, A. (2017b). Darla: Improving zero-shot transfer in reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1480–1490. JMLR. org.

Hill, F., Cho, K., and Korhonen, A. (2016). Learning distributed representations of sentences from unlabelled data. In *HLT-NAACL*.

Houthooft, R., Chen, X., Duan, Y., Schulman, J., De Turck, F., and Abbeel, P. (2016). Vime: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems*, pages 1109–1117.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. (2016). Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*.

Jaksch, T., Ortner, R., and Auer, P. (2008). Near-optimal regret bounds for reinforcement learning. In *J. Mach. Learn. Res.*

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

Jin, C., Allen-Zhu, Z., Bubeck, S., and Jordan, M. I. (2018). Is q-learning provably efficient? In *NeurIPS*.

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

Jonschkowski, R., Hafner, R., Scholz, J., and Riedmiller, M. A. (2017). Pves: Position-velocity encoders for unsupervised learning of structured state representations. *ArXiv*, abs/1705.09805.

Kakade, S. and Langford, J. (2002). Approximately optimal approximate reinforcement learning. In *ICML*, volume 2, pages 267–274.

Kalashnikov, D., Irpan, A., Pastor, P., Ibarz, J., Herzog, A., Jang, E., Quillen, D., Holly, E., Kalakrishnan, M., Vanhoucke, V., et al. (2018). Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*.

Kallus, N. (2018). Balanced policy evaluation and learning. In *Advances in Neural Information Processing Systems*, pages 8895–8906.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. (2020). Scaling laws for neural language models. *ArXiv*, abs/2001.08361.

Kaufmann, E., Cappé, O., and Garivier, A. (2016). On the complexity of best-arm identification in multi-armed bandit models. *J. Mach. Learn. Res.*, 17:1:1–1:42.

Kearns, M. and Singh, S. P. (1998). Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49:209–232.

Kim, H., Kim, J., Jeong, Y., Levine, S., and Song, H. O. (2018). Emi: Exploration with mutual information maximizing state and action embeddings. *arXiv preprint arXiv:1810.01176*.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

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

Kingma, D. P. and Welling, M. (2014). Auto-encoding variational Bayes. *International Conference on Learning Representations*.

Kiros, R., Zhu, Y., Salakhutdinov, R., Zemel, R., Urtasun, R., Torralba, A., and Fidler, S. (2015). Skip-thought vectors. In *NIPS*.

Kolter, J. Z. and Ng, A. (2009). Near-bayesian exploration in polynomial time. In *ICML '09*.

Kostrikov, I. (2018). Pytorch implementations of reinforcement learning algorithms. https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail.

Kostrikov, I., Tompson, J., Fergus, R., and Nachum, O. (2021). Offline reinforcement learning with fisher divergence critic regularization. *arXiv preprint arXiv:2103.08050*.

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. Technical report.

Kulkarni, T. D., Narasimhan, K., Saeedi, A., and Tenenbaum, J. (2016a). Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*, pages 3675–3683.

Kulkarni, T. D., Saeedi, A., Gautam, S., and Gershman, S. J. (2016b). Deep successor reinforcement learning. *arXiv preprint arXiv:1606.02396*.

Kumar, A., Fu, J., Soh, M., Tucker, G., and Levine, S. (2019). Stabilizing off-policy q-learning via bootstrapping error reduction. In *Advances in Neural Information Processing Systems*, pages 11761–11771.

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

Lange, S. and Riedmiller, M. A. (2010). Deep learning of visual control policies. In *ESANN*.

Langford, J. and Zhang, T. (2008). The epoch-greedy algorithm for multi-armed bandits with side information. In *Advances in neural information processing systems*, pages 817–824.

Laroche, R., Trichelair, P., and Des Combes, R. T. (2019). Safe policy improvement with baseline bootstrapping. In *International Conference on Machine Learning*, pages 3652–3661. PMLR.

LeCun, Y. (2015). Deep learning & convolutional networks. URL: https://www.hotchips.org/wp-content/uploads/hc_archives/hc27/HC27.24-Monday-Epub/HC27.24.19-Key1-Neural-Nets-Epub/HC27.24.190-Convolutional-Neural-LeCun-Facebook.pdf.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

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

Li, L., Chu, W., Langford, J., and Schapire, R. E. (2010). A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, pages 661–670.

Lillicrap, T., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Liu, X., Gao, J., Celikyilmaz, A., Carin, L., et al. (2019a). Cyclical annealing schedule: A simple approach to mitigating kl vanishing. *arXiv preprint arXiv:1903.10145*.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019b). RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*.

Ma, Y., Wang, Y.-X., et al. (2019). Imitation-regularized offline learning. *arXiv preprint arXiv:1901.04723*.

Machado, M. C., Bellemare, M. G., and Bowling, M. H. (2020). Count-based exploration with the successor representation. In *AAAI*.

McAllester, D. and Stratos, K. (2020). Formal limitations on the measurement of mutual information. *International Conference on Artificial Intelligence and Statistics*.

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

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015b). Human-level control through deep reinforcement learning. *Nature*, 518:529–533.

Momennejad, I., Russek, E. M., Cheong, J. H., Botvinick, M. M., Daw, N. D., and Gershman, S. J. (2017). The successor representation in human reinforcement learning. *Nature Human Behaviour*, 1(9):680.

Moore, A. W. and Atkeson, C. G. (1993). Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13:103–130.

Munos, R., Stepleton, T., Harutyunyan, A., and Bellemare, M. (2016). Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1054–1062.

Munos, R. and Szepesvári, C. (2008). Finite-time bounds for fitted value iteration. *Journal of Machine Learning Research*, 9(May):815–857.

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

Nachum, O., Gu, S., Lee, H., and Levine, S. (2018). Near-optimal representation learning for hierarchical reinforcement learning. *CoRR*, abs/1810.01257.

Neunert, M., Abdolmaleki, A., Wulfmeier, M., Lampe, T., Springenberg, J. T., Hafner, R., Romano,

F., Buchli, J., Heess, N., and Riedmiller, M. (2020). Continuous-discrete reinforcement learning for hybrid control in robotics.

Nota, C. and Thomas, P. S. (2020). Is the policy gradient a gradient? *ArXiv*, abs/1906.07073.

OpenAI, :, Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., d. O. Pinto, H. P., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., and Zhang, S. (2019a). Dota 2 with large scale deep reinforcement learning.

OpenAI, Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., Schneider, J., Tezak, N., Tworek, J., Welinder, P., Weng, L., Yuan, Q., Zaremba, W., and Zhang, L. (2019b). Solving rubik's cube with a robot hand. *ArXiv*, abs/1910.07113.

Osband, I., Blundell, C., Pritzel, A., and Roy, B. V. (2016). Deep exploration via bootstrapped dqn. In *NIPS*.

Osband, I., Roy, B. V., Russo, D. J., and Wen, Z. (2019). Deep exploration via randomized value functions. *Journal of Machine Learning Research*, 20(124):1–62.

Ostrovski, G., Bellemare, M. G., van den Oord, A., and Munos, R. (2017). Count-based exploration with neural density models. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2721–2730. JMLR. org.

Paine, T. L., Paduraru, C., Michi, A., Gulcehre, C., Zolna, K., Novikov, A., Wang, Z., and de Freitas, N. (2020). Hyperparameter selection for offline reinforcement learning.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chil-

amkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019a). Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019b). Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pages 8026–8037.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 16–17.

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

Peters, M., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. In *North American Chapter of the Association for Computational Linguistics*, pages 2227–2237.

Pimentel, T., Valvoda, J., Maudslay, R. H., Zmigrod, R., Williams, A., and Cotterell, R. (2020). Information-theoretic probing for linguistic structure. *arXiv preprint arXiv:2004.03061*.

Pinto, L. and Gupta, A. (2016). Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3406–3413. IEEE.

Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R. Y., Chen, X., Asfour, T., Abbeel, P., and Andrychowicz, M. (2018). Parameter space noise for exploration. *ArXiv*, abs/1706.01905.

Popov, I., Heess, N., Lillicrap, T., Hafner, R., Barth-Maron, G., Vecerík, M., Lampe, T., Tassa, Y., Erez, T., and Riedmiller, M. A. (2017). Data-efficient deep reinforcement learning for dexterous manipulation. *ArXiv*, abs/1704.03073.

Prasad, N., Cheng, L.-F., Chivers, C., Draugelis, M., and Engelhardt, B. (2017). A reinforcement learning approach to weaning of mechanical ventilation in intensive care units. *ArXiv*, abs/1704.06300.

Radford, A., Metz, L., and Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2019). Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*.

Raghu, A., Komorowski, M., Ahmed, I., Celi, L. A., Szolovits, P., and Ghassemi, M. (2017). Deep reinforcement learning for sepsis treatment. *ArXiv*, abs/1711.09602.

Rajeswaran, A., Kumar, V., Gupta, A., Vezzani, G., Schulman, J., Todorov, E., and Levine, S. (2017). Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*.

Rashid, T., Peng, B., Böhmer, W., and Whiteson, S. (2020). Optimistic exploration even with a pessimistic initialisation. *ArXiv*, abs/2002.12174.

Resnick, C., Zhan, Z., and Bruna, J. (2019). Probing the state of the art: A critical look at visual representation evaluation. *arXiv preprint arXiv:1912.00215*.

Rezende, D. J., Mohamed, S., and Wierstra, D. (2014a). Stochastic backpropagation and approximate inference in deep generative models. In *International Conference on Machine Learning*.

Rezende, D. J., Mohamed, S., and Wierstra, D. (2014b). Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*.

Riedmiller, M. A. (2005). Neural fitted q iteration - first experiences with a data efficient neural reinforcement learning method. In *ECML*.

Riedmiller, M. A., Hafner, R., Lampe, T., Neunert, M., Degrave, J., Wiele, T., Mnih, V., Heess, N., and Springenberg, J. T. (2018). Learning by playing - solving sparse reward tasks from scratch. In *ICML*.

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

Russo, D. (2016). Simple bayesian algorithms for best arm identification. In *COLT*.

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

Schoknecht, R. and Riedmiller, M. A. (2003). Reinforcement learning on explicitly specified time scales. *Neural Computing & Applications*, 12:61–80.

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

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

Shannon, C. (1948). A mathematical theory of communication. *Bell Syst. Tech. J.*, 27:379–423.

Shi, X., Padhi, I., and Knight, K. (2016). Does string-based neural MT learn source syntax? In *Empirical Methods in Natural Language Processing*, pages 1526–1534.

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

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Driessche, G. V. D., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D.

(2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *ICML*.

Stachenfeld, K. L., Botvinick, M. M., and Gershman, S. J. (2017). The hippocampus as a predictive map. *Nature neuroscience*, 20(11):1643.

Stadie, B. C., Levine, S., and Abbeel, P. (2015). Incentivizing exploration in reinforcement learning with deep predictive models. *ArXiv*, abs/1507.00814.

Strehl, A., Langford, J., Li, L., and Kakade, S. M. (2010). Learning from logged implicit exploration data. In *Advances in Neural Information Processing Systems*, pages 2217–2225.

Strehl, A. L., Li, L., Wiewiora, E., Langford, J., and Littman, M. (2006). Pac model-free reinforcement learning. In *ICML '06*.

Strehl, A. L. and Littman, M. L. (2008). An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S., Precup, D., and Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211.

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

Swaminathan, A. and Joachims, T. (2015b). The self-normalized estimator for counterfactual learning. In *advances in neural information processing systems*, pages 3231–3239.

Taiga, A. A., Fedus, W., Machado, M. C., Courville, A., and Bellemare, M. G. (2020). On bonus based exploration methods in the arcade learning environment. In *International Conference on Learning Representations*.

Talmor, A., Elazar, Y., Goldberg, Y., and Berant, J. (2019). oLMpics–on what language model pre-training captures. *arXiv preprint arXiv:1912.13283*.

Tang, H., Houthooft, R., Foote, D., Stooke, A., Xi Chen, O., Duan, Y., Schulman, J., DeTurck, F., and Abbeel, P. (2017). #exploration: A study of count-based exploration for deep reinforcement learning. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 2753–2762. Curran Associates, Inc.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. d. L., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., et al. (2018). Deepmind control suite. *arXiv preprint arXiv:1801.00690*.

Tennenholtz, G. and Mannor, S. (2019). The natural language of actions. *arXiv preprint arXiv:1902.01119*.

Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294.

Tishby, N., Pereira, F. C., and Bialek, W. (2000). The information bottleneck method. *arXiv preprint physics/0004057*.

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE.

van den Oord, A., Li, Y., and Vinyals, O. (2018). Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*.

Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*.

Van Hoof, H., Chen, N., Karl, M., van der Smagt, P., and Peters, J. (2016). Stable reinforcement learning with autoencoders for tactile and visual data. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3928–3934. IEEE.

Vapnik, V. (1982). *Estimation of Dependences Based on Empirical Data: Springer Series in Statistics (Springer Series in Statistics)*. Springer-Verlag, Berlin, Heidelberg.

Vehtari, A., Gelman, A., and Gabry, J. (2015). Pareto smoothed importance sampling. *arXiv: Computation*.

Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., and Kavukcuoglu, K. (2017). Feudal networks for hierarchical reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3540–3549. JMLR. org.

Vinyals, O., Babuschkin, I., Czarnecki, W., Mathieu, M., Dudzik, A., Chung, J., Choi, D., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J., Jaderberg, M., Vezhnevets, A., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T., Gulcehre, C., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Kavukcuoglu, K., Hassabis, D., Apps, C., and Silver, D. (2019). Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, pages 1–5.

Voita, E. and Titov, I. (2020). Information-theoretic probing with minimum description length. *arXiv preprint arXiv:2003.12298*.

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

Wang, Q., Xiong, J., Han, L., Liu, H., Zhang, T., et al. (2018). Exponentially weighted imitation learning for batched historical data. In *Advances in Neural Information Processing Systems*, pages 6288–6297.

Wang, R., Foster, D. P., and Kakade, S. M. (2020a). What are the statistical limits of offline rl with linear function approximation?

Wang, R., Wu, Y., Salakhutdinov, R., and Kakade, S. M. (2021). Instabilities of offline rl with pre-trained neural representation. *arXiv preprint arXiv:2103.04947*.

Wang, Z., Novikov, A., Zolna, K., Merel, J. S., Springenberg, J. T., Reed, S. E., Shahriari, B., Siegel, N., Gulcehre, C., Heess, N., et al. (2020b). Critic regularized regression. *Advances in Neural Information Processing Systems*, 33.

Whitney, W. F., Agarwal, R., Cho, K., and Gupta, A. (2020). Dynamics-aware embeddings. In *ICLR*.

Wu, Y., Tucker, G., and Nachum, O. (2019). Behavior regularized offline reinforcement learning.

Xu, Y., Zhao, S., Song, J., Stewart, R., and Ermon, S. (2020). A theory of usable information under computational constraints. In *International Conference on Learning Representations*.

Yarats, D. and Kostrikov, I. (2020). Soft actor-critic (sac) implementation in pytorch. https://github.com/denisyarats/pytorch_sac.

Yogatama, D., d'Autume, C. d. M., Connor, J., Kocisky, T., Chrzanowski, M., Kong, L., Lazaridou, A., Ling, W., Yu, L., Dyer, C., et al. (2019). Learning and evaluating general linguistic intelligence. *arXiv preprint arXiv:1901.11373*.

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

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2016). Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*.

Zhang, K. and Bowman, S. (2018). Language modeling teaches you more than translation does: Lessons learned through auxiliary syntactic task analysis. In *EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 359–361.

Zhou, Z., Athey, S., and Wager, S. (2018). Offline multi-action policy learning: Generalization and optimization. *arXiv preprint arXiv:1810.04778*.