Scalable Robot Learning

Ashvin Nair

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2022-204
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2022/EECS-2022-204.html

August 12, 2022

Copyright © 2022, by the author(s).
All rights reserved.

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

Acknowledgement

I would like to acknowledge my advisor Sergey Levine and my thesis committee - Pieter Abbeel and Alison Gopnik. I would also like to thank my family and friends who's support made this work possible.

Scalable Robot Learning

by

Ashvin Nair

A dissertation submitted in partial satisfaction

of the requirements for the degree of

Doctor of Philosophy

in

Engineering - Electrical Engineering & Computer Science

in the Graduate Division of the

University of California, Berkeley

Summer 2022

Committee:

Professor Sergey Levine, Chair
Professor Pieter Abbeel
Professor Alison Gopnik

Copyright © 2022 – Ashvin Nair
All rights reserved.

ABSTRACT

For robots to perform tasks in the unstructured environments of the real world, they must be able to be tasked with a desired objective in a general way, and learn to perform the desired task quickly if the robot does not already know how to accomplish it. In this thesis, we explore deep reinforcement learning as a solution to enable this vision for scalable learning-based real-world robotics through two main themes: accelerating reinforcement learning from prior data and self-supervised RL. Accelerating RL from prior data or prior knowledge is important for making reinforcement learning algorithms sufficiently sample-efficient to run directly in the real world. We discuss utilizing human demonstrations to accelerate reinforcement learning, using human-designed residual controllers in combination with reinforcement learning for industrial insertion tasks, and algorithms for offline reinforcement learning that can also benefit from a small amount of online fine-tuning. Concurrently, while sample-efficiency of reinforcement learning algorithms is a well-appreciated problem, additional problems arise around agents that can learn from rich observations such as images: in particular, reward supervision and collecting data autonomously. We discuss self-supervised RL through goal reaching with a generative model, allowing agents to evaluate their own success at reaching goals and autonomously propose and practice skills. In the final section, we consider combining offline policy learning with self-supervised practice, allowing robots to practice and perfect skills in novel environments. These directions enable robots to supervise their own data collection, learning complex and general manipulation skills from interaction.

To my parents.

ACKNOWLEDGMENTS

I would like to thank my advisor Sergey Levine for his guidance over the past years. The work in this thesis would not have been possible without his vision and support. He was a source of inspiration, motivation, and pushed me to work on the hard problems that matter. I have never met anyone more diligent or with so much commitment to excellence, and I was so lucky to have the opportunity to collaborate with him.

I would like to thank Pieter Abbeel for his mentorship. Joining his lab after being inspired by a talk he gave shifted my life trajectory and motivated me to pursue a research career. I would also like to thank Pulkit Agrawal for his mentorship, and for heavily influencing my thinking about robotics, machine learning, and research in general. I would like to thank Alison Gopnik and Alyosha Efros in addition to Sergey and Pieter for serving on my qualifying exam and thesis committee.

I would like to thank all my fantastic collaborators through the years. First, I would like to thank my more senior collaborators who's wisdom and experience I benefited from immensely: Abhishek Gupta, Amy Zhang, Bob McGrew, Eugen Solowjow, Glen Berseth, Ilya Kostrikov, Jitendra Malik, Kuan Fang, Marcin Andrychowicz, Phillip Isola, and Wojciech Zaremba. In particular, I would like to highlight Vitchyr Pong, who was my closest collaborator during my PhD. I learned so much from working with him, and research really started to click for me once we started collaborating. On top of being a great researcher and engineer, Vitchyr was also kind, encouraging, and a joy to work with. I would also like to thank Shikhar Bahl, Murtaza Dalal, Sasha Khazatsky, Graham Hughes, Daniel Jing, Catherine Huang, Patrick Yin, Brian Zhu and Charles Xu specifically for the privilege of being a graduate mentor; I learned so much from the experience and it was one of the best parts of research for me. And I would like to thank all the other collaborators who were instrumental in pursuing this research: Dian Chen, Gerrit Schoettler, Gokul Narayanan, Homer Walke, Jianlan Luo, Laura Smith, Matt Yan, Philippe Hansen-Estruch, Soroush Nasiriany, Steven Lin, Tobias Johannink.

The Robotic AI and Learning Lab (RAIL) at Berkeley was an amazing place to be - I was surrounded by brilliant people and it felt like the state-of-the-art RL algorithms, important insider knowledge, and creative research perspectives were all just a few desks away at most. For this I would like to thank Anusha Nagabandi, Anurag Ajay, Aravind Rajeswaran, Avi Singh, Aviral Kumar, Colin Li, Coline Devin, Dhruv Shah, Dibya Ghosh, Dinesh Jayaraman, Frederik Ebert, Greg Kahn, Kristian Hartikainen, Jason Peng, JD Co-Reyes, Justin Fu, Kate Rakelly, Katie Kang, Kelvin Xu, Laura Smith, Marvin Zhang, Michael Chang, Michael Janner, Natasha Jaques, Nick Rhinehart, Roberto Calandra, Rowan McAllister, Siddharth Reddy, Tuomas Haarnoja, Vikash Kumar, and Young Geng. I will cherish all the great times, conversations, and conference trips we got to experience together.

Spending my entire academic career at Berkeley, the times I got to step away and imbibe new perspectives was always refreshing. I would like to thank Marcin, Bob, Wojciech, and Pieter for my time at OpenAI which kickstarted my main research directions. I would like to thank Tobias, Gerrit, and Eugen for the external collaboration with Siemens which drove me to think about RL from an applications point of view. And I would like to thank Saurabh Gupta, Shubham Tulsiani, and Abhinav Gupta for their mentorship while at FAIR Pittsburgh; my time there really shifted my thinking about which ML problems are valuable to work on as well as about building research communities.

It takes a village to raise a child - I was lucky to have so many teachers and mentors in my life see potential in me, and go above and beyond their duty to help me and push me. I would like to thank them all, but especially David Petty, Peter Wolf, Robert Trakimas, Mike Hansen, Lauren Walsh, Amy Cooper, and Ms. Hunter for their dedication.

I am deeply grateful for the friends I made during graduate school, particularly Armin Askari, Ashish Kumar, Allan Jabri, Chandan Singh, Parsa Mahmoudieh, and Sasha Sax. Playing basketball with Armin, Chandan, and Marvin was a vital source of sanity for me through the pandemic times. I would like to thank Kelvin, Marvin, and Parsa for their support navigating the finish line of the PhD and searching for jobs. I would also like to thank Sasha Sax and Nick Rhinehart for being thoughtful roommates and friends.

I could not have done this work without the love and support of my friends. I would like to thank Aditya Pradhan, Alexandria Lai, Alex Joo, Anna Apa, Brian Giang, Carlos Lazo, Greg Bulpitt, Jong Lim, Karthik Gururangan, Michael You, Nayif Alhomoud, Ryan Goh, Sid Moghe, Shuyang Ren, Sua Kim, and Sunny Zhang: I am so fortunate to have you all in my life. Going further back, I would like to thank Donald Feng, George Zhang, Jessica Zhang, Kevin Gao, Ning Liu, Irene Wong, and Saavan Patel for their continued friendship. I was lucky to share the last 13 (!) years of school with Saavan, and I deeply appreciate his support and loyalty throughout.

Finally, I would like to thank my family. My entire extended family has been a source of comfort and well-wishes throughout this process and I am very grateful for their love and encouragement. I would like to thank my grandparents Damodaran Nair, Sarada

Damodaran Nair, Methil Somasundaram, and Chandramathi Panangadath for their love and care, for inspiring me, and for being excellent role models. I would like to thank my father Vinod Nair for providing me with direction and purpose. And I would like to thank my mother Asha Nair who inspired me with her love for others and sacrificed so much so that I could pursue my dreams.

CONTENTS

1

INTRODUCTION

Robots are becoming ubiquitous in manufacturing and other industries, for a variety of tasks such as bin picking, assembly, welding, painting, and so on. Yet, the autonomous capability of present-day robotics systems are still quite limited. Settings where robots operate are carefully controlled; they often require very specific end-effector tooling (Zhongkui Wang et al., 2020) combined with high precision motions and motion planning (Ang et al., 2005; LaValle, 2006; Karaman and Frazzoli, 2011; Zucker et al., 2013) to accomplish a particular task. In effect, robots rely on human ingenuity and engineering in order to do their job. But such systems are brittle, and the hardware and software must often be redesigned for slight variations of a task. Some adaptability or autonomy can be achieved with compliant robot controllers (Mason, 1981; Hogan, 1985), and with abstract task planning (Sacerdoti, 1974; Leslie Pack Kaelbling and Lozano-Perez, 2011) combined with perception. But if a manufacturing task actually requires significant adaptability or robustness to varying environment conditions based on perceptual inputs, designing a working system becomes much more difficult - as evidenced by the millions of human laborers doing these jobs today. And beyond relatively controlled manufacturing settings, we will expect the robots of tomorrow to do a lot more: cook meals, assist the elderly in homes and other human-centric environments, navigate unmapped terrain, operate machinery and appliances, manipulate objects, and interact safely in presence of humans. This kind of open-world capability requires adaptability, generalization, and is beyond the reach of most robots today.

In contrast, humans perform highly skillful dexterous manipulation so easily that it is sometimes hard to conceive the difficulty of replicating this capability in a robot. Most humans within the first five years of their life have developed complex fine motor skills, successfully performing bimanual dexterous manipulation of various unfamiliar and dynamic objects, and using tools with a tight sensorimotor loop that entails perception, functional grasping, and control (Adolph and Franchak, 2017). It remains a challenge to develop equivalently robust feedback controllers for robots that can adapt to a wide variety of situations to accomplish goals. If robots were equally skillful, it would be incredibly economically valuable - they could be used to automate many of the tasks that humans have to do today. How can we develop methods to allow for general-purpose robots that are similarly skillful?

The past decade of deep learning suggests that learning models from large datasets is the key to such open-world generalization, which is a prerequisite for general robotics. Expressive function approximation trained on broad datasets have driven recent progress in artificial intelligence research across a range of fields: in speech recognition (Graves et al., 2014), image classification (Krizhevsky et al., 2012) and segmentation (Ren et al., 2015), natural language processing (Devlin et al., 2019), and even protein structure prediction (Jumper et al., 2021), the recipe of large datasets combined with appropriate deep learning architectures has pushed forward the frontier. These models are trained on a broad enough dataset that the model can generalize from a broad training distribution and capture corner cases at test time, a challenge with manually designed solutions. If we could achieve such generality for control - the problem of selecting actions in order to maximize a reward function - it could enable truly general robots in the wild.

But control introduces two new problems not found in the supervised learning setting. The first problem is credit assignment: actions taken in the past affect the future. The second is exploration: actions taken change the distribution of data visited. To address these problems, a promising approach is deep reinforcement learning (RL), which combines reinforcement learning with deep function approximation. Deep reinforcement learning has been applied successfully on many sequential decision making problems: to achieve super-human performance on competitive games such as Atari (Mnih et al., 2015), Go (D. Silver et al., 2016a), Dota 2 (OpenAI et al., 2019), and Starcraft II (Vinyals et al., 2019), in robotics (Marc Peter Deisenroth and Rasmussen, 2011; Kober et al., 2013; Levine et al., 2017; J. ; Lee et al., 2020), navigation of stratospheric balloons Marc G. Bellemare et al., 2020, and even control of plasma in a nuclear fusion reactor Degrave et al., 2022. Yet, while algorithms for RL have been steadily advancing (Schulman et al., 2015; Lillicrap et al., 2016; Schulman et al., 2017; Haarnoja et al., 2018a), becoming more sample efficient and stable, there are still significant obstacles towards a general solution for robotics with RL. What challenges remain toward endowing robots with human-level manipulation skills?

Challenges in robot learning. The central issue is that the world is so varied that a robotic agent acquiring skills via learning based methods needs to experience both the diversity of perceptual inputs found in large-scale vision datasets, combined with the complexity of control including exploration and credit assignment. But unlike the supervised learning setting where collecting and labeling data is relatively straightforward, collecting diverse and useful data in robotics is much harder and more expensive. Existing work has used demonstrations, which require significant human effort, or run online policy learning in heavily instrumented environments which also requires effort to design reset mechanisms and reward engineering. Instead, robots would ideally be able to collect their own data continuously with little human supervision in diverse environments as well as utilize past data to learn a policy that generalizes well to diverse environments. How can we bootstrap this cycle, in which robots can successfully use prior data to explore and learn in diverse environments? Broadly, two key components need to be developed for a solution.

First, we must be able to embed prior knowledge - data of offline experience or demonstrations, human-engineered controllers, and environment models, into the learning process. The classic active formulation of RL learning from scratch necessitates a lengthy active exploration process for each behavior, making it difficult to apply in real-world settings such as robotic control. Beyond requiring a large amount of data, the initial exploration phase in RL can also be unsafe to execute on a robot. Instead of learning tasks from scratch, we ought to be able to utilize existing knowledge to make RL algorithms sample efficient and practical for running in the real world. If we can instead allow RL algorithms to effectively use prior knowledge to aid the online learning process, such applications could be made substantially more practical, providing a starting point that mitigates challenges due to exploration and sample complexity. But vitally, online training may still be necessary for the agent to perfect the desired skill, if perfect zero-shot generalization is not achieved. In this thesis, we explore several ideas in order to incorporate prior knowledge into policy optimization, significantly improving the sample complexity, reducing unsafe exploration behavior, and making these algorithms more amenable for real world use.

Second, we must address the problem of task specification. While reinforcement learning provides an appealing formalism for learning individual skills, a general-purpose robotic system must be able to master an extensive repertoire of behaviors. Moreover, it must correctly evaluate itself on whether it has succeeded or failed at accomplishing a task, and be able to be tasked with a specific desired goal by a human when needed. Ideally it could also transfer knowledge between tasks. Goal-conditioned reinforcement learning provides a potential solution to this problem: by conditioning the policy on a continuous goal space, we can enable this transfer and specify tasks in the goal space. For scalable robot learning, we also want the learning procedure to be *self-supervised*:

the robot should be able to evaluate its own success in order to practice, and also autonomously practice useful skills by setting goals that are feasible but diverse.

If we can overcome these challenges reliably, we can allow robots to use their prior experience to learn general policies, and then when tasked with a new objective, practice and improve through self-supervision. Moreover, the ability to explore in new environments can be the basis to bootstrap a cycle in which our agents use prior experience to learn a policy to collect high quality interaction data, improve the policy from that data, and so on. This thesis first explores facets of each challenge individually, demonstrating new algorithms on real-world robotic systems. We then integrate these solutions into real-world robotic systems that understand affordances and autonomously improve on novel tasks.

Outline. This thesis is organized into three parts. In Part I (chapter 2-4), we investigate the use of goal-conditioned reinforcement learning for self-supervised exploration from raw observations. In chapter 2, we describe the framework of reinforcement learning with imagined goals (RIG), which enables self-supervised practice (A. Nair et al., 2018b). In chapter 3, we discuss extending RIG to autonomously set goals in novel situations based on prior experience, using a context-conditioned generative model (A. Nair et al., 2019a). In chapter 4, we discuss a general framework for exploration in self-supervised goal-conditioned RL (Vitchyr H. Pong et al., 2019).

In Part 2 (chapter 5-10), we discuss utilizing prior data and prior knowledge in order to initialize and accelerate reinforcement learning. In chapter 5, we discuss utilizing demonstrations to solve long-horizon tasks with reinforcement learning (A. Nair et al., 2018a). In chapter 6, we discuss an algorithm to utilize arbitrary offline data and finetune policies and value functions online (A. Nair et al., 2020). In chapter 7, we discuss utilizing expectile regression for stable offline learning of value functions. The resulting algorithm, implicit Q-learning, achieves state-of-the-art results in both offline RL and online finetuning (Kostrikov et al., 2021b). In chapter 8, we discuss how to incorporate prior knowledge such as an expert controller using residual reinforcement learning (Johannink et al., 2019), and in chapter 9 we further extend residual reinforcement learning to solve industrial insertion tasks (Gerrit Schoettler et al., 2019). In chapter 10, we apply implicit Q-learning along with domain generalization to learn reward models and policies for industrial insertion, enabling on-the-job learning when a policy cannot solve an insertion task zero-shot.

Finally, in Part 3 (chapter 11-12), we show how these two directions dovetail to enable robots in the real world to explore novel situations. In chapter 11, we cover visuomotor affordance learning (VAL), a method to allow self-supervised learning from prior data (Khazatsky et al., 2021a). In chapter 12, we extend VAL with planning to enable finetuning of more complex skills. In chapter 13, we conclude by discussing future directions.

Part I

REINFORCEMENT LEARNING WITH IMAGINED GOALS

2

VISUAL REINFORCEMENT LEARNING WITH IMAGINED GOALS

2.1 INTRODUCTION

Reinforcement learning (RL) algorithms hold the promise of allowing autonomous agents, such as robots, to learn to accomplish arbitrary tasks. However, the standard RL framework involves learning policies that are specific to individual tasks, which are defined by hand-specified reward functions. Agents that exist persistently in the world can prepare to solve diverse tasks by setting their own goals, practicing complex behaviors, and learning about the world around them. In fact, humans are very proficient at setting abstract goals for themselves, and evidence shows that this behavior is already present from early infancy (Smith and Gasser, 2005), albeit with simple goals such as reaching. The behavior and representation of goals grows more complex over time as they learn how to manipulate objects and locomote. How can we begin to devise a reinforcement learning system that sets its own goals and learns from experience with minimal outside intervention and manual engineering?

In this paper, we take a step toward this goal by designing an RL framework that jointly learns representations of raw sensory inputs and policies that achieve arbitrary goals under this representation by practicing to reach self-specified random goals during training. To provide for automated and flexible goal-setting, we must first choose how a general goal can be specified for an agent interacting with a complex and highly variable environment. Even providing the state of such an environment to a policy is a challenge. For instance, a task that requires a robot to manipulate various objects would require a combinatorial representation, reflecting variability in the number and type of objects in the current scene. Directly using raw sensory signals, such as images, avoids this challenge, but learning from raw images is substantially harder. In particular, pixel-wise Euclidean distance is not an effective reward function for visual tasks since distances between images do not correspond to meaningful distances between states (Ponomarenko et al., 2015; Richard Zhang et al., 2018). Furthermore, although end-to-end model-free reinforcement learning can handle image observations, this comes at a high cost in sample complexity, making it difficult to use in the real world.

We propose to address both challenges by incorporating unsupervised representation learning into goal-conditioned policies. In our method, which is illustrated in Figure 47, a representation of raw sensory inputs is learned by means of a latent variable model, which in our case is based on the variational autoencoder (VAE) (D. P. Kingma and Welling, 2014). This model serves three complementary purposes. First, it provides a more structured representation of sensory inputs for RL, making it feasible to learn from images even in the real world. Second, it allows for sampling of new states, which can be used to set synthetic goals during training to allow the goal-conditioned policy to practice diverse behaviors. We can also more efficiently utilize samples from the environment by relabeling synthetic goals in an off-policy RL algorithm, which makes our algorithm substantially more efficient. Third, the learned representation provides a space where distances are more meaningful than the original space of observations, and can therefore provide well-shaped reward functions for RL. By learning to reach random goals sampled from the latent variable model, the goal-conditioned policy learns about the world and can be used to achieve new, user-specified goals at test-time.

The main contribution of our work is a framework for learning general-purpose goal-conditioned policies that can achieve goals specified with target observations. We call our method reinforcement learning with imagined goals (RIG). RIG combines sample-efficient off-policy goal-conditioned reinforcement learning with unsupervised representation learning. We use representation learning to acquire a latent distribution that can be used to sample goals for unsupervised practice and data augmentation, to provide a well-shaped distance function for reinforcement learning, and to provide a more structured representation for the value function and policy. While several prior methods, discussed in the following section, have sought to learn goal-conditioned policies, we can do so with image goals and observations without a manually specified reward signal. Our experimental evaluation illustrates that our method substantially improves the performance of image-based reinforcement learning, can effectively learn policies for complex image-based tasks, and can be used to learn real-world robotic manipulation skills with raw image inputs. Videos of our method in simulated and real-world environments can be found at `https://sites.google.com/site/visualrlwithimaginedgoals/`.

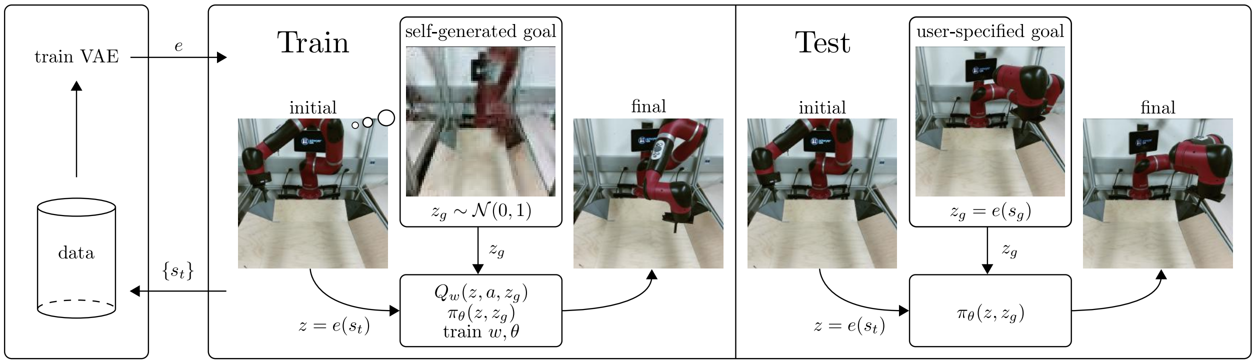


Figure 1: We train a VAE using data generated by our exploration policy (left). We use the VAE for multiple purposes during training time (middle): to sample goals to train the policy, to embed the observations into a latent space, and to compute distances in the latent space. During test time (right), we embed a specified goal observation o_g into a goal latent z_g as input to the policy.

2.2 RELATED WORK

While prior works on vision-based deep reinforcement learning for robotics can efficiently learn a variety of behaviors such as grasping (Pinto et al., 2018; Pinto and Abhinav Gupta, 2016; Levine et al., 2017), pushing (Agrawal et al., 2016; Ebert et al., 2017; Finn and Levine, 2016), navigation (Pathak et al., 2018; Lange et al., 2012c), and other manipulation tasks (Lillicrap et al., 2016; Levine et al., 2016a; Pathak et al., 2018), they each make assumptions that limit their applicability to training general-purpose robots. Levine et al. (2016a) uses time-varying models, which requires an episodic setup that makes them difficult to extend to non-episodic and continual learning scenarios. Pinto et al. (2018) proposed a similar approach that uses goal images, but requires instrumented training in simulation. Lillicrap et al. (2016) uses fully model-free training, but does not learn goal-conditioned skills. As we show in our experiments, this approach is very difficult to extend to the goal-conditioned setting with image inputs. Model-based methods that predict images (Watter et al., 2015; Finn and Levine, 2016; Ebert et al., 2017; Oh et al., 2015) or learn inverse models (Agrawal et al., 2016) can also accommodate various goals, but tend to limit the horizon length due to model drift. To our knowledge, no prior method uses model-free RL to learn policies conditioned on a single goal image with sufficient efficiency to train directly on real-world robotic systems, without access to ground-truth state or reward information during training.

Our method uses a goal-conditioned value function (Schaul et al., 2015b) in order to solve more general tasks (R. S. Sutton et al., 2011; L P Kaelbling, 1993). To improve the sample-efficiency of our method during off-policy training, we retroactively relabel samples in the replay buffer with goals sampled from the latent representation. Goal relabeling has been explored in prior work (L P Kaelbling, 1993; Andrychowicz et al., 2017b; Rauber et al., 2017; Levy et al., 2017; V. Pong et al., 2018). Andrychowicz et al. (2017b) and Levy et al. (2017) use goal relabeling for sparse rewards problems with known goal spaces, restricting the resampled goals to states encountered along that trajectory, since almost any other goal will have no reward signal. We sample random goals from our learned latent space to use as replay goals for off-policy Q-learning rather than restricting ourselves to states seen along the sampled trajectory, enabling substantially more efficient learning. We use the same goal sampling mechanism for exploration in RL. Goal setting for policy learning has previously been discussed (Baranes and Oudeyer, 2012) and recently Pr et al. (2018a) have also proposed using unsupervised learning for setting goals for exploration. However, we use a model-free Q-learning method that operates on raw state observations and actions, allowing us to solve visually and dynamically complex tasks.

A number of prior works have used unsupervised learning to acquire better representations for RL. These methods use the learned representation as a substitute for the state for the policy, but require additional information, such as access to the ground truth reward function based on the true state during training time (Higgins et al., 2017b; Ha and Schmidhuber, 2018; Watter et al., 2015; Finn et al., 2016b; Lange et al., 2012c; Jonschkowski et al., 2017b), expert trajectories (Srinivas et al., 2018), human demonstrations (Sermanet et al., 2017), or pre-trained object-detection features (A. Lee et al., 2017). In contrast, we learn to generate goals and use the learned representation to obtain a reward function for those goals without any of these extra sources of supervision. Finn et al. (2016b) combine unsupervised representation learning with reinforcement learning, but in a framework that trains a policy to reach a single goal. Many prior works have also focused on learning controllable and disentangled representations (Schmidhuber, 1992; X. Chen et al., 2016; Cheung et al., 2014; Reed et al., 2014; Desjardins et al., 2012; V. Thomas et al., 2017). We use a method based on variational autoencoders, but these prior techniques are complementary to ours and could be incorporated into our method.

2.3 BACKGROUND

Our method combines reinforcement learning with goal-conditioned value functions and unsupervised representation learning. Here, we briefly review the techniques that we build on in our method.

REINFORCEMENT LEARNING. We consider the standard Markov Decision Process framework for picking optimal actions to maximize rewards over discrete timesteps in an environment E. We assume that the environment is fully observable. At every timestep, an agent is in a state s, takes an action a, receives a reward r, and E evolves to state s' according to the environment dynamics $p(s'|s,a)$. The state, action, and reward at timestep t may also be denoted s_t, a_t, and r_t respectively but we will drop the subscript when considering individual transitions (s,a,r,s'). In reinforcement learning, the agent must learn a policy $a = \pi(s)$ to maximize expected returns. We denote the return starting at timestep t by $R_t = \sum_{i=t}^{T} \gamma^{(i-t)} r_i$ where T is the horizon that the agent optimizes over and γ is a discount factor for future rewards. The agent's objective is to maximize expected return from the start distribution $J = \mathbb{E}_{r_i,s_i \sim E, a_i \sim \pi}[R_0]$.

A variety of reinforcement learning algorithms have been developed to solve this problem. Many involve constructing an estimate of the expected return from a given state after taking an action:

$$\begin{aligned} Q^{\pi}(s,a) &= \mathbb{E}[R_t|s_t = s, a_t = a] && (1) \\ &= \mathbb{E}[r + \gamma \mathbb{E}_{a' \sim \pi}[Q^{\pi}(s',a')]], && (2) \end{aligned}$$

where the outer expectation is taken over the policy and environment dynamics.
We call Q^{π} the action-value function. Equation 13 is a recursive version of equation 12, and is known as the Bellman equation. The Bellman equation allows for methods to estimate Q that resemble dynamic programming, enabling data re-use and sample efficiency.

GOAL-CONDITIONED REINFORCEMENT LEARNING. Standard model-free RL learns policies that achieve a single task. If our aim is instead to obtain a policy that can accomplish a variety of tasks, we can construct a goal-conditioned policy and reward, and optimize the expected return with respect to a goal distribution: $\mathbb{E}_{g \sim G}[\mathbb{E}_{r_i,s_i \sim E, a_i \sim \pi}[R_0]]$, where $\mathcal{G}$ is the set of goals and the reward is also a function of g. A variety of algorithms can learn goal-conditioned policies, but to enable sample-efficient learning, we focus on algorithms that acquire goal-conditioned Q-functions, which can be trained off-policy. A goal-conditioned Q-function $Q(s,a,g)$ learns the expected return for the goal g starting from state s and taking action a. Given a state s, action a, next state s', goal g, and correspond reward r, one can train an approximate Q-function parameterized by w by minimizing the following Bellman error

$$\mathcal{E}(w) = \frac{1}{2}||Q_w(s, a, g) - (r + \gamma \max_{a'} Q_{\bar{w}}(s', a', g))||^2 \tag{3}$$

where $\bar{w}$ indicates that $\bar{w}$ is treated as a constant. Crucially, one can optimize this loss using off-policy data (s, a, s', g, r) with a standard actor-critic algorithm (Lillicrap et al., 2016; Fujimoto et al., 2018b; Mnih et al., 2016).

VARIATIONAL AUTOENCODERS. Variational autoencoders (VAEs) have been demonstrated to learn structured latent representations of high dimensional data (D. P. Kingma and Welling, 2014). The VAE consists of an encoder p_ϕ, which maps states to latent distributions, and a decoder p_ψ, which maps latents to distributions over states. The encoder and decoder parameters, ψ and ϕ respectively, are jointly trained to maximize

$$\mathcal{L}(\psi, \phi; s^{(i)}) = -\beta D_{KL}(q_\phi(z|s^{(i)})||p(z)) + \mathbb{E}_{q_\phi(z|s^{(i)})}[\log p_\psi(s^{(i)} \mid z)], \tag{4}$$

where $p(z)$ is some prior, which we take to be the unit Gaussian, D_{KL} is the Kullback-Leibler divergence, and β is a hyperparameter that balances the two terms. The use of β values other than one is sometimes referred to as a β-VAE (Higgins et al., 2017a). The encoder q_ϕ parameterizes the mean and log-variance diagonal of a Gaussian distribution, $q_\phi(s) = \mathcal{N}(\mu_\phi(s), \sigma^2_\phi(s))$. The decoder p_ψ parameterizes a Bernoulli distribution for each pixel value. This parameterization corresponds to training the decoder with cross-entropy loss on normalized pixel values. Full details of the hyperparameters are in the Supplementary Material.

2.4 GOAL-CONDITIONED POLICIES WITH UNSUPERVISED REPRESENTATION LEARNING

To devise a practical algorithm based on goal-conditioned value functions, we must choose a suitable goal representation. In the absence of domain knowledge and instrumentation, a general-purpose choice is to set the goal space $\mathcal{G}$ to be the same as the state observations space $\mathcal{S}$. This choice is fully general as it can be applied to any task, and still permits considerable user control since the user can choose a "goal state" to set a desired goal for a trained goal-conditioned policy. But when the state space $\mathcal{S}$ corresponds to high-dimensional sensory inputs such as images, [1] learning a goal-conditioned Q-function and policy becomes exceedingly difficult as we illustrate empirically in Section 9.5.

Our method jointly addresses a number of problems that arise when working with high-dimensional inputs such as images: sample efficient learning, reward specification, and automated goal-setting. We address these problems by learning a latent embedding using a β-VAE. We use this latent space to represent the goal and state and retroactively relabel data with latent goals sampled from the VAE prior to improve sample efficiency. We also show that distances in the latent space give us a well-shaped reward function for images. Lastly, we sample from the prior to allow an agent to set and "practice" reaching its own goal, removing the need for humans to specify new goals during training time. We next describe the specific components of our method, and summarize our complete algorithm in Section 2.4.5.

2.4.1 *Sample-Efficient RL with Learned Representations*

One challenging problem with end-to-end approaches for visual RL tasks is that the resulting policy needs to learn both perception and control. Rather than operating directly on observations, we embed the state s_t and goals g into a latent space $\mathcal{Z}$ using an encoder e to obtain a latent state $z_t = e(s_t)$ and latent goal $z_g = e(g)$. To learn a representation of the state and goal space, we train a β-VAE by executing a random policy and collecting state observations, $\{s^{(i)}\}$, and optimize Equation `equation` 4. We then use the mean of the encoder as the state encoding, i.e. $z = e(s) \triangleq \mu_\phi(s)$.

After training the VAE, we train a goal-conditioned Q-function $Q(z, a, z_g)$ and corresponding policy $\pi_\theta(z, z_g)$ in this latent space. The policy is trained to reach a goal z_g using the reward function discussed in Section 2.4.2. For the underlying RL algorithm, we use twin delayed deep deterministic policy gradients (TD3) (Fujimoto et al., 2018b), though any value-based RL algorithm could be used. Note that the policy (and Q-function) operates completely in the latent space. During test time, to reach a specific goal state g, we encode the goal $z_g = e(g)$ and input this latent goal to the policy.

As the policy improves, it may visit parts of the state space that the VAE was never trained on, resulting in arbitrary encodings that may not make learning easier. Therefore, in addition to procedure described above, we fine-tune the VAE using both the randomly generated state observations $\{s^{(i)}\}$ and the state observations collected during exploration.

1 We make the simplifying assumption that the system is Markovian with respect to the sensory input, and one could incorporate memory into the state for partially observed tasks.

We show in Section A.1.3 that this additional training helps the performance of the algorithm.

2.4.2 *Reward Specification*

Training the goal-conditioned value function requires defining a goal-conditioned reward $r(s, g)$. Using Euclidean distances in the space of image pixels provides a poor metric, since similar configurations in the world can be extremely different in image space. In addition to compactly representing high-dimensional observations, we can utilize our representation to obtain a reward function based on a metric that better reflects the similarity between the state and the goal. One choice for such a reward is to use the negative Mahalanobis distance in the latent space:

$$r(s, g) = -||e(s) - e(g)||_A = -||z - z_g||_A,$$

where the matrix A weights different dimensions in the latent space. This approach has an appealing interpretation when we set A to be the precision matrix of the VAE encoder, q_ϕ. Since we use a Gaussian encoder, we have that

$$r(s, g) = -||z - z_g||_A \propto \sqrt{\log e_\phi(z_g \mid s)} \tag{5}$$

In other words, minimizing this squared distance in the latent space is equivalent to rewarding reaching states that maximize the probability of the latent goal z_g. In practice, we found that setting $A = \mathbf{I}$, corresponding to Euclidean distance, performed better than Mahalanobis distance, though its effect is the same — to bring z close to z_g and maximize the probability of the latent goal z_g given the observation. This interpretation would not be possible when using normal autoencoders since distances are not trained to have any probabilistic meaning. Indeed, we show in Section 9.5 that using distances in a normal autoencoder representation often does not result in meaningful behavior.

2.4.3 *Improving Sample Efficiency with Latent Goal Relabeling*

To further enable sample-efficient learning in the real world, we use the VAE to relabel goals. Note that we can optimize Equation equation 24 using any valid (s, a, s', g, r) tuple. If we could artificially generate these tuples, then we could train our entire RL algorithm without collecting any data. However, we do not know the system dynamics, and therefore have to sample transitions (s, a, s') by interacting with the world. However, we have the freedom to relabel the goal and reward synthetically. In particular, if we have a mechanism for generating goals and computing rewards, then given (s, a, s'), we can generate a new goal g and new reward $r(s, a, s', g)$ to produce a new tuple (s, a, s', g, r). By artificially generating and recomputing rewards, we can convert a single (s, a, s') transition into potentially infinitely many valid training datums.

For image-based tasks, this procedure would require generating goal images, an onerous task on its own. However, our reinforcement learning algorithm operates directly in the latent space for goals and rewards. So rather than generating goals g, we generate latent goals z_g by sampling from the VAE prior $p(z)$. We then recompute rewards using Equation equation 5. By retroactively relabeling the goals and rewards, we obtain much more data to train our value function. This sampling procedure is made possible by our use of a latent variable model, which is explicitly trained so that sampling from the latent distribution is straightforward.

Retroactively generating goals is also explored in tabular domains by L P Kaelbling (1993) and in continuous domains by Andrychowicz et al. (2017b) using hindsight experience replay (HER). However, HER is limited to sampling goals seen along a trajectory, which greatly limits the number and diversity of goals with which one can relabel a given transition. Our final method uses a mixture of the two strategies: half of the goals are generated from the prior and half from goals seen along the trajectory. We show in Section 9.5 that relabeling the goal with samples from the VAE prior results in significantly better sample-efficiency.

Algorithm 1 RIG: Reinforcement learning with imagined goals

```
Require: VAE encoder $q_\phi$, VAE decoder $p_\psi$, policy
    $\pi_\theta$, goal-conditioned value function $Q_w$.
 1: Collect $\mathcal{D} = \{s^{(i)}\}$ using exploration policy.
 2: Train $\beta$-VAE on $\mathcal{D}$ by optimizing equation 4.
 3: for $n = 0, ..., N-1$ episodes do
 4:     Sample latent goal from prior $z_g \sim p(z)$.
 5:     Sample initial state $s_0 \sim E$.
 6:     for $t = 0, ..., H-1$ steps do
 7:         Get action $a_t = \pi_\theta(e(s_t), z_g) + \text{noise}$.
 8:         Get next state $s_{t+1} \sim p(\cdot \mid s_t, a_t)$.
 9:         Store $(s_t, a_t, s_{t+1}, z_g)$ into replay buffer $\mathcal{R}$.
10:         Sample transition $(s, a, s', z_g) \sim \mathcal{R}$.
11:         Encode $z' = e(s')$.
12:         (Probability 0.5) replace $z_g$ with $z'_g \sim p(z)$.
13:         Compute new reward $r = -||z' - z_g||$.
14:         Minimize equation 24 using $(z, a, z', z_g, r)$.
15:     end for
16:     Fine-tune $\beta$-VAE every $K$ episodes on mixture of $\mathcal{D}$ and $\mathcal{R}$.
17: end for=0
```

2.4.4 *Automated Goal-Generation for Exploration*

If we do not know which particular goals will be provided at test time, we would like our RL agent to carry out a self-supervised "practice" phase during training, where the algorithm proposes its own goals, and then practices how to reach them. Since the VAE prior represents a distribution over latent goals and state observations, we again sample from this distribution to obtain plausible goals. After sampling a goal latent from the prior $z_g \sim p(z)$, we give this to our policy $\pi(z, z_g)$ to collect data.

2.4.5 *Algorithm Summary*

We call the complete algorithm reinforcement learning with imagined goals (RIG) and summarize it in Algorithm 2. We first collect data with a simple exploration policy, though any exploration strategy could be used for this stage, including off-the-shelf exploration bonuses (Pathak et al., 2017; M. Bellemare et al., 2016) or unsupervised reinforcement learning methods (Eysenbach et al., 2018; Florensa et al., 2017). Then, we train a VAE latent variable model on state observations and finetune it over the course of training. We use this latent variable model for multiple purposes: We sample a latent goal z_g from the model and condition the policy on this goal. We embed all states and goals using the model's encoder. When we train our goal-conditioned value function, we resample goals from the prior and compute rewards in the latent space using Equation equation 5. Any RL algorithm that trains Q-functions could be used, and we use TD3 (Fujimoto et al., 2018b) in our implementation.

2.5 EXPERIMENTS

Our experiments address the following questions:

1. How does our method compare to prior model-free RL algorithms in terms of sample efficiency and performance, when learning continuous control tasks from images?
2. How critical is each component of our algorithm for efficient learning?
3. Does our method work on tasks where the state space cannot be easily specified ahead of time, such as tasks that require interaction with variable numbers of objects?
4. Can our method scale to real world vision-based robotic control tasks?

For the first two questions, we evaluate our method against a number of prior algorithms

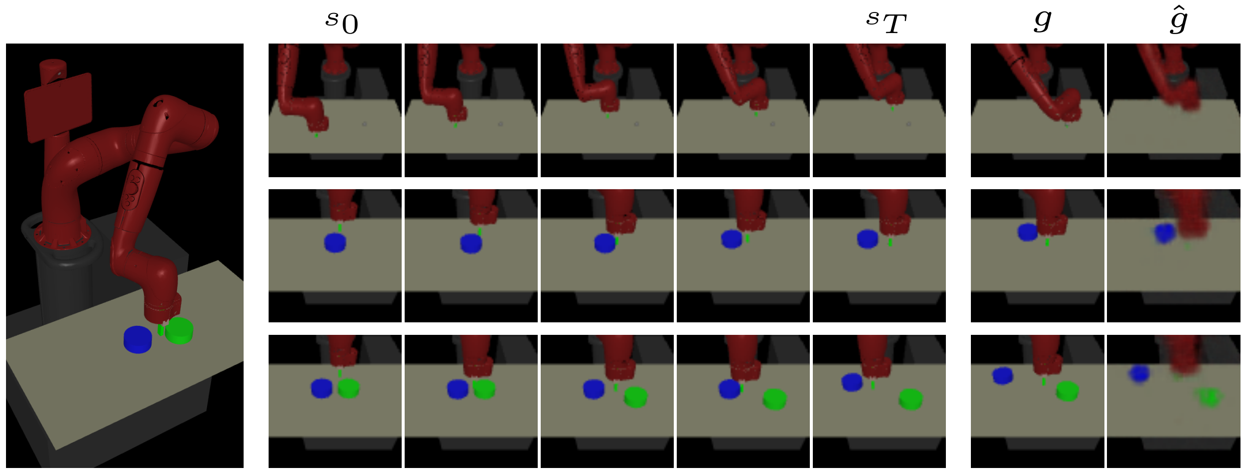


Figure 2: (Left) The simulated environment. (Right) Test rollouts from our learned policy on the three simulated environments. Each row is one rollout. The middle shows the goal image g and its VAE reconstruction ĝ. The right columns shows frames from the trajectory to reach the given goal.

and ablated versions of our approach on a suite of simulated and real-world tasks: *Visual Reacher*: a MuJoCo (Todorov et al., 2012) environment with a 7-dof Sawyer arm reaching goal positions. The arm is shown the left of Figure 2. The end-effector (EE) is constrained to a 2-dimensional rectangle parallel to a table. The action controls EE velocity within a maximum velocity. *Visual Pusher*: a MuJoCo environment with a 7-dof Sawyer arm and a small puck on a table that the arm must push to a target push. *Visual Multi-Object Pusher*: a copy of the Visual Pusher environment with two pucks. Detailed descriptions of the environments are provided in the Supplementary Material.

Solving these tasks directly from images poses a challenge since the controller must learn both perception and control. The evaluation metric is the distance of objects (including the arm) to their respective goals. To evaluate our policy, we set the environment to a sampled goal position, capture an image, and encode the image to use as the goal. Although we use the ground-truth positions for evaluation, **we do not use the ground-truth positions for training the policies**. The only inputs from the environment that our algorithm receives are the image observations. For Visual Reacher, we pretrained the VAE with 100 images. For other tasks, we used 10,000 images.

We compare our method with the following prior works. *L&R*: Lange and Riedmiller (Lange and M. A. Riedmiller, 2010) trains an autoencoder to handle images. *DSAE*: Deep spatial autoencoders (Finn et al., 2016b) learns a spatial autoencoder and uses

2 In all our simulation results, each plot shows a 95% confidence interval of the mean across 5 seeds.

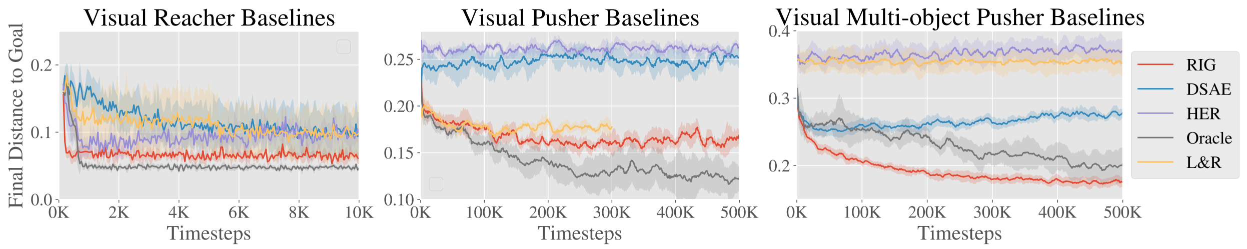


Figure 3: Simulation results, final distance to goal vs simulation steps[2]. RIG (red) consistently outperforms the baselines, except for the oracle which uses ground truth object state for observations and rewards. On the hardest task, only our method and the oracle discover viable solutions.

guided policy search (Levine et al., 2016a) to achieve a single goal image. *HER*: Hindsight experience replay (Andrychowicz et al., 2017b) utilizes a sparse reward signal and relabeling trajectories with achieved goals. *Oracle*: RL with direct access to state information for observations and rewards.

To our knowledge, no prior work demonstrates policies that can reach a variety of goal images without access to a true-state reward function, and so we needed to make modifications to make the comparisons feasible. L&R assumes a reward function from the environment. Since we have no state-based reward function, we specify the reward function as distance in the autoencoder latent space. HER does not embed inputs into a latent space but instead operates directly on the input, so we use pixel-wise mean squared error (MSE) as the metric. DSAE is trained only for a single goal, so we allow the method to generalize to a variety of test goal images by using a goal-conditioned Q-function. To make the implementations comparable, we use the same off-policy algorithm to train L&R, HER, and our method (TD3 (Fujimoto et al., 2018b)). Unlike our method, prior methods do not specify how to select goals during training, so we favorably give them real images as goals for rollouts, sampled from the same distribution that we use to test.

We see in Figure 3 that our method can efficiently learn policies from visual inputs to perform simulated reaching and pushing, without access to the object state. Our approach substantially outperforms the prior methods, for which the use of image goals and observations poses a major challenge. HER struggles because pixel-wise MSE is hard to optimize. Our latent-space rewards are much better shaped and allow us to learn more complex tasks. Finally, our method is close to the state-based "oracle" method in terms of sample efficiency and performance, without having any access to object state. Notably, in the multi-object environment, our method actually outperforms the oracle, likely because the state-based reward contains local minima. Overall, these result show that our method is capable of handling raw image observations much more effectively than previously proposed goal-conditioned RL methods. Next, we perform ablations to evaluate our contributions in isolation. Results on Visual Pusher are shown but see the Supplementary Material (section A.1) for experiments on all three simulated environments.

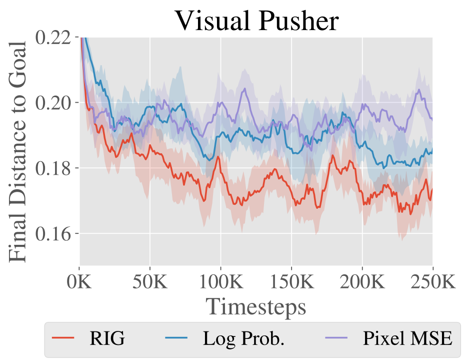


Figure 4: Reward type ablation results. RIG (red), which uses latent Euclidean distance, outperforms the other methods.

REWARD SPECIFICATION COMPARISON We evaluate how effective distances in the VAE latent space are for the Visual Pusher task. We keep our method the same, and only change the reward function that we use to train the goal-conditioned valued function. We include the following methods for comparison: *Latent Distance*, which is the reward used in RIG, i.e. $A = \mathbf{I}$ in Equation equation 5; *Log Probability*, which uses the Mahalanobis distance in Equation equation 5, where A is the precision matrix of the encoder; and *Pixel MSE*, which computes mean-squared error (MSE) between state and goal in pixel space. [3] In Figure 4, we see that latent distance significantly outperforms the log probability. We suspect that small variances of the VAE encoder results in drastically large rewards, making the learning more difficult. We also see that latent distances results in faster learning when compared to pixel MSE.

3 To compute the pixel MSE for a sampled latent goal, we decode the goal latent using the VAE decoder, p_ψ, to generate the corresponding goal image.

RELABELING STRATEGY COMPARISON As described in section 2.4.3, our method uses a novel goal relabeling method based on sampling from the generative model. To isolate how much our new goal relabeling method contributes to our algorithm, we vary the resampling strategy while fixing other components of our algorithm. The resampling strategies that we consider are: *Future*, relabeling the goal for a transition by sampling uniformly from future states in the trajectory as done in Andrychowicz et al. (2017b); *VAE*, sampling goals from the VAE only; *RIG*, relabeling goals with probability 0.5 from the VAE and probability 0.5 using the future strategy; and *None*, no relabeling. In Figure 5, we see that sampling from the VAE and Future is significantly better than not relabeling at all. In RIG, we use an equal mixture of the VAE and Future sampling strategies, which performs best by a large margin. Appendix section A.1.1 contains results on all simulated environments, and section A.1.4 considers relabeling strategies with a known goal distribution.

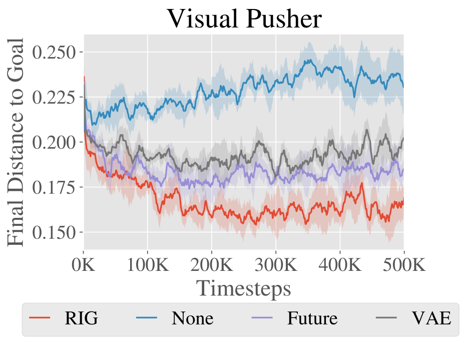


Figure 5: Relabeling ablation results. RIG (red), which uses a mixture of VAE and HER, outperforms the other methods.

LEARNING WITH VARIABLE NUMBERS OF OBJECTS A major advantage of working directly from pixels is that the policy input can easily represent combinatorial structure in the environment, which would be difficult to encode into a fixed-length state vector even if a perfect perception system were available. For example, if a robot has to interact with different combinations and numbers of objects, picking a single MDP state representation would be challenging, even with access to object poses. By directly processing images for both the state and the goal, no modification is needed to handle the combinatorial structure: the number of pixels always remains the same, regardless of how many objects are in the scene.

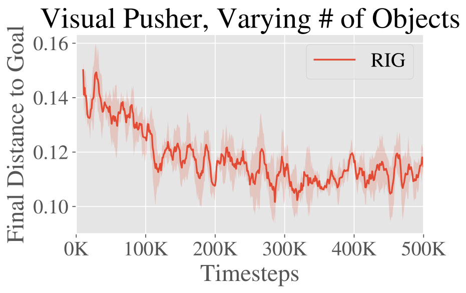


Figure 6: Training curve for learning with varying number of objects.

We demonstrate that our method can handle this difficult scenario by evaluating on a task where the environment, based on the Visual Multi-Object Pusher, randomly contains zero, one, or two objects in each episode during testing. During training, each episode still always starts with both objects in the scene, so the experiments tests whether a trained policy can handle variable numbers of objects at test time. Figure 6 shows that our method can learn to solve this task successfully, without decrease in performance from the base setting where both objects are present (in Figure 3). Developing and demonstrating algorithms that solve tasks with varied underlying structure is an important step toward creating autonomous agents that can handle the diversity of tasks present "in the wild."

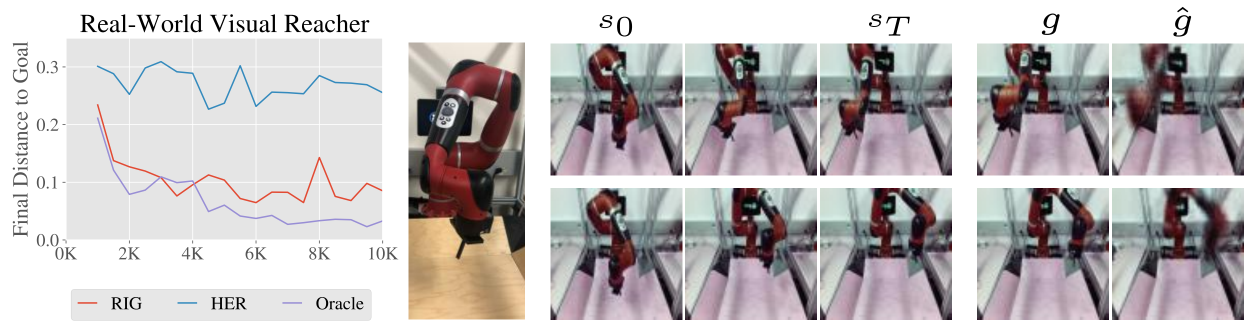
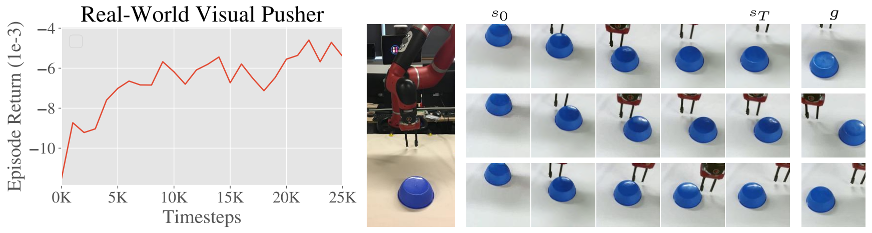


Figure 7: (Left) Our method compared to the HER baseline and oracle on a real-world visual reaching task. (Middle) Our robot setup is pictured. (Right) Test rollouts of our learned policy.



Puck Distance to Goal (cm)

RIG	HER
4.5 ± 2.5	14.9 ± 5.4

Figure 8: (Left) The learning curve for real-world pushing. (Middle) Our robot pushing setup is pictured, with frames from test rollouts of our learned policy. (Right) Our method compared to the HER baseline on the real-world visual pushing task. We evaluated the performance of each method by manually measuring the distance between the goal position of the puck and final position of the puck for 15 test rollouts, reporting mean and standard deviation.

2.5.1 *Visual RL with Physical Robots*

RIG is a practical and straightforward algorithm to apply to real physical systems: the efficiency of off-policy learning with goal relabeling makes training times manageable, while the use of image-based rewards through the learned representation frees us from the burden of manually design reward functions, which itself can require hand-engineered perception systems (Rusu et al., 2017). We trained policies for visual reaching and pushing on a real-world Sawyer robotic arm, shown in Figure 7. The control setup matches Visual Reacher and Visual Pusher respectively, meaning that **the only input from the environment consists of camera images**.

We see in Figure 7 that our method is applicable to real-world robotic tasks, almost matching the state-based oracle method and far exceeding the baseline method on the reaching task. Our method needs just 10,000 samples or about an hour of real-world interaction time to solve visual reaching.

Real-world pushing results are shown in Figure 12. To solve visual pusher, which is more visually complicated and requires reasoning about the contact between the arm and object, our method requires about 25,000 samples, which is still a reasonable amount of real-world training time. Note that unlike previous results, we do not have access to the true puck position during training so for the learning curve we report test episode returns on the VAE latent distance reward. We see RIG making steady progress at optimizing the latent distance as learning proceeds.

2.6 DISCUSSION AND FUTURE WORK

In this paper, we present a new RL algorithm that can efficiently solve goal-conditioned, vision-based tasks without access to any ground truth state or reward functions. Our method trains a generative model that is used for multiple purposes: we embed the state and goals using the encoder; we sample from the prior to generate goals for exploration; we also sample latents to retroactively relabel goals and rewards; and we use distances in the latent space for rewards to train a goal-conditioned value function. We show that these components culminate in a sample efficient algorithm that works directly from vision. As a result, we are able to apply our method to a variety of simulated visual tasks, including a variable-object task that cannot be easily represented with a fixed length vector, as well as real world robotic tasks. Algorithms that can learn in the real world and directly use raw images can allow a single policy to solve a large and diverse set of tasks, even when these tasks require distinct internal representations.

The method we presented can be extended in a number of ways. First, an exciting line of future work would be to combine our method with existing work on exploration and intrinsic motivation. In particular, our method already provides a natural mechanism for autonomously generating goals by sampling from the prior. Modifying this procedure to not only be goal-oriented but also, e.g., be information seeking or uncertainty aware could provide better and safer exploration. Second, since our method operates directly from images, a single policy could potentially solve a large diverse set of visual tasks, even if those tasks have different underlying state representations. Combining these ideas with methods from multitask learning and meta-learning is a promising path to creating general-purpose agents that can continuously and efficiently acquire skills. Lastly, while RIG uses goal images, extending the method to allow goals specified by demonstrations or more abstract representations such as language would enable our system to be much more flexible in interfacing with humans and therefore more practical.

2.7 CONTRIBUTION STATEMENT

The work in this chapter was performed in collaboration with Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine (A. Nair et al., 2018b). A.N. and V.P. were joint first co-authors and co-led all aspects of the project, including prototyping the initial idea, project planning, experimental design, experiments, and writing. The first five authors conducted the simulation experiments. A.N. and S.B. conducted the real-world experiments. Sergey Levine advised the project and assisted with writing.

3

CONTEXTUAL IMAGINED GOALS FOR SELF-SUPERVISED ROBOTIC LEARNING

3.1 INTRODUCTION

In the previous chapter, we discussed letting an agent learn from its sensor stream by automatically generating plausible goals during an unsupervised training phase, and then learning policies that reach those goals (A. Nair et al., 2018b; Nachum et al., 2018; Warde-Farley et al., 2019; Vitchyr H. Pong et al., 2019). Such goals can be defined in a variety of ways, but a simple choice is to use goal observations, such that each proposed task requires reaching a different observation. When the robot observes the world via raw camera images, this corresponds to using images as goals. At test time, a user then provides the robot with a new goal image.

While such methods have been demonstrated in both simulated and real-world settings, they are typically used to learn behaviors in domains with relatively little visual diversity. In the real world, a robot might interact with highly diverse scenes and objects, and the tasks that it can perform from each of the many possible initial states will be different. If the robot is presented with an object, it can learn to pick up or grasp it, and when it is presented with a door, it can learn to open it. However, it must generate and practice goals that are suitable for each scene. In this chapter, we propose and evaluate a self-supervised policy learning method that learns to propose goals that are suitable to the current scene via a conditional goal generation model, allowing it to learn in visually varied settings that prove challenging for prior algorithms.

The key idea is that representing every element in a visually complex scene is often not necessary for control. A scene is a visual form of context that can be factored out, while only the controllable entities in the environment need to be captured for goal setting and representing the state. To this end, we propose learning a context-conditioned generative model that learns a smooth, compressed latent variable with an information bottleneck,

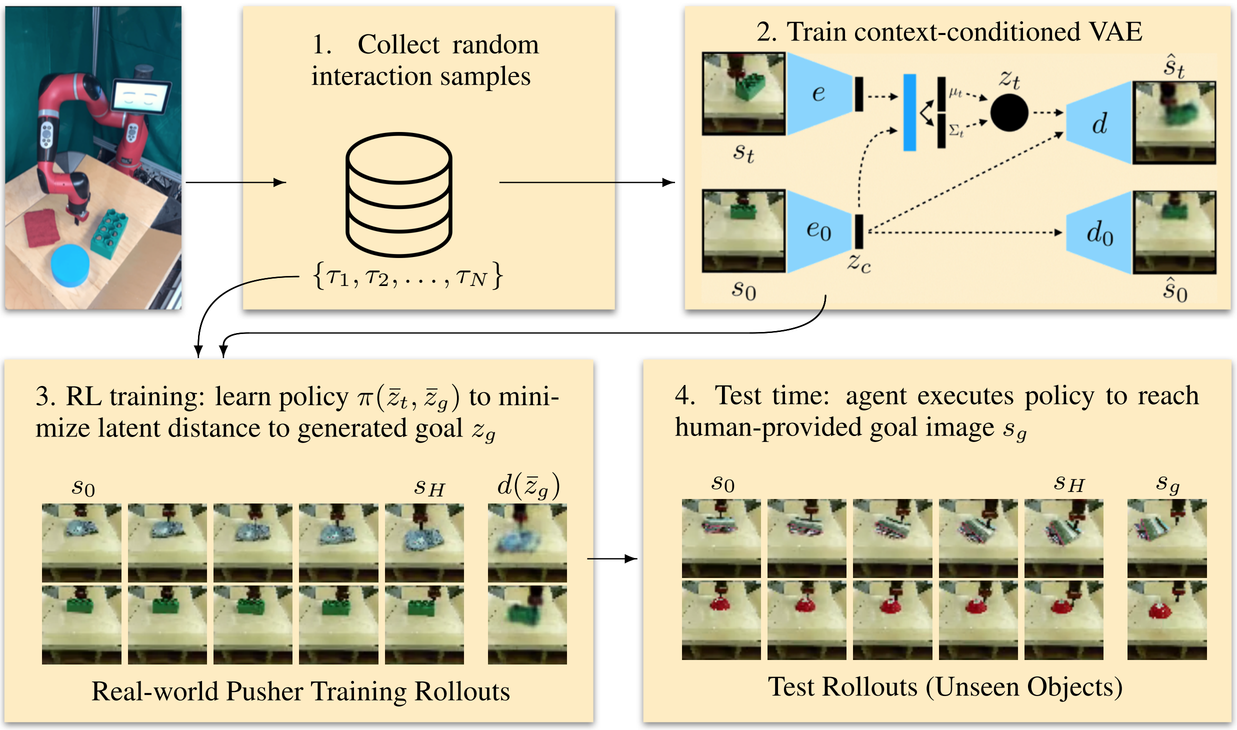

1. Collect random interaction samples
$\{\tau_1, \tau_2, \ldots, \tau_N\}$
2. Train context-conditioned VAE
s_t
e
μ_t
Σ_t
z_t
d
$\hat{s}_t$
s_0
e_0
z_c
d_0
$\hat{s}_0$
3. RL training: learn policy $\pi(\bar{z}_t, \bar{z}_g)$ to minimize latent distance to generated goal z_g
s_0
s_H
$d(\bar{z}_g)$
Real-world Pusher Training Rollouts
4. Test time: agent executes policy to reach human-provided goal image s_g
s_0
s_H
s_g
Test Rollouts (Unseen Objects)

Figure 9: System overview of our self-supervised learning algorithm. (1) The agent collects random interaction data, to be used for both representation learning and as additional off-policy data for RL. (2) We propose a context-conditioned generative model (CC-VAE) to learn generalizable skills. In order to improve the generation of plausible goal states that result from a starting state s_0, our model allows information to flow freely through the context z_c while an information bottleneck on z_t gives us the ability to generate samples by re-sampling only z_t. This architecture provides a compact representation of the scene disentangling information that changes within a rollout (z_t) and information that changes between rollouts (z_c). We then use $\bar{z}_t = (z_t, z_c)$ as the representation for RL. (3) Our proposed CC-RIG algorithm samples latent goals, using the above representation, and learns a policy to minimize the latent distance to the goal with off-policy RL. Rollouts are shown on the real-world Sawyer robot pusher environment with visual variation. We include the initial image s_0, selected frames from the rollout, final image s_H, and the decoded goal latent $d(\bar{z}_g)$. (4) At test time, the agent is given a goal image s_g and executes the policy to reach it. Our method successfully handles pushing novel objects that were unseen at training time. Example rollouts can be found at https://ccrig.github.io/

while allowing the context, in the form of the initial state image, to be used freely to reconstruct other images during the task. This context-conditioned generative model architecture is shown in Figure 47.

The main contribution in this chapter builds on this context-conditioned generative model to devise a complete self-supervised goal-conditioned reinforcement learning algorithm, which can handle visual variability in the scene via context-conditioned goal setting. Our method can learn policies that reach visually indicated goals without any additional supervision during training, using the context-conditioned generative model to set goals that are appropriate to the current scene. We show that our approach learns coherent representations of visually varied environments, capturing controllable dimensions of variation while ignoring dimensions that vary but cannot be influenced by the agent, such as lighting and object appearance. We further show that our approach can learn policies to solve tasks in visually varied environments, including in a real-world robotic pushing task with a wide variety of distinct objects in Figure 12.

3.2 RELATED WORK

While many practical robots today perform tasks by executing hand-engineered sequences of motor commands, machine learning is opening up a new avenue to train a wide variety of robotic tasks from interaction. This body of work includes grasping (Ekvall and Kragic, 2004; Kroemer et al., 2010; Bohg and Kragic, 2010), and general tasks (Peters and Schaal, 2008b; Kober et al., 2013), multi-task learning (M. P. Deisenroth et al., 2014), baseball (Peters and Schaal, 2008c), ping-pong (Peters et al., 2010), and various other tasks (Marc Peter Deisenroth and Rasmussen, 2011). More recently, using expressive function approximators such as neural networks has reduced manual feature engineering and has increased task complexity and diversity, finding use in decision-making domains, such as solving Atari games (Mnih et al., 2013) and Go (D. Silver et al., 2016a). Deep learning for robotics has proved to be difficult due to a host of challenges including noisy state estimation, specifying reward functions, and handling continuous action spaces, but has been used to investigate grasping (Pinto and Abhinav Gupta, 2016), pushing (Agrawal et al., 2016), manipulation of 3D object models (Krainin et al., 2011), active learning (Martinez et al., 2014) and pouring liquids (Schenck and Fox, 2017). Deep reinforcement learning, which autonomously maximizes a given reward function, has been used to solve precise manipulation tasks (Levine et al., 2016a), grasping (Pinto et al., 2017; Levine et al., 2017), door opening (S. Gu et al., 2016), and navigation (Kahn et al., 2018). These methods have succeeded on specific tasks, often with hard-coded reward functions. However, to scale task generalization robots may need to learn methods that can handle significant environment variation and require relatively little external supervision.

Several works have investigated self-supervised robotic interaction with varied objects in the deep learning setting with the goal of generalizing between objects. For example, in the domain of robotic grasping, several works have studied autonomous data collection to learn to grasp from a hand-specified grasping reward (Pinto and Abhinav Gupta, 2016; Levine et al., 2017). However, hand-specifying such rewards in general settings and for arbitrary manipulation skills is very cumbersome. Other work has focused on self-supervised learning with visual forward models, either by enforcing a simplified dynamical structure (Watter et al., 2015; M. Zhang et al., 2019) or with pixel transformer architectures (Finn and Levine, 2016; Ebert et al., 2017; Ebert et al., 2018; A. X. Lee et al., n.d.; Ebert et al., n.d.). However, these methods rely on accurate visual forward modelling, which is itself a very challenging problem. Instead, we build on self-supervised model-free approaches, which allow the agent to efficiently reach visual goals without planning with a visual forward model.

Prior work has also sought to perform self-supervised learning with model-free approaches. Using visual inverse models (Agrawal et al., 2016) is one such approach, but may not work well for complex interaction dynamics or longer horizon planning. Most closely related to our approach are prior methods on goal-conditioned reinforcement learning (L P Kaelbling, 1993; Schaul et al., 2015b; Andrychowicz et al., 2017b). The methods have been extended to frame self-supervised RL as learning goal reaching with automatically proposed goals, including visually-specified goals (A. Nair et al., 2018b; Vitchyr H. Pong et al., 2019; Warde-Farley et al., 2019; Florensa et al., 2019; Lin et al., 2019). However, they generally focus on learning in narrow environments with little between-trial variability. In this setting, any previously visited state represents a valid goal. However, in the general case, this is no longer true: when the robot is presented with a different scene or different objects on each trial, it must only set those goals that can be accomplished in the current scene. In contrast, we focus on enabling self-supervised learning from off-policy data in heterogeneous environments with increased factors of variability.

3.3 BACKGROUND

3.3.1 *Conditional Variational Auto-Encoders*

In the previous chapter, we discussed handling high-dimensional goals by learning a latent representation of the state using a variational auto-encoder (VAE). Instead of a generative model that learns to generate the dataset distribution, one might instead desire a more structured generative model that can generate samples based on structured input. One example of this is a conditional variational auto-encoder (CVAE) that conditions the output on some input variable c and samples from $p(x|c)$ (Sohn et al., 2015). For example, to train a model that generates images of digits given the desired digit, the input variable c might be a one-hot encoded vector of the desired digit.

A CVAE trains $q_\phi(z|s,c)$ and $q_\psi(s|z,c)$, where both the encoder and decoder has access to the input variable c. The CVAE then minimizes:

$$\mathcal{L}_{\text{CVAE}} = -\mathbb{E}_{q_\phi(z|s,c)}[\log p(s|z,c)] + \beta D_{\text{KL}}(q_\phi(z|s,c)||p(z)). \tag{6}$$

Samples are generated by first sampling a latent $z \sim p(z)$. Based on c, we can then decode z with $q_\psi(s|z,c)$ and visualize the output, which is in our case an image. In our framework $c = s_0$.

3.4 SELF-SUPERVISED LEARNING WITH CONTEXT-CONDITIONED REPRESENTATIONS

In this work, our goal is to enable the learning of flexible goal-conditioned policies that can be used to successfully perform a variety of tasks in a variety of contexts – e.g., with different objects in the scene. Such policies must learn from large amounts of experience, and although it is in principle possible to use random exploration to collect this experience, this quickly becomes impractical in the real world. It is, therefore, necessary for the robot to set its own goals during self-supervised training, to collect meaningful experience. However, in diverse settings, many randomly generated goals may not be feasible – e.g., the robot cannot push a red puck if the red puck is not present in the scene. We propose to extend off-policy goal-conditioned reinforcement learning with a conditional goal setting model, which proposes only those goals that are currently feasible. This enables a learning regime with imagined goals that is more realistic for real-world robotic systems that must generalize effectively to a range of objects and settings.

3.4.1 *Context-Conditioned VAEs*

To train a generative model that can improve the generation of feasible goals in varied scenes, we use a modified CVAE that uses the initial state s_0 in a rollout as the input c, which we call the "context" for that rollout. The modified CVAE, which we call a context-conditioned VAE (CC-VAE), is shown in Figure 47. While most CVAE applications use a one-hot vector as the input, we use an image s_0. This image is encoded with a convolutional encoder e_0 into a compact representation z_c. Note that by design, e and e_0 do not share weights, as they are intended to encode different factors of variation in the images. The context z_c is used to output the latent representation z_t, as well as the reconstruction of the state $\hat{s}_t$. In addition, z_c is used alone to (deterministically) decode $\hat{s}_0 = d_0(z_c)$. The objective is given by

$$\mathcal{L}_{\text{CC-VAE}} = \mathcal{L}_{\text{CVAE}} + \log p(s_0|z_c). \tag{7}$$

Due to the information bottleneck on z_t, this loss function penalizes information passing through z_t but allows for unrestricted information flow from z_c. Therefore, the optimal solution would encode as much information as possible in z_c, while only including the state information that changes within a trajectory in the latent variable z_t. These are precisely the features of most interest for control.

3.4.2 *Context-Conditioned Reinforcement Learning with Imagined Goals*

We propose to use our context-conditioned VAE in the RIG framework to learn policies over environments with visual diversity, where each episode might involve interacting with a different scene and different objects. We first collect a dataset of trajectories $\mathcal{D} = \{\tau^{(i)}\}$ by executing random actions in the environment. We then learn a CC-VAE, as detailed in Section 3.4.1, to learn a factored representation of the image observations. To use the CC-VAE for self-supervised learning, we save the first image s_0 when starting a rollout. We compute the encoding of s_0, $z_c = e_0(s_0)$. Let $\bar{z}$ denote the context concatenated vector (z, z_c), and let $\mu(s, s_0)$ denote the mean of $q_\phi(z|s, s_0)$. We then use RIG in the $\bar{z}$ latent space by encoding observations with μ, meaning that we train a goal-conditioned policy $\pi(\bar{z}, \bar{z_g})$ and a goal-conditioned Q-function $Q(\bar{z}, \bar{z_g})$.

To collect data, we sample a latent goal for each rollout from the prior $z_g \sim N(0, I)$, as in RIG. For every observation s_t, we compute the mean encoding $\mu_t(s_t, s_0)$. We then obtain a rollout of the policy by executing $\pi(\bar{z}, \bar{z_g})$. The reward at each timestep is the latent distance $||\bar{\mu}_t - \bar{z}_g||$.

Algorithm 2 Context-Conditioned RIG

```
Require: Encoders $\mu(s_t, s_0)$, $e_0(s_0)$, policy $\pi_\theta(\bar{z}, \bar{z_g})$, goal-conditioned value function $Q_w(\bar{z}, \bar{z_g})$, dataset $\mathcal{D} = \{\tau^{(i)}\}$ of trajectories.
 1: Train CC-VAE on $\mathcal{D}$ by optimizing equation 7.
 2: for $n = 0, ..., N-1$ episodes do
 3:     Sample latent goal $z_g \sim p(z)$, $\bar{z}_g = (z_g, z_c)$.
 4:     Sample initial state $s_0 \sim p(s_0)$.
 5:     Encode $z_c = e_0(s_0)$
 6:     for $t = 0, ..., H-1$ steps do
 7:         Observe $s_t$ and encode $\bar{z}_t = (\mu(s_t, s_0), z_c)$
 8:         Get action $a_t = \pi_\theta(\bar{z}_t, \bar{z}_g) + $ noise.
 9:         Get next state $s_{t+1} \sim p(\cdot \mid s_t, a_t)$.
10:         Store $(\bar{z}_t, a_t, \bar{z}_{t+1}, \bar{z}_g)$ into replay buffer $\mathcal{R}$.
11:         Sample transition $(\bar{z}, a, \bar{z}', \bar{z}_g) \sim \mathcal{R}$.
12:         Compute new reward $r = -\|\bar{z}' - \bar{z}_g\|$.
13:         Minimize equation 24 using $(\bar{z}, a, \bar{z}', \bar{z}_g, r)$.
14:     end for
15: end for=0
```

The policy and Q-function can be trained with any off-policy reinforcement learning algorithm. We use TD3 in our implementation (Fujimoto et al., 2018b). Our policy and Q-function are goal-conditioned, and we take advantage of being able to relabel the goals for each transition to improve sample efficiency (Andrychowicz et al., 2017b; A. Nair et al., 2018b; Vitchyr H. Pong et al., 2019). However, when relabeling a goal $\bar{z}_g$ with a random goal from the environment, the context-conditioning is still preserved. That is, if $z'_g \sim N(0, 1)$ is the new sampled goal, we use $\bar{z}'_g = (z'_g, z_c)$. This ensures that the relabeled goal is compatible with the scene for the corresponding transition.

After training, we can use the learned policy π to reach a visually indicated goal. Given a goal image s_g, we encode it into a latent goal $z_g = \mu(s_g, s_0)$. Then, we execute the policy with the latent goal $\bar{z}_g$, just as during the training phase. The complete algorithm is presented in Algorithm 2.

3.5 EXPERIMENTS

In our experiments, we aim to answer the following questions:

1. How does our method compare to prior work at learning self-supervised skills in visually diverse environments?

2. Do context-conditioned VAEs learn an image representation that produces coherent and diverse goals that are suitable for the current scene?
3. Can our proposed context-conditioned RIG method handle diverse real-world data and learn effective policies under visual variation in the real world?

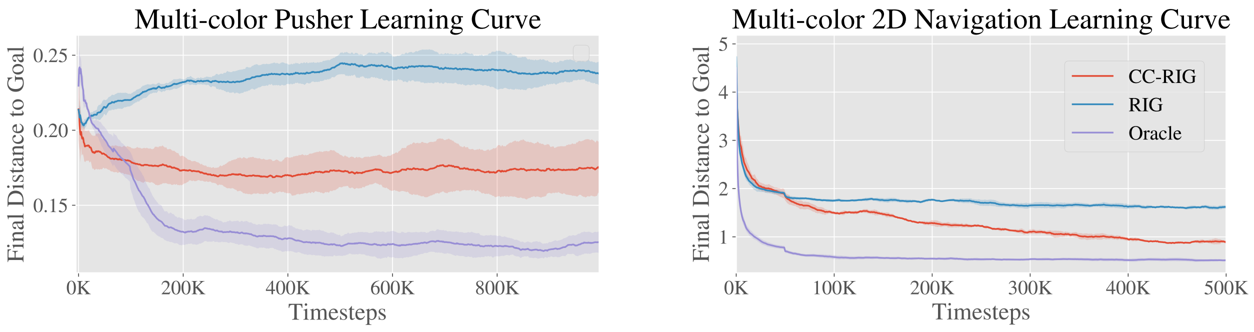


Figure 10: Self-supervised learning results in visually varied simulated environments. CC-RIG significantly outperforms RIG and is competitive with the oracle method that has direct access to ground truth states. The simulated pusher environment (left) is shown in Figure 11 and the navigation environment is shown in Figure 65.

3.5.1 *Self-Supervised Learning in Simulation*

In simulation, we can conduct controlled experiments and evaluate against known underlying state values to measure the performance of our approach and prior methods. As a simulation test-bed, we use a multi-color pusher environment simulated in MuJoCo (Todorov et al., 2012). In this environment (Figure 11(left)), a simulated Sawyer arm is tasked with pushing a circular puck to a target position, specified by a goal image at test time. On each rollout, the puck color is set to a random RGB value. Therefore, the goal proposals for each method must adequately account for the color of the puck – a goal that requires moving a red puck to a given location is impossible if only a blue puck is present in the scene.

We compare the following algorithms: **CC-RIG.** Our method using a CC-VAE for representation learning, as described in Section 3.4.2. **RIG.** Reinforcement learning with imagined goals (A. Nair et al., 2018b) using a standard VAE, as described in Section **??**. **Oracle.** The oracle agent runs goal-conditioned RL with direct access to state information. Achieving performance similar to the oracle indicates that an algorithm loses little from using raw image observations over ground truth state.

Learning curves comparing these methods are presented in the plot on the left in Figure 73. CC-RIG outperforms RIG significantly, and standard RIG is not able to improve

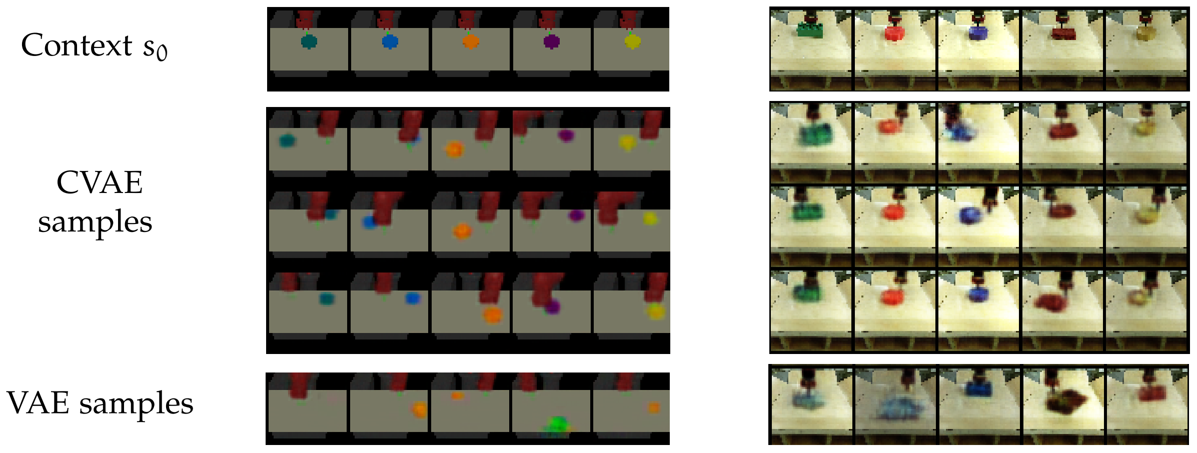


Figure 11: Comparing samples from our CC-VAE model to a standard VAE. The initial image s_0 is shown on the top row, and samples conditioned on s_0 are shown below. Our model coherently maintains object color and geometry in its samples, suggesting that the context conditioned model can successfully factor out the scene-specific object identity from the variable object position. This enables the use of the CC-VAE for goal proposals in visually diverse scenes.

beyond the initial random policy. The performance of CC-RIG approaches that of the oracle policy, which has access to the true state. This suggests that, in visually varied environments, self-supervised learning is possible so long as the visual complexity is factored out with representation learning, and the proposed goals are consistent with the appearance of the current scene.

3.5.2 *Generalizing to Varying Appearance and Dynamics with Self-Supervised Learning*

In this experiment, to study changing both visual appearance and physical dynamics, we study how well our method can generalize when the environment dynamics change. We use a simulated 2D navigation task, where the goal is to navigate a point robot around an obstacle. The arrangement of the obstacles is chosen from a set of 15 possible configurations, and the color of the point robot is generated from a random RGB value. Learning curves obtained by training the different methods above in this environment are presented in Figure 73. CC-RIG requires more samples to learn, but eventually approaches the oracle performance. RIG, in comparison, plateaus with poor performance. This environment is explained further in the supplementary, in Section B.1 and Figure 65.

3.5.3 *Context-Conditioned VAE Goal Sampling*

To better understand why CC-RIG outperforms RIG, we compare the samples from our CC-VAE to a standard VAE. Samples from both models are shown in Figure 11. The quality of the samples reveals why the CC-VAE provides better goal setting for self-supervised learning. In all environments, the samples from the CC-VAE maintain the background, object shape, and object color from the initial state. Therefore, the goals are more meaningful in the CC-VAE latent space.

This kind of visualization is a good indicator for the suitability of the representation for self-supervised learning. Diverse, coherent samples indicate that the latent space captures the appropriate factors of change in the environment and can be useful for self-supervised policy learning. Good samples also suggest that the latent space is well-structured, and therefore distances in the latent space should provide a good reward function for goal-reaching. In practice, we also look at the quality of the reconstructions. Good reconstructions confirm that the latent variables capture sufficient information about the image to be used in place of the image itself as a state representation.

3.5.4 *Real-World Robotic Evaluation*

In this experiment, we evaluate whether our method can handle manipulating visually varied objects in the real world. We use CC-RIG to train a Sawyer robot to manipulate a variety of objects, placed one at a time in a bin in front of the robot. As before, the training phase is self-supervised, and the robot must match a given goal image at test time. The robot setup is shown in Figure 47.

We first collect a large dataset with random actions and train a CC-VAE on the data. Samples from the model are shown in Figure 12. The CC-VAE learns to generate goals with the correct object. To handle varying brightness at different times of the day, we added data augmentation by applying a color jitter filter to (s_0, s_t) pairs. As seen in the figure, the model is robust to this factor of variation. Each sample contains the same type of object, brightness level, and background as the initial state that it is conditioned on. However, crucially, these factors of variation are not present in z_t, as evidenced by the fact they do not vary within each column of Figure 12, but the object position does.

Next, we run CC-RIG with the trained CC-VAE to learn to reach visually indicated goals in a self-supervised manner. We first conduct fully off-policy training using the same dataset as was used to train the CC-VAE, consisting of 50,000 samples (about 3 hours) of total interaction with 20 objects. Then, we collect a small amount of additional

Real-World Pushing Results

	CVAE distance	VAE distance	Pixel distance	Object distance (cm)
RIG	2.37 ± 0.97	2.41 ± 0.93	93.9 ± 41.7	17.1 ± 8.2
CC-RIG	1.66 ± 0.63	2.17 ± 0.88	56.8 ± 34.5	14.0 ± 6.9
CC-RIG, novel	1.51 ± 0.71	1.91 ± 0.87	53.1 ± 24.9	11.5 ± 2.9

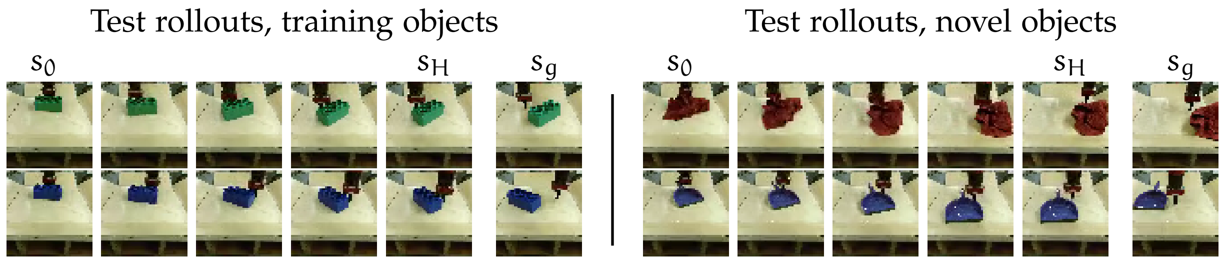


Figure 12: The table above shows the performance of our method in the real-world, evaluated with four different evaluation metrics[1]. CC-RIG outperforms RIG in each one, even when tested on novel objects that it has not been trained on. Test rollouts of our method are shown on training objects on the left and unseen novel objects on the right. Successful rollouts where the object is pushed to the goal location are shown in top row, and failure modes are shown in the bottom row.

on-policy data to finetune the policy, analogous to recent work on large-scale vision-based robotic reinforcement learning (Kalashnikov et al., n.d.). The robot learns to push objects to target locations, indicated by a goal image. The real-world results are presented in Figure 12. Because it is difficult to automatically detect the positions of objects, we show some representative rollout examples, and we compute several distance metrics between the final state of a rollout and the goal: **CVAE distance.** CC-VAE latent space distance between final image and goal. **VAE distance.** VAE latent space distance between final image and goal. **Pixel distance.** We manually label the center of mass of the object in the final image and goal image, and compute the distance between them. **Object distance.** We measure the distance between the physical goal position of the object and the final position. In each metric, CC-RIG outperforms RIG.

At training time, the dataset consists of interaction with 20 objects. The result of running CC-RIG on novel objects that were not included in the dataset are shown in the table as "CC-RIG, novel" and in the rollouts in Figure 12. These results show that our method can also generalize its experience to push novel objects it has not seen before.

3.6 Conclusion

We presented a method for sample-efficient, flexible self-supervised task learning for environments with visual diversity. Our method can learn effective behavior without external supervision in simulated environments with randomized colors and layout, and in a real-world pushing task with differently colored pucks. Each environment contains an axis of visual variation that requires our algorithm to utilize an intelligent goal-setting strategy, to ensure that the self-proposed goals are consistent with the tasks and feasible in the current scene.

The main idea behind our method is to devise a context-conditioned goal proposal mechanism, allowing our self-supervised reinforcement learning algorithm to propose goals for itself that are feasible to reach. This context-conditioned VAE model factors out the unchanging context of a rollout, such as which objects are present in the scene, from the controllable aspects, such as the object positions to construct a more generalizable goal proposal model.

We believe this contribution will enable scalable learning in the real world. An agent manipulating objects in the real world must handle many forms of variation: different manipulation skills to learn, objects to manipulate, as well as variation in lighting, textures, etc. Methods that learn from data must be able to represent these variations while at the same time taking advantage of common structure across objects and tasks in order to achieve practical sample efficiency. Future work will address the remaining challenges to achieve this vision.

3.7 Contribution statement

The work in this chapter was performed in collaboration with Shikhar Bahl, Alexander Khazatsky, Vitchyr Pong, Glen Berseth, and Sergey Levine (A. Nair et al., 2018b). A.N., S.B., and A.K. were joint co-first authors. A.N. proposed the context-conditioned VAE, led the project, designed the experiments, and wrote the paper. A.N, S.B, and A.K. conducted the simulation experiments. A.K. conducted the real-world experiments with assistance from A.N. V.P., G.B., and S.L. advised the project and assisted with writing.

1 The first three metrics are computed on 40 trajectories per method, and we report mean ± standard deviation. Object distance is computed on 10 trajectories per method, and we report median ± standard deviation.

4

SKEWFIT: STATE-COVERING SELF-SUPERVISED REINFORCEMENT LEARNING

4.1 INTRODUCTION

While reinforcement learning (RL) provides an appealing formalism for automated learning of behavioral skills, separately learning every potentially useful skill becomes prohibitively time consuming, both in terms of the experience required for the agent and the effort required for the user to design reward functions for each behavior. In the previous chapters, we covered self-supervised goal-conditioned reinforcement learning with a generative model as an approach to scalable multi-task learning with little human supervision. This approach however, crucially relies on the distribution of data that the generative model is trained on. What if we could instead design an unsupervised RL algorithm that automatically explores the environment and iteratively distills this experience into general-purpose policies that can accomplish new user-specified tasks at test time?

In the absence of any prior knowledge, an effective exploration scheme is one that visits as many states as possible, allowing a policy to autonomously prepare for user-specified tasks that it might see at test time. We can formalize the problem of visiting as many states as possible as one of maximizing the *state entropy* $\mathcal{H}(\mathbf{S})$ under the current policy.[2] Unfortunately, optimizing this objective alone does not result in a policy that can solve new tasks: it only knows how to maximize state entropy. In other words, to develop principled unsupervised RL algorithms that result in useful policies, maximizing $\mathcal{H}(\mathbf{S})$ is not enough. We need a mechanism that allows us to reuse the resulting policy to achieve new tasks at test-time.

We argue that this can be accomplished by performing *goal-directed exploration*: a policy

2 We consider the distribution over terminal states in a finite horizon task and believe this work can be extended to infinite horizon stationary distributions.

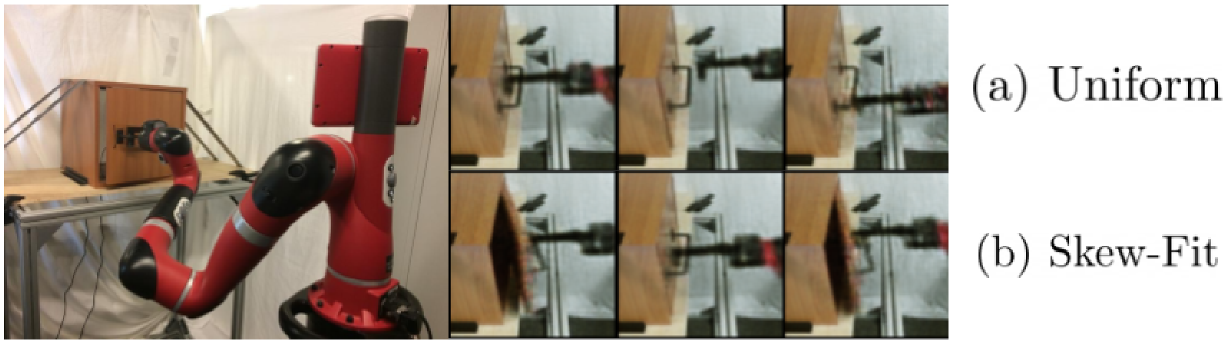


Figure 13: Left: Robot learning to open a door with Skew-Fit, without any task reward. Right: Samples from a goal distribution when using (a) uniform and (b) Skew-Fit sampling. When used as goals, the diverse samples from Skew-Fit encourage the robot to practice opening the door more frequently.

should autonomously visit as many states as possible, but after autonomous exploration, a user should be able to reuse this policy by giving it a goal $\mathbf{G}$ that corresponds to a state that it must reach. While not all test-time tasks can be expressed as reaching a goal state, a wide range of tasks can be represented in this way. Mathematically, the goal-conditioned policy should minimize the conditional entropy over the states given a goal, $\mathcal{H}(\mathbf{S} \mid \mathbf{G})$, so that there is little uncertainty over its state given a commanded goal. This objective provides us with a principled way to train a policy to explore all states (maximize $\mathcal{H}(\mathbf{S})$) such that the state that is reached can be determined by commanding goals (minimize $\mathcal{H}(\mathbf{S} \mid \mathbf{G})$).

Directly optimizing this objective is in general intractable, since it requires optimizing the entropy of the marginal state distribution, $\mathcal{H}(\mathbf{S})$. However, we can sidestep this issue by noting that the objective is the mutual information between the state and the goal, $I(\mathbf{S}; \mathbf{G})$, which can be written as:

$$\mathcal{H}(\mathbf{S}) - \mathcal{H}(\mathbf{S}|\mathbf{G}) = I(\mathbf{S}; \mathbf{G}) = \mathcal{H}(\mathbf{G}) - \mathcal{H}(\mathbf{G}|\mathbf{S}). \tag{8}$$

Equation 8 thus gives an equivalent objective for an unsupervised RL algorithm: the agent should set diverse goals, maximizing $\mathcal{H}(\mathbf{G})$, and learn how to reach them, minimizing $\mathcal{H}(\mathbf{G} \mid \mathbf{S})$.

While learning to reach goals is the typical objective studied in goal-conditioned RL (L P Kaelbling, 1993; Andrychowicz et al., 2017b), setting goals that have maximum diversity is crucial for effectively learning to reach all possible states. Acquiring such a maximum-entropy goal distribution is challenging in environments with complex, high-dimensional state spaces, where even knowing which states are valid presents a major challenge. For example, in image-based domains, a uniform goal distribution requires sampling uniformly from the set of realistic images, which in general is unknown a priori.

In this chapter, we present the following contributions. First, we propose a principled objective for unsupervised RL, based on Equation 8. While a number of prior works ignore the $\mathcal{H}(\mathbf{G})$ term, we argue that jointly optimizing the entire quantity is needed to develop effective exploration. Second, we present a general algorithm called Skew-Fit and prove that under regularity conditions Skew-Fit learns a sequence of generative models that converges to a uniform distribution over the goal space, even when the set of valid states is unknown (e.g., as in the case of images). Third, we describe a concrete implementation of Skew-Fit and empirically demonstrate that this method achieves state of the art results compared to a large number of prior methods for goal reaching with visually indicated goals, including a real-world manipulation task, which requires a robot to learn to open a door from scratch in about five hours, directly from images, and without any manually-designed reward function.

4.2 PROBLEM FORMULATION

To ensure that an unsupervised reinforcement learning agent learns to reach all possible states in a controllable way, we maximize the mutual information between the state $\mathbf{S}$ and the goal $\mathbf{G}$, $I(\mathbf{S};\mathbf{G})$, as stated in Equation 8. This section discusses how to optimize Equation 8 by splitting the optimization into two parts: minimizing $\mathcal{H}(\mathbf{G} \mid \mathbf{S})$ and maximizing $\mathcal{H}(\mathbf{G})$.

4.2.1 *Minimizing $\mathcal{H}(\mathbf{G} \mid \mathbf{S})$: Goal-Conditioned Reinforcement Learning*

Standard RL considers a Markov decision process (MDP), which has a state space $\mathcal{S}$, action space $\mathcal{A}$, and unknown dynamics $\rho(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t) : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \mapsto [0, +\infty)$. Goal-conditioned RL also includes a goal space $\mathcal{G}$. For simplicity, we will assume in our derivation that the goal space matches the state space, such that $\mathcal{G} = \mathcal{S}$, though the approach extends trivially to the case where $\mathcal{G}$ is a hand-specified subset of $\mathcal{S}$, such as the global XY position of a robot. A goal-conditioned policy $\pi(\mathbf{a} \mid \mathbf{s}, \mathbf{g})$ maps a state $\mathbf{s} \in \mathcal{S}$ and goal $\mathbf{g} \in \mathcal{S}$ to a distribution over actions $\mathbf{a} \in \mathcal{A}$, and its objective is to reach the goal, i.e., to make the current state equal to the goal.

Goal-reaching can be formulated as minimizing $\mathcal{H}(\mathbf{G} \mid \mathbf{S})$, and many practical goal-reaching algorithms (L P Kaelbling, 1993; Lillicrap et al., 2016; Schaul et al., 2015b; Andrychowicz et al., 2017b; A. Nair et al., 2018b; V. Pong et al., 2018; Florensa et al., 2018a) can be viewed as approximations to this objective by observing that the optimal goal-conditioned policy will deterministically reach the goal, resulting in a conditional entropy of zero: $\mathcal{H}(\mathbf{G} \mid \mathbf{S}) = 0$. See Section C.5 for more details. Our method may thus be used in conjunction with any of these prior goal-conditioned RL methods in order to jointly minimize $\mathcal{H}(\mathbf{G} \mid \mathbf{S})$ and maximize $\mathcal{H}(\mathbf{G})$.

4.2.2 *Maximizing $\mathcal{H}(\mathbf{G})$: Setting Diverse Goals*

We now turn to the problem of setting diverse goals or, mathematically, maximizing the entropy of the goal distribution $\mathcal{H}(\mathbf{G})$. Let $U_{\mathcal{S}}$ be the uniform distribution over $\mathcal{S}$, where we assume $\mathcal{S}$ has finite volume so that the uniform distribution is well-defined. Let q_{ϕ}^{G} be the goal distribution from which goals $\mathbf{G}$ are sampled, parameterized by ϕ. Our goal is to maximize the entropy of q_{ϕ}^{G}, which we write as $\mathcal{H}(\mathbf{G})$. Since the maximum entropy distribution over $\mathcal{S}$ is the uniform distribution $U_{\mathcal{S}}$, maximizing $\mathcal{H}(\mathbf{G})$ may seem as simple as choosing the uniform distribution to be our goal distribution: $q_{\phi}^{G} = U_{\mathcal{S}}$. However, this requires knowing the uniform distribution over valid states, which may be difficult to obtain when $\mathcal{S}$ is a subset of $\mathbb{R}^n$, for some n. For example, if the states correspond to images viewed through a robot's camera, $\mathcal{S}$ corresponds to the (unknown) set of valid images of the robot's environment, while $\mathbb{R}^n$ corresponds to all possible arrays of pixel values of a particular size. In such environments, sampling from the uniform distribution $\mathbb{R}^n$ is unlikely to correspond to a valid image of the real world. Sampling uniformly from $\mathcal{S}$ would require knowing the set of all possible valid images, which we assume the agent does not know when starting to explore the environment.

While we cannot sample arbitrary states from $\mathcal{S}$, we can sample states by performing goal-directed exploration. To derive and analyze our method, we introduce a simple model of this process: a goal $\mathbf{G} \sim q_{\phi}^{G}$ is sampled from the goal distribution q_{ϕ}^{G}, and then the goal-conditioned policy π attempts to achieve this goal, which results in a distribution of terminal states $\mathbf{S} \in \mathcal{S}$. We abstract this entire process by writing the resulting marginal distribution over $\mathbf{S}$ as $p_{\phi}^{S}(\mathbf{S}) \triangleq \int_{\mathcal{G}} q_{\phi}^{G}(\mathbf{G}) p(\mathbf{S} \mid \mathbf{G}) d\mathbf{G}$, where the subscript ϕ indicates that the marginal p_{ϕ}^{S} depends indirectly on q_{ϕ}^{G} via the goal-conditioned policy π. We assume that p_{ϕ}^{S} has full support, which can be accomplished with an epsilon-greedy goal reaching policy in a communicating MDP. We also assume that the entropy of the resulting state distribution $\mathcal{H}(p_{\phi}^{S})$ is no less than the entropy of the goal distribution

$\mathcal{H}(q_\phi^G)$. Without this assumption, a policy could ignore the goal and stay in a single state, no matter how diverse and realistic the goals are. [3] This simplified model allows us to analyze the behavior of our goal-setting scheme separately from any specific goal-reaching algorithm. We will however show in Section 9.5 that we can instantiate this approach into a practical algorithm that jointly learns the goal-reaching policy. In summary, our goal is to acquire a maximum-entropy goal distribution q_ϕ^G over valid states $\mathcal{S}$, while only having access to state samples from p_ϕ^S.

4.3 Skew-Fit: Learning a Maximum Entropy Goal Distribution

Our method, Skew-Fit, learns a maximum entropy goal distribution q_ϕ^G using samples collected from a goal-conditioned policy. We analyze the algorithm and show that Skew-Fit maximizes the goal distribution entropy, and present a practical instantiation for unsupervised deep RL.

4.3.1 *Skew-Fit Algorithm*

To learn a uniform distribution over *valid* goal states, we present a method that iteratively increases the entropy of a generative model q_ϕ^G. In particular, given a generative model $q_{\phi_t}^G$ at iteration t, we want to train a new generative model, $q_{\phi_{t+1}}^G$ that has higher entropy. While we do not know the set of valid states $\mathcal{S}$, we could sample states $\mathbf{s}_n \overset{\text{iid}}{\sim} p_{\phi_t}^S$ using the goal-conditioned policy, and use the samples to train $q_{\phi_{t+1}}^G$. However, there is no guarantee that this would increase the entropy of $q_{\phi_{t+1}}^G$.

The intuition behind our method is simple: rather than fitting a generative model to these samples $\mathbf{s}_n$, we *skew* the samples so that rarely visited states are given more weight. See Figure 14 for a visualization of this process. How should we skew the samples if we want to maximize the entropy of $q_{\phi_{t+1}}^G$? If we had access to the density of each state, $p_{\phi_t}^S(\mathbf{S})$, then we could simply weight each state by $1/p_{\phi_t}^S(\mathbf{S})$. We could then perform maximum likelihood estimation (MLE) for the uniform distribution by using the follow-

3 Note that this assumption does **not** require that the entropy of p_ϕ^S is strictly larger than the entropy of the goal distribution, q_ϕ^G.

ing importance sampling (IS) loss to train ϕ_{t+1}:

$$\begin{aligned}\mathcal{L}(\phi) &= \mathbb{E}_{\mathbf{S}\sim U_{\mathcal{S}}}\left[\log q^{G}_{\phi}(\mathbf{S})\right] \\ &= \mathbb{E}_{\mathbf{S}\sim p^{S}_{\phi_t}}\left[\frac{U_{\mathcal{S}}(\mathbf{S})}{p^{S}_{\phi_t}(\mathbf{S})}\log q^{G}_{\phi}(\mathbf{S})\right] \\ &\propto \mathbb{E}_{\mathbf{S}\sim p^{S}_{\phi_t}}\left[\frac{1}{p^{S}_{\phi_t}(\mathbf{S})}\log q^{G}_{\phi}(\mathbf{S})\right]\end{aligned}$$

where we use the fact that the uniform distribution $U_{\mathcal{S}}(\mathbf{S})$ has constant density for all states in $\mathcal{S}$. However, computing this density $p^{S}_{\phi_t}(\mathbf{S})$ requires marginalizing out the MDP dynamics, which requires an accurate model of both the dynamics and the goal-conditioned policy.

We avoid needing to model the entire MDP process by approximating $p^{S}_{\phi_t}(\mathbf{S})$ with our previous learned generative model: $p^{S}_{\phi_t}(\mathbf{S}) \approx q^{G}_{\phi_t}(\mathbf{S})$. We therefore weight each state by the following weight function

$$w_{t,\alpha}(\mathbf{S}) \triangleq q^{G}_{\phi_t}(\mathbf{S})^{\alpha}, \quad \alpha < 0. \tag{9}$$

where α is a hyperparameter that controls how heavily we weight each state. If our approximation $q^{G}_{\phi_t}$ is exact, we can choose $\alpha = -1$ and recover the exact IS procedure described above. If $\alpha = 0$, then this skew step has no effect. By choosing intermediate values of α, we trade off the reliability of our estimate $q^{G}_{\phi_t}(\mathbf{S})$ with the speed at which we want to increase the goal distribution entropy.

VARIANCE REDUCTION As described, this procedure relies on IS, which can have high variance, particularly if $q^{G}_{\phi_t}(\mathbf{S}) \approx 0$. We therefore choose a class of generative models where the probabilities are prevented from collapsing to zero, as we will describe in Section 4.4 where we provide generative model details. To further reduce the variance, we train $q^{G}_{\phi_{t+1}}$ with sampling importance resampling (SIR) (Rubin, 1988) rather than IS. Rather than sampling from $p^{S}_{\phi_t}$ and weighting the update from each sample by $w_{t,\alpha}$, SIR explicitly defines a skewed empirical distribution as

$$p_{\text{skewed}_t}(\mathbf{s}) \triangleq \frac{1}{Z_\alpha} w_{t,\alpha}(\mathbf{s}) \delta(\mathbf{s} \in \{\mathbf{s}_n\}_{n=1}^N) \tag{10}$$
$$Z_\alpha = \sum_{n=1}^N w_{t,\alpha}(\mathbf{s}_n), \ \mathbf{s}_n \overset{\text{iid}}{\sim} p^S_{\phi_t},$$

where δ is the indicator function and Z_α is the normalizing coefficient. We note that computing Z_α adds little computational overhead, since all of the weights already need to be computed. We then fit the generative model at the next iteration $q^G_{\phi_{t+1}}$ to p_{skewed_t} using standard MLE. We found that using SIR resulted in significantly lower variance than IS. See Section C.2.2 for this comparision.

GOAL SAMPLING ALTERNATIVE Because $q^G_{\phi_{t+1}} \approx p_{\text{skewed}_t}$, at iteration $t+1$, one can sample goals from either $q^G_{\phi_{t+1}}$ or p_{skewed_t}. Sampling goals from p_{skewed_t} may be preferred if sampling from the learned generative model $q^G_{\phi_{t+1}}$ is computationally or otherwise challenging. In either case, one still needs to train the generative model $q^G_{\phi_t}$ to create p_{skewed_t}. In our experiments, we found that both methods perform well.

SUMMARY Overall, Skew-Fit collects states from the environment and resamples each state in proportion to Equation 9 so that low-density states are resampled more often. Skew-Fit is shown in Figure 14 and summarized in Algorithm 4. We now discuss conditions under which Skew-Fit converges to the uniform distribution.

Algorithm 3 Skew-Fit

```
1: for Iteration $t = 1, 2, ...$ do
2:     Collect $N$ states $\{\mathbf{s}_n\}_{n=1}^N$ by sampling goals from $q^G_{\phi_t}$ (or $p_{\text{skewed}t-1}$) and running goal-conditioned policy.
3:     Construct skewed distribution $p_{\text{skewed}_t}$ (Equation 9 and Equation 10).
4:     Fit $q^G_{\phi_{t+1}}$ to skewed distribution $p_{\text{skewed}_t}$ using MLE.
5: end for=0
```

4.3.2 *Skew-Fit Analysis*

This section provides conditions under which $q^G_{\phi_t}$ converges in the limit to the uniform distribution over the state space $\mathcal{S}$. We consider the case where $N \to \infty$, which allows us to study the limit behavior of the goal distribution p_{skewed_t}. Our most general result is stated as follows:

Lemma 1. *Let $\mathcal{S}$ be a compact set. Define the set of distributions $\mathcal{Q} = \{p : \textit{support}(p) \subseteq \mathcal{S}\}$. Let $\mathcal{F} : \mathcal{Q} \mapsto \mathcal{Q}$ be continuous with respect to the pseudometric $d_{\mathcal{H}}(p, q) \triangleq |\mathcal{H}(p) - \mathcal{H}(q)|$ and $\mathcal{H}(\mathcal{F}(p)) \geqslant \mathcal{H}(p)$ with equality if and only if p is the uniform probability distribution on $\mathcal{S}$, denoted as $U_{\mathcal{S}}$. Define the sequence of distributions $P = (p_1, p_2, \dots)$ by starting with any $p_1 \in \mathcal{Q}$ and recursively defining $p_{t+1} = \mathcal{F}(p_t)$. The sequence P converges to $U_{\mathcal{S}}$ with respect to $d_{\mathcal{H}}$. In other words, $\lim_{t\to 0} |\mathcal{H}(p_t) - \mathcal{H}(U_{\mathcal{S}})| \to 0$.*

Proof. See Appendix Section C.1.1. □

We will apply Lemma 1 to be the map from p_{skewed_t} to $p_{\text{skewed}_{t+1}}$ to show that p_{skewed_t} converges to $U_{\mathcal{S}}$. If we assume that the goal-conditioned policy and generative model learning procedure are well behaved (i.e., the maps from $q^G_{\phi_t}$ to $p^S_{\phi_t}$ and from p_{skewed_t} to $q^G_{\phi_{t+1}}$ are continuous), then to apply Lemma 1, we only need to show that $\mathcal{H}(p_{\text{skewed}_t}) \geqslant \mathcal{H}(p^S_{\phi_t})$ with equality if and only if $p^S_{\phi_t} = U_{\mathcal{S}}$. For the simple case when $q^G_{\phi_t} = p^S_{\phi_t}$ identically at each iteration, we prove the convergence of Skew-Fit true for any value of $\alpha \in [-1, 0)$ in Section C.1.3. However, in practice, $q^G_{\phi_t}$ only approximates $p^S_{\phi_t}$. To address this more realistic situation, we prove the following result:

Lemma 2. *Given two distribution $p^S_{\phi_t}$ and $q^G_{\phi_t}$ where $p^S_{\phi_t} \ll q^G_{\phi_t}$* [a] *and*

$$\mathrm{Cov}_{\mathbf{S}\sim p^S_{\phi_t}}\left[\log p^S_{\phi_t}(\mathbf{S}), \log q^G_{\phi_t}(\mathbf{S})\right] > 0, \tag{11}$$

define the $p_{\textit{skewed}_t}$ as in Equation 10 and take $N \to \infty$. Let $\mathcal{H}_\alpha(\alpha)$ be the entropy of $p_{\textit{skewed}_t}$ for a fixed α. Then there exists a constant $a < 0$ such that for all $\alpha \in [a, 0)$,

$$\mathcal{H}(p_{\textit{skewed}_t}) = \mathcal{H}_\alpha(\alpha) > \mathcal{H}(p^S_{\phi_t}).$$

[a] $p \ll q$ means that p is absolutely continuous with respect to q, i.e. $p(\mathbf{s}) = 0 \implies q(\mathbf{s}) = 0$.

Proof. See Appendix Section C.1.2. □

This lemma tells us that our generative model $q^G_{\phi_t}$ does not need to exactly fit the sampled states. Rather, we merely need the log densities of $q^G_{\phi_t}$ and $p^S_{\phi_t}$ to be correlated, which we expect to happen frequently with an accurate goal-conditioned policy, since

$p^S_{\phi_t}$ is the set of states seen when trying to reach goals from $q^G_{\phi_t}$. In this case, if we choose negative values of α that are small enough, then the entropy of p_{skewed_t} will be higher than that of $p^S_{\phi_t}$. Empirically, we found that α values as low as $\alpha = -1$ performed well.

In summary, p_{skewed_t} converges to $U_{\mathcal{S}}$ under certain assumptions. Since we train each generative model $q^G_{\phi_{t+1}}$ by fitting it to p_{skewed_t} with MLE, $q^G_{\phi_t}$ will also converge to $U_{\mathcal{S}}$.

4.4 TRAINING GOAL-CONDITIONED POLICIES WITH SKEW-FIT

Thus far, we have presented Skew-Fit assuming that we have access to a goal-reaching policy, allowing us to separately analyze how we can maximize $\mathcal{H}(\mathbf{G})$. However, in practice we do not have access to such a policy, and this section discusses how we concurrently train a goal-reaching policy.

Maximizing $I(\mathbf{S};\mathbf{G})$ can be done by simultaneously performing Skew-Fit and training a goal-conditioned policy to minimize $\mathcal{H}(\mathbf{G} \mid \mathbf{S})$, or, equivalently, maximize $-\mathcal{H}(\mathbf{G} \mid \mathbf{S})$. Maximizing $-\mathcal{H}(\mathbf{G} \mid \mathbf{S})$ requires computing the density $\log p(\mathbf{G} \mid \mathbf{S})$, which may be difficult to compute without strong modeling assumptions. However, for any distribution q, the following lower bound on $-\mathcal{H}(\mathbf{G} \mid \mathbf{S})$:

$$\begin{aligned} -\mathcal{H}(\mathbf{G} \mid \mathbf{S}) &= \mathbb{E}_{(\mathbf{G},\mathbf{S})\sim q}\left[\log q(\mathbf{G} \mid \mathbf{S})\right] + pq \\ &\geqslant \mathbb{E}_{(\mathbf{G},\mathbf{S})\sim q}\left[\log q(\mathbf{G} \mid \mathbf{S})\right], \end{aligned}$$

where D_{KL} denotes Kullback–Leibler divergence as discussed by Barber and Agakov (2004). Thus, to minimize $\mathcal{H}(\mathbf{G} \mid \mathbf{S})$, we train a policy to maximize the reward

$$r(\mathbf{S},\mathbf{G}) = \log q(\mathbf{G} \mid \mathbf{S}).$$

The RL algorithm we use is reinforcement learning with imagined goals (RIG) (A. Nair et al., 2018b), though in principle any goal-conditioned method could be used. RIG is an efficient off-policy goal-conditioned method that solves vision-based RL problems in a learned latent space. In particular, RIG fits a β-VAE (Higgins et al., 2017a) and uses it to encode observations and goals into a latent space, which it uses as the state representation. RIG also uses the β-VAE to compute rewards, $\log q(\mathbf{G} \mid \mathbf{S})$. Unlike RIG, we use the goal distribution from Skew-Fit to sample goals for exploration and for relabeling goals during training (Andrychowicz et al., 2017b). Since RIG already trains a generative model over states, we reuse this β-VAE for the generative model q^G_ϕ of Skew-Fit. To make the most use of the data, we train q^G_ϕ on all visited state rather than only the terminal states, which we found to work well in practice. To prevent the estimated state likelihoods from collapsing to zero, we model the posterior of the β-VAE as a multivariate Gaussian distribution with a fixed variance and only learn the mean. We summarize RIG and provide details for how we combine Skew-Fit and RIG in Section C.3.4 and describe how we estimate the likelihoods given the β-VAE in Section C.3.1.

4.5 RELATED WORK

Many prior methods in the goal-conditioned reinforcement learning literature focus on training goal-conditioned policies and assume that a goal distribution is available to sample from during exploration (L P Kaelbling, 1993; Schaul et al., 2015b; Andrychowicz et al., 2017b; V. Pong et al., 2018), or use a heuristic to design a non-parametric (Colas et al., 2018b; Warde-Farley et al., 2018; Florensa et al., 2018a) or parametric (Pr et al., 2018b; A. Nair et al., 2018b) goal distribution based on previously visited states. These methods are largely complementary to our work: rather than proposing a better method for training goal-reaching policies, we propose a principled method for maximizing the entropy of a goal sampling distribution, $\mathcal{H}(\mathbf{G})$, such that these policies cover a wide range of states.

Our method learns without any task rewards, directly acquiring a policy that can be reused to reach user-specified goals. This stands in contrast to exploration methods that modify the reward based on state visitation frequency (M. Bellemare et al., 2016; Ostrovski et al., 2017; Tang et al., 2017; Chentanez et al., 2005; Lopes et al., 2012; Stadie et al., 2016; Pathak et al., 2017; Burda et al., 2018; Burda et al., 2019; Mohamed and Rezende, 2015; Tang et al., 2017; Fu et al., 2017). While these methods can also be used without a task reward, they provide no mechanism for distilling the knowledge gained from visiting diverse states into flexible policies that can be applied to accomplish new goals at test-time: their policies visit novel states, and they quickly forget about them as other states become more novel. Similarly, methods that provably maximize state entropy without using goal-directed exploration (Hazan et al., 2019) or methods that define new rewards to capture measures of intrinsic motivation (Mohamed and Rezende, 2015) and reachability (Savinov et al., 2018) do not produce reusable policies.

Other prior methods extract reusable skills in the form of latent-variable-conditioned policies, where latent variables are interpreted as options (R. S. Sutton et al., 1999) or abstract skills (Hausman et al., 2018; Abhishek Gupta et al., 2018b; Eysenbach et al., 2018; Abhishek Gupta et al., 2018a; Florensa et al., 2017). The resulting skills are diverse, but have no grounded interpretation, while Skew-Fit policies can be used immediately after unsupervised training to reach diverse user-specified goals.

Some prior methods propose to choose goals based on heuristics such as learning progress (Adrien Baranes and Pierre-Yves Oudeyer, 2012a; Veeriah et al., 2018; Colas et al., 2018a), how off-policy the goal is (Nachum et al., 2018), level of difficulty (Held et al., 2018), or likelihood ranking (R. Zhao and Tresp, 2019). In contrast, our approach provides a principled framework for optimizing a concrete and well-motivated exploration objective, can provably maximize this objective under regularity assumptions, and empirically outperforms many of these prior work (see Section 9.5).

4.6 EXPERIMENTS

Our experiments study the following questions: **(1)** Does Skew-Fit empirically result in a goal distribution with increasing entropy? **(2)** Does Skew-Fit improve exploration for goal-conditioned RL? **(3)** How does Skew-Fit compare to prior work on choosing goals for vision-based, goal-conditioned RL? **(4)** Can Skew-Fit be applied to a real-world, vision-based robot task?

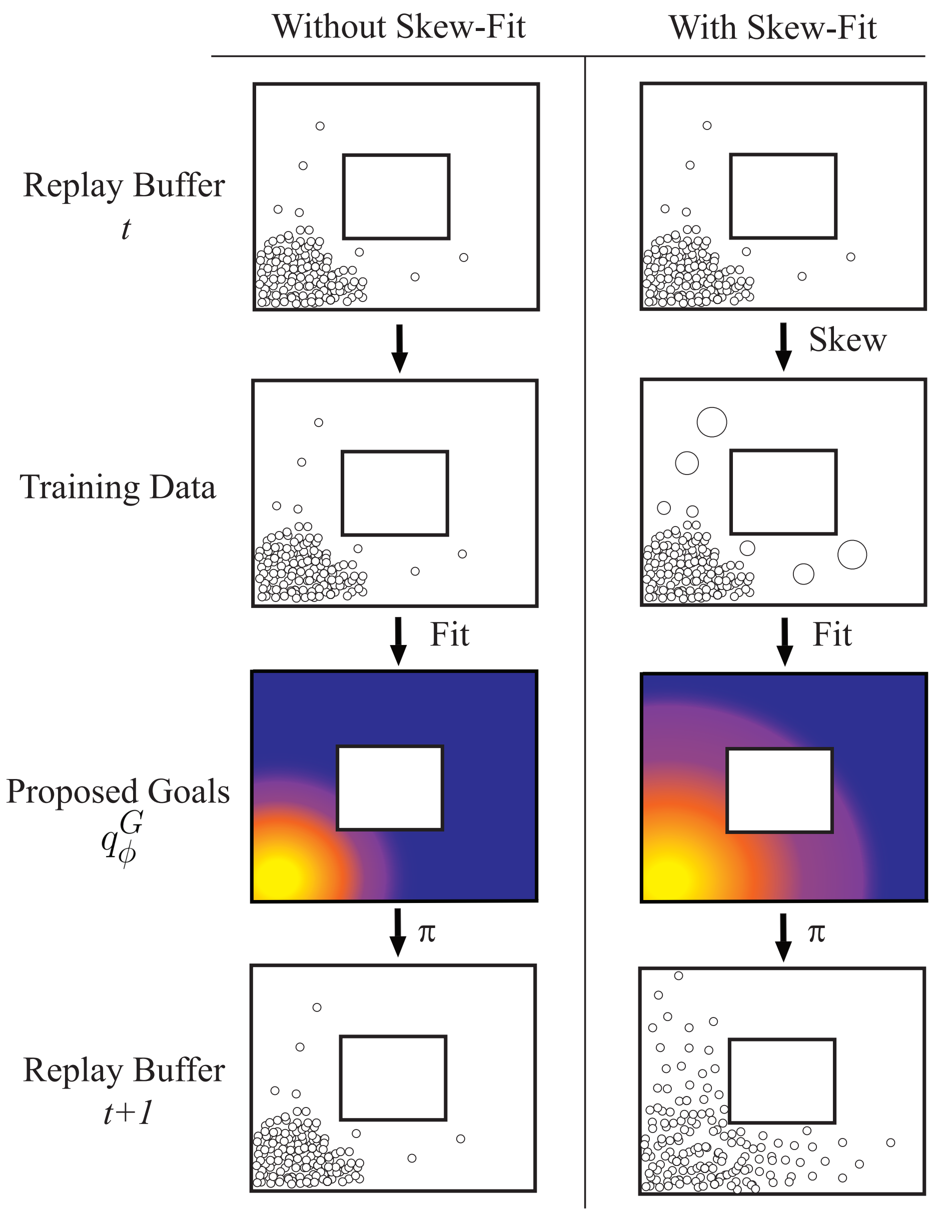


Figure 14: Our method, Skew-Fit, samples goals for goal-conditioned RL. We sample states from our replay buffer, and give more weight to rare states. We then train a generative model $q^G_{\phi\, t+1}$ with the weighted samples. By sampling new states with goals proposed from this new generative model, we obtain a higher entropy state distribution in the next iteration.

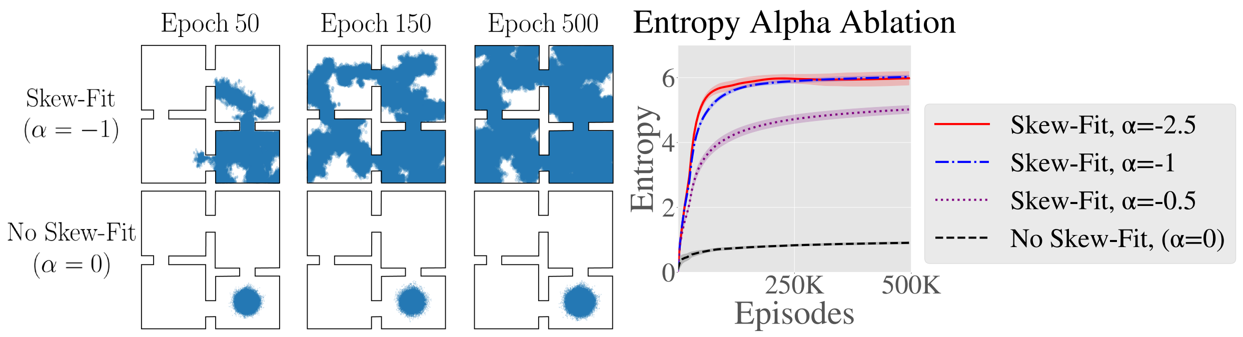


Figure 15: Illustrative example of Skew-Fit on a 2D navigation task. (Left) Visited state plot for Skew-Fit with $\alpha = -1$ and uniform sampling, which corresponds to $\alpha = 0$. (Right) The entropy of the goal distribution per iteration, mean and standard deviation for 9 seeds. Entropy is calculated via discretization onto an 11x11 grid. Skew-Fit steadily increases the state entropy, reaching full coverage over the state space.

DOES SKEW-FIT MAXIMIZE ENTROPY? To see the effects of Skew-Fit on goal distribution entropy in isolation of learning a goal-reaching policy, we study an idealized example where the policy is a near-perfect goal-reaching policy. The environment consists of four rooms (R. S. Sutton et al., 1999). At the beginning of an episode, the agent begins in the bottom-right room and samples a goal from the goal distribution $q^G_{\phi_t}$. To simulate stochasticity of the policy and environment, we add a Gaussian noise with standard deviation of 0.06 units to this goal, where the entire environment is 11×11 units. The policy reaches the state that is closest to this noisy goal and inside the rooms, giving us a state sample $\mathbf{s}_n$ for training $q^G_{\phi_t}$. Due to the relatively small noise, the agent cannot rely on this stochasticity to explore the different rooms and must instead learn to set goals that are progressively farther and farther from the initial state. We compare multiple values of α, where $\alpha = 0$ corresponds to not using Skew-Fit. The β-VAE hyperparameters used to train $q^G_{\phi_t}$ are given in Section C.3.2. As seen in Figure 15, sampling uniformly from previous experience ($\alpha = 0$) to set goals results in a policy that primarily sets goal near the initial state distribution. In contrast, Skew-Fit results in quickly learning a high entropy, near-uniform distribution over the state space.

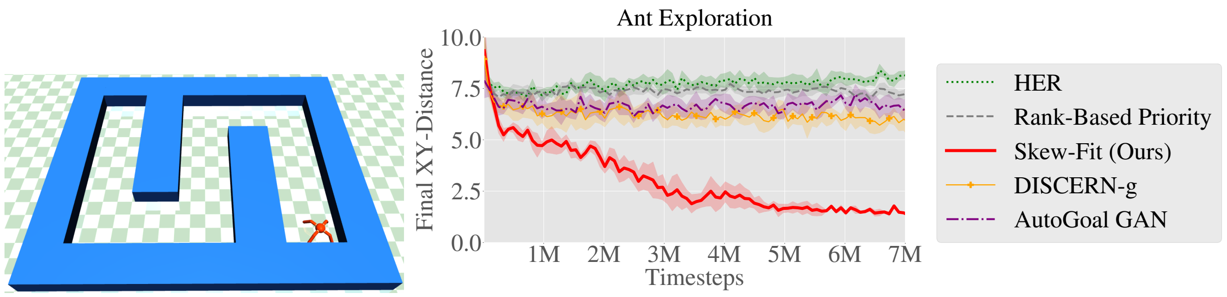


Figure 16: (Left) Ant navigation environment. (Right) Evaluation on reaching target XY position. We show the mean and standard deviation of 6 seeds. Skew-Fit significantly outperforms prior methods on this exploration task.

EXPLORATION WITH SKEW-FIT We next evaluate Skew-Fit while concurrently learning a goal-conditioned policy on a task with state inputs, which enables us study exploration performance independently of the challenges with image observations. We evaluate on a task that requires training a simulated quadruped "ant" robot to navigate to different XY positions in a labyrinth, as shown in Figure 35. The reward is the negative distance to the goal XY-position, and additional environment details are provided in Section C.4. This task presents a challenge for goal-directed exploration: the set of valid goals is unknown due to the walls, and random actions do not result in exploring locations far from the start. Thus, Skew-Fit must set goals that meaningfully explore the space while simultaneously learning to reach those goals.

We use this domain to compare Skew-Fit to a number of existing goal-sampling methods. We compare to the relabeling scheme described in the hindsight experience replay (labeled **HER**). We compare to curiosity-driven prioritization (**Ranked-Based Priority**) (R. Zhao et al., 2019), a variant of HER that samples goals for relabeling based on their ranked likelihoods. Held et al. (2018) samples goals from a GAN based on the difficulty of reaching the goal. We compare against this method by replacing q_ϕ^G with the GAN and label it **AutoGoal GAN**. We also compare to the non-parametric goal proposal mechanism proposed by Warde-Farley et al., 2018, which we label **DISCERN-g**. Lastly, to demonstrate the difficulty of the exploration challenge in these domains, we compare to **#-Exploration** (Tang et al., 2017), an exploration method that assigns bonus rewards based on the novelty of new states. We train the goal-conditioned policy for each method using soft actor critic (SAC) (Haarnoja et al., 2018b). Implementation details of SAC and the prior works are given in Section C.3.3.

We see in Figure 35 that Skew-Fit is the only method that makes significant progress on this challenging labyrinth locomotion task. The prior methods on goal-sampling primarily set goals close to the start location, while the extrinsic exploration reward in #-Exploration dominated the goal-reaching reward. These results demonstrate that Skew-Fit accelerates exploration by setting diverse goals in tasks with unknown goal spaces.

VISION-BASED CONTINUOUS CONTROL TASKS

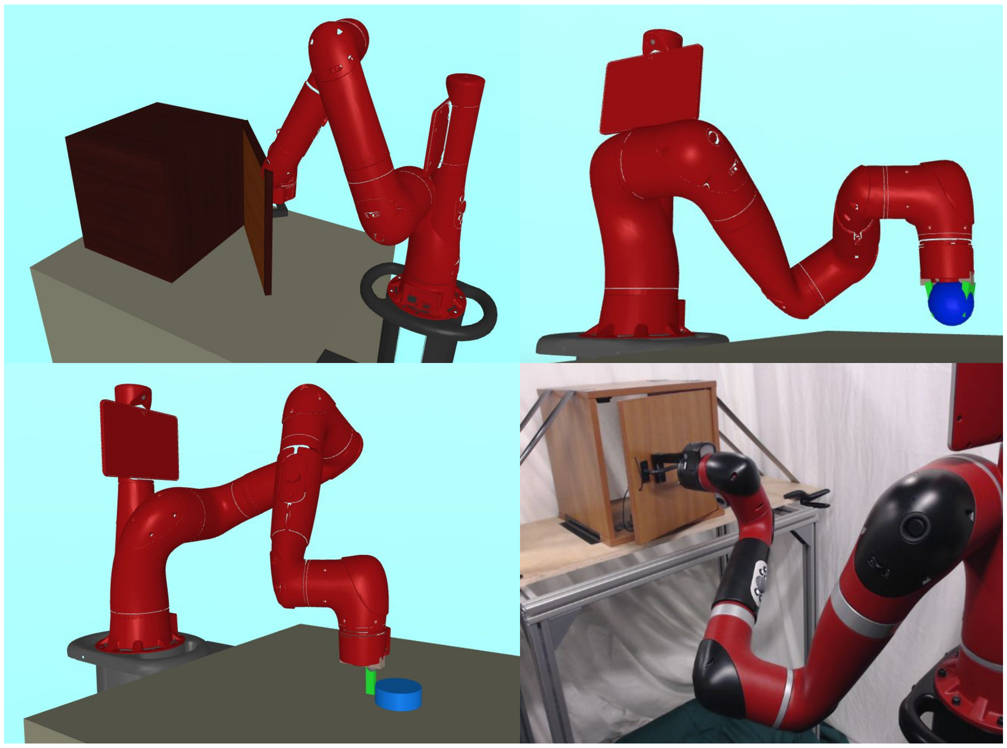


Figure 17: We evaluate on these continuous control tasks, from left to right: *Visual Door*, a door opening task; *Visual Pickup*, a picking task; *Visual Pusher*, a pushing task; and *Real World Visual Door*, a real world door opening task. All tasks are solved from images and without any task-specific reward. See Appendix C.4 for details.

We now evaluate Skew-Fit on a variety of image-based continuous control tasks, where the policy must control a robot arm using only image observations, there is no state-based or task-specific reward, and Skew-Fit must directly set image goals. We test our method on three different image-based simulated continuous control tasks released by the authors of RIG (A. Nair et al., 2018b): *Visual Door*, *Visual Pusher*, and *Visual Pickup*. These environments contain a robot that can open a door, push a puck, and lift up a ball to different configurations, respectively. To our knowledge, these are the only goal-conditioned, vision-based continuous control environments that are publicly available and experimentally evaluated in prior work, making them a good point of comparison. See Figure 17 for visuals and Section C.3 for environment details. The policies are trained in a completely unsupervised manner, without access to any prior information about the image-space or any pre-defined goal-sampling distribution. To evaluate their performance, we sample goal images from a uniform distribution over valid states and report the agent's final distance to the corresponding simulator states (e.g., distance of the object to the target object location), but the agent never has access to this true uniform distribution nor the ground-truth state information during training. While this evaluation method is only practical in simulation, it provides us with a quantitative measure of a policy's ability to reach a broad coverage of goals in a vision-based setting.

We compare Skew-Fit to a number of existing methods on this domain. First, we compare to the methods described in the previous experiment (HER, Rank-Based Priority, #-Exploration, Autogoal GAN, and DISCERN-g). These methods that we compare to were developed in non-vision, state-based environments. To ensure a fair comparison across methods, we combine these prior methods with a policy trained using RIG. We additionally compare to Hazan et al. (2019), an exploration method that assigns bonus rewards based on the likelihood of a state (labeled **Hazan et al.**). Next, we compare to **RIG** without Skew-Fit. Lastly, we compare to **DISCERN** (Warde-Farley et al., 2018), a vision-based method which uses a non-parametric clustering approach to sample goals and an image discriminator to compute rewards.

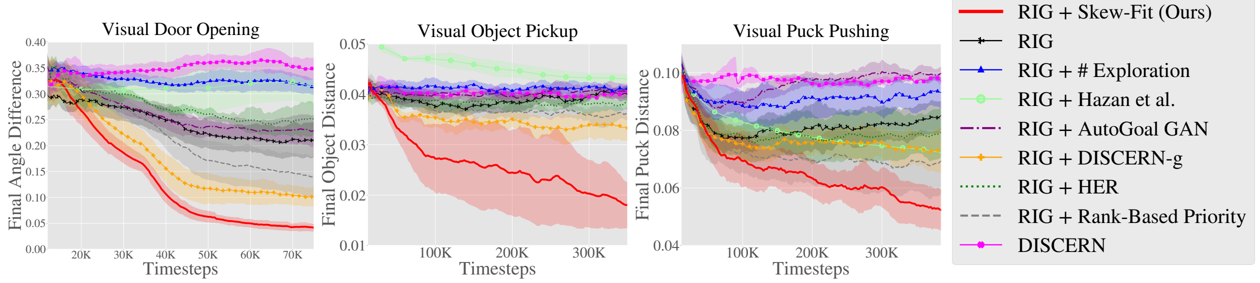


Figure 18: Learning curves for simulated continuous control tasks. Lower is better. We show the mean and standard deviation of 6 seeds and smooth temporally across 50 epochs within each seed. Skew-Fit consistently outperforms RIG and various prior methods. See text for description of each method.

We see in Figure 18 that Skew-Fit significantly outperforms prior methods both in terms of task performance and sample complexity. The most common failure mode for prior methods is that the goal distributions collapse, resulting in the agent learning to reach only a fraction of the state space, as shown in Figure 13. For comparison, additional samples of q_ϕ^G when trained with and without Skew-Fit are shown in Section C.2.3. Those images show that without Skew-Fit, q_ϕ^G produces a small, non-diverse distribution for each environment: the object is in the same place for pickup, the puck is often in the starting position for pushing, and the door is always closed. In contrast, Skew-Fit proposes goals where the object is in the air and on the ground, where the puck positions are varied, and the door angle changes.

We can see the effect of these goal choices by visualizing more example rollouts for RIG and Skew-Fit. These visuals, shown in Figure 72 in Section C.2.3, show that RIG only learns to reach states close to the initial position, while Skew-Fit learns to reach the entire state space. For a quantitative comparison, Figure 19 shows the cumulative total exploration pickups for each method. From the graph, we see that many methods have a near-constant rate of object lifts throughout all of training. Skew-Fit is the only method that significantly increases the rate at which the policy picks up the object during exploration, suggesting that only Skew-Fit sets goals that encourage the policy to interact with the object.

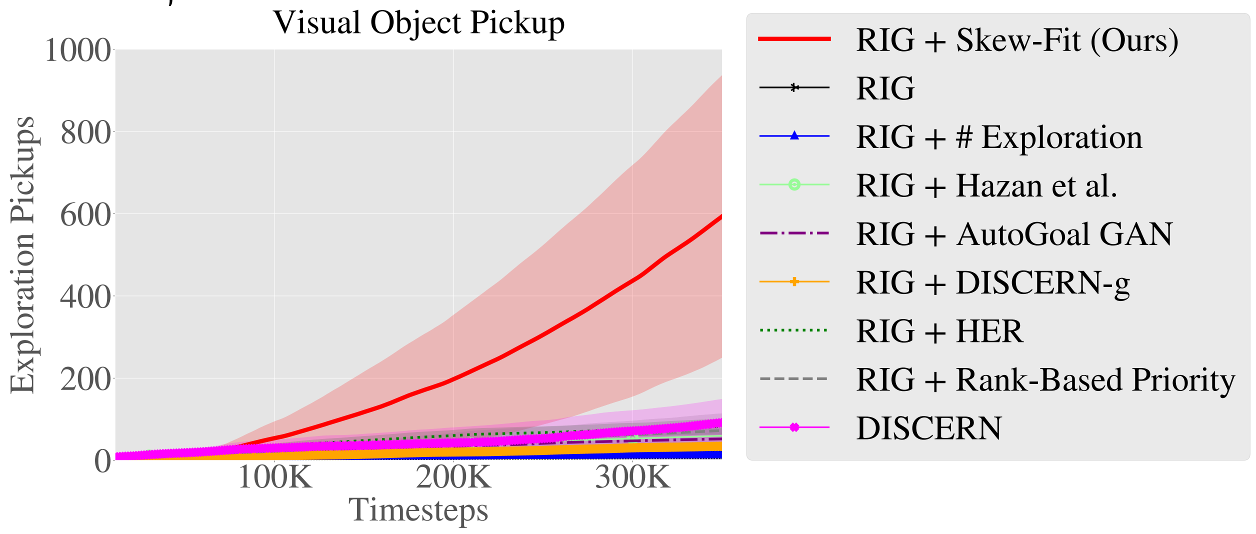
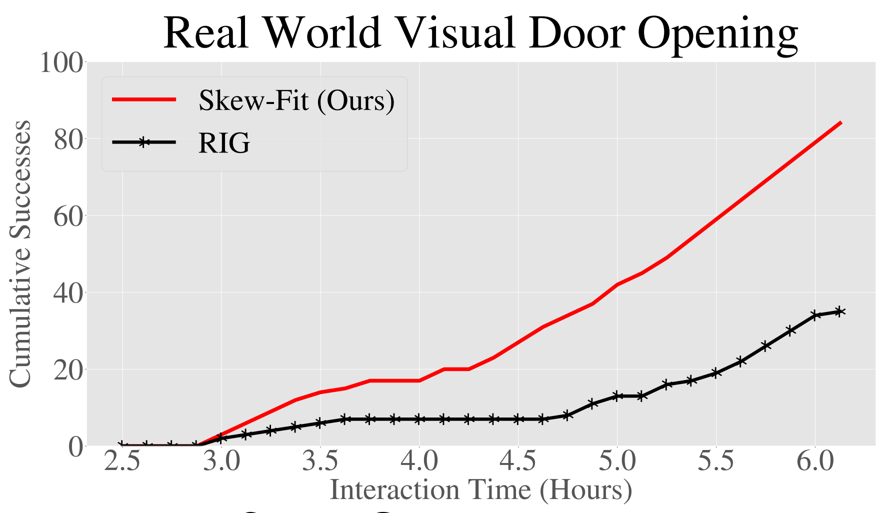


Figure 19: Cumulative total pickups during exploration for each method. Prior methods fail to pay attention to the object: the rate of pickups hardly increases past the first 100 thousand timesteps. In contrast, after seeing the object picked up a few times, Skew-Fit practices picking up the object more often by sampling the appropriate exploration goals.

REAL-WORLD VISION-BASED ROBOTIC MANIPULATION We also demonstrate that Skew-Fit scales well to the real world with a door opening task, *Real World Visual Door*, as shown in Figure 17. While a number of prior works have studied RL-based learning of door opening (Kalakrishnan et al., 2011b; Chebotar et al., 2017b), we demonstrate the first method for autonomous learning of door opening without a user-provided, task-specific reward function. As in simulation, we do not provide any goals to the agent and simply



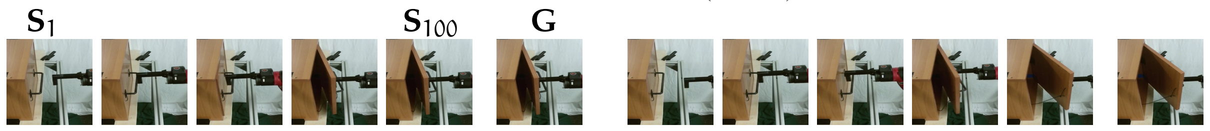


Figure 20: (Top) Learning curve for Real World Visual Door. Skew-Fit results in considerable sample efficiency gains over RIG on this real-world task. (Bottom) Each row shows the Skew-Fit policy starting from state $\mathbf{S}_1$ and reaching state $\mathbf{S}_{100}$ while pursuing goal **G**. Despite being trained from only images without any user-provided goals during training, the Skew-Fit policy achieves the goal image provided at test-time, successfully opening the door.

let it interact with the door, without any human guidance or reward signal. We train two agents using RIG and RIG with Skew-Fit. Every seven and a half minutes of interaction time, we evaluate on 5 goals and plot the cumulative successes for each method. Unlike in simulation, we cannot easily measure the difference between the policy's achieved and desired door angle. Instead, we visually denote a binary success/failure for each goal based on whether the last state in the trajectory achieves the target angle. As Figure 20 shows, standard RIG only starts to open the door after five hours of training. In contrast, Skew-Fit learns to occasionally open the door after three hours of training and achieves a near-perfect success rate after five and a half hours of interaction. Figure 20 also shows examples of successful trajectories from the Skew-Fit policy, where we see that the policy can reach a variety of user-specified goals. These results demonstrate that Skew-Fit is a promising technique for solving real world tasks without any human-provided reward function. Videos of Skew-Fit solving this task and the simulated tasks can be viewed on our website. [4]

4 https://sites.google.com/view/skew-fit

ADDITIONAL EXPERIMENTS To study the sensitivity of Skew-Fit to the hyperparameter α, we sweep α across the values $[-1, -0.75, -0.5, -0.25, 0]$ on the simulated image-based tasks. The results are in Section C.2 and demonstrate that Skew-Fit works across a large range of values for α, and $\alpha = -1$ consistently outperform $\alpha = 0$ (i.e. outperforms no Skew-Fit). Additionally, Section C.3 provides a complete description our method hyperparameters, including network architecture and RL algorithm hyperparameters.

4.7 CONCLUSION

We presented a formal objective for self-supervised goal-directed exploration, allowing researchers to quantify and compare progress when designing algorithms that enable agents to autonomously learn. We also presented Skew-Fit, an algorithm for training a generative model to approximate a uniform distribution over an initially unknown set of valid states, using data obtained via goal-conditioned reinforcement learning, and our theoretical analysis gives conditions under which Skew-Fit converges to the uniform distribution. When such a model is used to choose goals for exploration and to relabeling goals for training, the resulting method results in much better coverage of the state space, enabling our method to explore effectively. Our experiments show that when we concurrently train a goal-reaching policy using self-generated goals, Skew-Fit produces quantifiable improvements on simulated robotic manipulation tasks, and can be used to learn a door opening skill to reach a 95% success rate directly on a real-world robot, without any human-provided reward supervision.

4.8 CONTRIBUTION STATEMENT

The work in this chapter was performed in collaboration with Vitchyr Pong, Murtaza Dalal, Steven Lin, Shikhar Bahl, and Sergey Levine (Vitchyr H. Pong et al., 2019). V.P., M.D., and S.L. were joint co-first authors. The idea of self-supervised goal setting with an expanding goal space by iteratively retraining a generative model was developed jointly by V.P. and A.N. V.P. conducted the theoretical analysis, and managed the project, and led the writing of the paper. The simulation experiments were conducted by V.P., M.D., and S. Lin. The real-world experiments were conducted by V.P., M.D., A.N., and S.B. A.N. assisted with writing and analysis. S. Levine advised on the project, guided the theoretical analysis, and assisted with writing.

Part II

ACCELERATING REINFORCEMENT LEARNING WITH PRIOR KNOWLEDGE

5

OVERCOMING EXPLORATION IN REINFORCEMENT LEARNING WITH DEMONSTRATIONS

5.1 INTRODUCTION

In Part II, we turn to improving the underlying reinforcement learning algorithm itself for practical, real-world robot learning. RL has found significant success in decision making for solving games, so what makes it more challenging to apply in robotics? A key difference is the difficulty of exploration, which comes from the choice of reward function and complicated environment dynamics. In games, the reward function is usually given and can be directly optimized. In robotics, we often desire behavior to achieve some binary objective (e.g., move an object to a desired location or achieve a certain state of the system) which naturally induces a sparse reward. Sparse reward functions are easier to specify and recent work suggests that learning with a sparse reward results in learned policies that perform the desired objective instead of getting stuck in local optima (Andrychowicz et al., 2017b; Veerk et al., 2017). However, exploration in an environment with sparse reward is difficult since with random exploration, the agent rarely sees a reward signal.

The difficulty posed by a sparse reward is exacerbated by the complicated environment dynamics in robotics. For example, system dynamics around contacts are difficult to model and induce a sensitivity in the system to small errors. Many robotics tasks also require executing multiple steps successfully over a long horizon, involve high dimensional control, and require generalization to varying task instances. These conditions further result in a situation where the agent so rarely sees a reward initially that it is not able to learn at all.

All of the above means that random exploration is not a tenable solution. Instead, in this work we show that we can use demonstrations as a guide for our exploration. To test our method, we solve the problem of stacking several blocks at a given location from a

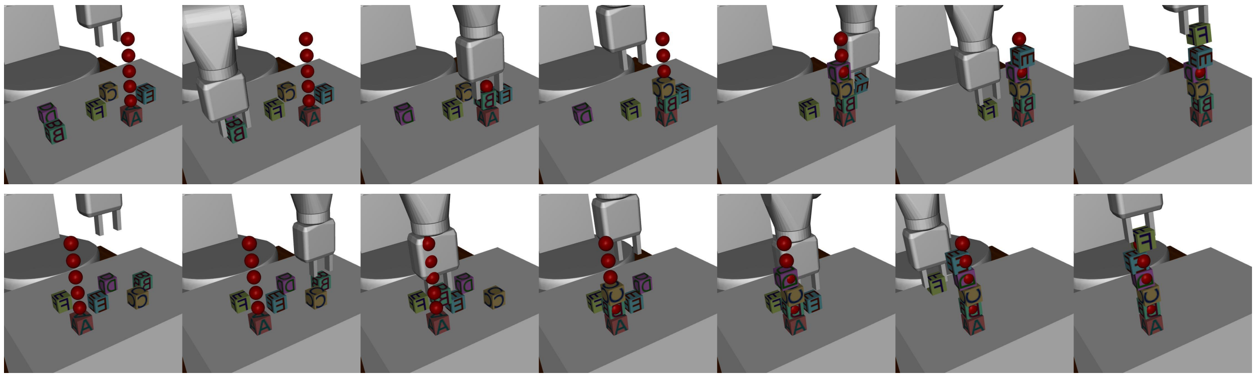


Figure 21: We present a method using reinforcement learning to solve the task of block stacking shown above. The robot starts with 6 blocks labelled A through F on a table in random positions and a target position for each block. The task is to move each block to its target position. The targets are marked in the above visualization with red spheres which do not interact with the environment. These targets are placed in order on top of block A so that the robot forms a tower of blocks. This is a complex, multi-step task where the agent needs to learn to successfully manage multiple contacts to succeed. Frames from rollouts of the learned policy are shown. A video of our experiments can be found at: `http://ashvin.me/demoddpg-website`

random initial state. Stacking blocks has been studied before in the literature (Marc Peter Deisenroth et al., 2011a; Duan et al., 2017) and exhibits many of the difficulties mentioned: long horizons, contacts, and requires generalizing to each instance of the task. We limit ourselves to 100 human demonstrations collected via teleoperation in virtual reality. Using these demonstrations, we are able to solve a complex robotics task in simulation that is beyond the capability of both reinforcement learning and imitation learning.

In this chapter, we show how demonstrations can be used within a reinforcement learning algorithm to solve complex tasks where exploration is difficult. We introduce a simple auxiliary objective on demonstrations, a method of annealing away the effect of the demonstrations when the learned policy is better than the demonstrations, and a method of resetting from demonstration states that significantly improves and speeds up training policies. By effectively incorporating demonstrations into RL, we short-circuit the random exploration phase of RL and reach nonzero rewards and a reasonable policy early on in training. Finally, we extensively evaluate our method against other commonly used methods, such as initialization with learning from demonstrations and fine-tuning with RL, and show that our method significantly outperforms them.

5.2 RELATED WORK

Learning methods for decision making problems such as robotics largely divide into two classes: imitation learning and reinforcement learning (RL). In imitation learning (also called learning from demonstrations) the agent receives behavior examples from an expert and attempts to solve a task by copying the expert's behavior. In RL, an agent attempts to maximize expected reward through interaction with the environment. Our work combines aspects of both to solve complex tasks.

Imitation Learning: Perhaps the most common form of imitation learning is behavior cloning (BC), which learns a policy through supervised learning on demonstration state-action pairs. BC has seen success in autonomous driving (Pomerleau, 1989; Bojarski et al., 2016), quadcopter navigation (Giusti et al., 2015), locomotion (Nakanishi et al., 2004; Kalakrishnan et al., 2009). BC struggles outside the manifold of demonstration data. Dataset Aggregation (DAgger) augments the dataset by interleaving the learned and expert policy to address this problem of accumulating errors (Ross et al., 2011). However, DAgger is difficult to use in practice as it requires access to an expert during all of training, instead of just a set of demonstrations.

Fundamentally, BC approaches are limited because they do not take into account the task or environment. Inverse reinforcement learning (IRL) (A. Ng and Russell, 2000) is another form of imitation learning where a reward function is inferred from the demonstrations. Among other tasks, IRL has been applied to navigation (Ziebart et al., 2008), autonomous helicopter flight (Abbeel and Andrew Y Ng, 2004), and manipulation (Finn et al., 2016a). Since our work assumes knowledge of a reward function, we omit comparisons to IRL approaches.

Reinforcement Learning: Reinforcement learning methods have been harder to apply in robotics, but are heavily investigated because of the autonomy they could enable. Through RL, robots have learned to play table tennis (Peters et al., 2010), swing up a cartpole, and balance a unicycle (Marc Peter Deisenroth and Rasmussen, 2011). A renewal of interest in RL cascaded from success in games (Mnih et al., 2015; D. Silver et al., 2016b), especially because of the ability of RL with large function approximators (ie. deep RL) to learn control from raw pixels. Robotics has been more challenging in general but there has been significant progress. Deep RL has been applied to manipulation tasks (**LevineFDA15**), grasping (Pinto and Abhinav Gupta, 2016; Levine et al., 2016b), opening a door (S. Gu et al., 2017), and locomotion (Lillicrap et al., 2016; Mnih et al., 2016; Schulman et al., 2015). However, results have been attained predominantly in simulation per high sample complexity, typically caused by exploration challenges.

Robotic Block Stacking: Block stacking has been studied from the early days of AI and robotics as a task that encapsulates many difficulties of more complicated tasks we want to solve, including multi-step planning and complex contacts. SHRDLU (Winograd, 1972) was one of the pioneering works, but studied block arrangements only in terms of logic and natural language understanding. More recent work on task and motion planning considers both logical and physical aspects of the task (Leslie Pack Kaelbling and Lozano-Perez, 2011; Kavraki et al., 1996; S. Srivastava et al., 2014), but requires domain-specific engineering. In this work we study how an agent can learn this task without the need of domain-specific engineering.

One RL method, PILCO (Marc Peter Deisenroth and Rasmussen, 2011) has been applied to a simple version of stacking blocks where the task is to place a block on a tower (Marc Peter Deisenroth et al., 2011a). Methods such as PILCO based on learning forward models naturally have trouble modelling the sharply discontinuous dynamics of contacts; although they can learn to place a block, it is a much harder problem to grasp the block in the first place. One-shot Imitation (Duan et al., 2017) learns to stack blocks in a way that generalizes to new target configurations, but uses more than 100,000 demonstrations to train the system. A heavily shaped reward can be used to learn to stack a Lego block on another with RL (Popov et al., 2017). In contrast, our method can succeed from fully sparse rewards and handle stacking several blocks.

Combining RL and Imitation Learning: Previous work has combined reinforcement learning with demonstrations. Demonstrations have been used to accelerate learning on classical tasks such as cart-pole swing-up and balance (Schaal, 1997). This work initialized policies and (in model-based methods) initialized forward models with demonstrations. Initializing policies from demonstrations for RL has been used for learning to hit a baseball (Peters and Schaal, 2008c) and for underactuated swing-up (Kober and Peter, 2008). Beyond initialization, we show how to extract more knowledge from demonstrations by using them effectively throughout the entire training process.

Our method is closest to two recent approaches — Deep Q-Learning From Demonstrations (DQfD) (Hester et al., 2018) and DDPG From Demonstrations (DDPGfD) (Veerk et al., 2017) which combine demonstrations with reinforcement learning. DQfD improves learning speed on Atari, including a margin loss which encourages the expert actions to have higher Q-values than all other actions. This loss can make improving upon the demonstrator policy impossible which is not the case for our method. Prior work has previously explored improving beyond the demonstrator policy in simple environments by introducing slack variables (Kim et al., 2013), but our method uses a learned value to actively inform the improvement. DDPGfD solves simple robotics tasks akin to peg insertion using DDPG with demonstrations in the replay buffer. In contrast to this prior work, the tasks we consider exhibit additional difficulties that are of key interest in robotics: multi-step behaviours, and generalization to varying goal states. While previous work focuses on speeding up already solvable tasks, we show that we can extend the state of the art in RL with demonstrations by introducing new methods to incorporate demonstrations.

5.3 BACKGROUND

5.3.1 *DDPG*

Recall that many RL algorithms involve estimating the expected return from a given state after taking an action:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{r_i, s_i \sim E, a_i \sim \pi}[R_t | s_t, a_t] \tag{12}$$

$$= \mathbb{E}_{r_t, s_{t+1} \sim E}[r_t + \gamma \mathbb{E}_{a_{t+1} \sim \pi}[Q^{\pi}(s_{t+1}, a_{t+1})]] \tag{13}$$

We call Q^{π} the action-value function. Equation 13 is a recursive version of equation 12, and is known as the Bellman equation. The Bellman equation allows for methods to estimate Q that resemble dynamic programming. The Bellman equation allows for methods to estimate Q that resemble dynamic programming, enabling data re-use and sample efficiency. Because data re-use and sample efficiency is extremely important for real-world robot learning, methods that use Bellman bootstrapping will be used throughout this thesis.

Our method combines demonstrations with one such method: Deep Deterministic Policy Gradients (DDPG) (Lillicrap et al., 2016). DDPG is an off-policy model-free reinforcement learning algorithm for continuous control which can utilize large function approximators such as neural networks. DDPG is an actor-critic method, which bridges the gap between policy gradient methods and value approximation methods for RL. At a high level, DDPG learns an action-value function (critic) by minimizing the Bellman error, while simultaneously learning a policy (actor) by directly maximizing the estimated action-value function with respect to the parameters of the policy.

Concretely, DDPG maintains an actor function $\pi(s)$ with parameters θ_{π}, a critic function $Q(s, a)$ with parameters θ_Q, and a replay buffer R as a set of tuples (s_t, a_t, r_t, s_{t+1}) for each transition experienced. DDPG alternates between running the policy to collect experience and updating the parameters. Training rollouts are collected with extra noise for exploration: $a_t = \pi(s) + \mathcal{N}$, where $\mathcal{N}$ is a noise process.

During each training step, DDPG samples a minibatch consisting of N tuples from R to update the actor and critic networks. DDPG minimizes the following loss L w.r.t. θ_Q to update the critic:

$$y_i = r_i + \gamma Q(s_{i+1}, \pi(s_{i+1})) \tag{14}$$

$$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta_Q))^2 \tag{15}$$

The actor parameters θ_π are updated using the policy gradient:

$$\nabla_{\theta_\pi} J = \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta_Q)|_{s=s_i, a=\pi(s)} \nabla_{\theta_\pi} \pi(s|\theta_\pi)|_{s_i} \tag{16}$$

To stabilize learning, the Q value in equation 14 is usually computed using a separate network (called the *target* network) whose weights are an exponential average over time of the critic network. This results in smoother target values.

Note that DDPG is a natural fit for using demonstrations. Since DDPG can be trained off-policy, we can use demonstration data as off-policy training data. We also take advantage of the action-value function $Q(s, a)$ learned by DDPG to better use demonstrations.

5.3.2 *Multi-Goal RL*

Instead of the standard RL setting, we train agents with parametrized goals, which lead to more general policies (Schaul et al., 2015b) and have recently been shown to make learning with sparse rewards easier (Andrychowicz et al., 2017b). Goals describe the task we expect the agent to perform in the given episode, in our case they specify the desired positions of all objects. We sample the goal g at he beginning of every episode. The function approximators, here π and Q, take the current goal as an additional input.

5.3.3 *Hindsight Experience Replay (HER)*

To handle varying task instances and parametrized goals, we use Hindsight Experience Replay (HER) (Andrychowicz et al., 2017b). The key insight of HER is that even in failed rollouts where no reward was obtained, the agent can transform them into successful ones by assuming that a state it saw in the rollout was the actual goal. HER can be used with any off-policy RL algorithm assuming that for every state we can find a goal corresponding to this state (i.e. a goal which leads to a positive reward in this state).

For every episode the agent experiences, we store it in the replay buffer twice: once with the original goal pursued in the episode and once with the goal corresponding to the final state achieved in the episode, as if the agent intended on reaching this state from the very beginning.

5.4 METHOD

Our method combines DDPG and demonstrations in several ways to maximally use demonstrations to improve learning. We describe our method below and evaluate these ideas in our experiments.

5.4.1 *Demonstration Buffer*

First, we maintain a second replay buffer R_D where we store our demonstration data in the same format as R. In each minibatch, we draw an extra N_D examples from R_D to use as off-policy replay data for the update step. These examples are included in both the actor and critic update. This idea has been introduced in (Veerk et al., 2017).

5.4.2 *Behavior Cloning Loss*

Second, we introduce a new loss computed only on the demonstration examples for training the actor.

$$L_{BC} = \sum_{i=1}^{N_D} \pi(s_i|\theta_\pi) - a_i{}^2 \tag{17}$$

This loss is a standard loss in imitation learning, but we show that using it as an auxiliary loss for RL improves learning significantly. The gradient applied to the actor parameters θ_π is:

$$\lambda_1 \nabla_{\theta_\pi} J - \lambda_2 \nabla_{\theta_\pi} L_{BC} \tag{18}$$

(Note that we maximize J and minimize L_{BC}.) Using this loss directly prevents the learned policy from improving significantly beyond the demonstration policy, as the actor is always tied back to the demonstrations. Next, we show how to account for sub-optimal demonstrations using the learned action-value function.

5.4.3 *Q-Filter*

We account for the possibility that demonstrations can be suboptimal by applying the behavior cloning loss only to states where the critic $Q(s, a)$ determines that the demonstrator action is better than the actor action:

$$L_{BC} = \sum_{i=1}^{N_D} \pi(s_i|\theta_\pi) - a_i{}^2 \, \mathbb{1}_{Q(s_i,a_i)>Q(s_i,\pi(s_i))} \tag{19}$$

The gradient applied to the actor parameters is as in equation 18. We label this method using the behavior cloning loss and Q-filter "Ours" in the following experiments.

5.4.4 *Resets to demonstration states*

To overcome the problem of sparse rewards in very long horizon tasks, we reset some training episodes using states and goals from demonstration episodes. Restarts from within demonstrations expose the agent to higher reward states during training. This method makes the additional assumption that we can restart episodes from a given state, as is true in simulation.

To reset to a demonstration state, we first sample a demonstration $D = (x_0, u_0, x_1, u_1, ...x_N, u_N)$ from the set of demonstrations. We then uniformly sample a state x_i from D. As in HER, we use the final state achieved in the demonstration as the goal. We roll out the trajectory with the given initial state and goal for the usual number of timesteps. At evaluation time, we do not use this procedure.

We label our method with the behavior cloning loss, Q-filter, and resets from demonstration states as "Ours, Resets" in the following experiments.

5.5 EXPERIMENTAL SETUP

5.5.1 *Environments*

We evaluate our method on several simulated MuJoCo (Todorov et al., 2012) environments. In all experiments, we use a simulated 7-DOF Fetch Robotics arm with parallel grippers to manipulate one or more objects placed on a table in front of the robot.

The agent receives the positions of the relevant objects on the table as its observations. The control for the agent is continuous and 4-dimensional: 3 dimensions that specify

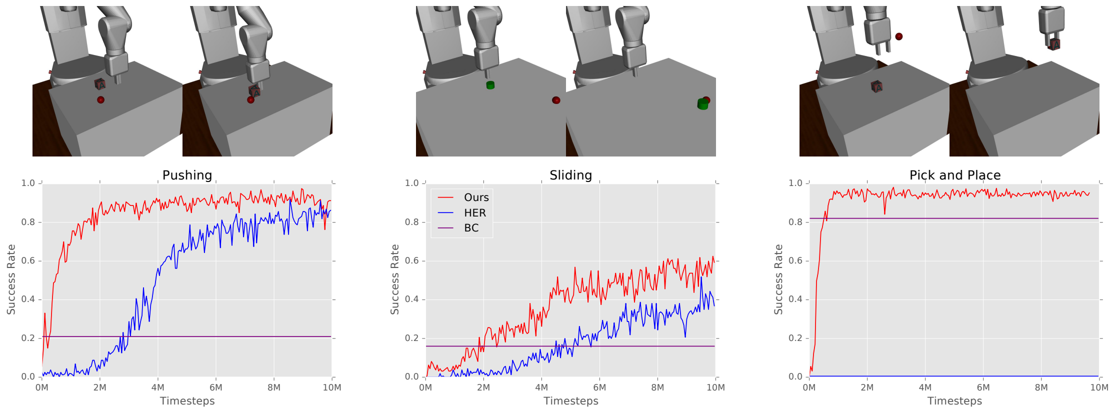


Figure 22: Baseline comparisons on tasks from (Andrychowicz et al., 2017b). Frames from the learned policy are shown above each task. Our method significantly outperforms the baselines. On the right plot, the HER baseline always fails.

the desired end-effector position[1] and 1 dimension that specifies the desired distance between the robot fingers. The agent is controlled at 50Hz frequency.

We collect demonstrations in a virtual reality environment. The demonstrator sees a rendering of the same observations as the agent, and records actions through a HTC Vive interface at the same frequency as the agent. We have the option to accept or reject a demonstration; we only accept demonstrations we judge to be mostly correct. The demonstrations are not optimal. The most extreme example is the "sliding" task, where only 7 of the 100 demonstrations are successful, but the agent still sees rewards for these demonstrations with HER.

5.5.2 *Training Details*

To train our models, we use Adam (D. Kingma and Ba, 2015) as the optimizer with learning rate 10^{-3}. We use $N = 1024, N_D = 128, \lambda_1 = 10^{-3}, \lambda_2 = 1.0/N_D$. The discount factor γ is 0.98. We use 100 demonstrations to initialize R_D. The function approximators π and Q are deep neural networks with ReLU activations and L2 regularization with the coefficient 5×10^{-3}. The final activation function for π is tanh, and the output value is scaled to the range of each action dimension. To explore during training, we sample random actions uniformly within the action space with probability 0.1 at every step, and the noise process $\mathcal{N}$ is uniform over $\pm 10\%$ of the maximum value of each action dimension. Task-specific information, including network architectures, are provided in

1 In the 10cm x 10cm x 10cm cube around the current gripper position

the next section.

5.5.3 *Overview of Experiments*

We perform three sets of experiments. In Sec. 5.6, we provide a comparison to previous work. In Sec. 5.7 we solve block stacking, a difficult multi-step task with complex contacts that the baselines struggle to solve. In Sec. 5.8 we do ablations of our own method to show the effect of individual components.

5.6 COMPARISON WITH PRIOR WORK

5.6.1 *Tasks*

We first show the results of our method on the simulated tasks presented in the Hindsight Experience Replay paper (Andrychowicz et al., 2017b). We apply our method to three tasks:

1. *Pushing*. A block placed randomly on the table must be moved to a target location on the table by the robot (fingers are blocked to avoid grasping).
2. *Sliding*. A puck placed randomly on the table must be moved to a given target location. The target is outside the robot's reach so it must apply enough force that the puck reaches the target and stops due to friction.
3. *Pick-and-place*. A block placed randomly on the table must be moved to a target location in the air. Note that the original paper used a form of initializing from favorable states to solve this task. We omit this for our experiment but discuss and evaluate the initialization idea in an ablation.

As in the prior work, we use a fully sparse reward for this task. The agent is penalized if the object is not at its goal position:

$$r_t = \begin{cases} 0, & \text{if } ||x_i - g_i|| < \delta \\ -1, & \text{otherwise} \end{cases} \tag{20}$$

where the threshold δ is 5cm.

5.6.2 *Results*

Fig. 22 compares our method to HER without demonstrations and behavior cloning. Our method is significantly faster at learning these tasks than HER, and achieves significantly better policies than behavior cloning does. Measuring the number of timesteps to get to convergence, we exhibit a 4x speedup over HER in pushing, a 2x speedup over HER in sliding, and our method solves the pick-and-place task while HER baseline cannot solve it at all.

The pick-and-place task showcases the shortcoming of RL in sparse reward settings, even with HER. In pick-and-place, the key action is to grasp the block. If the robot could manage to grasp it a small fraction of the time, HER discovers how to achieve goals in the air and reinforces the grasping behavior. However, grasping the block with random actions is extremely unlikely. Our method pushes the policy towards demonstration actions, which are more likely to succeed.

In the HER paper, HER solves the pick-and-place task by initializing half of the roll-outs with the gripper grasping the block. With this addition, pick-and-place becomes the easiest of the three tasks tested. This initialization is similar in spirit to our initialization idea, but takes advantage of the fact that pick-and-place with any goal can be solved starting from a block grasped at a certain location. This is not always true (for example, if there are multiple objects to be moved) and finding such a keyframe for other tasks would be difficult, requiring some engineering and sacrificing autonomy. Instead, our method guides the exploration towards grasping the block through demonstrations. Providing demonstrations does not require expert knowledge of the learning system, which makes it a more compelling way to provide prior information.

5.7 MULTI-STEP EXPERIMENTS

5.7.1 *Block Stacking Task*

To show that our method can solve more complex tasks with longer horizon and sparser reward, we study the task of block stacking in a simulated environment as shown in Fig. 47 with the same physical properties as the previous experiments. Our experiments show that our approach can solve the task in full and learn a policy to stack 6 blocks with demonstrations and RL. To measure and communicate various properties of our method, we also show experiments on stacking fewer blocks, a subset of the full task.

We initialize the task with blocks at 6 random locations $x_1 ... x_6$. We also provide 6 goal locations $g_1 ... g_6$. To form a tower of blocks, we let $g_1 = x_1$ and $g_i = g_{i-1} + (0, 0, 5cm)$ for $i \in 2, 3, 4, 5$.

By stacking N blocks, we mean N blocks reach their target locations. Since the target locations are always on top of x_1, we start with the first block already in position. So stacking N blocks involves $N - 1$ pick-and-place actions. To solve stacking N, we allow the agent $50 * (N - 1)$ timesteps. This means that to stack 6 blocks, the robot executes 250 actions or 5 seconds.

We recorded 100 demonstrations to stack 6 blocks, and use subsets of these demonstrations as demonstrations for stacking fewer blocks. The demonstrations are not perfect; they include occasionally dropping blocks, but our method can handle suboptimal demonstrations. We still rejected more than half the demonstrations and excluded them from the demonstration data because we knocked down the tower of blocks when releasing a block.

5.7.2 *Rewards*

Two different reward functions are used. To test the performance of our method under fully sparse reward, we reward the agent only if all blocks are at their goal positions:

$$r_t = \min_i \mathbb{1}_{||x_i - g_i|| < \delta} \tag{21}$$

The threshold δ is the size of a block, 5cm. We call this the "sparse" reward.

To enable solving the longer horizon tasks of stacking 4 or more blocks, we use the "step" reward :

$$r_t = -1 + \sum_i \mathbb{1}_{||x_i - g_i|| < \delta} \tag{22}$$

Note the step reward is still very sparse; the robot only sees the reward change when it moves a block into its target location. We subtract 1 only to make the reward more interpretable, as in the initial state the first block is already at its target.

Regardless of the reward type, an episode is considered successful for computing success rate if all blocks are at their goal position in their final state.

5.7.3 *Network architectures*

We use a 4 layer networks with 256 hidden units per layer for π and Q for the HER tasks and stacking 3 or fewer blocks. For stacking 4 blocks or more, we use an attention

Task	Ours	Ours, Resets	BC	HER	BC+ HER
Stack 2, Sparse	99%	97%	65%	0%	65%
Stack 3, Sparse	99%	89%	1%	0%	1%
Stack 4, Sparse	1%	54%	-	-	-
Stack 4, Step	91%	73%	0%	0%	0%
Stack 5, Step	49%	50%	-	-	-
Stack 6, Step	4%	32%	-	-	-

Figure 23: Comparison of our method against baselines. The value reported is the median of the best performance (success rate) of all randomly seeded runs of each method.

mechanism (Bahdanau et al., 2015) for the actor and a larger network. The attention mechanism uses a 3 layer network with 128 hidden units per layer to query the states and goals with one shared head. Once a state and goal is extracted, we use a 5 layer network with 256 hidden units per layer after the attention mechanism. Attention speeds up training slightly but does not change training outcomes.

5.7.4 *Baselines*

We include the following methods to compare our method to baselines on stacking 2 to 6 blocks. [2]
Ours: Refers to our method as described in section 5.4.3.
Ours, Resets: Refers to our method as described in section 5.4.3 with resets from demonstration states (Sec. 5.4.4).
BC: This method uses behavior cloning to learn a policy. Given the set of demonstration transitions R_D, we train the policy π by supervised learning. Behavior cloning requires much less computation than RL. For a fairer comparison, we performed a large hyperparameter sweep over various networks sizes, attention hyperparameters, and learning rates and report the success rate achieved by the best policy found.
HER: This method is exactly the one described in Hindsight Experience Replay

2 Because of computational constraints, we were limited to 5 random seeds per method for stacking 3 blocks, 2 random seeds per method for stacking 4 and 5 blocks, and 1 random seed per method for stacking 6 blocks. Although we are careful to draw conclusions from few random seeds, the results are consistent with our collective experience training these models. We report the median of the random seeds everywhere applicable.

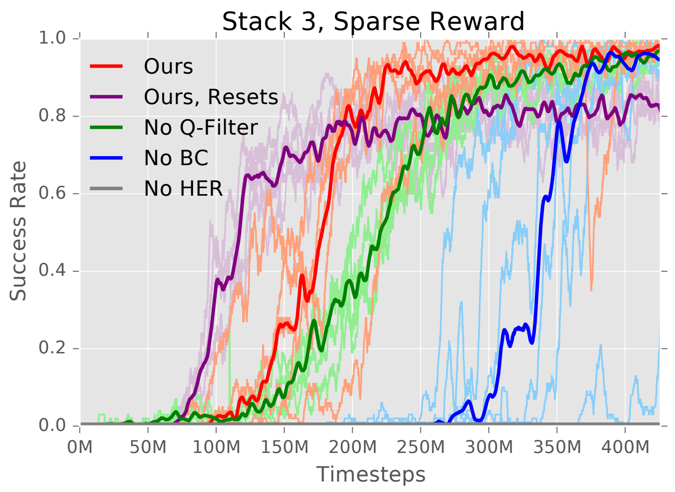


Figure 24: Ablation results on stacking 3 blocks with a fully sparse reward. We run each method 5 times with random seeds. The bold line shows the median of the 5 runs while each training run is plotted in a lighter color. Note "No HER" is always at 0% success rate. Our method without resets learns faster than the ablations. Our method with resets initially learns faster but converges to a worse success rate.

(Andrychowicz et al., 2017b), using HER and DDPG.
BC+HER: This method first initializes a policy (actor) with BC, then finetunes the policy with RL as described above.

5.7.5 *Results*

We are able to learn much longer horizon tasks than the other methods, as shown in Fig. 23. The stacking task is extremely difficult using HER without demonstrations because the chance of grasping an object using random actions is close to 0. Initializing a policy with demonstrations and then running RL also fails since the actor updates depend on a reasonable critic and although the actor is pretrained, the critic is not. The pretrained actor weights are therefore destroyed in the very first epoch, and the result is no better than BC alone. We attempted variants of this method where initially the critic was trained from replay data. However, this also fails without seeing on-policy data.

The results with sparse rewards are very encouraging. We are able to stack 3 blocks with a fully sparse reward without resetting to the states from demonstrations, and 4 blocks with a fully sparse reward if we use resetting. With resets from demonstration states and the step reward, we are able to learn a policy to stack 6 blocks.

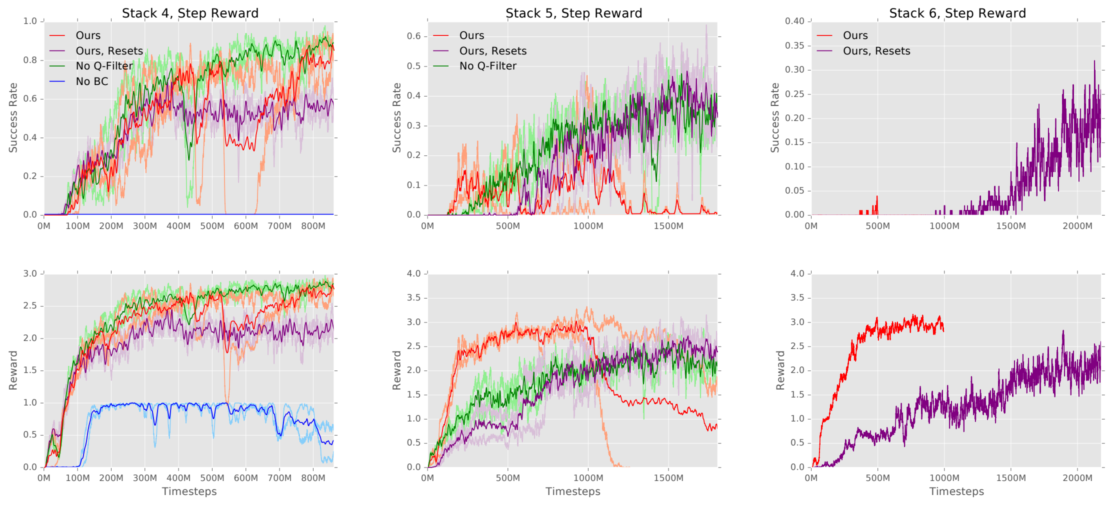


Figure 25: Ablation results on longer horizon tasks with a step reward. The upper row shows the success rate while the lower row shows the average reward at the final step of each episode obtained by different algorithms. For stacking 4 and 5 blocks, we use 2 random seeds per method. The median of the runs is shown in bold and each training run is plotted in a lighter color. Note that for stacking 4 blocks, the "No BC" method is always at 0% success rate. As the number of blocks increases, resets from demonstrations becomes more important to learn the task.

5.8 ABLATION EXPERIMENTS

In this section we perform a series of ablation experiments to measure the importance of various components of our method. We evaluate our method on stacking 3 to 6 blocks.

We perform the following ablations on the best performing of our models on each task:

No BC Loss: This method does not apply the behavior cloning gradient during training. It still has access to demonstrations through the demonstration replay buffer.

No Q-Filter: This method uses standard behavioral cloning loss instead of the loss from equation Eq. 19, which means that the actor tries to mimic the demonstrator's behaviour regardless of the critic.

No HER: Hindsight Experience Replay is not used.

5.8.1 *Behavior Cloning Loss*

Without the behavior cloning loss, the method is significantly worse in every task we try. Fig. 24 shows the training curve for learning to stack 3 blocks with a fully sparse reward. Without the behavior cloning loss, the system is about 2x slower to learn. On longer horizon tasks, we do not achieve any success without this loss.

To see why, consider the training curves for stacking 4 blocks shown in Fig. 25. The "No BC" policy learns to stack only one additional block. Without the behavior cloning loss, the agent only has access to the demonstrations through the demonstration replay buffer. This allows it to view high-reward states and incentivizes the agent to stack more blocks, but there is a stronger disincentive: stacking the tower higher is risky and could result in lower reward if the agent knocks over a block that is already correctly placed. Because of this risk, which is fundamentally just another instance of the agent finding a local optimum in a shaped reward, the agent learns the safer behavior of pausing after achieving a certain reward. Explicitly weighting behavior cloning steps into gradient updates forces the policy to continue the task.

5.8.2 *Q-Filter*

The Q-Filter is effective in accelerating learning and achieving optimal performance. Fig. 24 shows that the method without filtering is slower to learn. One issue with the behavior cloning loss is that if the demonstrations are suboptimal, the learned policy will also be suboptimal. Filtering by Q-value gives a natural way to anneal the effect of the demonstrations as it automatically disables the BC loss when a better action is found. However, it gives mixed results on the longer horizon tasks. One explanation is that in the step reward case, learning relies less on the demonstrations because the reward signal is stronger. Therefore, the training is less affected by suboptimal demonstrations.

5.8.3 *Resets From Demonstrations*

We find that initializing rollouts from within demonstration states greatly helps to learn to stack 5 and 6 blocks but hurts training with fewer blocks, as shown in Fig. 25. Note that even where resets from demonstration states helps the final success rate, learning takes off faster when this technique is not used. However, since stacking the tower higher is risky, the agent learns the safer behavior of stopping after achieving a certain reward. Resetting from demonstration states alleviates this problem because the agent regularly experiences higher rewards.

This method changes the sampled state distribution, biasing it towards later states. It also inflates the Q values unrealistically. Therefore, on tasks where the RL algorithm does not get stuck in solving a subset of the full problem, it could hurt performance.

5.9 DISCUSSION AND FUTURE WORK

We present a system to utilize demonstrations along with reinforcement learning to solve complicated multi-step tasks. We believe this can accelerate learning of many tasks, especially those with sparse rewards or other difficulties in exploration. Our method is very general, and can be applied on any continuous control task where a success criterion can be specified and demonstrations obtained.

An exciting future direction is to train policies directly on a physical robot. Fig. 22 shows that learning the pick-and-place task takes about 1 million timesteps, which is about 6 hours of real world interaction time. This can realistically be trained on a physical robot, short-cutting the simulation-reality gap entirely. Many automation tasks found in factories and warehouses are similar to pick-and-place but without the variation in initial and goal states, so the samples required could be much lower. With our method, no expert needs to be in the loop to train these systems: demonstrations can be collected by users without knowledge about machine learning or robotics and rewards could be directly obtained from human feedback.

A major limitation of this work is sample efficiency on solving harder tasks. While we could not solve these tasks with other learning methods, our method requires a large amount of experience which is impractical outside of simulation. To run these tasks on physical robots, the sample efficiency will have to improved considerably. We also require demonstrations which are not easy to collect for all tasks. If demonstrations are not available but the environment can be reset to arbitrary states, one way to learn goal-reaching but avoid using demonstrations is to reuse successful rollouts as in (Florensa et al., 2018b).

Finally, our method of resets from demonstration states requires the ability to reset to arbitrary states. Although we can solve many long-horizon tasks without this ability, it is very effective for the hardest tasks. Resetting from demonstration rollouts resembles curriculum learning: we solve a hard task by first solving easier tasks. If the environment does not afford setting arbitrary states, then other curriculum methods will have to be used.

6

AWAC: ACCELERATING ONLINE REINFORCEMENT LEARNING WITH OFFLINE DATASETS

6.1 INTRODUCTION

Learning models that generalize effectively to complex open-world settings, from image recognition (Krizhevsky et al., 2012) to natural language processing (Devlin et al., 2019), relies on large, high-capacity models as well as large, diverse, and representative datasets. Leveraging this recipe of pre-training from large-scale offline datasets has the potential to provide significant benefits for reinforcement learning (RL) as well, both in terms of generalization and sample complexity. But most existing RL algorithms collect data online from scratch every time a new policy is learned, which can quickly become impractical in domains like robotics where physical data collection has a non-trivial cost. In the same way that powerful models in computer vision and NLP are often pre-trained on large, general-purpose datasets and then fine-tuned on task-specific data, practical instantiations of reinforcement learning for real world robotics problems will need to be able to incorporate large amounts of prior data effectively into the learning process, while still collecting additional data online for the task at hand. Doing so effectively will make the online data collection process much more practical while still allowing robots operating in the real world to continue improving their behavior.

For data-driven reinforcement learning, offline datasets consist of trajectories of states, actions and associated rewards. This data can potentially come from demonstrations for the desired task (Schaal, 1997; Atkeson and Schaal, 1997), suboptimal policies (Gao et al., 2018), demonstrations for related tasks (Zhou et al., 2019), or even just random exploration in the environment. Depending on the quality of the data that is provided, useful knowledge can be extracted about the dynamics of the world, about the task being solved, or both. Effective data-driven methods for deep reinforcement learning should be able to use this data to pre-train offline while improving with online fine-tuning.

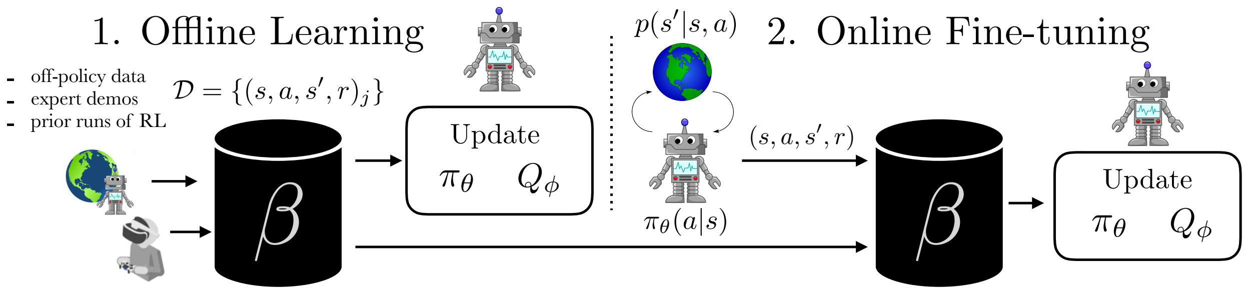


Figure 26: We study learning policies by offline learning on a prior dataset $\mathcal{D}$ and then fine-tuning with online interaction. The prior data could be obtained via prior runs of RL, expert demonstrations, or any other source of transitions. Our method, advantage weighted actor critic (AWAC) is able to learn effectively from offline data and fine-tune in order to reach expert-level performance after collecting a limited amount of interaction data. Videos and data are available at *awacrl.github.io*

Since this prior data can come from a variety of sources, we would like to design an algorithm that does not utilize different types of data in any privileged way. For example, the previous chapter incorporated demonstrations into RL directly by aiming to mimic these demonstrations (A. Nair et al., 2018a), which is desirable when the demonstrations are known to be optimal, but imposes strict requirements on the type of offline data, and can cause undesirable bias when the prior data is not optimal. While prior methods for fully offline RL provide a mechanism for utilizing offline data (Fujimoto et al., 2019a; Kumar et al., 2019a), as we will show in our experiments, such methods generally are not effective for fine-tuning with online data as they are often too conservative. In effect, prior methods require us to choose: Do we assume prior data is optimal or not? Do we use only offline data, or only online data? To make it feasible to learn policies for open-world settings, we need algorithms that learn successfully in any of these cases.

In this work, we study how to build RL algorithms that are effective for pre-training from off-policy datasets, but also well suited to continuous improvement with online data collection. We systematically analyze the challenges with using standard off-policy RL algorithms (Haarnoja et al., 2018a; Kumar et al., 2019a; Abdolmaleki et al., 2018) for this problem, and introduce a simple actor critic algorithm that elegantly bridges data-driven pre-training from offline data and improvement with online data collection. Our method, which uses dynamic programming to train a critic but a supervised learning style update to train a constrained actor, combines the best of supervised learning and actor-critic algorithms. Dynamic programming can leverage off-policy data and enable sample-efficient learning. The simple supervised actor update implicitly enforces a constraint that mitigates the effects of distribution shift when learning from offline

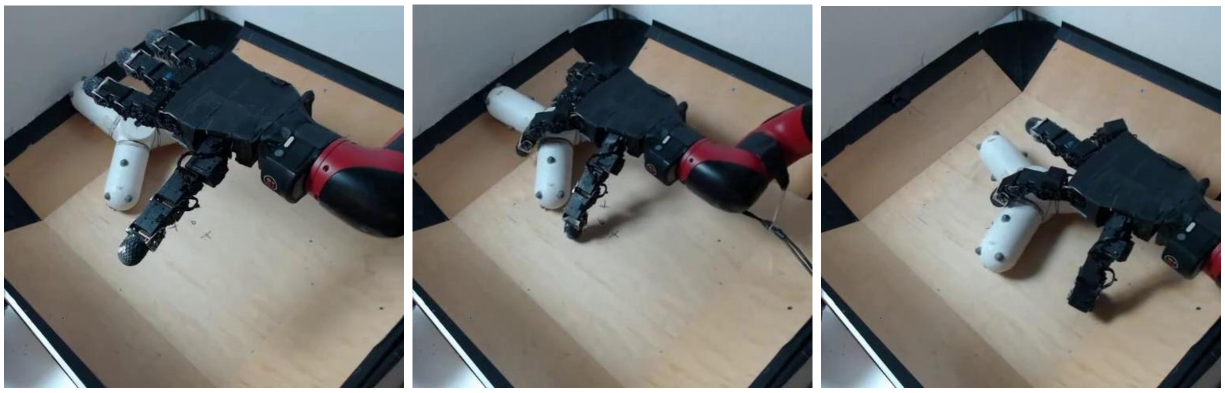


Figure 27: Utilizing prior data for online learning allows us to solve challenging real-world robotics tasks, such as this dexterous manipulation task where the learned policy must control a 4-fingered hand to reposition an object.

data (Fujimoto et al., 2019a; Kumar et al., 2019a), while avoiding overly conservative updates.

We evaluate our algorithm on a wide variety of robotic control tasks, using a set of simulated dexterous manipulation problems as well as three separate real-world robots: drawer opening with a 7-DoF robotic arm, picking up an object with a multi-fingered hand, and rotating a valve with a 3-fingered claw. Our algorithm, Advantage Weighted Actor Critic (AWAC), is able to quickly learn successful policies for these challenging tasks, in spite of high dimensional action spaces and uninformative, sparse reward signals. We show that AWAC finetunes much more efficiently after offline pretraining as compared to prior methods and, given a fixed time budget, attains significantly better performance on the real-world tasks. Moreover, AWAC can utilize different types of prior data without any algorithmic changes: demonstrations, suboptimal data, or random exploration data. The contribution of this work is not just another RL algorithm, but a systematic study of what makes offline pre-training with online fine-tuning unique compared to the standard RL paradigm, which then directly motivates a simple algorithm, AWAC, to address these challenges. We additionally discuss the design decisions required for applying AWAC as a practical tool for real-world robotic skill learning.

6.2 PRELIMINARIES

Recall the standard reinforcement learning notation, with states $\mathbf{s}$, actions $\mathbf{a}$, policy $\pi(\mathbf{a}|\mathbf{s})$, rewards $r(\mathbf{s}, \mathbf{a})$, and dynamics $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$. The discounted return is defined as $R_t = \sum_{i=t}^{T} \gamma^i r(\mathbf{s}_i, \mathbf{a}_i)$, for a discount factor γ and horizon T which may be infinite. The objective of an RL agent is to maximize the expected discounted return $J(\pi) = \mathbb{E}_{p_\pi(\tau)}[R_0]$ under the distribution induced by the policy. The optimal policy can be learned directly (e.g., using policy gradient to estimate $\nabla J(\pi)$ (R. J. Williams, 1992)), but this is often ineffective due to high variance of the estimator. Many algorithms attempt to reduce this variance by making use of the value function $V^\pi(\mathbf{s}) = \mathbb{E}_{p_\pi(\tau)}[R_t|\mathbf{s}]$, action-value function $Q^\pi(\mathbf{s}, \mathbf{a}) = \mathbb{E}_{p_\pi(\tau)}[R_t|\mathbf{s}, \mathbf{a}]$, or advantage $A^\pi(\mathbf{s}, \mathbf{a}) = Q^\pi(\mathbf{s}, \mathbf{a}) - V^\pi(\mathbf{s})$. The action-value function for a policy can be written recursively via the Bellman equation:

$$\begin{aligned} Q^\pi(\mathbf{s}, \mathbf{a}) &= r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{p(\mathbf{s}'|\mathbf{s},\mathbf{a})}[V^\pi(\mathbf{s}')] && (23) \\ &= r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{p(\mathbf{s}'|\mathbf{s},\mathbf{a})}[\mathbb{E}_{\pi(\mathbf{a}'|\mathbf{s}')}[Q^\pi(\mathbf{s}', \mathbf{a}')]]. && (24) \end{aligned}$$

Instead of estimating policy gradients directly, actor-critic algorithms maximize returns by alternating between two phases (Konda and Tsitsiklis, 2000): policy evaluation and policy improvement. During the policy evaluation phase, the critic $Q^\pi(\mathbf{s}, \mathbf{a})$ is estimated for the current policy π. This can be accomplished by repeatedly applying the Bellman operator $\mathcal{B}$, corresponding to the right-hand side of Equation 24, as defined below:

$$\mathcal{B}^\pi Q(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{p(\mathbf{s}'|\mathbf{s},\mathbf{a})}[\mathbb{E}_{\pi(\mathbf{a}'|\mathbf{s}')}[Q^\pi(\mathbf{s}', \mathbf{a}')]]. \quad (25)$$

By iterating according to $Q^{k+1} = \mathcal{B}^\pi Q^k$, Q^k converges to Q^π (R. S. Sutton and Barto, 1998). With function approximation, we cannot apply the Bellman operator exactly, and instead minimize the Bellman error with respect to Q-function parameters ϕ_k:

$$\begin{aligned} \phi_k &= \arg\min_\phi \ \mathbb{E}_\mathcal{D}[(Q_\phi(\mathbf{s}, \mathbf{a}) - y)^2], && (26) \\ y &= r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{\mathbf{s}',\mathbf{a}'}[Q_{\phi_{k-1}}(\mathbf{s}', \mathbf{a}')]. && (27) \end{aligned}$$

During policy improvement, the actor π is typically updated based on the current estimate of Q^π. A commonly used technique (Lillicrap et al., 2016; Fujimoto et al., 2018b; Haarnoja et al., 2018a) is to update the actor $\pi_{\theta_k}(\mathbf{a}|\mathbf{s})$ via likelihood ratio or pathwise derivatives to optimize the following objective, such that the expected value of the Q-function Q^π is maximized:

$$\theta_k = \arg\max_\theta \mathbb{E}_{\mathbf{s}\sim\mathcal{D}}[\mathbb{E}_{\pi_\theta(\mathbf{a}|\mathbf{s})}[Q_{\phi_k}(\mathbf{s}, \mathbf{a})]] \quad (28)$$

Actor-critic algorithms are widely used in deep RL (Mnih et al., 2016; Lillicrap et al., 2016; Haarnoja et al., 2018a; Fujimoto et al., 2018b). With a Q-function estimator, they can in principle utilize off-policy data when used with a replay buffer for storing prior transition tuples, which we will denote β, to sample previous transitions, although we show that this by itself is insufficient for our problem setting.

6.3 CHALLENGES IN OFFLINE RL WITH ONLINE FINE-TUNING

In this section, we study the unique challenges that exist when pre-training using offline data, followed by fine-tuning with online data collection. We first describe the problem, and then analyze what makes this problem difficult for prior methods.

6.3.1 *Problem Definition*

A static dataset of transitions, $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)_j\}$, is provided to the algorithm at the beginning of training. This dataset can be sampled from an arbitrary policy or mixture of policies, and may even be collected by a human expert. This definition is general and encompasses many scenarios: learning from demonstrations, random data, prior RL experiments, or even from multi-task data. Given the dataset $\mathcal{D}$, our goal is to leverage $\mathcal{D}$ for pre-training and use a small amount of online interaction to learn the optimal policy $\pi^*(\mathbf{a}|\mathbf{s})$, as depicted in Fig 26. This setting is representative of many real-world RL settings, where prior data is available and the aim is to learn new skills efficiently. We first study existing algorithms empirically in this setting on the HalfCheetah-v2 Gym environment[1]. The prior dataset consists of 15 demonstrations from an expert policy and 100 suboptimal trajectories sampled from a behavioral clone of these demonstrations. All methods for the remainder of this chapter incorporate the prior dataset, unless explicitly labeled "scratch".

6.3.2 *Data Efficiency*

One of the simplest ways to utilize prior data such as demonstrations for RL is to pre-train a policy with imitation learning, and fine-tune with on-policy RL (Abhishek Gupta et al., 2019a; Rajeswaran et al., 2018). This approach has two drawbacks: (1) prior data may not be optimal; (2) on-policy fine-tuning is data inefficient as it does not reuse the prior data in the RL stage. In our setting, data efficiency is vital. To this end, we require

1 We use this environment for analysis because it helps understand and accentuate the differences between different algorithms. More challenging environments like the ones shown in Fig 29 are too hard to solve to analyze variants of different methods.

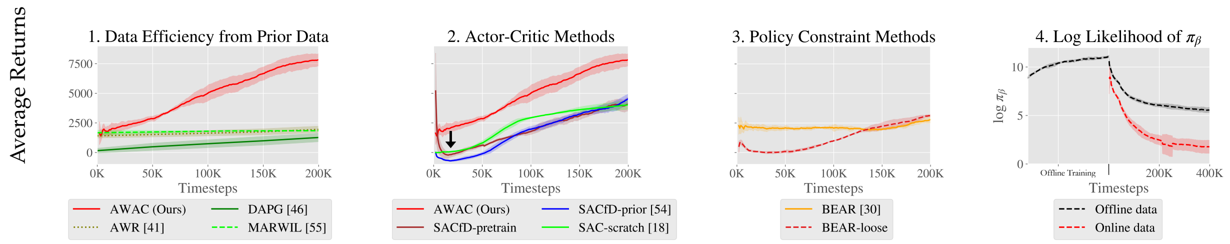


Figure 28: Analysis of prior methods on HalfCheetah-v2 using offline RL with online fine-tuning. (1) On-policy methods (DAPG, AWR, MARWIL) learn relatively slowly, even with access to prior data. We present our method, AWAC, as an example of how off-policy RL methods can learn much faster. (2) Variants of soft actor-critic (SAC) with offline training (performed before timestep 0) and fine-tuning. We see a "dip" in the initial performance, even if the policy is pretrained with behavioral cloning. (3) Offline RL method BEAR (Kumar et al., 2019a) on offline training and fine-tuning, including a "loose" variant of BEAR with a weakened constraint. Standard offline RL methods fine-tune slowly, while the "loose" BEAR variant experiences a similar dip as SAC. (4) We show that the fit of the behavior models $\hat{\pi}_\beta$ used by these offline methods degrades as new data is added to the buffer during fine-tuning, potentially explaining their poor fine-tuning performance.

algorithms that are able to reuse arbitrary off-policy data during online RL for data-efficient fine-tuning. We find that algorithms that use on-policy fine-tuning (Rajeswaran et al., 2018; Abhishek Gupta et al., 2019a), or Monte-Carlo return estimation (Peters and Schaal, 2007a; Q. Wang et al., 2018a; Peng et al., 2019a) are generally much less efficient than off-policy actor-critic algorithms, which iterate between improving π and estimating Q^π via Bellman backups. This can be seen from the results in Figure 28 plot 1, where on-policy methods like DAPG (Rajeswaran et al., 2018) and Monte-Carlo return methods like AWR (Peng et al., 2019a) and MARWIL (Q. Wang et al., 2018a) are an order of magnitude slower than off-policy actor-critic methods. Actor-critic methods, shown in Figure 28 plot 2, can in principle use off-policy data. However, as we will discuss next, naïvely applying these algorithms to our problem suffers from a different set of challenges.

6.3.3 *Bootstrap Error in Offline Learning with Actor-Critic Methods*

When standard off-policy actor-critic methods are applied to this problem setting, they perform poorly, as shown in the second plot in Figure 28: despite having a prior dataset in the replay buffer, these algorithms do not benefit significantly from offline training. We evaluate soft actor critic (Haarnoja et al., 2018a), a state-of-the-art actor-critic algorithm for continuous control. Note that "SAC-scratch," which does not receive the prior data, performs similarly to "SACfD-prior," which does have access to the prior data, indicating that the off-policy RL algorithm is not actually able to make use of the off-policy data for pre-training. Moreover, even if the SAC is policy is pre-trained by behavior cloning, labeled "SACfD-pretrain", we still observe an initial decrease in performance, and performance similar to learning from scratch.

This challenge can be attributed to off-policy bootstrapping error accumulation, as observed in several prior works (R. S. Sutton and Barto, 1998; Kumar et al., 2019a; Yifan Wu et al., 2020; Levine et al., 2020; Fujimoto et al., 2019a). In actor-critic algorithms, the target value $Q(\mathbf{s}', \mathbf{a}')$, with $\mathbf{a}' \sim \pi$, is used to update $Q(\mathbf{s}, \mathbf{a})$. When $\mathbf{a}'$ is outside of the data distribution, $Q(\mathbf{s}', \mathbf{a}')$ will be inaccurate, leading to accumulation of error on static datasets.

Offline RL algorithms (Fujimoto et al., 2019a; Kumar et al., 2019a; Yifan Wu et al., 2020) propose to address this issue by explicitly adding constraints on the policy improvement update (Equation 28) to avoid bootstrapping on out-of-distribution actions, leading to a policy update of this form:

$$\arg\max_{\theta} \mathbb{E}_{\mathbf{s}\sim\mathcal{D}}[\mathbb{E}_{\pi_\theta(\mathbf{a}|\mathbf{s})}[Q_{\phi_k}(\mathbf{s}, \mathbf{a})]] \text{ s.t. } D(\pi_\theta, \pi_\beta) \leqslant \epsilon. \tag{29}$$

Here, π_θ is the actor being updated, and $\pi_\beta(a|s)$ represents the (potentially unknown) distribution from which all of the data seen so far (both offline data and online data) was generated. In the case of a replay buffer, π_β corresponds to a mixture distribution over all past policies. Typically, π_β is not known, especially for offline data, and must be estimated from the data itself. Many offline RL algorithms (Kumar et al., 2019a; Fujimoto et al., 2019a; Noah Y. Siegel et al., 2020b) explicitly fit a parametric model to samples for the distribution π_β via maximum likelihood estimation, where samples from π_β are obtained simply by sampling uniformly from the data seen thus far: $\hat{\pi}_\beta = \max_{\hat{\pi}_\beta} \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\pi_\beta}[\log \hat{\pi}_\beta(\mathbf{a}|\mathbf{s})]$. After estimating $\hat{\pi}_\beta$, prior methods implement the constraint given in Equation 29 in various ways, including penalties on the policy update (Kumar et al., 2019a; Yifan Wu et al., 2020) or architecture choices for sampling actions for policy training (Fujimoto et al., 2019a; Noah Y. Siegel et al., 2020b). As we will see next, the requirement for accurate estimation of $\hat{\pi}_\beta$ makes these methods difficult to use with online fine-tuning.

6.3.4 *Excessively Conservative Online Learning*

While offline RL algorithms with constraints (Kumar et al., 2019a; Fujimoto et al., 2019a; Yifan Wu et al., 2020) perform well offline, they struggle to improve with fine-tuning, as shown in the third plot in Figure 28. We see that the purely offline RL performance (at "0K" in Fig. 28) is much better than the standard off-policy methods shown in Section 6.3.3. However, with additional iterations of online fine-tuning, the performance increases very slowly (as seen from the slope of the BEAR curve in Fig 28). What causes this phenomenon?

This can be attributed to challenges in fitting an accurate behavior model as data is collected online during fine-tuning. In the offline setting, behavior models must only be trained once via maximum likelihood, but in the online setting, the behavior model must be updated online to track incoming data. Training density models online (in the "streaming" setting) is a challenging research problem (Ramapuram et al., 2017), made more difficult by a potentially complex multi-modal behavior distribution induced by the mixture of online and offline data. To understand this, we plot the log likelihood of learned behavior models on the dataset during online and offline training for the HalfCheetah task. As we can see in the plot, the accuracy of the behavior models ($\log \pi_\beta$ on the y-axis) reduces during online fine-tuning, indicating that it is not fitting the new data well during online training. When the behavior models are inaccurate or unable to model new data well, constrained optimization becomes too conservative, resulting in limited improvement with fine-tuning. This analysis suggests that, in order to address our problem setting, we require an off-policy RL algorithm that constrains the policy to prevent offline instability and error accumulation, but not so conservatively that it prevents online fine-tuning due to imperfect behavior modeling. Our proposed algorithm, which we discuss in the next section, accomplishes this by employing an *implicit* constraint, which does not require *any* explicit modeling of the behavior policy.

6.4 ADVANTAGE WEIGHTED ACTOR CRITIC: A SIMPLE ALGORITHM FOR FINE-TUNING FROM OFFLINE DATASETS

In this section, we will describe the advantage weighted actor-critic (AWAC) algorithm, which trains an off-policy critic and an actor with an *implicit* policy constraint. We will show AWAC mitigates the challenges outlined in Section 6.3. AWAC follows the design for actor-critic algorithms as described in Section 11.3, with a policy evaluation step to learn Q^π and a policy improvement step to update π. AWAC uses off-policy temporal-difference learning to estimate Q^π in the policy evaluation step, and a policy improvement update that is able to obtain the benefits of offline RL algorithms at training from prior datasets, while avoiding the overly conservative behavior described in Section 6.3.4. We describe the policy improvement step in AWAC below, and then summarize the entire algorithm.

Policy improvement for AWAC proceeds by learning a policy that maximizes the value of the critic learned in the policy evaluation step via TD bootstrapping. If done naively, this can lead to the issues described in Section 6.3.4, but we can avoid the challenges of bootstrap error accumulation by restricting the policy distribution to stay close to the data observed thus far during the actor update, while maximizing the value of the critic. At iteration k, AWAC therefore optimizes the policy to maximize the estimated Q-function $Q^{\pi_k}(\mathbf{s}, \mathbf{a})$ at every state, while constraining it to stay close to the actions observed in the data, similar to prior offline RL methods, though this constraint will be enforced differently. Note from the definition of the advantage in Section 11.3 that optimizing $Q^{\pi_k}(\mathbf{s}, \mathbf{a})$ is equivalent to optimizing $A^{\pi_k}(\mathbf{s}, \mathbf{a})$. We can therefore write this optimization as:

$$\pi_{k+1} = \arg\max_{\pi \in \Pi} \mathbb{E}_{\mathbf{a} \sim \pi(\cdot|\mathbf{s})}[A^{\pi_k}(\mathbf{s}, \mathbf{a})] \tag{30}$$

$$\text{s.t. } D_{KL}(\pi(\cdot|\mathbf{s}) || \pi_\beta(\cdot|\mathbf{s})) \leqslant \epsilon. \tag{31}$$

As we saw in Section 6.3.3, enforcing the constraint by incorporating an explicit learned behavior model (Kumar et al., 2019a; Fujimoto et al., 2019a; Yifan Wu et al., 2020; Noah Y. Siegel et al., 2020b) leads to poor fine-tuning performance. Instead, we enforce the constraint *implicitly*, without learning a behavior model. We first derive the solution to the constrained optimization in Equation 30 to obtain a non-parametric closed form for the actor. This solution is then projected onto the parametric policy class *without* any explicit behavior model. The analytic solution to Equation 30 can be obtained by enforcing the KKT conditions (Peters and Schaal, 2007a; Peters et al., 2010; Peng et al., 2019a). The Lagrangian is:

$$\mathcal{L}(\pi, \lambda) = \mathbb{E}_{\mathbf{a} \sim \pi(\cdot|\mathbf{s})}[A^{\pi_k}(\mathbf{s}, \mathbf{a})] + \lambda(\epsilon - D_{KL}(\pi(\cdot|\mathbf{s}) || \pi_\beta(\cdot|\mathbf{s}))), \tag{32}$$

and the closed form solution to this problem is $\pi^*(\mathbf{a}|\mathbf{s}) \propto \pi_\beta(\mathbf{a}|\mathbf{s}) \exp\left(\frac{1}{\lambda} A^{\pi_k}(\mathbf{s}, \mathbf{a})\right)$. When using function approximators, such as deep neural networks as we do, we need to project the non-parametric solution into our policy space. For a policy π_θ with parameters θ, this can be done by minimizing the KL divergence of π_θ from the optimal non-parametric solution π^* under the data distribution $\rho_{\pi_\beta}(\mathbf{s})$:

$$\arg\min_{\theta} \ {}_{\rho_{\pi_\beta}(\mathbf{s})} \left[D_{\mathrm{KL}}(\pi^*(\cdot|\mathbf{s})||\pi_\theta(\cdot|\mathbf{s}))\right] \tag{33}$$

$$= \arg\min_{\theta} \ {}_{\rho_{\pi_\beta}(\mathbf{s})} \left[{}_{\pi^*(\cdot|\mathbf{s})}[-\log \pi_\theta(\cdot|\mathbf{s})]\right] \tag{34}$$

Note that the parametric policy could be projected with either direction of KL divergence. Choosing the reverse KL results in explicit penalty methods (Yifan Wu et al., 2020) that rely on evaluating the density of a learned behavior model. Instead, by using forward KL, we can compute the policy update by sampling directly from β:

$$\theta_{k+1} = \arg\max_{\theta} \ {}_{\mathbf{s},\mathbf{a}\sim\beta} \left[\log \pi_\theta(\mathbf{a}|\mathbf{s}) \exp\left(\frac{1}{\lambda} A^{\pi_k}(\mathbf{s},\mathbf{a})\right)\right]. \tag{35}$$

This actor update amounts to weighted maximum likelihood (i.e., supervised learning), where the targets are obtained by re-weighting the state-action pairs observed in the current dataset by the predicted advantages from the learned critic, *without* explicitly learning any parametric behavior model, simply sampling (s, a) from the replay buffer β. See Appendix D.1 for a more detailed derivation and Appendix D.2 for specific implementation details.

Avoiding explicit behavior modeling. Note that the update in Equation 35 completely avoids any modeling of the previously observed data β with a parametric model. By avoiding any explicit learning of the behavior model AWAC is far less conservative than methods which fit a model $\hat{\pi}_\beta$ explicitly, and better incorporates new data during online fine-tuning, as seen from our results in Section 9.5. This derivation is related to AWR (Peng et al., 2019a), with the main difference that AWAC uses an off-policy Q-function Q^π to estimate the advantage, which greatly improves efficiency and even final performance (see results in Section 6.6.1.1). The update also resembles ABM-MPO, but ABM-MPO *does* require modeling the behavior policy which, as discussed in Section 6.3.4, can lead to poor fine-tuning. In Section 6.6.1.1, AWAC outperforms ABM-MPO on a range of challenging tasks.

Policy evaluation. During policy evaluation, we estimate the action-value $Q^\pi(\mathbf{s},\mathbf{a})$ for the current policy π, as described in Section 11.3. We utilize a temporal difference learning scheme for policy evaluation (Haarnoja et al., 2018a; Fujimoto et al., 2018b), minimizing the Bellman error as described in Equation 25. This enables us to learn very efficiently from off-policy data. This is particularly important in our problem setting to effectively use the offline dataset, and allows us to significantly outperform alternatives using Monte-Carlo evaluation or TD(λ) to estimate returns (Peng et al., 2019a).

Algorithm 4 Advantage Weighted Actor Critic (AWAC)

```
1: Dataset $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)_j\}$
2: Initialize buffer $\beta = \mathcal{D}$
3: Initialize $\pi_\theta$, $Q_\phi$
4: for iteration $i = 1, 2, ...$ do
5:     Sample batch $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r) \sim \beta$
6:     $y = r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{\mathbf{s}', \mathbf{a}'}[Q_{\phi_{k-1}}(\mathbf{s}', \mathbf{a}')]$
7:     $\phi \leftarrow \arg\min_\phi \ \mathbb{E}_\mathcal{D}[(Q_\phi(\mathbf{s}, \mathbf{a}) - y)^2]$
8:     $\theta \leftarrow \arg\max_\theta \ _{\mathbf{s}, \mathbf{a} \sim \beta} \left[\log \pi_\theta(\mathbf{a}|\mathbf{s}) \exp\left(\frac{1}{\lambda} A^{\pi_k}(\mathbf{s}, \mathbf{a})\right)\right]$
9:     if $i >$ num_offline_steps then
10:        $\tau_1, \ldots, \tau_K \sim p_{\pi_\theta}(\tau)$
11:        $\beta \leftarrow \beta \cup \{\tau_1, \ldots, \tau_K\}$
12:    end if
13: end for=0
```

Algorithm summary. The full AWAC algorithm for offline RL with online fine-tuning is summarized in Algorithm 4. In a practical implementation, we can parameterize the actor and the critic by neural networks and perform SGD updates from Eqn. 35 and Eqn. 26. Specific details are provided in Appendix D.2. AWAC ensures data efficiency with off-policy critic estimation via bootstrapping, and avoids offline bootstrap error with a constrained actor update. By avoiding explicit modeling of the behavior policy, AWAC avoids overly conservative updates.

While AWAC is related to several prior RL algorithms, we note that there are key differences that make it particularly amenable to the problem setting we are considering – offline RL with online fine-tuning – that other methods are unable to tackle. As we show in our experimental analysis with direct comparisons to prior work, every one of the design decisions being made in this work are important for algorithm performance. As compared to AWR (Peng et al., 2019a), AWAC uses TD bootstrapping for significantly more efficient and even asymptotically better performance. As compared to offline RL techniques like ABM (Noah Y. Siegel et al., 2020b), MPO (Abdolmaleki et al., 2018), BEAR (Kumar et al., 2019a) or BCQ (Fujimoto et al., 2019a) this work is able to avoid the need for any behavior modeling, thereby enabling the *online* fine-tuning part of the problem much better. As shown in Fig 29, when these seemingly ablations are made to AWAC, the algorithm performs significantly worse.

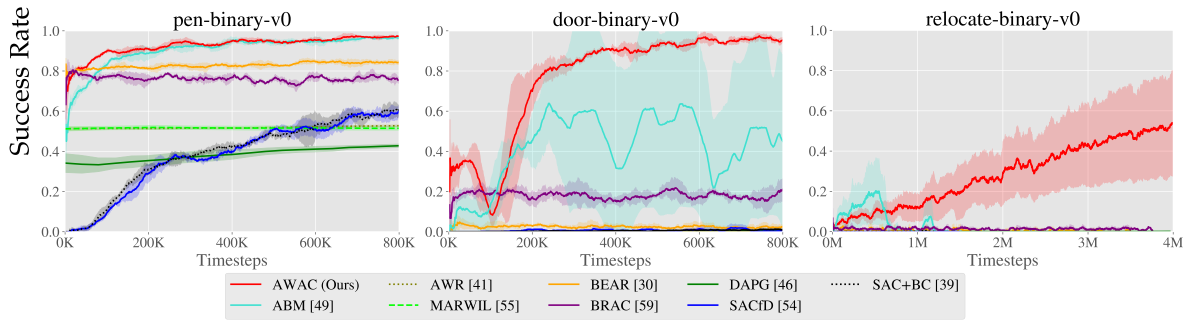


Figure 29: Comparative evaluation on the dexterous manipulation tasks. These tasks are difficult due to their high action dimensionality and reward sparsity. We see that AWAC is able to learn these tasks with little online data collection required (100K samples ≈ 16 minutes of equivalent real-world interaction time). Meanwhile, most prior methods are not able to solve the harder two tasks: door opening and object relocation.

6.5 RELATED WORK

Off-policy RL algorithms are designed to reuse off-policy data during training, and have been studied extensively (Konda and Tsitsiklis, 2000; Degris et al., 2012; Mnih et al., 2016; Haarnoja et al., 2018a; Fujimoto et al., 2018b; Bhatnagar et al., 2009; Peters and Schaal, 2008a; S. Zhang et al., 2019; Wawrzynski, 2009; Balduzzi and Ghifary, 2015). While standard off-policy methods are able to benefit from including data seen *during* a training run, as we show in Section 6.3.3 they struggle when training from previously collected offline data from other policies, due to error accumulation with distribution shift (Fujimoto et al., 2019a; Kumar et al., 2019a). Offline RL methods aim to address this issue, often by constraining the actor updates to avoid excessive deviation from the data distribution (Lange et al., 2012b; P. S. Thomas and Brunskill, 2016; Hallak et al., 2015; Hallak et al., 2016; Hallak and Mannor, 2017; Agarwal et al., 2019; Kumar et al., 2019a; Fujimoto et al., 2019a; Fakoor et al., 2019; Nachum et al., 2019; Noah Y. Siegel et al., 2020b; Levine et al., 2020; Ruiyi Zhang et al., 2020). One class of these methods utilize importance sampling (P. S. Thomas and Brunskill, 2016; Ruiyi Zhang et al., 2020; Nachum et al., 2019; Degris et al., 2012; Jiang and L. Li, 2016; Hallak and Mannor, 2017). Another class of methods perform offline reinforcement learning via dynamic programming, with an explicit constraint to prevent deviation from the data distribution (Lange et al., 2012b; Kumar et al., 2019a; Fujimoto et al., 2019a; Yifan Wu et al., 2020; Jaques et al., 2019). While these algorithms perform well in the purely offline settings, we show in Section 6.3.4 that such methods tend to be overly conservative, and therefore may not learn efficiently when fine-tuning with online data collection. In contrast, our algorithm AWAC is comparable to these algorithms for offline pre-training, but learns much more efficiently during subsequent fine-tuning.

Prior work has also considered the special case of learning from *demonstration* data. One class of algorithms initializes the policy via behavioral cloning from demonstrations, and then fine-tunes with reinforcement learning (Peters and Schaal, 2008c; Ijspeert et al., 2002; Theodorou et al., 2010; Kim et al., 2013; Rajeswaran et al., 2018; Abhishek Gupta et al., 2019a; H. Zhu et al., 2019). Most such methods use on-policy fine-tuning, which is less sample-efficient than off-policy methods that perform value function estimation. Other prior works have incorporated demonstration data into the replay buffer using off-policy RL methods (Veerk et al., 2017; A. Nair et al., 2017). We show in Section 6.3.3 that these strategies can result in a large dip in performance during online fine-tuning, due to the inability to pre-train an effective value function from offline data. In contrast, our work shows that using supervised learning style policy updates can allow for better bootstrapping from demonstrations as compared to Veerk et al. (2017) and A. Nair et al. (2017).

Our method builds on algorithms that implement a maximum likelihood objective for the actor, based on an expectation-maximization formulation of RL (Peters and Schaal, 2007a; Neumann and Peters, 2008; Theodorou et al., 2010; Peters et al., 2010; Peng et al., 2019a; Abdolmaleki et al., 2018; Q. Wang et al., 2018a). Most closely related to our method in this respect are the algorithms proposed by Peng et al. (2019a) (AWR) and Noah Y. Siegel et al. (2020b) (ABM). Unlike AWR, which estimates the value function of the *behavior* policy, V^{π_β} via Monte-Carlo estimation or TD$-\lambda$, our algorithm estimates the Q-function of the *current* policy Q^π via bootstrapping, enabling much more efficient learning, as shown in our experiments. Unlike ABM, our method does not require learning a separate function approximator to model the behavior policy π_β, and instead directly samples the dataset. As we discussed in Section 6.3.4, modeling π_β can be a major challenge for online fine-tuning. While these distinctions may seem somewhat subtle, they are important and we show in our experiments that they result in a large difference in algorithm performance. Finally, our work goes beyond the analysis in prior work, by studying the issues associated with pre-training and fine-tuning in Section 6.3. Closely to our work, Ziyu Wang et al. (2020b) proposed critic regularized regression for offline RL, which uses off-policy Q-learning and an equivalent policy update. In contrast to this concurrent work, we specifically study the offline pretraining online fine-tuning problem, which this prior work does not address, analyze why other methods are ineffective in this setting, and show that our approach enables strong fine-tuning results on challenging dextrous manipulation tasks and real-world robotic systems.

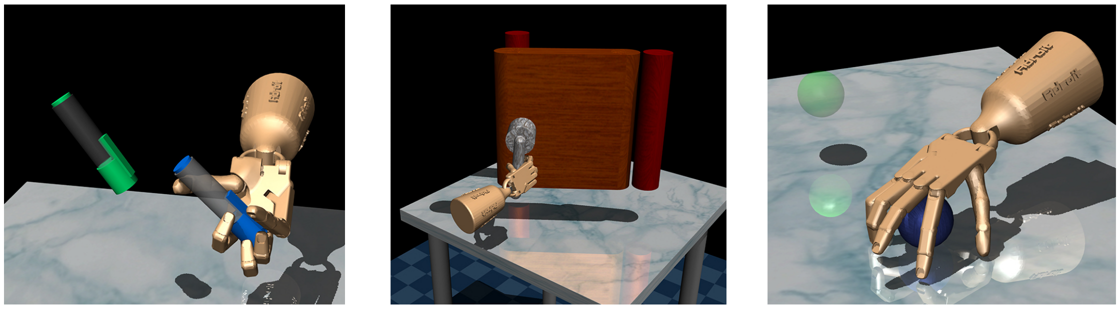


Figure 30: Illustration of dexterous manipulation tasks in simulation. These tasks exhibit sparse rewards, high-dimensional control, and complex contact physics.

The idea of bootstrapping learning from prior data for real-world robotic learning is not a new one; in fact, it has been extensively explored in the context of providing initial rollouts to bootstrap policy search (Kober and Peter, 2008; Peters and Schaal, 2008c; Kormushev et al., 2010), initializing dynamic motion primitives (Bentivegna et al., 2003; Kormushev et al., 2010; Mlling et al., 2013) in the context of on-policy reinforcement learning algorithms (Rajeswaran et al., 2018; H. Zhu et al., 2019), inferring reward shaping (Yuchen Wu et al., 2020) and even for inferring reward functions (Ziebart et al., 2008; Abbeel and Andrew Y Ng, 2004). Our work shows how we can generalize the idea of bootstrapping robotic learning from prior data to include arbitrary sub-optimal data rather than just demonstration data and shows the ability to continue improving beyond this data as well.

6.6 EXPERIMENTAL EVALUATION

In our experimental evaluation we aim to answer the following question:

1. Does AWAC effectively combine prior data with online experience to learn complex robotic control tasks more efficiently than prior methods?
2. Is AWAC able to learn from sub-optimal or random data?
3. Does AWAC provide a practical way to bootstrap real-world robotic reinforcement learning?

In the following sections, we study these questions using several challenging and high-dimensional simulated robotic tasks, as well as three separate real-world robotic platforms. Videos of all experiments are available at *awacrl.github.io*

6.6.1 *Simulated Experiments*

We study the first two questions in challenging simulation environments.

6.6.1.1 *Comparative Evaluation When Bootstrapping From Prior Data*

We study tasks in simulation that have significant exploration challenges, where offline learning and online fine-tuning are likely to be effective. We begin our analysis with a set of challenging sparse reward dexterous manipulation tasks proposed by Rajeswaran et al. (2018) in simulation. These tasks involve complex manipulation skills using a 28-DoF five-fingered hand in the MuJoCo simulator (Todorov et al., 2012) shown in Figure 29: in-hand rotation of a pen, opening a door by unlatching the handle, and picking up a sphere and relocating it to a target location. The reward functions in these environments are binary 0-1 rewards for task completion. [2] Rajeswaran et al. (2018) provide 25 human demonstrations for each task, which are not fully optimal but do solve the task. Since this dataset is small, we generated another 500 trajectories of interaction data by constructing a behavioral cloned policy, and then sampling from this policy.

First, we compare our method on these dexterous manipulation tasks against prior methods for off-policy learning, offline learning, and bootstrapping from demonstrations. Specific implementation details are discussed in Appendix D.4. The results are shown in Fig. 29. Our method is able to leverage the prior data to quickly attain good performance, and the efficient off-policy actor-critic component of our approach fine-tunes much more quickly than demonstration augmented policy gradient (DAPG), the method proposed by Rajeswaran et al. (2018). For example, our method solves the pen task in 120K timesteps, the equivalent of just 20 minutes of online interaction. While the baseline comparisons and ablations make some amount of progress on the pen task, alternative off-policy RL and offline RL algorithms are largely unable to solve the door and relocate task in the time-frame considered. We find that the design decisions to use off-policy critic estimation allow AWAC to outperform AWR (Peng et al., 2019a) while the implicit behavior modeling allows AWAC to significantly outperform ABM (Noah Y. Siegel et al., 2020b), although ABM does make some progress. Rajeswaran et al. (2018) show that DAPG can solve variants of these tasks with more well-shaped rewards, but requires considerably more samples.

2 Rajeswaran et al. (2018) use a combination of task completion factors as the sparse reward. For instance, in the door task, the sparse reward as a function of the door position d was $r = 10\mathbb{1}_{d>1.35} + 8\mathbb{1}_{d>1.0} + 2\mathbb{1}_{d>1.2} - 0.1\|d - 1.57\|_2$. We only use the fully sparse success measure $r = \mathbb{1}_{d>1.4}$, which is substantially more difficult.

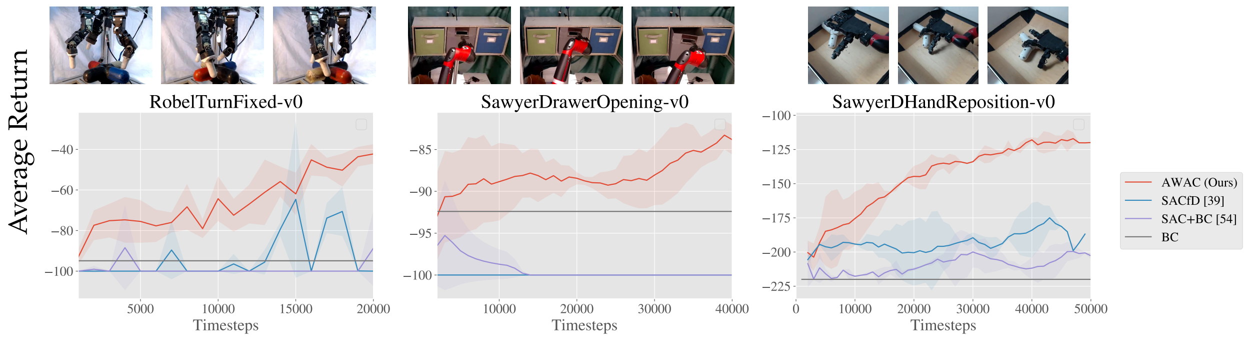


Figure 31: Algorithm comparison on three real-world robotic environments. Images of real world robotic tasks are pictured above. Left: a three fingered D'claw must rotate a valve 180°. Middle: a Sawyer robot must slide open a drawer using a hook attachment. Right: a dexterous hand attached to a Sawyer robot must re-position an object to to the center of the table. On each task, AWAC trained offline achieves reasonable performance (shown at timestep 0) and then steadily improves from online interaction. Other methods, which also all have access to prior data, fail to utilize the prior data effectively offline and therefore exhibit slow or no online improvement. Videos of all experiments are available at *awacrl.github.io*

Additionally, we evaluated all methods on the Gym MuJoCo locomotion benchmarks, similarly providing demonstrations as offline data. The results plots for these experiments are included in Appendix D.5 in the supplementary materials. These tasks are substantially easier than the sparse reward manipulation tasks described above, and a number of prior methods also perform well. SAC+BC and BRAC perform on par with our method on the HalfCheetah task, and ABM performs on par with our method on the Ant task, while our method outperforms all others on the Walker2D task. However, our method matches or exceeds the best prior method in all cases, whereas no other single prior method attains good performance on all tasks.

6.6.1.2 *Fine-Tuning from Random Policy Data*

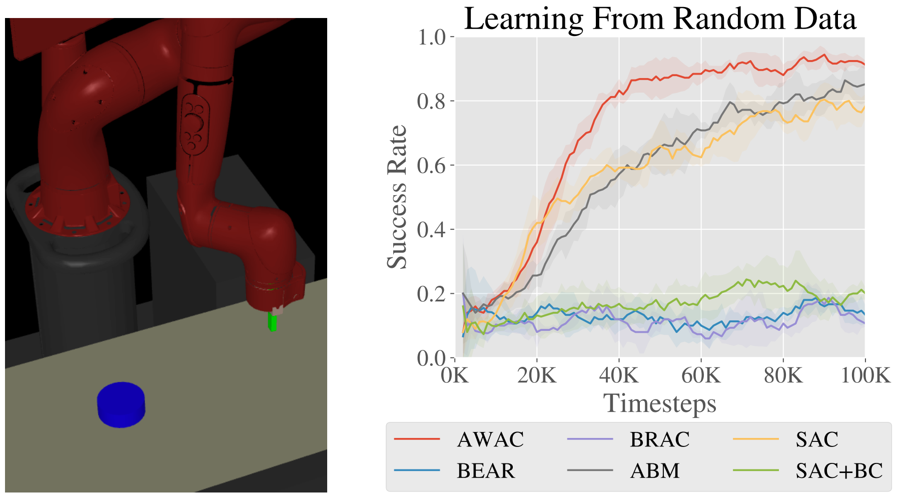


Figure 32: Comparison of fine-tuning from an initial dataset of suboptimal data on a Sawyer robot pushing task.

An advantage of using off-policy RL for reinforcement learning is that we can also incorporate suboptimal data, rather than demonstrations. In this experiment, we evaluate on a simulated tabletop pushing environment with a Sawyer robot pictured in Fig 29 and described further in Appendix D.3. To study the potential to learn from suboptimal data, we use an off-policy dataset of 500 trajectories generated by a random process. The task is to push an object to a target location in a 40cm x 20cm goal space. The results are shown in Figure 32. We see that while many methods begin at the same initial performance, AWAC learns the fastest online and is actually able to make use of the offline dataset effectively.

6.6.2 *Real-World Robot Learning with Prior Data*

We next evaluate AWAC and several baselines on a range of real-world robotic systems, shown in the top row of Fig 31. We study the following tasks: rotating a valve with a 3-fingered claw, repositioning an object with a 4-fingered hand, and opening a drawer with a Sawyer robotic arm. The dexterous manipulation tasks involve fine finger coordination to properly reorient and reposition objects, as well as high dimensional state and action spaces. The Sawyer drawer opening task requires accurate arm movements to properly hook the end-effector into the handle of the drawer. To ensure continuous operation, all environments are fitted with an automated reset mechanism that executes before each trajectory is collected, allowing us to run real-world experiments without human supervision. Since real-world experiments are significantly more time-consuming, we could not compare to the full range of prior methods in the real world, but we include comparisons with the following methods: direct behavioral cloning (BC) of the provided data (which is reasonable in these settings, since the prior data includes demonstrations), off-policy RL with soft actor-critic (SAC) (Haarnoja et al., 2018a), where the prior data is included in the replay buffer and used to pretrain the policy (which refer to as SACfD), and a modified version of SAC that includes an added behavioral cloning loss (SAC+BC), which is analogous to A. Nair et al. (2018a) or an off-policy version of Rajeswaran et al. (2018). Further implementation details of these algorithms are provided in Appendix D.4 in the supplementary materials.

Next, we describe the experimental setup for hardware experiments. Precise details of the hardware setup can be found in Appendix D.8 in the supplementary materials.

Dexterous Manipulation with a 3-Fingered Claw. This task requires controlling a 3-fingered, 9 DoF robotic hand, introduced by Ahn et al. (2019), to rotate a 4-pronged valve object by 180 degrees. To properly perform this task, multiple fingers need to coordinate to stably and efficiently rotate the valve into the desired orientation. The state space of the system consists of the joint angles of all the 9 joints in the claw, and the action space consists of the joint positions of the fingers, which are followed by the robot using a low-level PID controller. The reward for this task is sparse: -1 if the valve is rotated within 0.25 radians of the target, and 0 otherwise. Note that this reward function is significantly more difficult than the dense, well-shaped reward function typically used in prior work (Ahn et al., 2019). The prior data consists of 10 trajectories collected using kinesthetic teaching, combined this with 200 trajectories obtained through executing a policy trained via imitation learning in the environment.

Drawer Opening with a Sawyer Arm. This task requires controlling a Sawyer arm to slide open a drawer. The robot uses 3-dimensional end-effector control, and is equipped with a hook attachment to make the drawer opening possible. The state space is 4-dimensional, consisting of the position of the robot end-effector and the linear position of the drawer, measured using an encoder. The reward is sparse: -1 if the drawer is open beyond a threshold and 0 otherwise. For this task, the prior data consists of 10 demonstration trajectories collected using via teleoperation with a 3D mouse, as well as 500 trajectories obtained through executing a policy trained via imitation learning in the environment. This task is challenging because it requires very precise insertion of the hook attachment into the opening, as pictured in Fig 31, before the robot can open

the drawer. Due to the sparse reward, making learning progress on this task requires utilizing prior data to construct an initial policy that at least sometimes succeeds.
Dexterous Manipulation with a Robotic Hand. This task requires controlling a 4-fingered robotic hand mounted on a Sawyer robotic arm to reposition an object (Abhishek Gupta et al., 2021). The task requires careful coordination between the hand and the arm to manipulate the object accurately. The reward for this task is a combination of the negative distance between the hand and the object and the negative distance between the object and the target. The actions are 19-dimensional, consisting of 16-dimensional finger control and 3-dimensional end effector control of the arm. For this task, the prior data of 19 trajectories were collected using kinesthetic teaching and combined with 50 trajectories obtained by executing a policy trained with imitation learning on this data.

The results on these tasks are shown in Figure 31. We first see that AWAC attains performance that is comparable to the best prior method from offline training alone, as indicated by the value at time step 0 (which corresponds to the beginning of online fine-tuning). This means that, during online interaction, AWAC collects data that is of higher quality, and improves more quickly. The prior methods struggle to improve from online training on these tasks, likely because the sparse reward function and challenging dynamics make progress very difficult from a bad initialization. These results suggest that AWAC is effectively able to leverage prior data to bootstrap online reinforcement learning in the real world, even on tasks with difficult and uninformative reward functions.

6.7 DISCUSSION AND FUTURE WORK

We have discussed the challenges existing RL methods face when fine-tuning from prior datasets, and proposed an algorithm, AWAC, for this setting. The key insight in AWAC is that an implicitly constrained actor-critic algorithm is able to both train offline and continue to improve with more experience. We provide detailed empirical analysis of the design decisions behind AWAC, showing the importance of off-policy learning, bootstrapping and the particular form of implicit constraint enforcement. To validate these ideas, we evaluate on a variety of simulated and real world robotic problems.

While AWAC is able to effectively leverage prior data for significantly accelerating learning, it does run into some limitations. Firstly, it can be challenging to choose the appropriate threshold for constrained optimization. Resolving this would involve exploring adaptive threshold tuning schemes. Secondly, while AWAC is able to avoid over-conservative behavior empirically, in future work, we hope to analyze theoretical factors that go into building a good finetuning algorithm. And lastly, in the future we plan on applying AWAC to more broadly incorporate data across different robots, labs and tasks rather than just on isolated setups. By doing so, we hope to enable an even wider array of robotic applications.

6.8 CONTRIBUTION STATEMENT

The work in this chapter was performed in collaboration with Abhishek Gupta, Murtaza Dalal, and Sergey Levine (A. Nair et al., 2020). A.N. and A.G. were joint first co-authors. A.N. proposed the project, managed the project, and implemented the algorithm. A.N. and M.D. conducted the simulation experiments. A.G. advised the project and conducted the real-world dextrous manipulation experiments. A.N. conducted the remaining real-world robot experiments. A.N. and A.G. co-wrote the paper. S.L. advised the project and assisted with writing.

7

OFFLINE REINFORCEMENT LEARNING WITH IMPLICIT Q-LEARNING

Offline reinforcement learning (RL) addresses the problem of learning effective policies entirely from previously collected data, without online interaction (Fujimoto et al., 2019b; Lange et al., 2012a). This is very appealing in a range of real-world domains, from robotics to logistics and operations research, where real-world exploration with untrained policies is costly or dangerous, but prior data is available. However, this also carries with it major challenges: improving the policy beyond the level of the behavior policy that collected the data requires estimating values for actions other than those that were seen in the dataset, and this, in turn, requires trading off policy improvement against distributional shift, since the values of actions that are too different from those in the data are unlikely to be estimated accurately. Prior methods generally address this by either constraining the policy to limit how far it deviates from the behavior policy (Fujimoto et al., 2019b; Yifan Wu et al., 2019; Fujimoto and S. S. Gu, 2021; Kumar et al., 2019b; A. Nair et al., 2020; Ziyu Wang et al., 2020a), or by regularizing the learned value functions to assign low values to out-of-distribution actions (Kumar et al., 2020b; Kostrikov et al., 2021a). Nevertheless, this imposes a trade-off between how much the policy improves and how vulnerable it is to misestimation due to distributional shift. Can we devise an offline RL method that avoids this issue by never needing to directly query or estimate values for actions that were not seen in the data?

In this work, we start from an observation that *in-distribution* constraints widely used in prior work might not be sufficient to avoid value function extrapolation, and we ask whether it is possible to learn an optimal policy with *in-sample learning*, without *ever* querying the values of any unseen actions. The key idea in our method is to approximate an upper expectile of the distribution over values with respect to the distribution of dataset actions for each state. We alternate between fitting this value function with expectile regression, and then using it to compute Bellman backups for training the Q-function.

We show that we can do this simply by modifying the loss function in a SARSA-style TD backup, without ever using out-of-sample actions in the target value. Once this Q-function has converged, we extract the corresponding policy using advantage-weighted behavioral cloning. This approach does not require explicit constraints or explicit regularization of out-of-distribution actions during value function training, though our policy extraction step does implicitly enforce a constraint, as discussed in prior work on advantage-weighted regression (Peters and Schaal, 2007b; Peng et al., 2019b; A. Nair et al., 2020; Ziyu Wang et al., 2020a).

Our main contribution is implicit Q-learning (IQL), a new offline RL algorithm that avoids ever querying values of unseen actions while still being able to perform multi-step dynamic programming updates. Our method is easy to implement by making a small change to the loss function in a simple SARSA-like TD update and is computationally very efficient. Furthermore, our approach demonstrates the state-of-the-art performance on D4RL, a popular benchmark for offline reinforcement learning. In particular, our approach significantly improves over the prior state-of-the-art on challenging Ant Maze tasks that require to "stitch" several sub-optimal trajectories. Finally, we demonstrate that our approach is suitable for finetuning; after initialization from offline RL, IQL is capable of improving policy performance utilizing additional interactions.

7.1 RELATED WORK

A significant portion of recently proposed offline RL methods are based on either constrained or regularized approximate dynamic programming (e.g., Q-learning or actor-critic methods), with the constraint or regularizer serving to limit deviation from the behavior policy. We will refer to these methods as "multi-step dynamic programming" algorithms, since they perform true dynamic programming for multiple iterations, and therefore can in principle recover the optimal policy if provided with high-coverage data. The constraints can be implemented via an explicit density model (Yifan Wu et al., 2019; Fujimoto et al., 2019b; Kumar et al., 2019b; Ghasemipour et al., 2021), implicit divergence constraints (A. Nair et al., 2020; Ziyu Wang et al., 2020a; Peters and Schaal, 2007b; Peng et al., 2019b; Noah Y Siegel et al., 2020a), or by adding a supervised learning term to the policy improvement objective (Fujimoto and S. S. Gu, 2021). Several works have also proposed to directly regularize the Q-function to produce low values for out-of-distribution actions (Kostrikov et al., 2021a; Kumar et al., 2020b; Fakoor et al., 2021). Our method is also a multi-step dynamic programming algorithm. However, in contrast to prior works, our method completely avoids directly querying the learned Q-function with unseen actions during training, removing the need for any constraint during this stage, though the subsequent policy extraction, which is based on advantage-weighted regression (Peng et al., 2019b; A. Nair et al., 2020), does apply an implicit constraint. However, this policy does not actually influence value function training.

In contrast to multi-step dynamic programming methods, several recent works have proposed methods that rely either on a single step of policy iteration, fitting the value function or Q-function of the behavior policy and then extracting the corresponding greedy policy (Peng et al., 2019b; Brandfonbrener et al., 2021; Gulcehre et al., 2021), or else avoid value functions completely and utilize behavioral cloning-style objectives (L. Chen et al., 2021). We collectively refer to these as "single-step" approaches. These methods avoid needing to query unseen actions as well, since they either use no value function at all, or learn the value function of the behavior policy. Although these methods are simple to implement and effective on the MuJoCo locomotion tasks in D4RL, we show that such single-step methods perform very poorly on more complex datasets in D4RL, which require combining parts of suboptimal trajectories ("stitching"). Prior multi-step dynamic programming methods perform much better in such settings, as does our method. We discuss this distinction in more detail in Section 7.3.1. Our method also shares the simplicity and computational efficiency of single-step approaches, providing an appealing combination of the strengths of both types of methods.

Our method is based on estimating the characteristics of a random variable. Several recent works involve approximating statistical quantities of the value function distribution. In particular, quantile regression (Koenker and Hallock, 2001) has been previously used in reinforcement learning to estimate the quantile function of a state-action value function (Dabney et al., 2018b; Dabney et al., 2018a; Kuznetsov et al., 2020). Although our method is related, in that we perform expectile regression, our aim is not to estimate the distribution of values that results from stochastic transitions, but rather estimate expectiles of the state value function with respect to random actions. This is a very different statistic: our aim is not to determine how the Q-value can vary with different future outcomes, but how the Q-value can vary with different actions *while averaging together future outcomes due to stochastic dynamics*. While prior work on distributional RL can also be used for offline RL, it would suffer from the same action extrapolation issues as other methods, and would require similar constraints or regularization, while our method does not.

7.2 Implicit Q-Learning

Like many recent offline RL methods, our work builds on approximate dynamic programming methods that minimize temporal difference error, according to the following loss:

$$L_{TD}(\theta) = \mathbb{E}_{(s,a,s')\sim\mathcal{D}}[(r(s,a) + \gamma \max_{a'} Q_{\hat{\theta}}(s',a') - Q_\theta(s,a))^2], \tag{36}$$

where $\mathcal{D}$ is the dataset, $Q_\theta(s,a)$ is a parameterized Q-function, $Q_{\hat{\theta}}(s,a)$ is a target network (e.g., with soft parameters updates defined via Polyak averaging), and the policy is defined as $\pi(s) = \arg\max_a Q_\theta(s,a)$. Most recent offline RL methods modify either the value function loss (above) to regularize the value function in a way that keeps the resulting policy close to the data, or constrain the arg max policy directly. This is important because out-of-distribution actions a' can produce erroneous values for $Q_{\hat{\theta}}(s',a')$ in the above objective, often leading to *overestimation* as the policy is defined to maximize the (estimated) Q-value.

In this work, we aim to entirely avoid querying *out-of-sample* (unseen) actions in our TD loss. Although the goal of this work is to approximate the optimal Q-function, we start by considering fitted Q evaluation with a SARSA-style objective which has been considered in prior work on Offline Reinforcement Learning (Brandfonbrener et al., 2021; Gulcehre et al., 2021). This objective aims to learn the value of the dataset policy π_β (also called the behavior policy):

$$L(\theta) = \mathbb{E}_{(s,a,s',a')\sim\mathcal{D}}[(r(s,a) + \gamma Q_{\hat{\theta}}(s',a') - Q_\theta(s,a))^2]. \tag{37}$$

This objective never queries values for out-of-sample actions, in contrast to Equation (36). One specific property of this objective that is important for this work is that it uses mean squared error (MSE) that fits $Q_\theta(s,a)$ to predict the mean statistics of the TD targets. Thus, if we assume unlimited capacity and no sampling error, the optimal parameters should satisfy

$$Q_{\theta^*}(s,a) \approx r(s,a) + \gamma \mathbb{E}_{\substack{s'\sim p(\cdot|s,a)\\ a'\sim\pi_\beta(\cdot|s)}}[Q_{\hat{\theta}}(s',a')]. \tag{38}$$

Prior work (Brandfonbrener et al., 2021; Gulcehre et al., 2021; Peng et al., 2019b) has proposed directly using this objective to learn Q^{π_β}, and then train the policy π_ψ to maximize Q^{π_β}. This avoids any issues with out-of-distribution actions, since the TD loss only uses dataset actions. However, while this procedure works well empirically on simple MuJoCo locomotion tasks in D4RL, we will show that it performs very poorly on more complex tasks that benefit from multi-step dynamic programming. In our method, which we derive next, we retain the benefits of using this SARSA-like objective, but modify it so that it allows us to perform multi-step dynamic programming and learn a near-optimal Q-function.

Our method will perform a Q-function update similar to Equation (37), but we will aim to estimate the maximum Q-value over actions that are in the support of the data distribution. Crucially, we will show that it is possible to do this *without ever querying the learned Q-function on out-of-sample actions* by utilizing expectile regression. Formally, the value function we aim to learn is given by:

$$L(\theta) = \mathbb{E}_{(s,a,s')\sim\mathcal{D}}[(r(s,a) + \gamma \max_{\substack{a' \in \mathcal{A} \\ \text{s.t. } \pi_\beta(a'|s')>0}} Q_{\hat{\theta}}(s',a') - Q_\theta(s,a))^2]. \tag{39}$$

Our algorithm, implicit Q-Learning (IQL), aims to estimate this objective while evaluating the Q-function only on the state-action pairs in the dataset. To this end, we propose to fit $Q_\theta(s,a)$ to estimate state-conditional expectiles of the target values, and show that specific expectiles approximate the maximization defined above. In Section 7.2.4 we show that this approach performs multi-step dynamic programming in theory, and in Section 7.3.1 we show that it does so in practice.

7.2.1 *Expectile Regression*

Practical methods for estimating various statistics of a random variable have been thoroughly studies in applied statistics and econometrics. The $\tau \in (0,1)$ expectile of some random variable X is defined as a solution to the asymmetric least squares problem:

$$\arg\min_{m_\tau} \mathbb{E}_{x\sim X}[L_2^\tau(x - m_\tau)], \text{ where } L_2^\tau(u) = |\tau - 1(u<0)|u^2.$$

That is, for $\tau > 0.5$, this asymmetric loss function downweights the contributions of x values smaller than m_τ while giving more weights to larger values (see Figure 33, left). Expectile regression is closely related to quantile regression (Koenker and Hallock, 2001), which is a popular technique for estimating quantiles of a distribution widely used in reinforcement learning (Dabney et al., 2018b; Dabney et al., 2018a) [1]. The quantile

[1] Our method could also be derived with quantiles, but since we are not interested in learning all of the expectiles/quantiles, unlike prior work (Dabney et al., 2018b; Dabney et al., 2018a), it is more convenient to estimate a single expectile because this involves a simple modification to the MSE loss that is already used in standard RL methods. We found it to work somewhat better than quantile regression with its corresponding ℓ_1 loss.

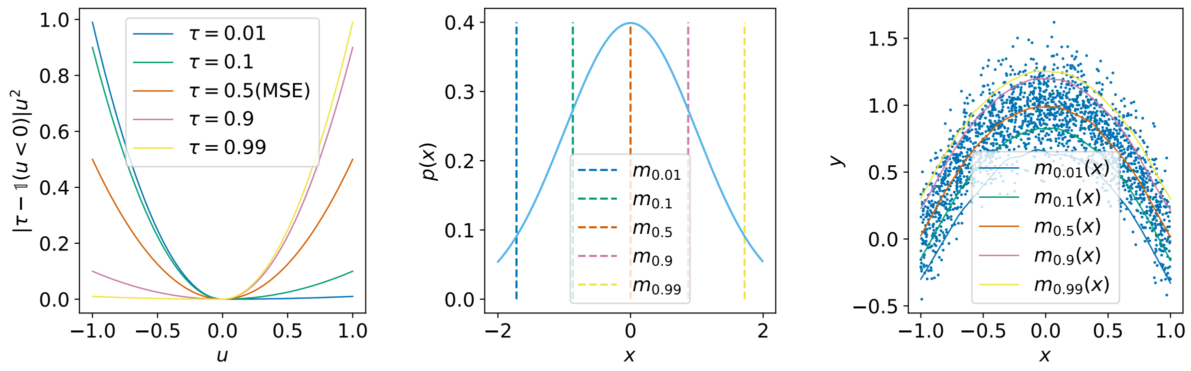


Figure 33: **Left:** The asymmetric squared loss used for expectile regression. $\tau = 0.5$ corresponds to the standard mean squared error loss, while $\tau = 0.9$ gives more weight to positives differences. **Center:** Expectiles of a normal distribution. **Right:** an example of estimating state conditional expectiles of a two-dimensional random variable. Each x corresponds to a distribution over y. We can approximate a maximum of this random variable with expectile regression: $\tau = 0.5$ correspond to the conditional mean statistics of the distribution, while $\tau \approx 1$ approximates the maximum operator over in-support values of y.

regression loss is defined as an asymmetric ℓ_1 loss.

We can also use this formulation to predict expectiles of a conditional distribution:

$$\arg\min_{m_\tau(x)} \mathbb{E}_{(x,y)\sim\mathcal{D}}[L_2^\tau(y - m_\tau(x))].$$

Figure 33 (right) illustrates conditional expectile regression on a simple two-dimensional distribution. Note that we can optimize this objective with stochastic gradient descent. It provides unbiased gradients and is easy to implement with standard machine learning libraries.

7.2.2 *Learning the Value Function with Expectile Regression*

Expectile regression provides us with a powerful framework to estimate statistics of a random variable beyond mean regression. We can use expectile regression to modify the policy evaluation objective in Equation (37) to predict an upper expectile of the TD targets that approximates the maximum of $r(s,a) + \gamma Q_{\hat{\theta}}(s', a')$ over actions a' constrained to the dataset actions, as in Equation (39). This leads to the following expectile regression objective:

$$L(\theta) = \mathbb{E}_{(s,a,s',a')\sim\mathcal{D}}[L_2^\tau(r(s,a) + \gamma Q_{\hat{\theta}}(s',a') - Q_\theta(s,a))].$$

However, this formulation has a significant drawback. Instead of estimating expectiles just with respect to the actions in the support of the data, it also incorporates stochasticity that comes from the environment dynamics $s' \sim p(\cdot|s,a)$. Therefore, a large target value might not necessarily reflect the existence of a single action that achieves that value, but rather a "lucky" sample that happened to have transitioned into a good state. We resolve this by introducing a separate value function that approximates an expectile only with respect to the action distribution, leading to the following loss:

$$L_V(\psi) = \mathbb{E}_{(s,a)\ \sim\mathcal{D}}[L_2^\tau(Q_{\hat{\theta}}(s,a) - V_\psi(s))]. \tag{40}$$

We can then use this estimate to update the Q-functions with the MSE loss, which averages over the stochasticity from the transitions and avoids the "lucky" sample issue mentioned above:

$$L_Q(\theta) = \mathbb{E}_{(s,a,s')\ \sim\mathcal{D}}[(r(s,a) + \gamma V_\psi(s') - Q_\theta(s,a))^2]. \tag{41}$$

Note that these losses do not use any explicit policy, and only utilize actions from the dataset for both objectives, similarly to SARSA-style policy evaluation. In Section 7.2.4, we will show that this procedure recovers the optimal Q-function under some assumptions. Also, even though only one action is available for every state in the dataset for continuous action spaces, due to neural network generalization, the expectile regression does not result in SARSA-style policy evaluation as shown in Section 7.3.2.

7.2.3 *Policy Extraction and Algorithm Summary*

While our modified TD learning procedure learns an approximation to the optimal Q-function, it does not explicitly represent the corresponding policy, and therefore requires a separate policy extraction step. While one can consider any technique for policy extraction that constrains the learned policy to stay close to the dataset actions, we aim for a simple method for policy extraction. As before, we aim to avoid using out-of-samples actions. Therefore, we extract the policy with advantage-weighted regression (Peters and Schaal, 2007b; Peng et al., 2019b) previously successfully used for policy extraction in

Offline RL (Q. Wang et al., 2018b; A. Nair et al., 2020; Brandfonbrener et al., 2021):

$$L_\pi(\phi) = \mathbb{E}_{(s,a)\sim\mathcal{D}}[\exp(\beta(Q_{\hat{\theta}}(s,a) - V_\psi(s))) \log \pi_\phi(a|s)], \tag{42}$$

where $\beta \in [0, \infty)$ is an inverse temperature. Note that this objective does not clone all actions from the dataset but, as shown in prior work, this objective learns a policy that maximizes the Q-values subject to a distribution constraint (Peters and Schaal, 2007b; Peng et al., 2019b; A. Nair et al., 2020). This step can be seen as selecting and cloning the most optimal actions in the dataset.

Our final algorithm consists of two stages. First, we fit the value function and Q, performing a number of gradient updates alternating between Eqn. (40) and (41). Second, we perform stochastic gradient descent on Equation (42). For both steps, we use a version of clipped double Q-learning (Fujimoto et al., 2018a), taking a minimum of two Q-functions for V-function and policy updates. We summarize our final method in Algorithm 5. Note that the policy does not influence the value function in any way, and therefore extraction could be performed either concurrently or after TD learning. Concurrent learning provides a way to use IQL with online finetuning, as we discuss in Section 7.3.3.

Algorithm 5 Implicit Q-learning

```
0: Initialize parameters $\psi$, $\theta$, $\hat{\theta}$, $\phi$.
0: TD learning (IQL):
0: for each gradient step do
0:     $\psi \leftarrow \psi - \lambda_V \nabla_\psi L_V(\psi)$
0:     $\theta \leftarrow \theta - \lambda_Q \nabla_\theta L_Q(\theta)$
0:     $\hat{\theta} \leftarrow (1-\alpha)\hat{\theta} + \alpha\theta$
0: Policy extraction (AWR):
0: for each gradient step do
0:     $\phi \leftarrow \phi - \lambda_\pi \nabla_\phi L_\pi(\phi)$
=0
```

7.2.4 *Analysis*

In this section, we will show that IQL can recover the optimal value function under the dataset support constraints. First, we prove a simple lemma that we will then use to show how our approach can enable learning the optimal value function.

Lemma 3. *Let X be a real-valued random variable with a bounded support and supremum of the support is x^*. Then,*

$$\lim_{\tau \to 1} m_\tau = x^*$$

One can show that expectiles of a random variable have the same supremum x^*. Moreover, for all τ_1 and τ_2 such that $\tau_1 < \tau_2$, we get $m_{\tau_1} \leqslant m_{\tau_2}$. Therefore, the limit follows from the properties of bounded monotonically non-decreasing functions.

In the following theorems, we show that under certain assumptions, our method indeed approximates the optimal state-action value Q^* and performs multi-step dynamical programming. We first prove a technical lemma relating different expectiles of the Q-function, and then derive our main result regarding the optimality of our method.

For the sake of simplicity, we introduce the following notation for our analysis. Let $\mathbb{E}^{\tau}_{x \sim X}[x]$ be a τ^{th} expectile of X (e.g., $\mathbb{E}^{0.5}$ corresponds to the standard expectation). Then, we define $V_\tau(s)$ and $Q_\tau(s, a)$, which correspond to optimal solutions of Eqn. 40 and 41 correspondingly, recursively as:

$$\begin{aligned} V_\tau(s) &= \mathbb{E}^{\tau}_{a \sim \pi_\beta(\cdot|s)}[Q_\tau(s, a)], \\ Q_\tau(s, a) &= r(s, a) + \gamma \mathbb{E}_{s' \sim p(\cdot|s,a)}[V_\tau(s')]. \end{aligned}$$

Lemma 4. *For all s, τ_1 and τ_2 such that $\tau_1 < \tau_2$ we get $V_{\tau_1}(s) \leqslant V_{\tau_2}(s)$.*

Proof. The proof follows the policy improvement proof (R. S. Sutton and Barto, 2018). See Appendix E.1. □

Corollary 1. *For any τ and s we have $V_\tau(s) \leqslant \max_{\substack{a \in \mathcal{A} \\ \text{s.t. } \pi_\beta(a|s) > 0}} Q^*(s, a)$ where $V_\tau(s)$ is defined as above and $Q^*(s, a)$ is an optimal state-action value function constrained to the dataset and defined as*

$$Q^*(s, a) = r(s, a) + \gamma \mathbb{E}_{s' \sim p(\cdot|s,a)} \left[\max_{\substack{a' \in \mathcal{A} \\ \text{s.t. } \pi_\beta(a'|s') > 0}} Q^*(s', a') \right].$$

Proof. The proof follows from the observation that convex combination is smaller than maximum. □

Theorem 1.

$$\lim_{\tau \to 1} V_\tau(s) = \max_{\substack{a \in \mathcal{A} \\ \text{s.t. } \pi_\beta(a|s) > 0}} Q^*(s, a).$$

Proof. Follows from combining Lemma 3 and Corollary 1. □

Therefore, for a larger value of $\tau < 1$, we get a better approximation of the maximum. On the other hand, it also becomes a more challenging optimization problem. Thus, we treat τ as a hyperparameter. Due to the property discussed in Theorem 1 we dub our method implicit Q-learning (IQL). We also emphasize that our value learning method defines the entire spectrum of methods between SARSA ($\tau = 0.5$) and Q-Learning ($\tau \to 1$). Note that in contrast to other multi-step methods, IQL absorbs the policy improvement step into value learning. Therefore, fitting Q-function corresponds to the policy evaluation step, while fitting the value function with IQL corresponds to *implicit* policy improvement.

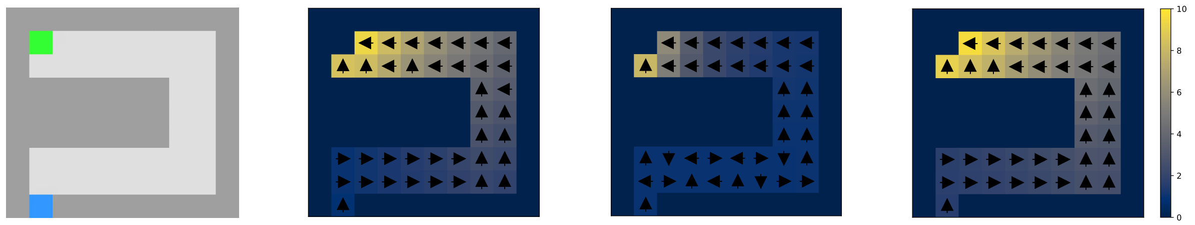
7.3 EXPERIMENTAL EVALUATION

Our experiments aim to evaluate our method comparatively, in contrast to prior offline RL methods, and in particular to understand how our approach compares both to single-step methods and multi-step dynamic programming approaches. We will first demonstrate the benefits of multi-step dynamic programming methods, such as ours, in contrast to single-step methods, showing that on some problems this difference can be extremely large. We will then compare IQL with state-of-the-art single-step and multi-step algorithms on the D4RL (Fu et al., 2020) benchmark tasks, studying the degree to which we can learn effective policies using only the actions in the dataset. We examine domains that contain near-optimal trajectories, where single-step methods perform well, as well as domains with no optimal trajectories at all, which require multi-step dynamic programming. Finally, we will study how IQL compares to prior methods when finetuning with online RL starting from an offline RL initialization.

7.3.1 *The Difference Between One-Step Policy Improvement and IQL*

We first use a simple maze environment to illustrate the importance of multi-step dynamic programming for offline RL. The maze has a u-shape, a single start state, and a single goal state (see Figure 34a). The agent receives a reward of 10 for entering the goal state and zero reward for all other transitions. With a probability of 0.25, the agent transitions to a random state, and otherwise to the commanded state. The dataset consists of 1 optimal trajectory and 99 trajectories with uniform random actions. Due to a short horizon of the problem, we use $\gamma = 0.9$.

Figure 34 (c, d) illustrates the difference between single-step methods which fit $Q^{\pi}(s, a)$ via SARSA-style objective, in this case represented by Onepstep RL (Brandfonbrener et al., 2021; Q. Wang et al., 2018b; Gulcehre et al., 2021) and IQL with $\tau = 0.95$. Note that these methods represent a special case of our method with $\tau = 0.5$. Although



(a) toy maze MDP (b) true optimal $V^{\star}$ (c) One-step Policy Eval. (d) IQL

Figure 34: Evaluation of our algorithm on a toy umaze environment (a). When the static dataset is heavily corrupted by suboptimal actions, one-step policy evaluation results in a value function that degrades to zero far from the rewarding states too quickly (c). Our algorithm aims to learn a near-optimal value function, combining the best properties of SARSA-style evaluation with the ability to perform multi-step dynamic programming, leading to value functions that are much closer to optimality (shown in (b)) and producing a much better policy (d).

states closer to the high reward state will still have higher values, these values decay much faster as we move further away than they would for the optimal value function, and the resulting policy is highly suboptimal. Since IQL (d) performs iterative dynamic programming, it correctly propagates the signal, and the values are no longer dominated by noise. The resulting value function closely matches the true optimal value function (b).

7.3.2 *Comparisons on Offline RL Benchmarks*

Next, we evaluate our approach on the D4RL benchmark in comparison to prior methods (see Table 1). The MuJoCo tasks in D4RL consist of the Gym locomotion tasks, the Ant Maze tasks, and the Adroit and Kitchen robotic manipulation environments. Some prior works, particularly those proposing one-step methods, focus entirely on the Gym locomotion tasks. However, these tasks include a significant fraction of near-optimal trajectories in the dataset. In contrast, the Ant Maze tasks, especially the medium and large ones, contain very few or no near-optimal trajectories, making them very challenging for one-step methods. These domains require "stitching" parts of suboptimal trajectories that travel between different states to find a path from the start to the goal of the maze (Fu et al., 2020). As we will show, multi-step dynamic programming is essential in these domains. The Adroit and Kitchen tasks are comparatively less discriminating, and we found that most RL methods perform similarly to imitation learning in these domains (Florence et al., 2021). We therefore focus our analysis on the Gym locomotion and Ant Maze domains, but include full Adroit and Kitchen results in Appendix E.2 for completeness.

COMPARISONS AND BASELINES. We compare to methods that are representative of both multi-step dynamic programming and one-step approaches. In the former category, we compare to CQL (Kumar et al., 2020b), TD3+BC (Fujimoto and S. S. Gu, 2021), and AWAC (A. Nair et al., 2020). In the latter category, we compare to Onestep RL (Brandfonbrener et al., 2021) and Decision Transformers (L. Chen et al., 2021). We obtained the Decision Transformers results on Ant Maze subsets of D4RL tasks using the author-provided implementation[2] and following authors instructions communicated over email. We obtained results for TD3+BC and Onestep RL (Exp. Weight) directly from the authors. Note that L. Chen et al. (2021) and Brandfonbrener et al. (2021) incorrectly report results for some prior methods, such as CQL, using the "-v0" environments. These generally produce lower scores than the "-v2" environments that these papers use for their own methods. We use the "-v2" environments for all methods to ensure a fair comparison, resulting in higher values for CQL. Because of this fix, our reported CQL scores are higher than all other prior methods. We obtained results for "-v2" datasets using an author-suggested implementation.[3] On the Gym locomotion tasks (halfcheetah, hopper, walker2d), we find that IQL performs comparably to the best performing prior method, CQL. On the more challenging Ant Maze task, IQL outperforms CQL, and outperforms the one-step methods by a very large margin.

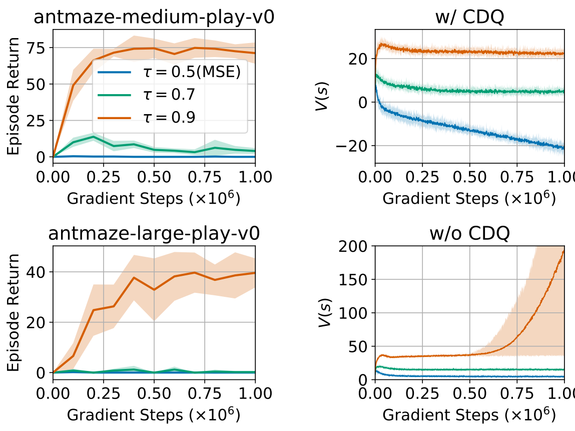


Figure 35: **Left**: Estimating a larger expectile τ is crucial for antmaze tasks that require dynamical programming ('stitching'). **Right:** Clipped double Q-Learning (CDQ) is crucial for learning values for $\tau = 0.9$.

RUNTIME. Our approach is also computationally faster than the baselines (see Table 1). For the baselines, we measure runtime for our reimplementations of the methods in JAX (Bradbury et al., 2018) built on top of JAXRL (Kostrikov, 2021), which are typically faster than the original implementations. For example, the original implementation of CQL takes more than 4 hours to perform 1M updates, while ours takes only 80 minutes. Even so, IQL still requires about 4x less time than our reimplementation of CQL

2 https://github.com/kzl/decision-transformer
3 https://github.com/young-geng/CQL

on average, and is comparable to the fastest prior one-step methods. We did not reimplement Decision Transformers due to their complexity and report runtime of the original implementation.

Effect of τ hyperparameter. We also demonstrate that it is crucial to compute a larger expectile on tasks that require "stitching" (see Figure 35). We provide complete results in Appendix E.2. With larger values of τ, our method approximates Q-learning better, leading to better performance on the Ant Maze tasks. Moreover, due to neural network generalization, values learned with expectile regression increase with a larger τ and do not degrade to behavior policy values ($\tau = 0.5$). Finally, clipped double Q-Learning is crucial for estimating values for a larger $\tau = 0.9$.

7.3.3 *Online Fine-tuning after Offline RL*

The policies obtained by offline RL can often be improved with a small amount of online interaction. IQL is well-suited for online fine-tuning for two reasons. First, IQL has strong offline performance, as shown in the previous section, which provides a good initialization. Second, IQL implements a weighted behavioral cloning policy extraction step, which has previously been shown to allow for better online policy improvement compared to other types of offline constraints (A. Nair et al., 2020). To evaluate the finetuning capability of various RL algorithms, we first run offline RL on each dataset, then run 1M steps of online RL, and then report the final performance. We compare to AWAC (A. Nair et al., 2020), which has been proposed specifically for online finetuning, and CQL (Kumar et al., 2020b), which showed the best performance among prior methods in our experiments in the previous section. Exact experimental details are provided in Appendix E.3. We use the challenging Ant Maze D4RL domains (Fu et al., 2020), as well as the high-dimensional dexterous manipulation environments from Rajeswaran et al. (2018), which A. Nair et al. (2020) propose to use to study online adaptation with AWAC. Results are shown in Table 21. On the Ant Maze domains, IQL significantly outperforms both prior methods after online finetuning. CQL attains the second best score, while AWAC performs comparatively worse due to much weaker offline initialization. On the dexterous hand tasks, IQL performs significantly better than AWAC on relocate-binary-v0, comparably on door-binary-v0, and slightly worse on pen-binary-v0, with the best overall score.

7.4 CONCLUSION

We presented implicit Q-Learning (IQL), a general algorithm for offline RL that completely avoids any queries to values of out-of-sample actions during training while still enabling multi-step dynamic programming. To our knowledge, this is the first method that combines both of these features. This has a number of important benefits. First, our algorithm is computationally efficient: we can perform 1M updates on one GTX1080 GPU in less than 20 minutes. Second, it is simple to implement, requiring only minor modifications over a standard SARSA-like TD algorithm, and performing policy extraction with a simple weighted behavioral cloning procedure resembling supervised learning. Finally, despite the simplicity and efficiency of this method, we show that it attains excellent performance across all of the tasks in the D4RL benchmark, matching the best prior methods on the MuJoCo locomotion tasks, and exceeding the state-of-the-art performance on the challenging ant maze environments, where multi-step dynamic programming is essential for good performance.

7.5 CONTRIBUTION STATEMENT

The work in this chapter was performed in collaboration with Ilya Kostrikov and Sergey Levine (Kostrikov et al., 2021b). I.K. led the project and was the first author. I.K. proposed to use expectile regression for offline RL, implemented the offline RL algorithm, conducted the offline RL experiments, and wrote the majority of the paper. A.N. assisted in developing the method, conducted the online finetuning experiments in 7.3.3, and released a re-implementation of the algorithm in PyTorch (the original implementation released by I.K. used Jax). S.L. advised the project and assisted with writing.

Table 1: Averaged normalized scores on MuJoCo locomotion and Ant Maze tasks. Our method outperforms prior methods on the challenging Ant Maze tasks, which require dynamic programming, and is competitive with the best prior methods on the locomotion tasks.

Dataset	BC	10%BC	BCQ	DT	ABM	AWAC	Onestep RL	TD3+BC	CQL	IQL (Ours)
halfcheetah-m-v2	42.6	42.5	**47.0**	42.6±0.1	53.6	43.5	**48.4±0.1**	**48.3±0.3**	44.0±5.4	**47.4±0.2**
hopper-m-v2	52.9	56.9	56.7	**67.6±1.0**	0.7	57.0	59.6±2.5	59.3±4.2	58.5±2.1	**66.2±5.7**
walker2d-m-v2	75.3	75.0	72.6	74.0±1.4	0.5	72.4	**81.8±2.2**	83.7±2.1	72.5±0.8	78.3± 8.7
halfcheetah-m-r-v2	36.6	40.6	40.4	36.6±0.8	**50.5**	40.5	38.1±1.3	**44.6±0.5**	**45.5±0.5**	**44.2±1.2**
hopper-m-r-v2	18.1	75.9	53.3	82.7±7.0	49.6	37.2	**97.5±0.7**	60.9±18.8	**95.0±6.4**	**94.7±8.6**
walker2d-m-r-v2	26.0	62.5	52.1	66.6±3.0	53.8	27.0	49.5±12.0	**81.8±5.5**	77.2±5.5	73.8±7.1
halfcheetah-m-e-v2	55.2	**92.9**	**89.1**	86.8±1.3	18.5	42.8	**93.4±1.6**	**90.7±4.3**	**91.6±2.8**	86.7±5.3
hopper-m-e-v2	52.5	**110.9**	81.8	**107.6±1.8**	0.7	55.8	103.3±1.9	98.0±9.4	**105.4±6.8**	91.5±14.3
walker2d-m-e-v2	**107.5**	**109.0**	**109.5**	**108.1±0.2**	3.5	74.5	**113.0±0.4**	**110.1±0.5**	**108.8±0.7**	**109.6±1.0**
locomotion-v2 total	466.7	**666.2**	602.5	**672.6±16.6**	231.4	450.7	**684.6±22.7**	**677.4±44.5**	**698.5±31.0**	**692.4±52.1**
antmaze-u-v0	54.6	62.8	**89.8**	59.2	59.9	56.7	64.3	78.6	74.0	**87.5 ± 2.6**
antmaze-u-d-v0	45.6	50.2	**83.0**	53.0	48.7	49.3	60.7	71.4	**84.0**	62.2 ± 13.8
antmaze-m-p-v0	0.0	5.4	15.0	0.0	0.0	0.0	0.3	10.6	61.2	**71.2 ± 7.3**
antmaze-m-d-v0	0.0	9.8	0.0	0.0	0.5	0.7	0.0	3.0	53.7	**70.0 ± 10.9**
antmaze-l-p-v0	0.0	0.0	0.0	0.0	0.	0.0	0.0	0.2	15.8	**39.6±5.8**
antmaze-l-d-v0	0.0	6.0	0.0	0.0	0.0	1.0	0.0	0.0	14.9	**47.5±9.5**
antmaze-v0 total	100.2	134.2	187.8	112.2	109.1	107.7	125.3	163.8	303.6	**378.0±49.9**
total	566.9	800.4	790.3	784.8	340.5	558.4	809.9	841.2	1002.1	**1070.4±102.0**
kitchen-v0 total	**154.5**	-	-	-	-	-	-	-	144.6	**159.8±22.6**
adroit-v0 total	104.5	-	-	-	-	-	-	-	93.6	**118.1±30.7**
total+kitchen+adroit	825.9	-	-	-	-	-	-	-	1240.3	**1348.3±155.3**
runtime	10m	10m		960m		20m	20m*	20m	80m	20m

*: Note that it is challenging to compare one-step and multi-step methods directly. Also, Brandfonbrener et al., 2021 reports results for a set of hyperparameters, such as batch and network size, that is significantly different from other methods. We report results for the original hyperparameters and runtime for a comparable set of hyperparameters.

Dataset	AWAC	CQL	IQL (Ours)
antmaze-umaze-v0	56.7 → 59.0	70.1 → **99.4**	**88.0** → **96.3**
antmaze-umaze-diverse-v0	49.3 → 49.0	31.1 → **99.4**	**67.0** → 49.0
antmaze-medium-play-v0	0.0 → 0.0	23.0 → 0.0	**69.0** → **89.2**
antmaze-medium-diverse-v0	0.7 → 0.3	23.0 → 32.3	**71.8** → **91.4**
antmaze-large-play-v0	0.0 → 0.0	1.0 → 0.0	**36.8** → **51.8**
antmaze-large-diverse-v0	1.0 → 0.0	1.0 → 0.0	**42.2** → **59.8**
antmaze-v0 total	107.7 → 108.3	151.5 → 231.1	**374.8** → **437.5**
pen-binary-v0	**44.6** → **70.3**	31.2 → 9.9	37.4 → 60.7
door-binary-v0	**1.3** → **30.1**	0.2 → 0.0	0.7 → **32.3**
relocate-binary-v0	**0.8** → 2.7	0.1 → 0.0	0.0 → **31.0**
hand-v0 total	**46.7** → 103.1	31.5 → 9.9	38.1 → **124.0**
total	154.4 → 211.4	182.8 → 241.0	**412.9** → **561.5**

Table 2: Online finetuning results showing the initial performance after offline RL, and performance after 1M steps of online RL. In all tasks, IQL is able to finetune to a significantly higher performance than the offline initialization, with final performance that is comparable to or better than the best of either AWAC or CQL on all tasks except pen-binary-v0.

8

RESIDUAL REINFORCEMENT LEARNING FOR ROBOT CONTROL

8.1 introduction

While prior knowledge arriving in the form of prior data is very general, in realistic applications we may have access to more structured types of prior knowledge. In the following chapters, we investigate industrial robotics as an application area. Robots in today's manufacturing environments typically perform repetitive tasks, and often lack the ability to handle variability and uncertainty. Commonly used control algorithms, such as PID regulators and the computed torque method, usually follow predefined trajectories with little adaptive behavior. Many manufacturing tasks require some degree of adaptability or feedback to the environment, but significant engineering effort and expertise is required to design feedback control algorithms for these industrial robots. The engineering time for fine tuning such a controller might be similar in cost to the robot hardware itself. Being able to quickly and easily design feedback controllers for industrial robots would significantly broaden the space of manufacturing tasks that can be automated by robots.

Why is designing a feedback controller for many tasks hard with classical methods? While conventional feedback control methods can solve tasks such as path following efficiently, applications that involve contacts between the robot and its environment are difficult to approach with conventional control methods. Identifying and characterizing contacts and friction is difficult—even if a physical model provides reasonable contact behavior, identifying the physical parameters of a contact interaction accurately is very hard. Hence, it is often difficult to achieve adaptable yet robust control behavior, and significant control tuning effort is required as soon as these elements are introduced. Another drawback of conventional control methods is their lack of behavior generalization. Thus, all possible system behaviors must be considered a priori at design time.

Reinforcement learning (RL) methods hold the promise of solving these challenges

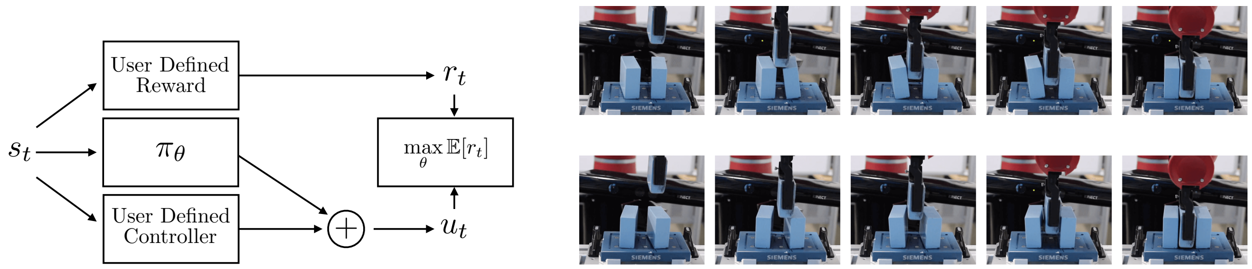


Figure 36: We train an agent directly in the real world to solve a model assembly task involving contacts and unstable objects. An outline of our method, which consists of combining hand-engineered controllers with a residual RL controller, is shown on the left. Roll-outs of residual RL solving the block insertion task are shown on the right. Residual RL is capable of learning a feedback controller that adapts to variations in the orientations of the standing blocks and successfully completes the task of inserting a block between them. Videos are available at `residualrl.github.io`

.

because they allow agents to learn behaviors through interaction with their surrounding environments and ideally generalize to new scenarios that differ from the specifications at the control design stage. Moreover, RL can handle control problems that are difficult to approach with conventional controllers because the control goal can be specified indirectly as a term in a reward function and not explicitly as the result of a control action. All of these aspects are considered enablers for truly autonomous manufacturing systems and important for fully flexible lot-size one manufacturing (Kannengiesser et al., 2017). However, standard RL methods require the robot learn through interaction, which can be unsafe initially, and collecting the amount of interaction that is needed to learn a complex skill from scratch can be time consuming.

In this chapter, we study control problems that are difficult to approach with conventional feedback control methods. However, the problems possess structure that can be partially handled with conventional feedback control, e.g. with impedance control. The residual part of the control task, which is the part that must consider contacts and external object dynamics, is solved with RL. The outputs of the conventional controller and RL are superposed to form the commanded control. The main contribution of this paper is a methodology that combines conventional feedback control with deep RL methods and is illustrated in Fig. 47. Our main motivation is a control approach that is suitable for real-world control problems in manufacturing, where the exploratory behavior of RL is a safety concern and the data requirements of deep RL can be expensive. We provide a thorough evaluation of our method on a block assembly task in simulation and on physical hardware. When the initial orientation of the blocks is noisy, our hand-designed controller fails to solve the task, while residual RL successfully learns to perform the task in under 3 hours. This suggests that we could usefully apply our method to practical manufacturing problems.

8.2 PRELIMINARIES

In this section, we set up our problem and summarize the foundations of classical control and reinforcement learning that we build on in our approach.

8.2.1 *Problem Statement - System Theoretic Interpretation*

The class of control problems that we are dealing with in this paper can be viewed from a dynamical systems point of view as follows. Consider a dynamical system that consists of a fully actuated robot and underactuated objects in the robot's environment. The robot and the objects in its environment are described by their states s_m and s_o, respectively. The robot can be controlled through the control input u while the objects cannot be directly controlled. However, the robot's states are coupled with the objects' states so that indirect control of s_o is possible through u. This is for example the case if the agent has large inertia and is interacting with small parts as is common in manufacturing. The states of agent and objects can either be fully observable or they can be estimated from measurements.

The time-discrete equations of motion of the overall dynamical system comprise the robot and objects and can be stated as

$$s_{t+1} = \begin{bmatrix} s_{m,t+1} \\ s_{o,t+1} \end{bmatrix} = \begin{bmatrix} A(s_{m,t}) & 0 \\ B(s_{m,t}, s_{o,t}) & C(s_{o,t}) \end{bmatrix} \begin{bmatrix} s_{m,t} \\ s_{o,t} \end{bmatrix} + D \begin{bmatrix} u_t \\ 0 \end{bmatrix}, \tag{43}$$

where the states can also be subject to algebraic constraints, which we do no state explicitly here.

The type of control objectives that we are interested in can be summarized as controlling the agent in order to manipulate the objects while also fulfilling a geometric objective such as trajectory following. It is difficult to solve the control problem directly with conventional feedback control approaches, which compute the difference between a desired and a measured state variable. In order to achieve best system performance feedback control methods require well understood and modeled state transition dynamics.

Finding the optimal control parameters can be difficult or even impossible if the system dynamics are not fully known.

In equation 43 the state transition matrices although $A(s_m)$ and $C(s_o)$ are usually known to a certain extent, because they represent rigid body dynamics, the coupling matrix $B(s_m, s_o)$ is usually not known. Physical interactions such as contacts and friction forces are the dominant effects that $B(s_m, s_o)$ needs to capture, which also applies to algebraic constraints, which are functions of s_m and s_o as well. Hence, conventional feedback control synthesis for determining u to control s_o is very difficult, and requires trial and error in practice. Another difficulty for directly designing feedback controllers is due to the fact that, for many control objectives, the states s_o need to fulfill conditions that cannot be expressed as deviations (errors) from desired states. This is often the case when we only know the final goal rather than a full trajectory.

Instead of directly designing a feedback control system, we can instead specify the goal via a reward function. These reward functions can depend on both s_m and s_o, where the terms that depend on s_m are position related objectives. Reinforcement learning can be used to maximize the reward function in a model-free way. In reinforcement learning, we simply attempt to maximize expected return. Unlike the previous section, RL does not attempt to model the unknown coupled dynamics of the agent and the object. Instead, it finds actions that maximizes rewards, without making any assumptions about the system dynamics. The final objective is to learn a policy $u_t = \pi(s_t)$ to maximize expected returns $R_t = \sum_{i=t}^{T} \gamma^{(i-t)} r_i$.

8.3 METHOD

Based on the analysis in Sec. 11.3, we introduce a control system that consists of two parts. The first part is based on conventional feedback control theory and maximize all reward terms that are functions of s_m. An RL method is superposed and maximizes the reward terms that are functions of s_o.

8.3.1 *Residual Reinforcement Learning*

In most robotics tasks, we consider rewards of the form:

$$r_t = f(s_m) + g(s_o). \tag{44}$$

The term $f(s_m)$ is assumed to be a function, which represents a geometric relationship of robot states, such as a Euclidean distance or a desired trajectory. The second term of the sum $g(s_o)$ can be a general class of functions. Concretely, in our model assembly task, $f(s_m)$ is the reward for moving the robot gripper between the standing blocks, while $g(s_o)$ is the reward for keeping the standing blocks upright and in their original positions.

The key insight of residual RL is that in many tasks, $f(s_m)$ can be easily optimized a priori of any environment interaction by conventional controllers, while $g(s_o)$ may be easier to learn with RL which can learn fine-grained hand-engineered feedback controllers even with friction and contacts. To take advantage of the efficiency of conventional controllers but also the flexbility of RL, we choose:

$$u = \pi_H(s_m) + \pi_\theta(s_m, s_o) \tag{45}$$

as the control action, where $\pi_H(s_m)$ is the human-designed controller and $\pi_\theta(s_m, s_o)$ is a learned policy parametrized by θ and optimized by an RL algorithm to maximize expected returns on the task.

Inserting equation 49 into equation 43 one can see that a properly designed feedback control law for $\pi_H(s_m)$ is able to provide exponentially stable error dynamics of s_m if the learned controller π_θ is neglected and the sub statespace is stabilizable. This is equivalent to maximizing equation 44 for the case f represents errors between actual and desired states.

The residual controller $\pi_\theta(s_m, s_o)$ can now be used to maximize the reward term $g(s_o)$ in equation 44. Since the control sequence equation 49 enters equation 43 through the dynamics of s_m and s_m is in fact the control input to the dynamics of s_o, we cannot simply use the a-priori hand-engineered feedback controller to achieve zero error of s_m and independently achieve the control objective on s_o. Through the coupling of states we need to perform an overall optimization of equation 49, whereby the hand-engineered feedback controller provides internal structures and eases the optimization related to the reward term $f(s_m)$.

8.3.2 *Method Summary*

Our method is summarized in Algorithm 6. The key idea is to combine the flexibility of RL with the efficiency of conventional controllers by additively combining a learnable parametrized policy with a fixed hand-engineered controller.

Algorithm 6 Residual reinforcement learning

```
Require: policy $\pi_\theta$, hand-engineered controller $\pi_H$.
 1: for $n = 0, ..., N-1$ episodes do
 2:     Initialize random process $\mathcal{N}$ for exploration
 3:     Sample initial state $s_0 \sim E$.
 4:     for $t = 0, ..., H-1$ steps do
 5:         Get policy action $u_t = \pi_\theta(s_t) + \mathcal{N}_t$.
 6:         Get action to execute $u'_t = u_t + \pi_H(s_t)$.
 7:         Get next state $s_{t+1} \sim p(\cdot \mid s_t, u'_t)$.
 8:         Store $(s_t, u_t, s_{t+1})$ into replay buffer $\mathcal{R}$.
 9:         Sample set of transitions $(s, u, s') \sim \mathcal{R}$.
10:         Optimize $\theta$ using RL with sampled transitions.
11:     end for
12: end for=0
```

As our underlying RL algorithm, we use a variant of twin delayed deep deterministic policy gradients (TD3) as described in Fujimoto et al., 2018b. TD3 is a value-based RL algorithm for continuous control based off of the deep deterministic policy gradient (DDPG) algorithm Lillicrap et al., 2016. We have found that TD3 is stable, sample-efficient, and requires little manual tuning compared to DDPG. We used the publicly available `rlkit` implementation of TD3 V. Pong et al., 2018. Our method is independent of the choice of RL algorithm, and we could apply residual RL to any other RL algorithm.

8.4 EXPERIMENTAL SETUP

We evaluate our method on the task shown in Fig. 37, both in simulation and in the real world. This section introduces the details of the experimental setup and provides an overview of the experiments.

8.4.1 *Simulated Environment*

We use MuJoCo (Todorov et al., 2012), a full-featured simulator for model-based optimization considering body contacts, to evaluate our method in simulation. This environment consists of a simulated Sawyer robot arm with seven degrees of freedom and a parallel gripper. We command the robot with a Cartesian-space position controller.

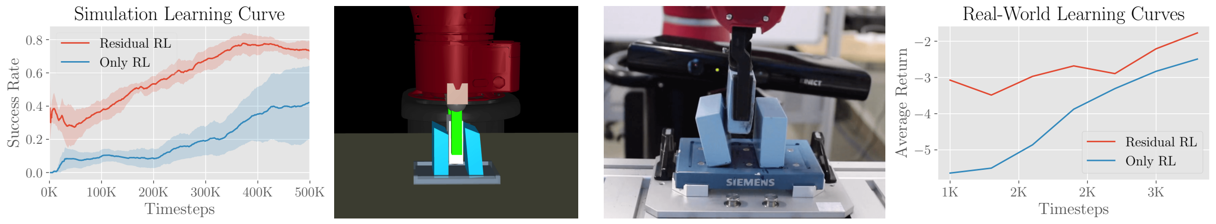


Figure 37: Block assembly task in simulation (left) and real-world (right). The task is to insert a block between the two blocks on the table without moving the blocks or tipping them over. In the learning curves, we compare our method with RL without any hand-engineered controller[1]. In both simulation and real-world experiments, we see that residual RL learns faster than RL alone, while achieving better performance than the hand-engineered controller.

8.4.2 *Real-World Environment*

The real-world environment is largely the same as the simulated environment, except for the controller, rewards, and observations. We command the robot with a compliant joint-space impedance controller we have developed to be smooth and tolerant of contacts. The positioning of the block being inserted is similar to the simulation but the observation is estimated from a camera-based tracking system as we do not have access to ground truth position information.

8.4.3 *Overview of Experiments*

In our experiments we evaluate the following research questions:

1. Does incorporating a hand-designed controller improve the performance and sample-efficiency of RL algorithms, while still being able to recover from an imperfect hand-designed controller?
2. Can our method allow robots to be more tolerant of variation in the environment?
3. Can our method successfully control noisy systems, compared to classical control methods?

1 In all simulation plots, we use 10 random seeds and report a 95% confidence interval for the mean.

8.5 EXPERIMENTS

8.5.1 *Sample Efficiency of Residual RL*

In this section, we compare our residual RL method with the human controller alone and RL alone. The following methods are compared:

1. Only RL: using the same underlying RL algorithm as our method but without adding a hand-engineered policy
2. Residual RL: our method which trains a superposition of the hand-engineered controller and a neural network policy, with RL

8.5.2 *Effect of Environment Variation*

In automation, environments can be subject to noise and solving manufacturing tasks become more difficult as variability in the environment increases. It is difficult to manually design feedback controllers that are robust to environment variation, as it might require significant human expertise and tuning. In this experiment, we vary the initial orientation of the blocks during each episode and demonstrate that residual RL can still solve the task. We compare its performance to that of the hand-engineered controller.

To introduce variability in simulation, on every reset we sampled the rotation of each block independently from a uniform distribution $U[-r, r], r \in \{0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3\}$.

Similarly, in the real world experiments, on every reset we randomly rotated each block to one of three orientations: straight, tilted clockwise, or tilted counterclockwise (tilt was $\pm$ 20° from original position).

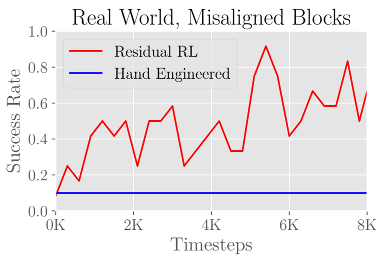
8.5.3 *Recovering from Control Noise*

Due to a host of issues, such as defective hardware or poorly tuned controller parameters, feedback controllers might have induced noise. Conventional feedback control policies are determined a priori and do not adapt their behavior from data. However, RL methods are known for their ability to cope with shifting noise distributions and are capable of recovering from such issues.

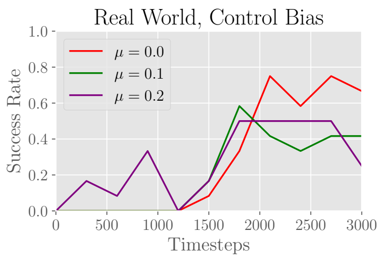
In this experiment, we introduce a control noise, including biased control noise, and demonstrate that residual RL can still successfully solve the task, while a hand-engineered controller cannot. The control noise follows a normal distribution and is

Misaligned?	No	Yes
Residual RL	20/20	15/20
Hand-engineered	20/20	2/20

(a)



(b)



(c)

Figure 38: Outcome of our residual RL method in different experiments during the block assembly task in the real-world. Success rate is recorded manually by human judgment of whether the blocks stayed upright and ended in the correct position. Plot (a) compares the insertion success of residual RL and hand-designed controller depending on the block orientation during run time. Plot (b) shows the success rate of the insertion process during training, where on every reset the blocks are randomly rotated: straight, tilted clockwise, or tilted counterclockwise (± 20°) and plot (c) shows the increasing success rate of our method for biased controllers as well even as control bias increases.

added to the control output of the system at every step:

$$u'_t = u_t + \mathcal{N}(\mu, \sigma^2) \tag{46}$$

To test tolerance to control noise, we set $\mu = 0$ and vary $\sigma \in [0.01, 0.1]$. In theory, RL could adapt to a noisy controller by learning more robust solutions to the task which are less sensitive to perturbations.

Furthermore, to test tolerance to a biased controller, we set $\sigma = 0.05$ and vary $\mu \in [0, 0.2]$. To optimize the task reward, RL can learn to simply counteract the bias.

8.5.4 *Sim-to-Real with Residual RL*

As an alternative to analytic solutions of real-world control problems, we can often instead model the forward dynamics of the problem (ie. a simulator). With access to such a model, we can first find a solution to the problem with our possibly inaccurate model, and then use residual RL to find a realistic control solution in the real world.

In this experiment, we attempt the block insertion task with the side blocks fixed in place. The hand-engineered policy π_H in this case comes from training a parametric policy in simulation of the same scenario (with deep RL). We then use this policy as initialization for residual RL in the real world.

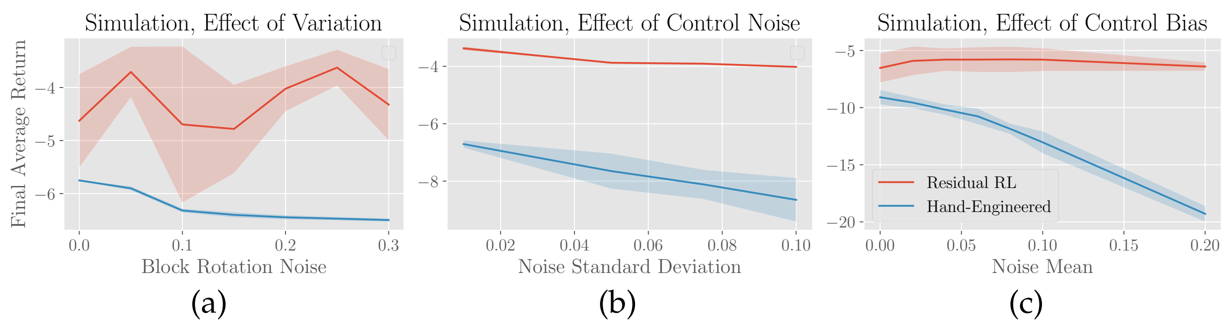


Figure 39: Simulation results for different experiments. In each plot, the final average return obtained by running the method for various settings of a parameter is shown. Plot (a) shows that residual RL can adjust to noise in the environment caused by rotation of the blocks in a range of 0 to $0.3\,\text{rad}$. In plot (b), residual RL finds robust strategies in order to reduce the effect of control noise, as the final average return is not greatly affected by the magnitude of noise. Plot (c) shows that residual RL can compensate for biased controllers and maintains good performance as control bias increases, while the performance of the hand-designed controller dramatically deteriorates with higher control bias.

8.6 RESULTS

We trained our method to optimize the insertion process in simulation as well as on physical hardware. This section provides the results of our discussed experiments and shows the functionality of our method.

8.6.1 *Sample Efficiency of Residual RL*

First, we compare residual RL and pure RL without a hand-engineered controller on the insertion task. Fig. 37 shows in simulation and real-world that residual RL achieves a better final performance and requires less samples than RL alone, both in simulation and on physical hardware. Unlike residual RL, the pure RL approach needs to learn the structure of the position control problem from scratch, which explains the difference in sample efficiency. As samples are expensive to collect in the real world, residual RL is better suited for solving real-world tasks. Moreover, RL shows a broader spatial variance during training and needs to explore a wider set of states compared to residual RL, which can be potentially dangerous in hardware deployments.

8.6.2 *Effect of Environment Variation*

In previous set of experiments, both standing blocks were placed in their initial position without any position or orientation error. In this case, the hand-engineered controller performs well, as both blocks are placed such that there is a sufficiently large defined gap for insertion. However, once the initial orientation of the blocks is randomized, the gap between the blocks and the goal position does not afford directly inserting from above. Therefore, the hand-engineered controller struggles to solve the insertion task, succeeding in only 2/20 trials, while residual RL still succeeds in 15/20 trials. These results are summarized in Fig. 38 (a). Rollouts from the learned policy are included in 47. In this experiment, the agent demonstrably learns consistent small corrective feedback behaviors in order to slightly nudge the blocks in the right direction without tipping them over, a behavior that is very difficult to manually specify.

The result of this experiment showcases the strength of residual RL. Since the human controller specifies the general trajectory of the optimal policy, environment samples are required only to learn this corrective feedback behavior. The real-world learning curve for the experiment in Fig. 38 (b) shows that this behavior is gradually acquired over the course of eight thousand samples, which is only about three hours of real-world training time.

We further studied the effect of the block orientation changing after every reset in simulation. The results are shown in Fig. 39 (a). The simulation results show that the performance of the hand-engineered controller decreases as the block rotation angle increases, whereas our control method maintains a constant average performance over different variations.

8.6.3 *Recovering from Control Noise*

In this experiment, we observe that residual RL is able to cope with actuator noise, including biased actuator noise. Quantitative results for simulation are shown in Fig. 39 (b) and (c). In Fig. 39 (c) our method keeps the average return constant and correct for biased controllers even as control bias increases, whereas the hand-engineered controller cannot compensate biased input and its performance deteriorates as control bias increases. The same applies for adding control noise to the control output as shown in Fig. 39 (b).

For the hardware experiments, only biased actuator noise is investigated. These results are shown in Fig. 38 (c). These learning curves show that even as more control bias is introduced, training in the real world proceeds without significant issues. This result suggests the potential for RL to address practical issues in automation such as sensor drift.

8.6.4 *Sim-to-Real with Residual RL*

The result of the sim-to-real experiment is shown in Fig. 40. In this experiment, each setting was run with three random seeds. Adding policy initialization from simulation significantly speeds up both RL and residual RL. In particular, residual RL with policy initialization from simulation successfully solves the task extremely quickly: in under one thousand timesteps of environment interaction. This method poses a highly sample efficient, practical way to solve robotics problems with difficult contact dynamics.

8.7 RELATED WORK

Reinforcement learning for robotics holds the promise of greater autonomy and reliability, which could be vital to improving our manufacturing processes beyond its current limitations. RL methods have been difficult to apply in robotics because of sample efficiency, safety, and stability issues. Still, RL has been used to allow robots to learn tasks such as playing table tennis (Peters et al., 2010), swinging up a cartpole and balancing a unicycle (Marc Peter Deisenroth and Rasmussen, 2011), grasping (Pinto et al., 2017; Levine et al., 2017), opening a door (S. Gu et al., 2017), and general manipulation tasks (Levine et al., 2016a; Haarnoja et al., 2018a). RL, particularly deep RL, tends to be data-hungry; even learning simple tasks can require many hours of interaction. To bring these methods into factories and warehouses, they must be able to consistently solve complex tasks, multi-step tasks. One way to enable these methods to solve these complex tasks is to introduce prior human knowledge into the learning system, as our method does.

Prior work in RL has incorporated human prior knowledge for solving tasks in various ways. One such way is reward shaping (Andrew Y. Ng et al., 1999), where additional rewards auxiliary to the real objective are included in order to guide the agent towards the desired behavior. Reward shaping can effectively encode a policy. For example, to train an agent to perform block stacking, each step can be encoded into the reward (Popov et al., 2017). Often, intensive reward shaping is key for RL to succeed at a task and prior work has even considered reward shaping as part of the learning system (Daniel et al., 2014).

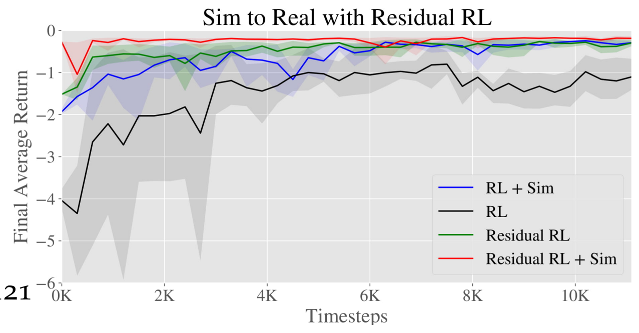


Figure 40: Real-world block insertion results using residual RL for sim-to-real transfer. "Sim" indicates that the real-world policy was initialized by reinforcement learning in simulation. Resid-

Reward shaping in order to tune agent behavior is a very manual process and recovering a good policy with reward shaping can be as difficult as specifying the policy itself. Hence, in our method we allow for human specification of both rewards and policies—whichever might be more practical for a particular task.

Further work has incorporated more specialized human knowledge into RL systems. One approach is to use trajectory planners in RL in order to solve robotics tasks (G. Thomas et al., 2018). However, since the method optimizes trajectory following instead of the task reward, generalization can be difficult when aspects of the environment change. Other work has focused on human feedback (Loftin et al., 2014; Saunders et al., 2018; Torrey and Taylor, 2013; Frank et al., 2008; Christiano et al., 2017) to inform the agent about rewards or to encourage safety. However, in many robotics tasks, providing enough information about the task through incremental human feedback is difficult.

Another way to include prior knowledge in RL is through demonstrations (Peters and Schaal, 2008c; Kober and Peter, 2008; Rajeswaran et al., 2018; Hester et al., 2018; Veerk et al., 2017; A. Nair et al., 2018a). Demonstrations can substantially simplify the exploration problem as the agent begins training having already received examples of high-reward transitions and therefore knows where to explore (Subramanian et al., 2016). However, providing demonstrations requires humans to be able to teleoperate the robot to perform the task. In contrast, our method only requires a conventional controller for motion, which ships with most robots.

Prior knowledge can also be induced through neural network architecture choices. Deep residual networks with additive residual blocks achieved state of the art results in many computer vision tasks by enabling training of extremely deep networks (He et al., 2016). In RL, structured control nets showed improvement on several tasks by splitting the policy into a linear module and a non-linear module (Srouji et al., 2018). Prior work in adaptive flight control has also considered compensating linearized controllers with neural networks (Johnson and Calise, 2000). Most closely related to our work, residual policy learning concurrently and independently explores training a residual policy in the context of simulated long-horizon, sparse-reward tasks with environment variation and sensor noise (T. Silver et al., 2018). Our work instead focuses on achieving practical real-world training of contact-intensive tasks.

8.8 CONCLUSION

In this chapter, we studied the combination of conventional feedback control methods with deep RL. We presented a control method that utilizes conventional feedback control along with RL to solve complicated manipulation tasks involving friction and contacts with unstable objects. We believe this approach can accelerate learning of many tasks, especially those where the control problem can be solved in large part by prior knowledge but requires some model-free reasoning to solve perfectly. Our results demonstrate that the combination of conventional feedback control and RL can circumvent the disadvantages of both and provide a sample efficient controller that can cope with contact dynamics.

8.9 CONTRIBUTION STATEMENT

The work in this chapter was performed in collaboration with Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, Sergey Levine (Johannink et al., 2019). T.J., S.B., and A.N. were joint co-first authors. A.N. proposed the project idea, managed the project, assisted with implementation, and wrote the paper. T.J. conducted the real-world experiments with assistance from J.L. The simulation experiments were conducted by S.B. The project was advised by A.K., M.L., and J.A.O. Both E.S. and S.L. advised on the project and assisted with writing.

9

DEEP REINFORCEMENT LEARNING FOR INDUSTRIAL INSERTION TASKS WITH VISUAL INPUTS AND NATURAL REWARDS

9.1 INTRODUCTION

In the previous chapter, we covered a method - residual RL - for incorporating conventional feedback controllers within deep reinforcement learning. In this chapter, we will use residual RL specifically for the task of industrial connector insertion. Many such industrial tasks are on the edge of automation but require a degree of adaptability that is difficult to achieve with conventional robotic automation techniques. While standard control methods, such as PID controllers, are heavily employed to automate many tasks in the context of positioning, tasks that require significant adaptability or tight visual perception-control loops are often beyond the capabilities of such methods, and therefore are typically performed manually. Standard control methods can struggle in presence of complex dynamical phenomena that are hard to model analytically, such as complex contacts. Reinforcement learning (RL) offers a different solution, relying on trial and error learning instead of accurate modeling to construct an effective controller. RL with expressive function approximation, i.e. deep RL, has further shown to automatically handle high dimensional inputs such as images (Mnih et al., 2013).

However, deep RL has thus far not seen wide adoption in the automation community due to several practical obstacles. Sample efficiency is one obstacle: tasks must be completed without excessive interaction time or wear and tear on the robot. Progress in recent years on developing better RL algorithms has led to significantly better sample efficiency, even in dynamically complicated tasks (Haarnoja et al., 2018a; Hessel et al., 2018), but remains a challenge for deploying RL in real-world robotics contexts. Another major, often underappreciated, obstacle is goal specification: while prior work in RL assumes a reward signal to optimize, it is often carefully shaped to allow the system to

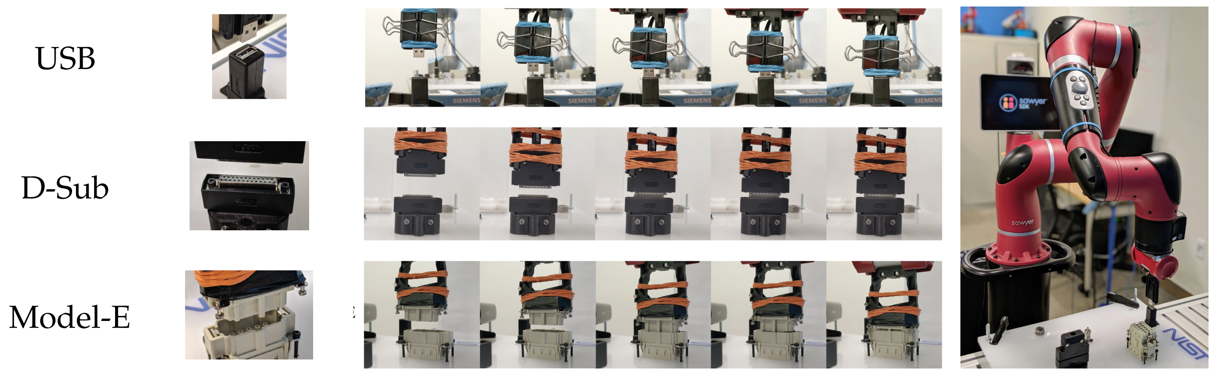


Figure 41: We train policies directly in the real world to solve connector insertion tasks from raw pixel input and without access to ground-truth state information for reward functions. Left: top-down views of the connectors. Middle: a rollout from a learned policy that successfully completes the insertion task for each connector is shown. Right: a full view of the robot setup. Videos of the results are available at industrial-insertion-rl.github.io

learn (Andrew Y. Ng et al., 1999; Popov et al., 2017; Daniel et al., 2014). Obtaining such dense reward signals can be a significant challenge, as one must additionally build a perception system that allows computing dense rewards on state representations. Shaping a reward function so that an agent can learn from it is also a manual process that requires considerable manual effort. An ideal RL system would learn from rewards that are natural and easy to specify. How can we enable robots to autonomously perform complex tasks without significant engineering effort to design perception and reward systems?

We first consider an end-to-end approach that learns a policy from images, where the images serve as both the state representation and the goal specification. Using goal images is not fully general, but can successfully represent tasks when the task is to reach a final desired state (A. Nair et al., 2018b). Specifying goals via goal images is convenient, and makes it possible to specify goals with minimal manual effort. Using images as the state representation also allows a robot to learn behaviors that utilize direct visual feedback, which provides some robustness to sensor and actuator noise.

Secondly, we consider learning from simple and sparse reward signals. Sparse rewards can often be obtained conveniently, for instance from human-provided labels or simple instrumentation. In many electronic assembly tasks, which we consider here, we can directly detect whether the electronics are functional, and use that signal as a reward. Learning from sparse rewards poses a challenge, as exploration with sparse reward signals is difficult, but by using sufficient prior information about the task, one can overcome this challenge. To handle this challenge, we extend the residual RL approach Johannink et al., 2019; T. Silver et al., 2018, which learns a parametric policy on top of a fixed, hand-specified controller, to the setting of vision-based manipulation.

In our experiments, we show that we can successfully complete real-world tight tolerance assembly tasks, such as inserting USB connectors, using RL from images with reward signals that are convenient for users to specify. We can learn from only a sparse reward based on the electrical connection for a USB adapter plug, and we demonstrate learning insertion skills with rewards based only on goal images. These reward signals require no extra engineering and are easy to specify for many tasks. Beyond showing the feasibility of RL for solving these tasks, we evaluate multiple RL algorithms across three tasks and study their robustness to imprecise positioning and noise.

9.2 RELATED WORK

Learning has been applied previously in a variety of robotics contexts. Different forms of learning have enabled autonomous driving (Pomerleau, 1989), biped locomotion (Nakanishi et al., 2004), block stacking (Marc Peter Deisenroth et al., 2011b), grasping (Pinto and Abhinav Gupta, 2016), and navigation (Giusti et al., 2015; Pathak et al., 2018). Among these methods, many involve reinforcement learning, where an agent learns to perform a task by maximizing a reward signal. Reinforcement learning algorithms have been developed and applied to teach robots to perform tasks such as balancing a robot (Marc Peter Deisenroth and Rasmussen, 2011), playing ping-pong (Peters et al., 2010) and baseball (Peters and Schaal, 2008c). The use of large function approximators, such as neural networks, in RL has further broadened the generality of RL (Mnih et al., 2013). Such techniques, called "deep" RL, have further allowed robots to be trained directly in the real world to perform fine-grained manipulation tasks from vision (Levine et al., 2016a), open doors (S. Gu et al., 2016), play hockey (Chebotar et al., 2017a), stack Lego blocks (M. Zhang et al., 2019), use dexterous hands (H. Zhu et al., 2019), and grasp objects (Kalashnikov et al., 2018b). In this work we further explore solving real-world robotics tasks using RL.

Many RL algorithms introduce prior information about the specific task to be solved. One common method is reward shaping (Andrew Y. Ng et al., 1999), but reward shaping can become arbitrarily difficult as the complexity of the task increases. Other methods incorporate a trajectory planner (G. Thomas et al., 2018) but for complex assembly tasks, trajectory planners require a host of information about objects and geometries which can be difficult to provide.

Another body of work on incorporating prior information studies using demonstrations either to initialize a policy (Peters and Schaal, 2008c; Kober and Peter, 2008), infer reward functions using inverse reinforcement learning (Finn et al., 2016a; Ziebart et al., 2008) or to improve the policy throughout the learning procedure (Hester et al., 2018; A. Nair et al., 2018a; Rajeswaran et al., 2018). These methods require multiple demonstrations, which can be difficult to collect, especially for assembly tasks, although learning a reward function by classifying goal states (Singh et al., 2019) may partially alleviate this issue. More recently, manually specifying a policy and learning the residual task has been proposed (Johannink et al., 2019; T. Silver et al., 2018). In this work we evaluate both residual RL and combining RL with learning from demonstrations.

Previous work has also tackled high precision assembly tasks, especially insertion-type tasks. One line of work focuses on obtaining high dimensional observations, including geometry, forces, joint positions and velocities (R. Li et al., 2014; Tamar et al., 2017; Inoue et al., 2017; Luo et al., 2019), but this information is not easily procured, increasing complexity of the experiments and the supervision required. Other work relies on external trajectory planning or very high precision control (Inoue et al., 2017; Tamar et al., 2017), but this can be brittle to error in other components of the system, such as perception. We show how our method not only solves insertion tasks with much less information about the environment, but also does so under noisy conditions.

9.3 ELECTRIC CONNECTOR PLUG INSERTION TASKS

In this work, we empirically evaluate learning methods on a set of electric connector assembly tasks, pictured in Fig. 41. Connector plug insertions are difficult for two reasons. First, the robot must be very precise in lining up the plug with its socket. As we show in our experiments, errors as small as ± 1 mm can lead to consistent failure.

Second, there is significant friction when the connector plug touches the socket, and the robot must learn to apply sufficient force in order to insert the plug. Image sequences of successful insertions are shown in Fig. 41, where it is also possible to see details of the gripper setup that we used to ensure a failure free, fully automated training process. In our experiments, we use a 7 degrees of free-

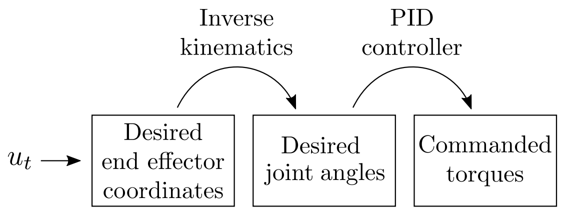


Figure 42: Illustration of the robot's cascade control scheme. The actions u_t are computed at a frequency of up to 10 Hz, desired joint angles are obtained by inverse kinematics, and a joint-space impedance controller with anti-windup PID control commands actuator torques at 1000 Hz.

dom Sawyer robot with end-effector con-
trol, meaning that the action signal u_t can be interpreted as the relative end-effector movement in Cartesian coordinates. The robot's underlying internal control pipeline is illustrated in Figure 42.

To comprehensively evaluate connector assembly tasks, we experiment on a variety of connectors. Each connector offers a different challenge in terms of required precision and force to overcome friction. We chose to benchmark the controllers performance on the insertion of a USB connector, a U-Sub connector, and a waterproof Model-E connector manufactured by MISUMI. All the explored use cases were part of the IROS 2017 Robotic Grasping and Manipulation Competition (Falco et al., 2018), included as part of a task board developed by NIST to benchmark the performance of assembly robots.

9.3.1 *Adapters*

In the following we describe the used adapters, USB, D-Sub, and Model-E. The observed difficulty of the insertion increases in that order.

USB. The USB connector is a ubiquitous, widely-used connector and offers a challenging insertion task. Because the adapter becomes smoother and therefore easier to insert over time due to wear and tear, we periodically replace the adapter. Of the three tested adapters, the USB adapter is the easiest.

D-sub. Inserting this adapter requires aligning several pins correctly, and is therefore more sensitive than inserting the USB adapter. It also requires more downward force due to a tighter fit.

Model-E. This adapter is the most difficult of the three tested connectors as it contains several edges and grooves to align and requires significant downward force to successfully insert the part.

9.3.2 *Experimental Settings*

We consider three settings in our experiments in order to evaluate how plausible it is to solve these tasks with more convenient state representations and reward functions and to evaluate the performance of different algorithms changes as the setting is modified.

3.2.1 Visual. In this experiment, we evaluate whether the RL algorithms can learn to perform the connector assembly tasks from vision without having access to state information. The state provided to the learned policy is a 32×32 grayscale image, such as shown in Fig. 43.

For goal specification, we use a goal image, avoiding the need for state information to compute rewards. The reward is the pixelwise L1 distance to the given goal image. Being able to learn from such a setup is compelling as it does not require any extra state estimation and many tasks can be specified easily by a goal image.

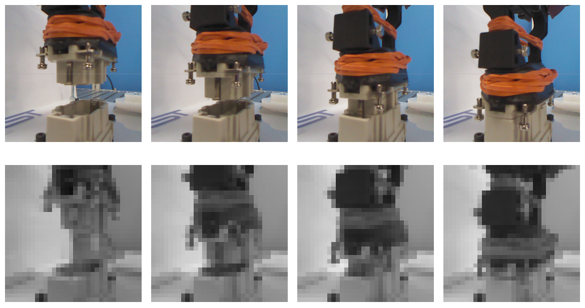


Figure 43: Successful insertion on the Model-E connector. The image-based RL algorithms receives only receives the 32×32 grayscale image as the observation.

3.2.2. Sparse. In this experiment, the reward is obtained by directly measuring whether the connection is alive and transmitting:

$$r = \begin{cases} 1, & \text{if insertion signal detected} \\ 0, & \text{else.} \end{cases} \tag{47}$$

This is the exact true reward for the task of connecting a cable, and can be naturally obtained in many manufacturing systems. As state, the robot is given the Cartesian coordinates of the end-effector x_t and the vertical force f_z that is acting on the end-effector. We could only automatically detect the USB connection, so we only include the USB adapter for the sparse experiments.

3.2.3. Dense. In this experiment, the robot receives a manually shaped reward based on the distance to the target location x^*. We use the reward function

$$r_t = -\alpha \cdot \|x_t - x^*\|_1 - \frac{\beta}{(\|x_t - x^*\|_2 + \varepsilon)} - \varphi \cdot f_z, \tag{48}$$

where $0 < \varepsilon \ll 1$. The hyperparameters are set to $\alpha = 100$, $\beta = 0.002$, and $\varphi = 0.1$. When an insertion is indicated through a distance measurement, the sign of the force term flips, so that $\varphi = -0.1$ when the connector is inserted. This rewards the agent for pressing down after an insertion and showed to improve the learning process. The force measurements are calibrated before each rollout to account for measurement bias and to decouple the measurements from the robot pose.

9.4 METHODS

To solve the connector insertion tasks, we consider and evaluate a variety of RL algorithms.

9.4.1 *Efficient Off-Policy Reinforcement Learning*

In this paper we specifically consider two off-policy continuous control reinforcement learning algorithms that lend themselves well to real-world learning as they are sample efficient, stable, and require little hyperparameter tuning.

Twin Delayed Deep Deterministic Policy Gradients (TD3). Like DDPG, TD3 optimizes a deterministic policy (Fujimoto et al., 2018b) but uses two Q-function approximators to reduce value overestimation Van Hasselt et al., 2016 and delayed policy updates to stabilize training.

Soft Actor Critic (SAC). SAC is an off-policy value-based reinforcement learning method based on the maximum entropy reinforcement learning framework with a stochastic policy (Haarnoja et al., 2018a).

We used the implementation of these RL algorithms publicly available at `rlkit` (V. Pong et al., 2018).

9.4.2 *Residual Reinforcement Learning*

As discussed in chapter 8, instead of randomly exploring from scratch, we can inject prior information into an RL algorithm in order to speed up the training process, as well as to minimize unsafe exploration behavior. In residual RL, actions u_t are chosen by additively combining a fixed policy $\pi_H(s_t)$ with a parametric policy $\pi_\theta(u_t|s_t)$:

$$u_t = \pi_H(s_t) + \pi_\theta(s_t). \tag{49}$$

The parameters θ can be learned using any RL algorithm. In this work, we evaluate both SAC and TD3, explained in the previous section.

A simple P-controller serves as the hand-designed controller π_H of our experiments. The P-controller operates in Cartesian space and calculates the current control action by

$$\pi_H(s_t) = -k_p \cdot (x_t - x^*), \tag{50}$$

where x^* denotes the commanded goal location. As control gains we use $k_p = [1, 1, 0.3]$.

This P-controller quickly centers the end-effector above the goal position and reaches the goal after about 10 time steps from the reset position, which is located 5cm above the goal.

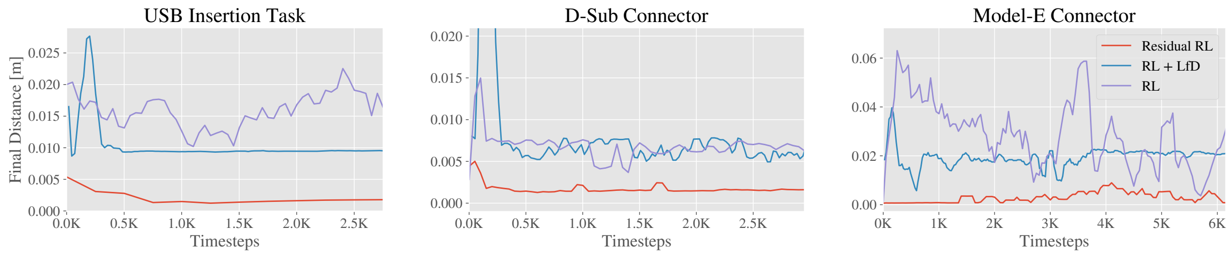


Figure 44: Resulting final mean distance during the vision-based training. The comparison includes RL, residual RL, and RL with learning from demonstrations. Only residual RL manages to deal with the high-dimensional input and consistently solve all the tasks after the given amount of training. The other methods learn to move downwards, but often get stuck in the beginning of the insertion and fail to recover from unsuccessful attempts.

9.4.3 *Learning from Demonstrations*

Another method to incorporate prior information is to use demonstrations from an expert policy to guide exploration during RL. We first collected demonstrations with a joystick controller. Then, we add a behavior cloning loss while performing RL that pushes the policy towards the demonstrator actions, as previously considered in (A. Nair et al., 2018a). Instead of DDPG, the underlying algorithm RL algorithm used is TD3.

9.5 EXPERIMENTS

We evaluate our method, which combines residual RL with easy-to-obtain reward signals, on a variety of connector assembly tasks performed on a real robot. In this section, we consider two types of natural rewards that are intuitive for users to provide: an image directly specifying a goal and a binary sparse reward indicating success. For both cases, we report success rates on tasks they solve. We aim to answer the following questions: (1) Can such trained policies provide comparable performance to policies that are trained with densely-shaped rewards? (2) Are these trained policies robust to small variations and noise?

5.1 Vision-based Learning. For the vision-based learning experiments, we use only raw image observations and ℓ_1 distance between the current image and goal image as the reward signal. Sample images that the robot received are shown in Fig. 43. We evaluate this type of reward on all three connectors. In our experiments, we use 32×32 grayscale images.

5.2 Learning from Sparse Rewards. In the sparse reward experiment, we use the binary signal of the connector being electrically connected as the reward signal. This experiment is most applicable to electronic manufacturing settings where the electrical connection between connectors can be directly measured. We only evaluate the sparse reward setting on the USB connector, as it was straightforward to obtain the electrical connection signal.

5.3 Perfect State Information. After evaluating the tasks in the above settings, we further evaluate with full state information with a dense and carefully shaped reward signal, given in Eq. 48, that incorporates distance to the goal and force information. Evaluating in this setting gives us an "oracle" that can be compared to the previous experiments in order to understand how much of a challenge sparse or image rewards pose for various algorithms.

5.4 Robustness. For safe and reliable future usage, it is required that the insertion controller is robust against small measurement or calibration errors that can occur when disassembling and reassembling a mechanical system. In this experiment, small goal perturbations are introduced in order to uncover the required setup precision of our algorithms.

5.5 Exploration Comparison. One advantage of using reinforcement learning is the exploratory behavior that allows the controller to adapt from new experiences unlike a deterministic control law. The two RL algorithms we consider in this paper, SAC and TD3, explore differently. SAC maintains a stochastic policy, and the algorithm also adapts the stochasticity through training. TD3 has a deterministic policy, but uses another noise process (in our case Gaussian) to inject exploratory behavior during training time. We compare the two algorithms, as well as when they are used in conjunction with residual RL, in order to evaluate the effect of the different exploration schemes.

9.6 RESULTS

We analyze the performance of policies learned with residual RL, as well as other methods, based on their ability to achieve the task goal, as well as the distance of the final object location to the goal pose over the course of training. To study the robustness of the learned policies, we also evaluate them in conditions where the goal connector posi-

D-Sub Connector		Goal	
		Perfect	Noisy
Pure RL	Dense	16%	0%
	Images, SAC	0%	0%
	Images, TD3	12%	12%
RL + LfD	Images	52%	52%
Residual RL	Dense	**100%**	60%
	Images, SAC	**100%**	**64%**
	Images, TD3	52%	52%
Human	P-Controller	**100%**	44%

Model-E Connector		Goal	
		Perfect	Noisy
Pure RL	Dense	0%	0%
	Images, SAC	0%	0%
	Images, TD3	0%	0%
RL + LfD	Images	20%	20%
Residual RL	Dense	**100%**	**76%**
	Images, SAC	**100%**	**76%**
	Images, TD3	0%	0%
Human	P-Controller	52%	24%

Table 3: We report average success out of 25 policy executions after training is finished for each method. For noisy goals, noise is added in form of ± 1 mm perturbations of the goal location. Residual RL, particularly with SAC, tends to be the best performing method across all three connectors. For the Model-E connector, only residual RL solves the task in the given amount of training time.

tion is perturbed, in order to understand the tolerance of RL policies to imprecise object placement.

9.6.1 *Vision-based Learning*

The results of the vision-based experiment are shown in Fig. 44. Our experiments show that a successful and consistent vision-based insertion policy can be learned from relatively few samples using residual RL.

This result suggests that goal-specification through images is a practical way to solve these types of industrial tasks. Although image-based rewards are often very sparse and hard to learn from, in this case the distance between images corresponds to a relatively dense reward signal which is sufficient to distinguish the different stages of the insertion process.

Interestingly, during training with standard RL, the policy would sometimes learn to "hack" the reward signal by moving down in the image in front of or behind the socket. In contrast, the stabilizing human-engineered controller in residual RL provides sufficient horizontal control to prevent this. The initial controller also scaffolds the learning process, by providing a very strong initialization that requires the reinforcement learning algorithm to only learn the final phase of the insertion. This produces substantially better performance in conjunction with vision-based rewards.

9.6.2 *Learning From Sparse Rewards*

In this experiment, we compare these methods on the USB insertion task with sparse rewards. The results are reported in Fig. 45. All methods are able to achieve very high success rates in the sparse setting. This result shows that we can learn precise industrial insertion tasks in sparse-reward settings, which can often be obtained much more easily than a dense, shaped reward. In fact, prior work has found that the final policy for sparse rewards can outperform the final policy for dense rewards as it does not suffer from a misspecified objective (Andrychowicz et al., 2017b).

9.6.3 *Perfect State Information*

The results of the experiment with perfect state information and dense rewards is shown in Fig. 46. In this case, residual RL still outperforms standard RL, though the better-

USB Connector		Goal	
		Perfect	Noisy
Pure RL	Dense	28%	20%
	Sparse, SAC	16%	8%
	Sparse, TD3	44%	28%
	Images, SAC	36%	32%
	Images, TD3	28%	28%
RL + LfD	Sparse	**100%**	32%
	Images	88%	60%
Residual RL	Dense	**100%**	**84%**
	Sparse, SAC	88%	**84%**
	Sparse, TD3	**100%**	36%
	Images, SAC	**100%**	80%
	Images, TD3	0%	0%
Human	P-Controller	**100%**	60%

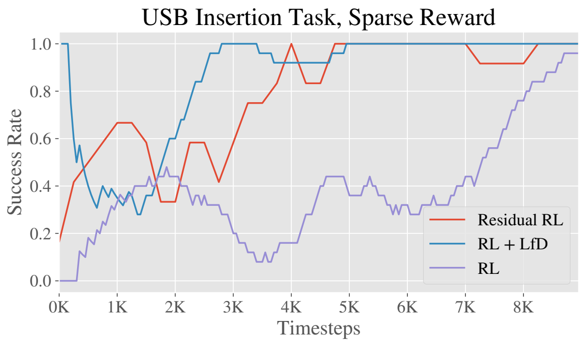


Figure 45: Learning curves for solving the USB insertion task with a sparse reward. Final distance to goal is shown; lower is better. Residual RL and RL with learning from demonstrations both solve the task relatively quickly, while RL alone takes about twice as long to solve the task at the same performance.

Table 4: Learning curves for solving the USB insertion task with a sparse reward. Final distance to goal is shown; lower is better. Residual RL and RL with learning from demonstrations both solve the task relatively quickly, while RL alone takes about twice as long to solve the task at the same performance.

shaped reward enables standard RL to make more initial progress than with the other reward signals. However, the hand-designed shaped reward function makes it harder for the policy to actually perform the full insertion, potentially because the more complex reward landscape provides other competing goals to the policy. The final performance with sparse rewards on the USB insertion task is substantially better.

9.6.4 *Robustness*

In the previous set of experiments, the goal locations were known exactly. In this case, the hand-engineered controller performs well. However, once noise is added to the goal location, the deterministic P-controller struggles. To test robustness, a goal perturbation is created artificially, and the controllers are tested under this condition. All results of our robustness evaluations are listed in Fig. 3 and Fig. **??**. In the presence of a ±1mm perturbation, the residual RL controller succeeds more often on the USB and D-Sub tasks, and significantly more often on the Model-E task. Unlike the hand-engineered

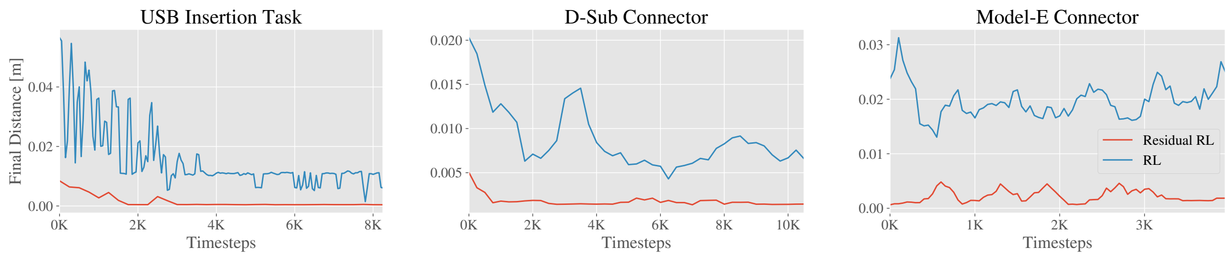


Figure 46: Plots of the final mean distance to the goal during the state-based training. Final distances greater than 0.01 m indicate unsuccessful insertions. Here, the residual RL approach performs noticeably better than pure RL and is often able to solve the task during the exploration in the early stages of the training.

controller, residual RL consistently solved this task and overcame goal perturbations in 16/25 trials. The agent demonstrably learns small but consistent corrective feedback behaviors in order to move in the right direction during the descent motion, a behavior that is very difficult to specify manually. This behavior illustrates the strength of residual RL. Since the human controller already specifies the general trajectory of the optimal policy, environment samples are only required to learn this corrective feedback behavior.

9.6.5 *Exploration Comparison*

All experiments were also performed using TD3 instead of SAC. The final success rates of these experiments are included in Fig. 3. When combined with residual RL, SAC and TD3 perform comparably. However, TD3 is often substantially less robust. These results are likely explained by the exploration strategy of the two algorithms. TD3 has a deterministic policy and fixed noise during training, so once it observes some high-reward states, it quickly learns to repeat that trajectory. SAC adapts the noise to the correct scale, helping SAC stay robust to small perturbations, and because SAC learns the value function for a stochastic policy, it is able to handle some degree of additive noise effectively. We found that the outputted action of TD3 approaches the extreme values at the edge of the allowed action space, while SAC executed less extreme actions, which likely further improved robustness.

9.7 CONCLUSION

In this paper, we studied deep reinforcement learning in a practical setting, and demonstrated that deep RL can solve complex industrial assembly tasks with tight tolerances.

We showed that we can learn insertion policies with raw image observations with either binary outcome-based rewards, or rewards based on on goal images. We conducted a series of experiments for various connector type assemblies, and demonstrated the feasibility of our method under challenging conditions, such as noisy goal specification and complex connector geometries. Reinforcement learning algorithms that can automatically learn complex assembly tasks with easy-to-specify reward functions have the potential to automate a wide range of assembly tasks, making this technology a promising direction forward for flexible and capable robotic manipulators.

There remains significant challenges for applying these techniques in more complex environments. One practical direction for future work is focusing on multi-stage assembly tasks through vision. This would pose a challenge to the goal-based policies as the background would be visually more complex. Moreover, multi-step tasks involve adapting to previous mistakes or inaccuracies, which could be difficult but should be able to be handled by RL. Extending the presented approach to multi-stage assembly tasks will pave the road to a higher robot autonomy in flexible manufacturing.

9.8 CONTRIBUTION STATEMENT

The work in this chapter was performed in collaboration with Gerrit Schoettler, Jianlan Luo, Shikhar Bahl, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine (Gerrit Schoettler et al., 2019). G.S., and A.N. were joint co-first authors. A.N. managed the project, assisted with implementation, and wrote the paper. G.S. conducted the real-world experiments with assistance from J.L. The simulation experiments were conducted by S.B. The project was advised by J.A.O. Both E.S. and S.L. advised on the project and assisted with writing.

10

LEARNING ON THE JOB: INDUSTRIAL INSERTION OF NOVEL CONNECTORS FROM VISION

10.1 INTRODUCTION

The previous chaper demonstrated industrial insertion for single connectors from vision, but the true promise of learning-based robotics is generalization beyond the training data for robust policies in novel scenarios. Generalizable policies require broad and diverse datasets, but for realistic applications, learning policies that can always generalize in zero shot to new objects and environments is often infeasible – indeed, even humans do not exhibit such universal generalization capabilities. Instead, when faced with a task that we don't precisely know how to do, we can quickly learn the task by leveraging our prior knowledge and a little bit of practice. Reinforcement learning provides us with a way to implement this kind of *learning on the job*, using online finetuning in the new domain or task, and potentially even extending it into a lifelong learning system where the robot improves its generalization capacity continually with each new task it masters.

However, instantiating this concept in a practical robotics setting requires overcoming a number of obstacles. The robot must be able to combine large amounts of diverse offline data with small amounts of targeted online experience, and do so in a way that doesn't require revisiting previously learned tasks or domains, which means that we need an offline RL algorithm that supports online finetuning. Perhaps more importantly, the entire finetuning process must be supported by the robot's own sensors, without privileged information or environment instrumentation, so as to retain the benefits of autonomous learning. In particular, this means that when adapting to a new task, the robot must be able to evaluate on its own whether it is making progress on the task, using a learned reward function.

We study this problem in the setting of learning a policy from vision for performing industrial insertion tasks. This family of assembly tasks, including plugging in connectors

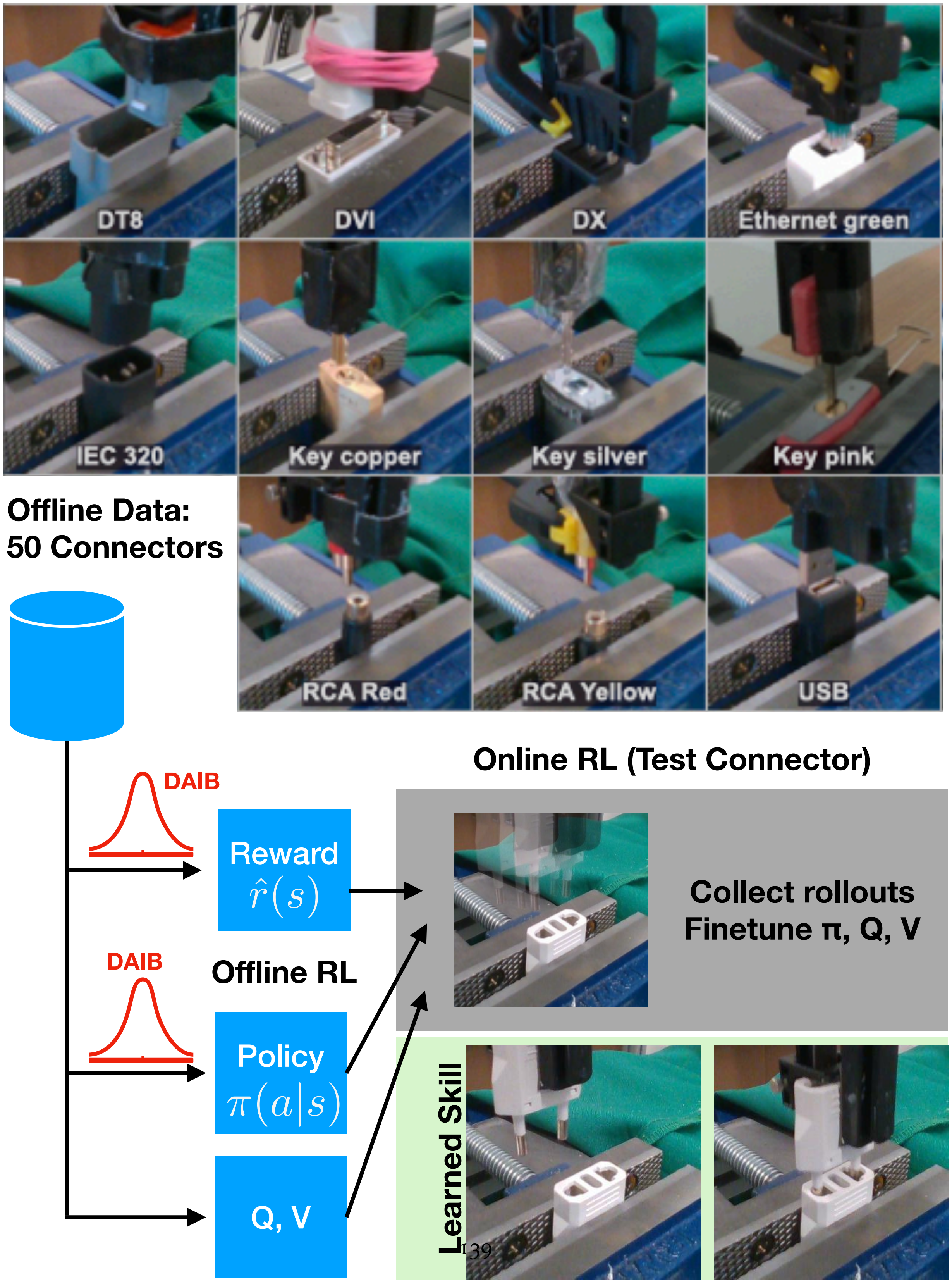


Figure 47: We use offline reinforcement learning on insertion data on a diverse set of connectors (left) followed by online finetuning to solve a connector-socket insertion task from vision on a previously unseen connector (right).

into sockets, screwdrivers into screw intrusions, setting screws, and so on, are found in many stages of manufacturing. When automated in factories today, these tasks are done by robots with specialized control algorithms that rely on precise localization of the socket location. For robots to perform these insertion tasks in industrial and warehouse settings with less human supervision, or in unstructured environments such as homes, they must rely on highly accurate state information of the external world (ie. socket position and in-hand pose estimation). But such state estimation, using either machine learning or computer vision approaches, is brittle on unseen connectors. To solve the general problem of inserting a novel connector, one promising solution is to generalize previously collected experience of connector insertion to learn a policy to insert connectors from vision. Among these tasks, there is enough variability to require generalization and adaptation, but also enough internal structural regularity that we expect transfer between different connectors. We first collected a large offline dataset with insertion data of 50 connectors across 2 robots and diverse backgrounds with actions, images, and sparse reward labels. How can a robot generalize from vision to test connectors in this setting, utilizing offline reinforcement learning from this data to enable active online fine-tuning on a new connector?

The key insight is that we need to (1) find common structure between domains while preserving important domain-specific information and (2) adapt to new tasks quickly with online data if the zero-shot solution is not sufficient. For finding common structure while preserving important domain-specific information, we propose a split representation that combines domain adversarial neural networks Ganin et al., 2016 for domain invariance and a variational information bottleneck Alemi et al., 2017 for controlling the flow of domain-specific information. This representation, which we call domain adversarial information bottleneck (DAIB) is used first for learning a robust reward function to detect successful insertions for an unseen connector. Next, we use the representation for training a reward function, policy, and Q function that can also generalize to new connectors, by modifying implicit Q-learning (IQL), an offline RL algorithm amenable to online fine-tuning, to use DAIB. Then, during online finetuning, this auxiliary objective can be used in combination with online RL to enable fast learning of novel connectors.

We present two major contributions. First, we propose a novel representation learning method that allows better generalization of policies and reward functions. This enables fast finetuning of vision-based IQL policies on a new domain. Second, we present a system based off of this representation learning method to insert connectors robustly from vision without the need of accurate socket localization, both for observations and rewards. We outperform a regression-based baseline on the same dataset that attempts to localize the socket. We show that new tasks can be fine-tuned within 200 trials, given our dataset of off-policy data from Y prior tasks of Z trajectories. This system allows us to finetune IQL to a test connector, increasing performance significantly over the offline performance. Our dataset of robotic insertion of 50 connectors, as well as pretrained features and reward models will be made public at <URL>.

10.2 RELATED WORK

Our work proposes a system for finetuning an offline reinforcement learning policy on new tasks in realistic industrial insertion scenarios, using a novel representation learning method for generalization to a test domain. In this section, we discuss prior work on reinforcement learning, industrial insertion, and representation learning.

Reinforcement learning for robotics. Reinforcement learning has been applied successfully to a variety of robotics tasks in both manipulation Peters and Schaal, 2008c; Kober and Peter, 2008; Marc Peter Deisenroth and Rasmussen, 2011; Levine et al., 2016a; Levine et al., 2017; H. Zhu et al., 2019 and locomotion Giusti et al., 2015; Nakanishi et al., 2004; Kalakrishnan et al., 2009 settings. To utilize offline datasets with diverse data in robotics, algorithms developed for offline reinforcement learning Fujimoto et al., 2019a; Kumar et al., 2020a; A. Nair et al., 2020; Yifan Wu et al., 2020 have been studied in the robotics setting Singh et al., 2020b; Singh et al., 2020a; Chebotar et al., 2021; Kalashnikov et al., 2021; Kumar et al., 2021. A subset of offline RL algorithms are amenable to finetuning A. Nair et al., 2020; Villaflor et al., 2020; Meng et al., 2021; S. Lee et al., 2021; Kostrikov et al., 2021b. Our work builds on the direction of offline pretraining followed by online finetuning in robotics. But beyond this line of work, we focus on finetuning from visual input in realistic settings with multiple domains and without ground truth reward functions at test time.

In this respect, our work is closest to prior work on self-supervised RL that does not assume an external reward function and instead learns it from data. One class of self-supervised reinforcement learning methods uses goal-conditioned RL with self-supervised rewards L P Kaelbling, 1993; Schaul et al., 2015b; Adrien Baranes and Pierre-Yves Oudeyer, 2012b; Andrychowicz et al., 2017b; A. Nair et al., 2018b; Nachum et al., 2018; Held et al., 2018; Pr et al., 2018c; Warde-Farley et al., 2019; Vitchyr H Pong et al., 2020a; Khazatsky et al., 2021b. While general, this class of methods is a poor fit for industrial insertion, as high precision is required in both the policy and in evaluating rewards. Instead, we train a domain generalizing reward classifier from prior data. Prior methods have used learned rewards Vitchyr H. Pong et al., 2022 and classifier rewards have been proposed as a scalable solution for robotics tasks previously Fu et al., 2018; Singh et al., 2019. However, learned rewards have not been shown to be useful for fine-tuning in novel robotic domains previously. Because we focus on applying offline RL and fine-tuning from vision in the industrial insertion setting in this work, few-domain generalization of the reward function is vital for our method to work in practice.

Robotic insertion. Prior work has discussed reinforcement learning for robotic insertion. Initial work in this direction focuses on learning for a single connector from ground-truth state information Lian et al., 2021; Johannink et al., 2019; Gerrit Schoettler et al., 2019. In this case, the RL algorithm must learn to navigate the specific dynamics of the single connector, but does not generalize across connectors. More recent work has considered using meta-learning to generalize and improve few-shot between domains G. Schoettler et al., 2020. Zhao et al. use offline reinforcement learning and finetuning combined with meta-learning to adapt to a new connector T. Z. Zhao et al., 2022. This work assumes a known position of the socket and consistent grasping of the connector, and is robust to a small amount ($\pm 1mm$) of noise. In the case of known socket position with a small amount of error, the learning algorithm must learn a structured noise or exploration strategy that can overcome these errors. In contrast, we initialize connectors within $\pm 20mm$ of the socket (20$\times$ the initial variance), which requires the robot to rely on visual feedback since blind exploration will rarely succeed.

Closest to our work is prior work that also uses pixel input for robotic insertion. Luo et al. incorporate vision alongside proprioception, using a VAE to embed pixel input Jianlan Luo et al., 2021. InsertionNet uses a vision system to localize the object and socket, operating on a "residual policy" which is learned from state by supervised learning Spector and D. D. Castro, 2021. InsertionNet 2.0 incorporates contrastive representation learning to improve performance. These prior works collect data on a single connector and then show robust insertion of that connector. In contrast, our work focuses on what can be done to leverage prior experience for a novel connector without having access to localization of the socket for supervision. Our work also demonstrates robustness to significantly larger variation in initial connector pose, up to 20mm error, than prior work. For visual generalization to a test connector from our offline dataset of 50 training connectors, a suitable representation learning method is vital.

Representation learning objectives for reinforcement learning. Many prior methods have explored representation methods for improving the sample efficiency of RL algorithms. These include reconstructive objectives Lange and M. A. Riedmiller, 2010; Lange et al., 2012c; Finn et al., 2016b, bisimulation Ferns et al., 2004; P. S. Castro, 2020; A. Zhang et al., 2021, contrastive methods Laskin et al., 2020; Nguyen et al., 2021, latent space prediction Schwarzer et al., 2020, mutual information , and other auxiliary tasks Jonschkowski et al., 2017a; Ghosh et al., 2018; Sax et al., 2019. In this work, the major representation learning challenge for finetuning to function is to be able to generalize to new domains from prior domains. Thus, most closely related to our work is work on domain generalization and domain adaptation. Domain adaptation methods generalize from source domains to a target domain, usually by matching the distribution of features between domains via matching statistics Tzeng et al., 2014; B. Sun and Saenko, 2016; Long et al., 2015 or using an adversarial loss Ganin et al., 2016; Bousmalis et al., 2016. Successfully matching distributions makes features indistinguishable; however, in our case, it may be possible that domain-specific information is also important. In robotics, domain adaptation has been applied successfully in the sim-to-real setting Bousmalis et al., 2017; James et al., 2018.

In this work, we focus on two unique aspects of representation learning for reinforcement learning. First, stabilizing and accelerating offline reinforcement learning and finetuning of convolutional neural network policies. Second, aligning domains via reward labels in order to generalize well to new domains and finetune quickly in the new domain.

10.3 BACKGROUND

10.3.1 *Reinforcement learning.*

In reinforcement learning, we consider a Markov Decision Process with states $s_t \in \mathcal{S}$, actions $a_t \in \mathcal{A}$, dynamics $p(s_{t+1}|s_t, a_t)$, and reward r_t. The reinforcement learning agent learns a policy $\pi(a_t|s_t)$ to maximize the discounted return $R_t = \sum_{i=t}^{T} \gamma^(i-t)r_t$, where the horizon T may be infinite, and γ is the discount factor.

A variety of algorithms have been developed for this purpose. In many practical robotics settings including ours, collecting data on a test task is difficult but one can collect a prior dataset of transitions $\mathcal{D}$. To take advantage of data from other domains, we use techniques from offline reinforcement learning. Offline reinforcement learning methods address the problem of exploiting value estimates of out-of-distribution actions by adding either a regularization term or avoiding sampling of out-of-distribution actions. Of these, a subset are amenable to online finetuning. In this work, we base our method off of implicit Q-learning (IQL) Kostrikov et al., 2021b, which has shown strong performance in both offline reinforcement learning and finetuning.

10.4 Problem setting

In this paper, we focus on learning "on the job" when an offline learned RL policy does not solve a task fully. To do so, our problem setup reflects the problem of domain generalization. We assume access to a prior dataset of n_d domains drawn from a distribution of domains $p(\mathcal{D})$, with each domain $\mathcal{D}^d = \{\tau_i^d\}_{d=1}^{n_d}$ containing trajectories of transitions $\tau_i^d = \{(s_t, a_t, r_t, s_{t+1})^d\}$. These trajectories can be of arbitrary quality as in the offline RL literature. The challenge is to extract artifacts - in our case, a policy, value function, and reward model - from this data that successfully maximize reward in a test domain $\mathcal{D}_{test}$. Since data from the test domain is not available for offline reinforcement learning, we must assume some structural similarity between the training domain and test domain. As shown in the domain generalization and adaptation literature, different assumptions result in different algorithms.

In our case, we study the family of industrial insertion tasks. At a coarse level, these tasks are all variants of peg insertion. However, at a finer level, the structural similarity of the reward function and expert policy to solve these tasks is clustered into groups of connectors - for instance, NEMA connectors or Deutsch DT series as shown in figure 48. Within each group of connectors, there is similarity between the visual features required to align the connector and socket, the depth at which a success is achieved.

10.5 Method

The overall goal of our method is to be able to learn to insert previously unseen connectors. Because this requires domain generalization, we will first describe our representation learning objective that enables this for reward functions that generalize to new domains using this objective. Then we describe how we modify offline RL and online fine-tuning method with this representation that allows us to generalize policies and value functions to new scenes.

10.5.1 *Domain adversarial information bottleneck.*

First, to detect success in novel environments, we will need a reward function that generalizes to new connectors. To do so, we train a reward classification model on the collected dataset consisting of images x, domain label d, and binary reward labels r. In this section, we will describe how use the domain adversarial information bottleneck to learn an intermediate representation $z = g_\phi(x)$ to improve generalization of the reward model in this setting in the context of supervised learning.

The base objective function for the reward model $R(z)$ is binary cross entropy.

$$\mathcal{L}_{\text{reward}}(R; x, d, r) = -r \log R(z) - (1-r)\log(1-R(z)). \tag{51}$$

The representation should allow us to generalize to an unseen domain by capturing the common structure from previously seen domains. The most commonly proposed approach for generalizing to new domains is domain invariance. However, full domain invariance is not always what we want. In our setting, if our representation was completely domain invariant, it could not take advantage of the domain-specific information about certain connectors: when the reward is obtained for each connector visually, domain-specific cues that the policy or Q function can take advantage of, and so on.

Instead, we can consider a split representation $z = (z_I, z_S)$ consisting of a domain invariant representation z_I and a domain specific representation z_S. Can we design an objective to factorize the two representations, such that it improves generalization to new domains?

For enforcing domain invariance of z_I, we can take inspiration from domain adversarial neural networks (DANN) Ganin et al., 2016, which backpropagates the signal from a domain classifier into the representation. The domain classifier F_ψ is trained to minimize negative log likelihood:

$$\mathcal{L}_{\text{domain}}(F; x, d) = -\log F_\psi(z_I)_d. \tag{52}$$

For training the reward model, $\mathcal{L}_{adv} = -\mathcal{L}_{\text{domain}}(F; x, d)$ is added as an auxiliary objective, adversarially optimizing z_I to worsen the domain classifier likelihood.

The auxiliary loss imposes a cost for any domain specific information in Z_I, but as discussed earlier, allowing for some domain-specific information may be important for the classifier to perform well. But if we simply concatenate a representation z_S without a domain invariance loss, the features z_I degenerate to be completely uninformative when trained, leading to reduced performance as we show in our experiments. We need to somehow limit the information content of z_S, so that the domain invariant structure is still captured by z_I. A natural tool to accomplish this is the variational information bottleneck (VIB) Alemi et al., 2017. To learn a split representation, we constrain the information through z_S:

$$\mathcal{L} = \mathcal{L}_{\text{reward}}(z) + \mathcal{L}_{adv} \text{ s.t. } I(x; z_S) \leqslant C. \tag{53}$$

Following Alemi et al. Alemi et al., 2017, we can turn this into an unconstrained problem and compute the evidence lower bound:

$$\mathcal{L}_{\mathcal{R}} = \mathbb{E}_{\epsilon \sim p(\epsilon)} \mathcal{L}(Z) + \underbrace{\mathcal{L}_{adv} + \beta[KL(p(Z|X) \| p(Z))]}_{\mathcal{L}_{DAIB}(Z)} \tag{54}$$

In the following sections, we will describe how we utilize this objective in an offline RL and online finetuning framework.

10.5.2 *Offline reinforcement learning.*

Next, we will describe how we incorporate domain generalization into IQL Kostrikov et al., 2021b, an offline reinforcement learning algorithm. IQL learns a Q function and value function by quantile regression, then extracts a policy with advantage-weighted regression Peng et al., 2019a. We want each model: policy, Q-function, and value function, to be trained with a domain adversarial information bottleneck. Although this representation could be shared in principle, in practice we find significantly worse performance with shared representations.

IQL learns a Q function and value function by quantile regression, optimizing:

$$\mathcal{L}_Q = \mathbb{E}_{(s,a,s') \sim \mathcal{D}} \left[(r(s,g) + \gamma \bar{V}(s') - Q(s,a))^2 \right] \tag{55}$$

$$\mathcal{L}_V = \mathbb{E}_{(s,a) \sim \mathcal{D}} \left[L_2^{\tau}(\bar{Q}(s,a) - V(s))^2 \right] \tag{56}$$

. where $L_2^{\tau}(u) = |\tau - 1(u < 0)|u^2$, and $\bar{\cdot}$ indicates a stop gradient. A policy is extracted from the value function with advantage-weighted regression Peng et al., 2019a:

$$\mathcal{L}_{\pi} = \mathbb{E}_{(s,a) \sim \mathcal{D}} \left[\log \pi(a|s) \exp(\bar{A}(s,a)/\beta) \right] \tag{57}$$

where $A(s,a) = Q(s,a) - V(s)$.

We incorporate a domain adversarial information bottleneck to the policy to enable domain generalization. Let $Z^{\pi} = \phi^{\pi}(X)$ represent the output of a convolutional neural network representation of the input image. We optimize the following policy loss:

$$\mathcal{L}'_{\pi} = \mathcal{L}_{\pi} + \mathcal{L}_{DAIB}(Z). \tag{58}$$

We find that using DAIB for the policy bottleneck enables consistent performance across connectors, as explained further in section 10.7.2.

10.5.3 *Online finetuning*

When faced with a new connector, the offline RL trained policy may not generalize zero-shot. In these instances, we want to be able to actively finetune the policy to solve the task. We rely on the pretrained reward model and value function to do so, using online interaction in the test environment. In the online finetuning phase, we collect trajectories using the pretrained policy with additional exploration noise. In this phase, we do not assume a ground truth reward, as this requires accurate state estimation and localization. Instead, we evaluate the reward from the reward model. For a transition (s, a, s'), we compute the predicted reward $\hat{r} = R(F(s'))$ and append $(s, a, s', \hat{r})$ to the replay buffer. We then run the batch gradient descent updates of the policy, Q function, and value function according to equations 55, 56, and 58. In each batch we use 25% online data, with no data augmentations, and 75% offline data with data augmentations.

For this phase, both the policy and value function use a deterministic encoding in the information bottleneck, using the mean of the output distribution as the sampled value. We freeze the representation and do not train it further in the online finetuning phase.

10.6 Robot setup

Our robot setup for connector insertion is pictured in Fig 48. We use two 7 DoF Sawyer robots from Rethink Robotics for collecting data and running experiments. The robot is commanded with an end-effector controller, where the action a corresponds to relative end-effector movement in Cartesian coordinates. The resulting desired joint angles are computed via inverse kinematics, and then executed with a joint-space velocity controller with force limits for safety and to prevent dislodging connectors from a grasped position.

We initially collect a dataset of insertions of 50 connectors. For each connector, it is grasped by the robot and the socket is mounted to a clamp. Before data collection, the robot is manually placed in a successful grasp position and a calibration procedure captures the initial pose for detecting ground truth success. During data collection, the robot follows a manual insertion strategy of moving downwards if within 1mm of the center-line of the socket, and moving towards the center-line of the connector otherwise. Noisy actions are executed with probability 0.2 to visit a diverse set of states and induce recovery behavior in the prior data.

We want to utilize this data to allow the robot to generalize to a held-out connector not seen during training.

Connector	Localize	Straight Down	Random Search	Spiral Search	Ours (offline)	Ours (online)
DT12-1	3/20	1/20	7/20	4/20	7/20	19/20
Mega 1x1	3/20	1/20	1/20	15/20	5/20	16/20
NEMA 15	0/20	1/20	0/20	3/20	11/20	15/20
Europlug	6/20	4/20	6/20	4/20	10/20	15/20

Table 5: Insertion scenario I: the initial position of the gripper is located ±10mm from the socket centered around the socket.

10.7 EXPERIMENTS

We design our experiments to answer the following questions.

- Does the domain adversarial information bottleneck improve accuracy for reward classification?
- Can offline reinforcement learning followed by online fine-tuning be a solution for industrial insertion of novel connectors, and does it outperform a supervised localization model combined with a manually scripted control strategy?

10.7.1 *Reward Classification*

To learn a generalizable reward function and to evaluate the proposed domain generalization method, we first need to learn a reward classifier from the offline data. The accuracy of this reward classifier has a significant impact on whether online finetuning with the learned reward will successfully solve a task; if the reward is inaccurate, the policy can adversarially learn to go to regions of incorrect high reward. We train the reward classifier on a subset of the data consisting of 25 connectors and evaluated on a set of five held-out connectors. During training and evaluation, the data is rebalanced to be 50% positive and 50% negative samples per connector. In Table 8, we report the mean accuracy of each method averaged across the five held-out connectors.

We compare the following methods. Our method, **DAIB** uses a domain adversarial objective in combination with a variational information bottleneck as detailed in section 10.5.1. **DANN** enforces domain invariance using a domain adversarial neural network Ganin et al., 2016. **DAIB,** $\lambda = 0$ adds an additional network path from the input im-

Connector	Localize	Straight Down	Random Search	Spiral Search	Ours (offline)	Ours (online)
DT12-1	0/20	0/20	0/20	0/20	0/20	18/20
Mega 1x1	0/20	1/20	0/20	17/20	0/20	20/20
NEMA 15	0/20	0/20	0/20	0/20	6/20	17/20
Europlug	0/20	0/20	0/20	0/20	2/20	15/20

Table 6: Insertion scenario II: the initial position of the gripper is sampled from a box 10-20mm from the socket. This scenario is more difficult as moving straight down almost never solves the task. The baselines in this series of experiments use localization model initially, then follow the baseline strategy.

age to the reward classifier as in DAIB, but without enforcing an information bottleneck. **VIB** enforces a variational information bottleneck only without domain invariance Alemi et al., 2017. **ERM** has no auxiliary representation learning objectives.

The results are presented in Table 8. We see that our method, DAIB, is able to achieve an 88% accuracy, significantly higher than other methods. DANN, which enforces domain invariance, is the worst performing. This poor performance shows that the domain invariance assumption is too strong for the task of reward classification; connector insertion does require domain-specific information. The DANN + no VIB method achieves similar accuracy as ERM, since the network can just make the domain invariant features degenerate and bypass the domain adversarial objective. The VIB alone slightly improves performance over ERM because of a regularizing effect. But all methods achieve significantly worse accuracy than DAIB. Next, armed with an accurate reward model, we investigate finetuning connectors using the learned reward model.

Method	Acc.
DAIB (Ours)	**88%**
DANN	71%
DAIB, $\lambda = 0$	77%
VIB	79%
ERM	76%

Table 8: Comparison of test accuracy on reward classification.

10.7.2 *Self-Supervised Fine-tuning*

For insertion of a novel connector, we first run offline training of RL on a set of connectors excluding that connector or close variants (eg. same connector on a different robot), to obtain a policy, Q function, and value function. Then, we run online finetuning of RL

	Reward	Policy	Offline	Online
DT12-1	DAIB	DAIB	0/20	18/20
	DAIB	-	0/20	4/20
	-	-	0/20	0/20
Mega 1x1	DAIB	DAIB	0/20	20/20
	DAIB	-	3/20	20/20
	-	-	3/20	0/20
NEMA 15	DAIB	DAIB	6/20	17/20
	DAIB	-	0/20	2/20
	-	-	0/20	0/20
Europlug	DAIB	DAIB	2/20	15/20
	DAIB	-	1/20	12/20
	-	-	0/20	15/20

Table 7: Ablation of using the domain adversarial information bottleneck (DAIB) for the reward and for the policy across all four test connectors. The only consistent setting where finetuning occurs across all four connectors is using the DAIB for both reward and policy. Removing the bottleneck for the policy significantly reduces finetuning performance on two tasks, but still finetunes on the other two. Removing the bottleneck for the reward prevents finetuning except for the Europlug connector.

on the novel connector. We evaluate four connector insertion tasks: a Deutsch DT 12-way connector, a Megablock, a NEMA 15-5 connector, and a Europlug connector. We also evaluate in two settings: a relatively easier scenario where the initial location of the connector is centered around the socket with $\pm 10\text{mm}$ noise, and a harder scenario where the initial location of the connector is offset from the socket by $10 - 20\text{mm}$.

In the easier $\pm 10\text{mm}$ setting, we compare against the four following baselines. **Localize**: regress on to the state positions in the same offline data at each step and execute an expert policy towards that goal location which moves horizontally until within 1mm of the goal, then move straight down. **Straight down**: move straight down from the starting position. **Random search**: from the initial starting position, move straight down and then move to 5 randomly sampled positions while pressing down. **Spiral search**: from the initial starting position, move straight down and then move in a spiral while pressing down. Each method including ours is executed for 20 time steps per trajectory for evaluation.

The results are reported in Table 5. We see that the performance of most methods are inconsistent, but DAIB online successfully solves the task to $> 75\%$ success for all four connectors.

In the harder $10 - 20\text{mm}$ setting, we compare against similar baselines but straight down, random search, and spiral search from the initial position always fail because the initial position is too far away from the socket. Instead, for these three methods, we first execute the localization model and move to that goal position, then run the corresponding strategy.

The results for the harder insertion scenario is reported in Table 6. In this case, the performance of most methods, including DAIB offline, on most connectors, is poor. However, DAIB online is able to solve these tasks after 200 trials of finetuning. Importantly, even when the initial performance is 0/20 as in the DT 12-way connector, DAIB can improve because of well-shaped value functions guiding the policy towards the correct solution. In practice, we see that even when the initial policy is poor and does not get close to the socket, the finetuned policy quickly makes contact with the socket within a few trials. Further trials are required for finetuning to actually observe successes through stochastic exploration to perfect the policy.

10.8 DISCUSSION

We have discussed utilizing a novel domain generalization method, the domain adversarial information bottleneck (DAIB), for enabling fast online finetuning of RL policies.

We applied this method combined with IQL on a family of industrial insertion tasks and demonstrated how the system can learn on the job to finetune to test connectors with minimal human supervision. This scheme can generally be incorporated in many scenarios where robot datasets contain highly correlated data from related domains - a common scenario due to hardware and environment setup costs.

One main limitation of the method is its strong reliance on the reward model. If the reward model predicts false positives in a region, that can cause the finetuning process to wrongly cause the policy to visit that region, especially if the false positive occurs at a dynamically easier to reach state. Combining classifiers with reinforcement learning is prone to this kind of failure mode, as discussed previously in Fu et al., 2018. More careful application of domain adaptation methods that incorporate statistics of the online finetuning process might alleviate this issue.

10.9 CONTRIBUTION STATEMENT

This work is in preparation for submission, done in collaboration with Brian Zhu, Gokul Narayanan, Eugen Solowjow, and Sergey Levine. AN led the project, proposed the idea, and designed the experiments. The implementation and experiments were shared between AN and BZ. The writing was done by AN primarily. GN, ES, and SL advised and assisted with writing.

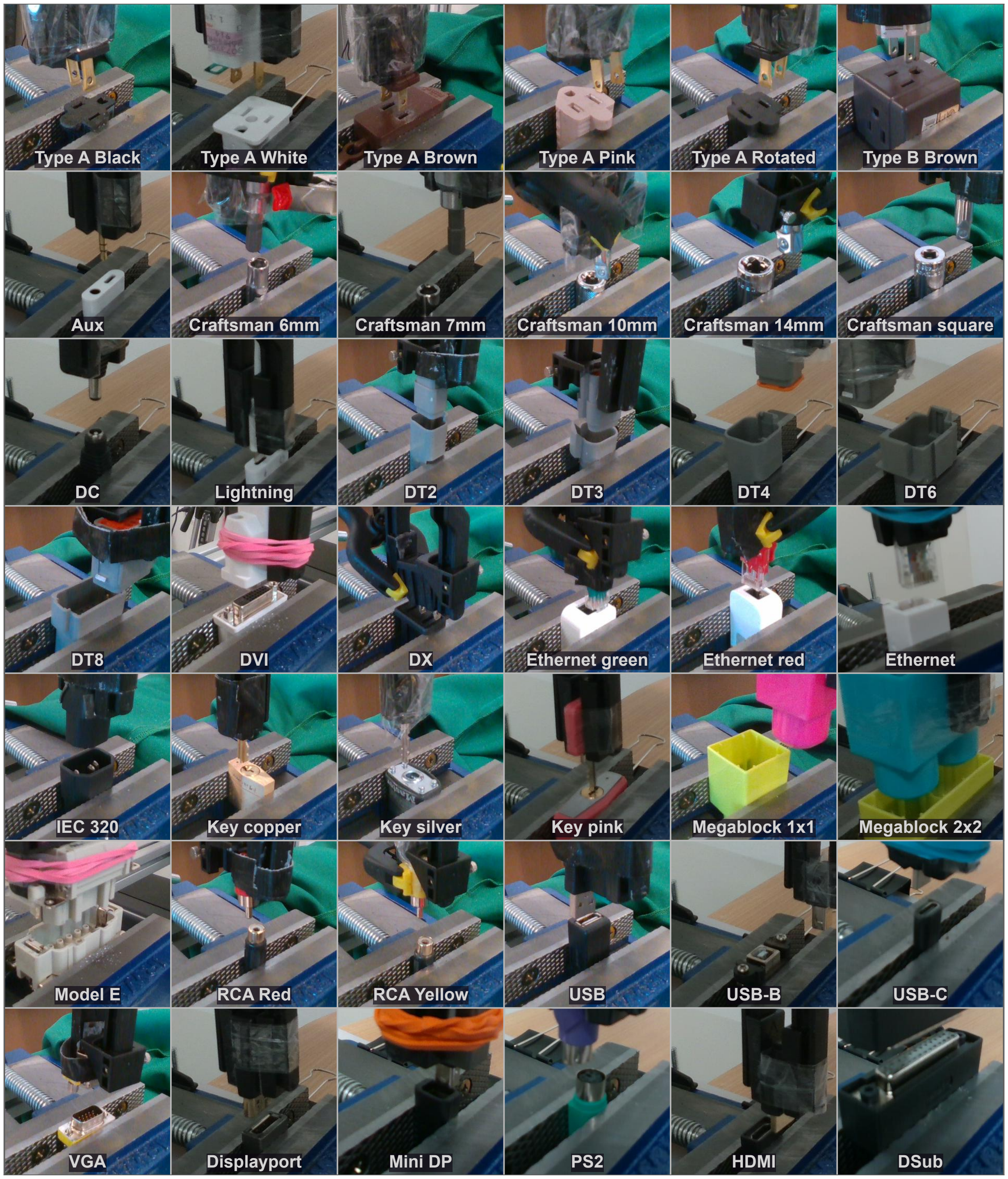


Figure 48: Frames from our dataset of 50 connectors.

Part III

AFFORDANCE LEARNING IN UNSEEN ENVIRONMENTS

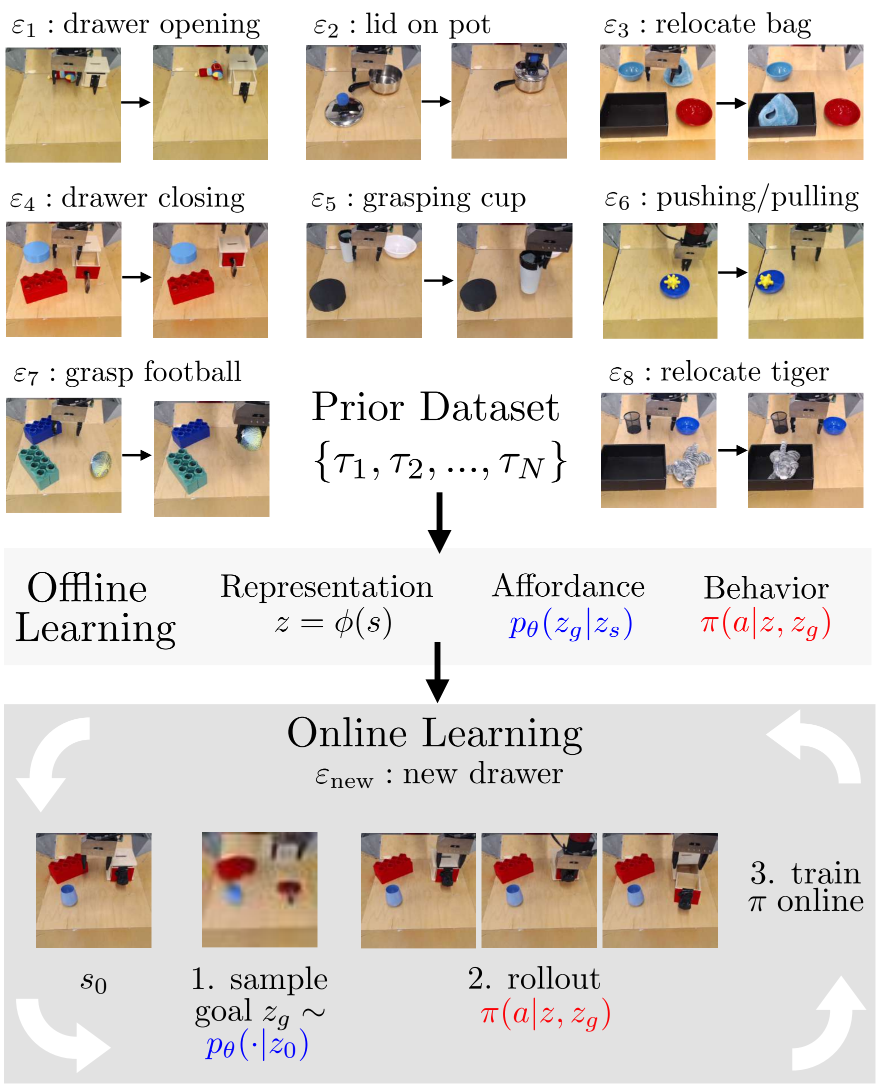
11

LEARNING NEW SKILLS BY IMAGINING VISUAL AFFORDANCES

11.1 INTRODUCTION

Suppose that you need to learn to open a new kind of drawer in a kitchen. While this new skill might demand some amount of trial and error, you would likely be able to use your mental model and past experience to *imagine* the drawer in the open position, and perhaps even imagine likely intermediate steps, such as grasping the handle, even if you do not yet know precisely how to perform the task. Borrowing the terminology put forward by Gibson (Gibson, 1979), the drawer presents the *affordance* of being "openable," and you are aware of this affordance from your past experience with other similar objects. In fact, infants learn about affordances such as movability, suckability, graspability, and digestibility through interaction and sensory feedback (Berger, 2014). Learning and utilizing affordances through interaction allows an agent to acquire diverse, meaningful experiences even in unfamiliar situations. However, this way of learning new skills differs markedly from the approach taken by most robotic learning algorithms: the most widely used exploration methods are generally *undirected*, and focus more on seeking out novelty and surprise (Houthooft et al., 2016; Tang et al., 2017; Pathak et al., 2017), rather than familiar and previously seen outcomes. In this chapter, we study how robots operating entirely from pixel input can learn about affordances and, when faced with a new and unfamiliar environment, can utilize a previously trained model of possible outcomes to propose potential *goals* that they can practice in this new environment, so as to explore and update their policy efficiently.

We study affordance learning through the framework of self-supervised goal-conditioned reinforcement learning (RL). Learning a new skill in this framework requires generalization in terms of goal setting (*affordances*) and generalization in terms of goal reaching (*behavior*). Prior methods for goal-conditioned RL learn a policy to reach a goal state without the need for an externally provided reward function, and are able to mas-



$\mathcal{E}_1$: drawer opening
$\mathcal{E}_2$: lid on pot
$\mathcal{E}_3$: relocate bag
$\mathcal{E}_4$: drawer closing
$\mathcal{E}_5$: grasping cup
$\mathcal{E}_6$: pushing/pulling
$\mathcal{E}_7$: grasp football
Prior Dataset $\{\tau_1, \tau_2, ..., \tau_N\}$
$\mathcal{E}_8$: relocate tiger
Offline Learning
Representation $z = \phi(s)$
Affordance $p_\theta(z_g|z_s)$
Behavior $\pi(a|z, z_g)$
Online Learning
$\mathcal{E}_{\text{new}}$: new drawer
s_0
1. sample goal $z_g \sim p_\theta(\cdot|z_0)$
2. rollout $\pi(a|z, z_g)$
3. train π online

Figure 49: We propose a system for efficient self-supervised robot learning in an unseen environment $\mathcal{E}_{\text{new}}$ by utilizing prior data $\mathcal{D}$ of trajectories from related similar environments $\mathcal{E}_i \sim p(\mathcal{E})$. Tasks are specified via a target goal image. From prior data, we learn an encoder $\mathbf{z} = \phi(\mathbf{s})$ of images (representation) for compressing observations and self-generating rewards, a model of what tasks might be tested in the new environment (affordance), and goal-conditioned policy to accomplish a given task (behavior). While this provides reasonable performance in some test environments, perfecting the policy in test environments may require additional interaction in the test environment. Dropped in a test environment $\mathcal{E}_{\text{new}}$ without a given goal, we run RL online in order to practice potential tasks, with goals sampled from the affordance model. Online behavior learning allows us to improve the policy π for $\mathcal{E}_{\text{new}}$ even when it contains new and unseen objects.

ter skills such as pushing and grasping objects from image observations (Agrawal et al., 2016; A. Nair et al., 2018b; Lynch et al., 2019; A. Nair et al., 2019a). They learn skills by setting goals to explore, and learning a policy to reach them. However, while these prior works have studied how to learn goal-conditioned policies in individual environments, they do not consider what happens when the robot enters a *new* environment where the policy does not simply generalize zero-shot and needs to be trained further. Our approach to solving tasks in this setting is to learn affordances, represented by a generative model of possible outcomes, and then sample possible affordances in a new environment to explore the new environment efficiently.

Our method, visuomotor affordance learning (VAL), uses expressive conditional models for learning generalizable affordances, along with off-policy RL to learn generalizable behaviors. First, we propose to use more expressive generative models for RL to learn compressed representations of images that can reconstruct unseen objects and help the policy generalize to them. Second, we propose to learn an expressive conditional generative model that generalizes past data to set meaningful goals for novel environments, enabling an understanding of object affordances. Third, we demonstrate how we can use off-policy RL with all prior data to initialize a general-purpose goal-conditioned policy, then fine-tune the policy with additional data to master a skill in a new environment. Combining RL and representation learning, we show that we are able to learn skills with a small amount of online exploration on novel objects in real-world robotic scenarios such as object grasping, relocation, and drawer opening and closing.

The main contribution of this work is to present a learning system that can learn robotic skills entirely from image pixels in novel environments by utilizing prior data to generate goals and generalize behavior in new settings. We demonstrate that our method can learn complex manipulation skills including grasping, drawer opening, pushing, and object re-positioning for a diverse set of objects in simulation. We also demonstrate our method in the real world on a Sawyer robot, where our method is able to learn tasks such as grasping and placing unseen objects and opening and closing unseen drawers after only five minutes of online interaction.

11.2 RELATED WORK

RL has been applied previously to robotic manipulation (Kober and Peter, 2008; Peters et al., 2010; Levine et al., 2016a), and also various other applications from playing games (Mnih et al., 2013; D. Silver et al., 2016a) to locomotion (Benbrahim and Franklin, 1997; Kohl and Stone, 2004; Marc Peter Deisenroth and Rasmussen, 2011; G. Williams et al., 2017). Such approaches require an external reward function, but obtaining this reward function itself poses a challenge to exploring in novel uninstrumented environments as we consider in this paper. Thus, in this work we focus on self-supervised RL methods that that do not assume externally provided reward functions.

When learning without an externally provided reward function, one common idea is to use novelty-based intrinsic reward functions (Chentanez et al., 2005; Lopes et al., 2012; M. Bellemare et al., 2016; Houthooft et al., 2016; Stadie et al., 2016; Pathak et al., 2017). State novelty-based methods eventually visit all possible states, but do not necessarily learn a useful policy from purely optimizing the exploration objective. An alternative exploration framework that learns a useful policy even solely from the intrinsic objective is goal reaching (L P Kaelbling, 1993; Schaul et al., 2015b; Adrien Baranes and Pierre-Yves Oudeyer, 2012b; Andrychowicz et al., 2017b; A. Nair et al., 2018b; Nachum et al., 2018; Held et al., 2018; Pr et al., 2018c; Warde-Farley et al., 2019; Vitchyr H Pong et al., 2020a): by picking a distance measure between states, setting goals, and attempting to reach them, an agent can discover all potential goals in its environment. We refer the reader to the survey of Colas et al. (Colas et al., 2021) for a full classification and discussion of these methods. However, these prior methods do not study the question of how to set goals in new environments, which is vital for collecting coherent experience when faced with a new task. Our method utilizes representation learning and off-policy RL to generalize prior experience to set exploration goals in new settings.

Another line of work explores the use of affordances in RL, robotics, and control, historically through the lens of perception (Zech et al., 2017; Hassanin et al., 2018; Yamanobe et al., 2018). Affordances have also been discussed previously in reinforcement learning in order to accelerate planning in model-based RL by planning over only a subset of relevant actions (Abel et al., 2014; Khetarpal et al., 2020; Xu et al., 2021). Our work is more related to the view of affordances in developmental robotics, where affordances were hypothesized to be useful for learning general manipulation skills (Hart and Grupen, 2010; Min et al., 2016). In our work, we show how goal-conditioned RL utilizing from prior data can put these ideas into practice on real-world robotics systems.

Most similar to our work is context-conditioned reinforcement learning with imagined goals (CCRIG), which learns a conditional variational auto-encoder (CVAE) (Sohn et al., 2015) that generates goals conditional on the current scene (A. Nair et al., 2019a), as covered in chapter 3. CCRIG was able to learn pushing skills that generalized mainly to object color and partially to object geometry. Our work extends CCRIG in a number of ways. First, we learn expressive generative models that are able to generate goals in scenes with significantly more visual diversity. Second, we learn a diverse set of skills (e.g., grasping, drawer opening, object placing) that require the goal generation to understand affordances of the environment. Finally, we show that we can use off-policy RL on prior experience, in addition to fine-tuning further on a single specific task to learn new skills. These differences allow our method to better operate in real-world scenarios, as borne out in our experiments.

11.3 PRELIMINARIES

In this section, we cover preliminaries on RL, goal-conditioned RL, and self-supervised visual RL.

Goal-conditioned reinforcement learning. In goal-conditioned RL, we augment the standard Markov decision process (MDP), which is defined in terms of states $\mathbf{s}_t \in \mathcal{S}$, actions $\mathbf{a}_t \in \mathcal{A}$, and environment dynamics $\mathbf{s}_{t+1} \sim p(\cdot|\mathbf{s}_t, \mathbf{a}_t)$, with goals $\mathbf{g} \in \mathcal{G}$ that represent the agent's intention to perform one of a variety of tasks drawn from the task family $p(\mathbf{g})$. The reward function is also goal-conditioned, and given by some function $r(\mathbf{s}, \mathbf{g})$. The discounted return is defined as $R_t = \sum_{i=0}^{H} \gamma^i r(\mathbf{s}_i, \mathbf{g})$, where γ is a discount factor and H is the horizon, which may be infinite. The aim of the agent is to optimize a policy $\pi(\mathbf{a}_t|\mathbf{s}_t, \mathbf{g})$ to maximize the expected discounted return $J(\pi) = \mathbb{E}_{\mathbf{g}}[R_0]$. Efficient off-policy RL algorithms have been proposed to learn goal-conditioned policies (Schaul et al., 2015b; Andrychowicz et al., 2017b).

Self-supervised visual reinforcement learning. For scalable robot learning, we cannot always assume known shaped reward functions for tasks. In the absence of such reward functions, Andrychowicz et al. propose goal-state reaching as a natural objective (Andrychowicz et al., 2017b): tasks are defined by state outcomes, where the goal space $\mathcal{G} = \mathcal{S}$, the state space, and the reward is a goal-reaching objective $r(\mathbf{s}, \mathbf{g}) = -\mathbb{1}_{||\mathbf{s}-\mathbf{g}||>\epsilon}$. The task distribution $p(\mathbf{g})$ is chosen to be the feasible states of the robot and objects it interacts with.

But when states are high-dimensional (e.g., images), two issues arise: we do not know the task distribution $p(\mathbf{g})$, and exact reaching of a target goal state is impractical. Reinforcement learning with imagined goals (RIG) (A. Nair et al., 2018b) addresses these issues with a generative model, which is used to learn a latent space of observations and similarity metric on images. Specifically, a variational auto-encoder (D. P. Kingma and Welling, 2014) with encoder $\phi(\mathbf{z}_t|\mathbf{s}_t)$ and prior $p(\mathbf{z})$ is learned. At training time, the robot sets goals for itself in latent space by sampling a goal latent $\mathbf{z}_g \sim p(\mathbf{z})$ and learns a policy $\pi(\mathbf{z}_t, \mathbf{z}_g)$ to reach latent goals. At test time, the robot can be tasked with a goal image $\mathbf{g}$ and execute the learned policy with goal latent $\mathbf{z}_g \sim \phi(\cdot|\mathbf{g})$ to match the goal image. In this way, goal-conditioned RL with generative models enables self-supervised learning in a single environment $\mathcal{E}$ where the task distribution $p(\mathbf{g})$ is not known apriori.

11.4 PROBLEM SETTING

We now consider acting in a distribution of environments $p(\mathcal{E})$ with shared structure.[1] Each environment $\mathcal{E}_i$ has its own task distribution $p_i(\mathbf{g})$. As before, tasks are defined by goal states, so $p_i(\mathbf{g})$ represents the potential outcomes of interest in that environment. We assume that the outcomes of interest depends only on the appearance of the environment. When $\mathcal{E}$ is fully observed, this means $p(\mathbf{g}|s_0)$ is shared across environments. As prior data, the robot has access to a training dataset $\mathcal{D} = \{\tau_1, \tau_2, \dots \tau_N\}$ of trajectories from prior environments, where the trajectories achieve a final outcome $\mathbf{s}_T \sim p_i(\mathbf{g})$ – that is, they succeed on tasks from the underlying task distribution for that environment. Now, the robot is placed in a new environment $\mathcal{E}_{new} \sim p(\mathcal{E})$, and must learn to solve tasks in the new environment through self-supervised practice at training time, such that it can accomplish tasks sampled from $p_i(\mathbf{g})$ at test time. The environments we consider vary visually and dynamically in terms of the objects that are present and potential tasks they afford, such as being able to lift various objects and open different drawers; such real-world variation is presented in Figure 49.

Thus, the agent in this new setting must generalize its prior experience $\mathcal{D}$ to practice potential skills it may be asked to perform at test time in the new environment efficiently, even when it encounters novel objects. At test time, the robot is evaluated in terms of its ability to accomplish a task in $\mathcal{E}_{new}$ specified by a goal image. To perform well in this setting, a method must attempt to infer $p_{new}(\mathbf{g})$, which should be possible since $p(\mathbf{g}|s_0)$ is common across environments and $\mathcal{D}$ contains trajectories that achieve goals from the same distribution. Note that given observations from $\mathcal{E}_{new}$, the agent may still have to practice the various behaviors the environment affords if there are multiple, since the test task distribution is unknown at training time.

1 Meta-learning approaches have also studied learning a new task quickly, given experience on a set of related MDPs (Duan et al., 2016; Finn et al., 2017; Rakelly et al., 2019). The aims of our method are related to meta-learning, in that we also aim to learn in new environments more quickly, but we do not assume being given user-specified tasks or rewards to solve in the new environment.

Algorithm 7 Visual Affordance Learning

```
Require: Dataset $\mathcal{D}$, policy $\pi(\mathbf{a}|\mathbf{z},\mathbf{z}_g)$, Q-function $Q(\mathbf{z},\mathbf{a},\mathbf{z}_g)$, RL algorithm $\mathcal{A}$, replay
    buffer $\mathcal{R}$, relabeling strategy $p_{RS}(\mathbf{z})$, environment family $p(\mathcal{E})$.
 1: Learn encoder $\phi(\mathbf{z}|\mathbf{s})$ by generative model of $\mathcal{D}$
 2: Learn affordances $p(\mathbf{z}_t|\mathbf{z}_0)$ by generative model of $\mathcal{D}$
 3: Add latent encoding of $\mathcal{D}$ to the replay buffer
 4: Initialize $\pi$ and Q by running $\mathcal{A}$ offline
 5: Sample $\mathcal{E}_{new} \sim p(\mathcal{E})$, $\mathcal{E}_{new} = (p_{new}(\mathbf{s}_0), \pi_{new}(\mathbf{s}_{t+1}|\mathbf{s},\mathbf{a}))$
 6: for $1, \ldots, N_{episodes}$ do
 7:     Sample initial state $\mathbf{s}_0 \sim p_{new}(\mathbf{s}_0)$.
 8:     Sample goal $\mathbf{z}_g \sim p(\mathbf{z}_t|\mathbf{z}_0)$
 9:     for $t = 0, \ldots, H$ do
10:         Sample $\mathbf{a}_t \sim \pi(\cdot|\mathbf{z}_t, \mathbf{z}_g)$
11:         Sample $\mathbf{s}_{t+1} \sim p_{new}(\cdot|\mathbf{s}_t, \mathbf{a}_t)$
12:     end for
13:     Store trajectory $(\mathbf{z}_1, \mathbf{a}_1, \ldots, \mathbf{z}_H)$ in replay buffer $\mathcal{R}$.
14:     for $1, \ldots, N_{train\_steps}$ do
15:         Sample transition $(\mathbf{z}_t, \mathbf{a}_t, \mathbf{z}_{t+1}, \mathbf{z}_g)$
16:         Relabel with $\mathbf{z}'_g \sim p_{RS}(\mathbf{z}_g)$ and recompute reward
17:         Update $\pi$ and Q with relabeled transition using $\mathcal{A}$
18:     end for
19: end for=0
```

11.5 VISUOMOTOR AFFORDANCE LEARNING

In this section, we present visuomotor affordance learning (VAL), our method for self-supervised learning in novel environments utilizing prior data from related environments. VAL consists of three learning phases: (A) an affordance learning phase to learn affordances from the prior data, (B) an offline behavior learning phase to learn behaviors from the prior data, and (C) an online behavior learning phase where the agent actively interacts with the test environment using affordances and learns potential behaviors in the new environment. The overall method is summarized in Algorithm 7.

11.5.1 *Affordance Learning*

The affordance learning system must generate goals that induce coherent exploration trajectories even in new environments. We would like to learn a model that, given an observation s_0 in a new environment, generates a potential goal state the agent might be tasked to reach. With high-dimensional image observations, attempting to sample such goal states directly is a difficult generative modeling problem. Instead, we first learn a lower-dimensional latent space $p(\mathbf{z}_t|\mathbf{s}_t)$ by training a generative model through image reconstruction. With sufficiently expressive models and enough data, the generative model represents images in a manner that it can reconstruct even unseen objects. Given such a latent space, we can then learn affordances by training a conditional model $p(\mathbf{z}_t|\mathbf{z}_0)$ to generate goals that are plausible outcomes of an initial state, even for unseen environments.

To instantiate these two models, we must choose a class of latent variable generative models for $p(\mathbf{s}_t|\mathbf{z}_t)$ and conditional affordance models $p(\mathbf{z}_t|\mathbf{z}_0)$. For the first, while many choices would be suitable, including models such as variational auto-encoders (VAEs) (D. P. Kingma and Welling, 2014) and generative adversarial networks (GANs) (Goodfellow et al., 2014; Donahue et al., 2017), in our implementation we use the VQVAE model (Aaron van den Oord et al., 2017). The VQVAE is expressive enough to represent very diverse datasets, and be able to reconstruct even unseen objects with a high level of detail. We do not require image reconstructions for our method, but the ability to partially reconstruct unseen objects suggests that model expressively represents geometry and color information that may be important for learning affordances and behaviors. In the VQVAE case, ϕ is deterministic, so we will use the shorthand for the latent embedding $\mathbf{z}_t = \phi(\mathbf{s}_t)$, where $\mathbf{z}_t$ is the continuous latent resulting after quantization. In our experiments, we compare this choice of model to other expressive models: VAE (D. P. Kingma and Welling, 2014), CVAE (Sohn et al., 2015), and BiGAN (Donahue et al., 2017).

Next, given a latent space, we need to sample potential goals from this space that is predictive of which goals might be tasked at test time in the new environment $\mathcal{E}_{\text{new}}$. We use a conditional PixelCNN model (Aaron van den Oord et al., 2016) in the latent space to do so, conditioned on the initial state s_0. The PixelCNN model is trained to maximize $\mathbb{E}_{\tau\sim\mathcal{D},(\mathbf{s}_0,\mathbf{s}_t)\sim\tau,z\sim\phi(\cdot|\mathbf{s})}[\log p_\theta(\mathbf{z}_t|\mathbf{z}_0)]$, where θ is the parameters of the PixelCNN density model. To generate exploration goals, we sample $z_g \sim p_\theta(\cdot|\mathbf{z}_0)$, where $\mathbf{z}_0 = \phi(\mathbf{s}_0)$ is the encoding of an image of the current setting. The PixelCNN model in VAL being conditional allows us to generate meaningful goals that might be achievable with a novel object. As we show in Section 11.6.1, the conditional model allows us to sample affordances such as opening drawers and lifting objects, even for new environments that are visually complex with variation in the identity, color, geometry, and functionality of objects.

11.5.2 *Offline Behavior Learning*

Given learned affordances of what behaviors the agent may be tasked with, the agent needs to learn how to actually accomplish those behaviors. In this phase, we wish to learn a reasonable goal-conditioned policy with offline RL. While trained on a limited fixed offline dataset, the hope is that the policy can generalize learning from offline data to accomplish desired behaviors in a new environment, allowing us to either perform tasks successfully zero-shot without further learning, or collect meaningful exploration data in the next phase (Section 11.5.3).

To learn with offline RL while allowing the possibility of quickly fine-tuning in a new environment, we use advantage weighted actor critic (AWAC) (A. Nair et al., 2020) as the underlying RL algorithm. AWAC is an off-policy RL algorithm that has shown strong performance in utilizing prior data for offline pretraining while still being amenable to online fine-tuning. The aim of behavior learning is to optimize a goal-conditioned policy $\pi(\mathbf{a}|\mathbf{z}, \mathbf{z}_g)$ which is able to solve any task in the task distribution $p(\mathbf{z}_g)$. We do not assume an external reward function, so we require a reward function to optimize. Following prior work (Andrychowicz et al., 2017b; A. Nair et al., 2018b), we optimize a goal reaching objective: to maximize the density $p_{\mathbf{Z}_g}(\mathbf{z} = \mathbf{z}_g)$; in practice we use the sparse reward function $r(\mathbf{z}, \mathbf{z}_g) = -\mathbb{1}_{\|\mathbf{z}-\mathbf{z}_g\|>\epsilon}$, where ϵ is a fixed threshold as it encourages fully solving tasks in a binary fashion. For a particular transition in the replay buffer $(\mathbf{z}_t, \mathbf{a}_t, \mathbf{z}_{t+1}, \mathbf{z}_g, r)$, we can also relabel the goal with a new goal $\mathbf{z}'_g$ and the recompute the reward. In practice we keep $\mathbf{z}_g$ with 20% probability, future hindsight experience replay with 40% probability, and sample $\mathbf{z}'_g \sim p(z_t|z_0)$ with 40% probability. Importantly, due to using a compressed representation, we can expect that z_t in any new environment will be semantically similar to past experience. Thus, in this phase, we can obtain a policy that generalizes partially to a new environment.

11.5.3 *Online Behavior Learning*

Guaranteeing zero-shot generalization for every possible new environment is in general impossible. Instead, we will discuss how to finetune in a specific environment $\mathcal{E}_{\text{new}}$ using affordances. In this phase, we utilize the learned affordances which inform *what* tasks to perform, and the offline learned behaviors that inform *how* to perform those tasks.

At online training time, the new task distribution $p_{\text{new}}(\mathbf{g})$ is unknown, so we use the affordance model to sample potential tasks. Thus, to collect coherent exploration data, we sample goals from the affordance module $z_g \sim p_\theta(\cdot|z_0)$ and roll out the goal-

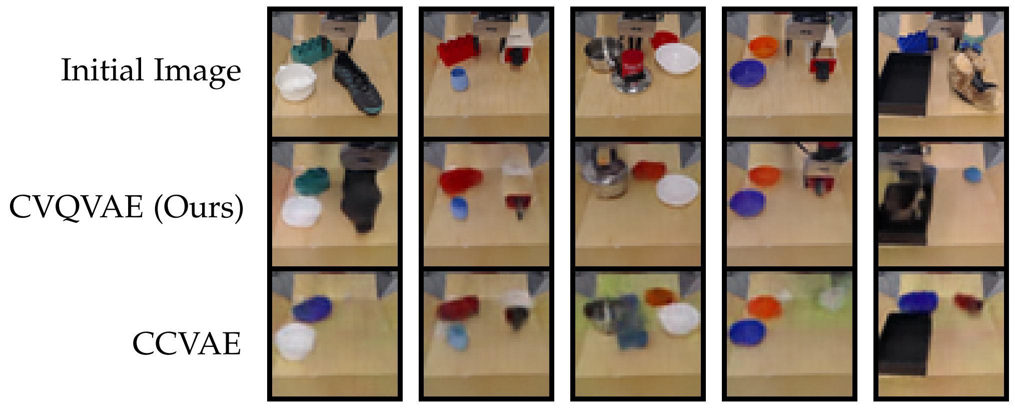


Figure 50: Samples on unseen objects in the real world. In each column, the top image is the conditioning image s_0 and the images below are conditionally sampled images from the corresponding generative model. Our model, CVQVAE, generates clear and diverse samples.

conditioned policy $\pi(\mathbf{a}|\mathbf{z}, \mathbf{z}_g)$. We then iterate between improving the policy with off-policy RL, and collecting exploration data and appending it to the replay buffer. To learn a new task, exploration data in the new environment for fine-tuning the policy is extremely valuable, so this iteration allows us to quickly fine-tune. The online learning process is illustrated in the bottom box in Figure 49.

In summary, VAL enables self-supervised learning in novel environments utilizing prior data. We first learn a generative model for learning a latent space and affordances, and learn how to accomplish tasks with off-policy RL. Then, both are used in an online behavior learning phase to perfect potential behaviors in a new environment. The overall method is summarized in Algorithm 7.

11.6 REAL-WORLD EXPERIMENTAL EVALUATION

We first evaluate our method in a real-world manipulation setting with a Sawyer robot and diverse objects and tasks. This setting is very challenging due to the variety of objects, scenes, and tasks that the robot interacts with, and being limited to using image inputs.

Real-world setting. The real-world setting is shown in Figure 49. A Sawyer robot is controlled at 5Hz with 4 degrees of control: 3 dimensions of end-effector velocity control in Euclidean space and one dimension to open and close the gripper. The environments span 10 drawer handles, 10 pot handles, 40 toys, and 60 distractor objects. To create a new environment $\mathcal{E}$ in the real world, we randomly sample one interaction object as well as 2-

Task	VAL (Ours) Offline → Online	CCRIG Offline → Online
(1) Pickup shoe	12.5% → **50%**	0% → 16.6%
(2) Drawer closing	25% → **100%**	0% → 12.5%
(3) Drawer opening	62.5% → **100%**	0% → 0%
(4) Place object in tray	25% → **75%**	0% → 0%
(5) Lid on pot	37.5% → **87.5%**	0% → 0%

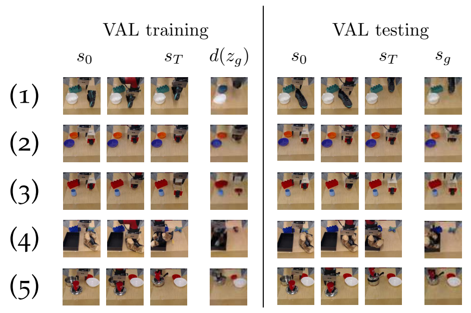


Figure 51: Real-world results. Left, success rates per method for the five tasks tested. We report the offline performance, followed by the performance after five minutes of online fine-tuning. With VAL, we see reasonable initial offline performance followed by significant improvement on all tasks. Meanwhile CCRIG fails to succeed any of the tasks. Right, film strips of VAL during training (left, with decoded affordance proposals) and testing (right, with goal images) are visualized. Videos are available at https://sites.google.com/view/val-rl

3 distractor objects. The behaviors that the environments afford therefore span grasping and relocating toys, opening and closing drawers, as well as covering and uncovering pots. Each test environment contains an unseen interaction object and a random set of 2-3 distractor objects, with all positions randomized.

Our experiments aim to answer the following questions:

1. Does the learned generative model sample plausible affordances in new scenes?
2. Is VAL able to accelerate learning of real-world manipulation skills online in new settings?

11.6.1 *Generative Models for Affordance Learning*

Expressive generative models enable us to propose desired outcomes in new environments even when the current policy cannot yet achieve them. We can preview this capability by inspecting the training procedure of the models, which also gives insights into which models will tend to perform well when used for interactive learning. Specifically, we inspect model samples to see if the model can actually output plausible candidate affordances. We evaluate our model, which combines a VQVAE with a conditional pixel CNN, and prior models that have been used in self-supervised RL as well as other expressive generative models. We compare (1) VAE (D. P. Kingma and Welling, 2014), (2) CVAE (Sohn et al., 2015), (3) BiGAN (Donahue et al., 2017), (4) Conditional BiGAN, (5) Ours, a VQVAE with a conditional PixelCNN.

Sampled affordances are shown in Figure 50. Our model is able to sample coherent tasks to perform where other methods produce indistinct or uninterruptible samples. For example, the VQVAE model turns an unseen drawer (with a wide handle) into one with a thin handle, which exists in the prior data. This illustrates its ability to utilize prior data for generating conducive goals for online RL. CCVAE samples are usually less coherent, often missing the object completely and not capturing the geometry of unseen objects. The CBiGAN also struggles to produce realistic images, while the VAE fails completely as it is not a conditional model cannot represent the wide diversity of potential images.

11.6.2 *Real-World Visuomotor Affordance Learning*

Next, we investigate whether VAL can handle real-world visual complexity and object diversity to learn control policies for varied tasks on a Sawyer robot. The setup is shown in Figure 49: the robot is tasked with fine-tuning in a particular environment, beginning with about 1,000 trajectories of prior data for affordance learning and offline behavior learning. The protocol for collecting prior data, as well as examples of images from the data are shown in Appendix F.0.1.

After pretraining affordances, a policy, and a Q function on the prior data, the robot is dropped in a new test environment. We evaluate five test environments which each contain distractor objects as well as objects that may be interacted with such a shoe that can be picked up, drawers than can be opened or closed, and a lid which may be placed on a pot. These behaviors are demonstrated on similar objects in the prior dataset, but the objects during test time are previously unseen - for instance, the drawer has a different handle. The agent can interact with the environment without supervision to collect more data and improve the policy; then to evaluate, the agent is tasked with a specific goal image that corresponds to a task such as opening a closed drawer.

Results running VAL in the real world are shown in the table in Figure 51, reporting the success rates on five test tasks for our method and CCRIG both offline and after one epoch of online training, which includes 10 interactive trials, amounting to less than five minutes of real-world interaction time. On all five tasks, VAL shows nontrivial offline performance followed by strong online improvement. Qualitatively, our method is able to learn behaviors such as recovering from a missed grasp by returning to the grasp. Film strips of our method are shown on the right side of Figure 51. In contrast, CCRIG struggles to make progress on all five tasks. CCRIG fails for two reasons. First,

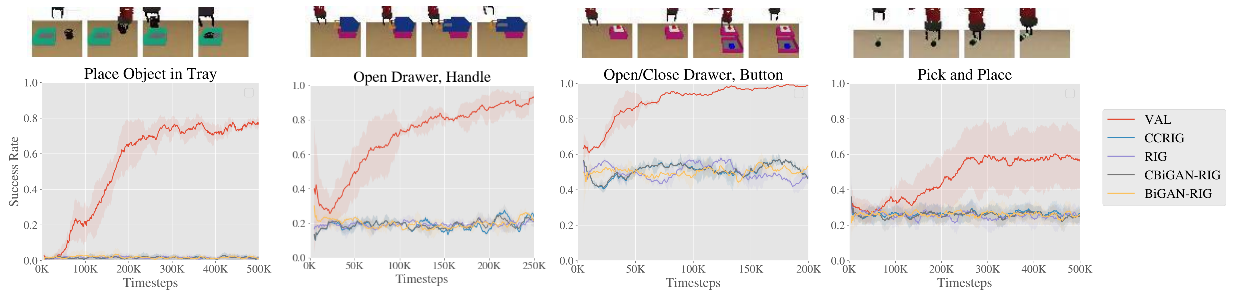


Figure 52: Learning curves for simulation experiments, fine-tuning on an unseen environment. Our method is able to learn these tasks online, while none of the baselines or prior methods are able to make meaningful learning progress in this setting. A successful rollout of each task in a test environment is shown above the corresponding plots.

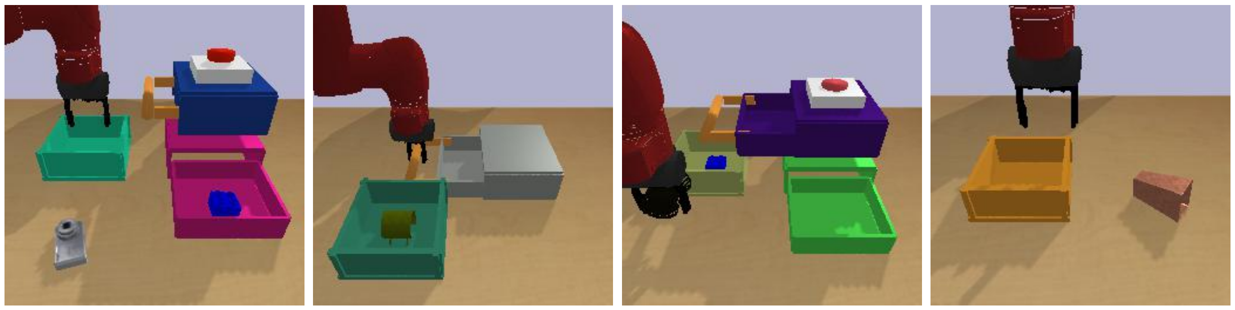


Figure 53: Randomly sampled scenes from our simulated multi-task environment. To practice in a sampled scene, the agent must infer the potential behaviors that the scene affords.

the quality of the sampled affordances are significantly poorer. Second, as can be seen in videos, CCRIG sometimes comes close to solving the task, but does not fully solve it, preventing improvement after subsequent training.

11.7 EXPERIMENTAL EVALUATION IN SIMULATION

In order to further study and understand VAL, we carry out more experiments in simulation, where we can better control the quantity of data and ablate parts of the method. These experiments aim to answer the following questions:

1. Does VAL outperform prior self-supervised RL methods in accelerating learning a new task from prior data?
2. Can VAL scale with additional data for affordance and behavior learning?

Simulated setting. Randomly generated workspaces in our simulated multi-task environment are shown in Figure 55. In this PyBullet simulation (Coumans and Bai, n.d.),

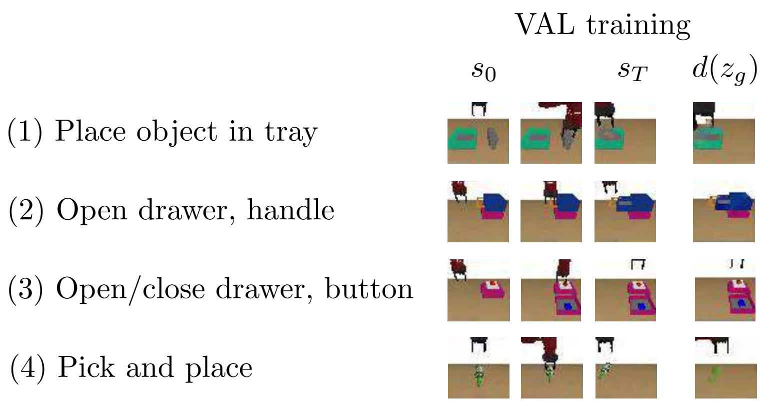


Figure 54: Training rollouts from VAL. For each environment, the reconstruction of a sampled affordance is shown on the rightmost column, and frames from a trajectory attempting to achieve that affordance is shown on the left.

the robot is faced with multiple potential tasks in each environment based on which objects are present: opening and closing a drawer by the handle, opening and closing a different drawer by pressing a button, grasping objects, and moving objects into drawers or a box. To sample a new environment $\mathcal{E}$, we randomize the existence, position, color, and orientation of the following: two drawers, a box, a button, and an object. If an object is present it is chosen from a set of 84 object geometries. To successfully learn in a test environment, the method must be able to explore in the environment based on the behaviors that environment affords.

11.7.1 *Self-Supervised Online Fine-Tuning from Prior Data*

We first compare VAL against prior methods in this simulated setting. The robot receives a prior dataset $\mathcal{D}$ of trajectories in the pre-training environments, which is utilized for learning affordances and offline RL. The details of this dataset are explained further in Appendix F.0.2. After the pre-training phase, the robot is placed in a test setting and begins online fine-tuning. We evaluate the policy on goal images in the new environment, sampled from expert trajectories, and report whether the final state of the policy's trajectory matches the state of the goal image within a chosen threshold.

Learning curves of the online training phase for various objects are shown in Figure 52. We see that for each task, VAL outperforms prior methods which do not make progress on learning these tasks at all. On the drawer opening task, offline RL achieves nontrivial performance of about 60%, but online interaction allows fine-tuning to over 90% success rate.

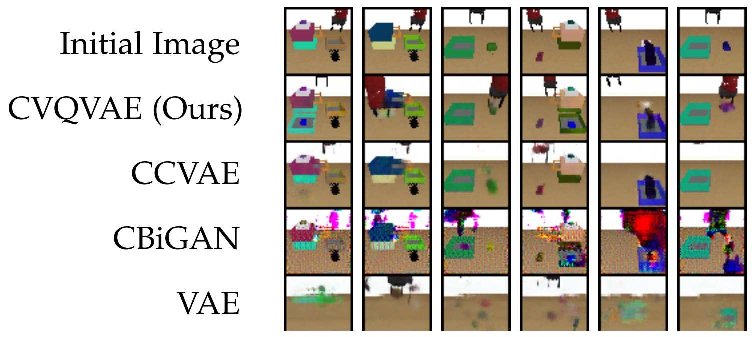


Figure 55: Samples on test environments in simulation. In each column, the top image is the conditioning image s_0 and the images below are conditionally sampled images. The left four columns are relatively successful samples for our affordance model, each showing the potential outcome of a behavior. The right two columns are failure modes, lacking diversity or altering an object's geometry or color.

How is VAL able to outperform prior methods so significantly? One major reason is the quality of sampled goals. Samples for the affordance model for the simulated domain are shown in Figure 55. We compare (1) VAE (D. P. Kingma and Welling, 2014), (2) CVAE (Sohn et al., 2015), (3) BiGAN (Donahue et al., 2017), (4) Conditional BiGAN, (5) Ours, a VQVAE with a conditional PixelCNN. Our model produces diverse, coherent samples of possible outcomes (i.e., affordances) in new scenes with novel objects. In comparison to our model, conditional VAE samples tend to be blurry and do not capture the geometry of unseen objects well.

11.7.2 *Scalable Robot Learning with VAL*

For general-purpose robot learning, we would ideally like to learn diverse skills continuously, using prior experience to perpetually improve at learning new skills. To study this possibility, we examine whether solving tasks in a new environment can be sped up by collecting larger amounts of prior experience. In this experiment conducted in simulation in order to carefully control the data quantity, the robot receives a prior dataset $\mathcal{D}$ of K trajectories for running VAL to grasp an unseen object in a new environment. We vary K to observe whether the method can benefit from larger amounts of prior data.

Learning curves of the online training phase, averaged over five test objects, are shown in Figure 56. First, we see from the starting point of each curve that the offline policy already generalizes to some level for grasping objects, but the average success rate is only around 35%. Importantly, note that training on more data only slightly improves generalization after offline training (at timestep 0). Then, fine-tuning results in rapid policy

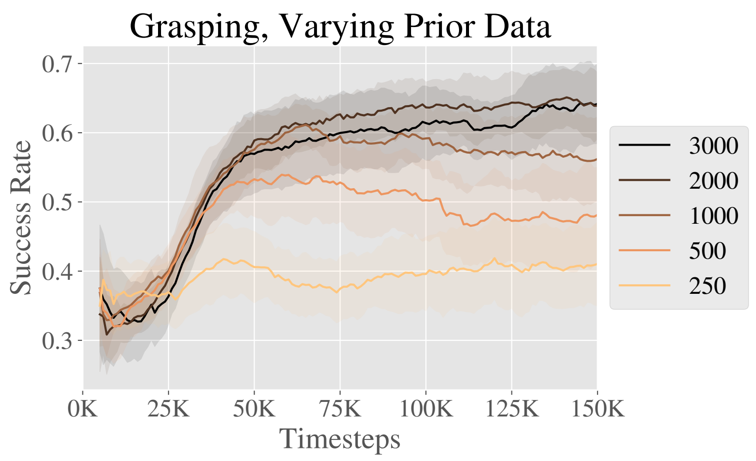


Figure 56: Learning curves for simulated grasping of novel objects with VAL, using data from an increasing number of training objects for offline RL. We collect 50 trajectories per training object, and each line is labeled with the total number of training trajectories.

improvement up to around 65% success rate after only 150,000 timesteps when utilizing the most prior data, compared to 40% with the least prior data. Thus, more prior data significantly accelerates learning in new environments even when the initial performance is comparable. This suggests that VAL can be deployed in a continual learning setting, with each new task being learned faster as it benefits from the increasing dataset size.

11.8 CONCLUSION

We present visuomotor affordance learning (VAL), a method for learning tasks online in a new environment without supervision, utilizing trajectories from other related environments. VAL uses expressive generative models to learn visual affordances, combines these affordances with off-policy goal-conditioned RL to learn skills offline, and then fine-tunes in a new environment online. Like deep learning in domains such as computer vision (Krizhevsky et al., 2012) and natural language processing (Devlin et al., 2019) which have been driven by large datasets and generalization, robotics will likely require learning from a similar scale of data. In future work, VAL could enable such systems by allowing autonomous collection of coherent exploration data in diverse real-world settings.

11.9 CONTRIBUTION STATEMENT

The work in this chapter was performed in collaboration with Alexander Khazatsky, Ashvin Nair, Daniel Jing, and Sergey Levine (Khazatsky et al., 2021b). A.K. and A.N. were joint first co-authors. A.N. proposed the affordance idea and managed the project. A.K. conducted the simulated experiments with assistance from A.N. A.K. and A.N. conducted the real-world experiments. D.J. assisted in implementing baselines in simulation experiments. S.L. advised the project and assisted with writing.

12

PLANNING TO PRACTICE: EFFICIENT ONLINE FINE-TUNING BY COMPOSING GOALS IN LATENT SPACE

12.1 INTRODUCTION

While goal-conditioned policies can be trained effectively for relatively short-horizon tasks, temporally extended multi-stage can pose a significant challenge for current methods. These tasks present a major exploration challenge during online learning, and a major challenge for credit assignment during offline learning. In this chapter, we aim to address these challenges by combining two ideas. The first is that long-horizon goal-reaching tasks can be decomposed into shorter-horizon tasks consisting of subgoals. The second is that these subgoals can be used to *fine-tune* a goal-conditioned policy online, even if its performance from offline data is poor. The first idea enables us to address the exploration challenge, by automatically generating intermediate subgoals that can be "practiced" on the way to a longer-horizon final goal. In the framework of goal-conditioned RL, solving long-horizon tasks can be reduced to the problem of optimization over a sequence of subgoals for the goal-conditioned policy, and this optimization over subgoals can be regarded as a kind of high-level planning, where the optimizer selects waypoints for achieving a distant goal. The high-level planner itself can use a learned high-level model.

However, if we rely entirely on offline data, credit assignment challenges make it difficult to perform longer-horizon tasks even with subgoal planning. Even if the offline RL policy performs well on each individual skill, there may be errors from stitching skills together because the initial states of each stage diverge from the offline data when they are composed together. In practice, this leads to poor performance when using only offline training. Therefore, the second key idea in our work is to utilize subgoal planning not merely to *perform* a multi-stage task, but also to make it possible to *practice* that task to finetune it online. While online training for temporally extended tasks is

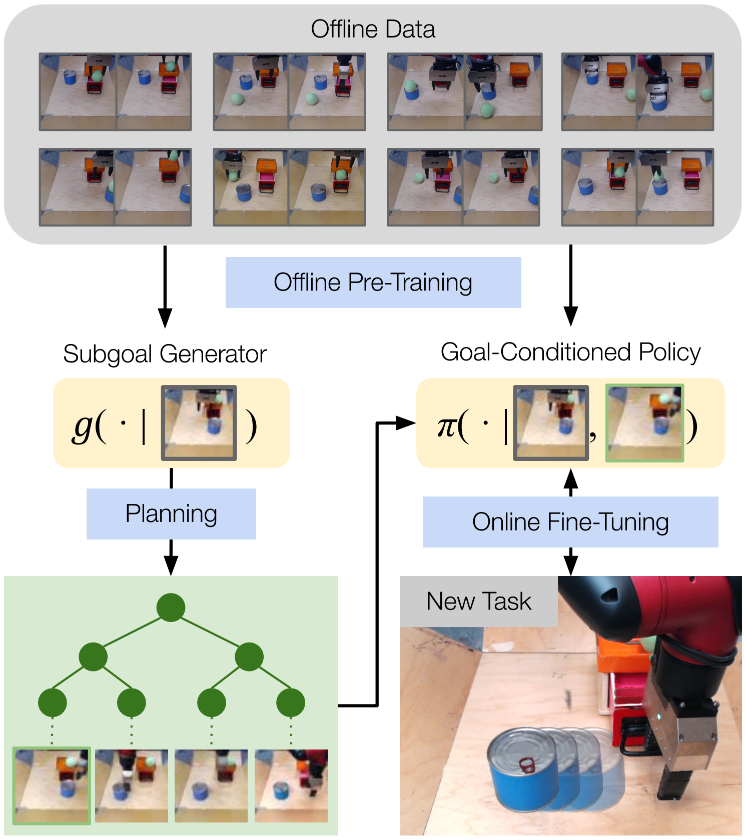


Figure 57: Our method, Plan to Practice (PTP), solves long-horizon goal-conditioned tasks by combining planning and fine-tuning. We begin with an offline dataset containing a variety of behaviors, and train a subgoal generator and goal-conditioned policy on this data. Then, to learn a more complex multi-stage tasks, we optimize over subgoals using the subgoal generator, which corresponds to a planning procedure over (visual) subgoals, and fine-tune the policy with online RL by practicing these subgoals. This enables the robot to solve multi-stage tasks directly from images.

ordinarily difficult, by addressing the exploration challenge with subgoal planning, we make it possible for the robot to practice a series of relatively short-horizon tasks, which makes this kind of finetuning feasible. Thus, the planner acts both as a higher level policy when performing the task, and as a scaffolding curriculum for finetuning the lower-level goal-conditioned policy By collecting data actively in a specific environment, we can directly experience the distribution shift and can use reinforcement learning to improve performance under this shift.

To this end, we propose Planning to Practice (PTP), an approach that efficiently trains a goal-conditioned policy to solve multi-step tasks by setting subgoals to exploit the compositional structure of the offline data. An outline is shown in Fig. 57. Our approach is based around a planner that composes generated subgoals to guide the goal-conditioned policy during an online fine-tuning phase. To propose diverse and reachable subgoals to form the candidate plans, we design a conditional subgoal generator based on conditional variational autoencoder (CVAE) (Sohn et al., 2015). Through training on the offline dataset, the conditional subgoal generator captures the distribution of reachable subgoals from a given state and generates sequences of subgoals from the learned latent space in a recursive manner. Our subgoal planning algorithm hierarchically searches for subgoals in a coarse-to-fine manner using multiple conditional subgoal generators that trained to generate goals at different temporal resolutions. Both the goal-conditioned policy and the conditional subgoal generators are pre-trained on the offline data, and the policy is fine-tuned on the novel target task.

Our main contribution is a system for learning to solve long-horizon goal-reaching tasks by fine-tuning the goal-conditioned policy with subgoal planning in a learned latent space. We evaluate our approach on multi-stage robotic manipulation tasks with raw image observations and image goals in both simulation and the real world. After being pre-trained on short demonstrations of primitive interactions, our approach is able to find feasible subgoal sequences as plans for unseen final goals by recursively generating subgoals with the learned conditional subgoal generators. By comparing our approach with both model-free methods and prior approaches that optimize over subgoals, we demonstrate that the produced plans significantly improve the learning efficiency and the resultant success rates during the online fine-tuning.

12.2 RELATED WORK

We propose to use a combination of optimization-based planning and fine-tuning with goal-conditioned reinforcement learning from prior data in order to allow robots to learn temporally extended skills. In this section, we cover prior methods in offline RL, planning, goal-conditioned RL, and how they relate to our method.

Learning from prior data. Offline reinforcement learning methods learn from prior data (Lange et al., 2012a; Fujimoto et al., 2019b; Kumar et al., 2019b; C. Zhang et al., 2021; Kumar et al., 2020b; Fujimoto and S. S. Gu, 2021; Singh et al., 2020b), and can also finetune through online interaction (A. Nair et al., 2020; Villaflor et al., 2020; Lu et al., 2021; Khazatsky et al., 2021a; S. Lee et al., 2021; Meng et al., 2021). Such methods have been used in a variety of robotic settings (Kalashnikov et al., 2018a; Cabi et al., 2019; Kalashnikov et al., 2021; Lu et al., 2021). Our focus is not on introducing new offline

RL methods. Rather, our work shows that planning over subgoals for a goal-conditioned policy that is pretrained offline can enable finetuning for temporally extended skills that would otherwise be very difficult to learn.

Goal-conditioned reinforcement learning. The aim of goal-conditioned reinforcement learning (GCRL) is to control the agent to efficiently reach specified goal states (Leslie Pack Kaelbling, 1993; Schaul et al., 2015a; Eysenbach et al., 2021). Compared to policies that are trained to solve a fixed task, the same goal-conditioned policy can perform a variety of tasks when it is commanded with different goals. Such flexibility allows GCRL to better share knowledge across different tasks and make use of goal relabeling techniques to improve the sample efficiency without meticulous reward engineering (Andrychowicz et al., 2017a; Vitchyr H. Pong et al., 2020b; M. Fang et al., 2019; Ding et al., 2019; Abhishek Gupta et al., 2019b; H. Sun et al., 2019; Eysenbach et al., 2020; Ghosh et al., 2021). Prior has explored various strategies for proposing goals for exploration (A. Nair et al., 2018b; A. Nair et al., 2019b; Khazatsky et al., 2021a; Chane-Sane et al., 2021), and studied goal-conditioned RL from offline data (Chebotar et al., 2021). However, such works generally aim to learn short-horizon behaviors, and learning to reach goals that require multiple stages (e.g., several manipulation primitives) is very difficult, as shown in our experiments. Our work aims to extend model-free goal-conditioned RL methods by incorporating elements of planning to enable effective finetuning for multi-stage tasks.

Planning. A wide range of methods have been developed for planning in robotics. At the most abstract level, symbolic task planning searches over discrete logical formulas to accomplish abstract goals (Fikes and Nilsson, 1971). Motion planning methods solve the geometric problem of reaching a goal configuration with dynamics and collision constraints (Kavraki et al., 1996; Koenig and Likhachev, 2002; Karaman and Frazzoli, 2011; Zucker et al., 2013; Kalakrishnan et al., 2011a). Prior methods have also considered task and motion planning as a combined problem (S. Srivastava et al., 2014). These methods generally assume high-level structured representations of environments and tasks, which can be difficult to actualize in real-world environments. Since in our setting we only have image inputs and not structured scene representations, we focus on methods that can handle raw images for observations and task specification.

Combining goal-conditioned RL and planning. A number of recent works have sought to integrate concepts from planning with goal-conditioned policies in order to plan sequences of subgoals for longer-horizon tasks (Nasiriany et al., 2019; Eysenbach et al., 2019; K. Fang et al., 2019; Charlesworth and Montana, 2020; Pertsch et al., 2020; Sharma et al., 2021; T. Zhang et al., 2021). These prior methods either propose subgoals from the set of previously seen states, or directly optimize over subgoals, often by utilizing a latent variable model to obtain a concise representation of image-based states (A. Nair et al., 2018b; Ichter et al., 2018; S. Nair and Finn, 2019; Nasiriany et al., 2019; Pertsch et al., 2020; Khazatsky et al., 2021a; Chane-Sane et al., 2021). The method we employ is most closely related conceptually to the method proposed by Pertsch et al. (Pertsch et al., 2020), which also employs a hierarchical subgoal optimization, and the method proposed by Nasiriany et al. (Nasiriany et al., 2019), which also optimizes over sequences of latent vectors from a generative model. Our approach makes a number of low-level improvements, including the use of a conditional generative model (A. Nair et al., 2019b), which we show leads to significantly better performance. More importantly, our method differs conceptually from these prior works in that our focus is specifically on utilizing subgoal optimization as a way to enable finetuning goal-conditioned policies for longer-horizon tasks. We show that it is in fact this capacity to enable effective finetuning that enables our method to solve more complex multi-stage tasks in our experiments.

12.3 PROBLEM STATEMENT

In this paper, we consider the problem of learning to complete a long-horizon task specified by a goal image. The robot learns over a variety of initial configurations and goal distributions, which cover a range of behaviors such as opening or closing a drawer, and picking, placing, or pushing an object. As prior data, the robot has access to an offline dataset of trajectories $\mathcal{D}_{\text{offline}} = \{\tau_1, \tau_2, \dots, \tau_N\}$ for offline pre-training. In each trajectory, the robot is controlled by a human tele-operator or a scripted policy to achieve one of the goals the environment affords. A goal-conditioned policy is pre-trained on this dataset using offline RL algorithms.

After offline pre-training, the robot is placed in a particular environment that it has online access to interact in. Even though the initial configuration of this environment may have been included in the set of training environments, the goal distribution for this environment at test time requires sequencing multiple skills together, which is not present in the offline data. For instance, as shown in Fig. 57, the robot would need to first slide away the can that blocks the drawer, then reaches the handle of the drawer, and finally opens the drawer.

Naïvely running offline RL may not solve the long-horizon test tasks for two reasons. First, the robot is given a test goal distribution that is long horizon but offline dataset consists of individual skills. The method needs to somehow compose these individual skills autonomously in order to succeed at goals drawn from the test distribution. Second, offline RL may not solve the task due to distribution shift. Distribution shift appears in two forms: distribution shift between transitions in the prior data and transitions obtained by the actively rolling out the policy, and the distribution shift introduced when performing tasks sequentially. If the robot may actively interact in the new environment to improve its policy, how can the robot further practice and improve its performance?

12.4 PRELIMINARIES

We consider a goal-conditioned Markov Decision Process (MDP) denoted by a tuple $M = (\mathcal{S}, \mathcal{A}, \rho, P, \mathcal{G}, \gamma)$ with state space $\mathcal{S}$, action space $\mathcal{A}$, initial state probability ρ, transition probability P, a goal space $\mathcal{G}$, and discount factor γ. In each episode, a desired goal $s_g \in \mathcal{G}$ is sampled for the robot to reach. At each time step t, a goal-conditioned policy $\pi(a_t|s_t, s_g)$ selects an action $a_t \in \mathcal{A}$ conditioned on the current state s_t and goal s_g. After each step, the robot receives the goal-reaching reward $r_t(s_{t+1}, s_g)$. The robot aims to reach the goal by maximizing the average cumulative reward $\mathbb{E}[\Sigma_t \gamma^t r_t]$. Our approach learns a goal-conditioned policy π for solving the target task specified by a desired final goal s_g. The goal-conditioned policy is pre-trained on a previously collected offline dataset $\mathcal{D}_{\text{offline}}$ and then fine-tuned to reach s_g by accumulating data into an online replay buffer $\mathcal{D}_{\text{online}}$. $\mathcal{D}_{\text{offline}}$ contains diverse short-horizon interactions with objects in the environment. During online fine-tuning, we would like the policy to learn to improve and compose these short-horizon behavior for multi-stage tasks specified by s_g.

Defining informative goal-reaching rewards and extracting useful state representations from high-dimensional raw observations such as images can be challenging. Following the practice in prior work (A. Nair et al., 2018b; Khazatsky et al., 2021a), we pre-train a state encoder $h = \phi(s)$ to extract the latent state representation h. By encoding the states and goals to the latent space, we can obtain an informative goal-reaching reward function $r_t = R(h_{t+1}, h_g)$ by computing $h_{t+1} = \phi(s_{t+1})$ and $h_g = \phi(s_g)$. Specifically, $R(h_{t+1}, h_g)$ returns 0 when $||h_{t+1}, h_g|| < \epsilon$ and -1 otherwise, where ϵ is a selected threshold. In addition, we also use $\phi(s_{t+1})$ as the backbone feature extractor in all of our models that take s as an input. For simplicity, we directly use s to denote h in the rest of the paper. The details of the state encoder are explained in Sec. 12.6.2.

12.5 PLANNING TO PRACTICE

We propose Planning to Practice (PTP), an approach that efficiently fine-tunes a goal-conditioned policy to solve novel tasks. To enable the robot to efficiently learn to solve the target task, we propose to use subgoals to facilitate the online fine-tuning of the goal-conditioned policy. Given the initial state s_0 and the goal state s_g, we search for a sequence of K subgoals $\hat{s}_1 : K = \hat{s}_1, ..., \hat{s}_K$ to guide the robot to reach s_g. Such subgoals will inform the goal-conditioned policy $\pi(a|s, s_g)$ what is the immediate next step on the path to s_g and provide the policy more dense reward signals compared to directly using the final goal. We choose the sequence of subgoals at the beginning of each episode and feed the first subgoal in the sequence to the goal-conditioned policy. The policy will switch to the next subgoal in the sequence when the current subgoal is reached or the time budget assigned for the current subgoal runs out.

The main challenge is to search for a sequence of subgoals that can lead to the desired final goals while ensuring each subgoal is a valid state that can be reached from the previous subgoal. Particularly when the states correspond to full images, most vectors will not actually represent valid states, and indeed naïvely optimizing over image pixels may simply result in out-of-distribution inputs that lead to erroneous results when input into the goal-conditioned policy.

As outlined in Fig. 57, we devise a method to effectively propose and select valid subgoal sequences to guide online fine-tuning by means of a generative model. At the heart of our approach is a conditional subgoal generator $g(\cdot|s_0)$ that recursively produces candidate subgoals in a hierarchical manner conditioned on the initial state s_0. To find the optimal sequence of subgoals $\hat{s}^*_{1:K}$, we first sample N candidate sequences $\hat{s}^1_{1:K}, ..., \hat{s}^N_{1:K}$ from the state space using the conditional subgoal generator. Then we rank the candidate sequences using a cost function $c(s_0, \hat{s}_{1:K}, s_g)$. The sequence that corresponds to the lowest cost will be selected as $\hat{s}^*_{1:K}$ for the goal-conditioned policy. Through this sampling-based planning procedure, we choose the subgoal for guiding the goal-conditioned policy π during online fine-tuning. The overall algorithm is summarized in Algorithm 8. Next we describe the design of each module in details.

12.5.1 *Conditional Subgoal Generation*

The effectiveness of our planner relies on the generation of diverse and feasible sequences of subgoals as candidates. Specifically, we would like to generate the candidates by sampling from the distribution of suitable subgoal sequences $p(\hat{s}_1, ..., \hat{s}_K|s_0)$ conditioned on the initial state s_0. Most existing methods independently sample the subgoal at each step from a learned prior distribution (Pertsch et al., 2020) or a replay buffer (Eysenbach et al., 2019), which is unlikely to propose useful plans for tasks with large, combinatorial state spaces (i.e., with multiple objects).

Algorithm 8 Planning To Practice (PTP)

```
Require: set of final goals $\mathcal{G}$, time horizon $T$, offline data $\mathcal{D}_{\text{offline}}$, number of subgoals $K$.
0: Train $\pi(a|s, s_g)$ and $g(s, z)$ on $\mathcal{D}_{\text{offline}}$.
0: Initialize the online replay buffer $\mathcal{D}_{\text{online}} \leftarrow \varnothing$.
0: while not converged do
0:   Reset the environment and observe $s_0$.
0:   Sample $s_g$ from $\mathcal{G}$.
0:   Plan for the subgoals $\hat{s}_{1:K}$.
0:   $k \leftarrow 1$
0:   for $t = 1, ..., T$ do
0:     Compute the action $a_t \leftarrow \pi(a_t|s_t, \hat{s}_K)$
0:     Observe the state $s_{t+1}$ and the reward $r_t$
0:     $\mathcal{D}_{\text{online}} \leftarrow \mathcal{D}_{\text{online}} \cup (s_t, a_t, r_t, s_{t+1})$.
0:     if $t \pmod{\Delta t} == 0$ or $||s_{t+1} - \hat{s}_K|| < \epsilon$ then
0:       $k \leftarrow \min(k+1, K)$
0:   Train $\pi$ on batches sampled from $\mathcal{D}_{\text{offline}}$ and $\mathcal{D}_{\text{online}}$.
   =0
```

We propose to break down $p(\hat{s}_1, ..., \hat{s}_K|s_0)$ into $p(\hat{s}_1|s_0)\Pi_{i=1}^{k} p(\hat{s}_i|\hat{s}_{i-1})$ through modeling the conditional distribution $p(s'|s)$ of the reachable next subgoal s'. By utilizing temporal compositionality, the conditional subgoal generation paradigm improves generalization and enables generation of sequences of arbitrary lengths.

We use a conditional variational encoder (CVAE) (Sohn et al., 2015) to capture the distribution of reachable goals $p(s'|s)$. In the CVAE, we define the decoder as $g(s, z)$ and the encoder as $q(z|s, s')$, where z is the learned latent representation of the transitions and it is sampled from a prior probability $p(z)$. To propose a sequence of subgoals, we use $g(s, z)$ as the conditional subgoal generator. Conditioned on the initial state s_0, the first subgoal $\hat{s}_1$ can be generated as $\hat{s}_1 = g(s_0, z_1)$ given the sampled z_1. Then the i_{th} subgoal can be recursively generated by sampling $z_i \sim p(z)$ and computing $\hat{s}_i = g(\hat{s}_{i-1}, z_i)$ given the previous subgoal $\hat{s}_{i-1}$. In this way, we could sample a sequence of i.i.d. latent representations $z_1, ..., z_K$ and recursively generate $\hat{s}_1, ..., \hat{s}_K$ conditioned on the initial state s_0 using the conditional subgoal generator.

The CVAE is trained to minimize the evidence lower bound (ELBO) (D. P. Kingma and Welling, 2014) of $p(s'|s)$ given the offline dataset $\mathcal{D}$. During training, we sample transitions (s_t, s_τ) from the offline dataset to form the minibatches, where $\tau = t + \Delta t$ is a future step that is Δt steps ahead. Instead of using a fixed Δt, we sample Δt from a range for each transition to provide richer data. To encourage the trained model to be robust

Algorithm 9 $\mathrm{Plan}(s_0, s_g, L, K, M, N)$

```
Require: the initial state $s_0$, the goal state $s_g$, number of subgoals $K$, number of levels $L$,
    multiplier $M$, number of samples $N$.
0: Sample $N$ latent action sequences $\{z^i_{1:K}\}^N_{i=1}$.
0: Recursively generate subgoals $\{\hat{s}^i_{1:K}\}^N_{i=1}$ using $g(s, z)$.
0: Select $z^*_{1:K}$ and $\hat{s}^*_{1:K}$ of the lowest cost.
0: Update $z^*_{1:K}$ and $\hat{s}^*_{1:K}$ using MPPI.
0: if $L = 1$ then
0:   return $\hat{s}^*_{1:K}$
0: else
0:   Denote $\hat{s}^*_0 \leftarrow s_0$.
0:   Initialize the plan $\hat{\mathcal{S}}$ as an empty list
0:   for $i = 1, ..., K$ do
0:     Append $\mathrm{Plan}(\hat{s}^*_{i-1}, \hat{s}^*_i, L-1, M, M, N)$ to $\mathcal{S}$
0:   return $\hat{\mathcal{S}}$
   =0
```

to compounding errors, we sample sequences composed of multiple states and use the subgoal reconstructed at the previous step as the context in the next step. Therefore, the objective for training the conditional subgoal generator is:

$$\mathbb{E}_{q(z|s_t, s_\tau)} ||s_\tau - g(s_t, z)||^2 + D_{KL}[q(z|s_t, s_\tau)||p(z)] \tag{59}$$

where $D_{KL}[\cdot||\cdot]$ indicates the KL-Divergence.

12.5.2 *Efficient Planning in the Latent Space*

We build a planner that efficiently searches for sequences of subgoals in the latent space as shown in Algorithm 9. To tackle the large search space of candidate subgoal sequences, we design a hierarchical planning algorithm that searches for subgoals in a coarse-to-fine manner and re-use the previously selected subgoals as candidates in new episodes.

The hierarchical planning is conducted at L levels with different temporal resolutions Δt_1, ..., Δt_L. The temporal resolution of each level is an integral multiple of that of the previous level, i. e., $\Delta t_i = M \Delta t_{i-1}$, where M is a scaling factor and is set to 2 in our experiments. We first plan for the subgoals $\hat{s}^1_1, \hat{s}^1_2, ...$ on the first level. Then the subgoals $\hat{s}^l_{1:K}$ of finer temporal resolution are planned on each level l to connect the subgoals planned on the previous level $l-1$. Specifically, given the adjacent subgoals $\hat{s}_i^{l-1}$ and $\hat{s}_{i+1}^{l-1}$ produced on the previous level, we plan for a segment of M subgoals $\hat{s}_{i*M+1}^{l}, ..., \hat{s}_{(i+1)*M}^{l}$ on the level l, by treating $\hat{s}_i^{l-1}$ and $\hat{s}_{i+1}^{l-1}$ as the initial state and final goal state in Eqn. 61. The planned segments are returned to the previous level and concatenated as a more fine-grained plan. For this purpose, we train L conditional subgoal generators to propose subgoals that are $\Delta t_1, ..., \Delta t_L$ steps away, respectively. In contrast to the prior work (Pertsch et al., 2020), the conditional subgoal generators enable us to plan for unseen goals that are beyond the temporal horizon of the demonstrations in the offline dataset by exploiting the compositional structure of the demonstrations. By recursively generating the subgoals across time at each level, we only need to enforce that the temporal resolution of the top level Δt_L is smaller than since the the conditional subgoal generator $f^1(s, z)$ needs to be trained on trajectories at least $\Delta t_L + 1$ steps in length.

We maintain a latent plan buffer for each level to further facilitate the planning with the conditional subgoal generator. After each episode, the selected latent representations on each level are appended to the corresponding latent plan buffer. In each target task, the subgoals are supposed to have the same semantic meaning. In spite of the variations of the initial and goals state in each episode, the optimal plans in the latent space can often be similar to each other. Therefore, we sample half of the latent representations from the prior distribution $p(z)$ and the other half from the latent plan buffer among the initial samples to enhance the chance of finding a close initial guess.

We build our planner upon the model predictive path integral (MPPI) (Gandhi et al., 2021), which iteratively optimizes the plan through importance sampling. In each interaction, we perturb the chosen plan in the latent space with a small Gaussian noise as new candidates.

12.5.3 *Cost Function For Feasible Subgoals*

To provide informative guidance to the policy $\pi(a|s, s_g)$, we would like that the final goal s_g can be reached at the end of the episode while encouraging the transition between each pair of subgoals to be feasible within a limited time budget. As explained in Sec. 12.4, the goal state is considered to be reached when the Euclidean distance between the last subgoal in the plan and the desired goal is less than a threshold δ in the learned latent space. The feasibility of each transition between adjacent subgoals can be measured using the goal-conditioned value function $V(s, s')$ trained by the reinforcement learning algorithm. Therefore, finding the subgoals $\hat{s}_{1:K}^*$ can be formulated as a

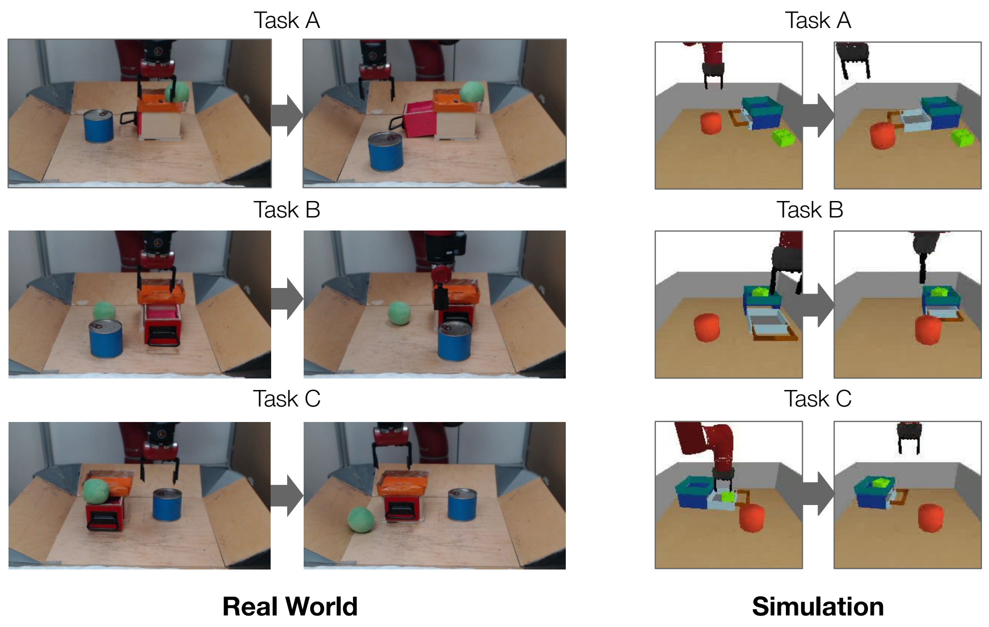


Figure 58: **Target tasks.** Three multi-stage tasks are designed for our experiments in the simulation and the real world respectively. In each target task, the robot needs to strategically interacts with the environment (e. g.first takes out an object in the drawer then closes the drawer). The initial state and the desired goal state are shown for each task.

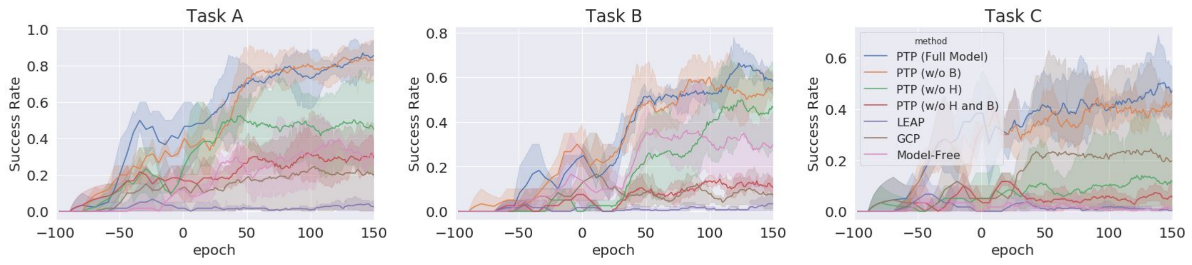


Figure 59: **Quantitative comparison in simulation.** The average success rate across 3 runs is shown with the shaded region indicating the standard deviation. The negative x-axis indicates the epochs of offline pre-training and positive x-axis indicates epochs of online fine-tuning. Using offline learning and planning, our method PTP is able to solve these tasks partially (at 0 epochs). Then with online finetuning the performance improves further. In contrast, prior methods have lower offline performance and do not fine-tune successfully in most cases, as they do not collect coherent online data.

constrained optimization problem:

$$\begin{aligned} \text{minimize} \quad & \|s_g - \hat{s}_K\| \\ \text{subject to} \quad & V(\hat{s}_i, \hat{s}_{i+1}) \geqslant \delta, \text{for } i = 0, ..., K-1 \end{aligned} \tag{60}$$

where we use $\hat{s}_0 = s_0$ to denote the initial state for convenience. By re-writing Eq. 60 as a Lagrangian, we obtain the cost function with a weight η:

$$c(s_0, \hat{s}_{1:K}, s_g) = \|s_g - \hat{s}_K\| + \eta \sum_{i=0}^{K-1} V(\hat{s}_i, \hat{s}_{i+1}) \tag{61}$$

The details of our method are explained in Sec. 12.6.2.

12.6 EXPERIMENTS

In our experiments, we aim to answer the following questions: 1) Can PTP propose and select feasible subgoals as plans for real-world robotic manipulation tasks? 2) Can the subgoals planned by PTP facilitate online fine-tuning of the goal-conditioned policies to solve target tasks unseen in the offline dataset? 3) How does each design option affect the performance of PTP? Videos of our experimental results are available on the project website: sites.google.com/view/planning-to-practice

12.6.1 *Experimental Setup*

Environment. As shown in Fig. 58, our experiments are conducted in a table-top manipulation environment with a Sawyer robot. At the beginning of each episode, a fixed drawer and two movable objects are randomly placed on the table. The robot can change the state of the environment by opening/closing the drawer, sliding the objects, and picking and placing objects into different destinations, . At each time step, the robot receives a 48 x 48 RGB image via a Logitech C920 camera as the observation and takes a 5-dimensional continuous action to change the gripper status through position control. The action dictates the change of the coordinates along the three axes, the change of the rotation, and the status of the fingers. We use PyBullet (Coumans and Bai, n.d.) for our simulated experiments.

Prior data. The prior data consists of varied demonstrations for different primitive tasks. In each demonstrated trajectory, we randomly initialize the environment and perform primitive interactions such as opening the drawer and poking the object. These trajectories are collected using teleoperation in the real world, and a scripted policy that uses privileged information of the environment (e.g., the object pose and the status of the drawer) in simulation. The trajectories vary in length from 5 to 150 time steps, with 2,344 trajectories in the real world and 4,000 in simulation.

Target tasks. In each target task, a desired goal state is specified by a 48 x 48 RGB image (same dimension with the observation). The robot is tasked to reach the goal state by interacting with the objects on the table. Task success for our evaluation is determined based on the object positions at the end of each episode (this metric is not used for learning). As shown in Fig. 58, we design three target tasks that require multi-stage interactions with the environment to complete. These target tasks are designed with temporal dependencies between stages (e. g.the robot needs to first move away a can that blocks the drawer before opening the drawer). The transitions from the initial state to the goal state are unseen in the offline data. The episode length is 400 steps in simulation and 125 steps in the real world, which are much longer than the time horizon of the demonstrations.

Baselines and ablations. We compare PTP with 3 baselines and 3 ablations. **Model-Free** uses a policy directly conditioned on the final goal and conducts online fine-tuning without using any subgoals. **LEAP** (Nasiriany et al., 2019) learns a variational auto-encoder (VAE) (D. P. Kingma and Welling, 2014) to capture the prior distribution of states and plans for subgoals without conditioning on any context. **GCP** (Pertsch et al., 2020) learns a goal predictor that hierarchically generates intermediate subgoals between the initial state and the goal state. To analyze the design options in PTP, we also compare with variations of our method by removing the latent plan buffer (**PTP (w/o B)**), the hierarchical planning algorithm (**PTP (w/o H)**), and both of these two designs (**PTP (w/o H and B)**). All methods use the same neural network architecture in the goal-conditioned policy and are pre-trained on the same offline dataset.

12.6.2 *Implementation Details*

Following Khazatsky et al., 2021a, we use a vector quantized variational autoencoder (VQ-VAE) (Aron van den Oord et al., 2017) as the state encoder, which encodes a $48 \times 48 \times 3$ image to a 720-dimensional encoding. The conditional subgoal generator is implemented with a U-Net architecture (Ronneberger et al., 2015) and decodes the subgoal from a 8-dimensional latent representations conditioned on the encoding of the current state. In our planner, we use $L = 3$, $K = 8$, $M = 2$, $N = 1024$, and we run MPPI for 5 iteration on each level. g is trained to predict subgoals that are 15, 30, and 60 steps away. Implicit Q-Learning (IQL) (Kostrikov et al., 2021b) is used as the underlying RL algorithm for offline pre-training and online fine-tuning with default hyperparameters. We use the same network architectures for the policy and the value functions from (Khazatsky et al., 2021a) for simulation experiments. For real-world experiments, we use

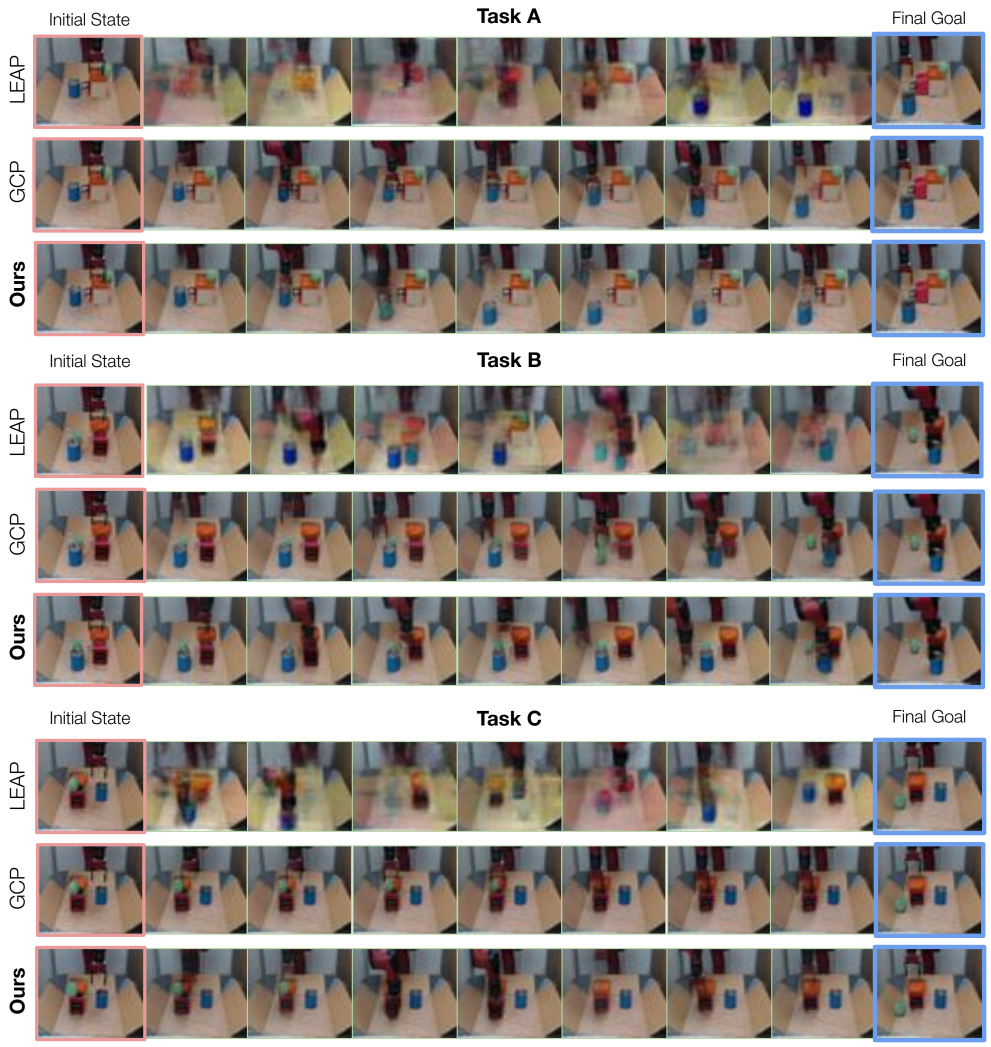


Figure 60: **Planned subgoal sequences.** Each row shows the sequence of subgoals produced by each method. The initial state and the final goal are shown at the two ends.

a convolutional neural network instead. We use Adam optimizer with a learning rate of $3 \cdot 10^{-4}$ and a batch size of 1024. During training, we relabel the goal with future hindsight experience replay (Andrychowicz et al., 2017a) with 70% probability. We use $\epsilon = 3$ for the reward function defined in Sec. 12.4, $\eta = 0.01$ in Eqn. 61.

12.6.3 *Quantitative Comparisons*

We evaluate PTP and baselines on three unseen target tasks. We use simulated versions of these tasks for comparisons and ablations, and real-world tasks, where all pretraining and finetuning uses only real-world data, to evaluate the practical effectiveness of the method.

Simulation. We first pre-train the goal-conditioned policy on the offline dataset for 100 epochs and the run online fine-tuning for the target task for 150 epochs. Each epoch takes 2,000 simulation steps (only during fine-tuning) and 2,000 training iterations. We run online fine-tuning using each method with 3 different random seeds. After each epoch, we test the policy in the target task for 5 episodes. We report the average success rate across 3 runs in Fig. 59 where the negative x-axis indicates the offline pre-training epochs and positive x-axis indicates the online fine-tuning epochs.

As shown in Fig. 59, our full model consistently outperforms baselines with a large performance gap. The generated subgoals not only enables the pre-trained policy to achieve higher success rate by breaking down the hard problems into easier pieces, but also introduces larger performance improvements during online fine-tuning. After fine-tuning for 150 epochs, the policy achieves the success rates of 84.9%, 59.9%, 49.3% in the three target tasks respectively. Compared to the policy pre-trained on the offline dataset, the performance is significantly improved (+31.6%, +37.8%, and +13.8%). When directly using the final goal or subgoals generated by baseline methods, the policy's performance plateaus at around 0.0% to 30.0% and does not improve much during online fine-tuning.

We found that the hierarchical planner and the latent plan buffer are crucial for PTP's performance. Without these two design options, the planner often suffers from the large search space of possible subgoal sequences and the resultant success rates decrease. The latent plan buffer significantly improves the performance of non-hierarchical PTP while it has a minor effect on hierarchical PTP.

Real-world evaluation. We pre-train the policy for 200 epochs and fine-tune it for 10 epochs. In each epoch, we run 10,000 training iterations and collect 1,000 steps in the real world. We train on three target tasks which are shown in Figure 58, and report the success rate of the goal-conditioned policy before and after online fine-tuning in Table 9. Planning enables the robot to succeed partially with just the offline initialized policy, achieving success rates of 12.5%, 75.0% and 25.0% on the three tasks. (When the offline policy is conditioned on only the final goal image without planning, the success rate is 0%.) Then in each task, we fine-tune to a significantly higher success rate.

Qualitatively, at the beginning of fine-tuning, the robot often fails, deviating from the

Table 9: The real-world success rates before and after online fine-tuning. The tasks are described in Sec. 12.6.1.

Task	PTP (Ours)	GCP
	Offline → Online	Offline → Online
Task A	12.5% → **62.5%**	12.5% → 0.0%
Task B	75.0% → **100.0%**	50.0% → 75.0%
Task C	25.0% → **50.0%**	25.0% → 12.5%

planned subgoals or colliding with the environment. With the planned subgoals, the original long-horizon task is broken down to short snippets that are easier to complete. Even if a subgoal is not reached successfully at first, the data is useful to collect additional experience and fine-tune the policy. After fine-tuning for 4-5 epochs, we already observe that the robot's performance reaching subgoals during training time significantly improves, collecting even more coherent and useful data. After 10 epochs, we achieve success rates of 62.5%, 100.0% and 50.0%. In comparison, GCP cannot provide useful guidance to the policy when the generated goals are noisy.

12.6.4 *Generated Subgoals*

In Fig. 60, we present qualitative results of the generated subgoals for each task in the real world. Each row shows a sequence of generated subgoals produced by the planner in each method. In all the three target tasks, PTP successfully plans for a sequence of subgoals that can lead to the desired final goal. The transition between adjacent subgoals are feasible within a short period of time. By comparison, both of the baseline methods fail to generate reasonable plans. Without conditioning on the current state, LEAP (Nasiriany et al., 2019) can hardly produce any realistic images of the environment. Most of the generated subgoals are highly noisy images with duplicated robot arms and objects. The quality of the subgoals produced by GCP (Pertsch et al., 2020) is higher than that of LEAP but still much worse than ours. GCP cannot generalize well for the initial state and the goal state that are out of the distribution of the offline dataset, which contains only short snippets of demonstrations.

12.7 CONCLUSION AND DISCUSSION

We presented PTP, a method for real-world learning of temporally extended skills by utilizing planning and fine-tuning to stitch together skills from prior data. First, planning is used to convert a long-horizon task into achievable subgoals for a lower level goal-conditioned policy trained from prior data. Then, the goal-conditioned policy is further fine-tuned with active online interaction, mitigating the distribution shift between the offline data and actual states seen during rollouts. This procedure allows robots to extend their capabilities autonomously, composing previously seen data into more complicated and useful skills.

12.8 CONTRIBUTION STATEMENT

The work in this chapter was performed in collaboration with Kuan Fang, Patrick Yin, and Sergey Levine (K. Fang et al., 2022). K.F. and P.Y. were joint first-coauthors. The idea of using an affordance model to set goals for finetuning was developed jointly by K.F. and A.N. The project was primarily managed by K.F. The majority of the simluation experiments were conducted by K.F. and P.Y. The first three authors conducted baseline experiments. The real-world experiments were conducted primarily by P.Y. with assistance from K.F. and A.N. The paper was written by K.F., with A.N. assisting. S.L. advised the project and assisted with writing.

13 CONCLUSION

In sum, we have covered two major directions forward for enabling scalable robot learning. In Part I, we covered self-supervised goal-conditioned learning for learning multitask policies from raw observations with minimal human supervision. In effect, this enables a robot to be dropped in an environment and set goals autonomously. In Part II, we covered using prior knowledge to accelerate reinforcement learning. In Part III, we explored combining these directions to enable affordance-driven robotic agents. In this final chapter, we discuss further implications of this work as well as future directions.

Offline reinforcement learning. The paradigm of offline reinforcement learning followed by online finetuning follows the broader trend in machine learning, where large expressive models trained on large datasets can be successfully fine-tuned for specific problems with relatively less data and compute available. The development of offline reinforcement learning algorithms, including the ones in this thesis (see chapter 6, 7), enables the analogous capability for control (Levine et al., 2020). Policies and value functions can be pretrained offline, and then fine-tuned online. This capability is especially important in the reinforcement learning setting because, unlike the supervised learning setting, reinforcement learning agents can often collect data, and this data will often be the most task-relevant data in the presence of distribution shift. For instance, a robot may be placed in a new home; the ability to fine-tune on newly collected data which may be out of the training distribution is vital for the robot to accomplish tasks in the new home, and reduces the necessity of perfect zero-shot performance. Such a capability can also be the basis for continual lifelong learning, by enabling the collection of high quality data in diverse environments.

In this thesis, we have discussed robotics as the main domain. Offline reinforcement learning is a clear fit for robotics as it is a difficult control problem and requires generalization from raw pixel and other sensory observations, an area of clear strength for deep learning methods. But beyond robotics, many other control and sequential decision making problems are on the cusp of being realistically automated - in finance, logistics, job scheduling, natural language understanding, chip design, science, healthcare, human-computer interaction, and more. Each domain has unique challenges, depending on the cost of collecting offline data, cost of building accurate simulators, cost of online data, and safety considerations of running policies online. But in many of these domains, the underlying control problem is often today treated as a prediction problem. The availability of stable, usable offline RL algorithms to capture the control nature of these problems can enable significant progress.

Self-supervision and goal-conditioned reinforcement learning. The other major theme covered in this thesis is multi-task learning via a continuous latent variable. We saw how a latent variable model can be used for autonomously setting goals, evaluating rewards, and learning a policy that can generalize across tasks. In this work, we mainly focused on reconstructive latent variable models, such as variational auto-encoders (D. P. Kingma and Welling, 2014). But more generally, the recipe of goal-conditioned reinforcement learning combined with latent variable models and goal relabeling can be applied to arbitrary latent variable models. This could enable more convenient task specification modalities such as language commands, demonstrations, or human feedback of policy executions, with significant sharing of the underlying RL algorithm when accommodating new modalities.

Even more generally, the recurring theme of machine learning – the "bitter lesson" according to R. Sutton (2019) – is that general methods that leverage computation and data excel in the long run as opposed to methods that limit themselves by embedding domain-specific assumptions. Realizing this vision in robotics is difficult due to all the unscalable components of real-world robotics systems: bespoke robot controllers, physical resets, instrumentation for reward and state estimation, and safety. Operating in the usual single-task reinforcement learning formalism hides this complexity. Self-supervised goal-conditioned RL combined with the ability to use offline datasets has the potential to address many of these questions in a scalable way: autonomous goal setting for resets, obviating the need for reward engineering with self-supervised reward functions, operating from raw sensor input without state estimation, and maintaining safe exploration by using offline learned policies. With these obstacles solved, goal-conditioned RL from prior data can be a data sponge that can take advantage of large-scale robot data to learn generalizble policies and create robust, adaptable robotic systems.

BIBLIOGRAPHY

Abbeel, Pieter and Andrew Y Ng (2004). “Apprenticeship learning via inverse reinforcement learning.” In: *International Conference on Machine Learning (ICML)*, p. 1 (cit. on pp. 59, 87).

Abdolmaleki, Abbas, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller (2018). “Maximum a Posteriori Policy Optimisation.” In: *International Conference on Learning Representations (ICLR)*, pp. 1–19 (cit. on pp. 75, 84, 86, 265).

Abel, David, Gabriel Barth-Maron, James Macglashan, and Stefanie Tellex (2014). “Toward Affordance-Aware Planning.” In: *RSS Workshop on Affordances in Vision for Cognitive Robotics*. URL: https://vimeo.com/88689171 (cit. on p. 158).

Adolph, Karen E and John M Franchak (2017). “The development of motor behavior.” In: DOI: 10.1002/wcs.1430 (cit. on p. 1).

Agarwal, Rishabh, Dale Schuurmans, and Mohammad Norouzi (2019). “An Optimistic Perspective on Offline Reinforcement Learning.” In: *International Conference on Machine Learning (ICML)*. eprint: 1907.04543v2 (cit. on p. 85).

Agrawal, Pulkit, Ashvin Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine (2016). “Learning to Poke by Poking: Experiential Learning of Intuitive Physics.” In: *Advances in Neural Information Processing Systems (NeurIPS)*. eprint: 1606.07419. URL: http://arxiv.org/abs/1606.07419 (cit. on pp. 9, 26, 27, 157).

Ahn, Michael, Henry Zhu, Kristian Hartikainen, Hugo Ponte, Abhishek Gupta, Sergey Levine, and Vikash Kumar (Sept. 2019). “ROBEL: Robotics Benchmarks for Learning with Low-Cost Robots.” In: *Conference on Robot Learning (CoRL)*. arXiv. eprint: 1909.11639. URL: http://arxiv.org/abs/1909.11639 (cit. on p. 91).

Alemi, Alexander A., Ian S. Fischer, Joshua V. Dillon, and Kevin P. Murphy (2017). “Deep Variational Information Bottleneck.” In: (cit. on pp. 140, 145, 149).

Andrychowicz, Marcin, Dwight Crow, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Joshua Tobin, P. Abbeel, and Wojciech Zaremba (2017a). “Hindsight Experience Replay.” In: *Advances in Neural Information Processing Systems* (cit. on pp. 175, 185).

Andrychowicz, Marcin, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob Mcgrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba (2017b).

"Hindsight Experience Replay." In: *Advances in Neural Information Processing Systems (NeurIPS)*. eprint: 1707.01495. URL: https://arxiv.org/pdf/1707.01495.pdf%20http://arxiv.org/abs/1707.01495 (cit. on pp. 10, 15, 18, 20, 27, 30, 37, 39, 44, 45, 57, 62, 65, 66, 70, 134, 141, 158, 159, 163, 229, 231, 250, 251, 257).

Ang, Kiam Heong, G. Chong, and Yun Li (2005). "PID control system analysis, design, and technology." In: 13.4, pp. 559–576. DOI: 10.1109/TCST.2005.847331 (cit. on p. 1).

Atkeson, Christopher G and Stefan Schaal (1997). "Robot Learning From Demonstration." In: *International Conference on Machine Learning (ICML)*. URL: http://www.cc.gatech.edu/fac/fChris. (cit. on p. 74).

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2015). "Neural Machine Translation by Jointly Learning to Align and Translate." In: *International Conference on Learning Representations (ICLR)*. URL: https://arxiv.org/pdf/1409.0473.pdf (cit. on p. 69).

Balduzzi, David and Muhammad Ghifary (2015). "Compatible Value Gradients for Reinforcement Learning of Continuous Deep Policies." In: abs/1509.03005. eprint: 1509.03005. URL: http://arxiv.org/abs/1509.03005 (cit. on p. 85).

Baranes, A and P-Y Oudeyer (2012). "Active Learning of Inverse Models with Intrinsically Motivated Goal Exploration in Robots." In: 61.1, pp. 49–73. DOI: 10.1016/j.robot.2012.05.008. eprint: arXiv:1301.4862v1. URL: http://dx.doi.org/10.1016/j.robot.2012.05.008 (cit. on p. 10).

Baranes, Adrien and Pierre-Yves Oudeyer (2012a). "Active Learning of Inverse Models with Intrinsically Motivated Goal Exploration in Robots." In: 61.1, pp. 49–73. DOI: 10.1016/j.robot.2012.05.008. eprint: arXiv:1301.4862v1. URL: http://dx.doi.org/10.1016/j.robot.2012.05.008 (cit. on p. 46).

Baranes, Adrien and Pierre-Yves Oudeyer (2012b). "Active Learning of Inverse Models with Intrinsically Motivated Goal Exploration in Robots." In: 61.1, pp. 49–73. DOI: 10.1016/j.robot.2012.05.008. eprint: arXiv:1301.4862v1. URL: http://dx.doi.org/10.1016/j.robot.2012.05.008 (cit. on pp. 141, 158).

Barber, David and Felix V Agakov (2004). "Information maximization in noisy channels: A variational approach." In: *Advances in Neural Information Processing Systems*, pp. 201–208 (cit. on p. 44).

Bellemare, Marc, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos (2016). "Unifying count-based exploration and intrinsic motivation." In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1471–1479 (cit. on pp. 16, 45, 158).

Bellemare, Marc G., Salvatore Candido, Pablo Samuel Castro, Jun Gong, Marlos C. Machado, Subhodeep Moitra, Sameera S. Ponda, and Ziyu Wang (Dec. 2020). "Autonomous navigation of stratospheric balloons using reinforcement learning." In: 588 (7836), pp. 77–82. ISSN: 1476-4687. DOI: 10.1038/s41586-020-2939-8. URL: https://www.nature.com/articles/s41586-020-2939-8 (cit. on p. 2).

Benbrahim, Hamid and Judy A. Franklin (Dec. 1997). "Biped dynamic walking using reinforcement learning." In: 22.3-4, pp. 283–302. ISSN: 09218890. DOI: 10.1016/S0921-8890(97)00043-2 (cit. on p. 157).

Bentivegna, Darrin C., Gordon Cheng, and Christopher G. Atkeson (2003). "Learning from Observation and from Practice Using Behavioral Primitives." In: *Robotics Research, The Eleventh International Symposium, ISRR, October 19-22, 2003, Siena, Italy*. Ed. by Paolo Dario and Raja Chatila. Vol. 15. Springer Tracts in Advanced Robotics. Springer, pp. 551–560. DOI: 10.1007/11008941_59. URL: https://doi.org/10.1007/11008941%5C_59 (cit. on p. 87).

Berger, Kathleen (2014). *The Developing Person Through the Life Span* (cit. on p. 155).

Bhatnagar, Shalabh, Richard S. Sutton, Mohammad Ghavamzadeh, and Mark Lee (2009). "Natural actor-critic algorithms." In: 45.11, pp. 2471–2482. DOI: 10.1016/j.automatica.2009.07.008. URL: https://doi.org/10.1016/j.automatica.2009.07.008 (cit. on p. 85).

Bohg, Jeannette and Danica Kragic (2010). "Learning grasping points with shape context." In: 58.4, pp. 362–377 (cit. on p. 26).

Bojarski, Mariusz, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, Karol Zieba, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba (2016). "End to End Learning for Self-Driving Cars." In: abs/1604.0, pp. 1–9. eprint: 1604.07316. URL: https://arxiv.org/pdf/1604.07316.pdf%20http://arxiv.org/abs/1604.07316 (cit. on p. 59).

Bousmalis, Konstantinos, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs, Julian Ibarz, Peter Pastor, Kurt Konolige, Sergey Levine, and Vincent Vanhoucke (2017). *Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping* (cit. on p. 143).

Bousmalis, Konstantinos, George Trigeorgis, Nathan Silberman, Dilip Krishnan, and Dumitru Erhan (2016). "Domain Separation Networks." In: Nips. eprint: 1608.06019 (cit. on p. 143).

Bradbury, James, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang (2018). *JAX: composable transformations of Python+NumPy programs*. Version 0.2.5. URL: http://github.com/google/jax (cit. on pp. 105, 274).

Brandfonbrener, David, William F Whitney, Rajesh Ranganath, and Joan Bruna (2021). "Offline RL Without Off-Policy Evaluation." In: (cit. on pp. 96, 97, 101, 103, 105, 108, 274).

Burda, Yuri, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A Efros (2019). "Large-scale study of curiosity-driven learning." In: *International Conference on Learning Representations (ICLR)* (cit. on p. 45).

Burda, Yuri, Harrison Edwards, Amos Storkey, and Oleg Klimov (2018). "Exploration by random network distillation." In: (cit. on p. 45).

Cabi, Serkan, Sergio Gmez Colmenarejo, Alexander Novikov, Ksenia Konyushkova, Scott Reed, Rae Jeong, Konrad Zolna, Yusuf Aytar, David Budden, Mel Vecerik, et al. (2019). "Scaling data-driven robotics with reward sketching and batch reinforcement learning." In: (cit. on p. 174).

Castro, Pablo Samuel (2020). "Scalable methods for computing state similarity in deterministic Markov Decision Processes." In: *Association for the Advancement of Artificial Intelligence (AAAI)* (cit. on p. 142).

Chane-Sane, Elliot, Cordelia Schmid, and Ivan Laptev (2021). "Goal-Conditioned Reinforcement Learning with Imagined Subgoals." In: *ICML* (cit. on pp. 175, 176).

Charlesworth, Henry and G. Montana (2020). "PlanGAN: Model-based Planning With Sparse Rewards and Multiple Goals." In: abs/2006.00900 (cit. on p. 175).

Chebotar, Yevgen, Karol Hausman, Yao Lu, Ted Xiao, Dmitry Kalashnikov, Jake Varley, Alex Irpan, Benjamin Eysenbach, Ryan Julian, Chelsea Finn, and Sergey Levine (Apr. 2021). "Actionable Models: Unsupervised Offline Reinforcement Learning of Robotic Skills." In: (cit. on pp. 141, 175).

Chebotar, Yevgen, Karol Hausman, Marvin Zhang, Gaurav Sukhatme, Stefan Schaal, and Sergey Levine (2017a). "Combining Model-Based and Model-Free Updates for Trajectory-Centric Reinforcement Learning." In: *International Conference on Machine Learning (ICML)*. URL: https://arxiv.org/pdf/1703.03078.pdf (cit. on p. 126).

Chebotar, Yevgen, Mrinal Kalakrishnan, Ali Yahya, Adrian Li, Stefan Schaal, and Sergey Levine (2017b). "Path integral guided policy search." In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 3381–3388 (cit. on p. 53).

Chen, Lili, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch (2021). "Decision transformer: Reinforcement learning via sequence modeling." In: (cit. on pp. 96, 105).

Chen, Xi, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel (2016). "Infogan: Interpretable representation learning by information maximizing generative adversarial nets." In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 2172–2180. eprint: 1606.03657. URL: http://arxiv.org/abs/1606.03657 (cit. on p. 10).

Chentanez, Nuttapong, Andrew G Barto, and Satinder P Singh (2005). "Intrinsically motivated reinforcement learning." In: *Advances in neural information processing systems*, pp. 1281–1288 (cit. on pp. 45, 158).

Cheung, Brian, Jesse A Livezey, Arjun K Bansal, and Bruno A Olshausen (2014). "Discovering hidden factors of variation in deep networks." In: (cit. on p. 10).

Christiano, Paul F, Jan Leike, Tom B Brown, Miljan Martic, Shane Legg, and Dario Amodei (2017). "Deep reinforcement learning from human preferences." In: *Advances in Neural Information Processing Systems (NeurIPS)*. eprint: 1706.03741. URL: https://arxiv.org/pdf/1706.03741.pdf%20http://arxiv.org/abs/1706.03741 (cit. on p. 122).

Colas, Cdric, Pierre Fournier, Olivier Sigaud, and Pierre-Yves Oudeyer (2018a). "CURIOUS: Intrinsically Motivated Multi-Task, Multi-Goal Reinforcement Learning." In: abs/1810.06284 (cit. on p. 46).

Colas, Cdric, Tristan Karch, Olivier Sigaud, and Pierre-Yves Oudeyer (2021). *Intrinsically Motivated Goal-Conditioned Reinforcement Learning: a Short Survey*. Tech. rep. eprint: 2012.09830v2 (cit. on p. 158).

Colas, Cdric, Olivier Sigaud, and Pierre-Yves Oudeyer (2018b). "GEP-PG: Decoupling Exploration and Exploitation in Deep Reinforcement Learning Algorithms." In: (cit. on p. 45).

Coumans, Erwin and Yunfei Bai (n.d.). *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. http://pybullet.org (cit. on pp. 167, 183).

Dabney, Will, Georg Ostrovski, David Silver, and Rmi Munos (2018a). "Implicit quantile networks for distributional reinforcement learning." In: *International conference on machine learning*. PMLR, pp. 1096–1105 (cit. on pp. 96, 98).

Dabney, Will, Mark Rowland, Marc G Bellemare, and Rmi Munos (2018b). "Distributional reinforcement learning with quantile regression." In: *Thirty-Second AAAI Conference on Artificial Intelligence* (cit. on pp. 96, 98).

Daniel, Christian, Malte Viering, Jan Metz, Oliver Kroemer, and Jan Peters (2014). "Active Reward Learning." In: *Robotics: Science and Systems (RSS)*. URL: http://www.roboticsproceedings.org/rss10/p31.pdf (cit. on pp. 121, 125).

Degrave, Jonas, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de las Casas, Craig Donner, Leslie Fritz, Cristian Galperti, Andrea Huber, James Keeling, Maria Tsimpoukelli, Jackie Kay, Antoine Merle, Jean Marc Moret, Seb Noury, Federico Pesamosca, David Pfau, Olivier Sauter, Cristian Sommariva, Stefano Coda, Basil Duval, Ambrogio Fasoli, Pushmeet Kohli, Koray Kavukcuoglu, Demis Hassabis, and Martin Riedmiller (Feb. 2022). "Magnetic control of tokamak plasmas through deep reinforcement learning." In: 602 (7897), pp. 414–419. ISSN: 1476-4687. DOI: 10.1038/s41586-021-04301-9. URL: https://www.nature.com/articles/s41586-021-04301-9 (cit. on p. 2).

Degris, Thomas, Martha White, and Richard S. Sutton (May 2012). "Off-Policy Actor-Critic." In: *International Conference on Machine Learning (ICML)*. eprint: 1205.4839. URL: http://arxiv.org/abs/1205.4839 (cit. on p. 85).

Deisenroth, M. P., P. Englert, J. Peters, and D. Fox (May 2014). "Multi-task policy search for robotics." In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3876–3881. DOI: 10.1109/ICRA.2014.6907421 (cit. on p. 26).

Deisenroth, Marc Peter and Carl Edward Rasmussen (2011). "PILCO: A model-based and data-efficient approach to policy search." In: *International Conference on Machine Learning (ICML)*, pp. 465–472. URL: http://mlg.eng.cam.ac.uk/pub/pdf/DeiRas11.pdf (cit. on pp. 2, 26, 59, 60, 121, 126, 141, 157).

Deisenroth, Marc Peter, Carl Edward Rasmussen, and Dieter Fox (2011a). "Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning." In: VII, pp. 57–64 (cit. on pp. 58, 60).

Deisenroth, Marc Peter, Carl Edward Rasmussen, and Dieter Fox (2011b). "Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning." In: VII, pp. 57–64. URL: http://www.roboticsproceedings.org/rss07/p08.pdf (cit. on p. 126).

Desjardins, Guillaume, Aaron Courville, and Yoshua Bengio (2012). "Disentangling factors of variation via generative entangling." In: abs/1210.5 (cit. on p. 10).

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (Oct. 2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." In: *Association for Compuational Linguistics (ACL)*. eprint: 1810.04805. URL: http://arxiv.org/abs/1810.04805 (cit. on pp. 2, 74, 170).

Ding, Yiming, Carlos Florensa, Mariano Phielipp, and P. Abbeel (2019). "Goal-conditioned Imitation Learning." In: *NeurIPS* (cit. on p. 175).

Donahue, Jeff, Philipp Krhenbhl, and Trevor Darrell (May 2017). "Adversarial Feature Learning." In: *International Conference on Learning Representations (ICLR)*. eprint: 1605.09782. URL: http://arxiv.org/abs/1605.09782 (cit. on pp. 162, 165, 169).

Duan, Yan, Marcin Andrychowicz, Bradly C. Stadie, Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba (2017). "One-Shot Imitation Learning." In: *Advances in Neural Information Processing Systems (NeurIPS)*. eprint: 1703.07326 (cit. on pp. 58, 60).

Duan, Yan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel (Nov. 2016). "RL$
2$: Fast Reinforcement Learning via Slow Reinforcement Learning." In: eprint: 1611.02779. URL: http://arxiv.org/abs/1611.02779 (cit. on p. 160).

Ebert, Frederik, Sudeep Dasari, Alex X Lee, Sergey Levine, and Chelsea Finn (2018). "Robustness via Retrying: Closed-Loop Robotic Manipulation with Self-Supervised Learning." In: *Conference on Robot Learning (CoRL)*. eprint: 1810.03043v1. URL: https://arxiv.org/pdf/1810.03043.pdf (cit. on p. 27).

Ebert, Frederik, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex Lee, and Sergey Levine (n.d.). "Visual Foresight: Model-Based Deep Reinforcement Learning for Vision-Based Robotic Control." In: (). eprint: 1812.00568v1. URL: https://sites.google.com/view/visualforesight (cit. on p. 27).

Ebert, Frederik, Chelsea Finn, Alex X Lee, and Sergey Levine (2017). "Self-Supervised Visual Planning with Temporal Skip Connections." In: *Conference on Robot Learning (CoRL)*. URL: https://128.84.21.199/pdf/1710.05268.pdf%20https://arxiv.org/pdf/1710.05268.pdf (cit. on pp. 9, 27).

Ekvall, Staffan and Danica Kragic (2004). "Interactive grasp learning based on human demonstration." In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*. Vol. 4. IEEE, pp. 3519–3524 (cit. on p. 26).

Eysenbach, Benjamin, Xinyang Geng, Sergey Levine, and Ruslan Salakhutdinov (2020). "Rewriting History with Inverse RL: Hindsight Inference for Policy Improvement." In: abs/2002.11089 (cit. on p. 175).

Eysenbach, Benjamin, Abhishek Gupta, Julian Ibarz, and Sergey Levine (2018). "Diversity is All You Need: Learning Skills without a Reward Function." In: (cit. on pp. 16, 45).

Eysenbach, Benjamin, Ruslan Salakhutdinov, and Sergey Levine (2019). "Search on the Replay Buffer: Bridging Planning and Reinforcement Learning." In: *NeurIPS* (cit. on pp. 175, 178).

Eysenbach, Benjamin, Ruslan Salakhutdinov, and Sergey Levine (2021). “C-Learning: Learning to Achieve Goals via Recursive Classification.” In: abs/2011.08909 (cit. on p. 175).

Fakoor, Rasool, Pratik Chaudhari, and Alexander J Smola (2019). “P3O: Policy-on Policy-off Policy Optimization.” In: *Conference on Uncertainty in Artificial Intelligence (UAI)*. eprint: 1905.01756v2. URL: https://github.com/rasoolfa/P3O. (cit. on p. 85).

Fakoor, Rasool, Jonas Mueller, Kavosh Asadi, Pratik Chaudhari, and Alexander J Smola (2021). “Continuous doubly constrained batch reinforcement learning.” In: (cit. on p. 95).

Falco, Joe, Yu Sun, and Maximo Roa (2018). “Robotic Grasping and Manipulation Competition: Competitor Feedback and Lessons Learned.” In: *Robotic Grasping and Manipulation*. Ed. by Yu Sun and Joe Falco. Springer International Publishing, pp. 180–189. ISBN: 978-3-319-94568-2 (cit. on p. 128).

Fang, Kuan, Patrick Yin, Ashvin Nair, and Sergey Levine (2022). “Planning to Practice: Efficient Online Fine-Tuning
by Composing Goals in Latent Space.” In: *International Conference on Robotics and Automation (ICRA)* (cit. on p. 188).

Fang, Kuan, Yuke Zhu, Animesh Garg, Silvio Savarese, and Li Fei-Fei (2019). “Dynamics Learning with Cascaded Variational Inference for Multi-Step Manipulation.” In: (cit. on p. 175).

Fang, Meng, Tianyi Zhou, Yali Du, Lei Han, and Zhengyou Zhang (2019). “Curriculum-guided Hindsight Experience Replay.” In: *NeurIPS* (cit. on p. 175).

Ferns, Norm, Prakash Panangaden, and Doina Precup (2004). “Metrics for Finite Markov Decision Processes.” In: *Uncertainty in Artificial Intelligence (UAI)*. Banff, Canada, pp. 162–169. ISBN: 0-9749039-0-6. URL: http://dl.acm.org/citation.cfm?id=1036843.1036863 (cit. on p. 142).

Fikes, Richard E and Nils J Nilsson (1971). “STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving.” In: (cit. on p. 175).

Finn, Chelsea, Pieter Abbeel, and Sergey Levine (2017). “Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks.” In: *International Conference on Machine Learning (ICML)*. URL: https://arxiv.org/pdf/1703.03400.pdf (cit. on p. 160).

Finn, Chelsea and Sergey Levine (2016). “Deep Visual Foresight for Planning Robot Motion.” In: *Advances in Neural Information Processing Systems (NeurIPS)*. eprint: 1610.00696. URL: https://arxiv.org/pdf/1610.00696.pdf%20http://arxiv.org/abs/1610.00696 (cit. on pp. 9, 27).

Finn, Chelsea, Sergey Levine, and Pieter Abbeel (2016a). “Guided Cost Learning: Deep Inverse Optimal Control via Policy Optimization.” In: *International Conference on Machine Learning (ICML)*. URL: https://arxiv.org/pdf/1603.00448.pdf (cit. on pp. 59, 127).

Finn, Chelsea, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel (2016b). “Deep spatial autoencoders for visuomotor learning.” In: *IEEE International Conference on Robotics and Automation (ICRA)*. Vol. 2016-June. IEEE, pp. 512–519. ISBN: 9781467380263. DOI: 10.1109/ICRA.2016.7487173. eprint: 1509.06113 (cit. on pp. 10, 17, 142).

Florence, Pete, Corey Lynch, Andy Zeng, Oscar Ramirez, Ayzaan Wahid, Laura Downs, Adrian Wong, Johnny Lee, Igor Mordatch, and Jonathan Tompson (2021). “Implicit Behavioral Cloning.” In: (cit. on p. 104).

Florensa, Carlos, Jonas Degrave, Nicolas Heess, Jost Tobias Springenberg, and Martin Riedmiller (2018a). “Self-supervised Learning of Image Embedding for Continuous Control.” In: *Workshop on Inference to Control at NeurIPS* (cit. on pp. 39, 45, 257).

Florensa, Carlos, Jonas Degrave, Nicolas Heess, Jost Tobias Springenberg, and Martin Riedmiller (2019). “Self-supervised Learning of Image Embedding for Continuous Control.” In: (cit. on p. 27).

Florensa, Carlos, Yan Duan, and Pieter Abbeel (2017). “Stochastic neural networks for hierarchical reinforcement learning.” In: (cit. on pp. 16, 45).

Florensa, Carlos, David Held, Markus Wulfmeier, and Pieter Abbeel (2018b). “Reverse Curriculum Generation for Reinforcement Learning.” In: *International Conference on Learning Representations (ICLR)*. eprint: 1707.05300. URL: https://arxiv.org/pdf/1707.05300.pdf%20http://arxiv.org/abs/1707.05300 (cit. on p. 73).

Frank, Jordan, Shie Mannor, and Doina Precup (2008). “Reinforcement Learning in the Presence of Rare Events.” In: *International Conference on Machine Learning (ICML)*. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.149.4737%7B%5C&%7Drep=rep1%7B%5C&%7Dtype=pdf (cit. on p. 122).

Fu, Justin, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine (Apr. 2020). “D4RL: Datasets for Deep Data-Driven Reinforcement Learning.” In: eprint: 2004.07219. URL: http://arxiv.org/abs/2004.07219 (cit. on pp. 103, 104, 106, 270, 271).

Fu, Justin, John D Co-Reyes, and Sergey Levine (2017). “EX 2 : Exploration with Exemplar Models for Deep Reinforcement Learning.” In: *Advances in Neural Information Processing Systems (NeurIPS)*. URL: https://papers.nips.cc/paper/6851-ex2-exploration-with-exemplar-models-for-deep-reinforcement-learning.pdf%20https://arxiv.org/pdf/1703.01260.pdf (cit. on p. 45).

Fu, Justin, Avi Singh, Dibya Ghosh, Larry Yang, and Sergey Levine (May 2018). "Variational Inverse Control with Events: A General Framework for Data-Driven Reward Definition." In: *Advances in Neural Information Processing Systems (NeurIPS)*. eprint: 1805.11686. URL: http://arxiv.org/abs/1805.11686 (cit. on pp. 142, 152).

Fujimoto, Scott and Shixiang Shane Gu (2021). "A Minimalist Approach to Offline Reinforcement Learning." In: (cit. on pp. 94, 95, 105, 174, 279).

Fujimoto, Scott, Herke Hoof, and David Meger (2018a). "Addressing function approximation error in actor-critic methods." In: *International Conference on Machine Learning*. PMLR, pp. 1587–1596 (cit. on p. 101).

Fujimoto, Scott, Herke van Hoof, and David Meger (2018b). "Addressing Function Approximation Error in Actor-Critic Methods." In: (cit. on pp. 12, 13, 16, 18, 30, 77, 83, 85, 115, 130, 244, 245, 250, 259).

Fujimoto, Scott, David Meger, and Doina Precup (Dec. 2019a). "Off-Policy Deep Reinforcement Learning without Exploration." In: *International Conference on Machine Learning (ICML)*. eprint: 1812.02900. URL: http://arxiv.org/abs/1812.02900 (cit. on pp. 75, 76, 80–82, 84, 85, 141).

Fujimoto, Scott, David Meger, and Doina Precup (2019b). "Off-policy deep reinforcement learning without exploration." In: *International Conference on Machine Learning*. PMLR, pp. 2052–2062 (cit. on pp. 94, 95, 174, 278, 279).

Gandhi, Manan S., Bogdan Vlahov, Jason Gibson, Grady Williams, and Evangelos A. Theodorou (2021). "Robust Model Predictive Path Integral Control: Analysis and Performance Guarantees." In: 6, pp. 1423–1430 (cit. on p. 181).

Ganin, Yaroslav, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, Franois Laviolette, Mario Marchand, and Victor Lempitsky (May 2016). "Domain-Adversarial Training of Neural Networks." In: URL: http://arxiv.org/abs/1505.07818 (cit. on pp. 140, 143, 145, 148).

Gao, Yang, Huazhe Xu, Ji Lin, Fisher Yu, Sergey Levine, and Trevor Darrell (2018). "Reinforcement Learning from Imperfect Demonstrations." In: abs/1802.05313. eprint: 1802.05313. URL: http://arxiv.org/abs/1802.05313 (cit. on p. 74).

Ghasemipour, Seyed Kamyar Seyed, Dale Schuurmans, and Shixiang Shane Gu (2021). "Emaq: Expected-max q-learning operator for simple yet effective offline and online rl." In: *International Conference on Machine Learning*. PMLR, pp. 3682–3691 (cit. on p. 95).

Ghosh, Dibya, Abhishek Gupta, and Sergey Levine (2018). "Learning actionable representations with goal-conditioned policies." In: *International Conference on Learning Representations (ICLR)* (cit. on p. 143).

Ghosh, Dibya, Abhishek Gupta, Ashwin Reddy, Justin Fu, Coline Devin, Benjamin Eysenbach, and Sergey Levine (2021). "Learning to Reach Goals via Iterated Supervised Learning." In: (cit. on p. 175).

Gibson, James (1979). *The Ecological Approach to Visual Perception* (cit. on p. 155).

Giusti, Alessandro, Jrme Jerome Guzzi, Dan C Cirean, Fang-Lin He, Juan P Rodrguez, Flavio Fontana, Matthias Faessler, Christian Forster, Jrgen Jurgen Schmidhuber, Gianni Di Caro, Davide Scaramuzza, Luca M Gambardella, Dan C. Ciresan, Fang-Lin He, Juan P. Rodriguez, Flavio Fontana, Matthias Faessler, Christian Forster, Jrgen Jurgen Schmidhuber, Gianni Di Caro, Davide Scaramuzza, and Luca M Gambardella (2015). "A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots." In: *IEEE Robotics and Automation Letters (RAL)*. Vol. 1. 2, pp. 2377–3766. ISBN: 9781467380256. DOI: 10.1109/LRA.2015.2509024. URL: http://bit.ly/perceivingtrails.%20http://ieeexplore.ieee.org/document/7358076/ (cit. on pp. 59, 126, 141).

Goodfellow, Ian J, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio (2014). "Generative Adversarial Nets." In: *Advances in Neural Information Processing Systems (NeurIPS)*. URL: https://arxiv.org/pdf/1406.2661.pdf (cit. on p. 162).

Graves, Alex, Greg Wayne, and Ivo Danihelka (2014). "Neural Turing Machines." In: abs/1410.5, pp. 1–26. ISSN: 2041-1723. DOI: 10.3389/neuro.12.006.2007. eprint: arXiv:1410.5401v2. URL: http://arxiv.org/abs/1410.5401 (cit. on p. 2).

Gu, Shixiang, Ethan Holly, Timothy Lillicrap, and Sergey Levine (2017). "Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates." In: ISSN: 0028-0836. DOI: 10.1038/nature20101. eprint: 1610.00633. URL: https://arxiv.org/pdf/1610.00633.pdf%20http://arxiv.org/abs/1610.00633 (cit. on pp. 59, 121).

Gu, Shixiang, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine (2016). "Continuous Deep Q-Learning with Model-based Acceleration." In: *International Conference on Machine Learning (ICML)*. ISBN: 3405062780. DOI: 10.3390/robotics2030122. eprint: 1603.00748. URL: https://arxiv.org/pdf/1603.00748.pdf%20http://arxiv.org/abs/1603.00748 (cit. on pp. 26, 126).

Gulcehre, Caglar, Sergio Gmez Colmenarejo, Ziyu Wang, Jakub Sygnowski, Thomas Paine, Konrad Zolna, Yutian Chen, Matthew Hoffman, Razvan Pascanu, and Nando de Freitas (2021). "Regularized behavior value estimation." In: (cit. on pp. 96, 97, 103).

Gupta, Abhishek, Benjamin Eysenbach, Chelsea Finn, and Sergey Levine (2018a). "Unsupervised Meta-Learning for Reinforcement Learning." In: abs:1806.04640 (cit. on p. 45).

Gupta, Abhishek, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman (Oct. 2019a). "Relay Policy Learning: Solving Long-Horizon Tasks via Imitation and Reinforcement Learning." In: *Conference on Robot Learning (CoRL)*. eprint: 1910.11956. URL: http://arxiv.org/abs/1910.11956 (cit. on pp. 78, 79, 86).

Gupta, Abhishek, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman (2019b). "Relay Policy Learning: Solving Long-Horizon Tasks via Imitation and Reinforcement Learning." In: *CoRL* (cit. on p. 175).

Gupta, Abhishek, Russell Mendonca, Yuxuan Liu, Pieter Abbeel, and Sergey Levine (2018b). "Meta-Reinforcement Learning of Structured Exploration Strategies." In: *Advances in Neural Information Processing Systems (NIPS)*. eprint: arXiv:1802.07245v1. URL: https://arxiv.org/pdf/1802.07245.pdf (cit. on p. 45).

Gupta, Abhishek, Justin Yu, Tony Zhao, Vikash Kumar, Kelvin Xu, Thomas Devlin, Aaron Rovinsky, and Sergey Levine (2021). "Reset-Free Reinforcement Learning via Multi-Task Learning: Learning Dexterous Manipulation Behaviors without Human Intervention." In: *International Conference on Robotics and Automation (ICRA)* (cit. on p. 92).

Ha, David and Jrgen Schmidhuber (2018). "World Models." In: (cit. on p. 10).

Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel, and Sergey Levine (2018a). "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor." In: *International Conference on Machine Learning (ICML)*. eprint: arXiv:1801.01290v2. URL: https://arxiv.org/pdf/1801.01290.pdf (cit. on pp. 2, 75, 77, 79, 83, 85, 91, 121, 124, 130, 259, 264, 268).

Haarnoja, Tuomas, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine (2018b). "Soft Actor-Critic Algorithms and Applications." In: abs/1812.05905 (cit. on pp. 50, 244, 245, 250).

Hallak, Assaf and Shie Mannor (2017). "Consistent On-Line Off-Policy Evaluation." In: *International Conference on Machine Learning (ICML)*. eprint: 1702.07121v1 (cit. on p. 85).

Hallak, Assaf, Francois Schnitzler, Timothy Mann, and Shie Mannor (2015). "Off-policy Model-based Learning under Unknown Factored Dynamics." In: *International Conference on Machine Learning (ICML)* (cit. on p. 85).

Hallak, Assaf, Aviv Tamar, Rmi Munos, and Shie Mannor (2016). "Generalized Emphatic Temporal Difference Learning: Bias-Variance Analysis." In: *Association for the Advancement of Artificial Intelligence (AAAI)*. eprint: 1509.05172v2 (cit. on p. 85).

Hart, Stephen and Roderic Grupen (2010). "Learning Generalizable Control Programs." In: *IEEE Transactions on Autonomous Mental Development* (cit. on p. 158).

Hassanin, Mohammed, Salman Khan, and Murat Tahtali (2018). *Visual Affordance and Function Understanding: A Survey*. Tech. rep. 1. eprint: 1807.06775v1 (cit. on p. 158).

Hausman, Karol, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, and Martin Riedmiller (2018). "Learning an Embedding Space for Transferable Robot Skills." In: *International Conference on Learning Representations (ICLR)*, pp. 1–16 (cit. on p. 45).

Hazan, Elad, Sham M. Kakade, Karan Singh, and Abby Van Soest (2019). "Provably Efficient Maximum Entropy Exploration." In: (cit. on pp. 45, 52, 251).

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2016). "Deep Residual Learning for Image Recognition." In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. eprint: 1512.03385v1. URL: http://image-net.org/challenges/LSVRC/2015/ (cit. on p. 122).

Heek, Jonathan, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee (2020). *Flax: A neural network library and ecosystem for JAX*. Version 0.3.5. URL: http://github.com/google/flax (cit. on p. 274).

Held, David, Xinyang Geng, Carlos Florensa, and Pieter Abbeel (2018). "Automatic Goal Generation for Reinforcement Learning Agents." In: *International Conference on Machine Learning (ICML)*. URL: https://arxiv.org/pdf/1705.06366.pdf (cit. on pp. 46, 50, 141, 158).

Hessel, Matteo, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver (2018). *Rainbow: Combining Improvements in Deep Reinforcement Learning*. eprint: 1710.02298v1. URL: www.aaai.org%20https://arxiv.org/pdf/1710.02298.pdf (cit. on p. 124).

Hester, Todd, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Gabriel Dulac-Arnold, Ian Osband, John Agapiou, Joel Z Leibo, and Audrunas Gruslys (2018). "Learning from Demonstrations for Real World Reinforcement Learning." In: *AAAI Conference on Artificial Intelligence*. eprint: 1704.03732. URL: https://arxiv.org/pdf/1704.03732.pdf%20http://arxiv.org/abs/1704.03732 (cit. on pp. 60, 122, 127).

Higgins, Irina, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner (2017a). "$$-VAE: Learning basic visual concepts with a constrained variational framework." In: (cit. on pp. 12, 44).

Higgins, Irina, Arka Pal, Andrei A Rusu, Loic Matthey, Christopher P Burgess, Alexander Pritzel, Matthew Botvinick, Charles Blundell, and Alexander Lerchner (2017b). "Darla: Improving zero-shot transfer in reinforcement learning." In: (cit. on p. 10).

Hogan, Neville (Mar. 1985). "Impedance Control: An Approach to Manipulation: Part IIImplementation." In: 107.1, pp. 8–16. ISSN: 0022-0434. DOI: 10.1115/1.3140713. eprint: https://asmedigitalcollection.asme.org/dynamicsystems/article-pdf/107/1/8/5492420/8_1.pdf. URL: https://doi.org/10.1115/1.3140713 (cit. on p. 1).

Houthooft, Rein, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel (2016). "Variational Information Maximizing Exploration." In: *Advances in Neural Information Processing Systems (NeurIPS)*. eprint: 1605.09674 (cit. on pp. 155, 158).

Ichter, Brian, James Harrison, and Marco Pavone (Sept. 2018). "Learning Sampling Distributions for Robot Motion Planning." In: pp. 7087–7094. ISSN: 10504729. DOI: 10.1109/ICRA.2018.8460730 (cit. on p. 176).

Ijspeert, Auke Jan, Jun Nakanishi, and Stefan Schaal (2002). "Learning Attractor Landscapes for Learning Motor Primitives." In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1547–1554. ISBN: 1049-5258. URL: https://papers.nips.cc/paper/2140-learning-attractor-landscapes-for-learning-motor-primitives.pdf (cit. on p. 86).

Inoue, Tadanobu, Giovanni De Magistris, Asim Munawar, Tsuyoshi Yokoya, and Ryuki Tachibana (2017). "Deep reinforcement learning for high precision assembly tasks." In: *IROS*, pp. 819–825 (cit. on p. 127).

James, Stephen, Paul Wohlhart, Mrinal Kalakrishnan, Dmitry Kalashnikov, Alex Irpan, Julian Ibarz, Sergey Levine, Raia Hadsell, and Konstantinos Bousmalis (Dec. 2018). "Sim-to-Real via Sim-to-Sim: Data-efficient Robotic Grasping via Randomized-to-Canonical Adaptation Networks." In: URL: http://arxiv.org/abs/1812.07252 (cit. on p. 143).

Jaques, Natasha, Asma Ghandeharioun, Judy Hanwen Shen, Craig Ferguson, gata Lapedriza, Noah Jones, Shixiang Gu, and Rosalind W. Picard (2019). "Way Off-Policy Batch Deep Reinforcement Learning of Implicit Human Preferences in Dialog." In: abs/1907.00456. eprint: 1907.00456. URL: http://arxiv.org/abs/1907.00456 (cit. on p. 85).

Jiang, Nan and Lihong Li (2016). “Doubly Robust Off-policy Value Evaluation for Reinforcement Learning.” In: *International Conference on Machine Learning (ICML)*. eprint: 1511.03722v3 (cit. on p. 85).

Johannink, Tobias, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine (2019). “Residual Reinforcement Learning for Robot Control.” In: *IEEE International Conference on Robotics and Automation (ICRA)*. eprint: 1812.03201v2. URL: https://arxiv.org/pdf/1812.03201.pdf (cit. on pp. 4, 123, 125, 127, 142).

Johnson, Eric and Anthony Calise (2000). “Pseudo-Control Hedging: A New Method For Adaptive Control.” In: *Advances in Navigation Guidance and Control Technology Workshop*. URL: https://www.researchgate.net/publication/2492500%7B%5C_%7DPseudo-Control%7B%5C_%7DHedging%7B%5C_%7DA%7B%5C_%7DNew%7B%5C_%7DMethod%7B%5C_%7DFor%7B%5C_%7DAdaptive%7B%5C_%7DControl (cit. on p. 122).

Jonschkowski, Rico, Roland Hafner, Jonathan Scholz, and Martin Riedmiller (May 2017a). “PVEs: Position-Velocity Encoders for Unsupervised Learning of Structured State Representations.” In: ISBN: 1705.09805v3. URL: https://arxiv.org/abs/1705.09805v3 (cit. on p. 143).

Jonschkowski, Rico, Roland Hafner, Jonathan Scholz, and Martin Riedmiller (2017b). “Pves: Position-velocity encoders for unsupervised learning of structured state representations.” In: (cit. on p. 10).

Jumper, John, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin dek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis (2021). “Highly accurate protein structure prediction with AlphaFold.” In: 596.7873, pp. 583–589. DOI: 10.1038/s41586-021-03819-2. URL: https://doi.org/10.1038/s41586-021-03819-2 (cit. on p. 2).

Kaelbling, L P (1993). “Learning to achieve goals.” In: *International Joint Conference on Artificial Intelligence (IJCAI)*. Vol. vol.2, pp. 1094–8 (cit. on pp. 9, 10, 15, 27, 37, 39, 45, 141, 158, 250, 257).

Kaelbling, Leslie Pack (1993). “Learning to Achieve Goals.” In: *IJCAI* (cit. on p. 175).

Kaelbling, Leslie Pack and Tomas Lozano-Perez (2011). “Hierarchical task and motion planning in the now.” In: pp. 1470–1477. ISSN: 10504729. DOI: 10.1109/ICRA.2011.

5980391. URL: http://people.csail.mit.edu/lpk/papers/hpn2.pdf%20http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5980391 (cit. on pp. 1, 60).

Kahn, Gregory, Adam Villaflor, Bosen Ding, Pieter Abbeel, and Sergey Levine (2018). "Self-supervised Deep Reinforcement Learning with Generalized Computation Graphs for Robot Navigation." In: *IEEE International Conference on Robotics and Automation (ICRA)*. eprint: 1709.10489v3. URL: https://arxiv.org/pdf/1709.10489.pdf (cit. on p. 26).

Kalakrishnan, Mrinal, Jonas Buchli, Peter Pastor, and Stefan Schaal (2009). "Learning Locomotion over Rough Terrain using Terrain Templates." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. URL: https://pdfs.semanticscholar.org/02db/2c0100ffb02592e8738d0ffcf454224f4b1b.pdf (cit. on pp. 59, 141).

Kalakrishnan, Mrinal, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal (2011a). "STOMP: Stochastic trajectory optimization for motion planning." In: pp. 4569–4574. ISSN: 10504729. DOI: 10.1109/ICRA.2011.5980280 (cit. on p. 175).

Kalakrishnan, Mrinal, Ludovic Righetti, Peter Pastor, and Stefan Schaal (2011b). "Learning force control policies for compliant manipulation." In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 4639–4644 (cit. on p. 53).

Kalashnikov, Dmitry, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. (2018a). "Scalable deep reinforcement learning for vision-based robotic manipulation." In: *Conference on Robot Learning*. PMLR, pp. 651–673 (cit. on p. 174).

Kalashnikov, Dmitry, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine (2018b). "QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation." In: *Conference on Robot Learning (CoRL)*. eprint: 1806.10293v3. URL: https://goo.gl/ykQn6g. (cit. on p. 126).

Kalashnikov, Dmitry, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine (n.d.). "Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation." In: *Proceedings of The 2nd Conference on Robot Learning*. Vol. 87. Proceedings of Machine Learning Research. PMLR, pp. 651–673 (cit. on p. 34).

Kalashnikov, Dmitry, Jacob Varley, Yevgen Chebotar, Benjamin Swanson, Rico Jonschkowski, Chelsea Finn, Sergey Levine, and Karol Hausman (2021). "MT-Opt: Continuous Multi-Task Robotic Reinforcement Learning at Scale." In: (cit. on pp. 141, 174).

Kannengiesser, Udo, Richard Heininger, Lubomir Billy, Pavol Terpak, Matthias Neubauer, Chris Stary, Dennis Majoe, Alexandra Totter, and David Bonaldi (Jan. 2017). *Lot-Size One Production*, pp. 69–111. ISBN: 978-3-319-48465-5. DOI: 10.1007/978-3-319-48466-2_4 (cit. on p. 111).

Karaman, Sertac and Emilio Frazzoli (June 2011). "Sampling-based algorithms for optimal motion planning:" in: 30 (7), pp. 846–894. ISSN: 02783649. DOI: 10.1177/0278364911406761 (cit. on pp. 1, 175).

Kavraki, Lydia, Petr Svestka, J-C Latombe, and Mark Overmars (1996). "Probabilistic roadmaps for path planning in high-dimensional configuration spaces." In: 12.4, pp. 566–580 (cit. on pp. 60, 175).

Khazatsky, Alexander, Ashvin Nair, Dan Jing, and Sergey Levine (2021a). "What Can I Do Here? Learning New Skills by Imagining Visual Affordances." In: pp. 14291–14297 (cit. on pp. 5, 174–177, 184).

Khazatsky, Alexander, Ashvin Nair, Daniel Jing, and Sergey Levine (June 2021b). "What Can I Do Here? Learning New Skills by Imagining Visual Affordances." In: *International Conference on Robotics and Automation (ICRA)*. eprint: 2106.00671. URL: https://arxiv.org/abs/2106.00671v2 (cit. on pp. 141, 171).

Khetarpal, Khimya, Zafarali Ahmed, Gheorghe Comanici, David Abel, and Doina Precup (2020). "What can I do here? A Theory of Affordances in Reinforcement Learning." In: *International Conference on Machine Learning (ICML)*. eprint: 2006.15085v1 (cit. on p. 158).

Kim, Beomjoon, Amir-Massoud Farahmand, Joelle Pineau, and Doina Precup (2013). "Learning from Limited Demonstrations." In: *Advances in Neural Information Processing Systems (NeurIPS)*. URL: https://papers.nips.cc/paper/4918-learning-from-limited-demonstrations.pdf (cit. on pp. 60, 86).

Kingma, Diederik and Jimmy Ba (2015). "Adam: A method for stochastic optimization." In: (cit. on pp. 65, 274).

Kingma, Diederik P and Max Welling (2014). "Auto-Encoding Variational Bayes." In: *International Conference on Learning Representations (ICLR)*. URL: https://arxiv.org/pdf/1312.6114.pdf (cit. on pp. 8, 12, 159, 162, 165, 169, 179, 184, 190).

Kober, Jens, J Andrew Bagnell, and Jan Peters (2013). "Reinforcement learning in robotics: A survey." In: 32.11, pp. 1238–1274 (cit. on pp. 2, 26).

Kober, Jens and J. Peter (2008). "Policy search for motor primitives in robotics." In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 97, pp. 83–117. ISBN:

1099401052236. DOI: 10.1007/978-3-319-03194-1_4. URL: http://papers.nips.cc/paper/3545-policy-search-for-motor-primitives-in-robotics.pdf (cit. on pp. 60, 87, 122, 126, 141, 157).

Koenig, Sven and Maxim Likhachev (2002). "D* Lite." In: (cit. on p. 175).

Koenker, Roger and Kevin F Hallock (2001). "Quantile regression." In: 15.4, pp. 143–156 (cit. on pp. 96, 98).

Kohl, Nate and Peter Stone (2004). "Machine Learning for Fast Quadrupedal Locomotion." In: *AAAI Conference on Artificial Intelligence*, pp. 611–616. URL: http://www.cs.utexas.edu/%7B%5C%%7D7Bnate,pstone%7B%5C%%7D7D (cit. on p. 157).

Konda, Vijay R and John N Tsitsiklis (2000). "Actor-Critic Algorithms." In: *Advances in Neural Information Processing Systems (NeurIPS)* (cit. on pp. 77, 85).

Kormushev, Petar, Sylvain Calinon, and Darwin G. Caldwell (2010). "Robot motor skill coordination with EM-based Reinforcement Learning." In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, October 18-22, 2010, Taipei, Taiwan*. IEEE, pp. 3232–3237. DOI: 10.1109/IROS.2010.5649089. URL: https://doi.org/10.1109/IROS.2010.5649089 (cit. on p. 87).

Kostrikov, Ilya (Oct. 2021). *JAXRL: Implementations of Reinforcement Learning algorithms in JAX.* DOI: 10.5281/zenodo.5535154. URL: https://github.com/ikostrikov/jaxrl (cit. on p. 105).

Kostrikov, Ilya, Rob Fergus, Jonathan Tompson, and Ofir Nachum (2021a). "Offline reinforcement learning with fisher divergence critic regularization." In: *International Conference on Machine Learning*. PMLR, pp. 5774–5783 (cit. on pp. 94, 95, 279).

Kostrikov, Ilya, Ashvin Nair, and Sergey Levine (2021b). "Offline reinforcement learning with implicit q-learning." In: (cit. on pp. 4, 107, 141, 143, 146, 184).

Krainin, Michael, Brian Curless, and Dieter Fox (2011). "Autonomous generation of complete 3D object models using next best view manipulation planning." In: *2011 IEEE International Conference on Robotics and Automation*. IEEE, pp. 5031–5037 (cit. on p. 26).

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "Imagenet classification with deep convolutional neural networks." In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1097–1105 (cit. on pp. 2, 74, 170).

Kroemer, Oliver, Renaud Detry, Justus Piater, and Jan Peters (2010). "Combining Active Learning and Reactive Control for Robot Grasping." In: 58.9, pp. 1105–1116. URL: https://orbi.uliege.be//bitstream/2268/60643/1/Kroemer-2010-RAS-author-postprint.pdf (cit. on p. 26).

Kumar, Aviral, Justin Fu, George Tucker, and Sergey Levine (June 2019a). "Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction." In: *Advances in Neural*

Information Processing Systems (NeurIPS). eprint: 1906.00949. URL: http://arxiv.org/abs/1906.00949 (cit. on pp. 75, 76, 79–82, 84, 85, 268, 270).

Kumar, Aviral, Justin Fu, George Tucker, and Sergey Levine (2019b). "Stabilizing off-policy q-learning via bootstrapping error reduction." In: (cit. on pp. 94, 95, 174, 279).

Kumar, Aviral, Anikait Singh, Stephen Tian, Chelsea Finn, and Sergey Levine (Sept. 2021). "A Workflow for Offline Model-Free Robotic Reinforcement Learning." In: DOI: 10.48550/arxiv.2109.10813. URL: https://arxiv.org/abs/2109.10813v2 (cit. on p. 141).

Kumar, Aviral, Aurick Zhou, George Tucker, and Sergey Levine (2020a). "Conservative Q-Learning for Offline Reinforcement Learning." In: *Advances in Neural Information Processing Systems (NeurIPS)*. eprint: 2006.04779v1 (cit. on pp. 141, 269).

Kumar, Aviral, Aurick Zhou, George Tucker, and Sergey Levine (2020b). "Conservative q-learning for offline reinforcement learning." In: (cit. on pp. 94, 95, 105, 106, 174, 277, 279).

Kuznetsov, Arsenii, Pavel Shvechikov, Alexander Grishin, and Dmitry Vetrov (2020). "Controlling overestimation bias with truncated mixture of continuous distributional quantile critics." In: *International Conference on Machine Learning*. PMLR, pp. 5556–5566 (cit. on p. 96).

Lange, Sascha, Thomas Gabel, and Martin Riedmiller (2012a). "Batch reinforcement learning." In: *Reinforcement learning*. Springer, pp. 45–73 (cit. on pp. 94, 174).

Lange, Sascha, Thomas Gabel, and Martin A. Riedmiller (2012b). "Batch Reinforcement Learning." In: *Reinforcement Learning*. Ed. by Marco Wiering and Martijn van Otterlo. Vol. 12. Adaptation, Learning, and Optimization. Springer, pp. 45–73. DOI: 10.1007/978-3-642-27645-3_2. URL: https://doi.org/10.1007/978-3-642-27645-3%5C_2 (cit. on p. 85).

Lange, Sascha, Martin Riedmiller, Arne Voigtlander, and Arne Voigtlnder (2012c). "Autonomous reinforcement learning on raw visual input data in a real world application." In: *International Joint Conference on Neural Networks (IJCNN)*. June. IEEE, pp. 1–8. ISBN: 9781467314909. DOI: 10.1109/IJCNN.2012.6252823 (cit. on pp. 9, 10, 142).

Lange, Sascha and Martin A Riedmiller (2010). "Deep learning of visual control policies." In: *European Symposium on Artificial Neural Networks (ESANN)*. Citeseer. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.226.6898%7B%5C&%7Drep=rep1%7B%5C&%7Dtype=pdf (cit. on pp. 17, 142).

Laskin, Michael, Aravind Srinivas, and Pieter Abbeel (2020). "CURL: Contrastive unsupervised representations for reinforcement learning." In: *International Conference on Machine Learning (ICML)*. PMLR, pp. 5639–5650 (cit. on p. 142).

LaValle, Steven M (2006). *Planning algorithms*. Cambridge university press (cit. on p. 1).

Lee, Alex, Sergey Levine, and Pieter Abbeel (2017). "Learning Visual Servoing with Deep Features and Fitted Q-Iteration." In: *International Conference on Learning Representations (ICLR)*. URL: https://arxiv.org/pdf/1703.11000.pdf (cit. on p. 10).

Lee, Alex X, Richard Zhang, Frederik Ebert, Pieter Abbeel, Chelsea Finn, and Sergey Levine (n.d.). "Stochastic Adversarial Video Prediction." In: eprint: 1804.01523v1. URL: https://alexlee-gk.github.io/video%7B%5C_%7Dprediction (cit. on p. 27).

Lee, Joonho ; Jemin ; Hwangbo, Lorenz ; Wellhausen, Vladlen ; Koltun, and Marco Hutter (2020). "Learning quadrupedal locomotion over challenging terrain." In: DOI: 10.3929/ethz-b-000448343. URL: https://doi.org/10.3929/ethz-b-000448343 (cit. on p. 2).

Lee, Seunghyun, Younggyo Seo, Kimin Lee, Pieter Abbeel, and Jinwoo Shin (2021). "Offline-to-Online Reinforcement Learning via Balanced Replay and Pessimistic Q-Ensemble." In: (cit. on pp. 141, 174).

Levine, Sergey, Chelsea Finn, Trevor Darrell, and Pieter Abbeel (2016a). "End-to-End Training of Deep Visuomotor Policies." In: 17.1, pp. 1334–1373. ISSN: 15337928. DOI: 10.1007/s13398-014-0173-7.2. eprint: 1504.00702. URL: https://arxiv.org/pdf/1504.00702.pdf (cit. on pp. 9, 18, 26, 121, 126, 141, 157).

Levine, Sergey, Aviral Kumar, George Tucker, and Justin Fu (2020). *Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems*. Tech. rep. eprint: 2005.01643v1 (cit. on pp. 80, 85, 189).

Levine, Sergey, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen (2016b). "Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection." In: (cit. on p. 59).

Levine, Sergey, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen (2017). "Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection." In: (cit. on pp. 2, 9, 26, 27, 121, 141).

Levy, Andrew, Robert Platt, and Kate Saenko (2017). "Hierarchical Actor-Critic." In: (cit. on p. 10).

Li, Rui, Robert Platt, Wenzhen Yuan, Andreas Ten Pas, Nathan Roscup, Mandayam A Srinivasan, and Edward Adelson (2014). "Localization and Manipulation of Small Parts Using GelSight Tactile Sensing." In: *International Conference on Intelligent Robots and Systems (IROS)*. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.448.7123%7B%5C&%7Drep=rep1%7B%5C&%7Dtype=pdf (cit. on p. 127).

Lian, Wenzhao, Tim Kelch, Dirk Holz, Adam Norton, and Stefan Schaal (2021). "Benchmarking Off-The-Shelf Solutions to Robotic Assembly Tasks." In: *IEEE International*

Conference on Intelligent Robots and Systems (IROS). eprint: 2103.05140v1 (cit. on p. 142).

Lillicrap, Timothy P, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra (2016). "Continuous control with deep reinforcement learning." In: *International Conference on Learning Representations (ICLR)*. ISBN: 0-7803-3213-X. DOI: 10.1613/jair.301. eprint: 9605103. URL: https://arxiv.org/pdf/1509.02971.pdf (cit. on pp. 2, 9, 12, 39, 59, 61, 77, 115, 257, 268).

Lin, Xingyu, Harjatin Singh Baweja, and David Held (2019). "Reinforcement Learning without Ground-Truth State." In: eprint: 1905.07866v1. URL: https://arxiv.org/pdf/1905.07866.pdf (cit. on p. 27).

Loftin, Robert, James Macglashan, Bei Peng, Matthew E Taylor, Michael L Littman, Jeff Huang, and David L Roberts (2014). "A Strategy-Aware Technique for Learning Behaviors from Discrete Human Feedback." In: *AAAI Conference on Artificial Intelligence*, pp. 937–943. ISBN: 9781577356783 (cit. on p. 122).

Long, Mingsheng, Yue Cao, Jianmin Wang, and Michael I. Jordan (Feb. 2015). "Learning Transferable Features with Deep Adaptation Networks." In: URL: http://arxiv.org/abs/1502.02791 (cit. on p. 143).

Lopes, Manuel, Tobias Lang, Marc Toussaint, and Pierre-Yves Oudeyer (2012). "Exploration in model-based reinforcement learning by empirically estimating learning progress." In: *Advances in Neural Information Processing Systems*, pp. 206–214 (cit. on pp. 45, 158).

Lu, Yao, Karol Hausman, Yevgen Chebotar, Mengyuan Yan, Eric Jang, Alexander Herzog, Ted Xiao, Alex Irpan, Mohi Khansari, Dmitry Kalashnikov, and Sergey Levine (2021). "AW-Opt: Learning Robotic Skills with Imitation and Reinforcement at Scale." In: (cit. on p. 174).

Luo, J, E Solowjow, C Wen, J Aparicio Ojea, A Agogino, A Tamar, and Abbeel P (2019). "Reinforcement Learning on Variable Impedance Controller for High-Precision Robotic Assembly." In: *ICRA* (cit. on p. 127).

Luo, Jianlan, Oleg Sushkov, Rugile Pevceviciute, Wenzhao Lian, Chang Su, Mel Vecerik, Ning Ye, Stefan Schaal, and Jon Scholz (Mar. 2021). "Robust Multi-Modal Policies for Industrial Assembly via Reinforcement Learning and Demonstrations: A Large-Scale Study." In: *Robotics: Science and Systems (RSS)*. eprint: 2103.11512. URL: https://arxiv.org/abs/2103.11512v3 (cit. on p. 142).

Lynch, Corey, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet (Mar. 2019). "Learning Latent Plans from Play." In: *Con-*

ference on Robot Learning (CoRL). eprint: 1903.01973. URL: http://arxiv.org/abs/1903.01973 (cit. on p. 157).

Martinez, David, Guillem Alenya, Pablo Jimenez, Carme Torras, Jrgen Rossmann, Nils Wantia, Eren Erdal Aksoy, Simon Haller, and Justus Piater (2014). "Active learning of manipulation sequences." In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 5671–5678 (cit. on p. 26).

Mason, Matthew T. (1981). "Compliance and Force Control for Computer Controlled Manipulators." In: 11.6, pp. 418–432. DOI: 10.1109/TSMC.1981.4308708 (cit. on p. 1).

Meng, Linghui, Muning Wen, Yaodong Yang, Chenyang Le, Xiyun Li, Weinan Zhang, Ying Wen, Haifeng Zhang, Jun Wang, and Bo Xu (2021). "Offline Pre-trained Multi-Agent Decision Transformer: One Big Sequence Model Tackles All SMAC Tasks." In: (cit. on pp. 141, 174).

Min, Huaqing, An Yi, Ronghua Luo, Jinhui Zhu, and Sheng Bi (2016). "Affordance Research in Developmental Robotics: A Survey." In: 8.4. DOI: 10.1109/TCDS.2016.2614992. URL: http://www.ieee.org/publications%7B%5C_%7Dstandards/publications/rights/index.html (cit. on p. 158).

Mlling, Katharina, Jens Kober, Oliver Kroemer, and Jan Peters (2013). "Learning to select and generalize striking movements in robot table tennis." In: 32.3, pp. 263–279. DOI: 10.1177/0278364912472380. URL: https://doi.org/10.1177/0278364912472380 (cit. on p. 87).

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller (2013). "Playing Atari with Deep Reinforcement Learning." In: *NIPS Workshop on Deep Learning*, pp. 1–9. ISBN: 1476-4687 (Electronic) 0028-0836 (Linking). DOI: 10.1038/nature14236. eprint: 1312.5602. URL: https://arxiv.org/pdf/1312.5602.pdf%20http://arxiv.org/abs/1312.5602%20https://www.cs.toronto.edu/%7B~%7Dvmnih/docs/dqn.pdf (cit. on pp. 26, 124, 126, 157).

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. (2015). "Human-level control through deep reinforcement learning." In: 518.7540, pp. 529–533 (cit. on pp. 2, 59).

Mnih, Volodymyr, Adri Puigdomnech Badia, Mehdi Mirza, Alex Graves, Tim Harley, Timothy P Lillicrap, David Silver, Koray Kavukcuoglu, Korayk@google Com, and Google Deepmind (2016). "Asynchronous Methods for Deep Reinforcement Learning." In: *International Conference on Machine Learning (ICML)*. URL: https://arxiv.org/pdf/1602.01783.pdf (cit. on pp. 12, 59, 77, 85).

Mohamed, Shakir and Danilo Jimenez Rezende (2015). "Variational information maximisation for intrinsically motivated reinforcement learning." In: *Advances in neural information processing systems*, pp. 2125–2133 (cit. on p. 45).

Nachum, Ofir, Yinlam Chow, Bo Dai, and Lihong Li (June 2019). "DualDICE: Behavior-Agnostic Estimation of Discounted Stationary Distribution Corrections." In: *Advances in Neural Information Processing Systems (NeurIPS)*. eprint: 1906.04733. URL: http://arxiv.org/abs/1906.04733 (cit. on pp. 85, 269).

Nachum, Ofir, Shane Shixiang Gu, Honglak Lee, and Sergey Levine (2018). "Data-Efficient Hierarchical Reinforcement Learning." In: *Advances in Neural Information Processing Systems (NeurIPS)*. eprint: arXiv:1805.08296v2. URL: https://sites.google.com/view/efficient-hrl (cit. on pp. 24, 46, 141, 158).

Nair, Ashvin, Shikhar Bahl, Alexander Khazatsky, Vitchyr Pong, Glen Berseth, and Sergey Levine (Oct. 2019a). "Contextual Imagined Goals for Self-Supervised Robotic Learning." In: *Conference on Robot Learning (CoRL)*. eprint: 1910.11670. URL: http://arxiv.org/abs/1910.11670 (cit. on pp. 4, 157, 158).

Nair, Ashvin, Shikhar Bahl, Alexander Khazatsky, Vitchyr H. Pong, Glen Berseth, and Sergey Levine (2019b). "Contextual Imagined Goals for Self-Supervised Robotic Learning." In: *CoRL* (cit. on pp. 175, 176).

Nair, Ashvin, Dian Chen, Pulkit Agrawal, Phillip Isola, Pieter Abbeel, Jitendra Malik, Sergey Levine, Dian Chen, Phillip Isola, Pieter Abbeel, Jitendra Malik, and Sergey Levine (2017). "Combining Self-Supervised Learning and Imitation for Vision-Based Rope Manipulation." In: *IEEE International Conference on Robotics and Automation (ICRA)*. ISBN: 9781509046331. DOI: 10.1109/ICRA.2017.7989247. eprint: 1703.02018 (cit. on p. 86).

Nair, Ashvin, Murtaza Dalal, Abhishek Gupta, and Sergey Levine (June 2020). "Accelerating Online Reinforcement Learning with Offline Datasets." In: eprint: 2006.09359. URL: http://arxiv.org/abs/2006.09359 (cit. on pp. 4, 93–96, 101, 105, 106, 141, 163, 174, 277, 282).

Nair, Ashvin, Bob Mcgrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel (2018a). "Overcoming Exploration in Reinforcement Learning with Demonstrations." In: *IEEE International Conference on Robotics and Automation (ICRA)*. URL: https://arxiv.org/pdf/1709.10089.pdf (cit. on pp. 4, 75, 91, 122, 127, 131, 268).

Nair, Ashvin, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine (2018b). "Visual Reinforcement Learning with Imagined Goals." In: *Advances in Neural Information Processing Systems (NeurIPS)*. eprint: arXiv:1807.04742v1. URL: https:

//sites.google.com/site/ (cit. on pp. 4, 23, 24, 27, 30, 31, 35, 39, 44, 45, 51, 125, 141, 157–159, 163, 175–177, 247, 250, 251, 257).

Nair, Suraj and Chelsea Finn (2019). "Hierarchical foresight: Self-supervised learning of long-horizon tasks via visual subgoal generation." In: (cit. on p. 176).

Nakanishi, Jun, Jun Morimoto, Gen Endo, Gordon Cheng, Stefan Schaal, and Mitsuo Kawato (2004). "Learning from demonstration and adaptation of biped locomotion." In: *Robotics and Autonomous Systems*. Vol. 47. 2-3, pp. 79–91. ISBN: 0921-8890. DOI: 10.1016/j.robot.2004.03.003. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.77.534%7B%5C&%7Drep=rep1%7B%5C&%7Dtype=pdf (cit. on pp. 59, 126, 141).

Nasiriany, Soroush, Vitchyr H. Pong, Steven Lin, and Sergey Levine (2019). "Planning with Goal-Conditioned Policies." In: *NeurIPS* (cit. on pp. 175, 176, 184, 187).

Neumann, Gerhard and Jan Peters (2008). "Fitted Q-iteration by Advantage Weighted Regression." In: *Advances in Neural Information Processing Systems (NeurIPS)* (cit. on pp. 86, 260).

Ng, Andrew and Stuart Russell (2000). "Algorithms for Inverse Reinforcement Learning." In: *International Conference on Machine Learning (ICML)*. URL: http://ai.stanford.edu/%7B~%7Dang/papers/icml00-irl.pdf (cit. on p. 59).

Ng, Andrew Y., Daishi Harada, and Stuart Russell (1999). "Policy invariance under reward transformations: Theory and application to reward shaping." In: *International Conference on Machine Learning (ICML)*. URL: https://www-cs.stanford.edu/people/ang/papers/shaping-icml99.pdf (cit. on pp. 121, 125, 126).

Nguyen, Tung, Rui Shu, Tuan Pham, Hung Bui, and Stefano Ermon (June 2021). "Temporal Predictive Coding For Model-Based Planning In Latent Space." In: DOI: 10.48550/arxiv.2106.07156. URL: https://arxiv.org/abs/2106.07156v1 (cit. on p. 142).

Nielsen, Frank and Richard Nock (2010). "Entropies and cross-entropies of exponential families." In: *Image Processing (ICIP), 2010 17th IEEE International Conference on*. IEEE, pp. 3621–3624 (cit. on pp. 242, 243).

Oh, Junhyuk, Xiaoxiao Guo, Honglak Lee, Richard Lewis, and Satinder Singh (2015). "Action-Conditional Video Prediction using Deep Networks in Atari Games." In: *Advances in Neural Information Processing Systems (NeurIPS)*. URL: https://arxiv.org/pdf/1507.08750v1.pdf (cit. on p. 9).

Oord, Aaron van den, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu (June 2016). "Conditional Image Generation with PixelCNN Decoders." In: *Advances in Neural Information Processing Systems*. Neural information processing systems foundation, pp. 4797–4805. eprint: 1606.05328. URL: http://arxiv.org/abs/1606.05328 (cit. on pp. 162, 282).

Oord, Aaron van den, Oriol Vinyals, and Koray Kavukcuoglu (Nov. 2017). "Neural Discrete Representation Learning." In: *Advances in Neural Information Processing Systems*. Vol. 2017-Decem. Neural information processing systems foundation, pp. 6307–6316. eprint: 1711.00937. URL: http://arxiv.org/abs/1711.00937 (cit. on pp. 162, 282).

Oord, Aron van den, Oriol Vinyals, and Koray Kavukcuoglu (2017). "Neural Discrete Representation Learning." In: *NeurIPS* (cit. on p. 184).

OpenAI, : Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysaw Dbiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Jzefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique P. d. O. Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang (Dec. 2019). "Dota 2 with Large Scale Deep Reinforcement Learning." In: DOI: 10.48550/arxiv.1912.06680. URL: https://arxiv.org/abs/1912.06680v1 (cit. on p. 2).

Ostrovski, Georg, Marc G Bellemare, Aron Oord, and Rmi Munos (2017). "Count-Based Exploration with Neural Density Models." In: *International Conference on Machine Learning (ICML)*, pp. 2721–2730 (cit. on p. 45).

Pathak, Deepak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell (2017). "Curiosity-Driven Exploration by Self-Supervised Prediction." In: *International Conference on Machine Learning (ICML)*. IEEE, pp. 488–489. eprint: 1705.05363 (cit. on pp. 16, 45, 155, 158).

Pathak, Deepak, Parsa Mahmoudieh, Guanghao Luo, Pulkit Agrawal, Dian Chen, Yide Shentu, Evan Shelhamer, Jitendra Malik, Alexei A Efros, and Trevor Darrell (2018). "Zero-Shot Visual Imitation." In: *International Conference on Learning Representations (ICLR)*. URL: https://arxiv.org/pdf/1804.08606.pdf (cit. on pp. 9, 126).

Peng, Xue Bin, Aviral Kumar, Grace Zhang, and Sergey Levine (Sept. 2019a). "Advantage-Weighted Regression: Simple and Scalable Off-Policy Reinforcement Learning." In: eprint: 1910.00177. URL: http://arxiv.org/abs/1910.00177 (cit. on pp. 79, 82–84, 86, 88, 146, 258, 260, 265, 267).

Peng, Xue Bin, Aviral Kumar, Grace Zhang, and Sergey Levine (2019b). "Advantage-weighted regression: Simple and scalable off-policy reinforcement learning." In: (cit. on pp. 95–97, 100, 101).

Pertsch, Karl, Oleh Rybkin, Frederik Ebert, Chelsea Finn, Dinesh Jayaraman, and Sergey Levine (2020). "Long-Horizon Visual Planning with Goal-Conditioned Hierarchical Predictors." In: abs/2006.13205 (cit. on pp. 175, 176, 178, 181, 184, 187).

Peters, Jan, Katharina Mlling, and Yasemin Altn (2010). “Relative Entropy Policy Search.” In: *AAAI Conference on Artificial Intelligence*, pp. 1607–1612. URL: https://pdfs.semanticscholar.org/ff47/526838ce85d77a50197a0c5f6ee5095156aa.pdf%20http://www-clmc.usc.edu/publications/P/Peters%7B%5C_%7DPOTTNCOAIPGAT%7B%5C_%7D2010.pdf (cit. on pp. 26, 59, 82, 86, 121, 126, 157, 258).

Peters, Jan and Stefan Schaal (2007a). “Reinforcement Learning by Reward-weighted Regression for Operational Space Control.” In: *International Conference on Machine Learning (ICML)*. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.79.6266%7B%5C&%7Drep=rep1%7B%5C&%7Dtype=pdf (cit. on pp. 79, 82, 86).

Peters, Jan and Stefan Schaal (2007b). “Reinforcement learning by reward-weighted regression for operational space control.” In: *Proceedings of the 24th international conference on Machine learning*, pp. 745–750 (cit. on pp. 95, 100, 101).

Peters, Jan and Stefan Schaal (2008a). “Natural Actor-Critic.” In: 71.7-9, pp. 1180–1190. DOI: 10.1016/j.neucom.2007.11.026. URL: https://doi.org/10.1016/j.neucom.2007.11.026 (cit. on p. 85).

Peters, Jan and Stefan Schaal (2008b). “Reinforcement learning of motor skills with policy gradients.” In: 21.4. Robotics and Neuroscience, pp. 682–697. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2008.02.003. URL: http://www.sciencedirect.com/science/article/pii/S0893608008000701 (cit. on p. 26).

Peters, Jan and Stefan Schaal (2008c). “Reinforcement learning of motor skills with policy gradients.” In: 21.4, pp. 682–697. ISSN: 08936080. DOI: 10.1016/j.neunet.2008.02.003. eprint: arXiv:1411.3159v1. URL: https://pdfs.semanticscholar.org/eb5b/459c8a3e56064158fb3514eeab763486e437.pdf (cit. on pp. 26, 60, 86, 87, 122, 126, 141).

Pinto, Lerrel, Marcin Andrychowicz, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel (2018). “Asymmetric Actor Critic for Image-Based Robot Learning.” In: (cit. on p. 9).

Pinto, Lerrel, James Davidson, Rahul Sukthankar, and Abhinav Gupta (2017). “Robust Adversarial Reinforcement Learning.” In: *International Conference on Machine Learning (ICML)*. URL: https://arxiv.org/pdf/1703.02702.pdf (cit. on pp. 26, 121).

Pinto, Lerrel and Abhinav Gupta (2016). “Supersizing Self-supervision: Learning to Grasp from 50K Tries and 700 Robot Hours.” In: (cit. on pp. 9, 26, 27, 59, 126).

Plappert, Matthias, Marcin Andrychowicz, Alex Ray, Bob Mcgrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, Vikash Kumar, and Wojciech Zaremba (2018). “Multi-Goal Reinforcement Learning: Challenging

Robotics Environments and Request for Research." In: eprint: arXiv:1802.09464v2. URL: http://fetchrobotics.com/ (cit. on p. 231).

Pomerleau, Dean A (1989). "Alvinn: An autonomous land vehicle in a neural network." In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 305–313. ISBN: 1-558-60015-9. URL: http://repository.cmu.edu/compsci (cit. on pp. 59, 126).

Pong, Vitchyr, Shixiang Gu, Murtaza Dalal, and Sergey Levine (2018). "Temporal Difference Models: Model-Free Deep RL For Model-Based Control." In: *International Conference on Learning Representations (ICLR)*. URL: https://arxiv.org/pdf/1802.09081.pdf (cit. on pp. 10, 39, 45, 115, 130, 257).

Pong, Vitchyr H, Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl, and Sergey Levine (2020a). "Skew-Fit: State-Covering Self-Supervised Reinforcement Learning." In: *International Conference on Machine Learning (ICML)*. eprint: 1903.03698v2. URL: https://arxiv.org/pdf/1903.03698.pdf (cit. on pp. 141, 158).

Pong, Vitchyr H., Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl, and Sergey Levine (2019). "Skew-Fit: State-Covering Self-Supervised Reinforcement Learning." In: abs/1903.03698. eprint: 1903.03698. URL: http://arxiv.org/abs/1903.03698 (cit. on pp. 4, 24, 27, 30, 55).

Pong, Vitchyr H., Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl, and Sergey Levine (2020b). "Skew-Fit: State-Covering Self-Supervised Reinforcement Learning." In: abs/1903.03698 (cit. on p. 175).

Pong, Vitchyr H., Ashvin Nair, Laura Smith, Catherine Huang, and Sergey Levine (2022). "Offline Meta-Reinforcement Learning with Online Self-Supervision." In: (cit. on p. 141).

Ponomarenko, Nikolay, Lina Jin, Oleg Ieremeiev, Vladimir Lukin, Karen Egiazarian, Jaakko Astola, Benoit Vozel, Kacem Chehdi, Marco Carli, Federica Battisti, et al. (2015). "Image database TID2013: Peculiarities, results and perspectives." In: 30, pp. 57–77 (cit. on p. 7).

Popov, Ivaylo, Nicolas Heess, Timothy Lillicrap, Roland Hafner, Gabriel Barth-maron, Matej Vecerik, Thomas Lampe, Yuval Tassa, Tom Erez, Martin Riedmiller, and Martin Riedmiller Deepmind (2017). "Data-efficient Deep Reinforcement Learning for Dexterous Manipulation." In: abs/1704.0. eprint: 1704.03073. URL: https://arxiv.org/pdf/1704.03073.pdf (cit. on pp. 60, 121, 125).

Pr, Alexandre, Sebastien Forestier, Olivier Sigaud, and Pierre-Yves Oudeyer (2018a). "Unsupervised Learning of Goal Spaces for Intrinsically Motivated Goal Exploration." In: *International Conference on Learning Representations (ICLR)*. URL: https://arxiv.org/pdf/1803.00781.pdf (cit. on p. 10).

Pr, Alexandre, Sebastien Forestier, Olivier Sigaud, and Pierre-Yves Oudeyer (2018b). "Unsupervised Learning of Goal Spaces for Intrinsically Motivated Goal Exploration." In: *International Conference on Learning Representations (ICLR)*. URL: https://arxiv.org/pdf/1803.00781.pdf (cit. on p. 45).

Pr, Alexandre, Sebastien Forestier, Olivier Sigaud, and Pierre-Yves Oudeyer (2018c). "Unsupervised Learning of Goal Spaces for Intrinsically Motivated Goal Exploration." In: *International Conference on Learning Representations (ICLR)*. URL: https://arxiv.org/pdf/1803.00781.pdf (cit. on pp. 141, 158).

Rajeswaran, Aravind, Vikash Kumar, Abhishek Gupta, John Schulman, Emanuel Todorov, and Sergey Levine (2018). "Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations." In: *Robotics: Science and Systems*. URL: https://arxiv.org/pdf/1709.10087.pdf (cit. on pp. 78, 79, 86–88, 91, 106, 122, 127, 261, 263, 266, 267, 277).

Rakelly, Kate, Aurick Zhou, Deirdre Quillen, Chelsea Finn, and Sergey Levine (2019). "Efficient Off-Policy Meta-Reinforcement Learning via Probabilistic Context Variables." In: *International Conference on Machine Learning (ICML)*. URL: https://github.com/katerakelly/oyster. (cit. on p. 160).

Ramapuram, Jason, Magda Gregorova, and Alexandros Kalousis (May 2017). "Lifelong Generative Modeling." In: eprint: 1705.09847. URL: http://arxiv.org/abs/1705.09847 (cit. on p. 81).

Rauber, Paulo, Filipe Mutz, and Juergen Jrgen Schmidhuber (2017). "Hindsight policy gradients." In: *CoRR*. Vol. abs/1711.0. URL: https://arxiv.org/pdf/1711.06006.pdf (cit. on p. 10).

Reed, Scott, Kihyuk Sohn, Yuting Zhang, and Honglak Lee (2014). "Learning to disentangle factors of variation with manifold interaction." In: *International Conference on Machine Learning (ICML)*, pp. 1431–1439 (cit. on p. 10).

Ren, Shaoqing, Kaiming He, Ross Girshick, and Jian Sun (June 2015). "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks." In: 39 (6), pp. 1137–1149. ISSN: 01628828. DOI: 10.48550/arxiv.1506.01497. URL: https://arxiv.org/abs/1506.01497v3 (cit. on p. 2).

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox (2015). "U-Net: Convolutional Networks for Biomedical Image Segmentation." In: *MICCAI* (cit. on p. 184).

Ross, Stphane, Geoffrey J Gordon, and J Andrew Bagnell (2011). "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning." In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. URL: https :

//arxiv.org/pdf/1011.0686.pdf%20https://www.cs.cmu.edu/%7B~%7Dsross1/publications/Ross-AIStats11-NoRegret.pdf (cit. on p. 59).

Rubin, Donald B (1988). "Using the SIR algorithm to simulate posterior distributions." In: 3, pp. 395–402 (cit. on p. 41).

Rusu, Andrei A, Matej Vecerik, Thomas Rothrl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell (2017). "Sim-to-real robot learning from pixels with progressive nets." In: (cit. on p. 21).

Sacerdoti, Earl D. (1974). "Planning in a hierarchy of abstraction spaces." In: 5.2, pp. 115–135. ISSN: 0004-3702. DOI: https://doi.org/10.1016/0004-3702(74)90026-5. URL: https://www.sciencedirect.com/science/article/pii/0004370274900265 (cit. on p. 1).

Saunders, William, Girish Sastry, Andreas Stuhlmller, and Owain Evans (2018). "Trial without Error: Towards Safe Reinforcement Learning via Human Intervention." In: *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. eprint: arXiv:1707.05173v1. URL: https://arxiv.org/pdf/1707.05173.pdf (cit. on p. 122).

Savinov, Nikolay, Anton Raichuk, Raphal Marinier, Damien Vincent, Marc Pollefeys, Timothy Lillicrap, and Sylvain Gelly (2018). "Episodic curiosity through reachability." In: (cit. on p. 45).

Sax, Alexander, Bradley Emi, Amir Zamir, Leonidas Guibas, Silvio Savarese, and Jitendra Malik (2019). "Mid-Level Visual Representations Improve Generalization and Sample Efficiency for Learning Visuomotor Policies." In: *Conference on Robot Learning (CoRL)*. eprint: 1812.11971v3. URL: http://perceptual.actor/ (cit. on p. 143).

Schaal, Stefan (1997). "Learning from demonstration." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 9, pp. 1040–1046. ISBN: 1558604863. DOI: 10.1016/j.robot.2004.03.001. URL: http://www.cc.gatech.edulfac%20http://wwwiaim.ira.uka.de/users/rogalla/WebOrdnerMaterial/ml-robotlearning.pdf%20http://www.cc.gatech.edulfac/Stefan.Schaal (cit. on pp. 60, 74).

Schaul, Tom, Dan Horgan, Karol Gregor, and David Silver (2015a). "Universal Value Function Approximators." In: *ICML* (cit. on p. 175).

Schaul, Tom, Daniel Horgan, Karol Gregor, and David Silver (2015b). "Universal Value Function Approximators." In: *International Conference on Machine Learning (ICML)*, pp. 1312–1320. ISBN: 9781510810587. URL: http://proceedings.mlr.press/v37/schaul15.pdf%20http://jmlr.org/proceedings/papers/v37/schaul15.html (cit. on pp. 9, 27, 39, 45, 62, 141, 158, 159, 251, 257).

Schenck, Connor and Dieter Fox (2017). “Visual closed-loop control for pouring liquids.” In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 2629–2636 (cit. on p. 26).

Schmidhuber, Jrgen (1992). “Learning factorial codes by predictability minimization.” In: 4.6, pp. 863–879 (cit. on p. 10).

Schoettler, G., A. Nair, J.A. Ojea, S. Levine, and E. Solowjow (2020). *Meta-reinforcement learning for robotic industrial insertion tasks* (cit. on p. 142).

Schoettler, Gerrit, Ashvin Nair, Jianlan Luo, Shikhar Bahl, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine (2019). “Deep Reinforcement Learning for Industrial Insertion Tasks with Visual Inputs and Natural Rewards.” In: eprint: 1906.05841v2 (cit. on pp. 4, 137, 142).

Schulman, John, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel (2015). “Trust Region Policy Optimization.” In: *International Conference on Machine Learning (ICML)*. ISBN: 0375-9687. DOI: 10.1063/1.4927398. eprint: 1502.05477. URL: https://arxiv.org/pdf/1502.05477.pdf%20http://arxiv.org/abs/1502.05477 (cit. on pp. 2, 59).

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov Openai (2017). “Proximal Policy Optimization Algorithms.” In: URL: https://arxiv.org/pdf/1707.06347.pdf (cit. on p. 2).

Schwarzer, Max, Ankesh Anand, Rishab Goel, R Devon Hjelm, Aaron Courville, and Philip Bachman (July 2020). “Data-Efficient Reinforcement Learning with Self-Predictive Representations.” In: DOI: 10.48550/arxiv.2007.05929. URL: https://arxiv.org/abs/2007.05929v4 (cit. on p. 143).

Sermanet, Pierre, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal, and Sergey Levine (2017). “Time-Contrastive Networks: Self-Supervised Learning from Video.” In: (cit. on p. 10).

Sharma, Archit, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn (2021). “Autonomous Reinforcement Learning via Subgoal Curricula.” In: (cit. on p. 175).

Siegel, Noah Y, Jost Tobias Springenberg, Felix Berkenkamp, Abbas Abdolmaleki, Michael Neunert, Thomas Lampe, Roland Hafner, Nicolas Heess, and Martin Riedmiller (2020a). “Keep doing what worked: Behavioral modelling priors for offline reinforcement learning.” In: (cit. on pp. 95, 279).

Siegel, Noah Y., Jost Tobias Springenberg, Felix Berkenkamp, Abbas Abdolmaleki, Michael Neunert, Thomas Lampe, Roland Hafner, Nicolas Heess, and Martin Riedmiller (2020b). *Keep Doing What Worked: Behavioral Modelling Priors for Offline Reinforcement Learning*. eprint: 2002.08396 (cit. on pp. 80, 82, 84–86, 88, 260, 265, 268).

Silver, David, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis (Jan. 2016a). "Mastering the game of Go with deep neural networks and tree search." In: 529.7587, pp. 484–489. ISSN: 0028-0836. DOI: 10.1038/nature16961. eprint: 1610.00633. URL: http://dx.doi.org/10.1038/nature16961%20http://10.0.4.14/nature16961%20http://www.nature.com/nature/journal/v529/n7587/abs/nature16961.html%7B%5C#%7Dsupplementary-information%20http://airesearch.com/wp-content/uploads/2016/01/deepmind-mastering-go.pdf (cit. on pp. 2, 26, 157).

Silver, David, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis (Jan. 2016b). "Mastering the game of Go with deep neural networks and tree search." In: 529.7587, pp. 484–489. ISSN: 0028-0836. DOI: 10.1038/nature16961. eprint: 1610.00633. URL: http://dx.doi.org/10.1038/nature16961%20http://10.0.4.14/nature16961%20http://www.nature.com/nature/journal/v529/n7587/abs/nature16961.html%7B%5C#%7Dsupplementary-information%20http://airesearch.com/wp-content/uploads/2016/01/deepmind-mastering-go.pdf (cit. on p. 59).

Silver, Tom, Kelsey Allen, Josh Tenenbaum, and Leslie Kaelbling (2018). "Residual Policy Learning." In: eprint: 1812.06298. URL: http://arxiv.org/abs/1812.06298 (cit. on pp. 122, 125, 127).

Singh, Avi, Huihan Liu, Gaoyue Zhou, Albert Yu, Nicholas Rhinehart, and Sergey Levine (2020a). "Parrot: Data-Driven Behavioral Priors for Reinforcement Learning." In: URL: https://sites.google.com/view/parrot-rl. (cit. on p. 141).

Singh, Avi, Larry Yang, Kristian Hartikainen, Chelsea Finn, and Sergey Levine (Apr. 2019). "End-to-End Robotic Reinforcement Learning without Reward Engineering." In: *Robotics: Science and Systems (RSS)*. eprint: 1904.07854. URL: http://arxiv.org/abs/1904.07854 (cit. on pp. 127, 142).

Singh, Avi, Albert Yu, Jonathan Yang, Jesse Zhang, Aviral Kumar, and Sergey Levine (2020b). “COG: Connecting new skills to past experience with offline reinforcement learning.” In: *CoRL* (cit. on pp. 141, 174).

Smith, Linda and Michael Gasser (2005). “The development of embodied cognition: Six lessons from babies.” In: 11.1-2, pp. 13–29. URL: https://www.cogsci.msu.edu/DSS/2010-2011/Smith/6lessons.pdf (cit. on p. 7).

Sohn, Kihyuk, Xinchen Yan, and Honglak Lee (2015). “Learning Structured Output Representation using Deep Conditional Generative Models.” In: *Advances in Neural Information Processing Systems (NeurIPS)*. URL: https://papers.nips.cc/paper/5775-learning-structured-output-representation-using-deep-conditional-generative-models.pdf (cit. on pp. 28, 158, 162, 165, 169, 174, 179, 282).

Spector, Oren and Dotan Di Castro (July 2021). “InsertionNet - A Scalable Solution for Insertion.” In: 6.3, pp. 5509–5516. DOI: 10.1109/LRA.2021.3076971 (cit. on p. 142).

Srinivas, Aravind, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn (2018). “Universal Planning Networks.” In: *International Conference on Machine Learning (ICML)*. eprint: arXiv:1804.00645v2. URL: https://sites.google. (cit. on p. 10).

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov (2014). “Dropout: a simple way to prevent neural networks from overfitting.” In: 15.1, pp. 1929–1958 (cit. on p. 274).

Srivastava, Siddharth, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stuart Russell, and Pieter Abbeel (2014). “Combined Task and Motion Planning Through an Extensible Planner-Independent Interface Layer.” In: *IEEE International Conference on Robotics and Automation (ICRA)*. URL: https://people.eecs.berkeley.edu/%7B~%7Drussell/papers/icra14-planrob.pdf (cit. on pp. 60, 175).

Srouji, Mario, Jian Zhang, and Ruslan Salakhutdinov (2018). “Structured Control Nets for Deep Reinforcement Learning.” In: *International Conference on Machine Learning (ICML)*. eprint: 1802.08311. URL: http://arxiv.org/abs/1802.08311 (cit. on p. 122).

Stadie, Bradly C, Sergey Levine, and Pieter Abbeel (2016). “Incentivizing Exploration In Reinforcement Learning With Deep Predictive Models.” In: *International Conference on Learning Representations (ICLR)*. URL: https://arxiv.org/pdf/1507.00814.pdf (cit. on pp. 45, 158).

Subramanian, Kaushik, Charles L Isbell, and Andrea L Thomaz (2016). “Exploration from Demonstration for Interactive Reinforcement Learning.” In: *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. URL: www.ifaamas.org (cit. on p. 122).

Sun, Baochen and Kate Saenko (July 2016). "Deep CORAL: Correlation Alignment for Deep Domain Adaptation." In: URL: http://arxiv.org/abs/1607.01719 (cit. on p. 143).

Sun, Hao, Zhizhong Li, Xiaotong Liu, Dahua Lin, and Bolei Zhou (2019). "Policy Continuation with Hindsight Inverse Dynamics." In: *NeurIPS* (cit. on p. 175).

Sutton, Richard (2019). *The Bitter Lesson*. URL: http://www.incompleteideas.net/IncIdeas/BitterLesson.html (cit. on p. 190).

Sutton, Richard S and Andrew G Barto (1998). *Reinforcement Learning: An Introduction*. URL: http://incompleteideas.net/sutton/book/bookdraft2016sep.pdf%20https://webdocs.cs.ualberta.ca/%7B~%7Dsutton/book/bookdraft2016sep.pdf (cit. on pp. 77, 80).

Sutton, Richard S and Andrew G Barto (2018). *Reinforcement learning: An introduction*. MIT press (cit. on p. 102).

Sutton, Richard S, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M Pilarski, Adam White, and Doina Precup (2011). "Horde: A Scalable Real-time Architecture for Learning Knowledge from Unsupervised Sensorimotor Interaction." In: *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. Vol. 10, pp. 761–768. URL: https://www.cs.swarthmore.edu/%7B~%7Dmeeden/DevelopmentalRobotics/horde1.pdf (cit. on p. 9).

Sutton, Richard S, Doina Precup, and Satinder Singh (1999). "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning." In: 112.1-2, pp. 181–211 (cit. on pp. 45, 49, 252).

Tamar, Aviv, Garrett Thomas, Tianhao Zhang, Sergey Levine, and Pieter Abbeel (2017). "Learning from the hindsight plan Episodic MPC improvement." In: *ICRA*, pp. 336–343 (cit. on p. 127).

Tang, Haoran, Rein Houthooft, Davis Foote, Adam Stooke, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel (2017). "#Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning." In: *Advances in Neural Information Processing Systems (NeurIPS)*. eprint: arXiv:1611.04717v3. URL: https://arxiv.org/pdf/1611.04717.pdf (cit. on pp. 45, 50, 155).

Theodorou, Evangelos A, Jonas Buchli, Jonas@buchli Org, Stefan Schaal, and Sschaal@usc Edu (2010). "A Generalized Path Integral Control Approach to Reinforcement Learning." In: 11, pp. 3137–3181. URL: http://www.jmlr.org/papers/volume11/theodorou10a/theodorou10a.pdf (cit. on p. 86).

Thomas, Garrett, Melissa Chien, Aviv Tamar, Juan Aparicio Ojea, and Pieter Abbeel (2018). "Learning Robotic Assembly from CAD." In: *IEEE International Conference on*

Robotics and Automation (ICRA). URL: https://arxiv.org/pdf/1803.07635.pdf (cit. on pp. 122, 126).

Thomas, Philip S. and Emma Brunskill (2016). "Data-Efficient Off-Policy Policy Evaluation for Reinforcement Learning." In: *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*. Ed. by Maria-Florina Balcan and Kilian Q. Weinberger. Vol. 48. JMLR Workshop and Conference Proceedings. JMLR.org, pp. 2139–2148. URL: http://proceedings.mlr.press/v48/thomasa16.html (cit. on p. 85).

Thomas, Valentin, Jules Pondard, Emmanuel Bengio, Marc Sarfati, Philippe Beaudoin, Marie-Jean Meurs, Joelle Pineau, Doina Precup, Yoshua Bengio, Valentin Thoma, Joelle Pineau, Doina Precup, and Yoshua Bengio (2017). "Independently Controllable Factors." In: *NIPS Workshop*. URL: https://arxiv.org/pdf/1703.07718.pdf%20https://arxiv.org/pdf/1708.01289.pdf (cit. on p. 10).

Todorov, Emanuel, Tom Erez, and Yuval Tassa (2012). "MuJoCo: A physics engine for model-based control." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5026–5033. ISBN: 9781467317375. DOI: 10.1109/IROS.2012.6386109. URL: https://homes.cs.washington.edu/%7B~%7Dtodorov/papers/TodorovIROS12.pdf (cit. on pp. 17, 31, 64, 88, 115, 256).

Torrey, Lisa and Matthew E Taylor (2013). "Teaching on a Budget: Agents Advising Agents in Reinforcement Learning." In: *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. URL: www.ifaamas.org (cit. on p. 122).

Tzeng, Eric, Judy Hoffman, Ning Zhang, Kate Saenko, and Trevor Darrell (Dec. 2014). "Deep Domain Confusion: Maximizing for Domain Invariance." In: URL: http://arxiv.org/abs/1412.3474 (cit. on p. 143).

Van Hasselt, Hado, Arthur Guez, and David Silver (2016). "Deep Reinforcement Learning with Double Q-learning." In: *Association for the Advancement of Artificial Intelligence (AAAI)*. ISBN: 1509.06461v3. eprint: 1509.06461v3. URL: www.aaai.org (cit. on p. 130).

Veeriah, Vivek, Junhyuk Oh, and Satinder Singh (2018). "Many-goals reinforcement learning." In: (cit. on p. 46).

Veerk, Matej, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothrl, Thomas Lampe, and Martin Riedmiller (2017). "Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards." In: abs/1707.0. URL: https://arxiv.org/pdf/1707.08817.pdf (cit. on pp. 57, 60, 63, 86, 122, 268).

Villaflor, Adam, John Dolan, and Jeff Schneider (2020). "Fine-tuning Offline Reinforcement Learning With Model-Based Policy Optimization." In: (cit. on pp. 141, 174).

Vinyals, Oriol et al. (Oct. 2019). "Grandmaster level in StarCraft II using multi-agent reinforcement learning." In: 575 (7782), pp. 350–354. ISSN: 1476-4687. DOI: 10.1038/s41586-019-1724-z. URL: https://www.nature.com/articles/s41586-019-1724-z (cit. on p. 2).

Wang, Qing, Jiechao Xiong, Lei Han, Peng Sun, Han Liu, and Tong Zhang (2018a). "Exponentially Weighted Imitation Learning for Batched Historical Data." In: *Neural Information Processing Systems (NeurIPS)* (cit. on pp. 79, 86, 260, 265, 267).

Wang, Qing, Jiechao Xiong, Lei Han, Peng Sun, Han Liu, and Tong Zhang (2018b). "Exponentially Weighted Imitation Learning for Batched Historical Data." In: *NeurIPS*, pp. 6291–6300 (cit. on pp. 101, 103).

Wang, Zhongkui, Keung Or, and Shinichi Hirai (2020). "A dual-mode soft gripper for food packaging." In: 125, p. 103427. ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2020.103427. URL: https://www.sciencedirect.com/science/article/pii/S0921889019300879 (cit. on p. 1).

Wang, Ziyu, Alexander Novikov, Konrad Zolna, Jost Tobias Springenberg, Scott Reed, Bobak Shahriari, Noah Siegel, Josh Merel, Caglar Gulcehre, Nicolas Heess, et al. (2020a). "Critic regularized regression." In: (cit. on pp. 94, 95).

Wang, Ziyu, Alexander Novikov, Konrad Zona, Jost Tobias Springenberg, Scott Reed, Bobak Shahriari, Noah Siegel, Josh Merel, Caglar Gulcehre, Nicolas Heess, and Nando De Freitas (2020b). "Critic Regularized Regression." In: eprint: 2006.15134v1 (cit. on p. 86).

Warde-Farley, David, Tom Van De Wiele, Tejas Kulkarni, Catalin Ionescu, Steven Hansen, and Mnih Volodymyr (2019). "Unsupervised Control Through Non-Parametric Discriminative Rewards." In: *International Conference on Learning Representations (ICLR)*. ISBN: 1811.11359v1. eprint: 1811.11359v1. URL: https://arxiv.org/pdf/1811.11359.pdf (cit. on pp. 24, 27, 141, 158).

Warde-Farley, David, Tom Van de Wiele, Tejas Kulkarni, Catalin Ionescu, Steven Hansen, and Volodymyr Mnih (2018). "Unsupervised Control Through Non-Parametric Discriminative Rewards." In: abs/1811.11359 (cit. on pp. 45, 50, 52, 257).

Watter, Manuel, Jost Tobias Springenberg, Joschka Boedecker, and Martin Riedmiller (2015). "Embed to Control: A Locally Linear Latent Dynamics Model for Control from Raw Images." In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 2728–2736. eprint: 1506.07365. URL: https://arxiv.org/pdf/1506.07365.pdf%20http://arxiv.org/abs/1506.07365 (cit. on pp. 9, 10, 27).

Wawrzynski, Pawel (2009). "Real-time reinforcement learning by sequential Actor-Critics and experience replay." In: 22.10, pp. 1484–1497. DOI: 10.1016/j.neunet.2009.05.011. URL: https://doi.org/10.1016/j.neunet.2009.05.011 (cit. on p. 85).

Williams, Grady, Nolan Wagener, Brian Goldfain, Paul Drews, James M Rehg, Byron Boots, and Evangelos A Theodorou (2017). "Information Theoretic MPC for Model-Based Reinforcement Learning." In: *IEEE International Conference on Robotics and Automation (ICRA)*. URL: http://www.cc.gatech.edu/%7B~%7Dbboots3/files/InformationTheoreticMPC.pdf (cit. on p. 157).

Williams, Ronald J (1992). "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning." In: pp. 229–256 (cit. on p. 77).

Winograd, Terry (1972). *Understanding Natural Language*. Academic Press (cit. on p. 60).

Wu, Yifan, George Tucker, and Ofir Nachum (2019). "Behavior regularized offline reinforcement learning." In: (cit. on pp. 94, 95, 279).

Wu, Yifan, George Tucker, and Ofir Nachum (Nov. 2020). "Behavior Regularized Offline Reinforcement Learning." In: *International Conference on Learning Representations (ICLR)*. eprint: 1911.11361. URL: http://arxiv.org/abs/1911.11361 (cit. on pp. 80–83, 85, 141, 264, 268).

Wu, Yuchen, Melissa Mozifian, and Florian Shkurti (2020). *Shaping Rewards for Reinforcement Learning with Imperfect Demonstrations using Generative Models*. eprint: 2011.01298 (cit. on p. 87).

Xu, Danfei, Ajay Mandlekar, Roberto Martn-Martn, Yuke Zhu, Silvio Savarese, and Li Fei-Fei (2021). "Deep Affordance Foresight: Planning Through What Can Be Done in the Future." In: *International Conference on Robotics and Automation (ICRA)*. eprint: 2011.08424v1. URL: https://sites.google.com/stanford.edu/daf (cit. on p. 158).

Yamanobe, Natsuki, Weiwei Wan, Ixchel G. Ramirez-Alpizar, Damien Petit, Tokuo Tsuji, Shuichi Akizuki, Manabu Hashimoto, Kazuyuki Nagata, and Kensuke Harada (Jan. 2018). "A Brief Review of Affordance in Robotic Manipulation Research." In: 36, pp. 327–337. DOI: 10.7210/jrsj.36.327 (cit. on p. 158).

Zech, Philipp, Simon Haller, Safoura Rezapour Lakani, Barry Ridge, Emre Ugur, and Justus Piater (2017). "Computational models of affordance in robotics: a taxonomy and systematic classification." In: 25.5, pp. 235–271. DOI: 10.1177/1059712317726357. eprint: https://doi.org/10.1177/1059712317726357. URL: https://doi.org/10.1177/1059712317726357 (cit. on p. 158).

Zhang, Amy, Rowan Thomas McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine (2021). "Learning Invariant Representations for Reinforcement Learning without Reconstruction." In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=-2FCwDKRREu (cit. on p. 142).

Zhang, Chi, Sanmukh Rao Kuppannagari, and Viktor Prasanna (2021). *{BRAC}+: Going Deeper with Behavior Regularized Offline Reinforcement Learning*. URL: https://openreview.net/forum?id=bMCfFepJXM (cit. on p. 174).

Zhang, Marvin, Sharad Vikram, Laura Smith, Pieter Abbeel, Matthew J. Johnson, and Sergey Levine (Aug. 2019). "SOLAR: Deep Structured Representations for Model-Based Reinforcement Learning." In: *International Conference on Machine Learning (ICML)*. eprint: 1808.09105. URL: http://arxiv.org/abs/1808.09105 (cit. on pp. 27, 126).

Zhang, Richard, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang (2018). "The Unreasonable Effectiveness of Deep Features as a Perceptual Metric." In: (cit. on p. 8).

Zhang, Ruiyi, Bo Dai, Lihong Li, and Dale Schuurmans (Feb. 2020). "GenDICE: Generalized Offline Estimation of Stationary Values." In: *International Conference on Learning Representations (ICLR)*. eprint: 2002.09072. URL: http://arxiv.org/abs/2002.09072 (cit. on p. 85).

Zhang, Shangtong, Wendelin Boehmer, and Shimon Whiteson (2019). "Generalized Off-Policy Actor-Critic." In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. dlch-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., pp. 2001–2011. URL: http://papers.nips.cc/paper/8474-generalized-off-policy-actor-critic.pdf (cit. on p. 85).

Zhang, Tianjun, Benjamin Eysenbach, Ruslan Salakhutdinov, Sergey Levine, and Joseph E. Gonzalez (2021). "C-Planning: An Automatic Curriculum for Learning Goal-Reaching Tasks." In: abs/2110.12080 (cit. on p. 175).

Zhao, Rui, Xudong Sun, and Volker Tresp (2019). "Maximum Entropy-Regularized Multi-Goal Reinforcement Learning." In: *International Conference on Machine Learning*, pp. 7553–7562 (cit. on p. 50).

Zhao, Rui and Volker Tresp (2019). "Curiosity-Driven Experience Prioritization via Density Estimation." In: abs/1902.08039. eprint: 1902.08039. URL: http://arxiv.org/abs/1902.08039 (cit. on p. 46).

Zhao, Tony Z., Jianlan Luo, Oleg Sushkov, Rugile Pevceviciute, Nicolas Heess, Jon Scholz, Stefan Schaal, and Sergey Levine (Oct. 2022). "Offline Meta-Reinforcement Learning for Industrial Insertion." In: DOI: 10.48550/arxiv.2110.04276. URL: https://arxiv.org/abs/2110.04276v3 (cit. on p. 142).

Zhou, Allan, Eric Jang, Daniel Kappler, Alexander Herzog, Mohi Khansari, Paul Wohlhart, Yunfei Bai, Mrinal Kalakrishnan, Sergey Levine, and Chelsea Finn (2019). "Watch, Try, Learn: Meta-Learning from Demonstrations and Reward." In: abs/1906.03352. eprint: 1906.03352. URL: http://arxiv.org/abs/1906.03352 (cit. on p. 74).

Zhu, Henry, Abhishek Gupta, Aravind Rajeswaran, Sergey Levine, and Vikash Kumar (Oct. 2019). "Dexterous Manipulation with Deep Reinforcement Learning: Efficient, General, and Low-Cost." In: *IEEE International Conference on Robotics and Automation (ICRA)*. Vol. 2019-May. Institute of Electrical and Electronics Engineers Inc., pp. 3651–3657. eprint: 1810.06045. URL: http://arxiv.org/abs/1810.06045 (cit. on pp. 86, 87, 126, 141).

Zhu, Yuke, Josiah Wong, Ajay Mandlekar, and Roberto Martn-Martn (2020). "robosuite: A Modular Simulation Framework and Benchmark for Robot Learning." In: *arXiv preprint arXiv:2009.12293* (cit. on p. 281).

Ziebart, Brian D, Andrew Maas, J Andrew Bagnell, and Anind K Dey (2008). "Maximum Entropy Inverse Reinforcement Learning." In: *AAAI Conference on Artificial Intelligence*, pp. 1433–1438. ISBN: 9781577353683 (ISBN). eprint: arXiv:1507.04888v2. URL: https://www.aaai.org/Papers/AAAI/2008/AAAI08-227.pdf%20http://www.scopus.com/inward/record.url?eid=2-s2.0-57749097473%7B%5C&%7DpartnerID=40%7B%5C%%7D5Cnhttp://www.aaai.org/Papers/AAAI/2008/AAAI08-227.pdf (cit. on pp. 59, 87, 127).

Zucker, Matt, Nathan Ratliff, Anca D Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M Dellin, J Andrew Bagnell, and Siddhartha S Srinivasa (2013). "CHOMP: Covariant Hamiltonian Optimization for Motion Planning." In: (cit. on pp. 1, 175).

A

APPENDIX: CHAPTER 2

Supplementary Material

A.1 COMPLETE ABLATIVE RESULTS

A.1.1 *Relabeling strategy ablation*

In this experiment, we compare different goal resampling strategies for training the Q function. We consider: *Future*, relabeling the goal for a transition by sampling uniformly from future states in the trajectory as done in Andrychowicz et al. (2017b); *VAE*, sampling goals from the VAE only; *RIG*, relabeling goals with probability 0.5 from the VAE and probability 0.5 using the future strategy; and *None*, no relabeling. Figure 61 shows the effect of different relabeling strategies with our method.

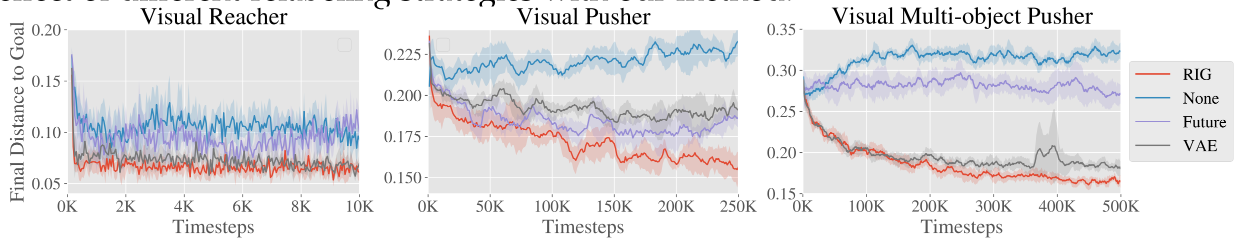


Figure 61: Relabeling ablation simulated results, showing final distance to goal vs environment steps. RIG (red), which uses a mixture of VAE and future, consistently matches or outperforms the other methods.

A.1.2 *Reward type ablation*

In this experiment, we change only the reward function that we use to train the goal-conditioned valued function to show the effect of using the latent distance reward. We include the following methods for comparison: *Latent Distance*, which is the reward used in RIG, i.e. $A = \mathbf{I}$ in Equation equation 5; *Log Probability*, which uses the Mahalanobis distance in Equation equation 5, where A is the precision matrix of the encoder; and *Pixel MSE*, which computes mean-squared error (MSE) between state and goal in pixel space. To compute the pixel MSE for a sampled latent goal, we decode the goal latent using the VAE decoder, p_ψ, to generate the corresponding goal image. Figure 62 shows the effect of different rewards with our method.

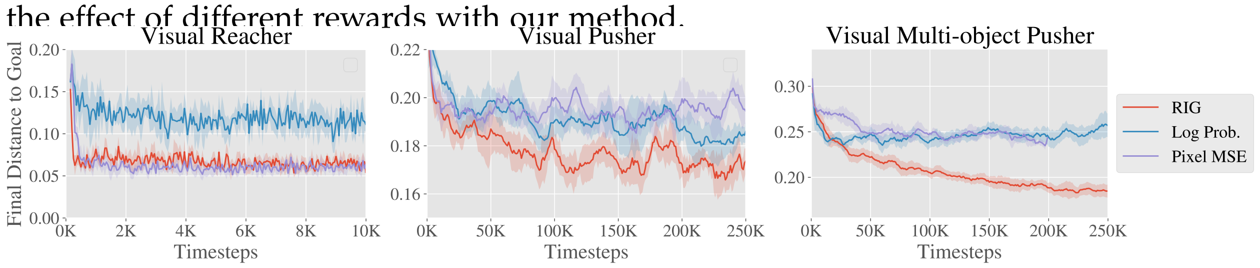


Figure 62: Reward type ablation simulated results, showing final distance to goal vs environment steps. RIG (red), which uses latent distance for the reward, consistently matches or outperforms the other reward types.

A.1.3 *Online training ablation*

Rather than pre-training the VAE on a set of images collected by a random policy, here we train the VAE in an online manner: the VAE is not trained when we initially collect data with our policy. After every 3000 environment steps, we train the VAE on all of the images observed by the policy. We show in Figure 63 that this online training results in a good policy and is substantially better than leaving the VAE untrained. These results show that the representation learning can be done simultaneously as the reinforcement learning portion of RIG, eliminating the need to have a predefined set of images to train the VAE.

The Visual Pusher experiment for this ablation is performed on a slightly easier version of the Visual Pusher used for the main results. In particular, the goal space is reduced to be three quarters of its original size in the lateral dimension.

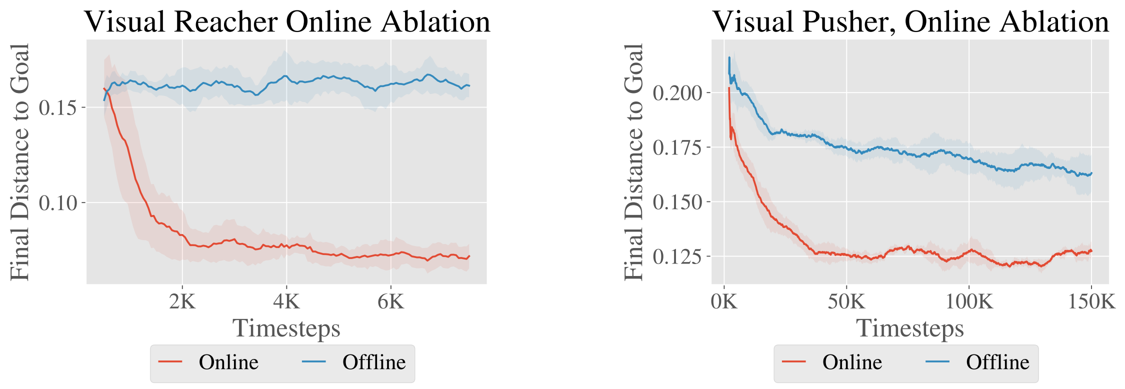


Figure 63: Online vs offline VAE training ablation simulated results, showing final distance to goal vs environment steps. Given no pre-training phase, training the VAE online (red), outperforms no training of the VAE, and also performs well.

A.1.4 *Comparison to Hindsight Experience Replay*

In this section, we study in isolation the effect of sampling goals from the goal space directly for Q-learning, as covered in Section 2.4.3. Like hindsight experience replay Andrychowicz et al., 2017b, in this section we assume access to state information and the goal space, so we do not use a VAE.

To match the original work as closely as possible, this comparison was based off of the OpenAI baselines code Plappert et al., 2018 and we compare on the same Fetch robotics tasks. To minimize sample complexity and due to computational constraints, we use single threaded training with `rollout_batch_size=1, n_cycles=1, batch_size=256`. For testing, `n_test_rollouts=1` and the results are averaged over the last 100 test episodes. Number of updates per cycle corresponds to `n_batches`.

On the plots, "Future" indicates the future strategy as presented in Andrychowicz et al. (2017b) with $k = 4$. "Ours" indicates resampling goals with probability 0.5 from the "future" strategy with $k = 4$ and probability 0.5 uniformly from the environment goal space. Each method is shown with dense and sparse rewards.

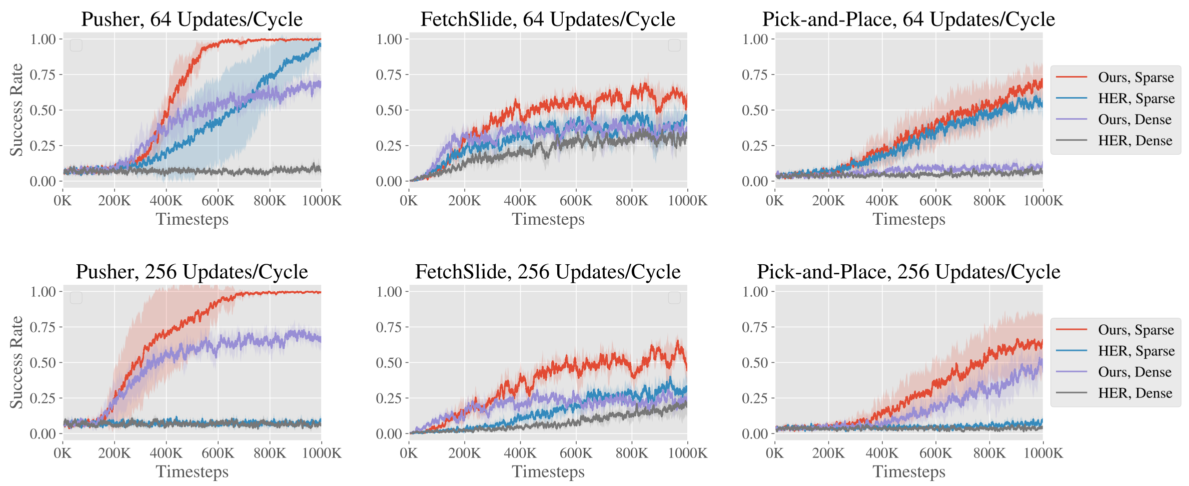


Figure 64: Comparison between our relabeling strategy and HER. Each column shows a different task from the OpenAI Fetch robotics suite. The top row uses 64 gradient updates per training cycle and the bottom row uses 256 updates per cycle. Our relabeling strategy is significantly better for both sparse and dense rewards, and for higher number of updates per cycle.

Results are shown in Figure 64. Our resampling strategy with sparse rewards consistently performs the best on the three tasks. Furthermore, it performs reasonably well with dense rewards, unlike HER alone which often fails with dense rewards. While the evaluation metric used here, success rate, is favorable to the sparse reward setting, learning with dense rewards is usually more sample efficient on most tasks and being able to do off-policy goal relabeling with dense rewards is important for RIG.

Finally, as the number of gradient updates per training cycle is increased, the performance of our strategy improves while HER does not improve and sometimes performs worse. As we apply reinforcement learning to real-world tasks, being able to reduce the required number of samples on hardware is one of the key bottlenecks. Increasing the number of gradient updates costs more compute but reduces the number of samples required to learn the tasks.

A.2 HYPERPARAMETERS

Table 10 lists the hyperparameters used for the experiments.

A.3 Environment Details

Below we provide a more detailed description of the simulated environments.

Visual Reacher: A MuJoCo environment with a 7-DoF Sawyer arm reaching goal positions. The arm is shown on the left of Figure 2 with two extra objects for the Visual Multi-Object Pusher environment (see below). The end-effector (EE) is constrained to a 2-dimensional rectangle parallel to a table. The action controls EE velocity within a maximum velocity. The underlying state is the EE position e, and the underlying goal is to reach a desired EE position, g_e.

Visual Pusher: A MuJoCo environment with a 7-DoF Sawyer arm and a small puck on a table that the arm must push to a target push. Control is the same as in Visual Reacher. The underlying state is the EE position, e and puck position p. The underlying goal is for the EE to reach a desired position g_e and the puck to reach a desired position p.

Visual Multi-Object Pusher: A copy of the Visual Pusher environment with two pucks. The underlying state is the EE position, e and puck positions p_1 and p_2. The underlying goal is for the EE to reach desired position g_e and the pucks to reach desired positions g_1 and g_2 respectively also constrained to each half of the workspace. Each puck and respective goal is initialized in half of the workspace.

Videos of our method in simulated and real-world environments can be found at `https://sites.google.com/site/visualrlwithimaginedgoals/`.

Hyperparameter	**Value**	**Comments**
Mixture coefficient λ	0.5	See relabeling strategy ablation
# training batches per time step	4	Marginal improvements after 4
Exploration Policy	OU, $\theta = 0.15, \sigma = 0.3$	Outperformed Gaussian and ϵ-greedy
β for β-VAE	5	Values around $[1, 10]$ were effective
Critic Learning Rate	10^{-3}	Did not tune
Critic Regularization	None	Did not tune
Actor Learning Rate	10^{-3}	Did not tune
Actor Regularization	None	Did not tune
Optimizer	Adam	Did not tune
Target Update Rate (τ)	10^{-2}	Did not tune
Target Update Period	2 time steps	Did not tune
Target Policy Noise	0.2	Did not tune
Target Policy Noise Clip	0.5	Did not tune
Batch Size	128	Did not tune
Discount Factor	0.99	Did not tune
Reward Scaling	10^{-4}	Did not tune
Normalized Observations	False	Did not tune
Gradient Clipping	False	Did not tune

Table 10: Hyper-parameters used for all experiments.

B

APPENDIX: CHAPTER 3

B.1 MULTI-COLOR 2D NAVIGATION EXPERIMENTS

In order to study generalizing to varying appearance and dynamics with CC-RIG, we introduced the multi-color 2D point navigation environment shown in 65. The goal is to navigate a point robot around the central walls. The arrangement of the walls is randomly chosen from a set of 15 possible configurations in each rollout, and the color of the circle indicating the position of the point robot is generated from a random RGB value. Thus at test time, the agent sees new colors it has never trained on.

First, we see from the samples in Figure 65 that the learned latent space for the CC-VAE is more reasonable than a VAE: it preserves color and wall information in samples and represents only the colored circle position in the latent variable z_t. This improves the capability of our algorithm to learn in several ways: it provides a more informative reward function, and gives us better goal sampling for both exploration rollouts and experience relabeling when training the Q function.

Learning curves obtained by training the different methods above in this environment are presented in the main paper Figure 73. This task is trivial for the oracle method to learn, as it directly receives state information and does not need to generalize between different object appearances. CC-RIG requires more samples to learn, but eventually approaches the oracle performance. RIG plateaus with poor performance in comparison.

B.2 OFF-POLICY EXPERIMENTS

Because we use off-policy RL methods, one major benefit is that we can bootstrap training from large interaction datasets rather than requiring on on-policy data collection. This is particularly vital in the real-world, where on-policy data collection is expensive in terms of human effort, and repeatedly tuning on-policy methods for complex tasks is likely to be impractical. Our robot experiments are therefore run by starting with a fixed initial dataset of 50,000 samples (about 3 hours) of random interaction with 20 objects, which is used for both training the CC-VAE as well as RL. Our simulated experiments are conducted with online data-collection to make comparison with prior work clearer, but in this section we show that bootstrapping with off-policy training is possible in these settings as well.

In our simulated experiments, we first collect 100,000 samples (1000 trajectories) with random actions. This data is used both to train the CC-VAE and as off-policy data. When we begin RL, we load these samples into the replay buffer and perform 100,000 gradient updates of RL. As shown in Figure 66, this allows us to begin online data collection with a reasonably good policy. But we see that online data collection does improve slightly beyond this initial policy. In dynamically sensitive environments or environments where random actions do not provide meaningful interaction, this online data collection may still be very valuable.

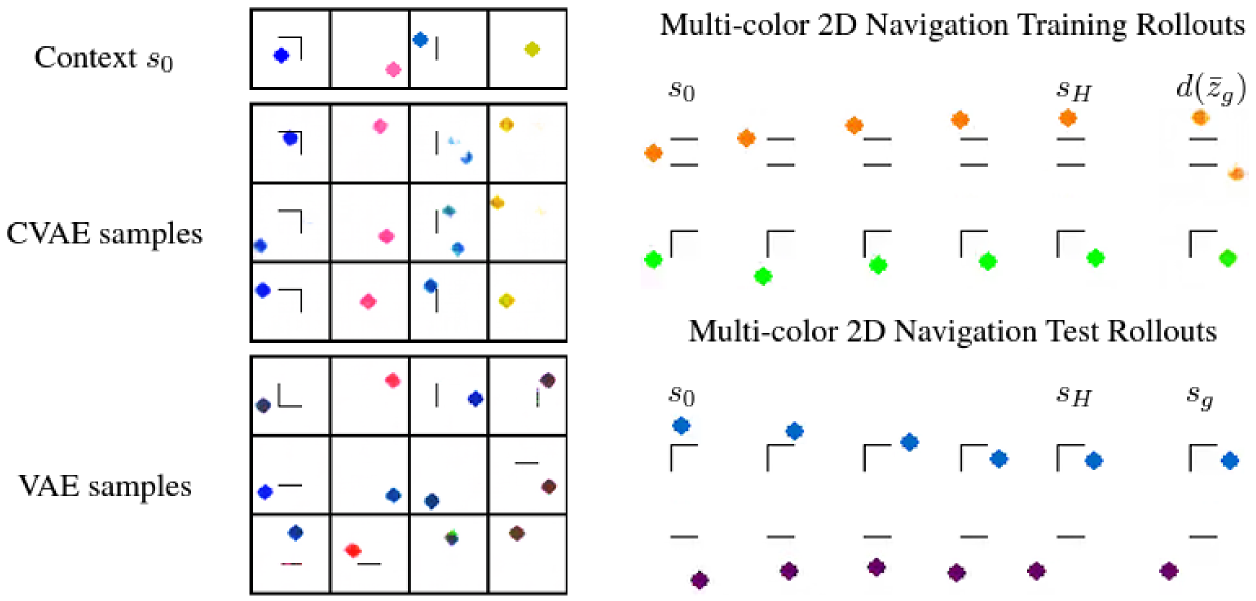
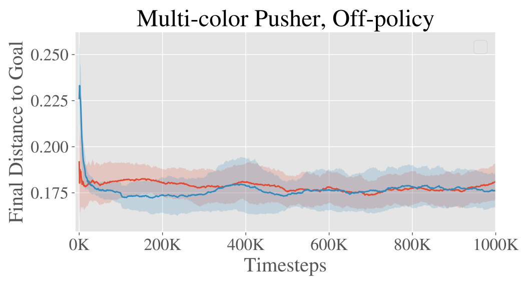

Context s_0
CVAE samples
VAE samples
Multi-color 2D Navigation Training Rollouts
s_0
s_H
$d(\bar{z}_g)$
Multi-color 2D Navigation Test Rollouts
s_0
s_H
s_g

Figure 65: The pointmass environment is shown. Left, we compare samples from our CC-VAE model to a standard VAE. The initial image s_0 is shown on the top row, and samples conditioned on s_0 are shown below. Our model coherently maintains object color and geometry in its samples, suggesting that the context conditioned model can successfully factor out the scene-specific object identity from the variable object position. This enables the use of the CC-VAE for goal proposals in visually diverse scenes. Right, rollouts from CC-RIG are shown. We see that during collecting training rollouts, the policy succeeds in coherent exploration by generating reachable goals. Then, given a goal image at test time, the policy successfully reaches the goal. Videos of rollouts on all environments, both simulation and real-world, can be found at `https://ccrig.github.io/`



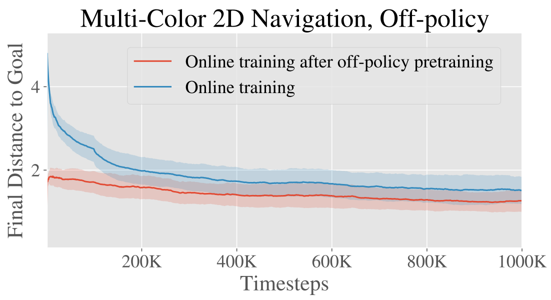


Figure 66: Off-policy learning results in simulated environments. These experiments begin with 100K samples from random interaction loaded into the replay buffer. The red lines (off-policy pre-training) additionally do 100K steps of Q learning before starting on-line data collection. We see that off-policy pre-training results in proficient initial performance from a fixed dataset, which is extremely useful in domains such as robotics where collecting new samples is expensive.

C

APPENDIX: CHAPTER 4

C.1 PROOFS

The definitions of continuity and convergence for pseudo-metrics are similar to those for metrics, and we state them below.

A function $f : \mathcal{Q} \mapsto \mathcal{Q}$ is continuous with respect to a pseudo-metric d if for any $p \in \mathcal{Q}$ and any scalar $\epsilon > 0$, there exists a δ such that for all $q \in \mathcal{Q}$,

$$d(p, q) < \delta \implies d(f(p), f(q)) < \epsilon.$$

An infinite sequence $p_1, p_2 \ldots$ converges to a value p with respect to a pseudo-metric d, which we write as

$$\lim_{t \to \infty} p_t \to p,$$

if

$$\lim_{t \to \infty} d(p_t, p) \to 0.$$

Note that if f is continuous, then

$$\lim_{t \to \infty} d_{\mathcal{H}}(p_t, q) \to 0 \implies \lim_{t \to \infty} d_{\mathcal{H}}(f(p_t), f(q)) \to 0.$$

C.1.1 *Proof of Lemma 3.1*

Lemma 5. *Let $\mathcal{S}$ be a compact set. Define the set of distributions $\mathcal{Q} = \{p : support(p) \subseteq \mathcal{S}\}$.*

Let $\mathcal{F} : \mathcal{Q} \mapsto \mathcal{Q}$ be continuous with respect to the pseudometric $d_{\mathcal{H}}(p, q) \triangleq |\mathcal{H}(p) - \mathcal{H}(q)|$ and $\mathcal{H}(\mathcal{F}(p)) \geqslant \mathcal{H}(p)$ with equality if and only if p is the uniform probability distribution on $\mathcal{S}$, denoted as $U_{\mathcal{S}}$. Define the sequence of distributions $P = (p_1, p_2, \dots)$ by starting with any $p_1 \in \mathcal{Q}$ and recursively defining $p_{t+1} = \mathcal{F}(p_t)$. The sequence P converges to $U_{\mathcal{S}}$ with respect to $d_{\mathcal{H}}$. In other words, $\lim_{t\to 0} |\mathcal{H}(p_t) - \mathcal{H}(U_{\mathcal{S}})| \to 0$.

Proof. The idea of the proof is to show that the distance (with respect to $d_{\mathcal{H}}$) between p_t and $U_{\mathcal{S}}$ converges to a value. If this value is 0, then the proof is complete since $U_{\mathcal{S}}$ uniquely has zero distance to itself. Otherwise, we will show that this implies that $\mathcal{F}$ is not continuous, which a contradiction.

For shorthand, define d_t to be the $d_{\mathcal{H}}$-distance to the uniform distribution, as in

$$d_t \triangleq d_{\mathcal{H}}(p_t, U_{\mathcal{S}}).$$

First we prove that d_t converges. Since the entropies of the sequence $(p_1, \dots)$ monotonically increase, we have that

$$d_1 \geqslant d_2 \geqslant \dots.$$

We also know that d_t is lower bounded by 0, and so by the monotonic convergence theorem, we have that

$$\lim_{t\to\infty} d_t \to d^*.$$

for some value $d^* \geqslant 0$.

To prove the lemma, we want to show that $d^* = 0$. Suppose, for contradiction, that $d^* \neq 0$. Then consider any distribution, q^*, such that $d_{\mathcal{H}}(q^*, U_{\mathcal{S}}) = d^*$. Such a distribution always exists since we can continuously interpolate entropy values between $\mathcal{H}(p_1)$ and $\mathcal{H}(U_{\mathcal{S}})$ with a mixture distribution. Note that $q^* \neq U_{\mathcal{S}}$ since $d_{\mathcal{H}}(U_{\mathcal{S}}, U_{\mathcal{S}}) = 0$. Since $\lim_{t\to\infty} d_t \to d^*$, we have that

$$\lim_{t\to\infty} d_{\mathcal{H}}(p_t, q^*) \to 0, \tag{62}$$

and so

$$\lim_{t\to\infty} p_t \to q^*.$$

Because the function $\mathcal{F}$ is continuous with respect to $d_{\mathcal{H}}$, Equation 62 implies that

$$\lim_{t\to\infty} d_{\mathcal{H}}(\mathcal{F}(p_t), \mathcal{F}(q^*)) \to 0.$$

However, since $\mathcal{F}(p_t) = p_{t+1}$ we can equivalently write the above equation as

$$\lim_{t\to\infty} d_{\mathcal{H}}(p_{t+1}, \mathcal{F}(q^*)) \to 0.$$

which, through a change of index variables, implies that

$$\lim_{t\to\infty} p_t \to \mathcal{F}(q^*)$$

Since q^* is not the uniform distribution, we have that $\mathcal{H}(\mathcal{F}(q^*)) > \mathcal{H}(q^*)$, which implies that $\mathcal{F}(q^*)$ and q^* are unique distributions. So, p_t converges to two distinct values, q^* and $\mathcal{F}(q^*)$, which is a contradiction. Thus, it must be the case that $d^* = 0$, completing the proof. □

C.1.2 *Proof of Lemma 3.2*

Lemma 6. *Given two distribution* $p(x)$ *and* $q(x)$ *where* $p \ll q$ *and*

$$0 < \mathrm{Cov}_p[\log p(X), \log q(X)] \tag{63}$$

define the distribution p_α *as*

$$p_\alpha(x) = \frac{1}{Z_\alpha} p(x) q(x)^\alpha$$

where $\alpha \in \mathbb{R}$ *and* Z_α *is the normalizing factor. Let* $\mathcal{H}_\alpha(\alpha)$ *be the entropy of* p_α*. Then there exists a constant* $a > 0$ *such that for all* $\alpha \in [-a, 0)$,

$$\mathcal{H}_\alpha(\alpha) > \mathcal{H}_\alpha(0) = \mathcal{H}(p). \tag{64}$$

Proof. Observe that $\{p_\alpha : \alpha \in [-1, 0]\}$ is a one-dimensional exponential family

$$p_\alpha(x) = e^{\alpha T(x) - A(\alpha) + k(x)}$$

with log carrier density $k(x) = \log p(x)$, natural parameter α, sufficient statistic $T(x) =$

$\log q(x)$, and log-normalizer $A(\alpha) = \int_{\mathcal{X}} e^{\alpha T(x)+k(x)} dx$. As shown in Nielsen and Nock, 2010, the entropy of a distribution from a one-dimensional exponential family with parameter α is given by:

$$\mathcal{H}_\alpha(\alpha) \triangleq \mathcal{H}(p_\alpha) = A(\alpha) - \alpha A'(\alpha) - \mathbb{E}_{p_\alpha}[k(X)]$$

The derivative with respect to α is then

$$\begin{aligned} \frac{d}{d\alpha}\mathcal{H}_\alpha(\alpha) &= -\alpha A''(\alpha) - \frac{d}{d\alpha}\mathbb{E}_{p_\alpha}[k(x)] \\ &= -\alpha A''(\alpha) - \mathbb{E}_\alpha[k(x)(T(x) - A'(\alpha)] \\ &= -\alpha \mathrm{Var}_{p_\alpha}[T(x)] - \mathrm{Cov}_{p_\alpha}[k(x), T(x)] \end{aligned}$$

where we use the fact that the nth derivative of $A(\alpha)$ give the n central moment, i.e. $A'(\alpha) = \mathbb{E}_{p_\alpha}[T(x)]$ and $A''(\alpha) = \mathrm{Var}_{p_\alpha}[T(x)]$. The derivative of $\alpha = 0$ is

$$\begin{aligned} \frac{d}{d\alpha}\mathcal{H}_\alpha(0) &= -\mathrm{Cov}_{p_0}[k(x), T(x)] \\ &= -\mathrm{Cov}_p[\log p(x), \log q(x)] \end{aligned}$$

which is negative by assumption. Because the derivative at $\alpha = 0$ is negative, then there exists a constant $a > 0$ such that for all $\alpha \in [-a, 0]$, $\mathcal{H}_\alpha(\alpha) > \mathcal{H}_\alpha(0) = \mathcal{H}(p)$. □

The paper applies to the case where $q = q^G_\phi$ and $p = p^S_\phi$. When we take $N \to \infty$, we have that p_{skewed} corresponds to p_α above.

c.1.3 *Simple Case Proof*

We prove the convergence directly for the (even more) simplified case when $p_\theta = p(\mathbf{S} \mid q^G_{\phi_t})$ using a similar technique:

Lemma 7. *Assume the set $\mathcal{S}$ has finite volume so that its uniform distribution $U_\mathcal{S}$ is well defined and has finite entropy. Given any distribution $p(\mathbf{s})$ whose support is $\mathcal{S}$, recursively define p_t with $p_1 = p$ and*

$$p_{t+1}(\mathbf{s}) = \frac{1}{Z^t_\alpha} p_t(\mathbf{s})^\alpha, \quad \forall \mathbf{s} \in \mathcal{S}$$

where Z^t_α is the normalizing constant and $\alpha \in [0, 1)$.

The sequence $(p_1, p_2, \dots)$ converges to $U_{\mathcal{S}}$, the uniform distribution $\mathcal{S}$.

Proof. If $\alpha = 0$, then p_2 (and all subsequent distributions) will clearly be the uniform distribution. We now study the case where $\alpha \in (0, 1)$.

At each iteration t, define the one-dimensional exponential family $\{p^t_\theta : \theta \in [0, 1]\}$ where p^t_θ is

$$p^t_\theta(\mathbf{s}) = e^{\theta T(\mathbf{s}) - A(\theta) + k(\mathbf{s})}$$

with log carrier density $k(\mathbf{s}) = 0$, natural parameter θ, sufficient statistic $T(\mathbf{s}) = \log p_t(\mathbf{s})$, and log-normalizer $A(\theta) = \int_{\mathcal{S}} e^{\theta T(\mathbf{s})} d\mathbf{s}$. As shown in Nielsen and Nock, 2010, the entropy of a distribution from a one-dimensional exponential family with parameter θ is given by:

$$\mathcal{H}^t_\theta(\theta) \triangleq \mathcal{H}(p^t_\theta) = A(\theta) - \theta A'(\theta)$$

The derivative with respect to θ is then

$$\begin{aligned}
\frac{d}{d\theta} d\mathcal{H}^t_\theta(\theta) &= -\theta A''(\theta) \\
&= -\theta \mathrm{Var}_{\mathbf{s} \sim p^t_\theta}[T(\mathbf{s})] \\
&= -\theta \mathrm{Var}_{\mathbf{s} \sim p^t_\theta}[\log p_t(\mathbf{s})] \qquad (65) \\
&\leqslant 0
\end{aligned}$$

where we use the fact that the nth derivative of $A(\theta)$ is the n central moment, i.e. $A''(\theta) = \mathrm{Var}_{\mathbf{s} \sim p^t_\theta}[T(\mathbf{s})]$. Since variance is always non-negative, this means the entropy is monotonically decreasing with θ. Note that p_{t+1} is a member of this exponential family, with parameter $\theta = \alpha \in (0, 1)$. So

$$\mathcal{H}(p_{t+1}) = \mathcal{H}^t_\theta(\alpha) \geqslant \mathcal{H}^t_\theta(1) = \mathcal{H}(p_t)$$

which implies

$$\mathcal{H}(p_1) \leqslant \mathcal{H}(p_2) \leqslant \dots .$$

This monotonically increasing sequence is upper bounded by the entropy of the uniform distribution, and so this sequence must converge.

The sequence can only converge if $\frac{d}{d\theta}\mathcal{H}_{\theta}^{t}(\theta)$ converges to zero. However, because α is bounded away from 0, Equation 65 states that this can only happen if

$$\mathrm{Var}_{\mathbf{s}\sim p_{\theta}^{t}}[\log p_t(\mathbf{s})] \to 0. \tag{66}$$

Because p_t has full support, then so does p_{θ}^{t}. Thus, Equation 66 is only true if $\log p_t(\mathbf{s})$ converges to a constant, i.e. p_t converges to the uniform distribution. □

C.2 ADDITIONAL EXPERIMENTS

C.2.1 *Sensitivity Analysis*

SENSITIVITY TO RL ALGORITHM In our experiments, we combined Skew-Fit with soft actor critic (SAC) (Haarnoja et al., 2018b). We conduct a set of experiments to test whether Skew-Fit may be used with other RL algorithms for training the goal-conditioned policy. To that end, we replaced SAC with twin delayed deep deterministic policy gradient (TD3) (Fujimoto et al., 2018b) and ran the same Skew-Fit experiments on Visual Door, Visual Pusher, and Visual Pickup. In Figure 67, we see that Skew-Fit performs consistently well with both SAC and TD3, demonstrating that Skew-Fit is beneficial across multiple RL algorithms.

SENSITIVITY TO α HYPERPARAMETER We study the sensitivity of the α hyperparameter by testing values of $\alpha \in [-1, -0.75, -0.5, -0.25, 0]$ on the Visual Door and Visual Pusher task. The results are included in Figure 68 and shows that our method is robust to different parameters of α, particularly for the more challenging Visual Pusher task. Also, the method consistently outperform $\alpha = 0$, which is equivalent to sampling uniformly from the replay buffer.

C.2.2 *Variance Ablation*

We measure the gradient variance of training a VAE on an unbalanced Visual Door image dataset with Skew-Fit vs Skew-Fit with importance sampling (IS) vs no Skew-Fit (labeled MLE). We construct the imbalanced dataset by rolling out a random policy in the environment and collecting the visual observations. Most of the images contained the door in a closed position; in a few, the door was opened. In Figure 69, we see that the gradient variance for Skew-Fit with IS is catastrophically large for large values of

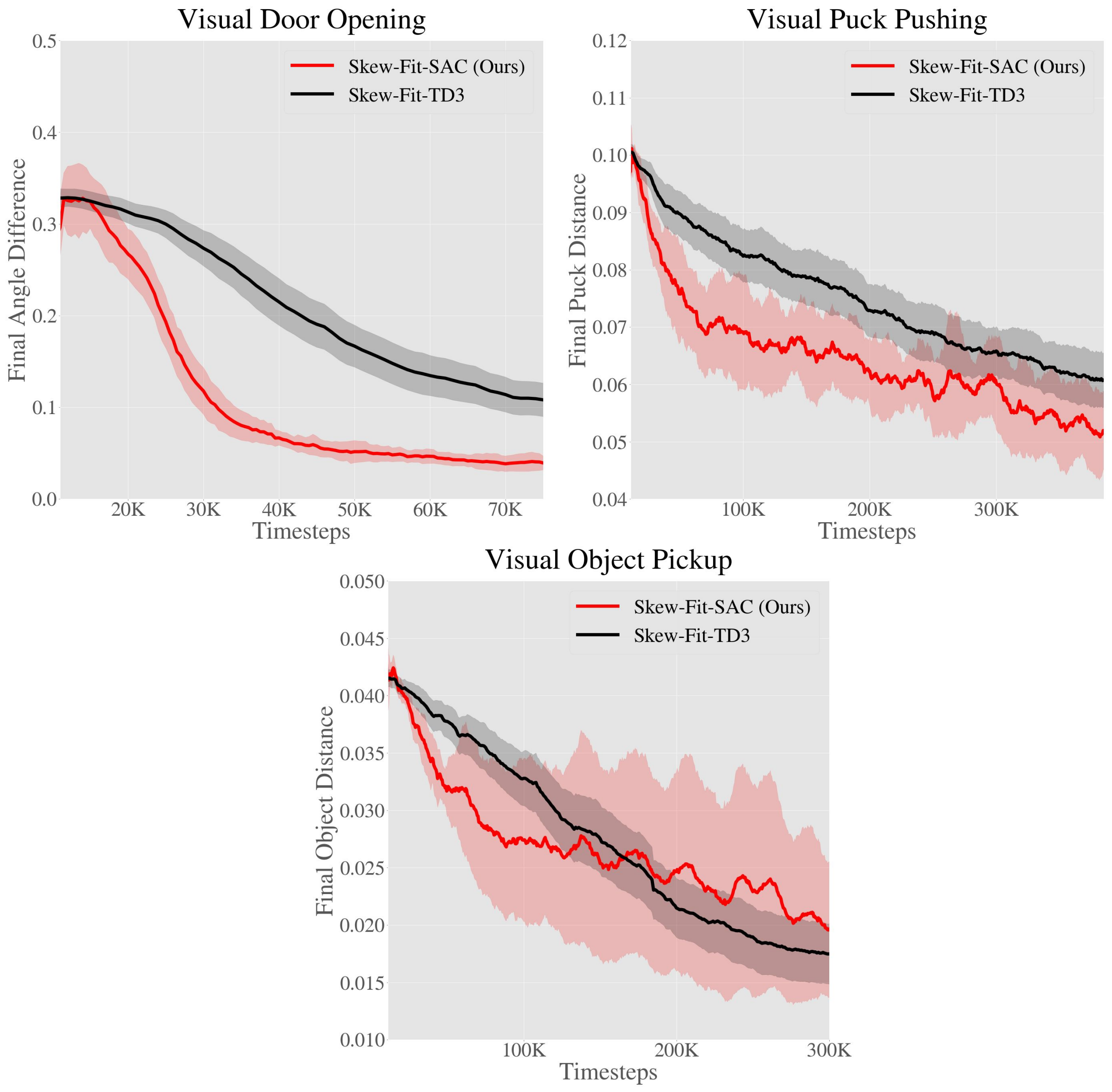


Figure 67: We compare using SAC (Haarnoja et al., 2018b) and TD3 (Fujimoto et al., 2018b) as the underlying RL algorithm on Visual Door, Visual Pusher and Visual Pickup. We see that Skew-Fit works consistently well with both SAC and TD3, demonstrating that Skew-Fit may be used with various RL algorithms. For the experiments presented in Section 9.5, we used SAC.

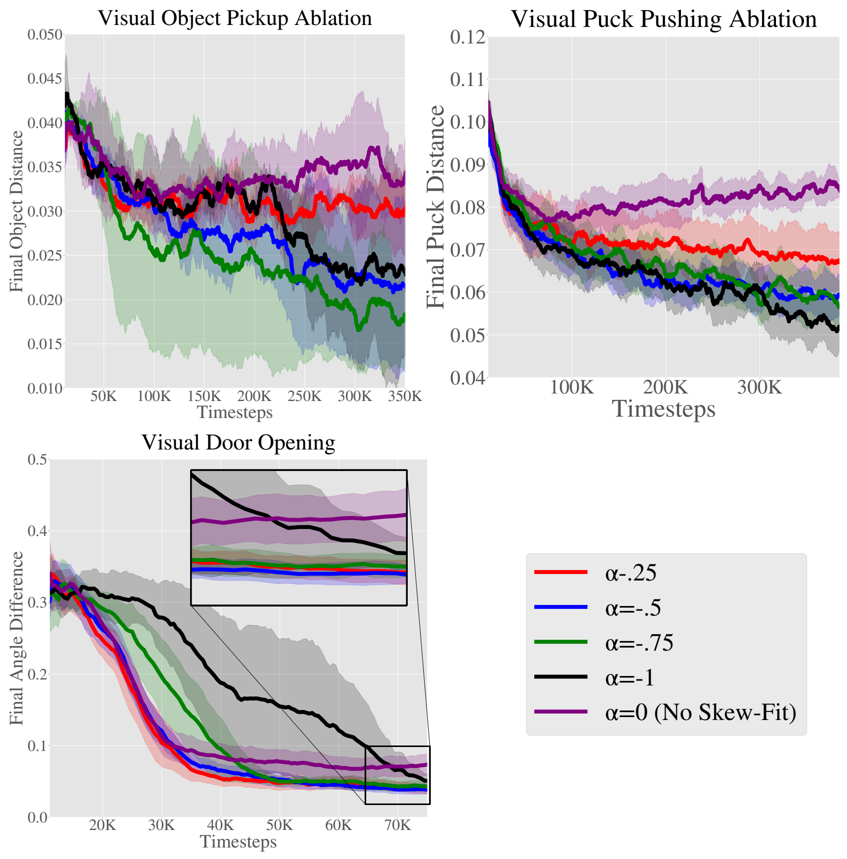


Figure 68: We sweep different values of α on Visual Door, Visual Pusher and Visual Pickup. Skew-Fit helps the final performance on the Visual Door task, and outperforms No Skew-Fit ($\alpha = 0$) as seen in the zoomed in version of the plot. In the more challenging Visual Pusher task, we see that Skew-Fit consistently helps and halves the final distance. Similarly, we observe that Skew-Fit consistently outperforms No Skew-fit on Visual Pickup. Note that alpha=-1 is not always the optimal setting for each environment, but outperforms $\alpha = 0$ in each case in terms of final performance.

α. In contrast, for Skew-Fit with SIR, which is what we use in practice, the variance is relatively similar to that of MLE. Additionally we trained three VAE's, one with MLE on a uniform dataset of valid door opening images, one with Skew-Fit on the unbalanced dataset from above, and one with MLE on the same unbalanced dataset. As expected, the VAE that has access to the uniform dataset gets the lowest negative log likelihood score. This is the oracle method, since in practice we would only have access to imbalanced data. As shown in Table 11, Skew-Fit considerably outperforms MLE, getting a much closer to oracle log likelihood score.

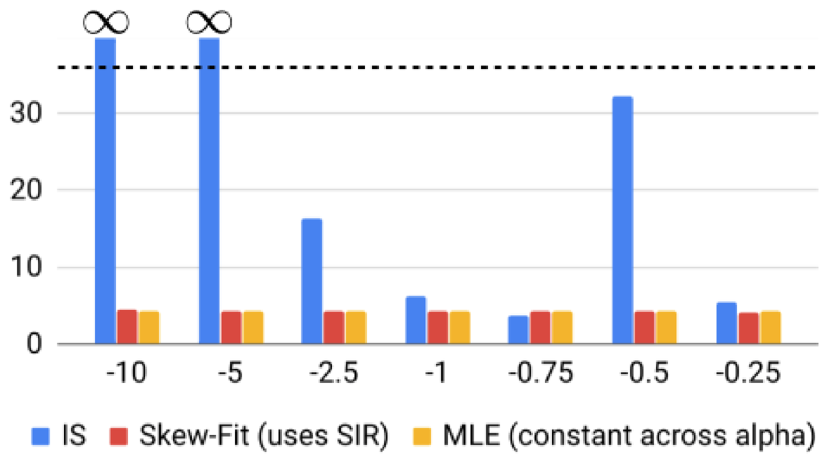


Figure 69: Gradient variance averaged across parameters in last epoch of training VAEs. Values of α less than -1 are numerically unstable for importance sampling (IS), but not for Skew-Fit.

Method	**NLL**
MLE on uniform (oracle)	20175.4
Skew-Fit on unbalanced	20175.9
MLE on unbalanced	20178.03

Table 11: Despite training on a unbalanced Visual Door dataset (see Figure 7 of paper), the negative log-likelihood (NLL) of Skew-Fit evaluated on a uniform dataset matches that of a VAE trained on a uniform dataset.

C.2.3 *Goal and Performance Visualization*

We visualize the goals sampled from Skew-Fit as well as those sampled when using the prior method, RIG (A. Nair et al., 2018b). As shown in Figure 70 and Figure 71, the generative model q_ϕ^G results in much more diverse samples when trained with Skew-Fit. We we see in Figure 72, this results in a policy that more consistently reaches the goal image.

C.3 IMPLEMENTATION DETAILS

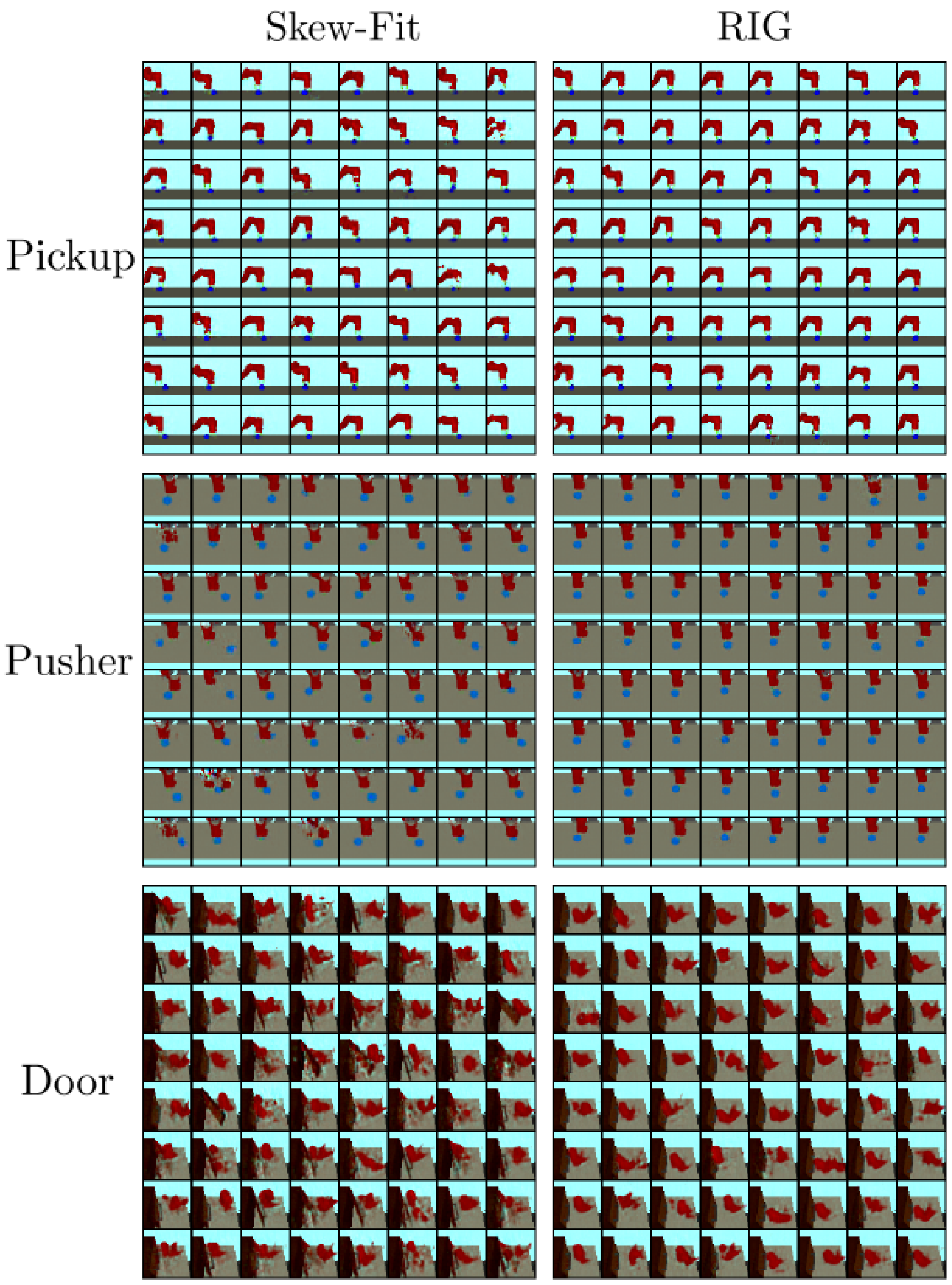


Figure 70: Proposed goals from the VAE for RIG and with Skew-Fit on the *Visual Pickup*, *Visual Pusher*, and *Visual Door* environments. Standard RIG produces goals where the door is closed and the object and puck is in the same position, while RIG + Skew-Fit proposes goals with varied puck positions, occasional object goals in the air, and both open and closed door angles.

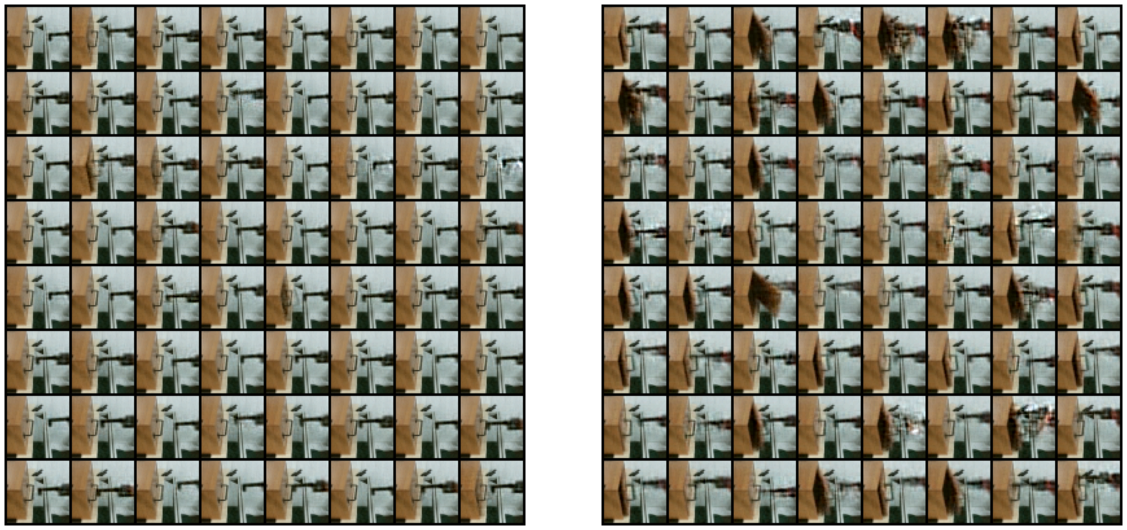


Figure 71: Proposed goals from the VAE for RIG (left) and with RIG + Skew-Fit (right) on the *Real World Visual Door* environment. Standard RIG produces goals where the door is closed while RIG + Skew-Fit proposes goals with both open and closed door angles.

C.3.1 *Likelihood Estimation using β-VAE*

We estimate the density under the VAE by using a sample-wise approximation to the marginal over x estimated using importance sampling:

$$\begin{aligned} q^{G}_{\phi_t}(x) &= \mathbb{E}_{z \sim q_{\theta_t}(z|x)} \left[\frac{p(z)}{q_{\theta_t}(z|x)} p_{\psi_t}(x \mid z) \right] \\ &\approx \frac{1}{N} \sum_{i=1}^{N} \left[\frac{p(z)}{q_{\theta_t}(z|x)} p_{\psi_t}(x \mid z) \right]. \end{aligned}$$

where q_θ is the encoder, p_ψ is the decoder, and $p(z)$ is the prior, which in this case is unit Gaussian. We found that sampling $N = 10$ latents for estimating the density worked well in practice.

C.3.2 *Oracle 2D Navigation Experiments*

We initialize the VAE to the bottom left corner of the environment for *Four Rooms*. Both the encoder and decoder have 2 hidden layers with [400, 300] units, ReLU hidden activations, and no output activations. The VAE has a latent dimension of 8 and a Gaussian decoder trained with a fixed variance, batch size of 256, and 1000 batches at each itera-

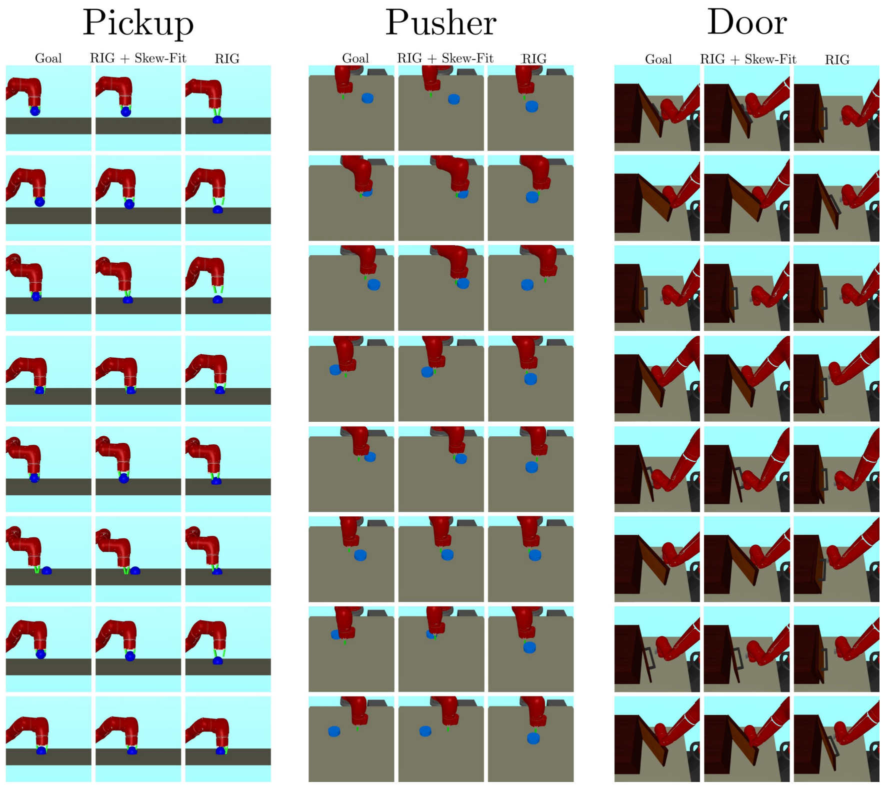


Figure 72: Example reached goals by Skew-Fit and RIG. The first column of each environment section specifies the target goal while the second and third columns show reached goals by Skew-Fit and RIG. Both methods learn how to reach goals close to the initial position, but only Skew-Fit learns to reach the more difficult goals.

tion. The VAE is trained on the exploration data buffer every 1000 rollouts.

C.3.3 *Implementation of SAC and Prior Work*

For all experiments, we trained the goal-conditioned policy using soft actor critic (SAC) (Haarnoja et al., 2018b). To make the method goal-conditioned, we concatenate the target XY-goal to the state vector. During training, we retroactively relabel the goals (L P Kaelbling, 1993; Andrychowicz et al., 2017b) by sampling from the goal distribution with probabilty 0.5. Note that the original RIG A. Nair et al., 2018b paper used TD3 Fujimoto et al., 2018b, which we also replaced with SAC in our implementation of RIG. We found that maximum entropy policies in general improved the performance of RIG, and that we did not need to add noise on top of the stochastic policy's noise. In the prior RIG method, the VAE was pre-trained on a uniform sampling of images from the state space of each environment. In order to ensure a fair comparison to Skew-Fit, we forego pre-training and instead train the VAE alongside RL, using the variant described in the RIG paper. For our RL network architectures and training scheme, we use fully connected networks for the policy, Q-function and value networks with two hidden layers of size 400 and 300 each. We also delay training any of these networks for 10000 time steps in order to collect sufficient data for the replay buffer as well as to ensure the latent space of the VAE is relatively stable (since we continuously train the VAE concurrently with RL training). As in RIG, we train a goal-conditioned value functions Schaul et al., 2015b using hindsight experience replay Andrychowicz et al., 2017b, relabelling 50% of exploration goals as goals sampled from the VAE prior $\mathcal{N}(0, 1)$ and 30% from future goals in the trajectory.

For our implementation of (Hazan et al., 2019), we trained the policies with the reward

$$r(s) = r_{\text{Skew-Fit}}(s) + \lambda \cdot r_{\text{Hazan et al.}}(s)$$

For $r_{\text{Hazan et al.}}$, we use the reward described in Section 5.2 of Hazan et al. (2019), which requires an estimated likelihood of the state. To compute these likelihood, we use the same method as in Skew-Fit (see Section C.3.1). With 3 seeds each, we tuned λ across values $[100, 10, 1, 0.1, 0.01, 0.001]$ for the door task, but all values performed poorly. For the pushing and picking tasks, we tested values across $[1, 0.1, 0.01, 0.001, 0.0001]$ and found that 0.1 and 0.01 performed best for each task, respectively.

C.3.4 *RIG with Skew-Fit Summary*

10 provides detailed pseudo-code for how we combined our method with RIG. Steps that were removed from the base RIG algorithm are highlighted in blue and steps that were added are highlighted in red. The main differences between the two are (1) not needing to pre-train the β-VAE, (2) sampling exploration goals from the buffer using p_{skewed} instead of the VAE prior, (3) relabeling with replay buffer goals sampled using p_{skewed} instead of from the VAE prior, and (4) training the VAE on replay buffer data data sampled using p_{skewed} instead of uniformly.

C.3.5 *Vision-Based Continuous Control Experiments*

In our experiments, we use an image size of 48x48. For our VAE architecture, we use a modified version of the architecture used in the original RIG paper A. Nair et al., 2018b.

Our VAE has three convolutional layers with kernel sizes: 5x5, 3x3, and 3x3, number of output filters: 16, 32, and 64 and strides: 3, 2, and 2. We then have a fully connected layer with the latent dimension number of units, and then reverse the architecture with deconvolution layers. We vary the latent dimension of the VAE, the β term of the VAE and the α term for Skew-Fit based on the environment. Additionally, we vary the training schedule of the VAE based on the environment. See the table at the end of the appendix for more details. Our VAE has a Gaussian decoder with identity variance, meaning that we train the decoder with a mean-squared error loss.

When training the VAE alongside RL, we found the following three schedules to be effective for different environments:

1. For first 5K steps: Train VAE using standard MLE training every 500 time steps for 1000 batches. After that, train VAE using Skew-Fit every 500 time steps for 200 batches.
2. For first 5K steps: Train VAE using standard MLE training every 500 time steps for 1000 batches. For the next 45K steps, train VAE using Skew-Fit every 500 steps for 200 batches. After that, train VAE using Skew-Fit every 1000 time steps for 200 batches.
3. For first 40K steps: Train VAE using standard MLE training every 4000 time steps for 1000 batches. Afterwards, train VAE using Skew-Fit every 4000 time steps for 200 batches.

We found that initially training the VAE without Skew-Fit improved the stability of the algorithm. This is due to the fact that density estimates under the VAE are constantly changing and inaccurate during the early phases of training. Therefore, it made little sense to use those estimates to prioritize goals early on in training. Instead, we simply train using MLE training for the first 5K timesteps, and after that we perform Skew-Fit according to the VAE schedules above. Table 12 lists the hyper-parameters that were shared across the continuous control experiments. Table 13 lists hyper-parameters specific to each environment. Additionally, Section C.3.4 discusses the combined RIG + Skew-Fit algorithm.

C.4 ENVIRONMENT DETAILS

Four Rooms: A 20 x 20 2D pointmass environment in the shape of four rooms (R. S. Sutton et al., 1999). The observation is the 2D position of the agent, and the agent must specify a target 2D position as the action. The dynamics of the environment are the following: first, the agent is teleported to the target position, specified by the action. Then a Gaussian

Hyper-parameter	**Value**	**Comments**
# training batches per time step	2	Marginal improvements after 2
Exploration Noise	None (SAC policy is stochastic)	Did not tune
RL Batch Size	1024	smaller batch sizes work as well
VAE Batch Size	64	Did not tune
Discount Factor	0.99	Did not tune
Reward Scaling	1	Did not tune
Policy Hidden Sizes	$[400, 300]$	Did not tune
Policy Hidden Activation	ReLU	Did not tune
Q-Function Hidden Sizes	$[400, 300]$	Did not tune
Q-Function Hidden Activation	ReLU	Did not tune
Replay Buffer Size	100000	Did not tune
Number of Latents for Estimating Density (N)	10	Marginal improvements beyond 10

Table 12: General hyper-parameters used for all *visual* experiments.

Hyper-parameter	**Visual Pusher**	**Visual Door**	**Visual Pickup**	**Real World Visual Door**
Path Length	50	100	50	100
β for β-VAE	20	20	30	60
Latent Dimension Size	4	16	16	16
α for Skew-Fit	-1	$-1/2$	-1	$-1/2$
VAE Training Schedule	2	1	2	1
Sample Goals From	q_ϕ^G	p_{skewed}	p_{skewed}	p_{skewed}

Table 13: Environment specific hyper-parameters for the *visual* experiments

Algorithm 10 RIG and RIG + Skew-Fit. Blue text denotes RIG specific steps and red text denotes RIG + Skew-Fit specific steps

```
Require: β-VAE mean encoder $q_\phi$, β-VAE decoder $p_\psi$, policy $\pi_\theta$, goal-conditioned value function $Q_w$,
    skew parameter $\alpha$, VAE Training Schedule.
 1: Collect $\mathcal{D} = \{s^{(i)}\}$ using random initial policy.
 2: Train β-VAE on data uniformly sampled from $\mathcal{D}$.
 3: Fit prior $p(z)$ to latent encodings $\{\mu_\phi(s^{(i)})\}$.
 4: for $n = 0, ..., N-1$ episodes do
 5:     Sample latent goal from prior $z_g \sim p(z)$.
 6:     Sample state $s_g \sim p_{\text{skewed}_n}$ and encode $z_g = q_\phi(s_g)$ if $\mathcal{R}$ is nonempty. Otherwise sample $z_g \sim p(z)$
 7:     Sample initial state $s_0$ from the environment.
 8:     for $t = 0, ..., H-1$ steps do
 9:         Get action $a_t \sim \pi_\theta(q_\phi(s_t), z_g)$.
10:         Get next state $s_{t+1} \sim p(\cdot \mid s_t, a_t)$.
11:         Store $(s_t, a_t, s_{t+1}, z_g)$ into replay buffer $\mathcal{R}$.
12:         Sample transition $(s, a, s', z_g) \sim \mathcal{R}$.
13:         Encode $z = q_\phi(s), z' = q_\phi(s')$.
14:         (Probability 0.5) replace $z_g$ with $z'_g \sim p(z)$.
15:         (Probability 0.5) replace $z_g$ with $q_\phi(s'')$ where $s'' \sim p_{\text{skewed}_n}$.
16:         Compute new reward $r = -\|z' - z_g\|$.
17:         Minimize Bellman Error using $(z, a, z', z_g, r)$.
18:     end for
19:     for $t = 0, ..., H-1$ steps do
20:         for $i = 0, ..., k-1$ steps do
21:             Sample future state $s_{h_i}$, $t < h_i \leqslant H-1$.
22:             Store $(s_t, a_t, s_{t+1}, q_\phi(s_{h_i}))$ into $\mathcal{R}$.
23:         end for
24:     end for
25:     Construct skewed replay buffer distribution $p_{\text{skewed}_{n+1}}$ using data from $\mathcal{R}$ with Equation 10.
26:     if total steps $< 5000$ then
27:         Fine-tune β-VAE on data uniformly sampled from $\mathcal{R}$ according to VAE Training Schedule.
28:     else
29:         Fine-tune β-VAE on data uniformly sampled from $\mathcal{R}$ according to VAE Training Schedule.
30:         Fine-tune β-VAE on data sampled from $p_{\text{skewed}_{n+1}}$ according to VAE Training Schedule.
31:     end if
32: end for=0
```

Hyper-parameter	**Value**
# training batches per time step	.25
Exploration Noise	None (SAC policy is stochastic)
RL Batch Size	512
VAE Batch Size	64
Discount Factor	$\frac{299}{300}$
Reward Scaling	10
Path length	300
Policy Hidden Sizes	$[400, 300]$
Policy Hidden Activation	ReLU
Q-Function Hidden Sizes	$[400, 300]$
Q-Function Hidden Activation	ReLU
Replay Buffer Size	1000000
Number of Latents for Estimating Density (N)	10
β for β-VAE	10
Latent Dimension Size	2
α for Skew-Fit	-2.5
VAE Training Schedule	3
Sample Goals From	p_{skewed}

Table 14: Hyper-parameters used for the *ant* experiment.

change in position with mean 0 and standard deviation 0.0605 is applied[1]. If the action would result in the agent moving through or into a wall, then the agent will be stopped

1 In the main paper, we rounded this to 0.06, but this difference does not matter.

at the wall instead.

Ant: A MuJoCo (Todorov et al., 2012) ant environment. The observation is a 3D position and velocities, orientation, joint angles, and velocity of the joint angles of the ant (8 total). The observation space is 29 dimensions. The agent controls the ant through the joints, which is 8 dimensions. The goal is a target 2D position, and the reward is the negative Euclidean distance between the achieved 2D position and target 2D position.

Visual Pusher: A MuJoCo environment with a 7-DoF Sawyer arm and a small puck on a table that the arm must push to a target position. The agent controls the arm by commanding x, y position for the end effector (EE). The underlying state is the EE position, e and puck position p. The evaluation metric is the distance between the goal and final puck positions. The hand goal/state space is a 10x10 cm^2 box and the puck goal/state space is a 30x20 cm^2 box. Both the hand and puck spaces are centered around the origin. The action space ranges in the interval $[-1, 1]$ in the x and y dimensions.

Visual Door: A MuJoCo environment with a 7-DoF Sawyer arm and a door on a table that the arm must pull open to a target angle. Control is the same as in *Visual Pusher*. The evaluation metric is the distance between the goal and final door angle, measured in radians. In this environment, we do not reset the position of the hand or door at the end of each trajectory. The state/goal space is a 5x20x15 cm^3 box in the x, y, z dimension respectively for the arm and an angle between $[0, .83]$ radians. The action space ranges in the interval $[-1, 1]$ in the x, y and z dimensions.

Visual Pickup: A MuJoCo environment with the same robot as *Visual Pusher*, but now with a different object. The object is cube-shaped, but a larger intangible sphere is overlaid on top so that it is easier for the agent to see. Moreover, the robot is constrained to move in 2 dimension: it only controls the y, z arm positions. The x position of both the arm and the object is fixed. The evaluation metric is the distance between the goal and final object position. For the purpose of evaluation, 75% of the goals have the object in the air and 25% have the object on the ground. The state/goal space for both the object and the arm is 10cm in the y dimension and 13cm in the z dimension. The action space ranges in the interval $[-1, 1]$ in the y and z dimensions.

Real World Visual Door: A Rethink Sawyer Robot with a door on a table. The arm must pull the door open to a target angle. The agent controls the arm by commanding the x, y, z velocity of the EE. Our controller commands actions at a rate of up to 10Hz with the scale of actions ranging up to 1cm in magnitude. The underlying state and goal is the same as in *Visual Door*. Again we do not reset the position of the hand or door at the end of each trajectory. We obtain images using a Kinect Sensor. The state/goal space for the environment is a 10x10x10 cm^3 box. The action space ranges in the interval $[-1, 1]$

(in cm) in the x, y and z dimensions. The door angle lies in the range $[0, 45]$ degrees.

C.5 GOAL-CONDITIONED REINFORCEMENT LEARNING MINIMIZES $\mathcal{H}(\mathbf{G} \mid \mathbf{S})$

Some goal-conditioned RL methods such as Warde-Farley et al. (2018); A. Nair et al. (2018b) present methods for minimizing a lower bound for $\mathcal{H}(\mathbf{G} \mid \mathbf{S})$, by approximating $\log p(\mathbf{G} \mid \mathbf{S})$ and using it as the reward. Other goal-conditioned RL methods (L P Kaelbling, 1993; Lillicrap et al., 2016; Schaul et al., 2015b; Andrychowicz et al., 2017b; V. Pong et al., 2018; Florensa et al., 2018a) are not developed with the intention of minimizing the conditional entropy $\mathcal{H}(\mathbf{G} \mid \mathbf{S})$. Nevertheless, one can see that goal-conditioned RL generally minimizes $\mathcal{H}(\mathbf{G} \mid \mathbf{S})$ by noting that the optimal goal-conditioned policy will deterministically reach the goal. The corresponding conditional entropy of the goal given the state, $\mathcal{H}(\mathbf{G} \mid \mathbf{S})$, would be zero, since given the current state, there would be no uncertainty over the goal (the goal must have been the current state since the policy is optimal). So, the objective of goal-conditioned RL can be interpreted as finding a policy such that $\mathcal{H}(\mathbf{G} \mid \mathbf{S}) = 0$. Since zero is the minimum value of $\mathcal{H}(\mathbf{G} \mid \mathbf{S})$, then goal-conditioned RL can be interpreted as minimizing $\mathcal{H}(\mathbf{G} \mid \mathbf{S})$.

D

APPENDIX: CHAPTER 6

D.1 ALGORITHM DERIVATION DETAILS

The full optimization problem we solve, given the previous off-policy advantage estimate A^{π_k} and buffer distribution π_β, is given below:

$$\pi_{k+1} = \arg\max_{\pi \in \Pi} \mathbb{E}_{\mathbf{a}\sim\pi(\cdot|\mathbf{s})}[A^{\pi_k}(\mathbf{s},\mathbf{a})] \tag{67}$$

$$\text{s.t. } D_{\mathrm{KL}}(\pi(\cdot|\mathbf{s})||\pi_\beta(\cdot|\mathbf{s})) \leqslant \epsilon \tag{68}$$

$$\int_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s})d\mathbf{a} = 1. \tag{69}$$

Our derivation follows Peters et al. (2010) and Peng et al. (2019a). The analytic solution for the constrained optimization problem above can be obtained by enforcing the KKT conditions. The Lagrangian is:

$$\mathcal{L}(\pi,\lambda,\alpha) = \mathbb{E}_{\mathbf{a}\sim\pi(\cdot|\mathbf{s})}[A^{\pi_k}(\mathbf{s},\mathbf{a})] \tag{70}$$

$$+\lambda(\epsilon - D_{\mathrm{KL}}(\pi(\cdot|\mathbf{s})||\pi_\beta(\cdot|\mathbf{s}))) \tag{71}$$

$$+\alpha(1-\int_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s})d\mathbf{a}). \tag{72}$$

Differentiating with respect to π gives:

$$\frac{\partial \mathcal{L}}{\partial \pi} = A^{\pi_k}(\mathbf{s},\mathbf{a}) - \lambda \log \pi_\beta(\mathbf{a}|\mathbf{s}) + \lambda \log \pi(\mathbf{a}|\mathbf{s}) + \lambda - \alpha. \tag{73}$$

Setting $\frac{\partial \mathcal{L}}{\partial \pi}$ to zero and solving for π gives the closed form solution to this problem:

$$\pi^*(\mathbf{a}|\mathbf{s}) = \frac{1}{Z(\mathbf{s})}\pi_\beta(\mathbf{a}|\mathbf{s}) \exp\left(\frac{1}{\lambda}A^{\pi_k}(\mathbf{s},\mathbf{a})\right), \tag{74}$$

Next, we project the solution into the space of parametric policies. For a policy π_θ with parameters θ, this can be done by minimizing the KL divergence of π_θ from the optimal non-parametric solution π^* under the data distribution $\rho_{\pi_\beta}(\mathbf{s})$:

$$\arg\min_\theta \ {}_{\rho_{\pi_\beta}(\mathbf{s})}\left[D_{KL}(\pi^*(\cdot|\mathbf{s})||\pi_\theta(\cdot|\mathbf{s}))\right] \tag{75}$$

$$= \arg\min_\theta \ {}_{\rho_{\pi_\beta}(\mathbf{s})}\left[{}_{\pi^*(\cdot|\mathbf{s})}[-\log \pi_\theta(\cdot|\mathbf{s})]\right] \tag{76}$$

Note that in the projection step, the parametric policy could be projected with either direction of KL divergence. However, choosing the reverse KL direction has a key advantage: it allows us to optimize θ as a maximum likelihood problem with an expectation over data $s, a \sim \beta$, rather than sampling actions from the policy that may be out of distribution for the Q function. In our experiments we show that this decision is vital for stable off-policy learning.

Furthermore, assume discrete policies with a minimum probably density of $\pi_\theta \geqslant \alpha_\theta$. Then the upper bound:

$$D_{KL}(\pi^*||\pi_\theta) \leqslant \frac{2}{\alpha_\theta} D_{TV}(\pi^*, \pi_\theta)^2 \tag{77}$$

$$\leqslant \frac{1}{\alpha_\theta} D_{KL}(\pi_\theta||\pi^*) \tag{78}$$

holds by the Pinsker's inequality, where D_{TV} denotes the total variation distance between distributions. Thus minimizing the reverse KL also bounds the forward KL. Note that we can control the minimum α if desired by applying Laplace smoothing to the policy.

D.2 IMPLEMENTATION DETAILS

We implement the algorithm building on top of twin soft actor-critic (Haarnoja et al., 2018a), which incorporates the twin Q-function architecture from twin delayed deep deterministic policy gradient (TD3) from Fujimoto et al. (2018b). All off-policy algorithm comparisons (SAC, BRAC, MPO, ABM, BEAR) are implemented from the same skeleton.

The base hyperparameters are given in Table 16. The policy update is replaced with:

$$\theta_{k+1} = \arg\max_{\theta} \ _{\mathbf{s},\mathbf{a}\sim\beta} \left[\log \pi_\theta(\mathbf{a}|\mathbf{s}) \frac{1}{Z(\mathbf{s})} \exp\left(\frac{1}{\lambda} A^{\pi_k}(\mathbf{s},\mathbf{a})\right)\right]. \tag{79}$$

Env	Use $Z(\mathbf{s})$	Omit $Z(\mathbf{s})$
pen	84%	98%
door	0%	95%
relocate	0%	54%

Table 15: Success rates after online fine-tuning (after 800K steps for pen, door and 4M steps for relocate) using AWAC with and without $Z(\mathbf{s})$ weight. These results show that although we can estimate $Z(\mathbf{s})$, weighting by $Z(\mathbf{s})$ actually results in worse performance.

Similar to advantage weight regression (Peng et al., 2019a) and other prior work (Neumann and Peters, 2008; Q. Wang et al., 2018a; Noah Y. Siegel et al., 2020b), we disregard the per-state normalizing constant $Z(\mathbf{s}) = \int_{\mathbf{a}} \pi_\theta(\mathbf{a}|\mathbf{s}) \exp\left(\frac{1}{\lambda} A^{\pi_k}(\mathbf{s},\mathbf{a})\right) d\mathbf{a} = \mathbb{E}_{\mathbf{a}\sim\pi_\theta(\cdot|\mathbf{s})}[A^{\pi_k}(\mathbf{s},\mathbf{a})]$. We did experiment with estimating this expectation per batch element with $K = 10$ samples, but found that this generally made performance worse, perhaps because errors in the estimation of $Z(\mathbf{s})$ caused more harm than the benefit the method derived from estimating this value. We report success rate results for variants of our method with and without $Z(\mathbf{s})$ estimation in Table 15.

While prior work (Neumann and Peters, 2008; Q. Wang et al., 2018a; Peng et al., 2019a) has generally ignored the omission of $Z(\mathbf{s})$ without any specific justification, it is possible to bound this value both above and below using the Cauchy-Schwarz and reverse Cauchy-Schwarz (Polya-Szego) inequalities, as follows. Let $f(\mathbf{a}) = \pi(\mathbf{a}|\mathbf{s})$ and $g(\mathbf{a}) = \exp(A(\mathbf{s},\mathbf{a})/\lambda)$. Note $f(\mathbf{a}) > 0$ for stochastic policies and $g(\mathbf{a}) > 0$. By Cauchy-Schwarz, $Z(s) = \int_{\mathbf{a}} f(\mathbf{a})g(\mathbf{a})d\mathbf{a} \leqslant \sqrt{\int_{\mathbf{a}} f(\mathbf{a})^2 d\mathbf{a} \int_{\mathbf{a}} g(\mathbf{a})^2 d\mathbf{a}} = C_1$. To apply Polya-Szego, let m_f and m_g be the minimum of f and g respectively and M_f, M_g be the maximum. Then $Z(\mathbf{s}) \geqslant 2(\sqrt{\frac{M_f M_g}{m_f m_g}} + \frac{m_f m_g}{M_f M_g})^{-1} C_1 = C_2$. We therefore have $C_1 \leqslant Z(\mathbf{s}) \leqslant C_2$, though the bounds are generally not tight.

A further, more intuitive argument for why omitting $Z(\mathbf{s})$ may be harmless in practice comes from observing that this normalizing factor only affects the relative weight of different *states* in the training objective, not different actions. The state distribution in β already differs from the distribution over states that will be visited by π_θ, and therefore preserving this state distribution is likely to be of limited utility to downstream policy performance. Indeed, we would expect that sufficiently expressive policies would be less affected by small to moderate variability in the state weights. On the other hand, inaccurate estimates of $Z(\mathbf{s})$ may throw off the training objective by increasing variance, similar to the effect of degenerate importance weights.

The Lagrange multiplier λ is treated as a hyperparameter in our method. In this work we use $\lambda = 0.3$ for the manipulation environments and $\lambda = 1.0$ for the MuJoCo benchmark environments. One could adaptively learn λ with a dual gradient descent procedure, but this would require access to π_β.

As rewards for the dextrous manipulation environments are non-positive, we clamp the Q value for these experiments to be at most zero. We find this stabilizes training slightly.

D.3 ENVIRONMENT-SPECIFIC DETAILS

We evaluate our method on three domains: dexterous manipulation environments, Sawyer manipulation environments, and MuJoCo benchmark environments. In the following sections we describe specific details.

D.3.1 *Dexterous Manipulation Environments*

These environments are modified from those proposed by Rajeswaran et al. (2018).

PEN-BINARY-V0. The task is to spin a pen into a given orientation. The action dimension is 24 and the observation dimension is 45. Let the position and orientation of the pen be denoted by x_p and x_o respectively, and the desired position and orientation be denoted by d_p and d_o respectively. The reward function is $r = \mathbb{1}_{|x_p - d_p| \leqslant 0.075} \mathbb{1}_{|x_o \cdot d_o| \leqslant 0.95}$ - 1. In Rajeswaran et al. (2018), the episode was terminated when the pen fell out of the hand; we did not include this early termination condition.

DOOR-BINARY-V0. The task is to open a door, which requires first twisting a latch. The action dimension is 28 and the observation dimension is 39. Let d denote the angle of the door. The reward function is $r = \mathbb{1}_{d>1.4}$ - 1.

RELOCATE-BINARY-V0. The task is to relocate an object to a goal location. The action dimension is 30 and the observation dimension is 39. Let x_p denote the object position

Hyper-parameter	**Value**
Training Batches Per Timestep	1
Exploration Noise	None (stochastic policy)
RL Batch Size	1024
Discount Factor	0.99
Reward Scaling	1
Replay Buffer Size	1000000
Number of pretraining steps	25000
Policy Hidden Sizes	$[256, 256, 256, 256]$
Policy Hidden Activation	ReLU
Policy Weight Decay	10^{-4}
Policy Learning Rate	3×10^{-4}
Q Hidden Sizes	$[256, 256, 256, 256]$
Q Hidden Activation	ReLU
Q Weight Decay	0
Q Learning Rate	3×10^{-4}
Target Network τ	5×10^{-3}

Table 16: Hyper-parameters used for RL experiments.

and d_p denote the desired position. The reward is $r = \mathbb{1}_{|x_p - d_p| \leqslant 0.1}$ - 1.

D.3.2 *Sawyer Manipulation Environment*

SAWYERPUSH-V0. This environment is included in the Multiworld library. The task is to push a puck to a goal position in a 40cm x 20cm, and the reward function is the negative distance between the puck and goal position. When using this environment, we use hindsight experience replay for goal-conditioned reinforcement learning. The random dataset for prior data was collected by rolling out an Ornstein-Uhlenbeck process with $\theta = 0.15$ and $\sigma = 0.3$.

D.3.3 *Off-Policy Data Performance*

The performances of the expert data, behavior cloning (BC) on the expert data (1), and BC on the combined expert+BC data (2) are included in Table 17. For Gym benchmarks we report average return, and expert data is collected by a trained SAC policy. For dextrous manipulation tasks we report the success rate, and the expert data consists of human demonstrations provided by Rajeswaran et al. (2018).

Env	Expert	BC (1)	BC (2)
cheetah	9962	2507	4524
walker	5062	2040	1701
ant	5207	687	1704
pen	1	0.73	0.76
door	1	0.10	0.00
relocate	1	0.02	0.01

Table 17: Performance of the off-policy data for each environment. BC (1) indicates BC on the expert data, while BC (2) indicates BC on the combined expert+BC data used as off-policy data for pretraining.

Name	$\hat{Q}$	Policy Objective	$\hat{\pi}_\beta$?	Constraint
SAC	Q^π	$D_{KL}(\pi_\theta \| \bar{Q})$	No	None
SAC + BC	Q^π	Mixed	No	None
BCQ	Q^π	$D_{KL}(\pi_\theta \| \bar{Q})$	Yes	Support (ℓ^∞)
BEAR	Q^π	$D_{KL}(\pi_\theta \| \bar{Q})$	Yes	Support (MMD)
AWR	Q^β	$D_{KL}(\bar{Q} \| \pi_\theta)$	No	Implicit
MPO	Q^π	$D_{KL}(\bar{Q} \| \pi_\theta)$	Yes*	Prior
ABM-MPO	Q^π	$D_{KL}(\bar{Q} \| \pi_\theta)$	Yes	Learned Prior
DAPG	-	$J(\pi_\theta)$	No	None
BRAC	Q^π	$D_{KL}(\pi_\theta \| \bar{Q})$	Yes	Explicit KL penalty
AWAC (Ours)	Q^π	$D_{KL}(\bar{Q} \| \pi_\theta)$	No	Implicit

Table 18: Comparison of prior algorithms that can incorporate prior datasets. See section D.4 for specific implementation details. We argue that avoiding estimating $\hat{\pi}_\beta$ (i.e., $\hat{\pi}_\beta$ is "No") is important when learning with complex datasets that include experience from multiple policies, as in the case of online fine-tuning, and maintaining a constraint of some sort is essential for offline training. At the same time, sample-efficient learning requires using Q^π for the critic. Our algorithm is the only one that fulfills all of these requirements.

D.4 BASELINE IMPLEMENTATION DETAILS

We used public implementations of prior methods (DAPG, AWR) when available. We implemented the remaining algorithms in our framework, which also allows us to understand the effects of changing individual components of the method. In the section, we describe the implementation details. The full overview of algorithms is given in Figure 18.

Behavior Cloning (BC). This method learns a policy with supervised learning on demonstration data.

Soft Actor Critic (SAC). Using the soft actor critic algorithm from (Haarnoja et al., 2018a), we follow the exact same procedure as our method in order to incorporate prior data, initializing the policy with behavior cloning on demonstrations and adding all prior data to the replay buffer.

Behavior Regularized Actor Critic (BRAC). We implement BRAC as described in (Yifan Wu et al., 2020) by adding policy regularization $\log(\pi_\beta(a|s))$ where π_β is a behavior policy trained with supervised learning on the replay buffer. We add all prior data to the replay buffer before online training.

Advantage Weighted Regression (AWR). Using the advantage weighted regression algorithm from (Peng et al., 2019a), we add all prior data to the replay buffer before online training. We use the implementation provided by Peng et al. (2019a), with the key difference from our method being that AWR uses TD(λ) on the replay buffer for policy evaluation.

Monotonic Advantage Re-Weighted Imitation Learning (MARWIL). Monotonic advantage re-weighted imitation learning was proposed by Q. Wang et al. (2018a) for offline imitation learning. MARWIL was not demonstrated in online RL settings, but we evaluate it for offline pretraining followed by online fine-tuning as we do other offline algorithms. Although derived differently, MARWIL and AWR are similar algorithms and only differ in value estimation: MARWIL uses the on-policy single-path advantage estimate $A(s, a) = Q^{\pi_\beta}(s, a) - V^{\pi_\beta}(s)$ instead of TD(λ) as in AWR. Thus, we implement MARWIL by modifying the implementation of AWR.

Maximum a Posteriori Policy Optimization (MPO). We evaluate the MPO algorithm presented by Abdolmaleki et al. (2018). Due to a public implementation being unavailable, we modify our algorithm to be as close to MPO as possible. In particular, we change the policy update in Skew-Fit to be:

$$\begin{aligned}\theta_i \longleftarrow \arg\max_{\theta_i} \quad & \mathbb{E}_{s\sim\mathcal{D}, a\sim\pi(a|s)} \\ & \left[\log \pi_{\theta_i}(a|s) \exp\left(\frac{1}{\beta} Q^{\pi_\beta}(s, a)\right)\right]. \end{aligned} \tag{80}$$

Note that in MPO, actions for the update are sampled from the policy and the Q-function is used instead of advantage for weights. We failed to see offline or online improvement with this implementation in most environments, so we omit this comparison in favor of ABM.

Advantage-Weighted Behavior Model (ABM). We evaluate ABM, the method developed in Noah Y. Siegel et al. (2020b). As with MPO, we modify our method to implement ABM, as there is no public implementation of the method. ABM first trains an advantage model $\pi_{\theta_{\text{abm}}}(a|s)$:

$$\theta_{\text{abm}} = \arg\max_{\theta_i} \mathbb{E}_{\tau\sim\mathcal{D}} \left[\sum_{t=1}^{|\tau|} \log \pi_{\theta_{\text{abm}}}(a_t|s_t) f(R(\tau_{t:N}) - \hat{V}(s)) \right]. \tag{81}$$

where f is an increasing non-negative function, chosen to be $f = 1_+$. In place of an advantage computed by empirical returns $R(\tau_{t:N}) - \hat{V}(s)$ we use the advantage estimate computed per transition by the Q value $Q(s,a) - V(s)$. This is favorable for running ABM online, as computing $R(\tau_{t:N}) - \hat{V}(s)$ is similar to AWR, which shows slow online improvement. We then use the policy update:

$$\theta_i \longleftarrow \arg\max_{\theta_i} \mathbb{E}_{s\sim\mathcal{D}, a\sim\pi_{\text{abm}}(a|s)} \left[\log \pi_{\theta_i}(a|s) \exp\left(\frac{1}{\lambda} (Q^{\pi_i}(s,a) - V^{\pi_i}(s)) \right) \right]. \tag{82}$$

Additionally, for this method, actions for the update are sampled from a behavior policy trained to match the replay buffer and the value function is computed as $V^{\pi}(s) = Q^{\pi}(s,a)$ s.t. $a \sim \pi$.

Demonstration Augmented Policy Gradient (DAPG). We directly utilize the code provided in (Rajeswaran et al., 2018) to compare against our method. Since DAPG is an on-policy method, we only provide the demonstration data to the DAPG code to bootstrap the initial policy from.

Bootstrapping Error Accumulation Reduction (BEAR). We utilize the implementation of BEAR provided in rlkit. We provide the demonstration and off-policy data to the method together. Since the original method only involved training offline, we modify the algorithm to include an online training phase. In general we found that the MMD constraint in the method was too conservative. As a result, in order to obtain the results displayed in our paper, we swept the MMD threshold value and chose the one with the best final performance after offline training with offline fine-tuning.

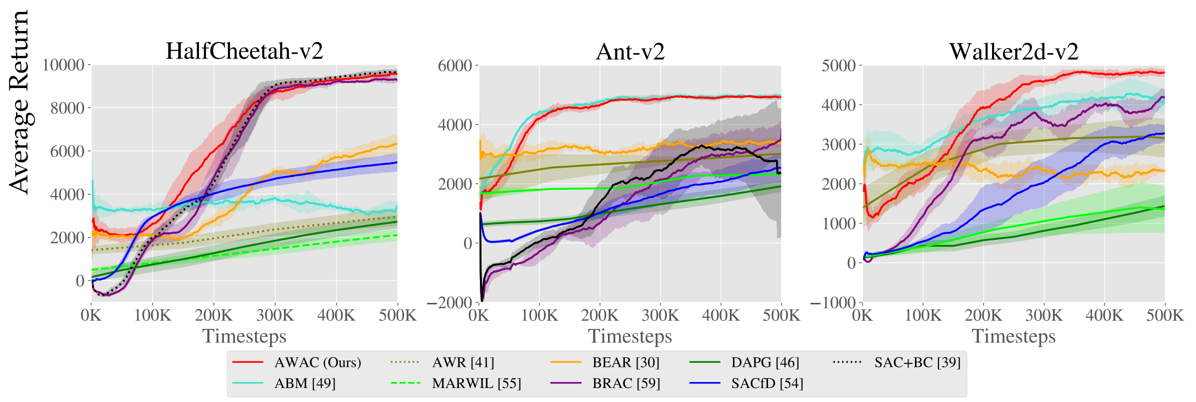


Figure 73: Comparison of our method and prior methods on standard MuJoCo benchmark tasks. These tasks are much easier than the dexterous manipulation tasks, and allow us to better inspect the performance of methods in the setting of offline pretraining followed by online fine-tuning. SAC+BC and BRAC perform on par with our method on the HalfCheetah task, and ABM performs on par with our method on the Ant task, while our method outperforms all others on the Walker2D task. Our method matches or exceeds the best prior method in all cases, whereas no other single prior method attains good performance on all of the tasks.

D.5 GYM BENCHMARK RESULTS FROM PRIOR DATA

In this section, we provide a comparative evaluation on MuJoCo benchmark tasks for analysis. These tasks are simpler, with dense rewards and relatively lower action and observation dimensionality. Thus, many prior methods can make good progress on these tasks. These experiments allow us to understand more precisely which design decisions are crucial. For each task, we collect 15 demonstration trajectories using a pre-trained expert on each task, and 100 trajectories of off-policy data by rolling out a behavioral cloned policy trained on the demonstrations. The same data is made available to all methods. The results are presented in Figure 73. AWAC is consistently the best or on par with the best-performing method. No other single method consistently attains the best results – on HalfCheetah, SAC + BC and BRAC are competitive, while on Ant-v2 ABM is competitive with AWAC. We summarize the results according to the challenges in Section 6.3.

Data efficiency. The three methods that do not estimate Q^π are DAPG (Rajeswaran et al., 2018), AWR (Peng et al., 2019a), and MARWIL (Q. Wang et al., 2018a). Across all three tasks, we see that these methods are somewhat worse offline than the best performing offline methods, and exhibit steady but very slow improvement during fine-tuning. In robotics, data efficiency is vital, so these algorithms are not good candidates for practical real-world applications.

Bootstrap error in offline learning. For SAC (Haarnoja et al., 2018a), across all three tasks, we see that the offline performance at epoch 0 is generally poor. Due to the data in the replay buffer, SAC with prior data does learn faster than from scratch, but AWAC is faster to solve the tasks in general. SAC with additional data in the replay buffer is similar to the approach proposed by Veerk et al. (2017). SAC+BC reproduces A. Nair et al. (2018a) but uses SAC instead of DDPG (Lillicrap et al., 2016) as the underlying RL algorithm. We find that these algorithms exhibit a characteristic dip at the start of learning. Although this dip is only present in the early part of the learning curve, a poor initial policy and lack of steady policy improvement can be a safety concern and a significant hindrance in real-world applications. Moreover, recall that in the more difficult dextrous manipulation tasks, these algorithms do not show any significant learning.

Conservative online learning. Finally, we consider conservative offline algorithms: ABM (Noah Y. Siegel et al., 2020b), BEAR (Kumar et al., 2019a), and BRAC (Yifan Wu et al., 2020). We found that BRAC performs similarly to SAC for working hyperparameters. BEAR trains well offline – on Ant and Walker2d, BEAR significantly outperforms prior methods before online experience. However, online improvement is slow for BEAR and the final performance across all three tasks is much lower than AWAC. The closest in performance to our method is ABM, which is comparable on Ant-v2, but much slower on other domains.

D.6 extra baseline comparisons (cql, algaedice)

In this section, we add comparisons to constrained Q-learning (CQL) (Kumar et al., 2020a) and AlgaeDICE (Nachum et al., 2019). For CQL, we use the authors' implementation, modified for additionally online-finetuning instead of only offline training. For AlgaeDICE, we use the publicly available implementation, modified to load prior data and perform 25K pretraining steps before online RL. The results are presented in Figure 74.

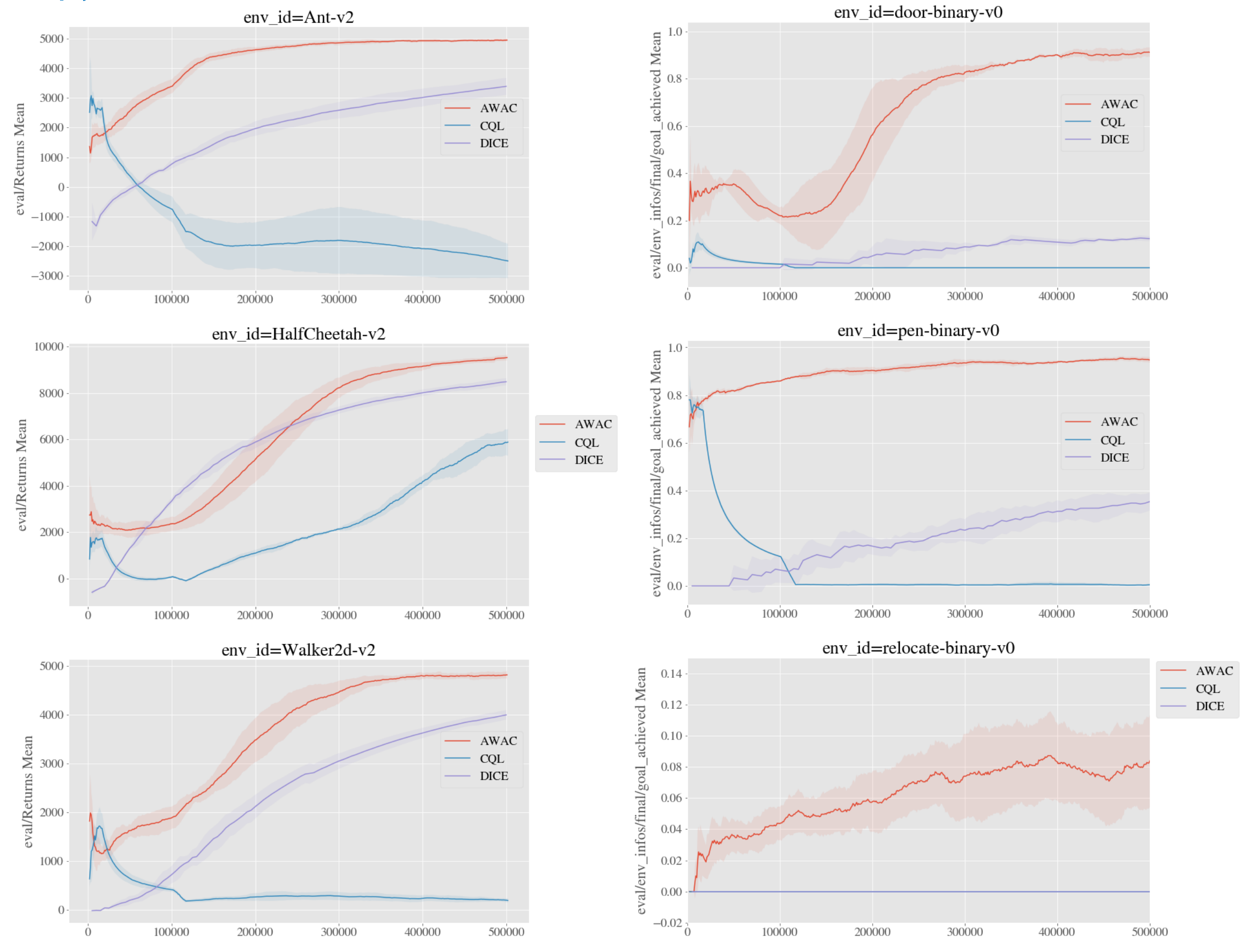


Figure 74: Comparison of our method (AWAC) with CQL and AlgaeDICE. CQL and AWAC perform similarly offline, but CQL does not improve when fine-tuning online. AlgaeDICE does not perform well for offline pretraining.

D.7 Online fine-tuning from D4RL

In this experiment, we evaluate the performance of varied data quality (random, medium, medium-expert, and expert) datasets included in D4RL (Fu et al., 2020), a dataset intended for offline RL. The results are obtained by first by training offline and then fine-tuning online on each setting for 500,000 additional steps. The performance of BEAR (Kumar et al., 2019a) is attached as reference. We attempted to fine-tune BEAR online using the same protocol as AWAC but the performance did not improve and often decreased; thus we report the offline performance. All performances are scaled to 0 to 100, where 0 is the average returns of a random policy and 100 is the average returns of an expert policy (obtained by training online with SAC), as is standard for D4RL.

The results are presented in Figure 75. First, we observe that AWAC (offline) is competitive with BEAR, a commonly used offline RL algorithm. Then, AWAC is able to make progress in solving the tasks with online fine-tuning, even when initialized from random data or "medium" quality data, as shown by the performance of AWAC (online). In almost all settings, AWAC (online) is the best performing or tied with BEAR. In four of the six lower quality (random or medium) data settings, AWAC (online) is significantly better than BEAR; it is reasonable that AWAC excels in the lower-quality data regime because there is more room for online improvement, while both offline RL methods often start at high performance when initialized from higher-quality data.

		AWAC (offline)	AWAC (online)	BEAR
HalfCheetah	random	2.2	**52.9**	25.5
	medium	37.4	41.1	38.6
	medium-expert	36.8	41.0	**51.7**
	expert	78.5	105.6	108.2
Hopper	random	9.6	**62.8**	9.5
	medium	72.0	**91.0**	47.6
	medium-expert	80.9	**111.9**	4.0
	expert	85.2	111.8	110.3
Walker2D	random	5.1	11.7	6.7
	medium	30.1	**79.1**	33.2
	medium-expert	42.7	**78.3**	10.8
	expert	57.0	103.0	106.1

Figure 75: Comparison of our method (AWAC) fine-tuning on varying data quality datasets in D4RL (Fu et al., 2020). AWAC is able to improve its offline performance by further fine-tuning online.

D.8 HARDWARE EXPERIMENTAL SETUP

Here, we provide further details of the hardware experimental setups, which are pictured in Fig 76.

Dexterous Manipulation with a 3 Fingered Claw.

- State space: 22 dimensions, consisting of joint angles of the robot and rotational position of the object.
- Action space: 9 dimensions, consisting of desired joint angles of the robot.
- Reward: -1 if the valve is rotated within 0.25 radians of the target, and 0 otherwise.
- Prior data: 10 demonstrations collected by kinesthetic teaching and 200 trajectories

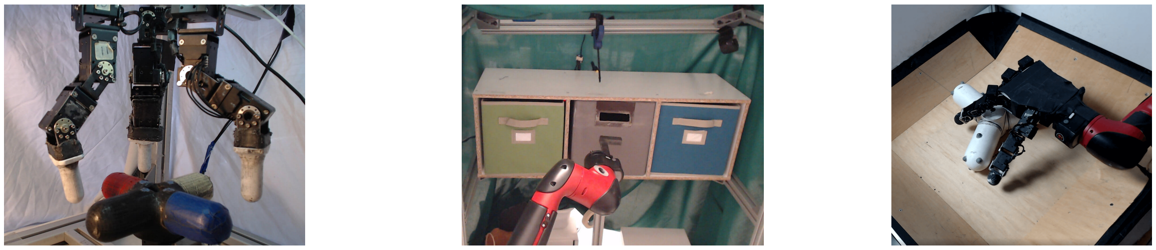


Figure 76: Full views of the robot hardware setups. Videos are available at *awacrl.github.io*

of behavior cloned data.

Drawer Opening with a Sawyer Arm.

- State space: 4 dimensions, end effector position of the robot and rotational position of the motor attached to the drawer.
- Action space: 3 dimensions, for velocity control of end-effector position.
- Reward: -1 if the motor is rotated more than 15 radians of the reset position, and 0 otherwise.
- Prior data: 10 demonstrations collected using a 3DConnexion Spacemouse device and 500 trajectories of behavior cloning data.

Dexterous Manipulation with a Robotic Hand.

- State space: 25 dimensions, consisting of joint angles of the hand, end effector positions of the arm, object position and target position.
- Action space: 19 dimensions, consisting of desired 16 joint angles of the hand and 3 dimensions for end-effector control of the arm.
- Reward: let o be the position of the object, h be the position of the hand, and g be the target location of the object. Then $r = -||o - h|| - 3||o - g||$.
- Prior data: 19 demonstrations obtained via kinesthetic teaching and 50 trajectories of behavior cloned data.

E

APPENDIX: CHAPTER 7

E.1 PROOFS

E.1.1 *Proof of Lemma 4*

Proof. We can rewrite $V_{\tau_1}(s)$ as

$$\begin{aligned}
V_{\tau_1}(s) &= \mathbb{E}^{\tau_1}_{a\sim\mu(\cdot|s)}[r(s,a)+\gamma\mathbb{E}_{s'\sim p(\cdot|s,a)}[V_{\tau_1}(s')]] \\
&\leqslant \mathbb{E}^{\tau_2}_{a\sim\mu(\cdot|s)}[r(s,a)+\gamma\mathbb{E}_{s'\sim p(\cdot|s,a)}[V_{\tau_1}(s')]] \\
&= \mathbb{E}^{\tau_2}_{a\sim\mu(\cdot|s)}[r(s,a)+\gamma\mathbb{E}_{s'\sim p(\cdot|s,a)}\mathbb{E}^{\tau_1}_{a'\sim\mu(\cdot|s')}[r(s',a')+\gamma\mathbb{E}_{s''\sim p(\cdot|s',a')}[V_{\tau_1}(s'')]] \\
&\leqslant \mathbb{E}^{\tau_2}_{a\sim\mu(\cdot|s)}[r(s,a)+\gamma\mathbb{E}_{s'\sim p(\cdot|s,a)}\mathbb{E}^{\tau_2}_{a'\sim\mu(\cdot|s')}[r(s',a')+\gamma\mathbb{E}_{s''\sim p(\cdot|s',a')}[V_{\tau_1}(s'')]] \\
&= \mathbb{E}^{\tau_2}_{a\sim\mu(\cdot|s)}[r(s,a)+\gamma\mathbb{E}_{s'\sim p(\cdot|s,a)}\mathbb{E}^{\tau_2}_{a'\sim\mu(\cdot|s')}[r(s',a')+\gamma\mathbb{E}_{s''\sim p(\cdot|s',a')}\mathbb{E}^{\tau_1}_{a''\sim\mu(\cdot|s'')}[r(s'',a'')+\ldots]] \\
&\vdots \\
&\leqslant V_{\tau_2}(s)
\end{aligned}$$

$\square$

E.2 EXPERIMENTAL DETAILS

EXPERIMENTAL DETAILS. For the MuJoCo locomotion tasks, we average mean returns overs 10 evaluation trajectories and 10 random seeds. For the Ant Maze tasks, we average over 100 evaluation trajectories. We standardize MuJoCo locomotion task rewards by dividing by the difference of returns of the best and worst trajectories in each dataset. Following the suggestions of the authors of the dataset, we subtract 1 from rewards for the Ant Maze datasets. We use $\tau = 0.9$ and $\beta = 10.0$ for Ant Maze tasks and $\tau = 0.7$ and $\beta = 3.0$ for MuJoCo locomotion tasks. We use Adam optimizer (D. Kingma and Ba, 2015) with a learning rate $3 \cdot 10^{-4}$ and 2 layer MLP with ReLU activations and 256 hidden units for all networks. We use cosine schedule for the actor learning rate. We parameterize the policy as a Gaussian distribution with a state-independent standard deviation. We update the target network with soft updates with parameter $\alpha = 0.005$. And following Brandfonbrener et al., 2021 we clip exponentiated advantages to $(-\infty, 100]$. We implemented our method in the JAX (Bradbury et al., 2018) framework using the Flax (Heek et al., 2020) neural networks library.

extended results on locomotion and ant maze tasks. We present learning curves for MuJoCo locomotion tasks in Figure 77. We also present results on Locomotion and Ant Maze for different values of τ in Figure 78 and Table 19. We want to emphasize that $\tau = 0.5$ corresponds to using the mean squared error instead of expectile regression.

Table 19: Effect of τ. Fitting $V(s)$ with mean squared error ($\tau = 0.5$) is not sufficient to propagate the signal through recursion and fails to solve more challenging medium and large tasks.

	IQL w/ $\tau = 0.5$ (MSE)	IQL w/ $\tau = 0.7$	IQL w/ $\tau = 0.9$
antmaze-umaze-v0	44.2±7.2	87.0±2.3	87.5±2.6
antmaze-umaze-diverse-v0	53.6 ±12.7	57.2±11.9	62.2±13.8
antmaze-medium-play-v0	0.0±0.0	4.0±2.0	71.2 ±7.3
antmaze-medium-diverse-v0	0.0 ±0.0	2.6±1.4	70.0±10.9
antmaze-large-play-v0	0.0±0.0	0.2±0.4	39.6 ±5.8
antmaze-large-diverse-v0	0.0 ±0.0	1.2±1.6	47.5±9.5
total	97.8±19.9	152.2±19.6	378.0±49.9

results on franca kitchen and adoit tasks. For Franca Kitchen and Adroit tasks we use $\tau = 0.7$ and the inverse temperature $\beta = 0.5$. Due to the size of the dataset, we also apply Dropout (N. Srivastava et al., 2014) with dropout rate of 0.1 to regularize the policy network. See complete results in Table 20.

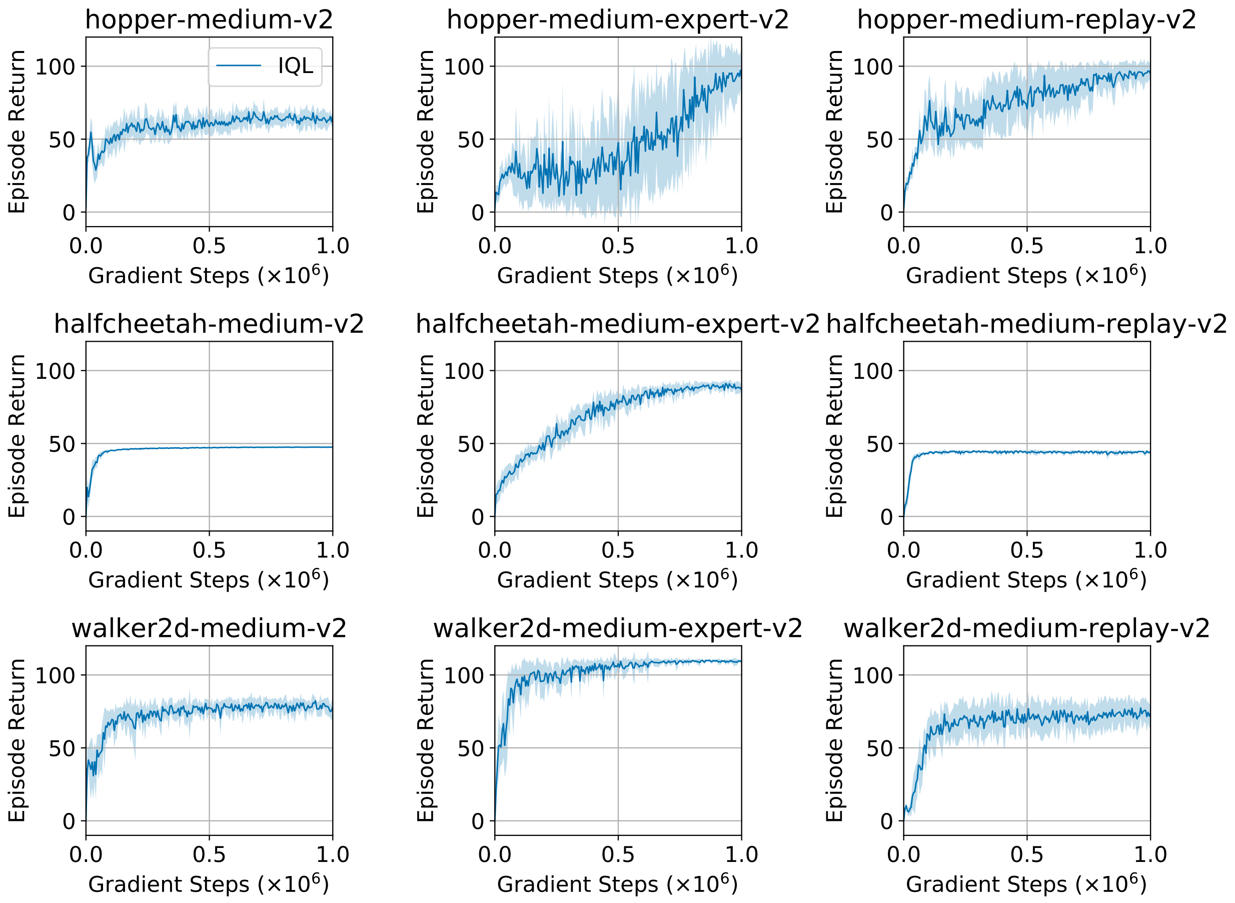


Figure 77: Learning curves for MuJoCo locomotion tasks.

Table 20: Evaluation on Franca Kitchen and Adroit tasks from D4RL

dataset	BC	BRAC-p	BEAR	Onestep RL	CQL	Ours
kitchen-complete-vo	**65.0**	0.0	0.0	-	43.8	**62.5**
kitchen-partial-vo	38.0	0.0	0.0	-	**49.8**	**46.3**
kitchen-mixed-vo	**51.5**	0.0	0.0	-	**51.0**	**51.0**
kitchen-vo total	**154.5**	0.0	0.0	-	144.6	**159.8**
pen-human-vo	63.9	8.1	-1.0	-	37.5	**71.5**
hammer-human-vo	1.2	0.3	0.3	-	**4.4**	1.4
door-human-vo	2	-0.3	-0.3	-	**9.9**	4.3
relocate-human-vo	0.1	-0.3	-0.3	-	0.2	0.1
pen-cloned-vo	37	1.6	26.5	**60.0**	39.2	37.3
hammer-cloned-vo	0.6	0.3	0.3	**2.1**	**2.1**	**2.1**
door-cloned-vo	0.0	-0.1	-0.1	0.4	0.4	**1.6**
relocate-cloned-vo	-0.3	-0.3	-0.3	-0.1	-0.1	-0.2
adroit-vo total	104.5	9.3	25.1	-	93.6	**118.1**
total	259	9.3	25.1	-	238.2	**277.9**

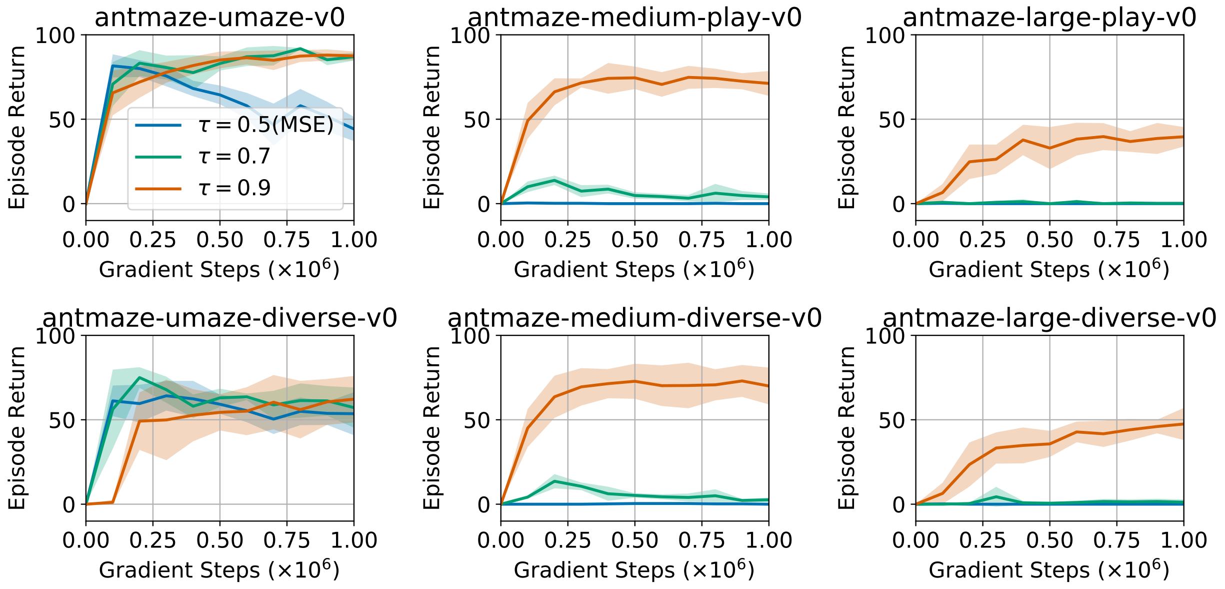


Figure 78: Results on Ant Maze for different values of τ. Note that $\tau = 0.5$ corresponds to using the mean squared error instead of expectile regression.

E.3 FINETUNING EXPERIMENTAL DETAILS

For finetuning experiments, we first run offline RL for 1M gradient steps. Then we continue training while collecting data actively in the environment and adding that data to the replay buffer, running 1 gradient update / environment step. All other training details are kept the same between the offline RL phase and the online RL phase. For dextrous manipulation environments (Rajeswaran et al., 2018), we use $\tau = 0.8$ and $\beta = 3.0$, 25000 offline training steps, and add Gaussian noise with standard deviation $\sigma = 0.03$ to the policy for exploration.

For baselines we compare to the original implementations of AWAC (A. Nair et al., 2020) and CQL (Kumar et al., 2020b). For AWAC we used `https://github.com/rail-berkeley/rlkit/tree/master/rlkit`. We found AWAC to overfit heavily with too many offline gradient steps, and instead used 25000 offline gradient steps as in the original paper. For the dextrous manipulation results, we report average return normalized from 0 to 100 for consistency, instead of success rate at the final timestep, as reported in A. Nair et al. (2020). For CQL, we used `https://github.com/aviralkumar2907/CQL`. Our reproduced results offline are worse than the reported results, particularly on medium and large antmaze environments. We were not able to improve these results after checking

Dataset	Offline	Online
antmaze-umaze-v0	70.1 → **99.4**	**86.7** → **96.0**
antmaze-umaze-diverse-v0	31.1 → **99.4**	**75.0** → 84.0
antmaze-medium-play-v0	23.0 → 0.0	**72.0** → **95.0**
antmaze-medium-diverse-v0	23.0 → 32.3	**68.3** → **92.0**
antmaze-large-play-v0	1.0 → 0.0	**25.5** → **46.0**
antmaze-large-diverse-v0	1.0 → 0.0	**42.6** → **60.7**
pen-binary-v0	46.2 ± 6.3	63.3 ± 1.9
door-binary-v0	1.3 ± 0.7	42.0 ± 3.2
relocate-binary-v0	0.3 ± 0.4	23.3 ± 14.5

Table 21: Error bars for fine-tuning experiments with 20 seeds, showing one standard deviation.

for discrepancies with the CQL paper authors and running CQL with an alternative implementation (`https://github.com/tensorflow/agents`). Thus, although for offline experiments (Table 1) we report results from the original paper, for finetuning experiments we did not have this option and report our own results running CQL in Table 21.

E.4 CONNECTIONS TO PRIOR WORK

In this section, we discuss how our approach is related to prior work on offline reinforcement learning. In particular, we discuss connections to BCQ Fujimoto et al., 2019b.

Our batch constrained optimization objective is similar to BCQ (Fujimoto et al., 2019b). In particular, the authors of BCQ build on the Q-learning framework and define the policy as

$$\pi(s) = \underset{\substack{a \\ \text{s.t.}(s,a)\in\mathcal{D}}}{\arg\max}\, Q(s,a). \tag{83}$$

Note that in contrast to the standard Q-learning, maximization in Equation (83) is performed only over the state-action pairs that appear in the dataset. In Fujimoto et al., 2019b, these constraints are implemented via fitting a generative model $\mu(\cdot|s)$ on the dataset, sampling several candidate actions from this generative model, and taking an argmax over these actions:

$$\pi(s) = \operatorname*{arg\,max}_{\{a_i | a_i \sim \mu(\cdot|s), i=1\ldots N\}} Q(s, a_i).$$

However, this generative model can still produce out-of-dataset actions that will lead to querying undefined Q-values. Thus, our work introduces an alternative way to optimize this objective without requiring an additional density model. Our approach avoids this issue by enforcing the hard constraints via estimating expectiles. Also, it is worth mentioning that a number of sampled actions N in BCQ has similar properties to choosing a particular expectile τ in our approach.

Note that our algorithm for optimal value approximation does not require an explicit policy, in contrast to other algorithms for offline reinforcement learning for continuous action spaces (Fujimoto et al., 2019b; Fujimoto and S. S. Gu, 2021; Yifan Wu et al., 2019; Kostrikov et al., 2021a; Kumar et al., 2019b; Kumar et al., 2020b). Thus, we do not need to alternate between actor and critic updates, though with continuous actions, we must still extract an actor at the end once the critic converges.

E.5 DIFFERENT ESTIMATORS OF $V(s)$

We also evaluate different ways to estimate the value function $V(s)$ (Table 22). We compare $V(s)$ learned with expectile regression as in IQL with $V(s)$ estimated with several samples from the learned policy as in ABM (Noah Y Siegel et al., 2020a). In particular, we use $N = 20$ to estimate the value function.

Table 22: Different estimators of $V(s)$

	IQL	$V(s) = \sum_{i=1}^{N} Q(s, a_i)/N$
hopper-medium-v2	66.2±5.7	69.5±3.9
hopper-medium-expert-v2	91.5±14.3	75.8±37.8
hopper-medium-replay-v2	94.7±8.6	64.7±22.6
halfcheetah-medium-v2	47.4±0.2	47.2±0.2
halfcheetah-medium-expert-v2	86.7±5.3	93.0±0.9
halfcheetah-medium-replay-v2	44.2±1.2	45.1±0.3
walker2d-medium-v2	78.3±8.7	72.0±24.6
walker2d-medium-expert-v2	109.6±1.0	110.7±0.4
walker2d-medium-replay-v2	73.8±7.1	83.3±3.0
locomotion total	692.4±52.1	661.4±93.7
antmaze-umaze-v0	87.5±2.6	96.4±1.8
antmaze-medium-play-v0	71.2±7.3	0.0±0.0
antmaze-large-play-v0	39.6±5.8	0.0±0.0
antmaze-umaze-diverse-v0	62.2±13.8	57.5±6.3
antmaze-medium-diverse-v0	70.0±10.9	0.0±0.0
antmaze-large-diverse-v0	47.5±9.5	0.0±0.0
antmaze total	378.0±49.9	153.9±8.1

F

APPENDIX: CHAPTER 11

F.0.1 *Real-World Experimental Details*

Our real-world data used in experiments consists of 830 trajectories (61,482 transitions) collected by a human using a 3Dconnexion SpaceMouse device. Instructions for interfacing with the SpaceMouse is available publicly at `https://github.com/vitchyr/rlkit/tree/master/rlkit/demos/spacemouse`, with the device code adapted from the RoboSuite library Y. Zhu et al., 2020. A full view of the robot and the view from the camera can be seen in Figure 79.

Across our dataset, we interact with 10 drawer handles, 10 pot handles, 40 toys, and 60 distractor objects. Every 10 trajectories we randomly sample one or more interaction objects as well as two or more distractor objects. Before each rollout we randomize all object positions. The trajectories can be grouped into four separate categories: picking and placing toys, putting toys into a tray, opening a door and closing a drawer, and placing and removing a lid on a pot. To artificially increase the size of our dataset and make our policy robust to light changes and camera nudges, we utilize color jittering and random cropping during training. As there was an unequal amount of data per category, we re-balanced the dataset by using a different number of data augmentations per category. The final amount of task-specific data used per experiment is reported in Table 28.

We additionally collected unscripted play data mixing all of the above behaviors with more object diversity, but did not use this data in this paper. With this data, there are 1,984 trajectories (137,111 transitions), covering 20 drawer handles, 20 pot handles, 60 toys, and 60 distractor objects. We also collected an additional 508 trajectories of on-policy robot data. The entire dataset is available on our website: `https://sites.google.com/view/val-rl`.

F.0.2 *Simulation Experimental Details*

Our simulated dataset consists of 8,000 trajectories (400,000 transitions). Before sampling each trajectory, we randomize the existence, position, color, and orientation of the following: two drawers, a box, a button, and an object. If an object is present it is chosen from a set of 84 object geometries. The trajectories are generated by a scripted policy which collects play data by interacting with all the present objects in a random order. The scripted behavior includes: opening and closing a drawer by the handle, opening and closing a different drawer by pressing a button, and re-positioning objects. All simulated RL experiments were run with 5 seeds.

F.0.3 *Algorithm Details*

Visuomotor affordance learning (VAL) builds off the `rlkit` codebase available at `https://github.com/vitchyr/rlkit`. We will release our code at our website, `https://sites.google.com/view/val-rl`. Below, we list the specific hyperparameters used in our experiments for each component. In VAL, we first collect an offline data $\mathcal{D}$, run representation learning, then offline RL, and finally online RL for a specific environment.

In the representation learning phase, we first train the VQVAE Aaron van den Oord et al., 2017 on $\mathcal{D}$. We then encode the entire dataset with the VQVAE to obtain discrete latent variables, and then independently train the PixelCNN Aaron van den Oord et al., 2016 on discrete latent code dataset. For the CCRIG experiments, we train a CCVAE Sohn et al., 2015 on $\mathcal{D}$.

In the offline RL phase, we run advantage weighted actor critic (AWAC) A. Nair et al., 2020 on the offline data to obtain a single policy and Q-function. This policy and Q-function can then be fine-tuned to a specific environment by running online RL.

All hyperparameters are provided below for these algorithms are provided below in tables 23, 24, 25, 26, 27.

Hyper-parameter	**Value**
Training Batches Per Timestep	1
Exploration Noise	None (stochastic policy)
RL Batch Size	1024
Discount Factor	0.99
Reward Scaling	1
Replay Buffer Size	1000000
Number of pretraining steps	25000
Policy Hidden Sizes	$[256, 256, 256, 256]$
Policy Hidden Activation	ReLU
Policy Weight Decay	10^{-4}
Policy Learning Rate	3×10^{-4}
Q Hidden Sizes	$[256, 256]$
Q Hidden Activation	ReLU
Q Weight Decay	0
Q Learning Rate	3×10^{-4}
Target Network τ	5×10^{-3}
Relabeling strategy $p_{RS}(\mathbf{z})$	50% future, 30% prior, 20% rollout

Table 23: Hyper-parameters used for RL (AWAC) experiments.

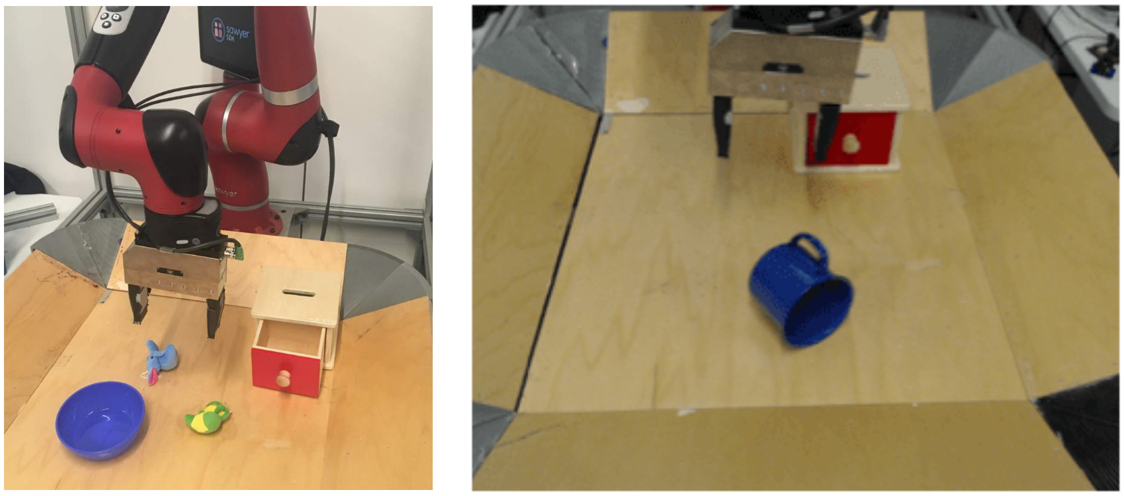


Figure 79: Left, full view of Sawyer robot setup. Right, camera view.

Hyper-parameter	Value
Brightness (Color Jitter)	$[0.75, 1.25]$
Contrast (Color Jitter)	$[0.9, 1.1]$
Saturation (Color Jitter)	$[0.9, 1.1]$
Hue (Color Jitter)	$[-0.1, 0.1]$
Size (Random Resized Crop)	48
Scale (Random Resized Crop)	$[0.9, 1.0]$
Ratio (Random Resized Crop)	$[0.9, 1.1]$
Interpolation (Random Resized Crop)	Antialiasing

Table 24: Hyper-parameters used for data augmentation.

Figure 80: Images from the prior dataset. The dataset contains 830 trajectories. We have released the full prior dataset (containing 1,984 trajectories) along with on-policy robot executions at our website, https://sites.google.com/view/val-rl

Hyper-parameter	**Value**
Convolution Layers	3
Convolution Hidden Size	128
Residual Layers	3
Residual Hidden Size	64
Embedding Size	5
Dictionary Size	512
Commitment Cost	0.25
EMA Embedding	False

Table 25: Hyper-parameters used for VQVAE training.

Hyper-parameter	**Value**
Batch Size	32
Layers	15
Learning Rate	0.0003
Latent Conditioning Type	Continuous

Table 26: Hyper-parameters used for PixelCNN experiments.

Hyper-parameter	**Value**
Convolution Layers	3
Convolution Hidden Size	128
Residual Layers	3
Residual Hidden Size	64
Embedding Size	5
Conditioning Embedding Size	1

Table 27: Hyper-parameters used for CCVAE training.

Hyper-parameter	**Value**
Tray	25% task specific data
Pick and Place	25% task specific data
Close Drawer	20% task specific data
Open Drawer	20% task specific data
Place Lid	17% task specific data

Table 28: Hyper-parameters used for task-specific replay buffer re-balancing (through data augmentation).