Real World Robot Learning: Learned Rewards, Offline Datasets and Skill Re-Use

Avi Singh

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2021-179
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-179.html

August 11, 2021

Real World Robot Learning: Learned Rewards, Offline Datasets and Skill Re-Use

by

Avi Singh

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Associate Professor Sergey Levine, Chair
Professor Pieter Abbeel
Associate Professor Anca Dragan
Professor Ken Goldberg

Summer 2021

Real World Robot Learning: Learned Rewards, Offline Datasets and Skill Re-Use

Abstract

Real World Robot Learning: Learned Rewards, Offline Datasets and Skill Re-Use

by

Avi Singh

Doctor of Philosophy in Computer Science

University of California, Berkeley

Associate Professor Sergey Levine, Chair

Robots that can operate in an open, unstructured environment and perform a wide range of tasks have been a long-standing goal of artificial intelligence. For such robots to operate effectively, they need the ability to a) perceive the world around them through general-purpose on-board sensors like cameras b) generalize to new situations c) improve their performance as they collect more data. In this thesis, we posit that deep reinforcement learning (deep RL) methods are well-positioned to overcome the aforementioned challenges, but are difficult to apply to real world domains like robotics. The central conjecture that we study in this work is the following: while dominant robot learning pipelines often rely on hand-engineering certain components (such as reward functions and physics simulators), we can overcome many bottlenecks of these pipelines via the adoption of a more data-driven perspective. We argue that, instead of hand-engineering reward functions, we should instead learn reward functions from data. Instead of learning mostly in a hand-designed simulation and then transferring learned policies to the real world, we should learn using real data, and re-use all past experience (as much as possible) to maintain sample efficiency. We show how this perspective change greatly simplifies robot learning, and demonstrate results on a variety of real world object manipulation tasks.

Contents

List of Figures

List of Tables

Acknowledgments

Over the course of the PhD, I have been fortunate to have the support of numerous mentors, friends and family members. This acknowledgement is unlikely to be exhaustive, so I apologize in advance for any omissions that I make.

First and foremost, I would like to thank my adviser Sergey Levine, both for giving me a chance to pursue a PhD in the first place, and for his unwavering support throughout this journey. Berkeley was the first place where I got the opportunity to lead research projects, mentor undergraduate researchers, and establish long-lasting collaborations, none of which would have been possible without Sergey's mentorship. My experience working with Sergey has deeply influenced the researcher that I have become, and I look forward to working together again.

I would like to thank Ashutosh Saxena and Ashesh Jain for taking a chance on me as an undergraduate researcher, and for introducing me to the world of academic research. The lessons I learned while closely working with Ashesh at Cornell in the summer of 2015 laid the foundations of my academic career, and I will always be grateful for the opportunity.

I would like to thanks the members of the RAIL lab for providing a phenomenal environment for research; I am convinced that I would not find a better set of peers anywhere else in the world. For the work presented this thesis, I am grateful to my collaborators Aviral Kumar, Kristian Hartikainen, Justin Fu, Chelsea Finn, and Nick Rhinehart. I would also like to thank other members of the RAIL lab, including Abhishek Gupta, Coline Devin, Greg Kahn, Anusha Nagabandi, JD Co-Reyes, Marvin Zhang, Vitchyr Pong, Michael Janner, Frederik Ebert, Ashvin Nair, Sid Reddy, Young Geng, Vikash Kumar, Dinesh Jayaraman, Rowan McAllistor, Natasha Jacques, Roberto Calandra, Vikash Kumar, Alex Lee, Tuomas Haarnoja, Kelvin Xu, and many others for the numerous discussions, debates, and office chit-chat that made these last few years pass by so quickly. I would like to thank the friends that I made at Berkeley, including Andreea Bobu, Vickie Ye, Suma Anand, Parsa Mahmoudieh, Erin Grant, Deepak Pathak and Pulkit Agarwal, for their kindness and support through this journey, and for helping me maintain my sanity through a global pandemic.

I would like to thank the numerous undergraduate researchers that I had the opportunity to work with, especially Larry Yang, Dibya Ghosh, Albert Yu, Jonathan Yang, Huihan Liu and Gaoyue Zhou. Introducing young researchers to the world of deep reinforcement learning research was one of the most fulfilling part of the PhD journey, and this thesis would not have been possible without their numerous contributions. In particular, I would like to thank Larry Yang and Dibya Ghosh for their contributions to Chapters 2 and 3, Huihan Liu and Gaoyue Zhou for their contributions to Chapter 4, and Albert Yu and Jonathan Yang for their contributions to Chapter 5.

I would like to thank May Simpson, for your companionship and support in the last several years. Having you in my corner made me a much stronger person, making the challenges of a graduate student career much easier to handle. Finally, I would like to thank my mother, Purnima, for her unconditional love and support throughout my life, and for shaping me into the person that I am today.

Chapter 1

Introduction

The last decade witnessed an unprecedented explosion in AI-powered applications: our phones unlock via face recognition, can recognize our speech with near-perfection, and are often uncannily good at suggesting email responses. However, a lot of these applications have been restricted to the digital realm: while we have AI systems that are adept at processing real world sensory data (from cameras and microphones), and agents that are adept at making decisions in the purely digital world (such as trading bots or chess bots), we don't yet have as many intelligent systems *acting* in the real, physical world. Even with billions of dollars invested in specific application domains (such as autonomous driving), we don't yet have clarity as to when such autonomous systems will become ubiquitous in the open, unstructured physical world.

Recent progress in areas such as computer vision and natural language processing relies on supervised (and self-supervised) deep learning. On the other hand, sequential decision-making problems such as robotics are more suited to a different mathematical framework: that of reinforcement learning (RL). Works such as Minh et al [108], Schulman et al [143], and Levine et al [94] saw the emergence of deep reinforcement learning methods: the fusion of reinforcement learning principals with powerful neural network-based function approximation. Scaled-up implementation of deep reinforcement learning methods have resulted in several compelling results, such as achieving super-human performance in the board game Go [147] and extremely competitive agents for the multi-player game Dota 2 [119]

The success of deep RL in simulated domains such as Go and Dota 2 is encouraging, but it hasn't yet translated into equally impressive results in real world domains like robotics. We posit that the reason for this gap in performance is two-fold. First, it is often straightforward to specify an objective in simulated settings. For a game like Go, the reward function is simply whether you won the game or not. On the other hand, for an agent attempting to achieve an objective in the real world, measuring whether you succeeded or not is a challenging problem in and of itself. Second, simulated settings make it easier to collect (and re-collect) large amounts of experience, as long as the data collection can be parallelized. The only limiting factor is the computational cost, which goes down every year. In the rest of this chapter, we will present a broad overview of how these limitations can be overcome in the context of

robotic learning.

1.1 End-to-End Data-Driven Robotics

The **central conjecture** of this thesis is the following: robot learning can be made more effective by making it more *data-driven*. While robot learning has a long history, work in this area has often focused on learning just one or two components of the overall robotics pipeline, in order to make the learning problem easier to tackle. This is not too different from the role that machine learning had in other AI areas such as computer vision and natural language understanding: there was some learning involved, but a significant part of the overall pipeline for tasks such as object detection was hand-engineered using heuristics and domain knowledge. The success of end-to-end data-driven methods in domains like computer vision and natural language processing naturally led to an increase in interest in end-to-end learned robotics, with works such as Levine et al [95] and Gu et al [45] being some of early adopters of deep RL for robotics. This thesis is another step towards end-to-end data-driven robot learning, and our goal is to remove hand-engineering from even more components of the robotic pipeline, and to introduce learned systems in their place.

When it comes to specifying rewards, instead of engaging in task-specific engineering or soliciting frequent human feedback (as has been done in prior deep RL for robotics work [141, 72, 8]), we propose *learning* reward functions from data. The benefits of a data-driven reward specification framework are several. First, it substantially reduces the time needed to specify a new task to the robot. While building a system for computing rewards for a particular task might require hours of engineering effort, the techniques we present for reward learning in this work reduce that time to a few minutes. Second, it allows non-expert users to specify objectives to a robot, making it possible to deploy robots that can be trained by their users. Finally, as we in show in subsequent chapters, the reward learning framework allows us to learn robotics skills that would have been nearly impossible to specify manually.

When it comes to utilizing experience for robotic RL, we argue that there is a need to move to a data-driven view of deep RL, instead of the prevailing simulation-driven paradigm. Developing and maintaining simulated environments for robotics requires substantial engineering effort, and if we rely solely on simulation, we need to put in additional effort every time the robot encounters a new scenario (such as a new object), or if there is a change in the environment (such as changing the lighting setup). While simulation is an excellent tool for domains that can be simulated perfectly (such as games), or those that can be simulated with near perfection (such as for aerospace applications), it cannot entirely remove the need for real data in robotics. Robots performing object manipulation need to precisely reason about contact dynamics and impact of physical forces on entities such as deformable objects, all of which are non-trivial with current simulation tools. More advanced robotic systems, such as those that need to interact with humans on a regular basis, are in even more need of real world experience, as simulating human behavior accurately will likely remain an open problem for the years to come (after all, simulating human intelligence is the central goal of

artificial intelligence research).

Now that we have established the importance of real world experience for robot learning, we need to acknowledge the the fact that collecting data in the real world is slow and expensive. To best utilize the experience collected in the real world, we propose the following: instead of collecting experience from scratch for any task we wish to learn, we should re-use past experience (from a variety of different scenarios) as much as possible, and never train from scratch (as is often the case for simulated RL experiments). The recent emergence of algorithms for offline RL [97] (also known as batch RL) is at least partially motivated by the reasons we mention above (which apply to many real world domains, not just robotics), and in this work we both build on the work in the offline RL literature and also take inspiration from generative pre-training in adjacent areas such as natural language processing.

In the remainder of this chapter, we discuss these issues in more detail, and provide a brief overview of the thesis.

1.2 Contributions and Related Work

Learning rewards from examples When applying reinforcement learning to a video game such as Breakout, the reward function is easy to specify: we can just pass in the game score. However, when applying reinforcement learning to a robot manipulation problem such as folding a shirt, a clear reward signal is now substantially harder to provide. In order to compute rewards for problems such as this one, real world robotic RL systems often involve substantial task-specific engineering effort, such as using thermal cameras for tracking fluids [141] or mocap sensors for tracking objects [80].

Inverse reinforcement learning methods [175, 163, 33, 59, 38] seek to automate reward definition by learning a reward function from expert demonstrations. However, these methods have limitations of their own: collecting expert demonstrations puts a significant data collection burden on the user, and demonstrations are usually non-intuitive for a person to provide (often involving kinesthetically moving a robot [33] or teleoperating it [158]). More importantly, demonstrating how a task is done defeats a central goal of reinforcement learning: autonomously discovering how to perform skills through trial and error.

In **Chapter 2**, we present an alternative view of task specification for RL: we can specify the objective for a task using visual examples of what we would like the robot to achieve. In the case of folding a shirt, we can present the robot with a few images of how we would like the folded shirt to look like. In Chapter 2, we present a theoretically grounded framework for robotic skill learning when the task objective is specified in this fashion. This work was published at Neural Information Processing Systems (NeurIPS) in 2018.

Learning rewards with active learning In **Chapter 3**, we continue to build on the example-driven task specification framework introduced in Chapter 2, and introduce a number of improvements that makes it possible to learn vision-based policies with learned reward functions for real robots. While the reward learning framework introduced in Chapter 2

was in the on-policy setting, here we develop an off-policy variant, allowing us to utilize sample efficient actor-critic methods for skill learning. In addition to specifying the task via user-provided outcome examples, we also allow the agent to make occasional queries to a human user to further improve its understanding of the user's intention, leading to fast and stable learning on a number of real world robotic manipulation tasks. This work was published at Robotics: Science and Systems (RSS) in 2019.

Incorporating multi-task demonstrations to pre-train RL agents RL agents typically explore their environment in a purely random fashion (as they assume no prior knowledge about the task at hand), which results in long exploration periods before the agent collects any useful experience at all. While this naïve approach is still effective in domains where data collection is cheap and can be sped up (such as games), it becomes a major bottleneck in robotics, where data collection is both time consuming and expensive. While demonstrations can be used to speed up RL [140, 127, 81, 56, 160, 113, 130, 148, 124, 70, 47], this usually requires collecting demonstrations for the specific task that is being learned. In **Chapter 4**, we present a method for pre-training RL agents using demonstrations from a wide range of previously seen tasks, to speed up learning on new tasks. Our work takes inspiration from the success of generative modeling-based pre-training in other deep learning areas such as computer vision [21] and natural language processing [18], and presents a method for pre-training in the RL setting. Prior works such as IntentionGAN [54] and InfoGAIL [98] learn multi-modal policies via interaction with an environment using an adversarial imitation approach [59], but we learn these distributions only from data. Other works learn these distributions from data [165, 136] and utilize them for planning at test time to optimize a user-provided cost function. In contrast, we use the behavioral prior to augment model-free RL of a new task. This allows us to learn policies for new tasks that may be substantially different from prior tasks, since we can collect data specific to the new task at hand, and we do not explicitly need to model the environment, which can be complicated for high-dimensional state and action spaces, such as when performing continuous control from images observations. This work was published at the International Conference on Representation Learning (ICLR) in 2021.

Incorporating unstructured/unlabeled prior datasets in RL. While Chapter 4 focuses on utilizing previously collected expert demonstrations to bootstrap reinforcement learning, how can we best utilize sub-optimal prior datasets, such as data from previous RL experiments or from unsupervised or undirected environment interaction? In **Chapter 5**, we propose an approach to incorporate such prior data to extend and generalize *new* skills. Incorporating prior datasets has emerged as an important technique to scale up robotic RL through works such as Kalashnikov et al [72], Julian et al [71], and Cabi et al [8]. However, these prior works focus on generalization to new objects, as well as fine-tuning to handle greater variability (e.g., more object types, changes in lighting, etc.). Our work instead focuses on changes in initial conditions that require entirely different skills than those learned

as part of the current task.For example, if a robot was trained to take an object out of an open drawer, and encounters a closed drawer at some point of time in its deployment, it should realize that it must open the drawer before it can achieve its objective. We build on recent advances in offline reinforcement learning, and our hardest experimental setting allows the trained agent to perform "common-sense" reasoning to compose four vision-based robotic skills in a row: picking, placing, drawer opening, and grasping, and a $+1/0$ sparse reward is provided only on task completion. This work was published at the Conference on Robot Learning (CoRL) in 2020.

Finally, in **Chapter 6**, we discuss some of the limitations of the solutions presented in this thesis, and point out some promising directions for future research.

Chapter 2

Variational Inverse Control with Events

2.1 Introduction

Reinforcement learning (RL) has shown remarkable promise in recent years, with results on a range of complex tasks such as robotic control [95] and playing video games [108] from raw sensory input. RL algorithms solve these problems by learning a policy that maximizes a reward function that is considered as part of the problem formulation, and there is little practical guidance on how these rewards should be designed. However, the design of the reward function is in practice critical for good results, and reward misspecification can easily cause unintended behavior [3]. For example, a vacuum cleaner robot rewarded to pick up dirt could exploit the reward by repeatedly dumping dirt on the ground and picking it up again [137]. Additionally, it is often difficult to write down a reward function at all. For example, when learning policies from high-dimensional visual observations, practitioners often resort to using motion capture [126] or specialized computer vision systems [138] to obtain rewards.

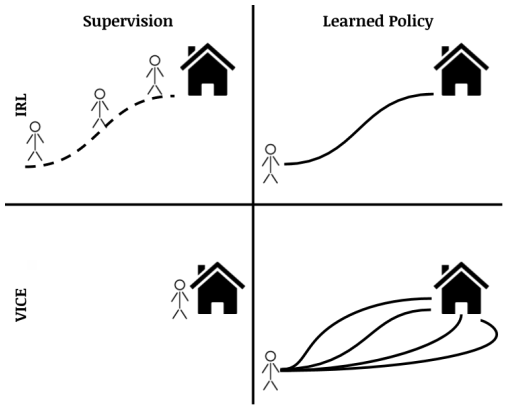


Figure 2.1: Standard IRL requires full expert demonstrations and aims to produce an agent that mimics the expert. VICE generalizes IRL to cases where we only observe final desired outcomes, which does not require the expert to actually know *how* to perform the task.

As an alternative to reward specification, imitation learning [5] and inverse reinforcement learning [115] instead seek to mimic expert behavior. However, such approaches require an expert to show *how* to solve a task. We instead propose a novel problem formulation, variational inverse control with events (VICE), which generalizes inverse reinforcement learning to alternative forms of expert supervision. In particular, we consider cases when

we have examples of a desired final outcome, rather than full demonstrations, so the expert only needs to show *what* the desired outcome of a task is (see Figure 2.1). A straightforward way to make use of these desired outcomes is to train a classifier [128, 157] to distinguish desired and undesired states. However, it is unclear if using this classifier as a reward will result in the intended behavior, since an RL agent can learn to exploit the classifier, in the same way it can exploit human-designed rewards. Our framework provides a more principled approach, where a particular form of classifier training corresponds to learning the parameters of a probabilistic graphical model (see Figure 2.2), and policy optimization corresponds to inferring the optimal actions in the same graphical model. By selecting an inference query which corresponds to our intentions, we can mitigate reward hacking scenarios similar to those previously described, and also specify the task with examples rather than manual engineering. This makes it practical to base rewards on raw observations, such as images.

Our inverse formulation is based on a corresponding forward control framework which reframes control as inference in a graphical model. Our framework resembles prior work [74, 156, 135], but we extend this connection by replacing the conventional notion of rewards with event occurence variables. Rewards correspond to log-probabilities of events, and value functions can be interpreted as backward messages that represent log-probabilities of those events occurring. This framework retains the full expressivity of RL, since any rewards can be expressed as log-probabilities, while providing more intuitive guidance on task specification. It further allows us to express various intentions, such as for an event to happen at least once, exactly once at any time step, or once at a specific timestep. Crucially, our framework does not require the agent to *observe* the event happening, but only to know the probability that it occurred. While this may seem unusual, it is more practical in the real world, where success may be determined by probabilistic models that themselves carry uncertainty. For example, the previously mentioned vacuum cleaner robot needs to estimate from its observations whether its task has been accomplished and would never receive direct feedback from the real world whether a room is clean.

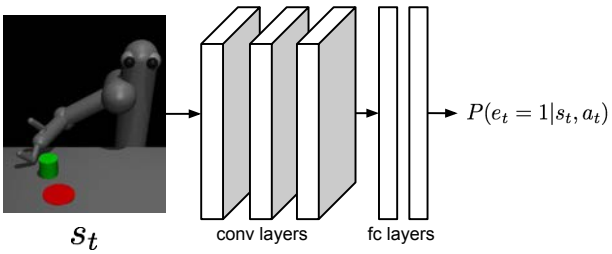


Figure 2.2: Our framework learns event probabilities from data. We use neural networks as function approximators to model this distribution, which allows us to work with high dimensional observations like images.

Our contributions are as follows. We first introduce the event-based control framework by extending previous control as inference work to alternative queries which we believe to be useful in practice. This view on control can ease the process of reward engineering by mapping a user's intention to a corresponding inference query in a probabilistic graphical model. Our experiments demonstrate how different queries can result in different behaviors which align with the corresponding intentions. We then propose methods to learn event probabilities from data, in a manner analogous to inverse reinforcement learning. This corresponds to the use

case where designing event probabilities by hand is difficult, but observations (e.g., images) of successful task completion are easier to provide. This approach is substantially easier to apply in practical situations, since full demonstrations are not required. Our experiments demonstrate that our framework can be used in this fashion for policy learning from high dimensional visual observations where rewards are hard to specify. Moreover, our method substantially outperforms baselines such as sparse reward RL, indicating that our framework provides an automated shaping effect when learning events, making it feasible to solve otherwise hard tasks.

2.2 Relation to prior work

Our reformulation of RL is based on the connection between control and inference [74, 174, 135]. The resulting problem is sometimes referred to as maximum entropy reinforcement learning, or KL control. Duality between control and inference in the case of linear dynamical systems has been studied in Kalman [73] and Todorov [154]. Maximum entropy objectives can be optimized efficiently and exactly in linearly solvable MDPs [155] and environments with discrete states. In linear-quadratic systems, control as inference techniques have been applied to solve path planning problems for robotics [156]. In the context of deep RL, maximum entropy objectives have been used to derive soft variants of Q-learning and policy gradient algorithms [49, 142, 117, 110]. These methods embed the standard RL objective, formulated in terms of rewards, into the framework of probabilistic inference. In contrast, we aim specifically to reformulate RL in a way that does not require specifying arbitrary scalar-valued reward functions.

In addition to studying inference problems in a control setting, we also study the problem of learning event probabilities in these models. This is related to prior work on inverse reinforcement learning (IRL), which has also sought to cast learning of objectives into the framework of probabilistic models [175, 174]. As explained in Section 2.5, our work generalizes IRL to cases where we only provide examples of a desired outcome or goal, which is significantly easier to provide in practice since we do not need to know how to achieve the goal.

Reward design is crucial for obtaining the desired behavior from RL agents [3]. Ng [115] showed that rewards can be modified, or shaped, to speed up learning without changing the optimal policy. Singh et al [150] study the problem of optimal reward design, and introduce the concept of a fitness function. They observe that a proxy reward that is distinct from the fitness function might be optimal under certain settings, and Sorg et al [151] study the problem of how this optimal proxy reward can be selected. Hadfield-Menell et al [52] introduce the problem of inferring the true objective based on the given reward and MDP. Our framework aids task specification by introducing two decisions: the selection of the inference query that is of interest (i.e., when and how many times should the agent cause the event?), and the specification of the event of interest. Moreover, as discussed in Section 5.6, we observe that our method automatically provides a reward shaping effect, allowing us to

solve otherwise hard tasks.

2.3 Preliminaries

In this section we introduce our notation and summarize how control can be framed as inference. Reinforcement learning operates on Markov decision processes (MDP), defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, r, \gamma, \rho_0)$. $\mathcal{S}, \mathcal{A}$ are the state and action spaces, respectively, r is a reward function, which is typically taken to be a scalar field on $\mathcal{S} \times \mathcal{A}$, and $\gamma \in (0, 1)$ is the discount factor. $\mathcal{T}$ and ρ_0 represent the dynamics and initial state distributions, respectively.

Control as inference

In order to cast control as an inference problem, we begin with the standard graphical model for an MDP, which consists of states and actions. We incorporate the notion of a goal with an additional variable e_t that depends on the state (and possibly also the action) at time step t, according to $p(e_t|s_t, a_t)$. If the goal is specified with a reward function, we can define $p(e_t = 1|s_t, a_t) = e^{r(s,a)}$ which, as we discuss below, leads to a maximum entropy version of the standard RL framework. This requires the rewards to be negative, which is not restrictive in practice, since if the rewards are bounded we can re-center them so that the maximum value is 0. The structure of this model is presented in Figure 2.3, and is also considered in prior work, as discussed in the previous section.

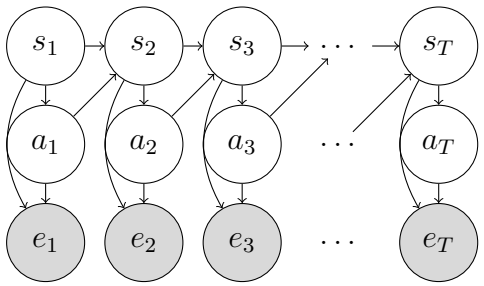


Figure 2.3: A graphical model framework for control. In maximum entropy reinforcement learning, we observe $e_{1:T} = 1$ and can perform inference on the trajectory to obtain a policy.

The maximum entropy reinforcement learning objective emerges when we condition on $e_{1:T} = 1$. Consider computing a backward message $\beta(s_t, a_t) = p(e_{t:T} = 1|s_t, a_t)$. Letting $Q(s_t, a_t) = \log \beta(s_t, a_t)$, notice that the backward messages encode the backup equations

$$Q(s_t, a_t) = r(s_t, a_t) + \log E_{s_{t+1}}[e^{V(s_{t+1})}] \qquad V(s_t) = \log \int_{a \in \mathcal{A}} e^{Q(s_t,a)} da \quad .$$

We include the full derivation in Appendix A.1, which resembles derivations discussed in prior work [175, 93]. This backup equation corresponds to maximum entropy RL, and is equivalent to soft Q-learning and causal entropy RL formulations in the special case of deterministic dynamics [49, 142]. For the case of stochastic dynamics, maximum-entropy RL is optimistic with respect to the dynamics and produces risk-seeking behavior, and we refer the reader to Appendix A.2, which covers a variational derivation of the policy objective which properly handles stochastic dynamics.

2.4 Event-based control

In control as inference, we chose $\log p(e_t = 1|s_t, a_t) = r(s, a)$ so that the resulting inference problem matches the maximum entropy reinforcement learning objective. However, we might also ask: what does the variable e_t, and its probability, represent? The connection to graphical models lets us interpret rewards as the log-probability that an event occurs, and the standard approach to reward design can also be viewed as specifying the probability of some binary event, that we might call an optimality event. This provides us with an alternative way to think about task specification: rather than using arbitrary scalar fields as rewards, we can specify the events for which we would like to maximize the probability of occurrence.

We now outline inference procedures for different types of problems of interest in the graphical model depicted in Figure 2.3. In Section 2.5, we will discuss learning procedures in this graphical model which allow us to specify objectives from data. The strength of the events framework for task specification lies in both its intuitive interpretation and flexibility: though we can obtain similar behavior in standard reinforcement learning, it may require considerable reward tuning and changes to the overall problem statement, including the dynamics. In contrast, events provides a single unified framework where the problem parameters remain unchanged, and we simply ask the appropriate queries. We will discuss:

- **ALL** query: $p(\tau|e_{1:T} = 1)$, meaning the event should happen at each time step.
- **AT** query: $p(\tau|e_{t^*} = 1)$, meaning the event should happen at a specific time t^*.
- **ANY** query: $p(\tau|e_1 = 1 \text{ or } e_2 = 1 \text{ or } ... \; e_T = 1)$ meaning the event should happen on at least one time step during each trial.

We present two derivations for each query: a conceptually simple one based on maximum entropy and message passing (see Section 2.3), and one based on variational inference, (see Appendix A.2), which is more appropriate for stochastic dynamics. The resulting variational objective is of the form:

$$J(\pi) = E_{s_{1:T}, a_{1:T} \sim q}[\hat{Q}(s_{1:T}, a_{1:T}) + H^\pi(\cdot|s_{1:T})],$$

where $\hat{Q}$ is an empirical Q-value estimator for a trajectory and $H^\pi(\cdot|s_{1:T}) = -\sum_{t=0}^{T} \log \pi(a_t|s_t)$ represents the entropy of the policy. This form of the objective can be used in policy gradient algorithms, and in special cases can also be written as a recursive backup equation for dynamic programming algorithms. We directly present our results here, and present more detailed derivations (including extensions to discounted cases) in Appendices A.3 and A.4.

ALL and AT queries

We begin by reviewing the ALL query, when we wish for an agent to trigger an event at every timestep. This can be useful, for example, when expressing some continuous task such as maintaining some sort of configuration (such as balancing on a unicycle) or avoiding an

adverse outcome, such as not causing an autonomous car to collide. As covered in Section 2.3, conditioning on the event at all time steps mathematically corresponds to the same problem as entropy maximizing RL, with the reward given by $\log p(e_t = 1|s_t, a_t)$.

Theorem 2.4.1 (ALL query)**.** *In the ALL query, the message passing update for the Q-value can be written as:*

$$Q(s_t, a_t) = \log p(e_t = 1|s_t, a_t) + \log E_{s_{t+1}}[e^{V(s_{t+1})}],$$

where $Q(s_t, a_t)$ represents the log-message $\log p(e_{t:T} = 1|s_t, a_t)$. The corresponding empirical Q-value can be written recursively as:

$$\hat{Q}(s_{t:T}, a_{t:T}) = \log p(e_t = 1|s_t, a_t) + \hat{Q}(s_{t+1:T}, a_{t+1:T}).$$

Proof. See Appendices A.3 and A.4 □

The AT query, or querying for the event at a specific time step, results in the same equations, except $\log p(e = 1|s_t, a_t)$, is only given at the specified time t^*. While we generally believe that the ANY query presented in the following section will be more broadly applicable, there may be scenarios where an agent needs to be in a particular configuration or location at the end of an episode. In these cases, the AT query would be the most appropriate.

ANY query

The ANY query specifies that an event should happen at least once before the end of an episode, without regard for when in particular it takes place. Unlike the ALL and AT queries, the ANY query does not correspond to entropy maximizing RL and requires a new backup equation. It is also in many cases more appropriate: if we would like an agent to accomplish some goal, we might not care when in particular that goal is accomplished, and we likely don't need it to accomplish it more than once. This query can be useful for specifying behaviors such as reaching a goal state, completion of a task, etc. Let the stopping time $t^* = \min\{t \geq 0|e_t = 1\}$ denote the first time that the event occurs.

Theorem 2.4.2 (ANY query)**.** *In the ANY query, the message passing update for the Q-value can be written as:*

$$Q(s_t, a_t) = \log \left(p(e_t = 1|s_t, a_t) + p(e_t = 0|s_t, a_t)E_{s_{t+1}}[e^{V(s_{t+1})}]\right)$$

where $Q(s_t, a_t)$ represents the log-message $\log p(t \leq t^* \leq T|s_t, a_t)$. The corresponding empirical Q-value can be written recursively as:

$$\hat{Q}(s_{t:T}, a_{t:T}) = \log \left(p(e_t = 1|s_t, a_t) + p(e_t = 0|s_t, a_t)e^{\hat{Q}(s_{t+1:T}, a_{t+1:T})}\right).$$

Proof. See Appendices A.3 and A.4 □

This query is related to first-exit RL problems, where an agent receives a reward of 1 when a specified goal is reached and is immediately moved to an absorbing state but it does not require the event to actually be observed, which makes it applicable to a variety of real-world situations that have uncertainty over the goal. The backup equations of the ANY query are equivalent to the first-exit problem when $p(e|s,a)$ is deterministic. This can be seen by setting $p(e=1|s,a) = r_F(s,a)$, where $r_F(s,a)$ is an goal indicator function that denotes the reward of the first-exit problem. In this case, we have $Q(s,a) = 0$ if the goal is reachable, and $Q(s,a) = -\infty$ if not. In the first-exit case, we have $Q(s,a) = 1$ if the goal is reachable and $Q(s,a) = 0$ if not - both cases result in the same policy.

Sample-based optimization using policy gradients

In small, discrete settings with known dynamics, we can use the backup equations in the previous section to solve for optimal policies with dynamic programming. For large problems with unknown dynamics, we can also derive model-free analogues to these methods, and apply them to complex tasks with high-dimensional function approximators. We can adapt the policy gradient to obtain an unbiased estimator for our variational objective:

$$\nabla_\theta J(\theta) = E_{s_{1:T}, a_{1:T} \sim \pi_\theta} \left[\sum_{t=1}^{T} \nabla \log \pi_\theta(a_t|s_t)(\hat{Q}(s_{1:T}, a_{1:T}) + H^\pi(\cdot|s_{t:T})) \right]$$

See Appendix A.5 for further explanation. Under certain simplifications we can replace $\hat{Q}(s_{1:T}, a_{1:T})$ with $\hat{Q}(s_{t:T}, a_{t:T})$ to obtain an estimator which only depends on future returns. This estimator can be integrated into standard policy gradient algorithms, such as TRPO [144], to train expressive inference models using neural networks. In the next chapter, we will also discuss how we can integrate actor-critic methods such as Soft Actor-Critic [51] with the VICE framework.

2.5 Learning event probabilities from data

In the previous section, we presented a control framework that operates on events rather than reward functions, and discussed how the user can choose from among a variety of inference queries to obtain a desired outcome. However, the event probabilities must still be obtained in some way, and may be difficult to hand-engineer in many practical situations - for example, an image-based deep RL system may need an image classifier to determine if it has accomplished its goal. In such situations, we can ask the user to instead supply examples of states or observations where the event has happened, and learn the event probabilities $p_\theta(e=1|s,a)$. Inverse reinforcement learning corresponds to the case when we assume the expert triggers an event at all timesteps (the ALL query), in which case we require full demonstrations. However, if we assume the expert is optimal under an ANY or AT query, full demonstrations are not required because the event is not assumed to be triggered at each

timestep. This means our supervision can be of the form of a desired set of states rather than full trajectories. For example, in the vision-based robotics case, this means that we can specify goals using images of a desired goal state, which are much easier to obtain than full demonstrations.

Formally, we assume that the user supplies the algorithm with a dataset of examples where the event happens. We derive variational inverse control with events (VICE) for the AT query in this section as it is conceptually the simplest, and include further derivations for the ALL and ANY queries in Appendix A.6. For the AT query, we assume examples are drawn from the distribution $\hat{p}_{data}(s_t, a_t|e_t = 1) \propto \hat{p}(e_t = 1|s_t, a_t)\hat{p}(s_t, a_t)$. , where $\hat{p}(s_t, a_t)$ is the state-action marginal of a reference policy.

We can use this data to train the factor $p_\theta(e_t = 1|s_t, a_t)$ in our graphical model, where θ corresponds to the parameters of this factor. For example, if we would like to use a neural network to predict the probability of the event, θ corresponds to the weights in this network. Our event model is accordingly of the form $p_\theta(s_t, a_t) \propto p_\theta(e_t = 1|s_t, a_t)p(s_t, a_t)$. The normalizing factor is $p_\theta(e_t = 1) = \int_{\mathcal{S},\mathcal{A}} p_\theta(e_t = 1|s_t, a_t)p(s_t, a_t)dsda$

We fit the model using the following maximum likelihood objective:

$$\mathcal{L}(\theta) = E_{\hat{p}_{data}}[\log p_\theta(s_t, a_t)] = E_{\hat{p}_{data}}[\log p_\theta(e_t = 1|s_t, a_t)] - \log p_\theta(e_t = 1) \tag{2.1}$$

The gradient of this objective with respect to θ is given by

$$\nabla_\theta \mathcal{L}(\theta) = E_{\hat{p}_{data}}[\nabla_\theta \log p_\theta(e_t = 1|s_t, a_t)] - E_{p_\theta(s_t,a_t)}[\nabla_\theta \log p_\theta(e_t = 1|s_t, a_t)].$$

A tractable way to compute this gradient is to use the previously mentioned variational inference procedure (Appendix A.2) to compute the distribution $q(s, a)$ to approximate $p_\theta(s, a)$, and then use it to evaluate the expectations for the gradient. This corresponds to an EM-like iterative algorithm, where we alternate training our event probability $p_\theta(e_t = 1|s_t, a_t)$ given the current q and training a policy $q(a|s)$ to draw samples from the distribution $p_\theta(s_t, a_t)$ in order to estimate the second term of the gradient. This procedure is analogous to MaxEnt IRL [175], except that, depending on the type of query we use, the event may not necessarily happen at every time step, and the data consists only of individual states rather than entire demonstrations. In high-dimensional settings, we can adapt the method of Fu et al [39], which alternates between training $p_\theta(s, a)$ by fitting a discriminator to distinguish policy samples from dataset samples, and training a policy by performing inference on the corresponding graphical model (which corresponds to trying to fool the discriminator). We present our algorithm pseudocode in Algorithm 1. In our experiments, we use a variant of TRPO [144] (as discussed in Section 2.4) to update the policy with respect to the event probabilities (line 7).

Interestingly, as we will discuss in the experimental evaluation, we've found in many cases that learning the event probabilities using VICE actually resulted in better performance than reinforcement learning directly from binary event indicators even when these indicators are available. Part of the explanation for this phenomenon is that the learned probabilities are smoother than binary event indicators, and therefore can provide a better shaped reward function for RL.

Algorithm 1 VICE: Variational Inverse Control with Events

1: Obtain examples of expert states and actions s_i^E, a_i^E
2: Initialize policy π and binary discriminator D_θ.
3: **for** step n in $\{1, \ldots, \text{N}\}$ **do**
4: Collect states and actions $s_i = (s_1, ..., s_T), a_i = (a_1, ..., a_T)$ by executing π.
5: Train D_θ via logistic regression to classify expert data s_i^E, a_i^E from samples s_i, a_i.
6: Update $\log \hat{p}(e = 1|s, a) \leftarrow \log D_\theta(s, a) - \log(1 - D_\theta(s, a))$
7: Update π with respect to $\log \hat{p}(e = 1|s, a)$ using the appropriate inference objective.
8: **end for**

2.6 Experimental evaluation

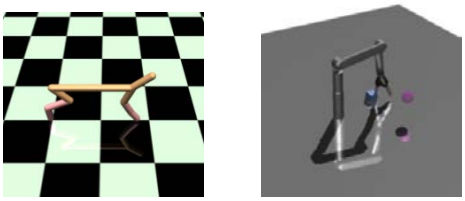


Figure 2.4: HalfCheetah and Lobber tasks.

Our experimental evaluation aims to answer the following questions: (1) How does the behavior of an agent change depending on the choice of query? We study this question in the case where the event probabilities are already specified. (2) Does our event learning framework (VICE) outperform simple alternatives, such as offline classifier training, when learning event probabilities from data? We study this question in settings where it is difficult to manually specify a reward function, such as when the agent receives raw image observations. (3) Does learning event probabilities provide better shaped rewards than the ground truth event indicators?

Inference with pre-specified event probabilities

We first demonstrate how the ANY and ALL queries in our framework result in different behaviors. We adapt TRPO [144], a natural policy gradient algorithm, to train policies using our query procedures derived in Section 2.4. Our examples involve two goal-reaching domains, HalfCheetah and Lobber, shown in Figure 2.4. The goal of HalfCheetah is to navigate a 6-DoF agent to a goal position, and in Lobber, a robotic arm must throw an block to a goal position. To study the inference process in isolation, we manually design the event probabilities as $e^{-||x_{agent} - x_{target}||_2}$ for the HalfCheetah and $e^{-\|x_{block} - x_{goal}\|^2}$ for the Lobber.

Query	Avg. Dist	Min. Dist
HalfCheetah-ANY	1.35 (0.20)	**0.97** (0.46)
HalfCheetah-ALL	**1.33** (0.16)	2.01 (0.48)
HalfCheetah-Random	8.95 (5.37)	5.41 (2.67)
Lobber-ANY	0.61 (0.12)	**0.25** (0.20)
Lobber-ALL	**0.59** (0.11)	0.36 (0.21)
Lobber-Random	0.93 (0.01)	0.91 (0.01)

Table 2.1: Results on HalfCheetah and Lobber tasks (5 trials). The ALL query generally results in superior returns, but the ANY query results in the agent reaching the target more accurately. Random refers to a random gaussian policy.

The experimental results are shown in Table 2.1. While the average distance to the goal for both queries was roughly the same, the ANY query results in a much closer minimum distance. This makes sense, since in the ALL query the agent is punished for every time step it is not near the goal. The ANY query can afford to receive lower cumulative returns and instead has max-seeking behavior which more accurately reaches the target. Here, the ANY query better expresses our intention of reaching a target.

Learning event probabilities

We now compare our event probability learning framework, which we call variational inverse control with events (VICE), against an offline classifier training baseline. We also compare our method to learning from true binary event indicators, to see if our method can provide some reward shaping benefits to speed up the learning process. The data for learning event probabilities comes from success states. That is, we have access to a set of states $\{s_i^E\}_{i=1...n}$, which may have been provided by the user, for which we know the event took place. This setting generalizes IRL, where instead of entire expert demonstrations, we simply have examples of successful states. The offline classifier baseline trains a neural network to distinguish success state ("positives") from states collected by a random policy. The number of positives and negatives in this procedure is kept balanced. This baseline is a reasonable and straightforward method to specify rewards in the standard RL framework, and provides a natural point of comparison to our approach, which can also be viewed as learning a classifier, but within the principled framework of control as inference. We evaluate these methods on the following tasks:

Table 2.2: Results on Maze, Ant and Pusher environments (5 trials). The metric reported is the final distance to the goal state (lower is better). VICE performs better than the classifier-based setup on all the tasks, and the performance is substantially better for the Ant and Pusher task.

	Query type	Classifier	VICE (ours)	True Binary
Maze	ALL	0.35 (0.29)	**0.20** (0.19)	0.11 (0.01)
	ANY	0.37 (0.21)	0.23 (0.15)	
Ant	ALL	2.71 (0.75)	0.64 (0.32)	1.61 (1.35)
	ANY	3.93 (1.56)	**0.62** (0.55)	
Push	ALL	0.25 (0.01)	**0.09** (0.01)	0.17 (0.03)
	ANY	0.25 (0.01)	0.11 (0.01)	

Maze from pixels. In this task, a point mass needs to navigate to a goal location through a small maze, depicted in Figure 4.3. The observations consist of 64x64 RGB images that correspond to an overhead view of the maze. The action space consists of X and Y forces on the robot. We use CNNs to represent the policy and the event distributions, training with 1000 success states as supervision.

Ant. In this task, a quadrupedal "ant" (shown in Figure 4.3) needs to crawl to a goal location, placed 3m away from its starting position. The state space contains joint angles and XYZ-coordinates of the ant. The action space corresponds to joint torques. We use 500 success states as supervision.

Pusher from pixels. In this task, a 7-DoF robotic arm (shown in Figure 4.3) must push a cylinder object to a goal location. The state space contains joint angles, joint velocities and

64x64 RGB images, and the action space corresponds to joint torques. We use 10K success states as supervision.

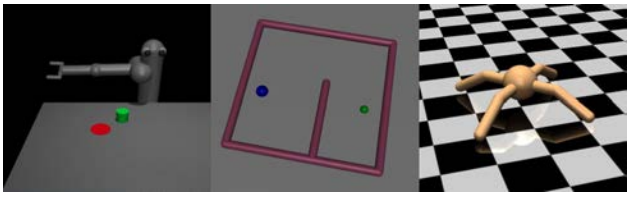


Figure 2.5: Visualizations of the Pusher, Maze, and Ant tasks. In the Maze and Ant tasks, the agent seeks to reach a pre-specified goal position. In the Pusher task, the agent seeks to place a block at the goal position.

Training details and neural net architectures can be found in Appendix A.7. We also compare our method against a reinforcement learning baseline that has access to the true binary event indicator. For all the tasks, we define a "goal region", and give the agent a $+1$ reward when it is in the goal region, and 0 otherwise. Note that this RL baseline, which is similar to vanilla RL from sparse rewards, "observes" the event, providing it with additional information, while our model only uses the event probabilities learned from the success examples and receives no other supervision. It is included to provide a reference point on the difficulty of the tasks. Results are summarized in Table 2.2, and detailed learning curves can be seen in Figure 2.6. We note the following salient points from these experiments.

VICE outperforms naïve classifier. We observe that for *Maze*, both the simple classifier and our method (VICE) perform well, though VICE achieves lower final distance. In the *Ant* environment, VICE is crucial for obtaining good performance, and the simple classifier fails to solve the task. Similarly, for the *Pusher* task, VICE significantly outperforms the classifier (which fails to solve the task). Unlike the naïve classifier approach, VICE actively integrates negative examples from the current policy into the learning process, and appropriately models the event probabilities together with the dynamical properties of the task, analogously to IRL.

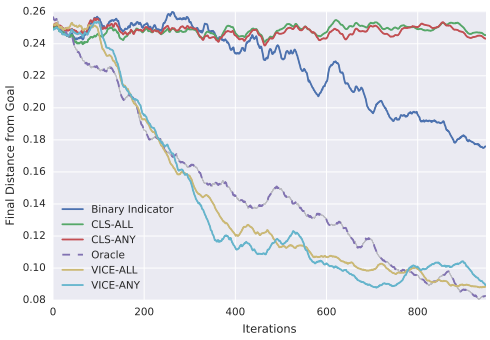


Figure 2.6: Results on the Pusher task (lower is better), averaged across five random seeds. VICE significantly outperforms the naïve classifier and true binary event indicators. Further, the performance is comparable to learning from an oracle hand-engineered reward (denoted in dashed lines). Curves for the Ant and Maze tasks can be seen in Appendix A.7.

Shaping effect of VICE. For the more difficult ant and pusher domains, VICE actually outperforms RL with the true event indicators. We analyze this shaping effect further in Figure 2.6: our framework obtains performance that is superior to learning with true event indicators, while requiring much weaker supervision. This indicates that the event probability distribution learned by our method has a reward-shaping effect, which greatly simplifies the policy search process. We further compare our method against a hand-engineered shaped reward, depicted in dashed

lines in Figure 2.6. The engineered reward is given by $-0.2 * \|x_{block} - x_{arm}\| - \|x_{block} - x_{goal}\|$, and is impossible to compute when we don't have access to x_{block}, which is usually the case when learning in the real world. We observe that our method achieves performance that is comparable to this engineered reward, indicating that our automated shaping effect is comparable to hand-engineered shaped rewards.

2.7 Conclusion

In this chapter, we described how the connection between control and inference can be extended to derive a reinforcement learning framework that dispenses with the conventional notion of rewards, and replaces them with events. Events have associated probabilities. which can either be provided by the user, or learned from data. Recasting reinforcement learning into the event-based framework allows us to express various goals as different inference queries in the corresponding graphical model. The case where we learn event probabilities corresponds to a generalization of IRL where rather than assuming access to expert demonstrations, we assume access to states and actions where an event occurs. IRL corresponds to the case where we assume the event happens at every timestep, and we extend this notion to alternate graphical model queries where events may happen at a single timestep.

Chapter 3

End-to-End Robotic Reinforcement Learning without Reward Engineering

3.1 Introduction

In the previous chapter, we introduced variational inverse control with events (VICE) [41], a framework for specifying objectives for a task using user-provided goal (or outcome) examples. As explained in Section 2.5, VICE involves training a classifier to distinguish between user-provided goal examples and data collected by the RL policy, and the success probabilities from this classifier can then be used as a reward for training reinforcement learning agents to achieve the specified goal. However, VICE relies entirely on the positive outcome examples provided at the beginning of training to understand the task, which in practice means that a large number of examples is needed. Further, because VICE relies on on-policy RL for training the policy and for gathering negative examples for the classifier, it requires millions of samples, which may be impractical in the real world. In this chapter, we address both of these issues to enable end-to-end reinforcement learning on real robots from pixel observations, and without any task-specific engineering for obtaining rewards. To remove the reliance on a large

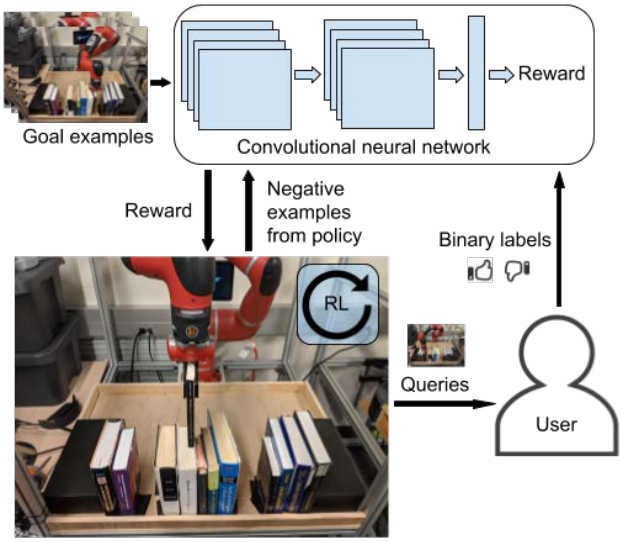


Figure 3.1: **Illustration of our approach.** During the reward learning process, the robot periodically queries a user with images, and the user provides a binary label indicating whether or not this image corresponds to a successful outcome.

amount of positive examples provided up front, we elicit a small number of additional queries from a human user as the robot collects additional experience. These active queries are selected based on uncertainty estimates from the classifier that is being used as a reward function, and allow us to learn effective rewards from a small number of initial examples. Further, we extend both the policy learning and the classifier training procedure in VICE to the off-policy setting, which allows us to learn robotic skills in the real world with only 1-4 hours of interaction time, entirely from image observations.

Our primary contribution in this chapter is a method for learning robotic skills from high-dimensional observations, such as images, without hand-designing reward functions. Our method uses a small number of examples of positive outcomes (without demonstrations), followed by a modest number of additional binary active queries, where the robot asks the user if a particular outcome is successful or not. Our approach is based on efficient off-policy reinforcement learning, making it well-suited for real-world learning. Our experiments demonstrate that our method can learn a variety of real-world robotic manipulation skills, directly from images, and directly in the real world. Results include draping cloth over an object, placing books on a bookshelf, and pushing mugs onto a coaster. Learning requires minimal user supervision and only 1-4 hours of interaction time, which is substantially less than that of prior work [50, 72, 129, 96, 24].

3.2 Relation To Prior Work

Reinforcement learning has been applied to a wide variety of robotic manipulation tasks, including grasping objects [72], in-hand object manipulation [120, 61, 131, 89], manipulating fluids [141], door opening [167, 11], and cloth folding [105]. However, applications of RL in the real world require considerable effort to design and evaluate the reward function. For example, using thermal cameras for tracking fluids [141], mocap sensors [80] or computer vision systems [138] for tracking objects, and accelerometers for determining the state of a door [167]. Since such instrumentation needs to be done for any new task that we may wish to learn, it poses a significant bottleneck to widespread adoption of reinforcement learning for robotics, and precludes the use of these methods directly in open-world environments that lack this instrumentation.

Data-driven approaches for reward specification [116, 1, 33, 59, 38, 176, 134, 26, 102, 12, 67] seek to overcome this issue, but typically require demonstration data to acquire rewards. Such data can be onerous and time-consuming for users to provide. Recent work on active learning for inverse RL has sought to reduce the required number of demonstrations [7, 15, 99, 13], but still requires some number of demonstrations to be provided manually. Our method only requires a modest number of examples of successful outcomes, followed by binary queries where the user indicates whether a particular outcome that the robot achieved is successful or not. Both of these can be provided easily, without any teleoperation or kinesthetic teaching. Related to our method, Daniel et al [16] propose to learn rewards from active queries that elicit numerical scores. In contrast to this approach, our method uses only binary success

queries, which are easier to provide, and can be readily combined with deep networks for learning skills from image observations. Another line of research that queries binary feedback from humans is that of learning from human preferences [67, 12], but these techniques have so far proven to be quite expensive in the deep reinforcement learning setting in terms of both the supervision needed from humans, and the overall sample complexity. Even in simulation with low-dimensional observations, Christiano et al [12] make about one thousand queries from humans (we make 50-75 such queries in the real world), and require tens of millions of timesteps of interaction with the environment (we require around tens of thousands of such interactions). Furthermore, comparing trajectories is often a harder form of feedback to elicit from humans, especially towards the start of training where all trajectories might be equally undesirable.

Classifier training serves as an alternative to inverse reinforcement learning for data-driven reward specification [159, 164, 129]. However, using classifier-based rewards for reinforcement learning is prone to exploitation by RL agents, as such agents quickly capitalize on any imperfections in the learned classifier [41, 164]. VICE (discussed in the previous chapter) overcomes this issue by adversarially mining negatives form the learned policy, but usually requires a large number of samples to learn due to using an on-policy algorithm [144] for policy improvement and classifier updates. Furthermore, it usually utilizes a large number of goal examples (on the order of 50K for image observations) for successfully learning the task. Our approach overcomes both of these issues, and we demonstrate that it can be used for practical real-world reinforcement learning.

3.3 Preliminaries

While we derived the VICE algorithm in a control as inference framework in the previous chapter, in this section we take a look at the same algorithm through a different lens: that of goal classifier training. We first describe a naïve classifier training procedure for reward inference, in which all positive and negative examples for training the classifier are provided up front, and then describe the VICE version, where data collected during RL is added to the set of negative examples.

Classifier-Based Rewards Engineering reward functions for object manipulation is difficult, especially when using image observations, because it requires identifying the state of the objects in the world and formulating a reliable success condition programmatically. Indeed, it is often easier for users to state whether a given outcome is successful or not than to write a program that will do so automatically. However, current RL algorithms require so many episodes that labeling the reward in each one manually would be impractical. A reasonable alternative is to use a goal classifier [164, 159], where the user provides a dataset of example states (e.g., images) before training the policy, denoted $\mathcal{D} := \{(s_n, y_n)\}$, and a binary classifier $g(s)$ is trained to predict whether a given state is a success or failure. Once trained, the classifier can be used to provide a reward during reinforcement learning. If the

classifier provides a distribution $p_g(y|s)$, then a particularly convenient form for the reward is given by $\log p_g(y|s)$. As discussed in the last chapter, this has an appealing theoretical interpretation based on a connection to control as inference [93], and in practice can provide some degree of shaping, as log-probabilities often increase smoothly as the agent approaches the goal.

Prior work generally requires both successful and failed examples to be part of $\mathcal{D}$ [164, 159]. When a policy is trained with this classifier, the policy can learn to *exploit* the classifier, reaching states that are different from those that the classifier was trained on and fool it into outputting a success label erroneously [41]. The degree of exploitation is strongly dependent on how the negative examples are provided, and can only be avoided if a comprehensive set of negative examples covering the entire state space is supplied.

VICE VICE overcomes this exploitation by alternating between training a *discriminator* to discriminate between the positive examples and the current policy's rollouts, and optimizing the policy with respect to a maximum entropy objective, using $\log p(y_t = 1|s_t, a_t)$ as the reward. The discriminator in VICE is parameterized by ψ and given by the following equation:

$$D_\psi(s, a) = \frac{\exp(f_\psi(s, a))}{\exp(f_\psi(s, a)) + \pi(a|s)}. \tag{3.1}$$

As shown in the last chapter, $f_\psi(s, a)$ recovers $\log p(e = 1|s, a)$ at convergence of this adversarial learning procedure. In the next section, we will describe an approach that instead uses a modest number of active queries from the user to address the exploitation problem.

3.4 Reinforcement Learning with Active Queries

The goal of our method, which we call reinforcement learning with active queries (RAQ), is to learn robotic skills via reinforcement learning without requiring hand-engineered reward functions, using data that can be easily obtained from the user. More specifically, we train classifiers to distinguish between goal and non-goal observations, and use them to compute rewards. Instead of learning this classifier from a pre-specified static dataset alone (as done in prior work [159, 164]), we introduce an active learning framework that queries a user for binary success labels for states that it would like to obtain ground truth labels for. This addresses two major challenges with classifier-based rewards: it removes the need for the user to provide a comprehensive set of negative examples up front, and it mitigates the classifier exploitation problem. Let $\{s_n\}_{n=1}^t$ denote the set of states that the agent encounters over the learning process, where t is the total number of environment steps that the agent has taken so far. At any given step t, our algorithm decides which states from $\{s_n\}_{n=1}^t$ (if any) it should query a label for. We first introduce and motivate our query mechanism, and we then show how it can be combined with a classifier-based reward learning technique to obtain a practical algorithm for reinforcement learning in the real world.

Active Queries

If the robot requests user labels for every single state it sees, it will have a very accurate reward. However, a typical RL run will collect tens or even hundreds of thousands of states worth of data, as discussed in Section 3.8, and labeling all of these states is impractical. Minimizing the required number of queries depends critically on the mechanism that decides which state should be labeled. The active learning literature provides a few potential mechanisms based on uncertainty, such as the maximum entropy heuristic. In practice, we found that the maximum entropy heuristic does not actually produce very good results, since the goal is not so much to obtain accurate goal classification everywhere, but rather to eliminate the "exploitation" problem, such that the classifier does not output false positives. To that end, we found that the most effective mechanism to select which states to label was to select the previously-unlabeled states with the highest probability of success according to the classifier. Recall that our reward is provided by a binary classifier, which specifies a distribution $p_g(y|s)$, where y is a binary variable indicating success. Following VICE (Chapter 2), the reward is given by $\log p_g(y|s)$. We can select the state s_k to label from the set of observed states according to

$$k = \arg\max_t \ \log p_g(y = 1|s_t) \ \forall\, t \text{ since last query.}$$

For most practical tasks, this query mechanism is also much more selective than the maximum entropy rule. First, negative examples are much easier to obtain than positive examples for most tasks, so requesting labels for only the potential positive examples avoids superfluous queries for high-entropy states that are unlikely to be informative of success. Second, because the policy is explicitly trying to visit states with high $p_g(y|s)$, classifier exploitation is due to false positives rather than negatives [41]. Since only states with positive predictions can be false positives, querying for labels for these states is an effective mechanism for mitigating classifier exploitation.

Aside from selecting which states to label, we must also choose how often to request labels. We adopt a simple scheme where labels are queried at fixed intervals. We found that simply choosing a query frequency based on the expected training length and a query budget was sufficient. For the real-world robot experiments presented this paper, we adjust the frequency so that we make between 25 to 75 active queries for a single run of a reinforcement learning experiment. Details on this can be found in Section 3.8.

Classifier-Based RL with Active Queries

We now explain how we combine our active query framework with classifier-based rewards for reinforcement learning. Our approach is summarized in Algorithm 2. Similar to standard classifier reward-based RL, we first train a classifier g on an initial dataset $\mathcal{D}$. The RL algorithm then uses $\log p_g(y|s)$ as the reward, and runs for a predefined number of time steps, at which point we select a new state to query by selecting the state with the largest value for $\log p_g(y|s)$. This labeled state is added to the dataset $\mathcal{D}$, and the classifier is then fine-tuned.

Algorithm 2 Reinforcement learning with active queries (RAQ)

Require: initial $\mathcal{D} := \{(s_n, y_n)\}$
Update the parameters of g to minimize $\sum_n \mathcal{L}(g(s_n), y_n)$
Initialize policy π, critic Q, replay buffer $\mathcal{R}$
for each iteration **do**
  for each environment step **do**
    $a_t \sim \pi_\theta(a_t|s_t)$
    $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
    $\mathcal{R} \leftarrow \mathcal{R} \cup \{(s_t, a_t, s_{t+1})\}$
  end for
  for each gradient step **do**
    Sample from $\mathcal{R}$
    Compute rewards: $r(s_t) \leftarrow \log p_g(y_t|s_t)$
    Update π and Q according to Haarnoja et al [51]
  end for
  if active query **then**
    $k \rightarrow \arg\max \log p_g(y_t|s_t)$ for all t since the last query
    if s_k is a successful outcome **then**
      $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_k, 1)\}$
    else
      $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_k, 0)\}$
    end if
    Update g to minimize $\sum_n \mathcal{L}(g(s_n), y_n)$
  end if
end for

We then continue training with RL, and repeat the process. This procedure is repeated until convergence or until a fixed budget of samples or queries is exceeded.

3.5 Off-Policy VICE with Active Queries

While standard RAQ, as described in Section 4.4, is effective in mitigating the classifier exploitation problem and can enable reinforcement learning with classifier-based rewards, it only utilizes a very small fraction of the data that is collected when running RL. RL algorithms typically collect tens or hundreds of thousands of transitions when learning to solve a robotic task in the real world, but RAQ only makes tens of queries, barely using 0.1% of the data collected. Ideally, we would like to make use of *all* the data collected during RL. However, as discussed in the previous chapter, VICE can effectively overcome the classifier exploitation problem, and does so by using all of the data collected during RL without making any active queries. In this section, we first show how VICE can be extended

into the off-policy setting, providing a practical method for robotic RL. We then discuss how RAQ can be combined with VICE. The resulting method, which we call VICE-RAQ, combines the best properties of both techniques.

Off-Policy VICE

As discussed in the previous chapter, the VICE algorithm is implemented as an on-policy reinforcement learning procedure, typically using policy gradient methods such as TRPO [144]. This makes it difficult to use for real-world robotic learning. Furthermore, VICE requires the success examples to include both the state s and action a, which is unnatural for a user to provide. In the following sections, we describe an extension of VICE that lifts both of these limitations, and then present our complete VICE-RAQ algorithm, which combines VICE with active queries.

In order to make VICE practical to use for real-world robotic learning, we must extend it so as to make it compatible with efficient off-policy deep reinforcement learning methods, and remove the need to obtain ground truth action labels for the positive examples, so that the user can readily specify examples simply by showing the robot example images of successful outcomes. Extending VICE to the off-policy setting first requires an off-policy reinforcement learning algorithm, and the soft actor-critic algorithm [51] provides one such method. Next, we need a way to train the discriminator (shown in Equation 3.1). While in principle this would require importance sampling if using off-policy data from the replay buffer $\mathcal{R}$, prior work has observed that adversarial IRL can drop the importance weights both in theory [34] and in practice [82]. We adopt the same approach, and sample negative examples for the discriminator directly from $\mathcal{R}$ without importance weighting.

The standard VICE algorithm also requires the user-specified success examples to consist of state-action tuples (s, a), since both s and a are required to update the discriminator $D_\theta(s, a)$ when it has the form in Equation 3.1. Even when $f_\psi(s)$ does not depend on the action, the term $\pi(a|s)$ in the denominator does. Providing the actions is unnatural to the user, since the examples consist of isolated individual states showing successful outcomes (e.g., images of successful outcomes). Therefore, we remove the need for the user to specify actions by integrating out the actions for the positive examples in the VICE discriminator update, by using the current policy $\pi(a|s)$. This amounts to sampling the actions for the positives, denoted a_i^E, from $\pi(a|s_i^E)$ as shown on line 10 in Algorithm 3. At convergence, since $\pi(a|s)$ approaches the expert's policy, this simplification produces the same action distribution, and therefore this update has the same fixed point as when the user supplies ground truth actions.

Off-Policy VICE-RAQ

Since the set of positives in VICE is fixed at the start of the algorithm, it typically requires a large set of positive examples provided by the user to begin with in order to prevent overfitting, sometimes as many as several thousand. By integrating VICE with our active query framework, we can substantially decrease the number of examples that are required,

Algorithm 3 Off-Policy VICE-RAQ with soft actor-critic

Require: : $\mathcal{D}_i := \{(s_n, 1)\}$
Initialize f_ψ (described in Equation 3.1), policy π, critic Q, replay buffer $\mathcal{R}$
for each iteration **do**
  for each environment step **do**
    $a_t \sim \pi_\theta(a_t|s_t)$
    $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
    $\mathcal{R} \leftarrow \mathcal{R} \cup \{(s_t, a_t, s_{t+1})\}$
  end for
  for each gradient step for f_ψ **do**
    Sample positives from $\mathcal{D}$ and negatives from $\mathcal{R}$
    Sample action labels $a_i^E \sim \pi(a|s_i^E)$
    Update f_ψ using Equation 3.1 as discriminator
  end for
  for each gradient step for the policy π **do**
    Sample from $\mathcal{R}$
    Compute rewards: $r(s_t) \leftarrow f_\psi(s_t)$
    Update π and Q according to Haarnoja et al [51]
  end for
  if active query **then**
    $k \rightarrow \arg\max f_\psi(s_t)$ for all t since the last query
    if s_k is a successful outcome **then**
      $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_k, 1)\}$
    end if
  end if
end for

at the cost of several tens of binary queries, as in standard RAQ. To integrate RAQ with VICE, we simply add the active queries, as in Algorithm 2, and append the labeled state to the example set if the label is positive. If the label is negative, there is no need to append the state, since VICE already uses all sampled states as negatives. The full VICE-RAQ algorithm is summarized in Algorithm 3. We start out with a dataset $\mathcal{D}$ consisting of positive examples. Every iteration, we collect data from the environment and use it to update f_ψ, the policy, and the Q-function. At fixed intervals, we query the user using our active query mechanism discussed in Section 3.4, and update our dataset $\mathcal{D}$ if the queried state is labeled as a successful outcome. We continue running RL and updating the event probabilities f_ψ, utilizing both our initial dataset and any positives that we obtained from successful queries. This procedure continues until the policy converges, or after a specific period of time.

3.6 VICE-RAQ for Image-based Manipulation

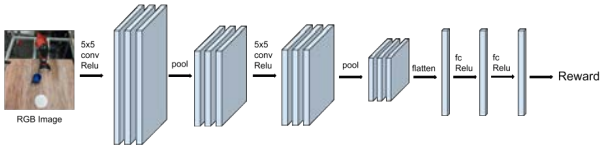


Figure 3.2: Our convolutional neural network architecture. The same architecture is used for the policy, critic, and the learned reward function.

We implemented our methods on top of a standard open-source implementation of the soft actor-critic algorithm [50]. We use standard hyperparameter values used for continuous control problems in this implementation, details of which can be found in Appendix B.1. The policy and the critic for each task are represented using convolutional neural networks, shown in Figure 3.2. It consists of two convolutional layers, each of which is followed by a max-pooling layer, with 8 filters in each of the convolutional layers for simulated tasks, and 32 filters per layer for real world tasks. The flattened output of the convolutional layers is followed by two fully-connected hidden layers with 256 units each. The ReLU non-linearity is applied after each of the convolutional and fully-connected layers. The reward function $f_\psi(s)$ is also represented using a convolutional neural network with the same architecture.

We use log-probabilities from a neural network-based classifier as reward for reinforcement learning. However, neural networks are known to drastically change their outputs even with small changes to the input [153]. Thus, they provide a hard decision boundary between different classes, which in our case is similar to running RL with sparse rewards. On the other hand, if the output probabilities of the classifier smoothly transition between positive and negative labels, this would provide a more shaped reward, increasing both the stability and efficiency of the reinforcement learning process. To this end, we found *mixup* regularization to be particularly well-suited for smoothing the classifier predictions [171]. We briefly summarize this technique here. Let $\mathbf{s}_i, \mathbf{s}_j$ be any two inputs to the classifier—either from the replay buffer, or from the set of human-provided goal examples—and let y_i, y_j be the corresponding labels. Mixup regularization takes these input/output pairs, and generates the following *virtual* training distribution:

$$\begin{aligned} \tilde{\mathbf{s}} &= \lambda \mathbf{s}_i + (1-\lambda)\mathbf{s}_j \\ \tilde{y} &= \lambda y_i + (1-\lambda) y_j, \end{aligned} \tag{3.2}$$

where $\lambda \sim \text{Beta}(\alpha, \alpha)$. The mixup hyperparameter α controls the extent of mixup, and higher α corresponds to a higher level of mixup (i.e., the sampled λ are closer to 0.5 than to 0). Zhang et al [171] showed that mixup enables smoother transitions between different classes by encouraging linear behavior, and our experiments indicate that it indeed makes the learned reward function smoother and more amenable to reinforcement learning. For details, see Appendix B.1.

3.7 Simulated Experiments

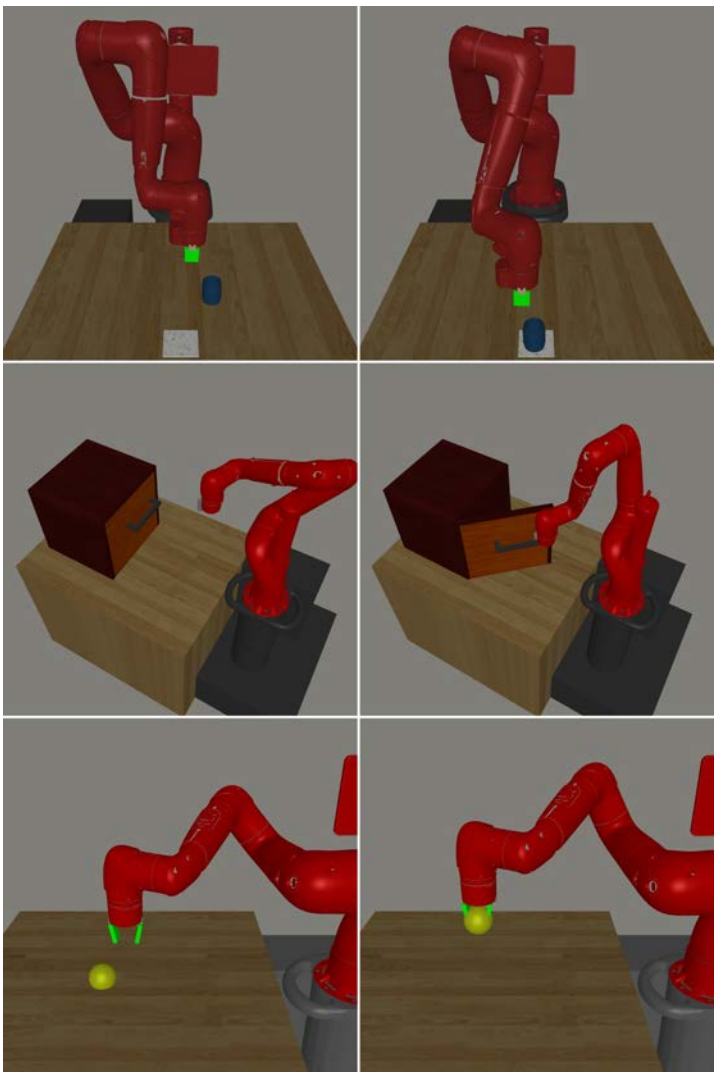


Figure 3.3: **Simulated tasks.** The left and right columns depict possible starting states and goal states for each task. In the *Visual Pusher* task (top), the goal is to push a mug onto a coaster, with a randomized initial position of the mug. The middle row shows the *Visual Door Opening* task, where the goal is to open a door of a cabinet by 45 degrees. Initially, the door is either completely closed with probability 0.5, or open up to 15 degrees. In the *Visual Picker* task (bottom), the goal is to pick up a tennis ball from a table and hold it at a particular spot 20cm above the table. The initial position of the tennis ball on the table is randomized.

Our simulated experiments are aimed at providing a rigorous comparison between RAQ (Section 3.4), our off-policy extension of VICE (Section 3.5), VICE-RAQ (Section 3.5), and standard classifier-based rewards [164, 159], as well as a comparison to RL with sparse rewards. The ability to run multiple trials for every method on multiple tasks allows us to provide a detailed comparison, and we have made an open source release of our code to reproduce all experiments in this section[1]. The simulated experiments are conducted on three different tasks, illustrated in Figure 3.3. Each of the tasks is performed using end-effector position control with a 7-DoF robotic arm modeled on the Rethink Sawyer. For the picking task, the policy can also continuously control the opening and closing of the gripper. The observation space of the robot is a 48x48 RGB image. No other observations (such as joint angles or end-effector position) are provided to the policy. The goal is specified using a set of goal images, as depicted in Figure 3.3. We start with 10 goal examples for each of the task, and perform an active query once every 25K timesteps for the *Visual Pusher* task, once every 10K timesteps for *Visual Door Opening* task, and once every 1K timesteps for the *Visual Picker* task. More details about the tasks can be found in Appendix B.1.

[1] `https://github.com/avisingh599/reward-learning-rl`

We perform runs with five different random seeds for every method and task. The results of our experiments are shown in Figure 3.4, and videos can be found on our project website[2]. Separate learning curves for all random seeds for each method can be found in Appendix B.1. For the pushing task, we observe that both off-policy VICE and VICE-RAQ perform well, and solve the task for all random seeds, while RAQ, the naïve classifier and the ground truth sparse reward baselines only succeed for some of the runs. For the harder door opening task, we observe that VICE-RAQ outperforms all other methods, and is able to achieve a success rate of nearly 100% across random seeds. The performance of off-policy VICE and RAQ is comparable for this task, with some runs failing and others succeeding for both the methods. The naiïve classifier and the sparse reward baselines completely fail for all seeds but one. We see a similar pattern for the *Visual Picker* task, with VICE-RAQ strongly outperforming all other methods, and naïve classifier-based reward and sparse rewards failing to solve the task for most seeds. Our off-policy extension of VICE slightly outperforms RAQ for this task.

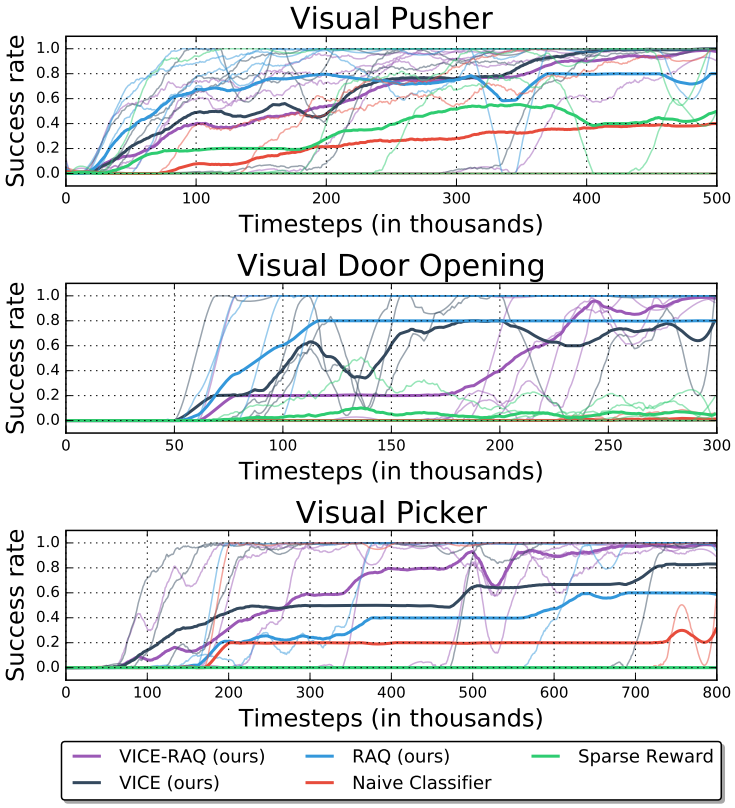


Figure 3.4: Results on simulated tasks. Each method is run with five different random seeds for each task. The lines in bold indicate the mean across five runs, while the faint lines depict the individual random seeds for each method. We observe that VICE-RAQ achieve the best performance on all tasks, with RAQ being comparable to VICE-RAQ on the Visual Pusher task. We also notice that both RAQ and VICE have significant variance among runs, while VICE-RAQ achieves relatively low variance towards the end of the learning process.

3.8 Real-World Experiments

[2] https://sites.google.com/view/reward-learning-rl/home

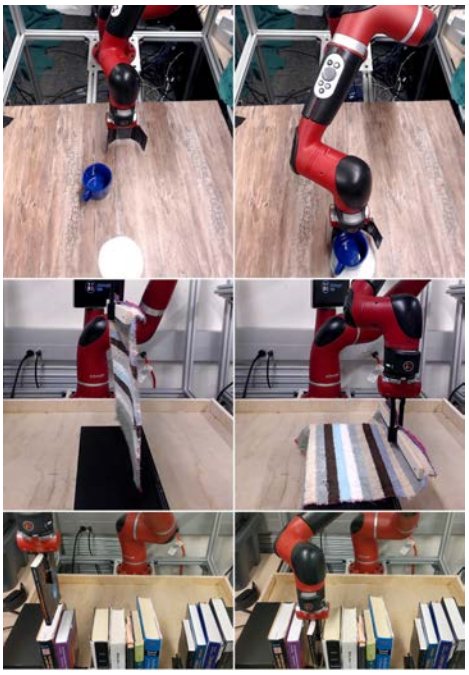


Figure 3.5: **Real-world tasks.** The left column depicts possible starting states of the task, while the right column depicts possible goal states. The top row shows the *Visual Pusher* task, in which the goal is to push a mug onto a coaster, and the initial position of the mug is randomized. The middle row presents the *Visual Draping* task, where the goal is to drape a cloth over an object. The bottom row depicts the *Visual Bookshelf* task, where the goal is to inset a book in one of the multiple empty slots in the bookshelf.

Our real world experiments aim to study end-to-end reinforcement learning from pixels without manually engineered rewards nor instrumentation of the environment to measure rewards. We evaluate all of our methods, RAQ, off-policy VICE, and VICE-RAQ, on three complex tasks from vision: pushing a mug onto a coaster, draping a cloth over a box, and a task that requires the robot to insert a book onto a shelf between other books. We also provide a naïve classifier-based baseline as a comparison for all of the tasks. The goal of these experiments is to verify that VICE-RAQ can successfully learn complex tasks, including non-prehensile manipulation (pushing), tasks with multiple success conditions (where the book can be placed in one of several locations), and tasks with deformable objects (cloth draping). For all our experiments, we use end-effector position control on a 7 DoF Sawyer robotic manipulator, and our observations consist of an RGB image of size 84×84. We do not make use of robot joint angles or end effector-positions. The final success rates of the trained policies for each of these tasks are shown in Figure 3.1. We first discuss the individual tasks, and then the experimental results.

Visual Pushing This non-prehensile manipulation task (depicted in Figure 3.5) requires the robot to push a mug onto a coaster. The position of the mug and coaster must be inferred from the images, and the initial position of the mug varies between different trials. In order to succeed, the robot should push the mug such that is gets placed completely within the coaster. Here, the robot must make use of the images to determine whether it has achieved the goal successfully, and the increased challenge of non-prehensile manipulation allows to better differentiate the performance of the different methods.

Figure 3.6: In this figure, we demonstrate why learning a reward function on pixels is necessary for solving complex tasks in the real world. The task here is to drape a cloth over a box. The top row shows a rollout from the final policy trained by our method, while the bottom row shows a rollout from a policy trained on a hand-defined reward on robot state alone. Our policy is able to successfully drape the cloth over the box, while the policy trained without vision only sees the end-effector position, which it succeeds in moving to the right place, but fails to drape the cloth on the box.

Figure 3.7: In this figure, we demonstrate how classifiers are more expressive than goal images for describing a task. The goal for this task is place a book in any empty slot in a bookshelf, and the initial position of the robot arm holding the book is randomized. The top row shows a rollout when the book starts on the right, while the bottom row shows a rollout when the book starts on the left. Here, we see that our method learns a policy to insert the book in different slots in the bookshelf depending on where the book is at the start of a trajectory. The robot usually prefers to put the book in the nearest slot, since this maximizes the reward that it can obtain from the classifier. On the other hand, if we were using goal images to specify the task, the robot would always place the book in one of the two slots, regardless of the starting position of the book.

Visual Draping This task requires draping a cloth over a box – essentially a miniaturized version of a tablecloth draping task. This task is depicted in Figures 3.5 and 3.6. The robot starts with holding the cloth in its gripper over the box. In order to succeed, it must drape the cloth smoothly, without crumpling it and without creating any wrinkles. In order to demonstrate the challenges associated with this task, we ran a baseline that only used the robot's end effector position as observation and a hand-defined a reward function on this observation (Euclidean distance to goal). We observed that this baseline failed to achieve the objective of this task, as it simply moved the end effector in a straight line motion to the goal, while this task cannot be solved using any straight line trajectory. See Figure 3.6 for more details.

Visual Bookshelf In this task, the goal is to insert a book into an empty slot on a bookshelf. The task is depicted in Figures 3.5 and 3.7. The initial position of the arm holding the book is randomized, requiring the robot to succeed from any starting position. Crucially, the bookshelf has several open slots, which means that, from different starting positions,

different slots may be preferred. We chose this task to emphasize that a goal classifier is fundamentally different from a goal state: there is not a single "goal image" that represents success at the task, but rather a condition on the position of the book that can be fulfilled in different ways (see Figure 3.7). The successful outcome examples correspondingly illustrate successful placements in both positions.

We provide 80 success examples each for the pushing, draping and book placing tasks. We query once every 250 timesteps for the pushing and book placing task, while querying once every 500 timesteps for the draping task. The *Visual Pusher* experiments are run for 6.2K timesteps (about 90 minutes of real world time), the *Visual Draping* experiments are run for 25K timesteps (about 4 hours), and the *Visual Bookshelf* experiment is run for 19K timesteps (about 3 hours). We therefore make 25 queries for the pushing experiment, 50 queries for the draping experiment, and 75 queries for the book placing experiment. Note that all of these tasks are learned directly from raw images, making these training times very efficient as compared to prior methods, including methods that use ground truth rewards [72, 50, 96, 129]. We hypothesize that the regularized classifiers learned by our method provide favorable reward shaping that makes image-based RL not only more practical, by not requiring engineered rewards, but also substantially more efficient.

The results of our experiments are provided in Figure 3.1, and videos of the tasks are provided on the project website[3]. All the tasks are evaluated in terms of success rates.

	VICE-RAQ (ours)	RAQ (ours)	VICE (ours)	Naïve Classifier
visual pushing	100%	60%	0%	0%
visual draping	100%	100%	100%	0%
visual bookshelf	100%	0%	60%	0%

Table 3.1: Results on the real world robot experiments. For all tasks, the reported numbers are success rates, indicating whether or not the object was successfully pushed to the goal, whether the cloth was successfully draped over the able, and whether the book was placed correctly on the shelf, averaged across 10 trials. In all cases, VICE-RAQ succeeds at learning the task, while VICE and RAQ succeed at some tasks while failing at others.

The *Visual Pushing* task requires the robot to interpret the RGB camera images and deal with variability in the mug placement. For this task, VICE-RAQ obtains a success rate of 100%, while RAQ only obtains a success rate of 60%. Both off-policy VICE and naïve classifier fail to solve this task. This indicates that including active queries in the classifier training process is helpful for obtaining good rewards, both with and without VICE. These experiments further show that VICE-RAQ can improve upon RAQ via leveraging the data collected by the policy during the reinforcement learning process.

For the *Visual Draping* task, we observe that all of our reward-learning methods (off-policy VICE, VICE-RAQ and RAQ) are able to solve the task, and only the naïve classifier baseline fails. We also compare to a baseline that is based on a reward defined on the robot state

[3] `https://sites.google.com/view/reward-learning-rl/home`

alone, in this case, the 3D position of the end-effector (the reward being the Euclidean distance to a goal end-effector position). We observe that this baseline fails to solve the task (see Figure 3.6). This indicates that performing this task requires the robot to actually pay attention, visually, to the deformations in the cloth, in order to perform the draping successfully. Manually designing reward functions for such deformable object manipulation tasks is generally extremely difficult, but all variants of our method are able to handle it successfully.

For the *Visual Bookshelf* task, where success corresponds to whether the book was successfully placed in an empty slot on the bookshelf or not), we see that RAQ alone is unable to solve this task, while off-policy VICE learns a policy that only succeeds sporadically. The policy learned with VICE-RAQ solves this task consistently, indicating that the combination of query labels and negative labels from all visited states from VICE provide improved classifier training.

3.9 Conclusion

In this chapter, we proposed an approach to reinforcement learning without hand-programmed reward functions. Our method, which we call VICE-RAQ, builds on the method introduced in the previous chapter and constructs a reward function from a modest number of user-provided examples of successful outcomes, which in practice might consist simply of pictures of the scene where the task has been successfully completed. Such examples are often substantially easier for a user to provide than either hand-programmed reward functions or full demonstrations. The initial reward is constructed out of a classifier trained on these examples and adversarially mined negatives. Beyond the initially provided success examples, our method uses a modest number of active queries, where the user is asked to label outcomes achieved by the robot as either successful or not. These additional queries are also simple to provide, and roughly correspond to the user directly reinforcing the robot's behavior. However, the user does not need to label all of the robot's experience – only about 50 queries are used in our experiments, out of tens of thousands of transitions.

By enabling robotic reinforcement learning without user-programmed reward functions or demonstrations, we believe that our approach represents a significant step towards making reinforcement learning a practical, automated, and readily usable tool for enabling versatile and capable robotic manipulation. By making it possible for robots to improve their skills directly in real-world environments, without any instrumentation or manual reward design, we believe that our method also represents a step toward enabling lifelong learning for robotic systems that learn directly "in the wild." This capability can make it feasible in the future for robots to acquire broad and highly generalizable skill repertoires directly through interaction with the real world.

Chapter 4

Parrot: Data-Driven Behavioral Priors for Reinforcement Learning

4.1 Introduction

Reinforcement Learning (RL) is an attractive paradigm for robotic learning because of its flexibility in being able to learn a diverse range of skills and its capacity to continuously improve. However, RL algorithms typically require a large amount of data to solve each individual task, including simple ones. Since an RL agent is generally initialized without any prior knowledge, it must try many largely unproductive behaviors before it discovers a high-reward outcome. In contrast, humans rarely attempt to solve new tasks in this way: they draw on their prior experience of what is *useful* when they attempt a new task, which substantially shrinks the task search space. For example, faced with a new task involving objects on a table, a person might grasp an object, stack multiple objects, or explore other object rearrangements, rather than re-learning how to move their arms and fingers.

Can we endow RL agents with a similar sort of *behavioral prior* from past experience? In other fields of machine learning, the use of large prior datasets to bootstrap acquisition of new capabilities has been studied extensively to good effect. For example, language models trained on large, diverse datasets offer representations that drastically improve the efficiency of learning downstream tasks [18]. What would be the analogue of this kind of pre-training in robotics and RL? One way we can approach this problem is to leverage successful trials from a wide range of previously seen tasks to improve learning for *new* tasks. The data could come from previously learned policies, from human demonstrations, or even unstructured teleoperation of robots [100]. In this paper, we show that behavioral priors can be obtained through *representation learning*, and the representation in question must not only be a representation of inputs, but actually a representation of input-output relationships – a space of possible and likely mappings from states to actions among which the learning process can interpolate when confronted with a new task.

What makes for a good representation for RL? Given a new task, a good representation

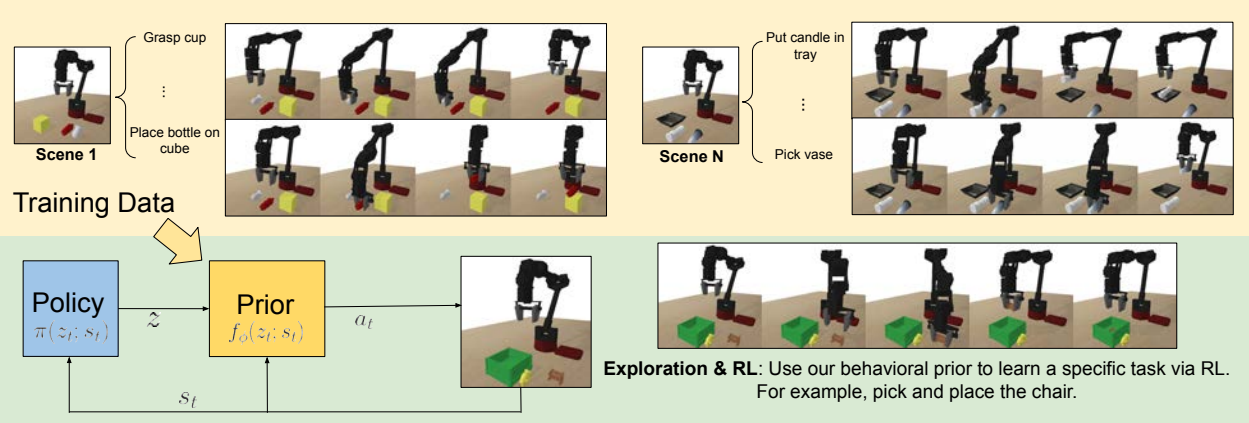


Figure 4.1: **Our problem setting**. Our training dataset consists of near-optimal state-action trajectories (without reward labels) from a wide range of tasks. Each task might involve interacting with a different set of objects. Even for the same set of objects, the task can be different depending on our objective. For example, in the upper right corner, the objective could be picking up a cup, or it could be to place the bottle on the yellow cube. We learn a behavioral prior from this multi-task dataset capable of trying many different useful behaviors when placed in a new environment, and can aid an RL agent to quickly learn a specific task in this new environment.

must (a) provide an effective exploration strategy, (b) simplify the policy learning problem for the RL algorithm, and (c) allow the RL agent to retain full control over the environment. In this paper, we address all of these challenges through learning an *invertible* function that maps noise vectors to complex, high-dimensional environment actions. Building on prior work in normalizing flows [20], we train this mapping to maximize the (conditional) log-likelihood of actions observed in successful trials from past tasks. When dropped into a new MDP, the RL agent can now sample from a unit Gaussian, and use the learned mapping (which we refer to as the *behavioral prior*) to generate likely environment actions, conditional on the current observation. This learned mapping essentially transforms the original MDP into a simpler one for the RL agent, as long as the original MDP shares (partial) structure with previously seen MDPs (see Section 4.3). Furthermore, since this mapping is invertible, the RL agent still retains full control over the original MDP: for every possible environment action, there exists a point within the support of the Gaussian distribution that maps to that action. This allows the RL agent to still try out new behaviors that are distinct from what was previously observed.

Our main contribution is a framework for pre-training in RL from a diverse multi-task dataset, which produces a behavioral prior that accelerates acquisition of new skills. We present an instantiation of this framework in robotic manipulation, where we utilize manipulation data from a diverse range of prior tasks to train our behavioral prior, and then use it to bootstrap exploration for new tasks. By making it possible to pre-train action representations on large prior datasets for robotics and RL, we hope that our method provides

a path toward leveraging large datasets in the RL and robotics settings, much like language models can leverage large text corpora in NLP and unsupervised pre-training can leverage large image datasets in computer vision. Our method, which we call Prior AcceleRated ReinfOrcemenT (or PARROT), is able to quickly learn tasks that involve manipulating previously unseen objects, from image observations and sparse rewards, in settings where RL from scratch fails to learn a policy at all. We also compare against prior works that incorporate prior data for RL, and show that PARROT substantially outperforms these prior works.

4.2 Relation to Prior Work

Generative modeling and RL. There is another line of work [44, 53, 43] that explores using generative models for RL, using a variational autoencoder [78] to model entire trajectories in an observation-independent manner, and then learning an open-loop, single-step policy using RL to solve the downstream task. Our approach differs in several key aspects: (1) our model is observation-conditioned, allowing it to prioritize actions that are relevant to the current scene or environment, (2) our model allows for closed-loop feedback control, and (3) our model is *invertible*, allowing the high-level policy to retain full control over the action space. Our experiments demonstrate these aspects are crucial for solving harder tasks.

Hierarchical learning. Our method can be interpreted as training a hierarchical model: the low-level policy is the behavioral prior trained on prior data, while the high-level policy is trained using RL and controls the low-level policy. This structure is similar to prior work in hierarchical RL [17, 122, 19, 152, 84]. We divide prior work in hierarchical learning into two categories: methods that seek to learn both the low-level and high-level policies through active interaction with an environment [90, 55, 6, 36, 48, 111, 9, 125], and methods that learn temporally extended actions, also known as *options*, from demonstrations, and then recompose them to perform long-horizon tasks through RL or planning [37, 83, 79, 146, 145]. Our work shares similarities with the data-driven approach of the latter methods, but work on options focuses on modeling the *temporal* structure in demonstrations for a small number of long-horizon tasks, while our behavioral prior is not concerned with temporally-extended abstractions, but rather with transforming the original MDP into one where potentially useful behaviors are more likely, and useless behaviors are less likely.

Meta-learning. Our goal in this paper is to utilize data from previously seen tasks to speed up RL for new tasks. Meta-RL [23, 161, 31, 107, 132, 106, 177, 28] and meta-imitation methods [22, 35, 64, 68, 121, 169, 63, 172] also seek to speed up learning for new tasks by leveraging experience from previously seen tasks. While meta-learning provides an appealing and principled framework to accelerate acquisition of future tasks, we focus on a more lightweight approach with relaxed assumptions that make our method more practically applicable, and we discuss these assumptions in detail in the next section.

4.3 Problem Setup

Our goal is to improve an agent's ability to learn new tasks by incorporating a behavioral prior, which it can acquire from previously seen tasks. Each task can be considered a Markov decision process (MDP), which is defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathrm{T}, r, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ represent state and action spaces, $\mathrm{T}(s'|s,a)$ and $r(s,a)$ represent the dynamics and reward functions, and $\gamma \in (0,1)$ represents the discount factor. Let $p(M)$ denote a distribution over such MDPs, with the constraint that the state and action spaces are fixed. In our experiments, we treat high-dimensional images as s, which means that this constraint is not very restrictive in practice. In order for the behavioral prior to be able to accelerate the acquisition of new skills, we assume the behavioral prior is trained on data that structurally resembles potential optimal policies for all or part of the new task. For example, if the new task requires placing a bottle in a tray, the prior data might include some behaviors that involve picking up objects. There are many ways to formalize this assumption. One way to state this formally is to assume that prior data consists of executions of near-optimal policies [] for MDPs drawn according to $M \sim p(M)$, and the new task $M^\star$ is likewise drawn from $p(M)$. In this case, the generative process for the prior data can be expressed as:

$$M \sim p(M), \quad \pi_M(\tau) = \arg\max_\pi \mathbb{E}_{\pi,M}[R_M], \quad \tau_M \sim \pi_M(\tau), \tag{4.1}$$

where $\tau_M = (s_1, a_1, s_2, a_2, \ldots, s_T, a_T)$ is a sequence of state and actions, $\pi_M(\tau)$ denotes a near-optimal policy [76] for MDP M and $R_M = \sum_{t=0}^{\infty} \gamma^t r_t$.

When incorporating the behavioral prior for learning a new task $M^\star$, our goal is the same as standard RL: to find a policy π that maximizes the expected return $\arg\max_\pi \mathbb{E}_{\pi,M^\star}[R_{M^\star}]$. Our assumption on tasks being drawn from a distribution $p(M)$ shares similarities with the meta-RL problem [161, 23], but our setup is different: it does not require accessing any task in $p(M)$ except the new task we are learning, $M^\star$. Meta-RL methods need to interact with the tasks in $p(M)$ during meta-training, with access to rewards and additional samples, whereas we learn our behavioral prior simply from data, without even requiring this data to be labeled with rewards. This is of particular importance for real-world problem settings such as robotics: it is much easier to store data from prior tasks (e.g., different environments) than to have a robot physically revisit those prior settings and retry those tasks, and not requiring known rewards makes it possible to use data from a variety of sources, including human-provided demonstrations. In our setting, RL is performed in only one environment, while the prior data can come from many environments.

Our setting is related to meta-imitation learning [22, 35], as we speed up learning new tasks using data collected from past tasks. However, meta-imitation learning methods require at least one demonstration for each new task, whereas our method can learn new tasks without any demonstrations. Further, our data requirements are less stringent: meta-imitation learning methods require all demonstrations to be optimal, require all trajectories in the dataset to have a task label, and requires "paired demonstrations", i.e. at least two demonstrations for each task (since meta-imitation methods maximize the likelihood of

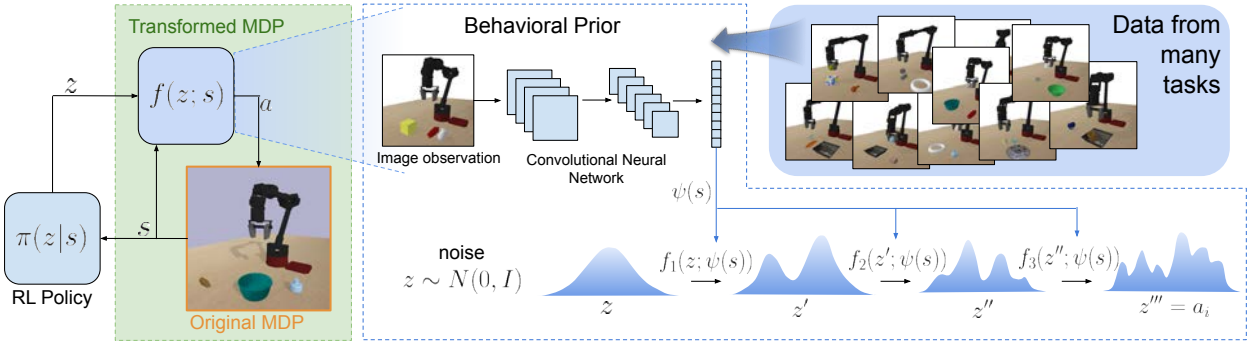


Figure 4.2: **PARROT**. Using successful trials from a large variety of tasks, we learn an invertible mapping f_ϕ that maps noise z to useful actions a. This mapping is conditioned on the current observation, which in our case is an RGB image. The image is passed through a stack of convolutional layers and flattened to obtain an image encoding $\psi(s)$, and this image encoding is then used to condition each individual transformation f_i of our overall mapping function f_ϕ. The parameters of the mapping (including the convolutional encoder) are learned through maximizing the conditional log-likelihood of state-action pairs observed in the dataset. When learning a new task, this mapping can simplify the MDP for an RL agent by mapping actions sampled from a randomly initialized policy to actions that are likely to lead to useful behavior in the current scene. Since the mapping is invertible, the RL agent still retains full control over the action space of the original MDP, simply the likelihood of executing a useful action is increased through use of the pre-trained mapping.

actions from one demonstration after conditioning the policy on another demonstration from the same task). Relaxing these requirements increases the scalability of our method: we can incorporate data from a wider range of sources, and we do not need to explicitly organize it into specific tasks.

4.4 Behavioral Priors For Reinforcement Learning

Our method learns a behavioral prior for downstream RL by utilizing a dataset $\mathcal{D}$ of (near-optimal) state-action pairs from previously seen tasks. We do so by learning a state-conditioned mapping $f_\phi: \mathcal{Z} \times \mathcal{S} \rightarrow \mathcal{A}$ (where ϕ denotes learnable parameters) that transforms a noise vector z into an action a that is likely to be useful in the current state s. This removes the need for exploring via "meaningless" random behavior, and instead enables an exploration process where the agent attempts behaviors that have been shown to be useful in previously seen domains. For example, if a robotic arm is placed in front of several objects, randomly sampling z (from a simple distribution, such as the unit Gaussian) and applying the mapping $a = f_\phi(z; s)$ should result in actions that, when executed, result in meaningful interactions with the objects. This learned mapping essentially transforms the MDP experienced by the RL agent into a simpler one, where every random action executed in this transformed MDP is much more likely to lead to a useful behavior.

How can we learn such a mapping? In this paper, we propose to learn this mapping

through state-conditioned generative modeling of the actions observed in the original dataset $\mathcal{D}$, and we refer to this state-conditioned distribution over actions as the behavioral prior $p_{\text{prior}}(a|s)$. A deep generative model takes noise as input, and outputs a plausible sample from the target distribution, i.e. it can represent $p_{\text{prior}}(a|s)$ as a distribution over noise z using a deterministic mapping $f_\phi : \mathcal{Z} \times \mathcal{S} \mapsto A$ When learning a new task, we can use this mapping to reparametrize the action space of the RL agent: if the action chosen by the randomly initialized neural network policy is z, then we execute the action $a = f_\phi(z; s)$ in the original MDP, and learn a policy $\pi(z|s)$ that maximizes the task reward through learning to control the inputs to the mapping f_ϕ. The training of the behavioral prior and the task-specific policy is decoupled, allowing us to mix and match RL algorithms and generative models to best suit the application of interest. An overview of our overall architecture is depicted in Figure 4.2. In the next subsection, we discuss what properties we would like the behavioral prior to satisfy, and present one particular choice for learning a prior that satisfies all of these properties.

Learning a Behavioral Prior With Normalizing Flows

For the behavioral prior to be effective, it needs to satisfy certain properties. Since we learn the prior from a *multi-task* dataset, containing several different behaviors even for the same initial state, the learned prior should be capable of representing complex, multi-modal distributions. Second, it should provide a mapping for generating "useful" actions from noise samples when learning a new task. Third, the prior should be state-conditioned, so that only actions that are relevant to the current state are sampled. And finally, the learned mapping should allow easier learning in the reparameterized action space *without* hindering the RL agent's ability to attempt novel behaviors, including actions that might not have been observed in the dataset $\mathcal{D}$. Generative models based on normalizing flows [20] satisfy all of these properties well: they allow maximizing the model's exact log-likelihood of observed examples, and learn a deterministic, invertible mapping that transforms samples from a simple distribution p_z to examples observed in the training dataset. In particular, the real-valued non-volume preserving (real NVP) architecture introduced by Dinh et al [20] allows using deep neural networks to parameterize this mapping (making it expressive) While the original real NVP work modelled unconditional distributions, follow-up work has found that it can be easily extended to incorporate conditioning information [4]. We refer the reader to prior work [20] for a complete description of real NVPs, and summarize its key features here. Given an invertible mapping $a = f_\phi(z; s)$, the change of variable formula allows expressing the likelihood of the observed actions using samples from $\mathcal{D}$ in the following way:

$$p_{\text{prior}}(a|s) = p_z\big(f_\phi^{-1}(a; s)\big) \left|\det\left(\partial f_\phi^{-1}(a;s) / \partial a\right)\right| \tag{4.2}$$

Dinh et al [20] propose a particular (unconditioned) form of the invertible mapping f_ϕ, called an affine coupling layer, that maintains tractability of the likelihood term above, while still allowing the mapping f_ϕ to be expressive. Several coupling layers can be composed together

to transform simple noise vectors into samples from complex distributions, and each layer can be conditioned on other variables, as shown in Figure 4.2.

Accelerated Reinforcement Learning via Behavioral Priors

After we obtain the mapping $f_\phi(z; s)$ from the behavioral prior learned by maximizing the likelihood term in Equation 4.2, we would like to use it to accelerate RL when solving a new task. Instead of learning a policy $\pi_\theta(a|s)$ that directly executes its actions in the original MDP, we learn a policy $\pi_\theta(z|s)$, and execute an action in the environment according to $a = f_\phi(z; s)$. As shown in Figure 4.2, this essentially transforms the MDP experienced for the RL agent into one where random actions $z \sim p_z$ (where p_z is the base distribution used for training the mapping f_ϕ) are much more likely to result in useful behaviors. To enable effective exploration at the start of the learning period, we initialize the RL policy to the base distribution used for training the prior, so that at the beginning of training, $\pi_\theta(z|s) := p_z(z)$. Since the mapping f_ϕ is invertible, the RL agent still retains full control over the action space: for any given a, it can always find a z that generates $z = f_\phi^{-1}(a; s)$ in the original MDP. The learned mapping increases the likelihood of useful actions without crippling the RL agent, making it ideal for fine-tuning from task-specific data. Our complete method is described in Algorithm 4 in Appendix C.1. Note that we need to learn the mapping f_ϕ only once, and it can be used for accelerated learning of any new task.

Implementation Details

We use real NVP to learn f_ϕ, and as shown in Figure 4.2, each coupling layer in the real NVP takes as input the the output of the previous coupling layer, and the conditioning information. The conditioning information in our case corresponds to RGB image observations, which allows us to train a single behavioral prior across a wide variety of tasks, even when the tasks might have different underlying states (for example, different objects). We train a real NVP model with four coupling layers; the exact architecture, and other hyperparameters, are detailed in Appendix C.2. The behavioral prior can be combined with any RL algorithm that is suitable for continuous action spaces, and we chose to use the soft actor-critic [50] due to its stability and ease of use.

4.5 Experiments

Our experiments seek to answer: **(1)** Can the behavioral prior accelerate learning of *new* tasks? **(2)** How does PARROT compare to prior works that accelerate RL with demonstrations? **(3)** How does PARROT compare to prior methods that combine hierarchical imitation with RL?

Domains. We evaluate our method on a suite of challenging robotic manipulation tasks, a subset of which are depicted in Figure 4.3. Each task involves controlling a 6-DoF robotic arm and its gripper, with a 7D action space. The observation is a 48×48 RGB image,

which allows us to use the same observation representation across tasks, even though each underlying task might have a different underlying state (e.g., different objects). No other observations (such as joint angles or end-effector positions) are provided. In each task, the robot needs to interact with one or two objects in the scene to achieve its objective, and there are three objects in each scene. Note that all of the objects in the test scenes are *novel* – the dataset $\mathcal{D}$ contains no interactions with these objects. The object positions at the start of each trial are randomized, and the policy must infer these positions from image observations in order to successfully solve the task. A reward of +1 is provided when the objective for the task is achieved, and the reward is zero otherwise. Detailed information on the objective for each task and example rollouts are provided in Appendix C.3, and on our anonymous project website[1].

Data collection. Our behavioral prior is trained on a diverse dataset of trajectories from a wide range of tasks, and then utilized to accelerate reinforcement learning of new tasks. As discussed in Section 4.3, for the prior to be effective, it needs to be trained on a dataset that structurally resembles the behaviors that might be optimal for the new tasks. In our case, all of the behaviors involve repositioning objects (i.e., picking up objects and moving them to new locations), which represents a very general class of tasks that might be performed by a robotic arm. While the dataset can be collected in many ways, such as from human demonstrations or prior tasks solved by the robot, we collect it using a set of randomized scripted policies, see Appendix C.3 for details. Since the policies are randomized, not every execution of such a policy results in a useful behavior, and we decide to keep or discard a collected trajectory based on a simple predefined rule: if the trajectory collected ends with a successful grasp or rearrangement of any one of the objects in the scene, we add this trajectory to our dataset. We collect a dataset of 50K trajectories, where each trajectory is of length 25 timesteps (≈5-6 seconds), for a total of 1.25m observation-action pairs. The observation is a 48×48 RGB image, while the actions are continuous 7D vectors. Data collection involves interactions with over 50 everyday objects (see Appendix C.3); the diversity of this dataset

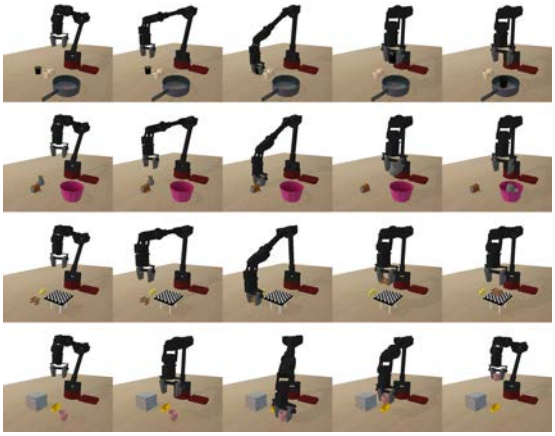


Figure 4.3: **Tasks.** A subset of our evaluation tasks, with one task shown in each row. In the first task (first row), the objective is to pick up a can and place it in the pan. In the second task, the robot must pick up the vase and put it in the basket. In the third task, the goal is to place the chair on top of the checkerboard. In the fourth task, the robot must pick up the mug and hold it above a certain height. Initial positions of all objects are randomized, and must be inferred from visual observations. Not all objects in the scene are relevant to the current task.

[1] `https://sites.google.com/view/parrot-rl`

enables learning priors that can produce useful behavior when interacting with a new object.

Results, Comparisons and Analysis

To answer the questions posed at the start of this section, we compare PARROT against a number of prior works, as well as ablations of our method. Additional implementation details and hyperparameters can be found in Appendix C.2.

Soft-Actor Critic (SAC). For a basic RL comparison, we compare against the vanilla soft-actor critic algorithm [50], which does not incorporate any previously collected data.

SAC with demonstrations (BC-SAC). We compare against a method that incorporates demonstrations to speed up learning of new tasks. In particular, we initialize the SAC policy by performing behavioral cloning on the entire dataset $\mathcal{D}$, and then fine-tune it using SAC. This approach is similar to what has been used in prior work [130], except we use SAC as our RL algorithm. While we did test other methods that are designed to use demonstration data with RL such as DDPGfD [160] and AWAC [112], we found that our simple BC + SAC variant performed better. This somewhat contradicts the results reported in prior work [112], but we believe that this is because the prior data is not labeled with rewards (all transitions are assigned a reward of 0), and more powerful demonstration + RL methods require access to these rewards, and subsequently struggle due to the reward misspecification.

Transfer Learning via Feature Learning (VAE-features). We compare against prior methods for transfer learning (in RL) that involve learning a robust representation of the input observation. Similar to Higgins et al [58], we train a β-VAE using the observations in our training set, and train a policy on top of the features learned by this VAE when learning downstream tasks.

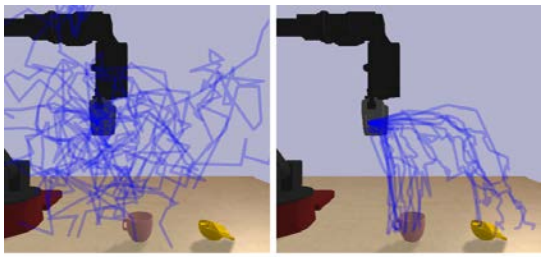


Figure 4.4: We plot trajectories from executing a random policy, with and without the behavioral prior. We see that the behavioral prior substantially increases the likelihood of executing an action that is likely to lead to a meaningful interaction with an object, while still exploring a diverse set of actions.

Trajectory modeling and RL (TrajRL). Ghadirzadeh et al [43] model entire trajectories using a VAE, and learn a one-step policy on top of the VAE to solve tasks using RL. Our implementation of this method uses a VAE architecture identical to the original paper's, and we then train a policy using SAC to solve new tasks with the action space induced by the VAE. We performed additional hyperparameter tuning for this comparison, the details of which can be found in Appendix C.2.

Hierarchical imitation and RL (HIRL). Prior works in hierarchical imitation learning [37, 145] train latent variable models over expert demonstrations to discover options, and later utilize these options to learn long-horizon tasks using RL. While PARROT can also be extended to model the temporal structure in trajectories through conditioning on past states

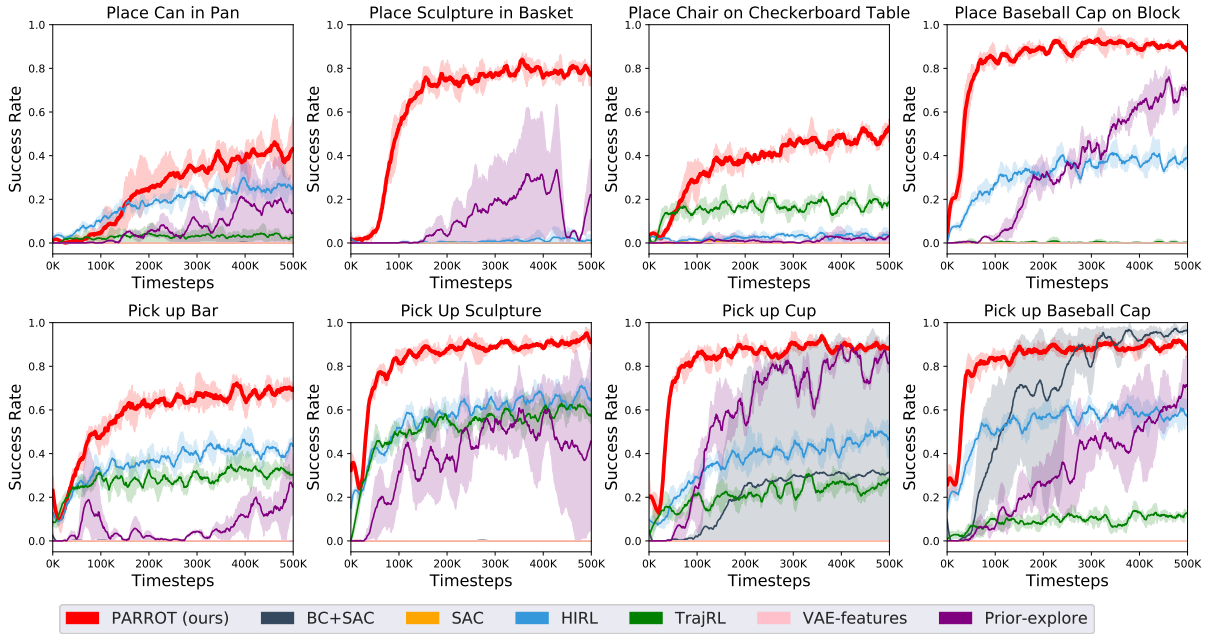


Figure 4.5: **Results.** The lines represent average performance across multiple random seeds, and the shaded areas represent the standard deviation. PARROT is able to learn much faster than prior methods on a majority of the tasks, and shows little variance across runs (all experiments were run with three random seeds, computational constraints of image-based RL make it difficult to run more seeds). Note that some methods that failed to make any progress on certain tasks (such as "Place Sculpture in Basket") overlap each other with a success rate of zero. SAC and VAE-features fail to make progress on any of the tasks.

and actions, by modeling $p_{\text{prior}}(a_t, |s_t, s_{t-1}, ..., a_{t-1}, ..., a_0)$ instead of $p_{\text{prior}}(a_t|s_t)$, we focus on a simpler version of the model in this paper that does not condition on the past. In order to provide a fair comparison, we modify the model proposed by Shankar et al [145] to remove the past conditioning, which then reduces to training a conditional VAE, and performing RL on the action space induced by the latent space of this VAE. This comparison is similar to our proposed approach, but with one crucial difference: the mapping we learn is invertible, and allows the RL agent to retain full control over the final actions in the environment (since for every $a \in \mathcal{A}$, there exist some $z = f_\phi^{-1}(a; s)$, while a latent space learned by a VAE provides no such guarantee).

Exploration via behavioral prior (Prior-explore). We also run experiments with an ablation of our method: instead of using a behavioral prior to transform the MDP being experienced by the RL agent, we use it to simply aid the exploration process. While collecting data, an action is executed from the prior with probability ϵ, else an action is executed from the learned policy. We experimented with $\epsilon = 0.1, 0.3, 0.7, 0.9$, and found 0.9 to perform best.

Main results. Our results are summarised in Figure 4.5. We see that PARROT is able

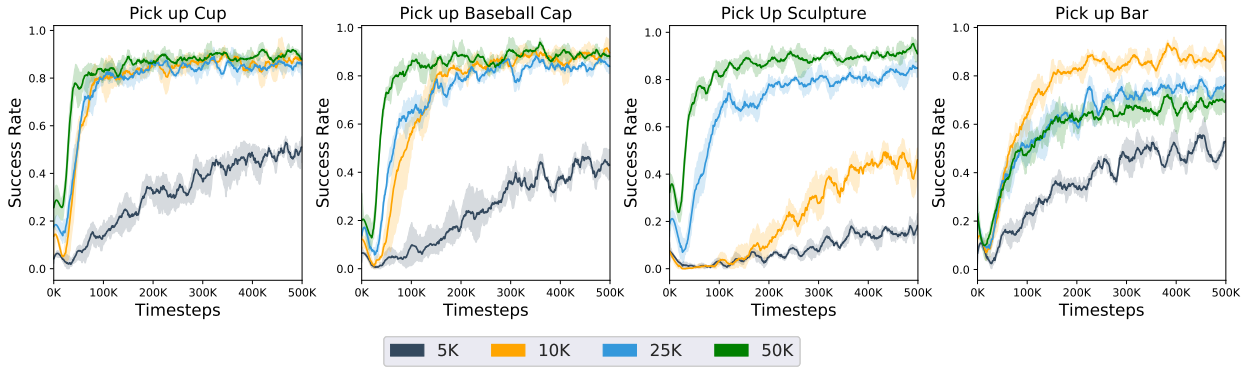


Figure 4.6: **Impact of dataset size on performance.** We observe that training on 10K, 25K or 50K trajectories yields similar performance.

to solve all of the tasks substantially faster and achieve substantially higher final returns than other methods. The SAC baseline (which does not use any prior data) fails to make progress on any of the tasks, which we suspect is due to the challenge of exploring in sparse reward settings with a randomly initialized policy. Figure 4.4 illustrates a comparison between using a behavioral prior and a random policy for exploration. The VAE-features baseline similarly fails to make any progress, and due to the same reason: the difficulty of exploration in a sparse reward setting. Initializing the SAC policy with behavior cloning allows it to make progress on only two of the tasks, which is not surprising: a Gaussian policy learned through a behavior cloning loss is not expressive enough to represent the complex, multi-modal action distributions observed in dataset $\mathcal{D}$. Both **TrajRL** and **HIRL** perform much better than any of the other baselines, but their performance plateaus a lot earlier than PARROT. While the initial exploration performance of our learned behavioral prior is not substantially better from these methods (denoted by the initial success rate in the learning curves), the flexibility of the representation it offers (through learning an invertible mapping) allows the RL agent to improve far beyond its initial performance. **Prior-explore**, an ablation of our method, is able to make progress on most tasks, but is unable to learn as fast as our method, and also demonstrates unstable learning on some of the tasks. We suspect this is due to the following reason: while off-policy RL methods like SAC aim to learn from data collected by any policy, they are in practice quite sensitive to the data distribution, and can run into issues if the data collection policy differs substantially from the policy being learned [87].

Impact of dataset size on performance. We conducted additional experiments on a subset of our tasks to evaluate how final performance is impacted as a result of dataset size, results from which are shown in Figure 4.6. As one might expect, the size of the dataset positively correlates with performance, but about 10K trajectories are sufficient for obtaining good performance, and collecting additional data yields diminishing returns. Note that initializing with even a smaller dataset size (like 5K trajectories) yields much better

performance than learning from scratch.

Mismatch between train and test tasks. We ran experiments in which we deliberately bias the training dataset so that the training tasks and test tasks are functionally different (i.e. involve substantially different actions), the results from which are shown in Figure 4.7. We observe that if the prior is trained on pick and place tasks alone, it can still solve downstream grasping tasks well. However, if the prior is trained only on grasping, it is unable to perform well when solving pick and place tasks. We suspect this is due to the fact that pick and place tasks involve a completely new action (that of opening the gripper), which is never observed by the prior if it is trained only on grasping, making it difficult to learn this behavior from scratch for downstream tasks.

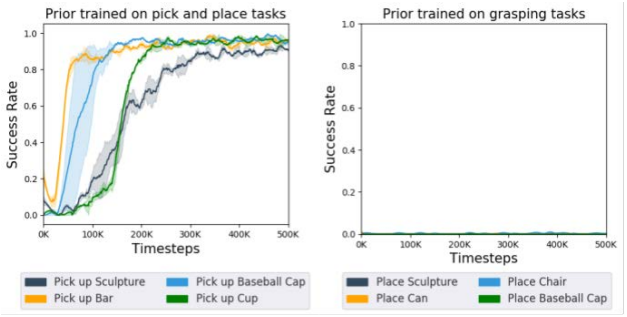


Figure 4.7: **Impact of train/test mismatch on performance.** Each plot shows results for four tasks. Note that for the pick and place tasks, the performance is close to zero, and the curves mostly overlap each other on the x-axis.

4.6 Conclusion

We presented PARROT, a method for learning behavioral priors using successful trials from a wide range of tasks. Learning from priors accelerates RL on new tasks–including manipulating previously unseen objects from high-dimensional image observations–which RL from scratch often fails to learn. Our method also compares favorably to other prior works that use prior data to bootstrap learning for new tasks. While our method learns faster and performs better than prior work in learning novel tasks, it still requires thousands of trials to attain high success rates. Improving this efficiency even further, perhaps inspired by ideas in meta-learning, could be a promising direction for future work. Our work opens the possibility for several exciting future directions. PARROT provides a mapping for executing actions in new environments structurally similar to those of prior tasks. While we primarily utilized this mapping to accelerate learning of new tasks, future work could investigate how it can also enable safe exploration of new environments [65, 136]. While the invertibility of our learned mapping ensures that it is theoretically possible for the RL policy to execute any action in the original MDP, the probability of executing an action can become very low if this action was never seen in the training set. This can be an issue if there is a significant mismatch between the training dataset and the downstream task (as shown in our experiments), and tackling this issue would make for an interesting problem.

Chapter 5

COG: Connecting New Skills to Past Experience with Offline Reinforcement Learning

5.1 Introduction

Consider a robot that has been trained using reinforcement learning (RL) to take an object out of an open drawer. It learns to grasp the object and pull it out of the drawer. If the robot is then placed in a scene where the drawer is instead closed, it will likely fail to take the object out, since it has not seen this scenario or initial condition before. How can we enable learning-based robotic systems to reason effectively about such scenarios? Conventionally, we might expect that complex methods based on hierarchies or explicit skill decomposition would be needed to integrate a drawer opening skill with the grasping behavior. But what if simply combining previously collected (and *unlabeled*) robot interaction data, which might include drawer opening and other behaviors, together with offline RL methods [97], can allow these behaviors to be combined *automatically*, without any explicit separation into individual skills? In this paper, we study how model-free RL algorithms can utilize prior data to extend and generalize learned behaviors, incorporating segments of experience from this prior data as needed at test-time.

Standard online RL algorithms typically require a tight interaction loop between data collection and policy learning. There has been a significant amount of recent interest in devising methods for offline RL [97, 42, 86, 88, 2, 69], which can leverage previously collected datasets without environment interaction. In this paper, we build on recent advances in offline RL, and show that a "data-driven" RL paradigm allows us to build real-world robotic learning systems that can learn increasingly flexible and general skills. We will show that collecting a large dataset that covers a large repertoire of skills provides a useful foundation for learning new tasks, enabling learned policies to perform multi-stage tasks from previously unseen conditions, despite never having seen all of the stages together in a single episode.

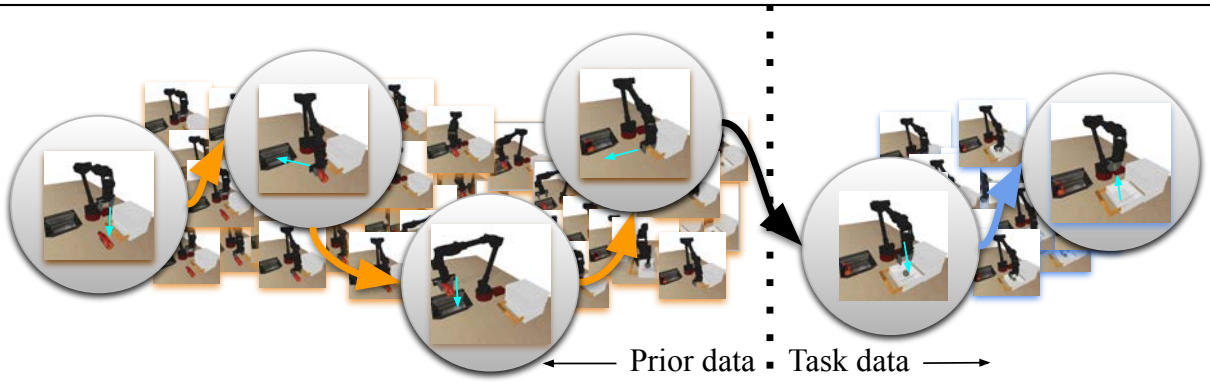


Figure 5.1: **Incorporating unlabeled prior data into the process of learning a new skill.** We present a system that allows us to extend and generalize robotic skills by using unlabeled prior datasets. Learning a new skill requires collecting some task-specific data (right), which may not contain all the necessary behaviors needed to set up the initial conditions for this skill in a new setting (e.g., opening a drawer before taking something out of it). The prior data (left) can be used by the robot to automatically figure out that, when it encounters a closed drawer at test time, it can first open it, and then remove the object. The task data does not contain drawer opening, and the prior data does not contain any examples of lifting the new object.

Consider the drawer example from before: if the robot can draw on its prior experience of drawer opening, it can take an object out of the drawer even if the drawer is initially closed, by figuring out that opening the drawer will place it into a state from which the grasping skill can succeed. Furthermore, even if there is an obstruction in front of the drawer, the robot can once again draw on the prior data, which might also include pick and place behaviors, to remove the obstruction. Crucially, this does not require the robot to practice drawer opening again when learning a new task – it can collect data for the new task in isolation, and combine it with past data to compose novel behaviors. We illustrate this idea in Figure 5.1.

How can this prior data from different tasks be utilized for learning a new task? If the prior data contains successful behavior for the new task, this would be easy. However, we will show that even prior data that is unsuccessful at the new task can be useful for solving it, since it provides the policy with a deeper understanding of the mechanics of the world. In contrast to prior works that approach this problem from a hierarchical skill learning [29, 114] or planning perspective [24, 27], we show that this "understanding" can be acquired without any explicit predictive model or planning, simply via model-free RL. Our method builds on a recently proposed model-free offline RL method called conservative Q-learning (CQL) [86]. We extend CQL to our problem setting as following: we initialize the replay buffer with our prior dataset, and assign zero reward to every transition in this dataset, since the new task is different from the prior executed behavior. We then run CQL on both this prior dataset and data collected specifically for the new task, and evaluate the policy on a variety of initial conditions which were not seen in the task-specific data. We can further fine-tune the policy resulting from offline training with limited online data collection, which further improves the

performance of the new skill while retaining the aforementioned generalization benefits.

Our main contribution is to demonstrate that model-free offline RL can learn to combine task data with prior data, producing previously unseen combinations of skills to meet the pre-conditions of the task of interest, and effectively generalizing to new initial conditions. We call our approach **COG**: Connecting skills via Offline RL for Generalization. We describe a robotic learning system that builds on this idea to learn a variety of manipulation behaviors, directly from images. We evaluate our system on several manipulation tasks, where we collect prior datasets consisting of imperfect scripted behaviors, such as grasping, pulling, pushing, and placing a variety of objects. We use this prior data to learn several downstream skills: opening and closing drawers, taking objects out of drawers, putting objects in a tray, and so on. We train neural network-based policies on raw, high-dimensional image observations, and only use sparse binary rewards as supervision. We demonstrate the effectiveness of COG in both simulated domains and on a real world low-cost robotic arm.

5.2 Relation To Prior Work

Robotic RL. RL has been applied to a wide variety of robotic manipulation tasks, including grasping objects [72, 170], in-hand object manipulation [120, 61, 131, 89], pouring fluids [141], door opening [167, 11], and manipulating cloth [105, 149]. Most of these works use online RL methods, relying on a well-tuned interaction loop between data collection and policy training, instead of leveraging prior datasets. Our paper is more closely related to Kalashnikov et al [72], Julian et al [71], and Cabi et al [8], which also use offline RL and large prior datasets. However, these prior works focus on generalization to new objects, as well as fine-tuning to handle greater variability (e.g., more object types, changes in lighting, etc.). Our work instead focuses on changes in initial conditions that require entirely different skills than those learned as part of the current task, such as opening a drawer before grasping an object.

Data-driven robotic learning. In addition to RL-based robotics, data-driven robotics in general has become increasingly popular in recent years, and several works have investigated using large-scale datasets to tackle long-standing challenges in robotics, such as grasping novel objects. However, most prior work in this category focuses on executing the same actions on novel objects [75, 129, 96, 101, 46]. In contrast, we explicitly target problems where new behavior needs to be learned to perform the task in a new scenario, and use prior interaction datasets to achieve this ability via model-free RL. Visual foresight [32] and its followups [25, 24, 165, 62] also address temporally extended tasks with large datasets by learning video prediction models, but with significantly shorter time horizons than demonstrated in our work, due to the difficulty of long-horizon video prediction. Mandlekar et al [103] use an alternate approach of explicitly learning hierarchical policies for control, and Mandlekar et al [104] utilizes offline imitation learning to compose different demonstration trajectories together.

Offline deep RL. While offline (or "batch") RL is a well-studied area [92, 91, 109, 97], there has been a significant amount of recent interest in offline deep RL, where deep RL agents

are trained on a static, previously collected dataset without any environment interaction [97, 86, 42, 88, 2, 162, 123, 168]. These works largely focuses on developing more effective offline deep RL algorithms by correcting for distribution shift [40, 86, 88, 162, 123, 168] which renders standard off-policy RL algorithms inadmissible in purely offline settings [88, 97]. In contrast, our work does not propose a new algorithm, but rather adapts existing offline RL methods to the setting where prior data from a *different* domain must be integrated into learning a new task such that it succeeds under a variety of conditions.

5.3 Incorporating Prior Data into Robotic Reinforcement Learning

Our goal is to develop a system that can learn new robotic skills with RL, while incorporating diverse previously collected data to provide greater generalization. We hypothesize that, even if a *subset of* this prior data illustrates useful behaviors that are *in support* of the new skill, including this data into the training process via offline RL can endow the robot with the ability to reason, such that when the conditions at test-time do not match those seen in the data for the specific new skill, the robot might still generalize because other transitions in the prior data have allowed it to learn how to react intelligently. For instance, when a robot trained to perform object grasping is faced with a closed or obstructed drawer at test time, without any prior experience, the robot would have no way to know how to react. Humans generally handle such situations gracefully, by drawing on their past experience. We use the term "common sense" to refer to this ability to handle small test-time variations in the task, such as the need to open a drawer before taking something out of it. How can we endow learned policies with this sort of "common sense"? If the prior training data illustrates opening of drawers and removing obstructions, then even without being told that such behaviors are useful for grasping an object from an obstructed drawer, model-free offline RL methods can learn to reason and utilize such behaviors at test time. Crucially, we will show how these "common sense" behaviors emerge entirely from combining rich prior data and offline RL, without any explicit reasoning about preconditions or hierarchical higher-level planning.

We formalize our problem in the standard RL framework. The goal in RL is to optimize the infinite horizon discounted return $R_t = \sum_{t=0}^{\infty} \gamma^t r_t$ in a Markov decision process (MDP), which is defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathrm{T}, r, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ represent state and action spaces, $\mathrm{T}(s'|s,a)$ and $r(s,a)$ represent the dynamics and reward function, and $\gamma \in (0,1)$ represents the discount factor. We operate in the *offline* RL setting as opposed to the standard online regime since we are interested in leveraging most out of prior datasets.

In most prior works on offline RL [88, 42, 2], the method is typically provided with data for the specific task that we wish to train a policy for, and the entire dataset is annotated with rewards that defines our objective for that task. In contrast, there are two distinct sources of data in our problem setting: task-agnostic, unlabeled prior data, which we denote as $\mathcal{D}_{\text{prior}}$, and task-specific data, which we denote as $\mathcal{D}_{\mathbb{T}}$, where $\mathbb{T}$ represents our task. The

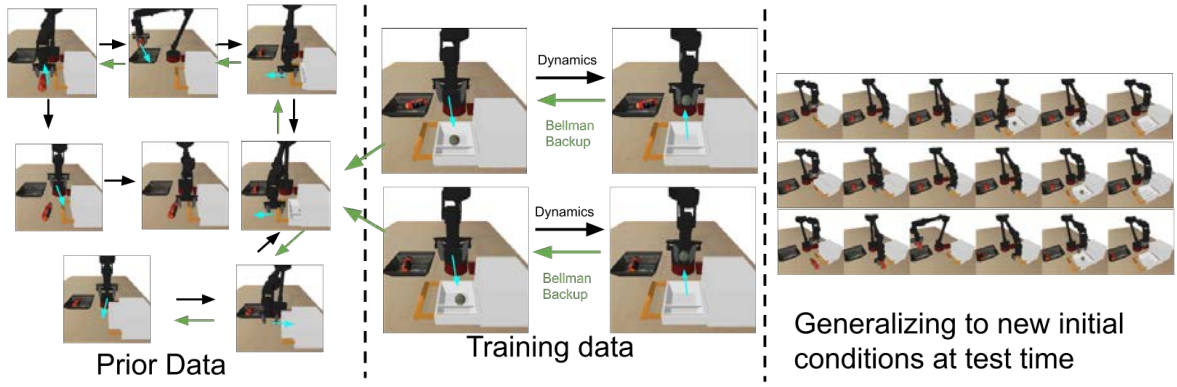


Figure 5.2: **Connecting new skills to past experience.** Q-learning propagates information backwards in a trajectory (middle) and by stitching together trajectories via Bellman backups from the task-agnostic prior data (left), it can learn optimal actions from initial conditions appearing in the prior data (right).

datapoints in $\mathcal{D}_{\text{prior}}$ simply consist of (s, a, s') transitions, and do not have any associated reward labels. While we cannot train a policy to achieve any particular objective from this data alone, it is informative about the dynamics of the MDP where the data was collected. On the other hand, the datapoints in $\mathcal{D}_{\mathbb{T}}$ consist of (s, a, s', r) tuples, and can be used for learning a policy that maximizes the observed reward. To summarize, the input and output of our problem setting are as follows:

Input: Datasets $\mathcal{D}_{\text{prior}}$ (with no reward annotations), $\mathcal{D}_{\mathbb{T}}$ (with sparse rewards for task $\mathbb{T}$).
Return: Policy π trained to execute task $\mathbb{T}$, which should be able to generalize broadly to new initial conditions. We would like to leverage $\mathcal{D}_{\text{prior}}$ for the latter.

5.4 Connecting New Skills to Past Experience via Dynamic Programming

Our approach is conceptually very simple: use offline RL to incorporate prior data into the training for the new skill. However, the reasons why this approach should be effective in our problem setting are somewhat nuanced. In this section, we will discuss how model-free dynamic programming methods based on Q-learning can be used to connect new tasks to past experience. Before presenting COG, let's first briefly revisit off-policy deep RL algorithms.

Standard off-policy deep RL methods, such as SAC [50], maintain a parametric action-value or Q-function, $Q_\theta(\mathbf{s}, \mathbf{a})$, and optionally a parametric policy, $\pi_\phi(\mathbf{a}|\mathbf{s})$, with parameters θ and ϕ, respectively. These methods typically train $Q_\theta(\mathbf{s}, \mathbf{a})$ to predict the return under π_ϕ in the policy evaluation step, and then update π_ϕ in the direction of increasing Q-values in the

policy improvement step:

$$\theta^{k+1} \leftarrow \arg\min_{\theta} \mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}' \sim \mathcal{D}_{\mathbb{T}}} \left[\left((r(\mathbf{s},\mathbf{a}) + \gamma \mathbb{E}_{\mathbf{a}' \sim \hat{\pi}^k(\mathbf{a}'|\mathbf{s}')}[\hat{Q}^k(\mathbf{s}',\mathbf{a}')]) - Q_\theta(\mathbf{s},\mathbf{a}) \right)^2 \right] \text{ (policy evaluation)}$$

$$\phi^{k+1} \leftarrow \arg\max_{\phi} \mathbb{E}_{\mathbf{s} \sim \mathcal{D}_{\mathbb{T}}, \mathbf{a} \sim \pi_\phi^k(\mathbf{a}|\mathbf{s})} \left[\hat{Q}_\theta^{k+1}(\mathbf{s},\mathbf{a}) \right] \quad \text{(policy improvement)}$$

In order to see how this model-free procedure can help us stitch together different behaviors (which we will define shortly), we start with some intuition about the Q-learning process. Q-learning propagates information backwards through a trajectory. State-action pairs at the end of a trajectory with a high reward are assigned higher values, and these values propagate to states that are further back in time, all the way to the beginning of the trajectory. Recall the example from Section 5.1, where the goal is to take an object out of a drawer. Assume that a reward of $+1$ is obtained when the object has been taken out of the drawer, and otherwise the reward is 0. Under this reward, the state where the object has been taken out will have the highest value, and the state where the robot has grasped the object (but not yet lifted it) will have a slightly lower value, since it is farther away from the successful completion of the task. The initial state, where the robot gripper is far away from the object, will have very low value. See Figure 5.2 (right) for an illustration of such states. However, the initial state will still have a non-zero value, since the Q-function "understands" that it is possible to reach high-reward states. Any state for which there does not exist a valid path to a high-reward state will have a value that is equal to zero, and all other states will have non-zero values, decreasing exponentially (in the discount) with distance to the state where object is out of the drawer.

We may now ask, what happens if at test-time, the robot is asked to perform the task from some new state that was not seen in $\mathcal{D}_{\mathbb{T}}$, such as a state where the drawer is closed? Such a state would either have a value of zero or, even worse, an arbitrary value, since it is outside of the training distribution. Therefore, the robot would likely not succeed at the task from this situation. Of course, we could train the new skill from a wider range of initial states, but if each skill must be learned from every possible starting state, it will quickly become prohibitively costly to train large skill repertoires.

What if the Q-learning method was now augmented with an additional large dataset that contains a wide variety of other behaviors, but *does not contain any data for the new task*? Such a dataset can still help us learn a much more useful policy, even in the absence of reward annotation: it can provide us with trajectories that (approximately) connect states not observed in $\mathcal{D}_{\mathbb{T}}$ (e.g., a closed drawer) to states appearing in successful executions in $\mathcal{D}_{\mathbb{T}}$ (e.g., open drawer), as shown in Figure 5.2. If the prior dataset $\mathcal{D}_{\text{prior}}$ is large enough, it can inform the policy of different ways *of reaching states from which the new task is solvable*. For example, if an object obstructs the drawer, and the prior dataset contains pick and place trajectories, then the policy can reason that it can unobstruct the drawer before opening it. Model-free Q-learning alone, without any skill decomposition or planning, can propagate values from $\mathcal{D}_{\mathbb{T}}$ into $\mathcal{D}_{\text{prior}}$, allowing us to learn a policy that can execute the task from a much broader distribution of initial states without actually seeing full executions of the task

from these states. Even without a single trajectory that both opens the drawer and takes the object out, as long as there is a non-zero overlap between $\mathcal{D}_{\text{prior}}$ and $\mathcal{D}_{\mathbb{T}}$, Q-learning can still learn from $\mathcal{D}_{\text{prior}}$.

Offline RL via conservative Q-learning (CQL). In order to incorporate the prior data $\mathcal{D}_{\text{prior}}$ into the RL process, we require an algorithm that can effectively utilize such prior data without actually interacting with the environment from the same initial states. Standard off-policy Q-learning and actor-critic algorithms are susceptible to out-of-distribution actions in this setting [97, 88, 85]. We instead utilize the conservative Q-learning (CQL) [86] algorithm that additionally penalizes Q-values on out-of-distribution actions during training. CQL learns Q-functions, $Q_\theta(\mathbf{s}, \mathbf{a})$, such that the expected policy value under Q_θ lower-bounds the true policy value π_ϕ, by minimizing the log-sum-exp of the Q-values at each state $\mathbf{s}$, while maximizing the expected Q-value on the dataset action distribution, in addition to standard Bellman error training as shown in Equation 5.1. The training objective shown in Equation 5.1 is the variant of CQL used in this paper:

$$\min_Q \; \alpha \mathbb{E}_{\mathbf{s}\sim\mathcal{D}_{\text{prior}}\cup\mathcal{D}_{\mathbb{T}}}\left[\log \sum_{\mathbf{a}} \exp(Q(\mathbf{s},\mathbf{a})) - \mathbb{E}_{\mathbf{a}\sim\mathcal{D}_{\text{prior}}\cup\mathcal{D}_{\mathbb{T}}}\left[Q(\mathbf{s},\mathbf{a})\right]\right] + \frac{1}{2}\mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}'\sim\mathcal{D}_{\mathbb{T}}\cup\mathcal{D}_{\text{prior}}}\left[\left(Q - \mathcal{B}^{\pi_k}\bar{Q}\right)^2\right]. \tag{5.1}$$

We instantiate CQL as an actor-critic algorithm. The policy improvement step remains unchanged as compared to a standard off-policy RL method, as discussed previously.

Online fine-tuning. In addition to the completely offline phase that utilizes the conservative Q-learning algorithm, we can further fine-tune the resulting policy using a small amount of online interaction with the environment. In order to do so, we train completely offline using CQL for 1m gradient steps, and then perform online training by periodically unrolling the policy in the environment and training on this additional data, starting from the solution obtained in the purely offline phase. We still utilize the CQL algorithm in this phase, except we only utilize the data collected via online interaction for fine-tuning purposes.

5.5 End-to-End Robotic Learning with Prior Datasets

In this section, we discuss how our method can be instantiated in a practical robotic learning system.

MDP for robotic skills. The state observation $s \in \mathcal{S}$ consists of the robot's current camera observation, which is an RGB image of size 48×48 for simulated experiments, and 64×64 for real robot experiments, and the current robot state. The robot state consists of the end-effector pose (Cartesian coordinates and Euler angles), and the extent to which the gripper is open (represented using a continuous value). The action space $\mathcal{A}$ consists of six continuous actions and two discrete actions. The six continuous actions correspond to controlling the end-effector's 3D coordinates and its orientation. The first discrete action corresponds to opening or closing the gripper, while the second discrete action executes a return to the robot's starting configuration. We use sparse reward functions for all of our

tasks: a reward of +1 is provided when a task has been executed successfully, while a zero reward is provided for all other states. We do not have any terminal states in our MDP.

Data collection. Our prior data collection takes place before any task-specific learning has happened. Since a completely random policy will seldom execute behaviors of interest, we bias our data collection policy towards executing more interesting behavior through the use of weak scripted policies: these policies typically have a success rate of 30-50% depending on the complexity of the task they are performing. More details on the scripted policies can be found in Appendix D.1.

Neural network architectures. Since we learn to stitch together behaviors directly from raw, visual observation inputs, we utilize convolutional neural networks (ConvNets) to represent our policy π_ϕ and the Q-function Q_θ. For both simulated and real world experiments, we use a 8-layer network: the first five layers consist of alternating convolutional and max-pooling layers (starting with a convolution), with a stride of 1 and kernel size of 3 for convolutional layers, and a stride of two for max-pooling layers. Each convolutional layer has 16 filters. We then pass the flattened output of the convolutional layer to three fully connected layers, of size 1024, 512 and 256. We use the ReLU non-linearity for all convolutional and fully-connected layers. A pictorial block diagram of these architectures is provided in Appendix D.1.

5.6 Experiments

We aim to answer the following questions through our experiments: **(1)** Can model-free RL algorithms effectively leverage prior, task-agnostic robotic datasets for learning new skills? **(2)** Can our learned policies solve new tasks, especially from novel initial conditions, by stitching together behavior observed during training? **(3)** How does our approach compare to alternative methods for incorporating prior data (such as behavior cloning)? **(4)** Is the addition of prior data essential for learning to solve the new task? To this end, we evaluate our approach on a number of long-horizon, multi-step reasoning robotic tasks with different choices of $\mathcal{D}_{\mathbb{T}}$ and $\mathcal{D}_{\text{prior}}$ and then perform an ablation study to understand the benefits of incorporating unlabeled offline datasets into robotic learning systems via offline reinforcement learning methods.

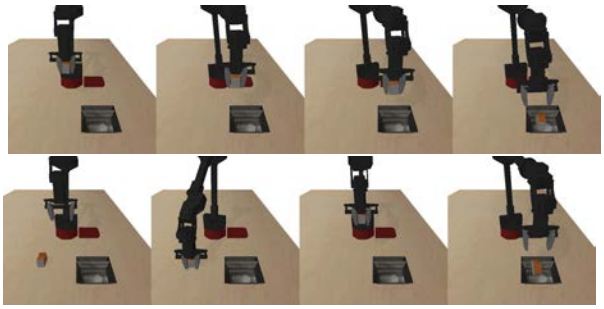


Figure 5.3: **Picking and placing**. Example executions from our *learned* policy. The first row shows the training condition, where the robot starts out already holding the object, and it only needs to place it in the tray. In the second condition (shown in second row), the robot must first grasp the object before placing it into the tray.

Experimental Setup

We evaluate our approach in simulation (see Figures 5.3 and 5.4) and on a real-robot task with a WidowX low-cost arm (see Figure 5.6).

Pick and place. Our first simulated environment is shown in Figure 5.3. It consists of a 6-DoF WidowX robot in front of a tray containing a small object and a tray. The objective is to put the object inside the tray. The reward is +1 when the object has been placed in the box, and zero otherwise. A simple initial condition for this task involves the robot already holding the object at the start of the episode, while a harder initial condition is when the robot has to first pick up the object. For this simplified experimental setting, the prior data consists of 10K grasping attempts from a randomized scripted policy (that has a success rate of about 40%, details of this policy are in Appendix D.1). Note that we do not provide any labels for which attempts were successful, and which were not. The task-agnostic prior dataset also consists of behaviors that may be irrelevant for the task. The task-specific data consists of 5K placing attempts from a different scripted policy (with a high success rate of over 90%, since the tray position is unchanged across trials), and these trajectories are labeled with rewards. Note that there is no single trajectory in our dataset that solves the complete pick and place task, but the prior and task-specific datasets have a non-zero overlap in their state distribution.

Grasping from a drawer. Our second and more complex simulated environment is shown in Figure 5.4. It consists of a 6-DoF WidowX robot and a larger variety of objects. The robot can open or close a drawer, grasp objects from inside the drawer or on the table, and place them anywhere in the scene. Some of these behaviors require various pre-conditions. For example, grasping an object from the drawer might require opening that drawer, which in turn might require moving an obstruction out of the way. The task here consists of taking an object out of a drawer (Figure 5.4). A reward of +1 is obtained when the object has been taken out, and zero otherwise. When learning the new task, the drawer always starts out open. The more difficult test conditions include ones where the drawer starts out closed, the top drawer starts out open (which blocks the handle for the lower drawer),

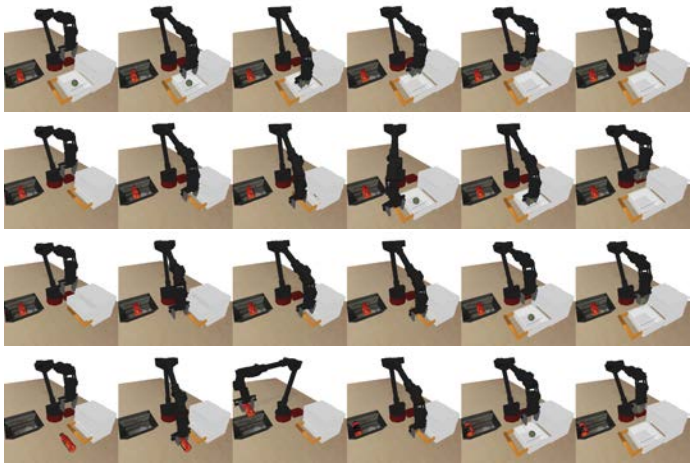


Figure 5.4: **Grasping from the drawer with our learned policy.** The first row shows the training condition, which requires grasping from an open drawer. The robot only needs to grasp the object and take it out of the drawer to get a reward. The subsequent rows show the harder test-time initial conditions which require, respectively: opening the drawer before taking out the object, closing the top drawer before opening the bottom one and taking out the object, and removing an obstruction (red) bottle before opening the drawer.

Task & Initial Condition	No prior data	BC			COG
		init	all	oracle	
place in box					
object in gripper	**1.00 (0.00)**	**1.00 (0.00)**	0.94 (0.01)	0.95 (0.01)	**1.00 (0.00)**
object in tray	0.00 (0.00)	0.00 (0.00)	0.02 (0.00)	0.02 (0.01)	**0.96 (0.04)**
grasp from drawer					
open drawer	0.98 (0.01)	**0.99 (0.00)**	0.63 (0.01)	0.82 (0.01)	0.98 (0.02)
closed drawer	0.00 (0.00)	0.00 (0.00)	0.23 (0.01)	0.27 (0.03)	**0.68 (0.07)**
blocked drawer 1	0.00 (0.00)	0.00 (0.00)	0.34 (0.03)	0.35 (0.03)	**0.78 (0.07)**
blocked drawer 2	0.00 (0.00)	0.00 (0.00)	0.22 (0.03)	0.25 (0.01)	**0.76 (0.09)**

Table 5.1: **Results for simulated experiments.** Mean (Standard Deviation) success rate of the learned policies for our method (COG), its ablations and prior work. For the grasping from drawer task, blocked drawer 1 and 2 are initial conditions corresponding to the third and fourth rows of Figure 5.4. Note that COG successfully performs both tasks in the majority of cases, from all initial conditions. SAC (–) diverged in our runs.

and an object starts out in front of the closed drawer, which must be moved out of the way before opening. These settings are illustrated in Figure 5.4. The prior data for this environment is collected from a collection of scripted randomized policies. These policies are capable of opening and closing both drawers with 40-50% success rates, can grasp objects in the scene with about a 70% success rate, and place those objects at random places in the scene (with a slight bias for putting them in the tray). The prior data does *not* contain any interactions with the object inside the drawer and also contains data irrelevant to solving the task, such as behavior that blocks the drawer by placing objects in front of it. There are 1.5m datapoints (transitions) in the prior dataset, and 300K datapoints in the task-specific dataset.

Baselines and comparisons. We compare COG to: **(1)** pre-training a policy via behavioral cloning on the prior data and then fine-tuning with offline RL on the new task, denoted as **BC-init**, **(2)** a naïve behavioral cloning baseline, denoted as **BC**, which trains with BC on all data, **(3)** an "oracle" version of behavioral cloning that is provided with handpicked successful trajectories on the new task, denoted as **BC-oracle** that is indicative of an upper bound on performance of selective cloning methods on a task, **(4)** a standard baseline off-policy RL method, **SAC** [50], and finally **(5)** an ablation of our method without any prior data, indicated as **no prior data**.

Empirical Results

Simulation results. The results for our simulation experiments are summarized in Table 5.1. For all methods, we train the policy via offline RL until the success rate converges, and evaluate the final policy on 250 trials, after training has converged. We then average these success rates across three random seeds for each experiment. Detailed learning curves can be found in Appendix D.3. We first note that our data-driven approach generally performs well for all initial conditions on both tasks. The policy is able to leverage the prior data

to automatically determine that a closed drawer should be opened before grasping, and obstructions should be moved out of the way prior to drawer opening, despite never having seen complete episodes that involve both opening the drawer and taking out the object, and not having any reward or success labels in the prior data.

COG also significantly outperforms the behavioral cloning baselines, ablations, and a standard off-policy SAC algorithm for all novel initial conditions. Since we use an entropy-regularized stochastic policy for the behavioral cloning variants (BC in Table 5.1), they are able to achieve a non-zero success rate, but the performance is far below what is achieved by COG. This is likely due to their inability to distinguish between behavior that supports the new task and behavior that is meaningful but not useful at the current time (for example, grasping an arbitrary object in the drawer task is not useful if that object is not blocking the drawer). Even when a behavioral cloning method is provided with handpicked successful trajectories for the new task, shown as BC-oracle in Table 5.1), we find that it is unable to complete the new task. Unsurprisingly, an ablation of our method that does not make use of prior data is only able to solve the task from states that were previously seen in the task-specific dataset. BC-init, which pre-trains the policy on the prior data via cloning followed by RL on the task data, is unable to solve the task for unseen initial conditions. We believe this is due to catastrophic forgetting of behavior learned during pre-training, and this adversely affects the policy's performance when evaluated on new initial conditions.

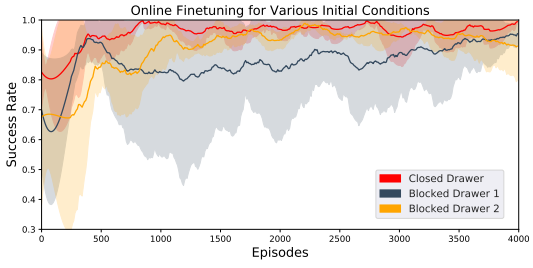


Figure 5.5: **Results from online fine-tuning.** We see that online fine-tuning further improves the performance of the learned policy, bringing it to over 90% success rates for all possible initial conditions for the drawer task, and only requires a small amount of additional data.

Online fine-tuning. While our method is able to obtain high success rates from offline learning alone, we also evaluated if the learned policies can be further improved via online fine-tuning. The results from these online fine-tuning experiments are shown in Figure 5.5. We see that for all of the novel initial conditions for the drawer task, the policy is able to achieve a success rate of over 90% from collecting only a small amount of additional episodes (500-4000, depending on the task). In our real world experimental setup (which we describe below), we are able to collect 3K episodes in a single day (autonomously), making this requirement quite feasible for real world problems. We compared this fine-tuning experiment against fine-tuning with SAC (starting from a behavior cloned policy), which performed substantially worse (see Figure D.3 in Appendix D.2), likely due to the low initial performance of the BC policy, and since the Q-function for SAC needs to be learned from scratch in this setting. Prior work has also observed that fine-tuning a behavior-cloned policy with SAC typically leads to some unlearning at the start, which further reduces the performance of this policy [112]. More details can be found in Appendix D.2.

Real-world evaluation. Our real world setup (see Figure 5.6) consists of a WidowX robotic arm in front of a drawer, and an object inside the drawer. The task is to take the object out of the drawer. As before, the reward is +1 on completion, zero otherwise. During training for the new task, the drawer starts open. The prior dataset consists of drawer opening and closing from a scripted randomized policy (details in Appendix D.1). As before, the prior dataset has no reward labels, and no instances of the new task (object grasping). The task-specific data is collected using a scripted grasping policy, which has a success rate of about 50%. The prior dataset consists of 80K transitions collected over 20 hours, and the new task dataset has 80K transitions (4K grasp attempts) collected over 34 hours. Our learned policy succeeds in **7/8** trials when the drawer starts out closed, and substantially outperforms the BC-oracle baseline, which **never** succeeds on this real-world task.

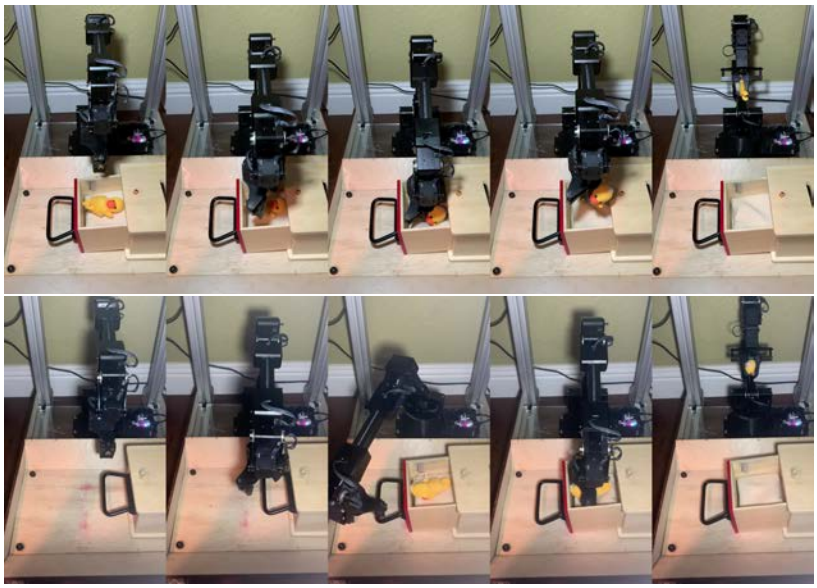


Figure 5.6: **Real world drawer opening and grasping.** The top row shows the training condition, which requires grasping an object from an open drawer. The bottom row shows the behavior of the learned policy in the test condition, where the drawer starts closed, and shows a rollout from the learned policy, which never saw a complete trajectory of opening a drawer and grasping together at training time.

5.7 Discussion

We showed how model-free reinforcement learning can utilize prior data to improve the generalization of learned skills, effectively stitching together behaviors so that, when the initial conditions at test time differ from those under which a new task was learned, the agent can still perform the task by stitching together appropriate behaviors seen in the prior data. For instance, a robot that learned to grasp an object from an open drawer can automatically learn to open the drawer at test time when it starts closed, without ever having seen a complete opening and grasping trajectory. This sort of composition of skills is typically seen as the domain of model-based planning, and often approached in learning-based control from the standpoint of hierarchy. Our approach suggests that such seemingly complex multi-stage behaviors could emerge automatically as a consequence of model-free reinforcement learning with the right training data. In effect, we show that model-free RL can use prior data to learn

the "mechanics" of the world, without any explicit hierarchy or model representation. Our initial instantiation of this idea does leave considerable room for future research. COG also employs a very naïve approach for labeling rewards in the prior data, and more sophisticated reward inference methods could also improve performance. More broadly, we believe that further exploration into the ways that model-free RL methods can enable acquisition of implicit mechanical models of the world is a promising direction for data-driven robotics.

Chapter 6

Conclusion

We started this thesis with the following conjecture: robot learning can be made more effective by making the entire process more data-driven. Our work built on recent advances in deep reinforcement learning, and introduced new problem settings and algorithms that enabled transferring these advances to real world robotic learning. While we believe the contributions made in thesis strongly support this conjecture, there are still numerous open questions that need to be resolved. We discuss some of the these challenges and potential directions for future research in the rest of this chapter.

Chapters 2 and 3 focused on the problem of specifying rewards for robotic tasks in a simple, intuitive and efficient fashion. We presented techniques for learning the reward function from goal examples and occasional human feedback, and were able to end-to-end train both policies and reward functions on image observations, in the real world. The problem setting we introduced (learning from goal examples) substantially improved upon prior problem settings for data-driven reward specification (such as inverse RL or learning from preferences) by significantly reducing the data requirements for learning an accurate reward function.

While our methods improve upon prior work in this area, they do have some limitations. First, providing visual goal examples to specify a task limits the type of tasks we can learn using the proposed methods, since some tasks cannot be described via static images. For examples, the task of juggling a set of balls. Second, the number of active queries required per task in VICE-RAQ—around 50 per training run—is still non-trivial. It is quite possible that a more intelligent query criterion could reduce this number further, and a promising direction for future work is to incorporate techniques for quantifying model uncertainty in the classifier, such as Bayesian neural networks. Lastly, our method does not benefit from any shared structure *between* tasks. In reality, tasks have considerable shared structure, and a classifier that incorporates data from multiple tasks, analogously to prior work on meta-learning [164, 166], could in principle further reduce the number of queries and examples that is needed.

Chapters 4 and 5 focused on incorporating previously collected datasets for robotic RL. While using demonstrations to speed up reinforcement learning is a popular strategy, this typically requires collecting demonstrations for the specific task of interest. In Chapter 4, we presented PARROT, a generative pre-training approach that allowed us to speed up learning

on a task by transferring knowledge from related, but distinct tasks. One key limitation of this work is that it requires a large prior dataset of near-expert trajectories to work well, and collecting such a dataset can difficult, especially on real robots. One way that future work can overcome this limitation is by building the prior dataset incrementally. We can do so by adopting a lifelong learning approach: we start out with a prior trained on N tasks, use it to speed up learning on the N+1 task, and expand the prior to incorporate the newly solved tasks. Overcoming the technical challenges in adapting PARROT to a lifelong learning setting would make an exciting direction of research.

In Chapter 5, we presented COG, method for incorporating prior data into RL from a more diverse set of sources (not just expert trajectories). We showed that unlabeled prior datasets can enable RL agents to stitch together relevant behaviors from such prior datasets, allowing robots to perform a type of "common-sense" reasoning when they encounter new scenarios. While COG employed a simple technique for labeling past experiences, future work could utilize more sophisticated reward inference approaches for this purpose. While the current method treats all prior data as equally important, it would be interesting to investigate approaches that allow prioritizing the different datapoints in the prior data for faster learning.

In addition to the problems discussed above, there are parts of our central conjecture that remain open. While we are convinced that learning-based approaches for task specification provide much more flexibility when compared to hand-engineered reward functions, it remains to be seen if learning from real world experience alone will be enough for learning more complex tasks. While the presented algorithms are able to learn efficiently, most robotic tasks in this work involve performing simple object re-arrangements using a parallel jaw gripper. One potential direction for pushing the envelope on task complexity is to re-think the relationship between real world data and simulation. While this thesis focused on effectively utilizing real world data (as opposed to simulation) for robotic learning, it would be interesting to explore how real data could be used to learn more realistic simulators, and how we can utilize these data-driven simulators for effective robotic learning.

Works in visual model-based reinforcement learning (typically referred to as visual foresight [25]) are one interesting line of work in data-driven simulation, but suffer from the challenge of making pixel-perfect predictions several seconds into the future. Another promising line of research utilizes cyclic generative adversarial networks (CycleGANs [173]) to generate realistic images from simulated images, but rely on having access to an accurate underlying physics engine [133]. Recently, graph neural networks have emerged as a promising tool for learning simulators from data [139], and future work could look into combining them with GAN-based approaches for rendering realistic images and deep reinforcement learning for learning control policies. For tasks that are particularly difficult to learn from real world data alone (such as dexterous in-hand manipulation with a five-fingered robot hand [120]), efforts to combine real world data with simulation could be particularly fruitful. This could help overcome a critical limitation discussed above: limited task complexity.

We hope this thesis can serve as a useful step towards building robots that can operate autonomously in open, unstructured environments, and the methods presented here can serve

as useful building blocks for future work.

Bibliography

[1] Pieter Abbeel and Andrew Y. Ng. "Apprenticeship learning via inverse reinforcement learning". In: *Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004), Banff, Alberta, Canada, July 4-8, 2004*. 2004.

[2] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. "An Optimistic Perspective on Offline Reinforcement Learning". In: (2019).

[3] Dario Amodei et al. "Concrete Problems in AI Safety". In: *ArXiv Preprint* abs/1606.06565 (2016).

[4] Lynton Ardizzone et al. "Analyzing Inverse Problems with Invertible Neural Networks". In: *ICLR*. 2019.

[5] Brenna D. Argall et al. "A survey of robot learning from demonstration". In: *Robotics and autonomous systems* 57(5) (2009), pp. 469–483.

[6] Pierre-Luc Bacon, Jean Harb, and Doina Precup. "The Option-Critic Architecture". In: *AAAI*. 2017.

[7] Daniel S. Brown, Yuchen Cui, and Scott Niekum. "Risk-Aware Active Inverse Reinforcement Learning". In: *Conference on Robot Learning (CoRL)*. 2018.

[8] Serkan Cabi et al. "A Framework for Data-Driven Robotics". In: *CoRR* (2019). eprint: 1909.12200.

[9] Yash Chandak et al. "Learning Action Representations for Reinforcement Learning". In: *ICML*. 2019.

[10] Angel X. Chang et al. "ShapeNet: An Information-Rich 3D Model Repository". In: *CoRR* abs/1512.03012 (2015).

[11] Yevgen Chebotar et al. "Path integral guided policy search". In: *2017 IEEE International Conference on Robotics and Automation, ICRA 2017, Singapore, Singapore, May 29 - June 3, 2017*. 2017, pp. 3381–3388.

[12] Paul F. Christiano et al. "Deep Reinforcement Learning from Human Preferences". In: *NIPS*. 2017.

[13] Robert Cohn, Edmund H. Durfee, and Satinder P. Singh. "Comparing Action-Query Strategies in Semi-Autonomous Agents". In: *AAAI Conference on Artificial Intelligence AAAI*. 2011.

[14] E Coumans and Y Bai. "Pybullet, a python module for physics simulation for games, robotics and machine learning". In: *GitHub repository* (2016).

[15] Yuchen Cui and Scott Niekum. "Active Reward Learning from Critiques". In: *International Conference on Robotics and Automation (ICRA)*. 2018.

[16] Christian Daniel et al. "Active Reward Learning". In: *Proceedings of Robotics: Science and Systems*. Berkeley, USA, July 2014. DOI: 10.15607/RSS.2014.X.031.

[17] Peter Dayan and Geoffrey E. Hinton. "Feudal Reinforcement Learning". In: *NIPS*. 1992.

[18] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *NAACL-HLT*. 2019.

[19] Thomas G. Dietterich. "The MAXQ Method for Hierarchical Reinforcement Learning". In: *ICML*. 1998.

[20] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. "Density estimation using Real NVP". In: *ICLR*. 2017.

[21] Jeff Donahue and Karen Simonyan. "Large Scale Adversarial Representation Learning". In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by Hanna M. Wallach et al. 2019, pp. 10541–10551. URL: https://proceedings.neurips.cc/paper/2019/hash/18cdf49ea54eec029238fcc95f76ce41-Abstract.html.

[22] Yan Duan et al. "One-shot Imitation Learning". In: *Neural Information Processing Systems (NIPS)* (2017).

[23] Yan Duan et al. "RL2: Fast Reinforcement Learning via Slow Reinforcement Learning". In: *arXiv preprint arXiv:1611.02779* (2016).

[24] Frederik Ebert et al. "Visual Foresight: Model-Based Deep Reinforcement Learning for Vision-Based Robotic Control". In: *CoRR* (2018). eprint: 1812.00568. URL: http://arxiv.org/abs/1812.00568.

[25] Frederik Ebert et al. "Visual foresight: Model-based deep reinforcement learning for vision-based robotic control". In: *arXiv arXiv:1812.00568* (2018).

[26] Ashley D Edwards. "Perceptual Goal Specifications for Reinforcement Learning". PhD thesis. Georgia Institute of Technology, 2017.

[27] Ben Eysenbach, Ruslan Salakhutdinov, and Sergey Levine. "Search on the Replay Buffer: Bridging Planning and Reinforcement Learning". In: *NeurIPS*. Ed. by Hanna M. Wallach et al. 2019, pp. 15220–15231.

[28] Rasool Fakoor et al. "Meta-Q-Learning". In: *ICLR*. 2020.

[29] Nima Fazeli et al. "See, feel, act: Hierarchical learning for complex manipulation skills with multisensory fusion". In: *Science Robotics* (2019).

[30] C. Finn et al. "Deep Spatial Autoencoders for Visuomotor Learning". In: *ICRA*. 2016.

[31] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". In: *arXiv preprint arXiv:1703.03400* (2017).

[32] Chelsea Finn and Sergey Levine. "Deep visual foresight for planning robot motion". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 2786–2793.

[33] Chelsea Finn, Sergey Levine, and Pieter Abbeel. "Guided Cost Learning: Deep Inverse Optimal Control via Policy Optimization". In: *International Conference on Machine Learning (ICML)*. 2017.

[34] Chelsea Finn et al. "A Connection between Generative Adversarial Networks, Inverse Reinforcement Learning, and Energy-Based Models". In: abs/1611.03852 (2016).

[35] Chelsea Finn et al. "One-shot visual imitation learning via meta-learning". In: *Conference on Robot Learning (CoRL)* (2017).

[36] Carlos Florensa, Yan Duan, and Pieter Abbeel. "Stochastic Neural Networks for Hierarchical Reinforcement Learning". In: *ICLR*. 2017.

[37] Roy Fox et al. "Multi-Level Discovery of Deep Options". In: *CoRR* abs/1703.08294 (2017). arXiv: 1703.08294.

[38] Justin Fu, Katie Luo, and Sergey Levine. "Learning Robust Rewards with Adversarial Inverse Reinforcement Learning". In: *International Conference on Learning Representations (ICLR)*. 2018.

[39] Justin Fu, Katie Luo, and Sergey Levine. "Learning Robust Rewards with Adversarial Inverse Reinforcement Learning". In: *International Conference on Learning Representations (ICLR)*. 2018.

[40] Justin Fu et al. "Diagnosing bottlenecks in deep Q-learning algorithms". In: *arXiv preprint arXiv:1902.10250* (2019).

[41] Justin Fu et al. "Variational Inverse Control with Events: A General Framework for Data-Driven Reward Definition". In: *Advances in Neural Information Processing Systems*. 2018.

[42] Scott Fujimoto, David Meger, and Doina Precup. "Off-policy deep reinforcement learning without exploration". In: *arXiv preprint arXiv:1812.02900* (2018).

[43] Ali Ghadirzadeh et al. "Data-efficient visuomotor policy training using reinforcement learning and generative models". In: *CoRR* abs/2007.13134 (2020). arXiv: 2007.13134.

[44] Ali Ghadirzadeh et al. "Deep predictive policy training using reinforcement learning". In: *International Conference on Intelligent Robots and Systems*. 2017.

[45] Shixiang Gu et al. "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates". In: *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2017, pp. 3389–3396.

[46] Abhinav Gupta et al. *Robot Learning in Homes: Improving Generalization and Reducing Dataset Bias*. 2018. arXiv: 1807.07049 [cs.RO].

[47] Abhishek Gupta et al. "Relay Policy Learning: Solving Long-Horizon Tasks via Imitation and Reinforcement Learning". In: *Conference on Robot Learning*. Ed. by Leslie Pack Kaelbling, Danica Kragic, and Komei Sugiura. 2019.

[48] Tuomas Haarnoja et al. "Latent Space Policies for Hierarchical Reinforcement Learning". In: *ICML*. Ed. by Jennifer G. Dy and Andreas Krause. 2018.

[49] Tuomas Haarnoja et al. "Reinforcement Learning with Deep Energy-Based Policies". In: *International Conference on Machine Learning (ICML)*. 2017.

[50] Tuomas Haarnoja et al. *Soft Actor-Critic Algorithms and Applications*. Tech. rep. 2018.

[51] Tuomas Haarnoja et al. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: 2018.

[52] Dylan Hadfield-Menell et al. "Inverse Reward Design". In: *NIPS*. 2017.

[53] Aleksi Hämäläinen et al. "Affordance Learning for End-to-End Visuomotor Robot Control". In: *IROS*. 2019.

[54] Karol Hausman et al. "Multi-Modal Imitation Learning from Unstructured Demonstrations using Generative Adversarial Nets". In: *NIPS*. Ed. by Isabelle Guyon et al.

[55] Nicolas Heess et al. "Learning and Transfer of Modulated Locomotor Controllers". In: *CoRR* abs/1610.05182 (2016).

[56] Todd Hester et al. "Learning from Demonstrations for Real World Reinforcement Learning". In: *CoRR* abs/1704.03732 (2017). arXiv: 1704.03732.

[57] Irina Higgins et al. "beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework". In: *ICLR*. 2017.

[58] Irina Higgins et al. "DARLA: Improving Zero-Shot Transfer in Reinforcement Learning". In: *ICML*. 2017.

[59] Jonathan Ho and Stefano Ermon. "Generative Adversarial Imitation Learning". In: *Advances in Neural Information Processing Systems (NIPS)*. 2016.

[60] Jonathan Ho and Stefano Ermon. "Generative Adversarial Imitation Learning". In: *Advances in Neural Information Processing Systems (NIPS)*. 2016.

[61] Herke van Hoof et al. "Learning Robot In-Hand Manipulation with Tactile Features". In: (2015).

[62] Yordan Hristov, Alex Lascarides, and Subramanian Ramamoorthy. "Interpretable latent spaces for learning from demonstration". In: *arXiv preprint arXiv:1807.06583* (2018).

[63] De-An Huang et al. "Continuous Relaxation of Symbolic Planner for One-Shot Imitation Learning". In: *IROS*. 2019.

[64] De-An Huang et al. "Neural Task Graphs: Generalizing to Unseen Tasks from a Single Video Demonstration". In: (2018).

[65] Nathan Hunt et al. "Verifiably Safe Exploration for End-to-End Reinforcement Learning". In: *CoRR* abs/2007.01223 (2020).

[66] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *ICML*. 2015.

[67] Ashesh Jain et al. "Learning Trajectory Preferences for Manipulators via Iterative Improvement". In: *NIPS*. 2013.

[68] Stephen James, Michael Bloesch, and Andrew J Davison. "Task-Embedded Control Networks for Few-Shot Imitation Learning". In: *arXiv preprint arXiv:1810.03237* (2018).

[69] Natasha Jaques et al. "Way off-policy batch deep reinforcement learning of implicit human preferences in dialog". In: *arXiv preprint arXiv:1907.00456* (2019).

[70] Tobias Johannink et al. "Residual Reinforcement Learning for Robot Control". In: *ICRA*. 2019.

[71] Ryan Julian et al. "Efficient Adaptation for End-to-End Vision-Based Robotic Manipulation". In: *arXiv arXiv:2004.10190* (2020).

[72] Dmitry Kalashnikov et al. "QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation". In: *Conference on Robot Learning (CoRL)*. 2018.

[73] Rudolf Kalman. "A new approach to linear filtering and prediction problems". In: 82 (1 1960), pp. 35–45.

[74] Hilbert J. Kappen, Vicenc Gomez, and Manfred Opper. "Optimal control as a graphical model inference problem". In: 2009.

[75] Daniel Kappler, Jeannette Bohg, and Stefan Schaal. "Leveraging big data for grasp planning". In: *International Conference on Robotics and Automation*. IEEE, 2015.

[76] Michael J. Kearns and Satinder P. Singh. "Near-Optimal Reinforcement Learning in Polynomial Time". In: *Machine Learning* 49.2-3 (2002), pp. 209–232.

[77] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference for Learning Representations (ICLR)*. 2015.

[78] Diederik P. Kingma and Max Welling. "Auto-Encoding Variational Bayes". In: *ICLR*. Ed. by Yoshua Bengio and Yann LeCun. 2014.

[79] Thomas Kipf et al. "CompILE: Compositional Imitation Learning and Execution". In: *ICML*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. 2019.

[80] Petar Kormushev, Sylvain Calinon, and Darwin G Caldwell. "Robot motor skill coordination with EM-based reinforcement learning". In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2010.

[81] Petar Kormushev, Sylvain Calinon, and Darwin G. Caldwell. "Robot motor skill coordination with EM-based Reinforcement Learning". In: *IROS*. 2010.

[82] Ilya Kostrikov et al. "Discriminator-Actor-Critic: Addressing Sample Inefficiency and Reward Bias in Adversarial Imitation Learning". In: *arxiv: 1809.02925* (2018).

[83] Sanjay Krishnan et al. "DDCO: Discovery of Deep Continuous Options for Robot Learning from Demonstrations". In: *CoRL*. 2017.

[84] Tejas D. Kulkarni et al. "Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation". In: *Advances in Neural Information Processing Systems*. 2016.

[85] Aviral Kumar. *Data-Driven Deep Reinforcement Learning*. `https://bair.berkeley.edu/blog/2019/12/05/bear/`. BAIR Blog. 2019.

[86] Aviral Kumar et al. "Conservative Q-Learning for Offline Reinforcement Learning". In: *arXiv preprint arXiv:2006.04779* (2020).

[87] Aviral Kumar et al. "Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction". In: *NeurIPS*. 2019.

[88] Aviral Kumar et al. "Stabilizing off-policy q-learning via bootstrapping error reduction". In: *NeurIPS*. 2019.

[89] Vikash Kumar et al. "Learning Dexterous Manipulation Policies from Experience and Imitation". In: *CoRR* abs/1611.05095 (2016).

[90] Andras Gabor Kupcsik et al. "Data-Efficient Generalization of Robot Skills with Contextual Policy Search". In: *AAAI*. 2013.

[91] Michail G Lagoudakis and Ronald Parr. "Least-squares policy iteration". In: *Journal of machine learning research* 4.Dec (2003), pp. 1107–1149.

[92] Sascha Lange, Thomas Gabel, and Martin A. Riedmiller. "Batch Reinforcement Learning". In: *Reinforcement Learning*. Ed. by Marco Wiering and Martijn van Otterlo. Vol. 12. Adaptation, Learning, and Optimization. Springer, 2012, pp. 45–73.

[93] Sergey Levine. "Reinforcement Learning and Control as Probabilistic Inference: Tutorial and Review". In: *ArXiv Preprint* abs/1805.00909 (2018).

[94] Sergey Levine and Pieter Abbeel. "Learning Neural Network Policies with Guided Policy Search under Unknown Dynamics". In: *NIPS*. 2014.

[95] Sergey Levine et al. "End-to-End Training of Deep Visuomotor Policies". In: *Journal of Machine Learning (JMLR)* (2016).

[96] Sergey Levine et al. "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection". In: *I. J. Robotics Res.* (2018).

[97] Sergey Levine et al. "Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems". In: *arXiv preprint arXiv:2005.01643* (2020).

[98] Yunzhu Li, Jiaming Song, and Stefano Ermon. "InfoGAIL: Interpretable Imitation Learning from Visual Demonstrations". In: *Advances in Neural Information Processing Systems*. 2017.

[99] Manuel Lopes, Francisco S. Melo, and Luis Montesano. "Active Learning for Reward Estimation in Inverse Reinforcement Learning". In: *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML PKDD*. 2009.

[100] Corey Lynch et al. "Learning Latent Plans from Play". In: *Conference on Robot Learning*. 2019.

[101] Jeffrey Mahler et al. "Dex-Net 1.0: A cloud-based network of 3D objects for robust grasp planning using a Multi-Armed Bandit model with correlated rewards". In: *ICRA*. Ed. by Danica Kragic, Antonio Bicchi, and Alessandro De Luca. 2016.

[102] Anirudha Majumdar et al. "Risk-sensitive Inverse Reinforcement Learning via Coherent Risk Models". In: *Proceedings of Robotics: Science and Systems*. 2017.

[103] Ajay Mandlekar et al. "Iris: Implicit reinforcement without interaction at scale for learning control from offline robot manipulation data". In: *arXiv preprint arXiv:1911.05321* (2019).

[104] Ajay Mandlekar et al. *Learning to Generalize Across Long-Horizon Tasks from Human Demonstrations*. 2020. eprint: 2003.06085.

[105] Jan Matas, Stephen James, and Andrew J. Davison. "Sim-to-Real Reinforcement Learning for Deformable Object Manipulation". In: *Conference on Robot Learning (CoRL)*. 2018.

[106] Russell Mendonca et al. "Guided Meta-Policy Search". In: *arXiv preprint arXiv:1904.00956* (2019).

[107] Nikhil Mishra et al. "Meta-Learning with Temporal Convolutions". In: *arXiv:1707.03141* (2017).

[108] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. ISSN: 0028-0836.

[109] Rémi Munos. "Error bounds for approximate value iteration". In: *Proceedings of the National Conference on Artificial Intelligence*. 2005.

[110] Ofir Nachum et al. "Bridging the gap between value and policy based reinforcement learning." In: *Advances in Neural Information Processing Systems (NIPS)*. 2017.

[111] Ofir Nachum et al. "Data-Efficient Hierarchical Reinforcement Learning". In: *Advances in Neural Information Processing Systems*. Ed. by Samy Bengio et al. 2018.

[112] Ashvin Nair et al. "Accelerating Online Reinforcement Learning with Offline Datasets". In: *CoRR* abs/2006.09359 (2020). arXiv: 2006.09359.

[113] Ashvin Nair et al. "Overcoming Exploration in Reinforcement Learning with Demonstrations". In: *ICRA*. 2018.

[114] Suraj Nair and Chelsea Finn. "Hierarchical foresight: Self-supervised learning of long-horizon tasks via visual subgoal generation". In: *arXiv preprint arXiv:1909.05829* (2019).

[115] Andrew Ng and Stuart Russell. "Algorithms for Inverse Reinforcement Learning". In: *International Conference on Machine Learning (ICML)*. 2000.

[116] Andrew Y. Ng and Stuart J. Russell. "Algorithms for Inverse Reinforcement Learning". In: *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*. 2000, pp. 663–670.

[117] Brendan O'Donoghue et al. "Combining policy gradient and Q-learning". In: 2016.

[118] Aäron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. "Neural Discrete Representation Learning". In: *NeurIPS*. 2017.

[119] OpenAI. "Dota 2 with Large Scale Deep Reinforcement Learning". In: *Arxiv*. 2019.

[120] OpenAI. "Learning Dexterous In-Hand Manipulation". In: *arXiv preprint arXiv:1808.00177*. 2018.

[121] Tom Le Paine et al. "One-shot high-fidelity imitation: Training large-scale deep nets with rl". In: *arXiv preprint arXiv:1810.05017* (2018).

[122] Ronald Parr and Stuart J. Russell. "Reinforcement Learning with Hierarchies of Machines". In: *Advances in Neural Information Processing Systems*. 1997.

[123] Xue Bin Peng et al. "Advantage-Weighted Regression: Simple and Scalable Off-Policy Reinforcement Learning". In: *arXiv preprint arXiv:1910.00177* (2019).

[124] Xue Bin Peng et al. "DeepMimic: example-guided deep reinforcement learning of physics-based character skills". In: *ACM Trans. Graph.* (2018).

[125] Xue Bin Peng et al. "MCP: Learning Composable Hierarchical Control with Multiplicative Compositional Policies". In: *NeurIPS*. 2019.

[126] Xue Bin Peng et al. "Sim-to-Real Transfer of Robotic Control with Dynamics Randomization". In: *CoRR* abs/1710.06537 (2017).

[127] Jan Peters and Stefan Schaal. "Policy Gradient Methods for Robotics". In: *IROS*. 2006.

[128] Lerrel Pinto and Abhinav Gupta. "Supersizing Self-supervision: Learning to Grasp from 50K Tries and 700 Robot Hours". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2016.

[129] Lerrel Pinto and Abhinav Gupta. "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours". In: *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE. 2016, pp. 3406–3413.

[130] Aravind Rajeswaran et al. "Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations". In: *Robotics: Science and Systems*. 2018.

[131] Aravind Rajeswaran et al. "Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations". In: *RSS*. 2018.

[132] Kate Rakelly et al. "Efficient Off-Policy Meta-Reinforcement Learning via Probabilistic Context Variables". In: *ICML*. 2019.

[133] Kanishka Rao et al. "RL-CycleGAN: Reinforcement Learning Aware Simulation-to-Real". In: *CVPR*. 2020.

[134] Nathan D. Ratliff, J. Andrew Bagnell, and Martin Zinkevich. "Maximum margin planning". In: *ICML 2006*. 2006.

[135] Konrad Rawlik, Marc Toussaint, and Sethu Vijayakumar. "On Stochastic Optimal Control and Reinforcement Learning by Approximate Inference". In: *Robotics: Science and Systems (RSS)*. 2012.

[136] Nicholas Rhinehart, Rowan McAllister, and Sergey Levine. "Deep Imitative Models for Flexible Inference, Planning, and Control". In: *ICLR*. 2020.

[137] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 2nd ed. Pearson Education, 2003. ISBN: 0137903952.

[138] Andrei A. Rusu et al. "Sim-to-Real Robot Learning from Pixels with Progressive Nets". In: *Conference on Robot Learning (CoRL)*. 2017.

[139] Alvaro Sanchez-Gonzalez et al. "Learning to Simulate Complex Physics with Graph Networks". In: *ICML*. 2020.

[140] Stefan Schaal. "Learning from Demonstration". In: *NIPS*. Ed. by Michael Mozer, Michael I. Jordan, and Thomas Petsche. 1996.

[141] Connor Schenck and Dieter Fox. "Visual closed-loop control for pouring liquids". In: *International Conference on Robotics and Automation (ICRA)*. 2017.

[142] John Schulman, Xi Chen, and Pieter Abbeel. "Equivalence Between Policy Gradients and Soft Q-Learning". In: 2017.

[143] John Schulman et al. "Trust Region Policy Optimization". In: *International Conference on Machine Learning (ICML)*. 2015.

[144] John Schulman et al. "Trust Region Policy Optimization". In: *International Conference on Machine Learning (ICML)*. 2015.

[145] Tanmay Shankar and Abhinav Gupta. "Learning Robot Skills with Temporal Variational Inference". In: (2020).

[146] Tanmay Shankar et al. "Discovering Motor Programs by Recomposing Demonstrations". In: *ICLR*. 2020.

[147] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* (2016).

[148] Tom Silver et al. "Residual Policy Learning". In: *CoRR* abs/1812.06298 (2018). arXiv: 1812.06298.

[149] Avi Singh et al. "End-to-End Robotic Reinforcement Learning without Reward Engineering". In: *Robotics: Science and Systems* (2019).

[150] S. Singh, R. Lewis, and A. Barto. "Where do rewards come from?" In: *Proceedings of the International Symposium on AI Inspired Biology - A Symposium at the AISB 2010 Convention*. 2010.

[151] Jonathan Sorg, Satinder P. Singh, and Richard L. Lewis. "Reward Design via Online Gradient Ascent". In: *NIPS*. 2010.

[152] Richard S. Sutton, Doina Precup, and Satinder P. Singh. "Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning". In: *Artificial Intelligence* (1999).

[153] Christian Szegedy et al. "Intriguing properties of neural networks". In: *ICLR 2014*. 2014. URL: http://arxiv.org/abs/1312.6199.

[154] Emo Todorov. "General duality between optimal control and estimation". In: *IEEE Conference on Decision and Control (CDC)*. 2008.

[155] Emo Todorov. "Linearly-solvable Markov decision problems". In: *Advances in Neural Information Processing Systems (NIPS)*. 2007.

[156] Marc Toussaint. "Robot trajectory optimization using approximate inference". In: *International Conference on Machine Learning (ICML)*. 2009.

[157] Hsiao-Yu Fish Tung et al. "Reward Learning from Narrated Demonstrations". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.

[158] Dizan Vasquez, Billy Okal, and Kai Oliver Arras. "Inverse Reinforcement Learning algorithms and features for robot navigation in crowds: An experimental comparison". In: *2014 IEEE/RSJ International Conference on Intelligent Robots*. 2014.

[159] Mel Vecerik et al. "A Practical Approach to Insertion with Variable Socket Position Using Deep Reinforcement Learning". In: *arxiv: 1810.01531* (2018).

[160] Matej Vecerık et al. "Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards". In: *CoRR* abs/1707.08817 (2017). arXiv: 1707.08817.

[161] Jane X Wang et al. "Learning to reinforcement learn". In: *arXiv preprint arXiv:1611.05763* (2016).

[162] Yifan Wu, George Tucker, and Ofir Nachum. "Behavior Regularized Offline Reinforcement Learning". In: *arXiv preprint arXiv:1911.11361* (2019).

[163] Markus Wulfmeier, Peter Ondruska, and Ingmar Posner. "Maximum entropy deep inverse reinforcement learning". In: *arXiv preprint arXiv:1507.04888*. 2015.

[164] Annie Xie et al. "Few-Shot Goal Inference for Visuomotor Learning and Planning". In: *Conference on Robot Learning (CoRL)*. 2018.

[165] Annie Xie et al. "Improvisation through Physical Understanding: Using Novel Objects As Tools with Visual Foresight". In: *Robotics: Science and Systems*. 2019.

[166] Kelvin Xu et al. "Learning a Prior over Intent via Meta-Inverse Reinforcement Learning". In: *arXiv:1805.12573* (2018).

[167] Ali Yahya et al. "Collective robot reinforcement learning with distributed asynchronous guided policy search". In: *International Conference on Intelligent Robots and Systems (IROS)*. 2017.

[168] Tianhe Yu et al. "MOPO: Model-based Offline Policy Optimization". In: *arXiv preprint arXiv:2005.13239* (2020).

[169] Tianhe Yu et al. "One-Shot Imitation from Observing Humans via Domain-Adaptive Meta-Learning". In: *Robotics: Science and Systems (RSS)* (2018).

[170] Andy Zeng et al. "Learning Synergies between Pushing and Grasping with Self-supervised Deep Reinforcement Learning". In: (2018).

[171] Hongyi Zhang et al. "mixup: Beyond Empirical Risk Minimization". In: *ICLR 2018* (2018). URL: `http://arxiv.org/abs/1710.09412`.

[172] Allan Zhou et al. "Watch, Try, Learn: Meta-Learning from Demonstrations and Reward". In: (2020).

[173] Jun-Yan Zhu et al. "Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks". In: *ICCV*. 2017.

[174] Brian Ziebart. "Modeling purposeful adaptive behavior with the principle of maximum causal entropy". In: *PhD thesis, Carnegie Mellon University* (2010).

[175] Brian Ziebart et al. "Maximum Entropy Inverse Reinforcement Learning". In: *AAAI Conference on Artificial Intelligence (AAAI)*. 2008.

[176] Brian D. Ziebart et al. "Maximum Entropy Inverse Reinforcement Learning". In: *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*. 2008, pp. 1433–1438.

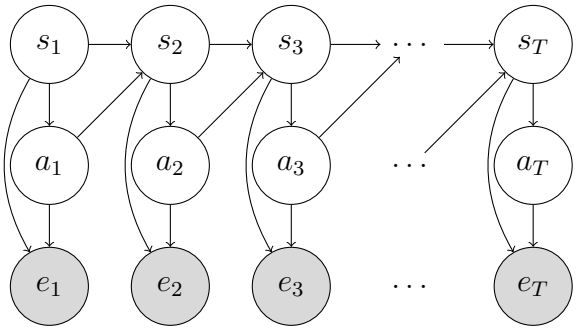
[177] Luisa M. Zintgraf et al. "VariBAD: A Very Good Method for Bayes-Adaptive Deep RL via Meta-Learning". In: *ICLR*. 2020.

Appendix A

Variational Inverse Control with Events

A.1 Message Passing Updates for Reinforcement Learning

In this section, we derive message passing updates that can be used to obtain an optimal policy in the graphical model for control (visualized below).



We define two backward messages, a state-action message $\beta(s_t, a_t) = p(e_{t:T} = 1|s_t, a_t)$ and a state message $\beta(s_t) = p(e_{t:T} = 1|s_t)$. The state message can be expanded in terms of the state-action message as:

$$\beta(s_t) = p(e_{t:T} = 1|s_t) = \int_{\mathcal{A}} \beta(s_t, a_t) p(a_t|s_t) da_t$$

We can then write a recursive form for the state-action message in terms of the state message:

$$\begin{aligned}
\beta(s_t, a_t) &= p(e_{t:T} = 1|s_t, a_t) = p(e_t = 1|s_t, a_t) p(e_{t+1:T} = 1|s_t, a_t) \\
&= p(e_t = 1|s_t, a_t) \int_{\mathcal{S},\mathcal{A}} \beta(s_{t+1}, a_{t+1}) p(a_{t+1}|s_{t+1}) p(s_{t+1}|s_t, a_t) ds_{t+1} da_{t+1} \\
&= p(e_t = 1|s_t, a_t) E_{s_{t+1}}[\beta(s_{t+1})]
\end{aligned}$$

Next, we define $\log p(e_t = 1|s_t, a_t) = r(s_t, a_t)$ as the reward factor and set the reference policy $p(a_t|s_t) = C$ to the uniform distribution as before. Non-uniform reference policies correspond to policy optimization with a modified reward function $r^{new}(s, a) = r^{old}(s, a) + \log C - \log p(a_t|s_t)$ and with a uniform reference policy. We can now assign familiar names to these messages, by defining $Q(s, a) = \log \beta(s, a)$ and $V(s) = \log \beta(s) - \log C$. Our message passing updates now resemble soft variants of Bellman backup equations:

$$V(s_t) = \log \int_{\mathcal{A}} \exp\{Q(s_t, a_t)\} da_t$$

$$Q(s_t, a_t) = [r(s_t, a_t) + \log C] + \log E_{s_{t+1}}[\exp\{V(s_{t+1})\}]$$

The constant $\log C$ term can be absorbed into the reward function to exactly match the equations we presented in Section 2.3, but we leave the term explicit for clarity of explanation. For the fixed horizon task we presented, adding a constant offset to the reward cannot change the optimal policy. As previously mentioned in Section A.2, under deterministic dynamics, $Q(s_t, a_t) = r(s_t, a_t) + V(s_{t+1})$, which aligns with MaxCausalEnt [174] and soft Q-learning [49, 110].

From these value functions, we can easily obtain the optimal policy $p(a_t|s_t, e_{1:T} = 1)$. First note that due to conditional independence, $p(a_t|s_t, e_{1:T} = 1) = p(a_t|s_t, e_{t:T} = 1)$. Applying Bayes' rule, we now have:

$$p(a_t|s_t, e_{t:T} = 1) = \frac{p(e_{t:T} = 1|s_t, a_t)p(a_t|s_t)}{p(e_{t:T} = 1|s_t)} = \frac{\beta(s_t, a_t)C}{\beta(s_t)} = \exp\{Q(s_t, a_t) - V(s_t)\}$$

A.2 Control as Variational Inference

Performing inference directly in the graphical model for control produces solutions that are optimistic with respect to stochastic dynamics, and produces risk-seeking behavior. This is because posterior inference is not constrained to force $p(s_{t+1}|s_t, a_t, e_{1:T}) = p(s_{t+1}|s_t, a_t)$: that is, it assumes that, like the action distribution, the next state distribution will "conspire" to make positive outcomes more likely. Prior work has sought to address this issue via the framework of causal entropy [174]. To provide a more unified treatment of control as inference, we instead present a variational inference derivation that also addresses this problem. f When conditioning the graphical model in Figure 2.3 on $e_{1:T} = 1$ as before, the optimal trajectory distribution is

$$p(\tau|e_{1:T}) \propto p(s_1) \prod_{t=1}^{T-1} p(s_{t+1}|s_t, a_t)p(a_t|s_t)e^{r(s,a)}.$$

We will assume that the action prior $p(a_t|s_t)$ is uniform without loss of generality, since non-uniform distributions can be absorbed into the reward term $e^{r(s,a)}$, as discussed in Appendix A.1.

The correct maximum entropy reinforcement learning objective emerges when performing variational inference in this model, with a variational distribution of the form

$q_\theta(\tau) = p(s_1)\prod_{t=1}^{T-1} p(s_{t+1}|s_t, a_t)q_\theta(a_t|s_t)$. In this distribution, the initial state distribution and dynamics are forced to be equal to the true dynamics, and only the action conditional $q_\theta(a_t|s_t)$, which corresponds to the policy, is allowed to vary. Writing out the variational objective and simplifying, we get

$$-D_{KL}(q_\theta(\tau)||p(\tau|e_{1:T})) = E_{\tau\sim q(\tau)}\left[\sum_{t=1}^{T} r(s_t, a_t) - \log q_\theta(a|s)\right].$$

We see that we obtain the same problem as (undiscounted) entropy-regularized reinforcement learning, where $q_\theta(a|s)$ serves as the policy. For more in-depth discussion, see Appendix A.4. We can recover the discounted objective by modifying the dynamics such that the agent has a $(1-\gamma)$ probability of transitioning into an absorbing state with 0 reward.

We have thus derived how maximum entropy reinforcement learning can be recovered by applying variational inference with a specific choice of variational distribution to the graphical model for control.

A.3 Derivations for Event-based Message Passing Updates

ALL query

The goal of the ALL query is to trigger an event at every timestep. Mathematically, we want trajectories such that $e_{1:T} = 1$. As the ALL query is mathematically identical to MaxEnt RL, we redirect the reader to Appendix A.1 for the derivation.

ANY query

The goal of the ANY query is to trigger an event at least once. Mathematically, we want trajectories such that $e_1 = 1$ or $e_2 = 1$ ... $e_T = 1$.

First, we introduce a more concise notation by introducing a stopping time $t^* = \text{argmin}_{t\geq 0}\{e_t = 1\}$ which denotes the first time that an event happens. Asking for the stopping time to be within a certain interval is the same as asking the event to happen at least once within that interval:

$$p(t^* \in [t, T]) = p(e_t = 1 \text{ or } e_{t+1} = 1 \text{ ... } e_T = 1)$$

We can now derive the message passing updates. We derive the state messages as:

$$\beta(s_t) = p(t^* \in [t, T]|s_t) = \int_{\mathcal{A}} p(t^* \in [t, T]|s_t, a_t)p(a_t|s_t)da_t$$

The state-action message can be derived as:

$$\begin{aligned}
\beta(s_t, a_t) &= p(t^* \in [t, T] | s_t, a_t) \\
&= p(e_t = 1 | s_t, a_t) + p(t^* \in [t+1, T] | s_t, a_t) - p(e_t = 1 | s_t, a_t) p(t^* \in [t+1, T] | s_t, a_t) \\
&= p(e_t = 1 | s_t, a_t) + p(e_t = 0 | s_t, a_t) p(t^* \in [t+1, T] | s_t, a_t) \\
&= p(e_t = 1 | s_t, a_t) + p(e_t = 0 | s_t, a_t) \int_{\mathcal{S}, \mathcal{A}} p(t^* \in [t+1, T], s_{t+1}, a_{t+1} | s_t, a_t) ds_{t+1} da_{t+1} \\
&= p(e_t = 1 | s_t, a_t) + p(e_t = 0 | s_t, a_t) E_{s_{t+1}} \left[\int_{\mathcal{A}} p(t^* \in [t+1, T] | s_{t+1}, a_{t+1}) p(a_{t+1} | s_{t+1}) da_{t+1} \right] \\
&= p(e_t = 1 | s_t, a_t) + p(e_t = 0 | s_t, a_t) E_{s_{t+1}} [\beta(s_{t+1})]
\end{aligned}$$

We can now define our Q and value functions as log-messages as done in Appendix A.1 to obtain the following backup rules:

$$V(s_t) = \log \int_{\mathcal{A}} \exp\{Q(s_t, a_t)\} da_t$$

$$Q(s_t, a_t) = \log \left(p(e_t = 1 | s_t, a_t) + p(e_t = 0 | s_t, a_t) E_{s_{t+1}} [\exp\{V(s_t)\}] \right)$$

One caveat here is that the policy, $p(a_t | s_t, t^* \in [t, T])$, always seeks to make the event happening in the future, which we refer to as the *seeking* policy. The correct non-seeking policy would be indifferent to actions after the event has happened. However, in terms of achieving the objective, both policies will behave exactly the same until the event is triggered, after which the behavior of the policy will no longer matter. For example, if we operate in the first exit scenario, and consider the episode terminated after the goal event is achieved, then we never encounter the scenario when the event occurs in the past.

If we would like to compute the non-seeking policy, we can compute a forward pass which keeps track of the probability that the event has happened:

$$p(t^* \in [1, t] | s_{1:t}, a_{1:t}) = p(e_t = 1 | s_t, a_t) + p(e_t = 0 | s_t, a_t) p(t^* \in [1, t-1] | s_{1:t-1}, a_{1:t-1})$$

We can then use this forward message in conjunction with our backward messages to obtain a non-seeking policy as:

$$p(a_t | s_{1:t}, a_{1:t-1}, t^* \in [1, T]) = \frac{p(a_t | s_{1:t}, a_{1:t}, t^* \in [1, T]) p(a_t | s_t)}{p(a_t | s_{1:t}, a_{1:t-1}, t^* \in [1, T])}$$

Where

$$p(t^* \in [1, T] | s_{1:t}, a_{1:t}) = p(t^* \in [1, t-1] | s_{1:t-1}, a_{1:t-1}) + p(t^* \notin [1, t-1] | s_{1:t-1}, a_{1:t-1}) p(t \in [t, T] | s_t, a_t)$$

$$p(t^* \in [1, T] | s_{1:t}, a_{1:t-1}) = \int_{\mathcal{A}} p(t^* \in [1, T] | s_{1:t}, a_{1:t}) p(a_t | s_t) da_t$$

Note that while the policy is conditioned on all past states and actions, it only depends on them through the forward message, or the cumulative probability that the event has happened.

A.4 Derivations for Variational Objectives

ALL query

We briefly reviewed the variational derivation for standard RL in Section A.2. In this section, we present a more thorough derivation under the events framework and additionally discuss extensions to discounted formulations.

First, we write down the joint trajectory-event distribution, which is simply the product of all factors in the graphical model:

$$p(\tau, e_{1:T} = 1) = p(s_1) \prod_{t=1}^{T-1} p(s_{t+1}|s_t, a_t)p(a_t|s_t)p(e_t = 1|s_t, a_t)$$

We can obtain the optimal trajectory distribution by conditioning and setting the reference policy $p(a_t|s_t)$ as the uniform distribution:

$$p(\tau|e_{1:T} = 1) \propto p(s_1) \prod_{t=1}^{T-1} p(s_{t+1}|s_t, a_t)p(e_t = 1|s_t, a_t)$$

We now perform variational inference with a distribution of the following form, where the dynamics have been forced to equal the true dynamics of the MDP:

$$q_\theta(\tau) = p(s_1) \prod_{t=1}^{T-1} p(s_{t+1}|s_t, a_t)q_\theta(a_t|s_t)$$

Here, $q_\theta(a_t|s_t)$ is the only term that is allowed to vary, and represents the learned policy. When we minimize the KL divergence between q and p, the dynamics terms cancel and we recover the following entropy-regularized policy objective:

$$\begin{aligned} -D_{KL}(q_\theta(\tau)||p(\tau|e_{1:T} = 1)) &= -E_{q_\theta}[\sum_{t=0}^{T} \log q_\theta(a_t|s_t) - \sum_{t=0}^{T} \log p(e_t = 1|s_t, a_t)] + C \\ &= E_{q_\theta}[\sum_{t=0}^{T} \log p(e_t = 1|s_t, a_t) + H(\pi(\cdot|s_t))] + C \end{aligned}$$

The constant C is due to proportionality in the optimal trajectory distribution, and can be ignored in the optimization process.

If we define the empirical returns $\hat{Q}$ as $\hat{Q}(s_t, a_t) = \sum_{t'=t}^{T} \log p(e_{t'} = 1|s_{t'}, a_{t'})$, we can write the returns recursively as:

$$\hat{Q}(s_t, a_t) = \log p(e_t = 1|s_t, a_t) + \hat{Q}(s_{t+1}, a_{t+1})$$

In this discounted case, we consider the case when the dynamics has a $(1 - \gamma)$ chance of transitioning into an absorbing state with reward or $\log p(e_t = 1|s_t, a_t) = 0$. This means we now adjust the recursion as:

$$\hat{Q}(s_t, a_t) = \log p(e_t = 1|s_t, a_t) + \gamma\hat{Q}(s_{t+1}, a_{t+1})$$

ANY query

As with our derivation in the RL case, we begin by writing down our trajectory distribution. Our target trajectory distribution is going to be $p(\tau|t^* \in [1,T])$, which are trajectories where the event happens at least once.

First, we can use Bayes' rule to obtain:

$$\log p(\tau|t^* \in [1,T]) = \log p(t^* \in [1,T]|\tau) + \log p(\tau) - \log p(t^* \in [1,T])$$

The last term is a proportionality constant with respect to the trajectories. The second term is the trajectory distribution induced by the reference policy. The first term can be simplified further.

Note that the probability that the event first happens at t^* is $p(t^* = t|\tau) = p(e_t = 1|s_t, a_t) \prod_{t'=1}^{t-1} p(e_{t'} = 0|s_{t'}, a_{t'})$ (i.e. the event happens at t^* but not before). Now we can write:

$$p(t^* \in [1,T]|\tau) = \sum_{t=1}^{T} p(t^* = t|\tau) = \sum_{t=1}^{T} p(e_t = 1|s_t, a_t) \prod_{t'=1}^{t-1} p(e_{t'} = 0|s_{t'}, a_{t'})$$

To write down a recursion, we now define the quantity $\hat{\beta}(s_{t:T}, a_{t:T}) = p(t^* \in [t,T]|s_{t:T}, a_{t:T})$. We can now express the above term recursively as:

$$\begin{aligned}\sum_{t=1}^{T} p(e_t = 1|s_t, a_t) \prod_{t'=1}^{t-1} p(e_{t'} = 0|s_{t'}, a_{t'}) &= p(e_1 = 1|s_1, a_1) + p(e_1 = 0|s_1, a_1)\hat{\beta}(s_{2:T}, a_{2:T}) \\ &= \hat{\beta}(s_{1:T}, a_{1:T})\end{aligned}$$

Thus, if we define our empirical Q-function $\hat{Q}(s_t, a_t) = \log \hat{\beta}(s_{1:T}, a_{1:T})$, our recursion now becomes:

$$\hat{Q}(s_t, a_t) = \log(p(e_t = 1|s_t, a_t) + p(e_t = 0|s_t, a_t)e^{\hat{Q}(s_{t+1}, a_{t+1})})$$

Using the same variational distribution $q_\theta(\tau) = p(s_1) \prod_{t=1}^{T} p(s_{t+1}|s_t, a_t) q_\theta(a_t|s_t)$ as before, we can write our optimization objective as:

$$-D_{KL}(q_\theta(\tau)||p(\tau|t^* \in [1,T])) = E_q[\hat{Q}(s_{1:T}, a_{1:T}) - \sum_{t=1}^{T} \log q_\theta(a_t|s_t)] + C$$

Where the constant C absorbs terms from the reference policy $p(a_t|s_t)$ which we set to uniform, and the proportionality constant $\log p(t^* \in [1,T])$.

To achieve a discounted objective case, we consider the case when the dynamics has a $(1-\gamma)$ chance of transitioning into an absorbing state where the event can never happen $p(e_t = 1|s_t, a_t) = 0$. Note that this is different from the all query. This means we now adjust the recursion as:

$$\hat{Q}(s_t, a_t) = \gamma \log \left(p(e_t = 1|s_t, a_t) + p(e_t = 0|s_t, a_t)e^{\hat{Q}(s_{t+1}, a_{t+1})}\right) + (1-\gamma) \log p(e_t = 1|s_t, a_t)$$

A.5 Policy Gradients for Events

Because the ALL query is mathematically identical to standard RL, we do not derive the policy gradient estimator here.

For the ANY query, we consider the objective

$$J(\pi) = E_\pi[\hat{Q}(s_{1:T}, a_{1:T}) - \sum_{t=1}^{T} \log \pi(a_t|s_t)]$$

. For simplicity we disregard the entropy term as that portion remains unchanged from standard RL.

Applying logarithmic differentiation, and simplifying, we can obtain the gradient estimator.

$$E_\pi[\sum_{t=1}^{T} \nabla \log \pi(a_t|s_t)\hat{Q}(s_{1:T}, a_{1:T})]$$

The next step is that we wish to only consider future returns, i.e. we wish to replace $\hat{Q}(s_{1:T}, a_{1:T})$ with $\hat{Q}(s_{t:T}, a_{t:T})$. First, note that before the event happens before t, then $\hat{Q}(s_{1:T}, a_{1:T})$ and $\hat{Q}(s_{t:T}, a_{t:T})$ are identical, but if t is after then event then the returns estimator should be 0. Thus, we need to keep track of the cumulative probability that an event occurs and rewrite the estimator as:

$$E_\pi[\sum_{t=1}^{T} \nabla \log \pi(a_t|s_t)p(t \leq t^*|s_{1:t}, a_{1:t})\hat{Q}(s_{t:T}, a_{t:T})]$$

To obtain an estimator that only depends on future returns, we assume that the event always happens in the future and set $p(t \leq t^*|s_{1:t}, a_{1:t}) = 1$. The justification is that after the event happens, we no longer care about the behavior of the policy. We also discuss this point towards the end of Appendix A.3 - it is exactly correct in a first-exit scenario where the episode terminates upon triggering the event so that the case when the event happened in the past is impossible, and otherwise still provides reasonable behavior.

A.6 Variational Inverse Control with Events (VICE)

In this section, we derive the update rules for performing inverse event-based control. We present one derivation for each query type, and follow the following steps. For each query, we assume our dataset of states and actions $(s, a) \sim p_{data}(s, a|\text{evidence})$ is generated given evidence corresponding to the query type. We then wish to learn the parameters of the graphical model $p_\theta(s, a|\text{evidence})$, where θ parametrizes the event probability arc $p_\theta(e_t = 1|s_t, a_t)$.

We can train this model with the maximum likelihood objective:

$$\mathcal{L}(\theta) = -E_{p_{data}}[\log p_\theta(s, a|\text{evidence})]$$

which can be trained by optimizing the cross-entropy loss of the discriminator.

We also train a sampling distribution $q(s, a)$ to match $p_\theta(s, a|\text{evidence})$. This is to approximate the gradient of this model as we typically do not know the partition function of $p_\theta(e_t = 1|s_t, a_t)$. To that end, we use variational inference to obtain a sampling distribution q by minimizing $D(q(s, a)||p_\theta(s, a|\text{evidence}))$. Rather than directly minimizing this KL-divergence, we instead minimize the trajectory divergence $D(q(\tau)||p_\theta(\tau|\text{evidence}))$, which we show is an upper bound to the original KL-divergence in each case. The resulting objective $D(q(\tau)||p_\theta(\tau|\text{evidence}))$ corresponds to the inference procedures derived in Appendix A.4.

AT query VICE

In the AT query, we assume we observe states and actions where the event occurred at a specific timestep, denoted as t. We assume our data comes from the distribution $p_{data}(s_t, a_t|e_t = 1) \propto p_{data}(e_t = 1|s_t, a_t)p(s_t, a_t)$, and likewise we fit a model of the form $p_\theta(s_t, a_t|e_t = 1) \propto p_\theta(e_t = 1|s_t, a_t)p(s_t, a_t)$, and a variational sampling distribution of the form $q(s_t, a_t)$.

The maximum likelihood objective is:

$$\mathcal{L}(\theta) = -E_{p_{data}} \left[\log p_\theta(s_t, a_t|e_t = 1)\right]$$

which as the corresponding gradient:

$$\nabla_\theta \mathcal{L}(\theta) = E_{p_{data}} \left[p_\theta(e_t = 1|s_t, a_t)\right] - E_{p_\theta} \left[p_\theta(e_t = 1|s_t, a_t)\right]$$

We now derive the objective for training our sampler $q(s_t, a_t)$ so that it matches $p_\theta(s_t, a_t)$. By the chain rule for KL divergence, we have the upper-bound $D_{KL}(q(s_t, a_t)||p_\theta(s_t, a_t|e_t = 1)) \leq D_{KL}(q(\tau)||p_\theta(\tau|e_t = 1))$. After obtaining $q(\tau)$, we can sample by executing full trajectories and picking the states and actions that correspond to timestep t.

ALL query VICE

In the ALL query, we assume our data comes from the distribution of states and actions along trajectories where the event happens at all timesteps (averaged over timesteps):

$$p_{data}(s, a|e_{1:T} = 1) = \frac{1}{T} \sum_t p_{data}(s_t, a_t|e_{1:T} = 1)$$

This corresponds to matching the occupancy measure of a policy, which is equivalent to inverse reinforcement learning as shown by [60].

We can upper-bound the KL-divergence of interest between the sampler and the model with a KL-divergence on trajectories as:

$$\begin{aligned}
&D(\frac{1}{T}\sum_t q(s_t, a_t)||\frac{1}{T}\sum_t p_\theta(s_t, a_t|e_{1:T} = 1)) \\
&\leq \frac{1}{T}\sum_t D(q(s_t, a_t)||p_\theta(s_t, a_t|e_{1:T} = 1)) \\
&\leq \frac{1}{T}\sum_t D(q(\tau)||p_\theta(\tau|e_{1:T} = 1)) \\
&= D(q(\tau)||p_\theta(\tau|e_{1:T} = 1))
\end{aligned}$$

The first inequality comes from the log-sum inequality, and the second inequality comes from the chain rule for KL divergence.

ANY query VICE

In the ANY query formulation, we assume our data comes from the distribution of states and actions at the first timestep an event happens.

$$p_{data}(s_{t^*}, a_{t^*}|t^* \in [1, T]) = p_{data}(t^* \in [1, T]|s, a)\rho(s, a) = [\sum_t p_{data}(t^* = t|S_t = s, A_t = a)]\rho(s, a)$$

We have substituted $p_{data}(t^* \in [1, T]|s, a) = \sum_t p_{data}(t^* = t|S_t = s, A_t = a)$ because each $t^* = t$ is a disjoint event. We can further simplify this as $\sum_t p_{data}(t^* = t|S_t = s, A_t = a) = p_{data}(e_1 = 1|S_1 = s, A_1 = s) + p_{data}(e_1 = 0|S_1 = s, A_1 = s)[p_{data}(e_2 = 1|S_2 = s, A_2 = s) + ...] = Q_{data}(s, a)$. Thus, assuming our model p_θ takes the same form, we obtain the maximum likelihood objective:

$$\mathcal{L}(\theta) = -E_{p_{data}}[\log p_\theta(s, a|t^* \in [1, T])]$$

and the corresponding gradient

$$\mathcal{L}(\theta) = -E_{p_{data}}[\log p_\theta(s, a|t^* \in [1, T])] - E_{p_\theta}[\log p_\theta(s, a|t^* \in [1, T])]$$

Where $p_\theta(s, a|t^* \in [1, T]) = p_\theta(e_1 = 1|S_1 = s, A_1 = s) + p_\theta(e_1 = 0|S_1 = s, A_1 = s)[p_\theta(e_2 = 1|S_2 = s, A_2 = s) + ...]$.

Lemma A.6.1. *Let $\boldsymbol{X} = (x_1, x_2, ...)$, $\boldsymbol{Y} = (y_1, y_2, ...)$. Let $\bar{\mu}$ denote a set of weights which sum to one, and denote $\bar{p}(\boldsymbol{X}) = E_{\bar{\mu}}[p_i(x_i)]$, and $\bar{p}(\boldsymbol{X}, \boldsymbol{Y}) = E_{\bar{\mu}}[p_i(x_i, y_i)]$ denote convex combinations of the individual distributions p_i. Then,*

$$D(\bar{p}(\boldsymbol{X}, \boldsymbol{Y})||\bar{q}(\boldsymbol{X}, \boldsymbol{Y})) \geq D(\bar{p}(\boldsymbol{X})||\bar{q}(\boldsymbol{X}))$$

Proof. This statement directly follows from the chain rule for KL divergences, which implies:

$$D(\bar{p}(\mathbf{X}, \mathbf{Y})||\bar{q}(\mathbf{X}, \mathbf{Y})) = D(\bar{p}(\mathbf{X})||\bar{q}(\mathbf{X})) + D(\bar{p}(\mathbf{X}|\mathbf{Y})||\bar{q}(\mathbf{X}|\mathbf{Y})) \geq D(\bar{p}(\mathbf{X})||\bar{q}(\mathbf{X}))$$

□

Now, we can apply Lemma A.6.1 to derive the upper-bound:

$$\begin{aligned}
&D(\sum_t q(s_t, a_t)p(t^* = t)||\sum_t p_\theta(s_t, a_t|t^* = t)p(t^* = t)) \\
&\leq D(\sum_t q(\tau)p(t^* = t)||\sum_t p_\theta(\tau|t^* = t)p(t^* = t)) \\
&= D(q(\tau)||p_\theta(\tau|t^* \in [1, T]))
\end{aligned}$$

We can obtain samples from $q(s_{t^*}, a_{t^*})$ by executing full trajectories and using the first state when an event is triggered.

A.7 Experiments

Experimental details for prespecified events

On the *Lobber* task, we use a diagonal gaussian policy where the mean is parametrized by a 32x32 neural network. We use a TRPO batch size of 40000 and train for 1000 iterations.

On the *HalfCheetah* task, we use a diagonal gaussian policy where the mean is parametrized by a 32x32 neural network. We use a TRPO batch size of 10000 and train for 1000 iterations.

Experimental details for learning event probabilities

We evaluate the performance of VICE in learning event probabilities on the *Ant*,*Maze*, and *Pusher* tasks, providing comparisons to classifier-based methods. Although the binary indicator baseline is not comparable to VICE (since it observes the event while the other methods do not), we present comparisons to provide a general idea of the difficulty of the task. All experiments are run with five random seeds, and mean results are presented.

We use Gaussian policies, where the mean is parametrized by a neural network, and the covariance a learned diagonal matrix. The event distribution is represented by a neural network as well. Further hyperparameters are presented in Table A.1.

On the *Ant* task, both the policy mean network and event distribution network have two hidden layers with 200 units and ReLu activations.

On the *Maze* task, the mean network has two convolutional layers, with filter size 5×5, followed by two fully connected layers with 32 units each with ReLu activations. The event distribution is represented using a convolutional neural network with two convolutional layers with 5×5 filters, and a final fully-connected layer with 16 units.

On the *Pusher* task, the policy is represented by a convolutional neural network with three convolutional layers, with a stride of 2 in the first layer, and a stride of 1 in the subsequent layers. We use a filter size of 3x3 in all the layers, and the number of filters are 64, 32 and 16. In line with prior work [30], we pre-train the convolutional layers using an auto-encoder loss on data collected from random policies. The fully connected part of the neural network consists of two layers, each having 200 units and ReLu activations to represent the policy. The event distribution is also represented by the same architecture.

	Ant	Maze	Pusher
Batch Size	10000	5000	10000
Iterations	1000	150	1000
Discount	0.99	0.99	0.99
Entropy	0.1	0.1	0.01
# Demonstrations	500	1000	10000

Table A.1: Hyperparameters used for VICE on the Ant,Maze, and Pusher tasks

Appendix B

End-to-End Robotic Reinforcement Learning without Reward Engineering

B.1 Experimental details

Simulated tasks

Detailed descriptions of our simulated tasks are provided here.

Visual Pusher The goal of this task is to push a mug onto a coaster. The robot end effector is constrained to move in a 2D XY plane, and the initial position of the mug is randomized over a 20cm x 15cm region. A success is when the final position of the mug is within 3cm of the goal.

Visual Door The goal of this task is to open a door by 45 degrees. Initially, the door is either completely closed (with probability 0.5), or open up to an angle of 15 degrees. The robot end effector is equipped with a hook, and the robot needs needs to trap the handle of the door in its hook and pull it open. The robot end effector is allowed to move in the full 3D XYZ space.

Visual Picker The goal of this task is to pick up a tennis ball from a table. The robot end effector is free to move in the 3D XYZ space, and the initial position of the ball is randomized over a 10cm x 10cm square region over the table. Along with the robot end-effector, we also control the opening and closing of a parallel jaw gripper. The robot needs to pick up the ball and move it to a fixed location that is 20 cm above the table. A success is when the final position of the ball is within 3cm of the goal.

Hyperparameter details

Parameters for Soft Actor-Critic We did not tune any hyperparameter of the soft actor-critic algorithm, and used default values provided in the authors' open-source implementation [50]. These value are a learning rate of 3e-4 on the Adam optimizer [77], a batch size of 256, $\tau = 0.05$, a discount factor of 0.99, a target update frequency of 1, and a target entropy of $\frac{1}{d_a}$, where d_a is the action dimension for the environment.

Parameters for Discriminator Training We train the discriminator also using Adam with a learning rate of 3e-4 and a batch size of 256. We take N update steps on the discriminator at the end of every epoch of RL updates (where each epoch consists of 1000 timesteps). We sweeped over the value of N individually for each of our methods and all the simulated tasks. We sweeped over N=[5,10,100] and found that N=10 works well for all task/method combinations, except the Visual Door Opening task with VICE-RAQ, where N=5 performs slightly better. We also sweeped over using mixup (with parameter $\alpha = 1$) and not using mixup, and found that mixup greatly helped for the Visual Door Opening and the Visual Picker task, but slightly decreased the performance for the Visual Pushing task. After the hyperparameter sweep, fresh runs were made for all tasks and methods for reporting the final results.

Detailed learning curves

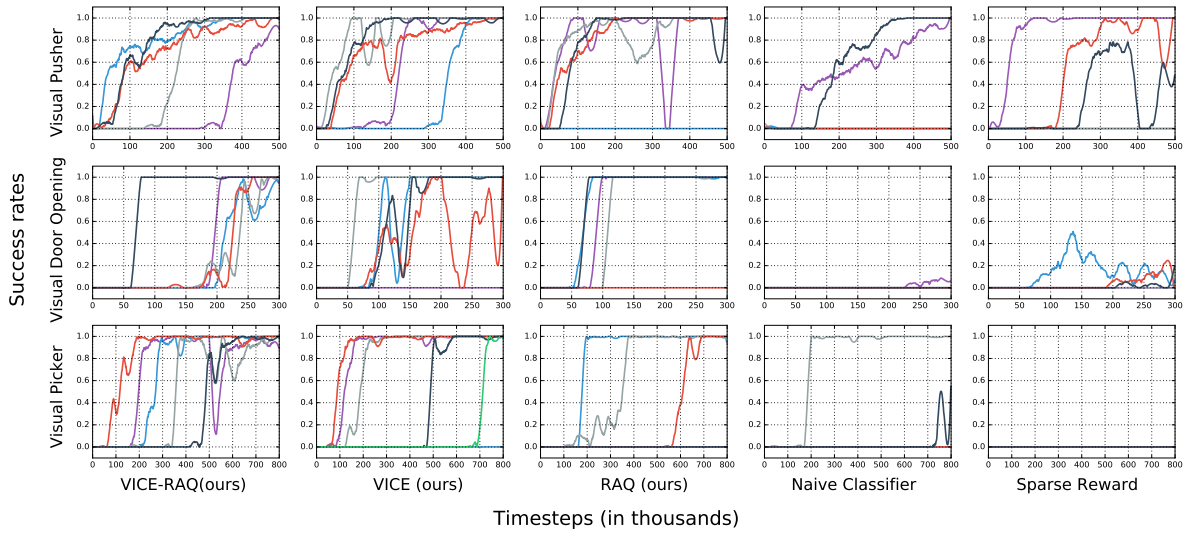


Figure B.1: This plot shows the results on all of our simulated tasks for all of our methods and baselines. Each plots shows results from each of the five random seeds run for that task and method.

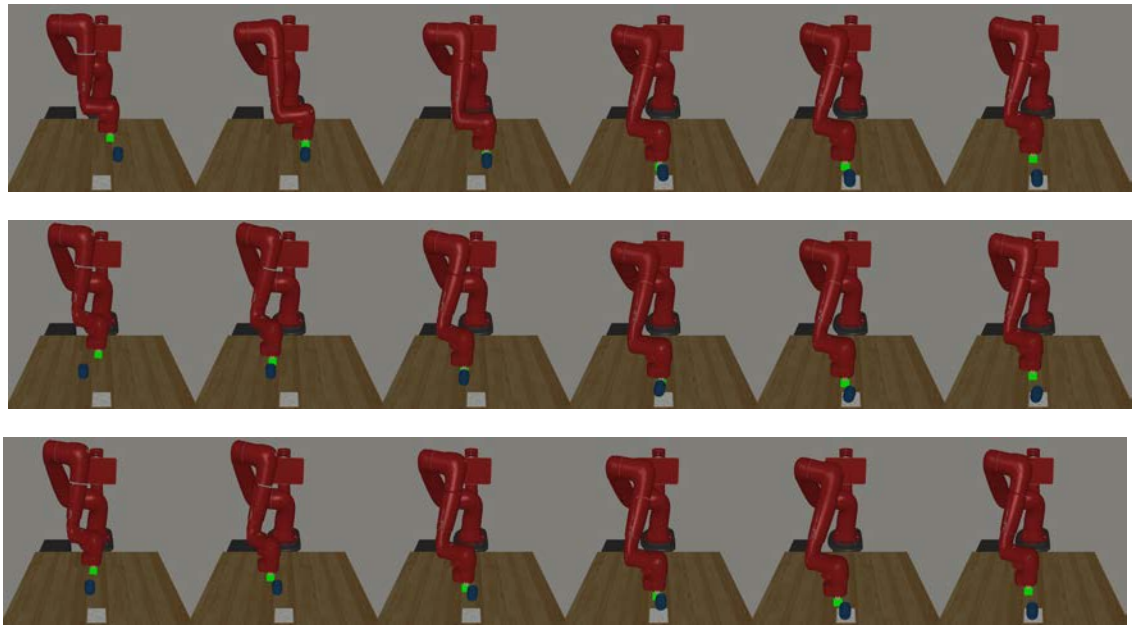


Figure B.2: Example evaluation rollouts for the simulated *Visual Pusher* task from a policy learned using VICE-RAQ.

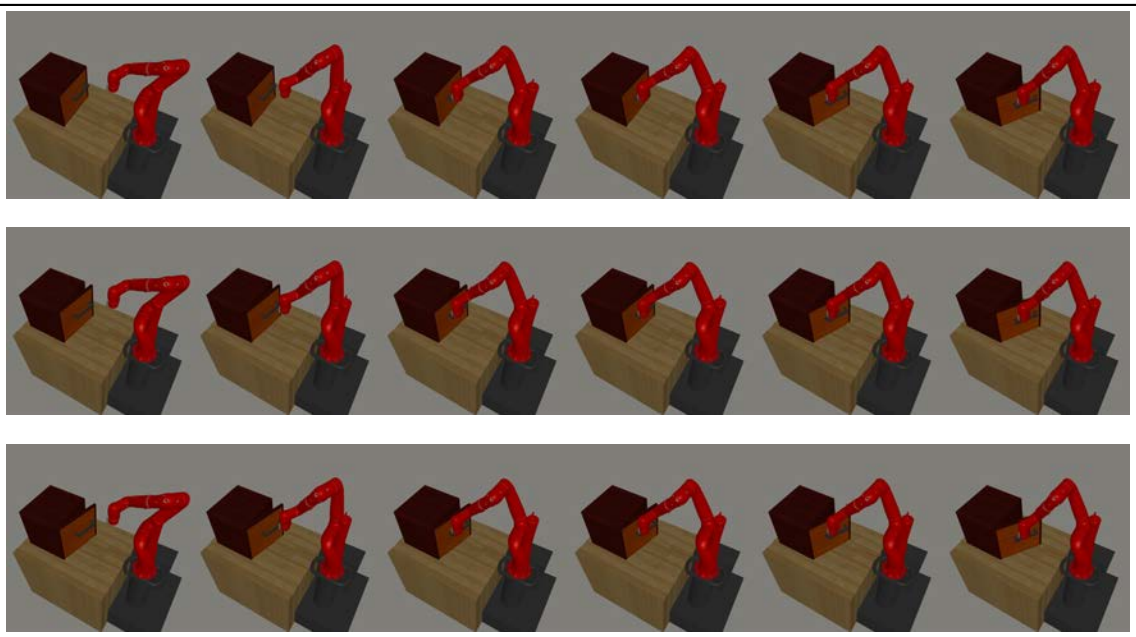


Figure B.3: Example evaluation rollouts for the simulated *Visual Door Opening* task from a policy learned using VICE-RAQ.

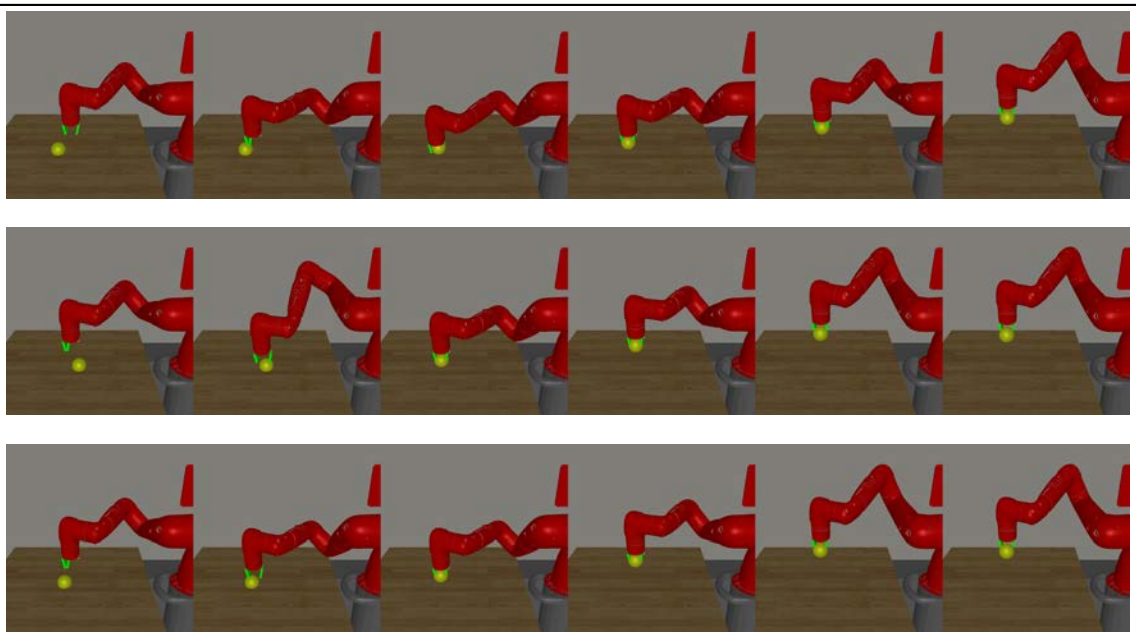


Figure B.4: Example evaluation rollouts for the simulated *Visual Picker* task from a policy learned using VICE-RAQ.

Appendix C

Parrot: Data-Driven Behavioral Priors for Reinforcement Learning

C.1 Algorithm

Algorithm 4 RL with Behavioral Priors

Input: Dataset $\mathcal{D}$ of state-action pairs (s, a) from previous tasks, new task $M^\star$
Learn f_ϕ by maximizing the likelihood term in Equation 4.2
for step k in {1, . . . , N} **do**
    $s \leftarrow$ current observation
    Sample $z \sim \pi_\theta(z|s)$
    $a \leftarrow f_\phi(z; s)$
    $s', r \leftarrow$ Execute a in $M^\star$
    Update $\pi_\theta(z|s)$ with (s, z, s', r)
end for
Return: Policy $\pi_\theta(z|s)$ for task $M^\star$.

C.2 Implementation Details and Hyperparameter Tuning

We now provide details of the neural network architectures and other hyperparameters used in our experiments.

Behavioral prior. We use a conditional real NVP with four affine coupling layers as our behavioral prior. The architecture for a single coupling layer is shown in Figure C.1. We use a learning rate of $1e{-}4$ and the Adam [77] optimizer to train the behavioral prior for 500K steps.

TrajVAE. For this comparison, we use the same architecture as Ghadirzadeh et al [43]. The decoder consists of three fully connected layers with 128, 256, and 512 units respectively. BatchNorm [66] and ReLU nonlinearity are applied after each layer. The encoder is symmetric:

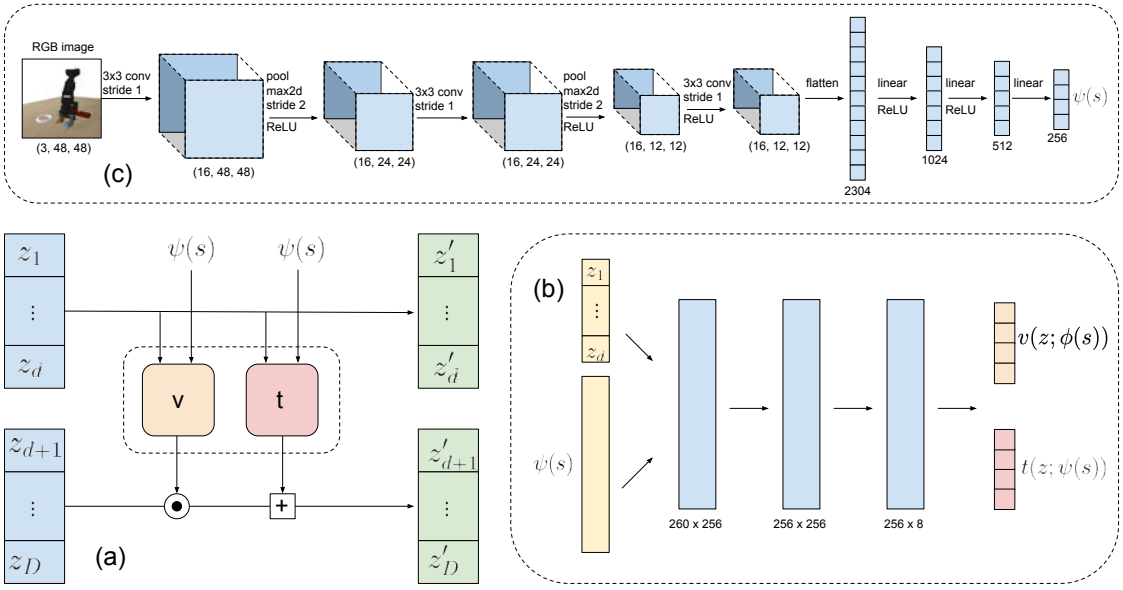


Figure C.1: **Coupling layer architecture.** A computation graph for a single affine coupling layer is shown in (a). Given an input noise z, the coupling layers transform it into z' through the following operations: $z'_{1:d} = z_{1:d}$ and $z'_{d+1:D} = z_{d+1:D} \odot \exp(v(z_{1:d}; \phi(s))) + t(z_{1:d}; \phi(s))$, where the v, t and ψ are functions implemented using neural networks whose architectures are shown in (b) and (c). Since v and t have the same input, they are implemented using a single fully connected neural network (shown in (b)), and the output of this network is split into two. The image encoder, $\psi(s)$ is implemented using a convolutional neural network with parameters shown in (c).

512, 256, and 128 layers, respectively. The size of the latent space is 8 (same as the behavioral prior). We sweep the following values for the β parameter [57]: 0.1, 0.01, 0.005, 0.001, 0.0005, and find 0.001 to be optimal. We initialize β to zero at the start of training, and anneal it to the target β value using a logistic function, achieving half of the target value in 25K steps. We use a learning rate of $1e{-}4$ and the Adam [77] optimizer to train this model for 500K steps.

HIRL. This comparison is implemented using a conditional variational autoencoder, and uses an architecture that is similar to the one used by the **TrajVAE**, but with two differences: since this comparison uses image conditioning, we use the same convolutional network ψ as the behavioral prior to encode the image (shown in Figure C.1), and pass it as conditioning information to both the encoder and decoder networks. Second, instead of modeling the entire trajectory in a single forward pass, it instead models individual actions, allowing the high-level policy to perform closed-loop control, similar to the behavioral prior model. We sweep the following values for the β parameter: 0.1, 0.01, 0.005, 0.001, 0.0005, and find 0.001 to be optimal. We found the annealing process to be essential for obtaining good RL performance using this method. We use a learning rate of $1e{-}4$ and the Adam [77] optimizer

to train this model for 500K steps.

Behavior cloning (BC). We implement behavior cloning via maximum likelihood with a Gaussian policy (and entropy regularization [49]). For both behavior cloning and RL with SAC, we used the same policy network architecture as shown in Figure C.2. We train this model for 2M steps, using Adam with a learning rate of $3e^{-4}$.

VAE-features. For this comparison, we use the standard VAE architecture used for CIFAR-10 experiments [118]. The encoder consists of two strided convolutional layers (stride 2, window size 4×4), which is followed by two residual 3×3 blocks, all of which have 256 hidden units. Each residual block is implemented as ReLU, 3x3 conv, ReLU, 1x1 conv. The decoder is symmetric to the encoder. We train this model for 1.5M steps, using Adam with a learning rate of $1e^{-3}$ and a batch size of 128.

Soft Actor Critic (SAC). We use the soft actor critic method [50] as our RL algorithm, with the hyperparameters shown in Table 1. We use the same hyperparameters for all of our RL experiments (our method, HIRL, TrajRL, BC+SAC, SAC).

Table C.1: Hyperparameters for soft-actor critic (SAC)

Hyperparameter	value used
Target network update period	1000 steps
discount factor γ	0.99
policy learning rate	$3e^{-4}$
Q-function learning rate	$3e^{-4}$
reward scale	1.0
automatic entropy tuning	enabled
number of update steps per env step	1

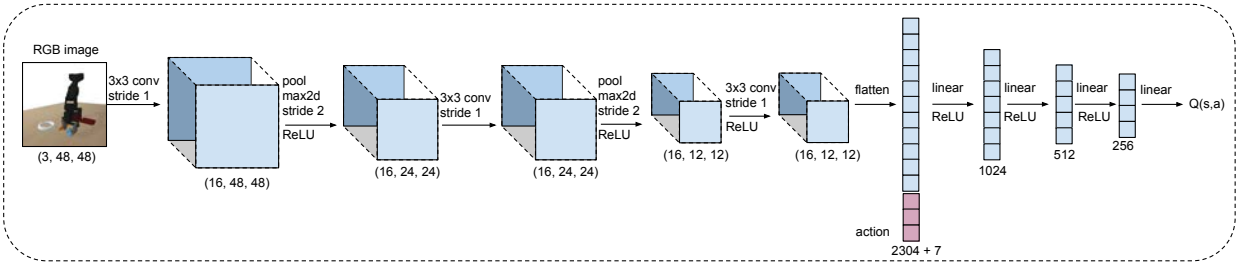


Figure C.2: **Policy and Q-function network architectures.** We use a convolutional neural network to represent the Q-function for SAC, shown in this figure. The policy network is identical, except it does not take in an action as an input and outputs a 7D action instead of a scalar Q-value.

C.3 Experimental setup

Tasks

We provided a visual depiction of 4 of our 8 evaluations tasks in Figure 4.3, and the remaining tasks are shown here in Figure C.3.

Figure C.3: In the first row, the objective is to grasp a can and lift it above a certain height. Rows two and three are similar, except the objective is to grasp a vase and a baseball cap, respectively. The final row depicts a task where the goal is to pick the baseball cap and place it on the marble cube.

Data collection

We collected our dataset using scripted policies detailed in Algorithms 7 and 8.

Algorithm 5 Scripted Grasping

```
threshold ← 0.02
numTimesteps ← 25
targetPoint ← object position
for t in (0, numTimesteps) do
    eePos ← end effector position
    targetEEDist ← distance(targetPoint, eePos)
    if targetEEDist > threshold then
        action ← targetPoint − eePos
    else if gripperOpened then
        action ← close gripper
    else if object not raised high enough then
        action ← lift upward
    else
        action ← 0
    end if
    noise ∼ 𝒩(0, 0.1)
    action ← action + noise
    s′ ← env.step(action)
end for
```

Algorithm 6 Scripted Pick and Place

```
threshold ← 0.02
numTimesteps ← 25
placeAttempted ← False
dropPos ← point above container
for t in (0, numTimesteps) do
    eePos ← end effector position
    objectDropDist ← distance(eePos, dropPos)
    if placeAttempted then
        action ← 0
    else if object not grasped AND objectDropDist
    > threshold then
        Execute grasp using Algorithm 7
    else if objectDropDist > threshold then
        action ← dropPos − eePos
    else
        action ← open gripper
        placeAttempted ← True
    else
        action ← 0
    end if
    noise ∼ 𝒩(0, 0.1)
    action ← action + noise
    s′ ← env.step(action)
end for
```

Simulation Objects

To collect data in diverse environments, we used 3D object models from the ShapeNet dataset [10] and the PyBullet [14] object libraries.

Figure C.4: **Train objects**.

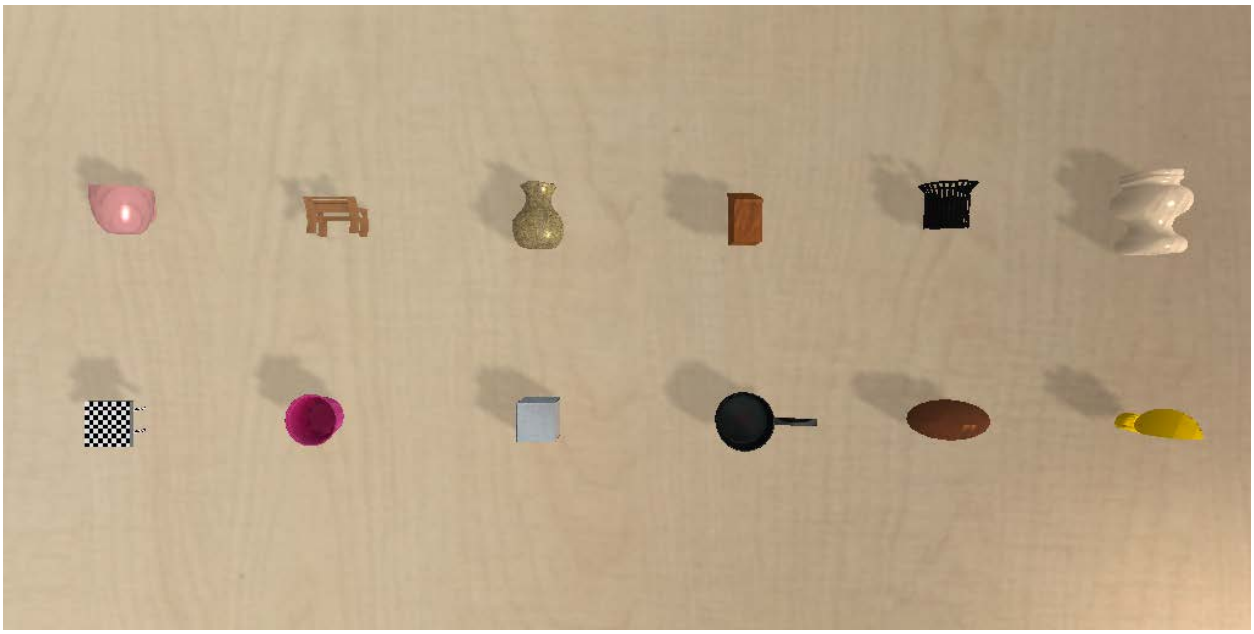


Figure C.5: **Test objects**

Appendix D

COG: Connecting New Skills to Past Experience with Offline Reinforcement Learning

D.1 Experimental Setup Details

Both our real and simulated environments use the following 8-dimensional control scheme:

`[x,y,z,alpha,beta,gamma,gripperOpen,moveToNeutral]`

where the `x,y,z` dimensions command changes in the end-effector's position in 3D space, `alpha,beta,gamma` command changes in the end-effector's orientation, `gripperOpen` is a continuous value from $[-1, 1]$ that triggers the gripper to close completely when less than -0.5 and open completely when greater than 0.5, and `moveToNeutral` is also a continuous value from $[-1, 1]$ that triggers the robot to move to its starting joint position when greater than 0.5. The code for our environments can be found on our project website: `https://sites.google.com/view/cog-rl`.

Data Collection Policies

We describe our scripted data collection policies in this section. More details can be found in Algorithms 1-3.

Scripted grasping. Our scripted grasping policy is supplied with the object's (approximate) coordinates. In simulation, this information is readily available, while in the real world we use background subtraction and a calibrated monocular camera to approximately localize the object. Note that this information does not need to be perfect, as we add a significant amount of noise to the scripted policy's action at each timestep. After the object has been localized, the scripted policy takes actions that move the gripper toward the object (i.e

action ← object_position − gripper_position). Once the gripper is within some pre-specified distance of the object, it closes the gripper (which is a discrete action). Note that this distance threshold is also randomized – sampled from a Gaussian distribution with a mean of 0.04 and a standard deviation of 0.01 (in meters). It then raised the object until it is above a certain height threshold. For the simulated pick and place environment, the scripted policy for grasping obtains a success rate of 50%, while the success rate is 70% for the drawer environment. For the real world drawer environment, the scripted success rate is 30%.

Scripted pick and place. The pick part of the pick and place scripted policy is identical to the grasping policy described above. After the grasp has been attempted, the scripted policy uniformly randomly selects a point in the workspace to place the object on, and then takes actions to move the gripper above that point. Once within a pre-specified (and randomized) distance to that point, it opens the gripper. The policy is biased to sample more drop points that lie inside the tray to ensure we see enough successful pick and place attempts. Once the object has been dropped, the robot returns to its starting configuration (using the moveToNeutral action).

Scripted place. This policy is used in scenes where the robot is already holding the object at the start of the episode. The placing policy is identical to the place component of the pick and place policy described above.

Drawer opening and closing. The scripted drawer opening policy moves the gripper to grasp the drawer handle, then pulls on it to open the drawer. The drawer closing policy is similar, except it pushes on the drawer instead of pulling it. Even if the correct action for a particular task might involve only opening the drawer, we collect data (without reward labels) that involves both opening and closing the drawer during prior data collection. Gaussian noise is added to the policy actions at every timestep. After the opening/closing is completed, the robot returns to its starting configuration.

Ending scripted trajectories with return to starting configuration We ended the scripted trajectories with a return to the robot's starting configuration. We believe that this return to starting configuration increases the state-distribution overlap of the various datasets collected from scripted policies, making it possible to stitch together relevant trajectories from the prior dataset to extend the skill learned for the downstream task.

Algorithm 7 Scripted Grasping

threshold $\sim \mathcal{N}(0.04, 0.01)$
numTimesteps $\leftarrow 30$
for t **in** (0, numTimesteps) **do**
    objPos $\leftarrow$ object position
    eePos $\leftarrow$ end effector position
    objGripperDist $\leftarrow$ distance(objPos, eePos)
    if objGripperDist $>$ threshold **then**
        action $\leftarrow$ objPos $-$ eePos
    else if gripperOpened **then**
        action $\leftarrow$ close gripper
    else if object not raised high enough **then**
        action $\leftarrow$ lift upward
    else
        action $\leftarrow 0$
    end if
    noise $\sim \mathcal{N}(0, 0.2)$
    action $\leftarrow$ action + noise
    $(s, r, s') \leftarrow$ env.step(action)
end for

Algorithm 8 Scripted Pick and Place

threshold, dropDistThreshold $\sim \mathcal{N}(0.04, 0.01)$
numTimesteps $\leftarrow 30$
for t **in** (0, numTimesteps) **do**
    eePos $\leftarrow$ end effector position
    dropPos $\leftarrow \begin{cases} \text{point above box} & \text{w/ prob. } 0.5 \\ \text{point outside box} & \text{w/ prob. } 0.5 \end{cases}$
    objectDropDist $\leftarrow$ distance(eePos, dropPos)
    if object not grasped **AND** objectDropDist $>$ dropDistThreshold **then**
        Execute grasp using Algorithm 7
    else if objectDropDist $>$ boxDistThreshold **then**
        action $\leftarrow$ dropPos $-$ eePos
        action $\leftarrow$ lift upward
    else if object not dropped **then**
        action $\leftarrow$ open gripper
    else
        action $\leftarrow 0$
    end if
    noise $\sim \mathcal{N}(0, 0.2)$
    action $\leftarrow$ action + noise
    $(s, r, s') \leftarrow$ env.step(action)
end for

Algorithm 9 Scripted Drawer Opening/Closing

threshold $\sim \mathcal{N}(0.04, 0.01)$
numTimesteps $\leftarrow 30$
for t **in** (0, numTimesteps) **do**
    handlePos $\leftarrow$ handle center position
    eePos $\leftarrow$ end effector position
    targetGripperDist $\leftarrow$ dist(targetPos, eePos)
    if targetGripperDist $>$ threshold **AND** not drawerOpened **then**
        action $\leftarrow$ targetPos $-$ eePos
    else if not drawerOpened (or closed) **then**
        action $\leftarrow$ move left to open drawer, or right to close drawer
    else if gripper not above drawer **then**
        action $\leftarrow$ lift upward
    else
        action $\leftarrow$ `moveToNeutral`
        End scripted trajectory
    end if
    noise $\sim \mathcal{N}(0, 0.2)$
    action $\leftarrow$ action + noise
    $(s, r, s') \leftarrow$ env.step(action)
end for

Neural Network Architectures

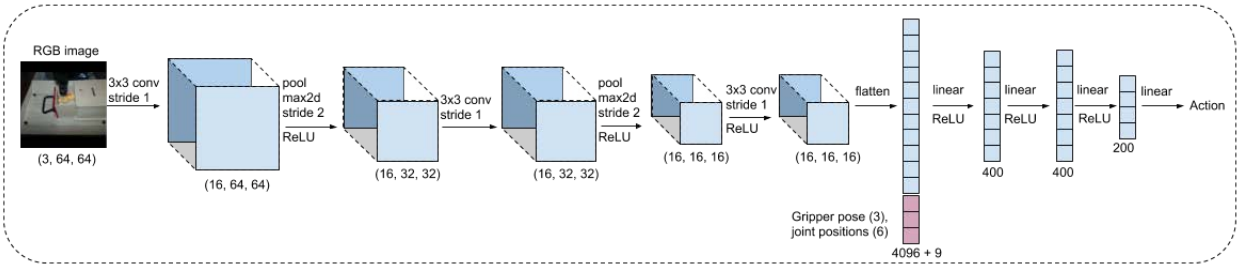


Figure D.1: **Neural network architecture for real robot experiments.** Here we show the architecture for the policy network for real robot experiments. The Q-function architecture is identical, except it also has the action as an input that is passed in after the flattening step. We map high dimensional image observations to low level robot commands, i.e. desired position of the end-effector, and gripper opening/closing.

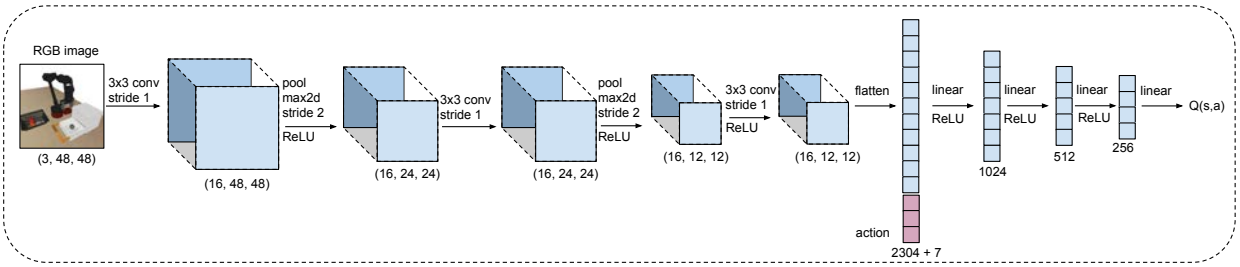


Figure D.2: **Neural network architecture for simulated experiments**. Here we show the architecture for the Q-function in our simulated experiments. The policy architecture is identical, except no action is passed to the network. Note that we omit the information about the gripper position and orientation, since including this information did not seem to make a difference in our simulated experiments.

Figures D.1 and D.2 show the neural network architectures used in our real world and simulated experiments, respectively. We experimented with several different architectures (varying the number of convolutional layers from 2 to 4, and varying the number of filters in each layer from 4 to 16), and found these two architectures to perform the best.

Hyperparameters for Reinforcement Learning

We used the conservative Q-learning (CQL) [86] algorithm in our method. Source code can be found on our project website: `https://sites.google.com/view/cog-rl` We now present the hyperparameters used by our method below:

- **Discount factor**: 0.99 (identical to SAC, CQL),
- **Learning rates**: Q-function: 3e-4, Policy: 3e-5 (identical to CQL),
- **Batch size**: 256 (identical to SAC, CQL),
- **Target network update rate**: 0.005 (identical to SAC, CQL),
- **Ratio of policy to Q-function updates**: 1:1 (identical to SAC, CQL),
- **Number of Q-functions**: 2 Q-functions, $\min(Q_1, Q_2)$ used for Q-function backup and policy update (identical to SAC, CQL),
- **Automatic entropy tuning**: True, with target entropy set to $-\log|\mathcal{A}|$ (identical to SAC, CQL),
- **CQL version**: CQL($\mathcal{H}$) (note that this doesn't contain an additional $-\alpha \log \pi(\mathbf{a}|\mathbf{s})$ term in the Q-function backup),
- α **in CQL**: 1.0 (we used the non-Lagrange version of CQL($\mathcal{H}$)),
- **Number of negative samples used for estimating logsumexp:** 1 (instead of the default of 10 used in CQL; reduces training wall-clock time substantially when learning from image observations)
- **Initial BC warmstart period**: 10K gradient steps
- **Evaluation maximum trajectory length**: 80 timesteps for simulated drawer environment, 40 timesteps for simulated pick and place. For real world drawer environment, this value is equal to 35 timesteps.

D.2 Comparison to BC + SAC for online fine-tuning

We compared our CQL fine-tuning results to fine-tuning a behavior-cloned policy with SAC, and observed that fine-tuning with CQL yields substantially better results. The comparison between between CQL fine-tuning, and this BC+SAC baselines is shown in Figure D.3 for the grasping from a drawer task (see Figure 5.4), for the initial condition where the drawer starts out closed. We see that the initial SAC performance is low, which is partly due to the low success rate of the BC policy, and also because SAC typically undergoes some unlearning at the start of the fine-tuning process. This unlearning when fine-tuning with SAC has been

observed in prior work [112], and is due to the fact that a randomly initialized critic is used to update the policy. For harder (i.e. long-horizon) tasks with more complicated initial conditions (such as blocked drawer 1 and blocked drawer 2), we were unable to get SAC to perform well from a BC initialization, even after we collecting over 5K new episodes.

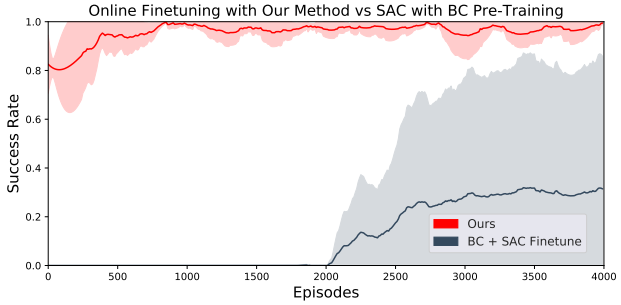


Figure D.3: **Fine-tuning with CQL vs BC+SAC** We compared fine-tuning with CQL to fine-tuning a BC policy with SAC. SAC experiences some unlearning at the start (resulting in a success rate of zero at the start of training), and needs to collect a somewhat large number of samples before it can recover. Further, the variance across three random seeds was quite high for BC+SAC.

D.3 Learning Curves

Here are detailed learning curves for the experiments we summarized in Table 1. Note that the x-axis here denotes number of update steps made to the policy and Q-function, and not the amount of data available to the method. Since we operate in an offline reinforcement learning setting, all data is available to the methods at the start of training. We see that COG is able to achieve a high performance across all initial conditions for both the tasks. We substantially outperform comparisons to prior approaches that are based on pretraining using behavior cloning, including an oracle version that only uses trajectories with a high reward.

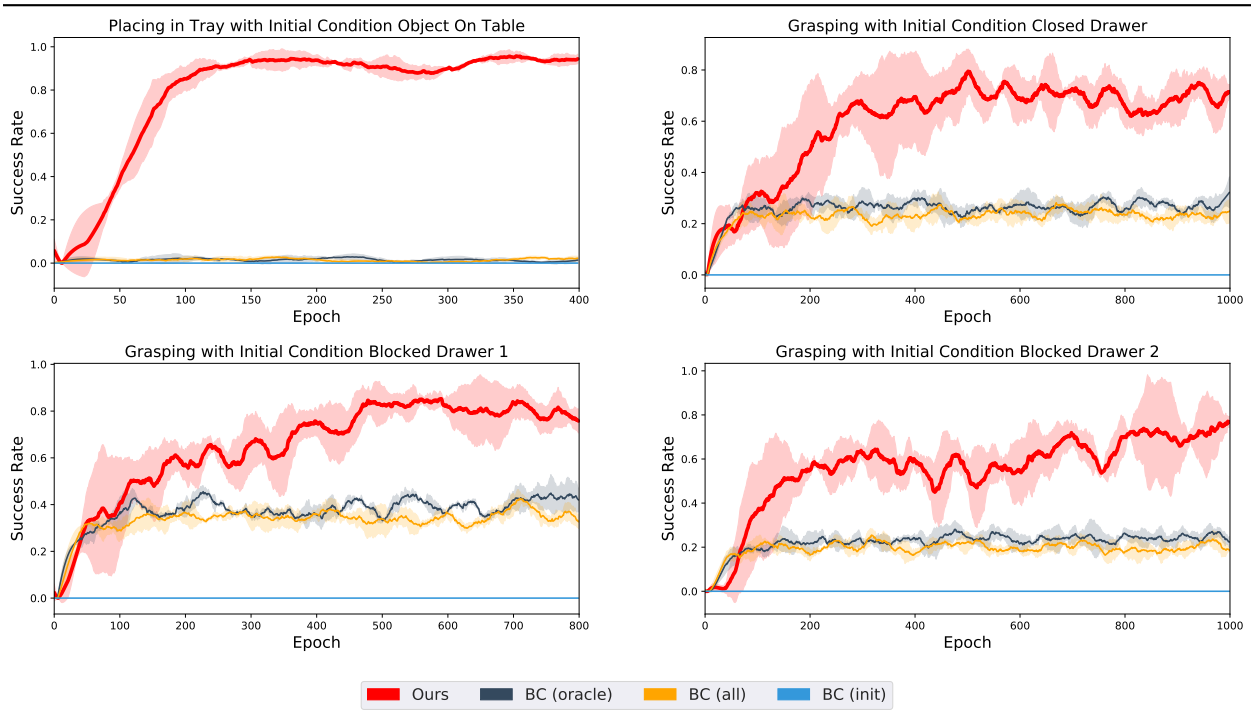


Figure D.4: **Learning curves for simulated experiments by method and initial condition**. Here we compare the success rate curves of our method (COG) to the three behavioral cloning baselines in the four settings of Table 5.1 where prior data is essential for solving the task: the place in tray task with the object starting in the tray (upper left), as well as the grasp from drawer task with a closed drawer (upper right), blocked drawer 1 (lower left), and blocked drawer 2 (lower right).