How to Train Your Robot: Techniques for Enabling Robotic Learning in the Real World

Abhishek Gupta

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2021-191
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-191.html

August 13, 2021

Acknowledgement

I would like to acknowledge my advisors - Pieter Abbeel and Sergey Levine, my committee - Ken Goldberg, Hannah Stuart, Trevor Darrell and my friends and family for their support.

How to Train Your Robot: Techniques for Enabling Robotic Learning in the Real World

by

Abhishek Gupta

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Sergey Levine, Co-chair
Pieter Abbeel, Co-chair
Ken Goldberg

Summer 2021

The dissertation of Abhishek Gupta, titled How to Train Your Robot: Techniques for Enabling Robotic Learning in the Real World, is approved:

Co-chair ____ Date ______________

Co-chair ____ Date ______________

____ Date ______________

University of California, Berkeley

How to Train Your Robot: Techniques for Enabling Robotic Learning in the Real World

Abstract

How to Train Your Robot: Techniques for Enabling Robotic Learning in the Real World

by

Abhishek Gupta

Doctor of Philosophy in Computer Science

University of California, Berkeley

Sergey Levine, Co-chair

Pieter Abbeel, Co-chair

Reinforcement learning has been a powerful tool for building continuously improving systems in domains like video games and animated character control, but has proven relatively more challenging to apply to problems in real world robotics. In this talk, I will argue that this challenge can be attributed to a mismatch in assumptions between typical RL algorithms and what the real world actually provides, making data collection and utilization difficult. In this talk, I will discuss how to build algorithms and systems to bridge these assumptions and allow robotic learning systems to operate under the assumptions of the real world - under realistic and practical assumptions on non-determinism, uncertainty and human supervision. In particular, I will describe how we can develop algorithms to ensure easily scalable supervision from humans, perform safe, directed exploration in practical time scales and enable uninterrupted autonomous data collection at scale. I will show how these techniques can be applied to real world robotic systems. Lastly, I will provide some perspectives on how this opens the door towards future deployment of robots into unstructured human-centric environments such as our homes, hospitals and shopping malls.

Acknowledgments

I first recall mentioning I wanted to pursue a Ph.D and become a professor back in the 12th grade. A lot has happened since then, and I owe it all to the many people who have been such a crucial part of my journey and supported me through it all. I truly believe that the outcomes in your life are a product of the doors that have been opened for you, whether it be by your parents, advisors or friends but I have been blessed with a wonderful support system that has kept me going through the last 6 years and has taught me everything.

First of all I would like to thank the members of my committee. The advice and guidance of Trevor Darrell, Hannah Stuart and Ken Goldberg has helped reshape the way I think about research problems and I feel very grateful to have been a beneficiary of their time and guidance.

If I were to name two people who have changed my life the most in the last 10 years, it would be Sergey Levine and Pieter Abbeel. It is hard for me to express in words how they've both impacted me in their own unique ways. Being advised by both of them has been an amazing privilege and I owe them both a tremendous debt of gratitude for all they have taught me about research, mentorship and life. Pieter was one of the first people to take a chance on me, as a freshman back in 2012. He has continued to push me to be a better researcher, a more visionary thinker and to be independent in all my endeavors. I am grateful to him for being my foremost advocate, and being a wonderful friend. Sergey has been my closest collaborator, sounding board and mentor over the last 6 years. No one has pushed me more than him, whether it be in terms of technical expertise, mentorship or taking risks, and I have grown immensely from our interactions. Both Pieter and Sergey have my heartfelt gratitude for all that they have done for me, I know we will continue to be friends and collaborators for many years to come.

Besides my advisors I have been fortunate to have excellent mentors along the way who have taught me how to approach research the right way and have shown me how to be an excellent mentor and collaborator. Particularly, I would like to thank Karol Hausman, Chelsea Finn and Vikash Kumar, Alex Lee for their friendship and mentorship. They have guided me through the rough times and celebrated the good ones with me and I am lucky to have had the opportunity to work with them.

I would also like to thank my collaborators who have made the work in this thesis possible - Coline Devin, Clemens Eppner, YuXuan Liu, Aravind Rajeswaran, Giulia Vezzanni, Siddhartha Srivastava, Ben Eysenbach, Russell Mendonca, Chelsea Finn, Karol Hausman, Vikash Kumar, JD Co-Reyes, Michael Chang, Dibya Ghosh, Nick Altieri, Jacob Andreas, Henry Zhu, Justin Yu, Tony Zhao, Corey Lynch, Ashwin Reddy, Justin Fu, Allan Jabri, Kyle Hsu, Tianhe Yu, Saurabh Kumar, Aviral Kumar, Dhruv Shah, Avi Singh, Kristian Hartikainen, Ashvin Nair, Murtaza Dalal, Glen Berseth, Charles Sun, Brandon Kinman, Garrett Peake, Kelvin Xu, Marvin Zhang, Suvansh Sanjeev, Michael Ahn, Olivia Watkins, Kate Rakelly, Colin Li and many others. Each collaboration has been a wonderful learning experience and I am thankful to have met each of you during my time at Berkeley.

I also want to thank my lab-mates both in RAIL and RLL, and the larger BAIR community for making walking into SDH such a joyful experience. I will greatly miss our Thursday afternoon chats and just laying on the floor at 3am right before a paper deadline. In particular, I would like to thank friends from the 7th floor of SDH - Alex Lee, Marvin Zhang, Sandy Huang, Chelsea Finn, Greg Kahn, Coline Devin, Rocky Duan, Haoran Tang, Tuomas Haarnoja, Carlos Florensa, Adam Stooke, Thanard Kurutach, Vitchyr Pong, JD Co-Reyes, Anusha Nagabandi, Justin Fu, Sid Reddy, Ashvin Nair, Kelvin Xu, Erin Grant, Pulkit Agrawal, Deepak Pathak, Parsa Mahmoudieh, David Held, Aviv Tamar, Joshua Achiam, Bradly Stadie, Rowan McCallister, Frederik Ebert, Ignasi Clavera, Aviral Kumar, Xinyang Geng, Kate Rakelly, Somil Bansal for making SDH seem like home and for all the wonderful memories. I want to give special thanks to Greg Kahn, Coline Devin, Anusha Nagabandi and Marvin Zhang - I'm grateful for the dark humor, the commiseration, the brainstorming sessions, the words of encouragement, the evening NBA games, the LeConte parties and the late night La Burrita runs, you all are like family to me. And a very special shout out to Ignasi Clavera, keeping me sane through the long and stressful times, calling me out when I need it, pushing me to be the best version of myself. You've been an inspiration to me and I couldn't have asked for a better partner in crime.

I have been lucky to have an amazing group of friends who have supported me throughout my PhD journey, right from 2011. I'm grateful they have not given up on me despite me constantly being buried in work and busy with paper deadlines. In particular, I want to thank Aayush Dawra, Apratim Gupta, Rohan Bhasin, Ashmi Chakraborty, Siddhanth Puri, Kush Agrawal, William Wu, Garrett Gordon, Cody Holik, Shiv Sundram, Srijit Ghosh, Tanay Jaeel, Sahithi Rani, Jesar Shah, Daniel Machado, Sukriti Gandhi, Rabia Shah, Navneet Kahlon, Prithi Polavarapu, Shruti Dubey, Aashik Sekharan, Vishnu Jayaprakash, Antonia Acquistapace, Ronald Lee, Sidharth Gupta, Pranav Kaundinya, Prashanth Nambiar, Vinit Nayak, Sanat Daga, Saveen Sahni, Ramandeep Dhillon, Sneha Singh, Kate Rakelly, Mostafa Rohaninejad, Nikhil Mishra, Kunsel Tenzin, Erin Grant for being a constant source of support and encouragement.

Last but not least I want to thank my family for being my biggest supporters. My family - Madhurima Ghose, Satyanarayan Gupta and Anubha Ghose taught me how to pursue excellence without compromising ideals, how to push yourself to be better while helping others, how to question the little details and take nothing for granted. They have provided me with so much opportunity, supporting me at every step of my life and have been a constant source of energy, support and love. I want to thank my aunt and uncle - Mitali Biswas and Prasenjit Biswas for being second parents to me, supporting me through various ups and downs and being a consistent voice of reason. I want to thank my paternal grandparents - Kamala Devi and B.L Gupta for the many sacrifices they have made to bring our family to this point. I want to thank the Grewal family - especially Sukhbir Grewal for showing me kindness, love and compassion beyond the ordinary. I'm grateful to have all of you in my life. And most importantly, I want to thank Sabina Grewal for being the light in my life and providing me unwavering and constant support in every step of my journey.

Contents

Contents iii

1 Introduction **1**
1.1 Why Care About Robotic Learning? 1
1.2 What Learning Methodology should be used? 2
1.3 Reinforcement Learning 3
1.4 Real World Reinforcement Learning 4
1.5 Where does this work fit into the bigger picture of robotic learning? 7

I Supervision 9

2 Supervision from Human Videos **11**
2.1 Why Should We Learn from Raw Human Videos? 11
2.2 Relationship to Prior Work 13
2.3 Problem Formulation and Overview 14
2.4 Learning to Translate Between Contexts 15
2.5 Learning Policies via Context Translation 17
2.6 Experiments 18
2.7 Discussion and Future Work 23

3 Supervision from Outcome Examples **25**
3.1 Why Should we Study Uncertainty-Aware Outcome Driven RL? 25
3.2 Relationship to Prior Work 27
3.3 Preliminaries 27
3.4 Bayesian Success Classifiers for Reward Inference 29
3.5 MURAL: Training Uncertainty-Aware Success Classifiers for Outcome Driven RL via Meta-Learning and CNML 32
3.6 Experimental Evaluation 34
3.7 Discussion 37

4 Supervision from Language Corrections **38**
4.1 Why Should We Use Language Feedback to Supervise RL algorithms? 38

4.2 Relationship to Prior Work 39
4.3 Problem Formulation 41
4.4 The Language-Guided Policy Learning Model 41
4.5 Meta-Training the GPL Model to Learn From Corrections 42
4.6 Learning New Tasks with The GPL Model 44
4.7 Experiments 44

5 Learning Skills Without Reward Supervision 51
5.1 Why Is Unsupervised Skill Discovery Important? 51
5.2 Related Work 52
5.3 Diversity is All You Need 53
5.4 Experiments 56
5.5 Conclusion 61

6 Unsupervised Pre-Training for Quick Reinforcement Learning 62
6.1 Motivating a General Unsupervised Meta-RL Framework 62
6.2 Related Work 63
6.3 Unsupervised Meta-RL 64
6.4 Experimental Evaluation 73
6.5 Discussion and Future Work 75

7 Relationship to Other Work on Supervision in Reinforcement Learning 76
7.1 Connections to Prior Work 76
7.2 Related Work Subsequent to Publishing 77

II Distributions 78

8 Bootstrapping On-Policy Reinforcement Learning with Human Demonstrations 81
8.1 Why Does Complex Dexterous Manipulation Require Demonstration Bootstrapped RL? 81
8.2 Related Work 83
8.3 Dexterous Manipulation Tasks 85
8.4 Demo Augmented Policy Gradient (DAPG) 89
8.5 Results and Discussion 91
8.6 Conclusion 95

9 Applying Bootstrapped On-Policy RL to Real World Robotic Systems 96
9.1 Contributions 96
9.2 Hardware Setup 97
9.3 Tasks 98
9.4 Experimental Results and Analysis 100
9.5 Discussion and Future Work 107

10 **Bootstrapping Hierarchical Reinforcement Learning with Human Demonstrations for Long Horizon Reasoning** **108**
10.1 How Can Demonstration Bootstrapped RL Solve Long Horizon Tasks? 108
10.2 Relationship to Prior Work 109
10.3 Relay Policy Learning 110
10.4 Experimental Results 114
10.5 Conclusion and Future Work 118

11 **Bootstrapping Off-Policy Reinforcement Learning with Offline Datasets and Online Finetuning** **119**
11.1 Why Should We Care About Bootstrapped Off-Policy RL? 119
11.2 Preliminaries 121
11.3 Challenges in Offline RL with Online Fine-tuning 122
11.4 Advantage Weighted Actor Critic: A Simple Algorithm for Fine-tuning from Offline Datasets 125
11.5 Related Work 127
11.6 Experimental Evaluation 129
11.7 Discussion and Future Work 133

12 **Relationship to Other Work on Bootstrapping Reinforcement Learning** **134**

III **Continual Data Collection** **135**

13 **Instrumentation Free Learning Systems for Real World Reinforcement Learning** **137**
13.1 Motivation 137
13.2 The Structure of a Real-World RL System 138
13.3 The Challenges of Real World RL 141
13.4 A Real-world Robotic Reinforcement Learning System 142
13.5 Related Work 144
13.6 Algorithm details 145
13.7 Experiments 146
13.8 Discussion 150

14 **Building Reset-Free Reinforcement Learning Algorithms via Multi-Task Learning** **151**
14.1 Introduction 151
14.2 Learning Dexterous Manipulation Behaviors Reset-Free via Multi-Task RL 153
14.3 Task and System Setup 155
14.4 Experimental Evaluation 158
14.5 Discussion 162

15 **Bootstrapping Reset-Free Reinforcement Learning Algorithms with Human Data** **164**
15.1 Introduction 164

15.2 Preliminaries and Problem Statement 166
15.3 Demonstration Augmented Autonomous Practicing for Multi-Task Reinforcement Learning 166
15.4 System Description 169
15.5 Experimental Evaluation 170
15.6 Discussion 173

16 Relationship to Other Work on Continual Data Collection in Reinforcement Learning 174

17 Conclusion 175

Bibliography 178

18 Appendices 213
18.1 Appendix A: Appendix for Chapter 3 214
18.2 Appendix B: Appendix for Chapter 4 234
18.3 Appendix C: Appendix for Chapter 5 239
18.4 Appenix D: Appendix for Chapter 6 254
18.5 Appendix E: Appendix for Chapter 10 258
18.6 Appendix F: Appendix for Chapter 11 261
18.7 Appendix G: Appendix for Chapter 13 272
18.8 Appendix H: Appendix for Chapter 14 281
18.9 Appendix I: Appendix for Chapter 15 290

Chapter 1

Introduction

1.1 Why Care About Robotic Learning?

The ultimate goal for many roboticists is to build robotic systems that are able to master complex skills involving object interaction, contact rich manipulation and unmapped navigation in complex, unstructured real world environments. The challenge with complex real world environments is the fact that these environments introduce a massive amount of diversity, complexity and variability that are significantly different from the environments that most robots find use in today —warehouses and very controlled navigation problems. The question becomes - what perception and control techniques for robotics can actually scale to these kinds of unstructured real world environments? For instance, can we build a robotic learning agent that is able to actually operate in someone's ever changing home?

A variety of techniques in control theory [1]–[3], motion planning [4]–[6], state machine driven robotic control [7], [8] and task and motion planning [9] have seen a lot of success in a variety of different problems like grasping and table-top rearrangement for instance. However, while these techniques have seen success in these types of problems in controlled settings, they often have prohibitively expensive requirements such as a known model of the world [6] or assume that there is minimal to no interaction with objects in the scene. These requirements are certainly fulfilled in many of the domains where robots are currently applied - bin picking, warehouse navigation, flat terrain navigation and so on. However, for the most general case discussed above, in unstructured open-world environments like a home, hospital or shopping mall, these algorithms struggle to scale without very significant human engineering and careful robot programming.

An alternative that has significant potential is using a learning based paradigm for robotic perception and control. Machine learning techniques [10]–[12] have shown tremendous potential in extracting features and being able to process high dimensional, unstructured data in domains like computer vision [10], [12] and natural language processing [13], [14]. These successes are a product of large amounts of data, powerful computational tools, expressive predictive models and powerful optimization tools [15]. Given the success of machine learning tools in these domains and their ability to scale more gracefully than rule based or expert systems to complex but data rich domains, we posit that these types of tools have the potential to scale well to unstructured, open world, environments like the home or a hospital with relatively little human engineering. This hypothesis is tangentially based on the success of machine learning systems in dealing with unstructured data in domains like computer vision, natural language and domains like protein folding.

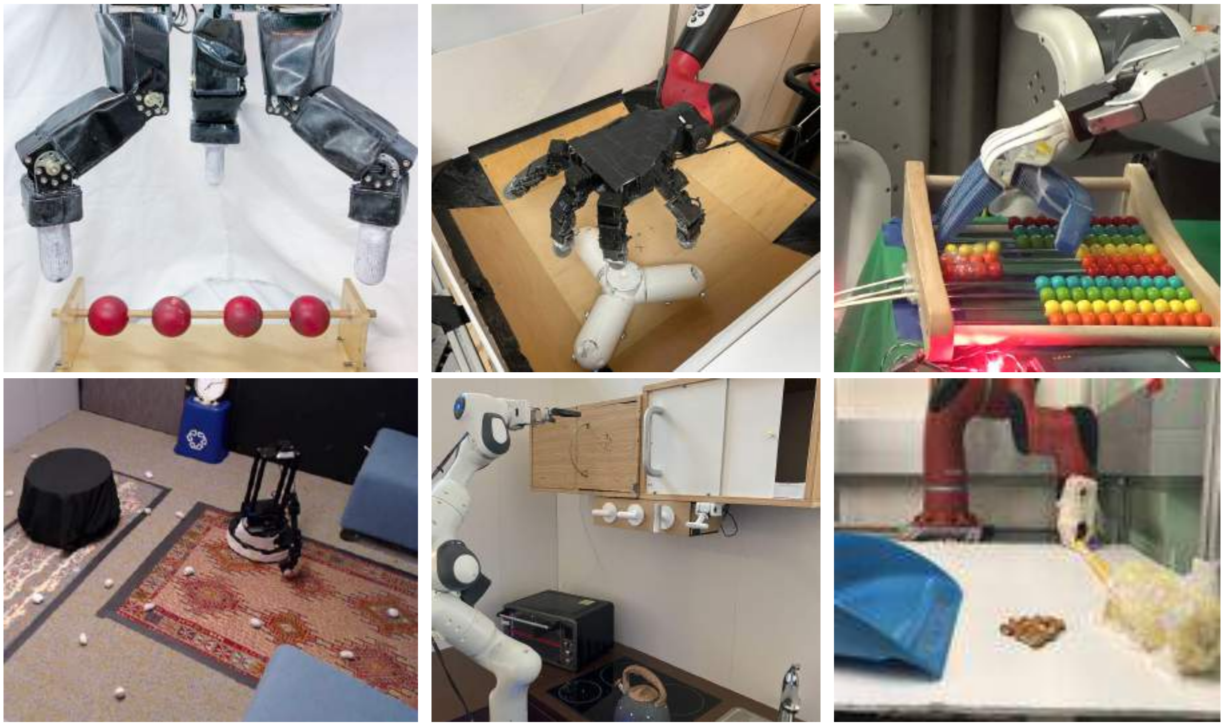


Figure 1.1: The goal of this thesis is to enable a variety of real world robotic systems to learn behaviors via reinforcement learning by training directly in the real world. Note that this is in contrast to the paradigm of training in simulation and transferring to the real world, as discussed in detail in Section 1.5

1.2 What Learning Methodology should be used?

Given that we want to apply a learning based algorithm to the aforementioned robotics problems, the question becomes —what type of learning algorithm should be utilized for robotic perception and control in unstructured environments? The answer to this question lies in the desiderata for a real world learning system; we want to be able to build continually improving learning systems that can keep improving as they collect more and more data in the real world. Given that the world is incredibly complex and unstructured, it's unlikely a model will be perfect when deployed in a new environment. In these scenarios, a continually improving learning system is able to adapt to new scenarios and deal with the variety of scenarios encountered in the real world. A typical supervised machine learning system [12] would assume access to a large labeled dataset, use it to learn a suitable model via techniques for maximum likelihood estimation and then simply deploy this model into the desired application. On the other hand, a continually improving learning system continues to collect data in a directed way on deployment, improving it's behavior as it is able to collect more and more experience in the environment. For instance, given a robotic home assistant, we'd want it to keep getting better the longer it actually spends in someone's home practicing different tasks.

Given the goal of building continually learning systems, reinforcement learning is a powerful paradigm for actually building systems that can keep improving by collecting their own data. Typically the paradigms of supervised or unsupervised learning rely on provided datasets, but reinforcement learning systems collect their own data and use this data to continue improving their behavior. While certain applications of supervised [16] or unsupervised learning [17] have also been explored in the continual setting where they are exposed to different tasks one after another, these techniques do not actually collect their own data with an embodied system and rely on this continuum of data to be provided by a human supervisor. To build systems that can collect their own data to keep improving autonomously, we delve deeper into reinforcement learning next.

1.3 Reinforcement Learning

Reinforcement learning can most intuitively be understood as the process of learning behaviors through repeated trial and error interactions with an environment, with the goal of maximizing some notion of reward in the environment. An example of this in our day to day interactions includes the mechanisms through which dogs are taught new tricks, through repeated interaction and tricks and the mechanisms through which babies (and even adults!) learn new motor and higher order skills. More formally, reinforcement learning usually operates in the formalism of Markov decision processes (MDP), with a MDP being denoted as $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma, \rho, H)$, where $\mathcal{S}$ represents the state space, $\mathcal{A}$ represents the action space, $\mathcal{T}$ represents the transition likelihood (often referred to as the dynamics of the world), $\mathcal{R}$ refers to the reward function, γ the discount factor, ρ the initial state distribution, and H the episode horizon. Under this notation, the goal of a reinforcement learning agent is to utilize interactions in the MDP to learn a policy $\pi(a|s)$ that learns how to command actions a at a state s, such that it maximizes the expected sum of discounted rewards.

$$\pi \leftarrow \arg\max_{\pi} \mathbb{E}_{s_0 \sim \rho, a_t \sim \pi(a_t|s_t), s_{t+1} \sim T(s_{t+1}|s_t, a_t)} \left[\sum_{t=0}^{H} \gamma^t r(s_t, a_t) \right] \tag{1.1}$$

This framework is particularly appealing because it simply requires sampling from the dynamics model T and reward function R, rather than an actual analytic model of the world. This allows reinforcement learning agents to operate with minimal pre-provided knowledge, instead obtaining an understanding of the dynamics of the world T, the reward function R through interaction with the environment. While this formalism is very succinct and convenient, as we will discuss next, there arise some challenges when the real world is represented under this formalism.

Very briefly, there are three major classes of solutions to problems in reinforcement learning - model free policy-gradient algorithms [18]–[20], model free dynamic programming algorithms [21]–

[23] and model-based algorithms [24]–[26]. While each of these techniques aim to maximize the expected return, they do so in very different ways. The policy gradient aims to directly learn the policy by performing gradient ascent on the expected return objective, model-based algorithms aim to learn an explicit model of transition dynamics T and reward function R, which can then be used to obtain an optimal policy, and dynamic programming algorithms make use of the famed notion of Bellman consistency in order to learn representative models of expected future return. The focus of this thesis is less on building better optimizers for the reinforcement learning objective and more on actually bridging the gap between the formalism as described here and the conditions present in real world robotics problems. Let us try and understand this gap next.

1.4 Real World Reinforcement Learning

The reinforcement learning formalism under the MDP framework is ideal for analytic problems [27], certain types of games [11], [28], simulated domains [29], and has seen tremendous success in these domains. However, the success of reinforcement learning in actually enabling "real-world" reinforcement learning has been relatively limited, either being restricted to very simplistic tasks [30], [31], tasks with significant instrumentation [32], [33], or domains which require large amounts of human engineering and intervention. This is not for lack of trying on the algorithmic front, algorithms for RL are constantly getting better at optimizing the RL objective. This thesis posits the limiting factor that has kept RL algorithms from widespread adoption in the domain of real world robotic learning, is the mismatch between the assumptions made by the typical RL formulation described above and what is actually available in the real world.

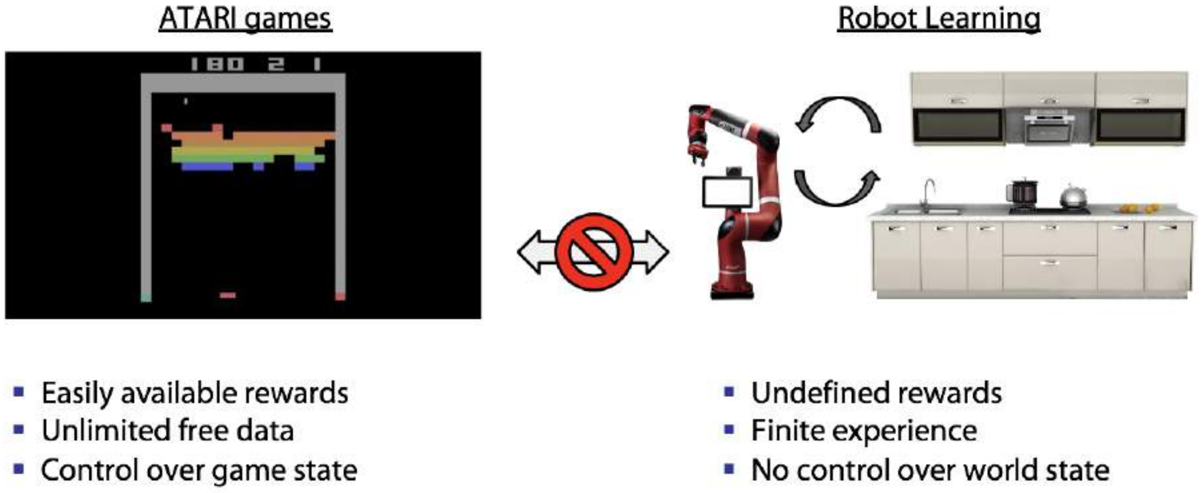


Figure 1.2: Mismatched assumptions between ATARI games and real world robotics

This is perhaps best illustrated by an example —let us consider the difference between a domain where the RL formalism is easy to satisfy, the ATARI video game domain, and domains where this is more challenging, for instance a robot operating in a kitchen. As we can see in Fig 1.2, the video game domain has naturally and easily available rewards in terms of the score, it allows for the collection of unlimited amounts of data for free, and it provides essentially complete control over game state to reset whenever needed to keep attempting the task over and over. On the other hand, a robot operating in a kitchen does not have a clearly defined notion of reward, it is limited to a finite amount of experience based on the system's physical constraints, and there is no oracle control over the state of the world to automatically provide resets and such. In addition, a real world environment faces a variety of other conditions such as. uncertainty in state, non-determinism in dynamics and complex physics that is often ignored in simulation or video games. There is clearly a significant mismatch between the assumptions across these domains, how can this be characterized more formally?

One way of thinking about this problem is through the lens of data. The mismatch between typical RL algorithms and the real world lies in how data is collected and utilized by these algorithms. In RL, unlike supervised learning or unsupervised learning, there is no fixed dataset that is provided to the agent; this has to be collected by the agent through interaction with the environment. This dataset can be represented as a set of state-action-next state tuples $\mathcal{D} = \{(s_0, a_0, s_1, r_0), (s_0, a_0, s_1, r_0), \cdots (s_N, a_N, s_{N+1}, r_N)\}$. For the sake of notation, let us also denote the (potentially non stationary) joint distribution that these tuples are drawn from by p. p is obviously a product of the transition dynamics of the world T and the current policy π. This notation is illustrated more clearly in Fig 1.4.

Given this form of data collected in RL, the mismatch in assumption lies in assumptions how the agent obtains the right supervision r (i.e. the right rewards), collects data from the right distribution p safely and efficiently, and finally ensuring that these systems are able to autonomously collect enough data N without unreasonable amounts of human effort. Most RL algorithms as they stand now make unreasonable assumptions about how r, p and N are obtained and these can often be difficult to actually satisfy in the real world. In this thesis, we argue that by dividing the real world RL problem into three different sub-problems - obtaining the right supervision (r), collecting data from the right distributions (p)and building systems for large scale data collection (N), we can actually start to deploy large scale deep reinforcement learning systems into real world environments. Throughout the rest of this thesis, I will provide a detailed account of different ways of tackling these problems - supervision in Part I, distributions in Part II and continual data collection in Part III.

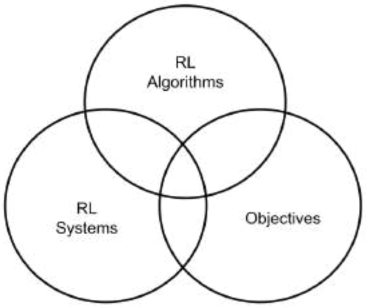


Figure 1.3: An overview of the different components of this thesis. In order to enable real world robotic learning and overcome mismatched assumptions between most RL algorithms and the real world, we require a combination of robust continually learning robotic learning systems, techniques for inferring suitable reward supervision and algorithms for utilizing these systems and rewards while bootstrapping from small amounts of human provided data.

Under Part I, in Section 2 we provide an overview of what supervision signifies in the context of real world RL. Following this in Section 3 we discuss techniques for learning rewards from raw human video, in Section 4 we discuss how to infer rewards from examples of successful outcomes, in Section 5 we discuss how we can infer reward functions from language feedback and in Section 6, 7 we discuss how we can actually learn behaviors *unsupervised* without any task-specific rewards. Through this discussion we show how moving reward inference towards a data driven process allows for scaling to real world scenarios, inferring rewards under realistic assumptions.

Under Part II, in Section 10, we discuss an algorithm for collecting data efficiently and in a directed way by combining on-policy RL and human demonstrations. Following this, in Section 11 we show how this can be applied on dexterous manipulation tasks in the real world. We then discuss how this paradigm can be extended to long horizon behaviors with hierarchical policy representations in Section 12. We then discuss how we can move from on-policy algorithms to significantly more efficient off-policy ones in Section 13. Through this discussion, we show how bootstrapping from a small amount of human provided data can allow for much more directed data collection and learning, scaling to much more complex tasks than was previously possible.

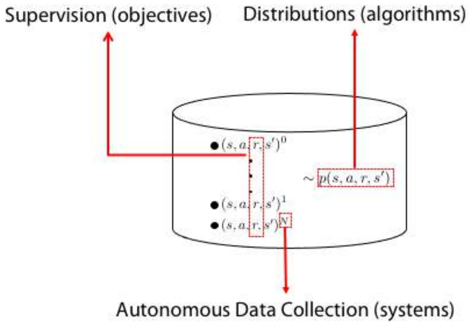


Figure 1.4: Data in the context of RL algorithms

Under Part III, we largely study the problem of instrumentation free, reset-free learning. In Section 17 we discuss a system for reset-free learning from visual inputs without any human intervention for simple dexterous manipulation problems. We then show how this paradigm can be extended to solve much more complex tasks via multi-task RL in Section 18. And finally, in Section 19 we discuss how the ideas from Part II can aid in improving the efficiency of the multi-task RL algorithm described in Section 18. Through this lens, we show that it is important to think about the systems problem for large scale robotic learning and we show that robotic learning systems constructed with these principles in mind can leverage large amounts of autonomously collected data to learn complex behaviors. In particular, this thesis covers content from the following papers with the listed co-authors and venues.

1. Y. Liu*, A. Gupta*, P. Abbeel, and S. Levine, “Imitation from observation: Learning to imitate behaviors from raw video via context translation,” International Conference on Robotics and Automation (ICRA) 2018

2. K. Li*, A. Gupta*, A. Reddy, V. Pong, A. Zhou, J. Yu, S. Levine, ”MURAL: Meta-Learning Uncertainty-Aware Rewards for Outcome-Driven," International Conference on Machine Learning (ICML) 2021

3. JD. Co-Reyes, A. Gupta, S. Sanjeev, N. Altieri, J. Andreas, J. DeNero, P. Abbeel, S. Levine, “Guiding Policies with Language via Meta-Learning," International Conference on Learning Representations (ICLR) 2018

4. B. Eysenbach, A. Gupta, J. Ibarz, S. Levine, “Diversity is All You Need: Learning Skills without a Reward Function", International Conference on Learning Representations (ICLR) 2018

5. A. Gupta*, B. Eysenbach*, C. Finn, S. Levine, “Unsupervised Meta-Learning for Reinforcement Learning", arXiv preprint 2020
6. A. Rajeswaran*, V. Kumar*, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, S. Levine, “Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations," Robotics Science and Systems (RSS) 2018
7. H. Zhu*, A. Gupta*, A. Rajeswaran, S. Levine, V. Kumar, “Dexterous Manipulation with Deep Reinforcement Learning: Efficient, General, and Low-Cost," International Conference on Robotics and Automation (ICRA) 2018
8. A. Gupta, V. Kumar, C. Lynch, S. Levine, K. Hausman, “Relay Policy Learning: Solving Long-Horizon Tasks via Imitation and Reinforcement Learning," Conference on Robot Learning (CoRL) 2019
9. A. Nair*, A. Gupta*, M. Dalal, S. Levine, “AWAC: Accelerating Online Reinforcement Learning with Offline Datasets," arXiv preprint 2020
10. H. Zhu*, J. Yu*, A. Gupta*, D. Shah, K. Hartikainen, A. Singh, V. Kumar, S. Levine, “The Ingredients of Real-World Robotic Reinforcement Learning," International Conference on Learning Representations (ICLR) 2020
11. A. Gupta*, J. Yu*, Z. Zhao*, V. Kumar*, A. Rovinsky, K. Xu, T. Devlin, S. Levine, “Reset-Free Reinforcement Learning via Multi-Task Learning: Learning Dexterous Manipulation Behaviors without Human Intervention," International Conference on Robotics and Automation (ICRA) 2021
12. A. Gupta, C. Lynch, B. Kinman, G. Peake, S. Levine, K. Hausman, “Demonstration-Augmented Autonomous Practicing via Multi-Task Reinforcement Learning," under review 2021

These co-authors really were tremendous and all the credit goes to them for putting together some amazing work.

1.5 Where does this work fit into the bigger picture of robotic learning?

It is important to understand how this thesis fits into the broader context of robotic learning. The techniques described in this thesis are largely techniques for enabling end to end deep reinforcement learning systems to solve robotic manipulation tasks. In the broader literature, end-to-end algorithms for robotic learning have been explored for manipulation [31], [34]–[37] as well as locomotion and navigation [38]–[40]. While many of these techniques are able to use reinforcement learning as a tool to learn how to solve certain tasks effectively, they have been restricted to relatively small amounts of training time [31], [33] or controlled training setups [31], [32], [35]. This thesis aims to remove those limitations and allow robotic learning agents to operate in natural environments, with minimal amounts of human instrumentation or intervention.

An alternative paradigm that is often quite popular is the paradigm of simulation to reality transfer for robotic learning [41]–[46]. In simulation to reality (sim2real) transfer algorithms, the agent is first trained in simulation using suitable robustness or adaptation techniques [41]–[43], following which the agent is deployed into the real world to perform a task at test time. The convenient part about this paradigm is that it is able to avoid the various training wheels needed for real world robotic learning and can utilize simulations in parallel [46], but it is challenging to scale to many new environments and tasks (especially those which are hard to simulate [47], [48]), not to mention the challenging transfer learning problem. For every new environment that the agent is deployed in, the simulation must be accurately created, tested and designed. This becomes tedious and impractical as robots are deployed at huge scale in homes, hospitals and shopping malls. This thesis argues that with suitable algorithmic and system developments, real world RL will scale significantly better than simulation to realty transfer. We would like to acknowledge the possibility that simulators drastically improve and improved inverse graphics techniques [49] make it easy to generate simulated environments just via images of the real world. In this scenario, the. applicability of simulation will probably be significantly increased but there is still likely to be some level of model mis-specification and uncertainty in this process. This will need real world finetuning, with actual real world data collection, making the ideas in this thesis very applicable even under this training paradigm.

Lastly, it is important to also consider algorithms for non “end-to-end" robotic learning. These techniques adopt a modular approach, where representations or state estimates are first extracted for the purposes of perception, following which an explicit control or planning algorithm can be deployed to command the robots actions [6], [9], [50]–[54]. Learning is often deployed in solving perception in these cases [50], [52], [54] identifying object types and poses, which allows for planning algorithms to solve high dimensional problems accordingly. While these techniques can be effective in simple scenarios, as scenes get more complex the burden of actually providing labels and supervision for a modular approach can become prohibitive. However, even more fundamentally, for large scale problems, the choice of modular abstraction chosen by an algorithm designer may not be the most effective given the data and can lead to poorly propagating errors between modules if not designed properly [55]. On the flip-side, some amount of structure can indeed make the learning process significantly easier, for instance imposing structure on the state and action space can make learning significantly more efficient. Answering the question of what structure is best, and how much structure is useful is out of scope of this thesis but forms a fascinating topic of study. In this thesis, we particularly focus on what it takes to actually enable (mostly) end to end learning systems to scale. As we discuss in Part 2 and Part 3, this does not mean we begin tabula rasa, it simply means that we are not imposing very particular modular abstractions onto the learning process. We also want to acknowledge that this thesis shares its name with an excellent children’s science book by Blooma Goldberg, Ken Goldberg, Ashley Chase [56].

Part I

Supervision

To start off our investigation, we think about the role of supervision in robotic reinforcement learning. Reinforcement learning as described in Chapter 1 requires a well defined reward function to optimize. While this is trivially available in games [11], [21] in terms of the score and certain simulation domains [29], it is typically much more challenging to provide for robots in the real world. What makes it so challenging is the fact that reward functions don't simply exist in the real world and must either be provided by hand or somehow inferred from the robots own observations, both of which can be challenging in natural uninstrumented environments.

The typical pipeline as it stands now for most robotic RL applications is to first have a hand engineering state estimation or tracking system, use this to identify entities and objects in the world. These state estimates are then used in a hand defined reward function to provide reward for the optimization. The challenge with this pipeline is that every new task and environment requires considerable human effort to actually set up the reward provision mechanism, making it impractical for large scale robot learning.

An alternative paradigm is to move the design of reward functions from a hand designed one to a more data driven process. Given a small amount of human data, indicating the task to be solved, we can try and learn reward inference models that are able to extract reward r from sensory observations s, without actually requiring significant instrumentation or hand specification. This data driven paradigm can come in several different forms —in this thesis we consider inferring rewards from easy to provide and natural sources of supervision like learning from raw videos of human performing tasks, learning from examples of successful outcomes only, learning from language corrections and we even take this to its logical extreme, learning skills without any reward functions at all. Through this investigation of data driven reward inference, we show that the supervision problem can be tackled by utilizing human provided task specifications in a directed way to infer rewards.

Chapter 2

Supervision from Human Videos

As we described in Chapter 2, this work aims to make a move from programmatic rewards to a more data driven approach for reward design. Raw video supervision from human experts is a form of supervision that is relatively easy to provide, while still providing a significant amount of guidance on how to perform tasks. We start our investigation into supervision in reinforcement learning through a consideration of how we can actually infer rewards and learn from raw human videos with reinforcement learning.

2.1 Why Should We Learn from Raw Human Videos?

Learning can enable autonomous agents, such as robots, to acquire complex behavioral skills that are suitable for a variety of unstructured environments. In order for autonomous agents to learn such skills, they must be supplied with a supervision signal that indicates the goal of the desired behavior. This supervision typically comes from one of two sources: a reward function in reinforcement learning that specifies which states and actions are desirable, or expert demonstrations in imitation learning that provide examples of successful behaviors. Both modalities have been combined with high-capacity models such as deep neural networks to enable learning of complex skills with raw sensory observations [21], [31], [57], [58]. One major advantage of reinforcement learning is that the agent can acquire a skill through trial and error with only a high-level description of the goal provided through the reward function. However, reward functions can be difficult to specify by hand, particularly when the success of the task can only be determined from complex observations such as camera images [59].

Imitation learning bypasses this issue by using examples of successful behavior. Popular approaches to imitation learning include direct imitation learning via behavioral cloning [58] and reward function learning through inverse reinforcement learning [60]. Both settings typically assume that an agent receives examples that consist of sequences of observation-action tuples, and try to learn either a function that maps observations to actions from these example sequences or a reward function to explain this behavior while generalizing to new scenarios. However, this notion of imitation is quite different from the kind of imitation carried out by humans and animals: when

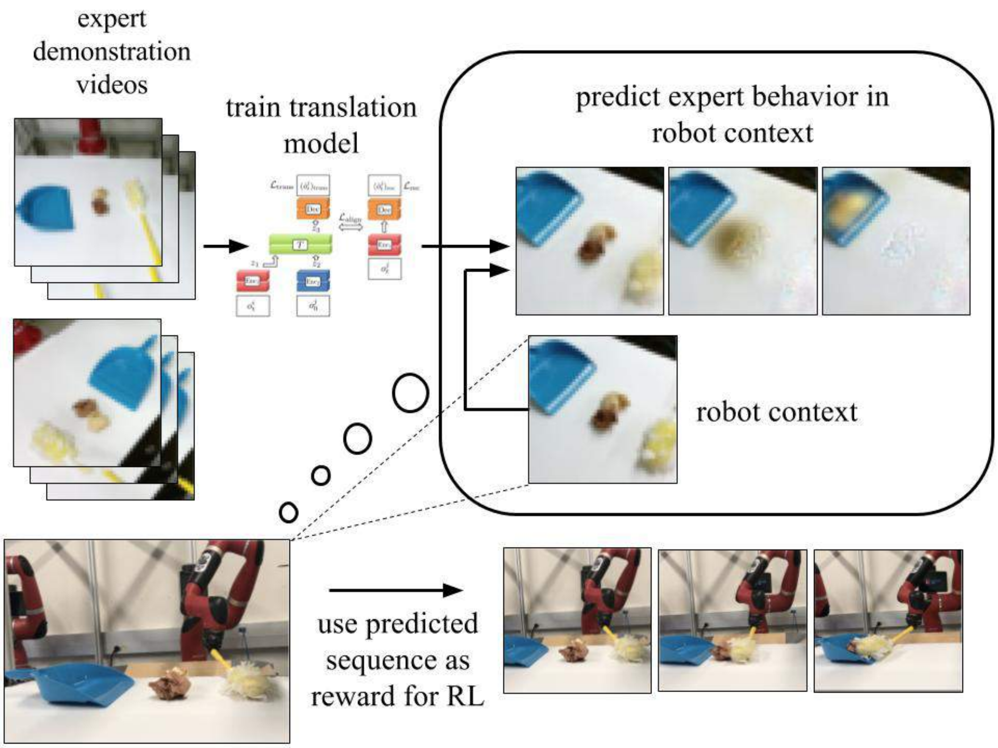


Figure 2.1: Imitation from Observation using Context-Aware Translation. We collect a number of videos of expert demonstrations from a human demonstrator, and use them to train a context translation model. At learning time, the robot sees the context of the task it needs to perform. Then, the model predicts what an expert would do in the robot context. This predicted sequence is used to define a cost function for reinforcement learning thus enabling imitation from observation. The task shown here is illustrative of a wide range of tasks that we evaluate.

we learn new skills from observing other people, we do not receive egocentric observations and ground truth actions. The observations are obtained from an alternate viewpoint and the actions are not known. Furthermore, humans are not only capable of learning from live observations of demonstrated behavior, but also from video recordings of behavior provided in settings considerably different than their own. Can we design imitation learning methods that can succeed in such situations? A solution to this problem would be of considerable practical value in robotics, since the resulting imitation learning algorithm could directly make use of natural videos of people performing the desired behaviors obtained, for instance, from the Internet.

We term this problem imitation-from-observation. The goal in imitation-from-observation is to learn policies only from a sequence of observations (which can be extremely high dimensional such as camera images) of the desired behavior, with each sequence obtained under differences in context. Differences in context might include changes in the environment, changes in the objects being manipulated, and changes in viewpoint, while observations might consist of sequences of images. We define this problem formally in Section 2.3.

Our imitation-from-observation algorithm is based on learning a context translation model that can convert a demonstration from one context (e.g., a third person viewpoint and a human demonstrator) to another context (e.g., a first person viewpoint and a robot). By training a model to perform this conversion, we acquire a feature representation that is suitable for tracking demonstrated behavior. We then use deep reinforcement learning to optimize for the actions that optimally track the translated demonstration in the target context. As we illustrate in our experiments, this method is significantly more robust than prior approaches that learn invariant feature spaces [61], perform adversarial imitation learning [62], or directly track pre-trained visual features [63]. Our translation method is able to provide useful perceptual reward functions, and performs well on a number of simulated and real manipulation tasks, including tasks that require a robot to emulate human tool use. Videos can be found on `https://sites.google.com/site/imitationfromobservation/`

2.2 Relationship to Prior Work

Imitation learning is usually thought of as the problem of learning an expert policy that generalizes to unseen states, given a number of expert state-action demonstration trajectories [64], [65]. Imitation learning has enabled the successful performance of tasks in a number of complex domains such as helicopter flight through apprenticeship learning [66], learning how to put a ball in a cup and playing table tennis [67], performing human-like reaching motions [68] among others. These methods have been very effective however typically require demonstrations provided through teleoperation or kinesthetic teaching, unlike our work which aims to learn from observed videos of other agents performing the task. Looking at the imitation learning literature from a more methodological standpoint, imitation learning algorithms can largely be divided into two classes of approaches: behavioral cloning and inverse reinforcement learning.

Behavioral cloning casts the problem of imitation learning as supervised learning, where the policy is learned from state(or observation)-action tuples provided by the expert. In imitation-from-observation, the expert does not provide actions, and only provides observations of the state in a different context, so direct behavioral cloning cannot be used. Inverse reinforcement learning (IRL) methods instead learn a reward function from the expert demonstrations [60], [69]–[71]. This reward function can then be used to recover a policy by running standard reinforcement learning [72], [73], though some more recent IRL methods alternate between steps of forward and inverse RL [62], [74]–[76]. While IRL methods can in principle learn from observations, in practice using them directly on high-dimensional observations such as images has proven difficult.

Aside from handling high-dimensional observations such as raw images, our method is also designed to handle differences in context. Context refers to changes in the observation function between different demonstrations and between the demonstrations and the learner. These may include changes in viewpoint, object positions, surroundings, etc. Along similar lines, [61] directly address learning under domain shift. However, this method has a number of restrictive requirements, including access to expert and non-expert policies, directly optimizing for invariance between only two contexts (whereas in practice demonstrations may come from several different contexts), and performs poorly on the more complex manipulation tasks that we consider, as illustrated in Section 13.7. [63] proposes to address differences in context by using pretrained visual features, but does not provide for any mechanism for context translation, as we do in our work, relying instead on the inherent invariance of visual features for learning. Follow-up work proposes to further increase the invariance of the visual features through multi-viewpoint training [77]. [78] propose to learn robotic skills from first person videos of humans by using explicit hand detection and a carefully engineered vision pipeline. In contrast, our approach is trained end-to-end, and does not require any prior visual features, detectors, or vision systems. [79] proposed to use demonstrations as input to policies by training on paired examples of state sequences, however our method operates on raw observations and does not require any actions in the demonstrations, while this prior method operates only on low-dimensional state variables and does not deal with context shift like our method.

Our technical approach is related to work in visual domain adaptation and image translation. Several works have proposed pixel level domain adaptation [80]–[82], as well as translation of visual style between domains [83], by using generative adversarial networks (GANs). The applications of these methods have been in computer vision, rather than robotic control. Our focus is instead on translating demonstrations from one context to another, conditioned on the first observation in the target context, so as to enable an agent to physically perform the task. Although we do not use GANs, these prior methods are complementary to ours, and incorporating a GAN loss could improve the performance of our method further.

In our work we consider tasks like sweeping, pushing, ladling(similar to pouring) and striking. Several prior methods have looked at performing tasks like these although typically with significantly different methods. Tasks involving cleaning with a brush, similar to our sweeping tasks was studied in [84] but is done using a low cost tool attachment and kinesthetic programming by demonstration. Besides [63], tasks involving pouring were also studied in [85] using a simple PID controller with a specified objective volume rather than inferring the objective from demonstrations. Similar flavors of tasks were also considered in [86], [87], but we leave those specific tasks to future work. Other work [88] also considers tasks of pushing objects on a table-top but uses predictive models on point-cloud data and uses a significantly different intuitive physics model with depth data.

2.3 Problem Formulation and Overview

In the imitation-from-observation setting that we consider in this work, an agent observes demonstrations of a task in a variety of *contexts*, and must then execute the demonstrated behavior in its own context. We use the term context to refer to properties of the environment and agent that can vary across demonstrations, which may include the viewpoint, the background, the positions and identities of objects in the environment, and so forth. The demonstrations $\{D_1, D_2, ...D_n\} = \{[o_0^1, o_1^1, ...o_T^1], [o_0^2, o_1^2, ...o_T^2], ..., [o_0^n, o_1^n, ...o_T^n]\}$ consist of observations o_t that are produced by a partially observed Markov process governed by an observation distribution $p(o_t|s_t, \omega)$, dynamics $p(s_{t+1}|s_t, a_t, \omega)$, and the expert's policy $p(a_t|s_t, \omega)$, with each demonstration being produced in a different context ω. Here, s_t represents the unknown Markovian state, a_t represents the action (which is not observed in the demonstrations), and ω represents the context. We assume that ω is sampled independently from $p(\omega)$ for each demonstration, and that the imitation learner has some fixed ω_l from the same distribution. Throughout the technical section, we use o_t^i to refer to the observation at time t from a context ω_i.

While a practical real-world imitation-from-observation application might also have to contend with systematic *domain shift* where, e.g., the learner's embodiment differs systematically from that of the demonstrator, and therefore the learner's context ω cannot be treated as a sample from $p(\omega)$, we leave this challenge to prior work, and instead focus on the basic problem of imitation-from-observation. This means that the context can vary between the demonstrations and the learner, but the learner's context still comes from the same distribution. We elaborate on the practical implications of this assumption in Section 13.7, and discuss how it might be lifted in future work.

Any algorithm for imitation-from-observation must contend with two challenges: first, it must be able to determine what information from the observations to track in its own context ω_l, which may differ from those of the demonstrations, and second, it must be able to determine which actions will allow it to track the demonstrated observations. Reinforcement learning (RL) offers a tool for addressing the latter problem: we can use some measure of distance to the demonstration as a reward function, and learn a policy that takes actions to minimize this distance. But which distance to use? If the observations correspond, for example, to raw image pixels, a Euclidean distance measure may not give a well-shaped objective: roughly matching pixel intensities does not necessarily correspond to a semantically meaningful execution of the task, unless the match is almost perfect. Fortunately, the solution to the first problem – context mismatch – naturally lends us a solution to the problem of choosing a distance metric. In order to address context mismatch, we can train a model that explicitly translates demonstrations from one context into another, by using the different demonstrations as training data. The internal representation learned by such a model provides a much more well-structured space for evaluating distances between observations, since proper context translation requires understanding the underlying factors of variation in the scene. As we empirically illustrate in our experiments, we can use squared Euclidean distances between features of the context translation model as a reward function to learn the demonstrated task, while using the model itself to translate these features from the demonstration context to the learner's context. We first describe the translation model, and then show how it can be used to create a reward function for RL.

2.4 Learning to Translate Between Contexts

Since each demonstration D_k is generated from an unknown context ω_k, the learner cannot directly track these demonstrations in its own context ω_l. However, since we have demonstrations from multiple unknown but different contexts, we can learn a context translation model on these demonstrations without any explicit knowledge of the context variables themselves. We only assume that the first frame o_0^k of a demonstration in a particular context ω_k can be used to implicitly extract information about the context ω_k.

Our translation model is trained on pairs of demonstrations $D_i = [o_0^i, o_1^i..., o_T^i]$ and $D_j = [o_0^j, o_1^j..., o_T^j]$, where D_i comes from a context ω_i (the source context) and D_j comes from a context ω_j (the target context). The model must learn to output the observations in D_j conditioned on D_i and the first observation o_0^j in the target context ω_j. Thus, the model looks at a single observation from a target context, and predicts what future observations in that context will look like by translating a demonstration from a source context. Once trained, this model can be provided with any demonstration D_k to translate it into the learner's context ω_l for tracking, as discussed in the next section.

The model (Fig 5.1), assumes that the demonstrations D_i and D_j are aligned in time, though this assumption could be relaxed in future work by using iterative time alignment [89]. The goal is to learn the overall translation function $M(o_t^i, o_0^j)$ such that its output $M(o_t^i, o_0^j) = (\hat{o}_t^j)_{\text{trans}}$ closely matches o_t^j for all t and each pair of training demonstrations D_i and D_j. That is, the model translates observations from D_i into the context ω_j, conditioned on the first observation o_0^j in D_j.

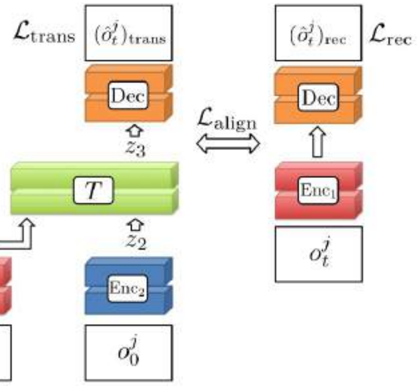

$\mathcal{L}_{\text{trans}}$
$(\hat{o}_t^j)_{\text{trans}}$
Dec
$\mathcal{L}_{\text{align}}$
$(\hat{o}_t^j)_{\text{rec}}$
$\mathcal{L}_{\text{rec}}$
Dec
z_3
T
Enc$_1$
z_1
z_2
o_t^j
Enc$_1$
Enc$_2$
o_t^i
o_0^j

Figure 2.2: Context translation model: The source observation o_t^i is translated to give the prediction of the observation in the target context $(\hat{o}_t^j)_{\text{trans}}$, given the context image o_0^j from the target context. The convolutional encoders are Enc$_1$ and Enc$_2$, while the deconvolutional decoder Dec decodes features back into observations. Colors indicate tied weights.

The model consists of four components: a source observation encoder $\text{Enc}_1(o_t^i)$ and a target initial observation encoder $\text{Enc}_2(o_0^j)$ that encode the observations into source and target features, referred to as z_1 and z_2, a translator $z_3 = T(z_1, z_2)$ that translates the features z_1 into features for the context of z_2, which are denoted z_3, and finally a target context decoder $\text{Dec}(z_3)$, which decodes these features into $\hat{o}_t^j$. We will use $F(o_t^i, o_0^j) = z_3$ to denote the feature extractor that generates the features z_3 from an input observation and a context image. The encoders Enc$_1$ and Enc$_2$ can have either different weights or tied weights depending on the diversity of the demonstration scenes. To deal with the complexities of pixel-level reconstruction, we include skip connections from Enc$_2$ to Dec. The model is supervised with a squared error loss $\mathcal{L}_{\text{trans}} = \|(\hat{o}_t^j)_{\text{trans}} - o_t^j\|_2^2$ on the output o_t^j and trained end-to-end.

However, we need the features z_3 to carry useful information, in order to provide an informative distance metric between demonstrations for feature tracking. To ensure that the translated features z_3 form a representation that is internally consistent with the encoded image features z_1, we jointly train the translation model encoder Enc$_1$ and decoder Dec as an autoencoder, with a reconstruction loss $\mathcal{L}_{\text{rec}} = \|\text{Dec}(\text{Enc}_1(o_t^j)) - o_t^j\|_2^2$. We simultaneously regularize the feature representation of this autoencoder to align it with the features z_3, using the loss $\mathcal{L}_{\text{align}} = \|z_3 - \text{Enc}_1(o_t^j)\|_2^2$. This forces the encoder Enc$_1$ and decoder Dec to adopt a consistent feature representation, so that the observation from the target context o_t^j is encoded into features that are similar to the translated features z_3. The training objective for the entire model is then given by the combined loss function $\mathcal{L} = \sum_{(i,j)}(\mathcal{L}_{\text{trans}} + \lambda_1 \mathcal{L}_{\text{rec}} + \lambda_2 \mathcal{L}_{\text{align}})$, with D_i and D_j being a pair of expert demonstrations chosen randomly from the training set, and λ_1 and λ_2 being hyperparameters. If we don't regularize the encoded features of learning trajectories and translated features of experts to lie in the same feature space, the reward function described in Section 2.5 is not effective since we are tracking features which have no reason to be in the same space. Examples of translated demonstrations are shown in Section 13.7 and the project website.

2.5 Learning Policies via Context Translation

The model described in the previous section can translate observations and features from the demonstration context into the learner's context ω_l. However, in order for the learning agent to actually perform the demonstrated behavior, it must be able to acquire the actions that track the translated features. We can choose between a number of deep reinforcement learning algorithms to learn to output actions that track the translated demonstrations given the reward function we describe below.

Reward Functions for Feature Tracking

The first component of the feature tracking reward function is a penalty for deviations from the translated features. At each time step, the translation function F (which gives us z_3) can be used to translate each of the demonstration observations o_t^i into the learner's context ω_l. The reward function then corresponds to minimizing the squared Euclidean distance between the encoding of the current observation to all of these translated demonstration features, which is approximately tracking their average, resulting in

$$\hat{R}_{\text{feat}}(o_t^l) = -\|\text{Enc}_1(o_t^l) - \frac{1}{n}\sum_i^n F(o_t^i, o_0^l)\|_2^2,$$

where $\text{Enc}_1(o_t^l)$ computes the features of the learner's observation at time step t, given by o_t^l, and $F(o_t^i, o_0^j)$ computes translated features of experts.

Unfortunately, feature tracking by itself may be insufficient to successfully imitate complex behaviors. The reason for this is that the distribution of observations fed into Enc_1 during policy learning may not match the distribution from the demonstrations that are seen during training: although a successful policy will closely track the translated policy, a poor initial policy might produce observations that are very different. In these cases, the encoder Enc_1 must contend with out-of-distribution samples, which may not be encoded properly. In fact, the model will be biased toward predicting features that are closer to the expert, since it only saw expert data during training. To address this, we also introduce a weak image tracking reward. This reward directly penalizes the policy for experiencing observations that differ from the translated observations, using the full observation translation model M:

$$\hat{R}_{\text{img}}(o_t^l) = -\|o_t^l - \frac{1}{n}\sum_i^n M(o_t^i, o_0^l)\|_2^2$$

The final reward is then the weighted combination $\hat{R}(o_t^l) = \hat{R}_{\text{feat}}(o_t^l) + w_{\text{rec}}\hat{R}_{\text{img}}(o_t^l)$, where w_{rec} is a small constant (tuned as a hyperparameter in our implementation).

Reinforcement Learning Algorithms for Feature Tracking

With the reward described in Section 2.5, we perform reinforcement learning in order to learn control policies in our learning environment. Our method can be used with any reinforcement learning algorithm. We use trust region policy optimization (TRPO) [90] for our simulated experiments but not for real world experiments because of it's high sample complexity. For the real-world robotic experiments, we use the trajectory-centric RL method used for local policy optimization in guided policy search (GPS) [31], which is based on fitting locally linear dynamics and performing LQR-based updates. We compute image features z_3, and include these as part of the state. The cost function for GPS is then a squared Euclidean distance in state space, and we omit the image tracking cost described in Section 2.5. For simulated striking and real robot pushing, this cost function is also weighted by a quadratic ramp function weighting squared Euclidean distances at later time steps higher than initial ones.

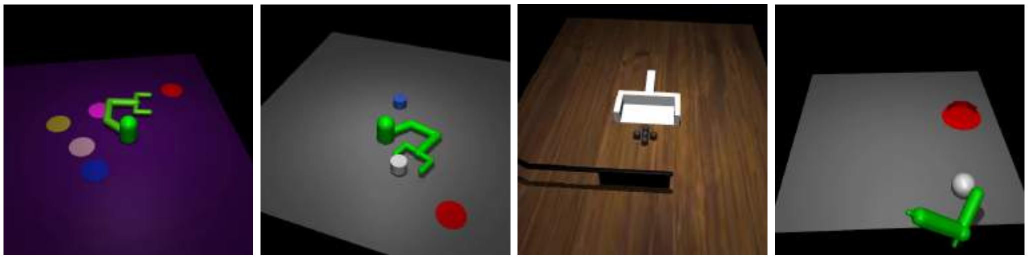


Figure 2.3: Four simulated tasks, from left to right: reaching (goal is to reach the red circle), pushing (goal is to push the white can to the red goal), sweeping (goal is to sweep grey balls into the pan), and striking (goal is to strike the white ball to the red goal).

2.6 Experiments

Our experiments aim to evaluate whether our context translation model can enable imitation-from-observation, and how well representative prior methods perform on this type of imitation learning task. The specific questions that we aim to answer are: (1) Can our context translation model handle raw image observations, changes in viewpoint, and changes in the appearance and positions of objects between contexts? (2) How well do prior imitation learning methods perform in the presence of such variation, in comparison to our approach? (3) How well does our method perform on real-world images, and can it enable a real-world robotic system to learn manipulation skills? All results, including illustrative videos, video translations and further experiment details can be found on: `https://sites.google.com/site/imitationfromobservation/`

In order to provide detailed comparisons with alternative prior methods for imitation learning, we set up four simulated manipulation tasks using the MuJoCo simulator [91]. To provide expert demonstrations, we hand-specified a reward function for each task and used a prior policy optimization algorithm [90] to train an expert policy. We collected video demonstrations of rollouts from the final expert policy acting in a number of randomly generated contexts.

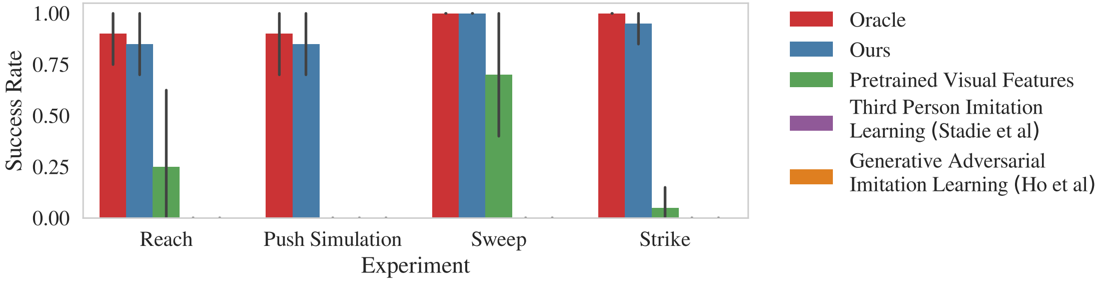


Figure 2.4: Comparisons with prior methods on the reaching, pushing, sweeping, and striking tasks. The results show that our method successfully learned each task, while the prior methods struggled to perform the reaching, pushing and striking tasks, and only the pretrained visual features approach was able to make a reasonable improvement on the sweeping task. Third person imitation learning [61] and generative adversarial imitation [62] learning are both at 0% success rate on the graph.

The tasks are illustrated in Fig. 18.11. The first task requires a robotic arm to reach varying goal positions indicated by a red disk, in the presence of variation in color and appearance. The second task requires pushing a white cylinder onto a red coaster, both with varying position, in the presence of varied distractor objects. The third task requires the simulated robot to sweep five grey balls into a dustpan, under variation in viewpoint. The fourth task involves using a 4 DoF arm to strike a white ball toward a red target which varies in position. The project website illustrates the variability in appearance and object positioning in the tasks, and also presents example translations of individual demonstration sequences.

Network Architecture and Training

For encoders Enc_1 and Enc_2 in simulation we perform four 5×5 stride-2 convolutions with filter sizes 64, 128, 256, and 512 followed by two fully-connected layers of size 1024. We use LeakyReLU activations with leak 0.2 for all layers. The translation module $T(z_1, z_2)$ consists of one hidden layer of size 1024 with input as the concatenation of z_1 and z_2. For the decoder Dec in simulation we use a fully connected layer from the input to four fractionally-strided convolutions with filter sizes 256, 128, 64, 3 and stride $\frac{1}{2}$. We have skip connections from every layer in the context encoder Enc_2 to its corresponding layer in the decoder Dec by concatenation along the filter dimension. For real world images, the encoders perform 4 convolutions with filter sizes 32, 16, 16, 8 and strides 1, 2, 1, 2 respectively. All fully connected layers and feature layers are size 100 instead of 1024. The decoder uses fractionally-strided convolutions with filter sizes 16, 16, 32, 3 with strides $\frac{1}{2}$, 1, $\frac{1}{2}$, 1 respectively. For the real world model only, we apply dropout for every fully connected layer with probability 0.5, and we tie the weights of Enc_1 and Enc_2.

We train using the ADAM optimizer with learning rate 10^{-4}. We train using 3000 videos for reach, 4500 videos for simulated push, 894 videos for sweep, 3500 videos for strike, 135 videos for real push, 85 videos for real sweep with paper, 100 videos for real sweep with almonds, and 60 videos for ladling almonds. We downsample videos to 36×64 pixels for simulated sweeping and 48×48 for all other videos.

Comparative Evaluation of Context Translation

Results for the comparative evaluation of our approach are presented in Fig 2.4. Performance is evaluated in terms of the final distance of the target object to the goal during testing. In the reaching task, this is the distance of the robot's hand from the goal, in the pushing task, this is the distance of the cylinder from the goal, in the sweeping task, this corresponds to the mean distance of the balls from the inside of the dustpan, and in the striking task this is the final distance of the ball from the goal position. All distances are normalized by dividing by the initial distance at the start of the task, and success is measured as a thresholding of the normalized distance. We evaluate each task on 10 randomly generated environment conditions, each time performing 100 iterations of reinforcement learning with 12,500 samples per iteration.

Our comparisons include our method with TRPO for policy learning, an oracle that trains a policy with TRPO on the ground truth reward function in simulation, which represents an upper bound on performance, and three prior imitation learning methods. The first prior method learns a reward using pre-trained visual features, similar to the work of [63]. In this method, features from an Inception-v3 network trained on ImageNet classification [92] are used to encode the goal image from the demonstration, and the reward function corresponds to the distance to these features averaged over all the training demonstrations. We experimented with several different feature layers from the Inception-v3 network and chose the one that performed best. The second prior method, third person imitation learning (TPIL), is an IRL algorithm that explicitly compensates for domain shift using an adversarial loss [61], and the third is an adversarial IRL-like algorithm called generative adversarial imitation learning (GAIL) [62], using a convolutional model to process images as suggested by [93]. Among these, only TPIL explicitly addresses changes in domain or context.

The results, shown in Fig 2.4, indicate that our method was able to successfully learn each of the tasks when the demonstrations were provided from random contexts. Notably, none of the prior methods were actually successful on the reaching, pushing or striking tasks, and struggled to perform the sweeping task. This indicates that imitation-from-observation in the presence of context differences is an exceedingly challenging problem.

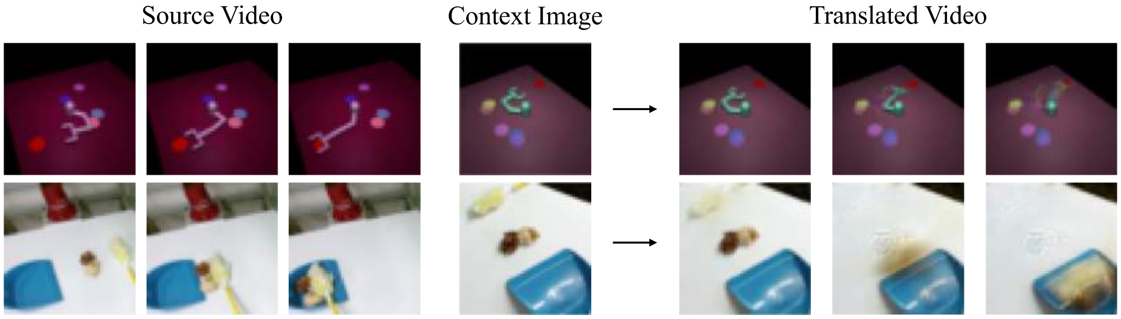


Figure 2.5: Translations from a video in the holdout set to a new context for the reaching task (top) and paper sweeping task (bottom).

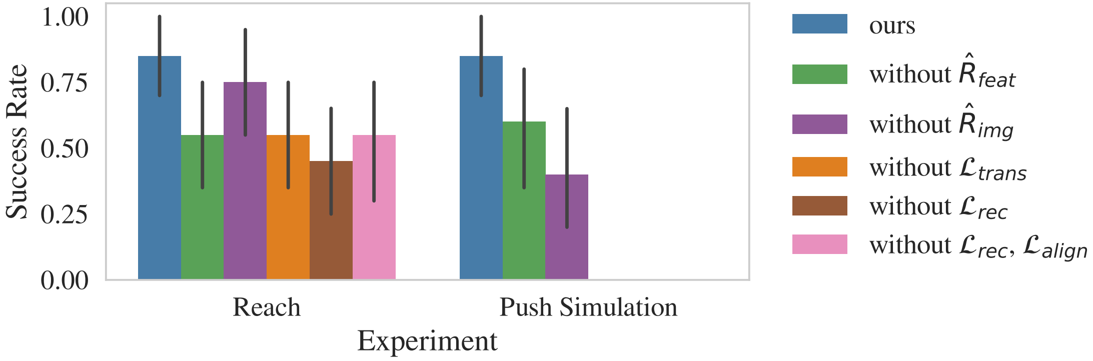


Figure 2.6: Ablations on model losses and reward functions for the simulated reaching and pushing tasks.Our method with all components does consistently the best across tasks. Note: for the Push Simulation task, we did not perform ablations "without $\mathcal{L}_{trans}$", "without $\mathcal{L}_{rec}$", and "without $\mathcal{L}_{rec}, \mathcal{L}_{align}$"

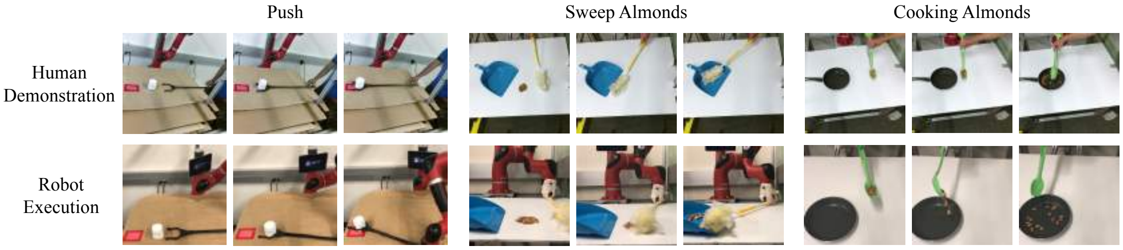


Figure 2.7: Top Row: Demonstrations by a human demonstrator showing the robot how to perform the pushing, sweeping and ladling almonds task in the real world. Bottom Row: Execution of the robot successfully performing the pushing, sweeping and ladling almonds tasks.

Ablation Study

To evaluate the importance of different loss functions while training our translation model, and the different components for the reward function while performing imitation, we performed ablations by removing these components one by one during model training or policy learning. To understand the importance of the translation cost, we remove cost $\mathcal{L}_{\text{trans}}$, to understand whether features z_3 need to be properly aligned we remove model losses $\mathcal{L}_{\text{rec}}$ and $\mathcal{L}_{\text{align}}$. In Fig 2.6 we see that the removal of each of these losses significantly hurts the performance of subsequent imitation. On removing the feature tracking loss $\hat{R}_{\text{feat}}$ or the image tracking loss $\hat{R}_{\text{image}}$ we see that overall performance across tasks is worse.

Natural Images and Real-World Robotic Manipulation

To evaluate whether our method is able to scale to real-world images and robots, we focus on manipulation tasks involving tool use, where object positions and camera viewpoints differ between contexts. All demonstrations were provided by a human, while the learned skills were performed by a robot. Since our method assumes that the contexts of the demonstrations and the learner are sampled from the same distribution, the human and the robot both used the same tool to perform the tasks, avoiding any systematic domain shift that might result from differences in the appearance of the human or robot arm. To this end, we apply a cropping of each video around task-relevant areas of each demonstration. Investigating domain shift is left for future work, and could be done, for example, using domain adaptation [94]. In the present experiments, we assume that the demonstration and test contexts come from the same distribution, which is a reasonable assumption in settings such as tool use and navigation, or tasks where the focus is on the objects in the scene rather than the arm or end-effector.

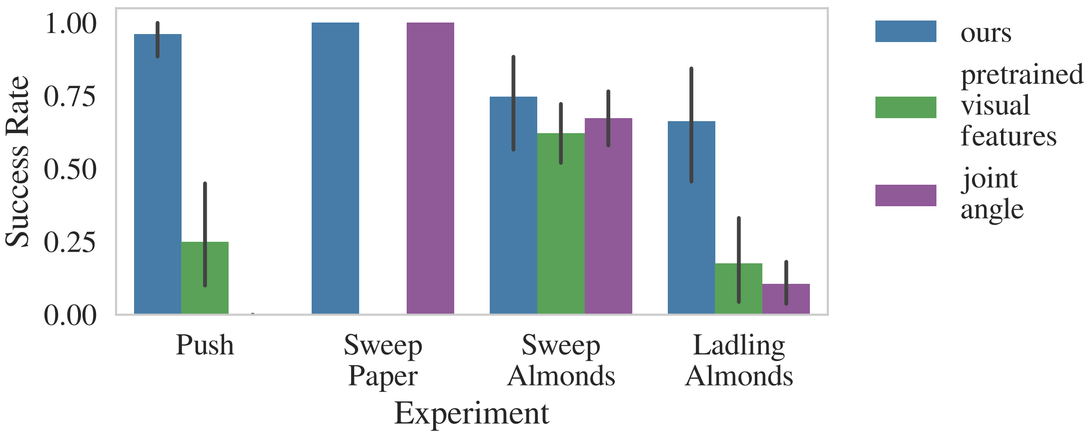


Figure 2.8: Plot depicting success rate for our method versus other baselines on the real world tasks with the Sawyer robot. Success metrics differ per task as described in Section 2.6. As is seen clearly, our method consistently performs well on all the real world tasks, and outperforms the baseline methods.

Pushing

In the first task, the goal is to push a cylinder to a marked goal position. The success metric is defined as whether the final distance between the cylinder and goal is within a predefined threshold. We evaluate our method in the setting where real-world demonstrations are provided by a human and imitation is done by a robot in the real world.

We evaluated how our method can be used to learn a pushing behavior with a real-world robotic manipulator, using a 7-DoF Sawyer robot. Since the TRPO algorithm is too data intensive to learn on real-world physical systems, we use GPS for policy learning (Section 2.5).

For comparison, we also test GPS with a reward that involves tracking pre-trained visual features from the Inception-v3 network (Section 2.6), as well as a baseline reward function that attempts to reach a fixed set of joint angles, specified through kinesthetic demonstration. Note that our method itself does not use any kinesthetic demonstrations, only video demonstrations provided by the human. In order to include the high-dimensional visual features in the state for guided policy search, we apply PCA to reduce their dimensionality from 221952 to 100, while our method uses all 100 dimensions of z_3 as part of the state. We found that our method could successfully learn the skill using demonstrations from different viewpoints, and outperforms the pre-trained features and kinesthetic baseline, as shown in Fig 2.8.

Sweeping

The pushing task illustrates the basic capability of our method to learn skills involving manipulation of rigid objects. However, one major advantage of learning visual reward functions from demonstration is the ability to acquire representations that can be used to manipulate scenes that are harder to represent analytically, such as granular media. In this next experiment, we study how well our method can learn two variants of a sweeping task: in the first, the robot must sweep crumpled paper into a dustpan, and in the second it must sweep a pile of almonds. We used almonds in place of dirt or fluids to avoid damaging the robot. Quantitative results are summarized in Fig 2.8.

On the easier crumpled paper task, both our method and the kinesthetic teaching approach works well, but the reward that uses pre-trained visual features is insufficient to accomplish the task. On the almond sweeping task (Fig 2.7), our method achieves a higher success rate than the alternative approaches. The success metric is defined as the average percentage of almonds or paper pieces that end up inside the dustpan.

Ladling Almonds

Our last task combines granular media (almonds) and a more dynamic behavior. In this task, the robot must ladle almonds into a cooking pan (Fig 2.7). This requires keeping the ladle upright until over the pan, and then dumping them into the pan by turning the wrist. The success metric is the average fraction of almonds that were ladled into the pan. Learning from only raw videos of the task being performed by a human in different contexts, our method achieved a success rate of 66% , while the alternative approaches generally could not perform this task. An insight into why the joint angles approach wouldn't work on this task is that the spoon has to remain upright until just the right position over the pan after which it should rotate and pour into the pan. The joint angle baseline can simply interpolate between the final turned spoon position and the initial position and pour the almonds in the wrong location. Quantitative results and comparisons are summarized in Fig 2.8.

2.7 Discussion and Future Work

We investigated how imitation-from-observation can be performed by learning to translate demonstration observation sequences between different contexts, such as differences in viewpoint. After translating observations into a target context, we can track these observations with RL, allowing the learner to reproduce the observed behavior. The translation model is trained by translating between the different contexts observed in the training set, and generalizes to the unseen context of the learner. Our experiments show that our method can be used to perform a variety of manipulation skills, and can be used for real-world robotic control on a diverse range of tasks patterned after common household chores.

Although our method performs well on real-world tasks and several tasks in simulation, it has a number of limitations. First, it requires a substantial number of demonstrations to learn the translation model. Training an end-to-end model from scratch for each task may be inefficient in practice, and combining our method with higher level representations proposed in prior work would likely lead to more efficient training [63], [77]. Second, we require observations of demonstrations from multiple contexts in order to learn to translate between them. In practice, the number of available contexts may be scarce. Future work would explore how multiple tasks can be combined into a single model, where different tasks might come from different contexts. Finally, it would be exciting to explore explicit handling of domain shift in future work, so as to handle large differences in embodiment and learn skills directly from videos of human demonstrators obtained, for example, from the Internet.

We will place this work into the broader context and discuss connections to other work post publication in Chapter 8.

Chapter 3

Supervision from Outcome Examples

In the previous chapter, we considered how to learn from raw videos of humans demonstrating particular tasks. While this technique can enable solving of several robotic tasks, it can still be expensive to provide *entire* demonstrations, as opposed to just examples of what a successful outcome would look like. In this chapter, we explore this paradigm, investigating whether we can supervised RL algorithms, simply by utilizing examples of successful outcomes.

3.1 Why Should we Study Uncertainty-Aware Outcome Driven RL?

While reinforcement learning (RL) has been shown to successfully solve problems with careful reward design [95], RL in its most general form, with no assumptions on the dynamics or reward function, requires solving a challenging uninformed search problem in which rewards are sparsely observed. Techniques that explicitly provide "reward-shaping" [96], or modify the reward function to guide learning, can help take some of the burden off of exploration, but shaped rewards are often difficult to provide without significant domain knowledge. Moreover, in many domains of practical significance, actually specifying rewards in terms of high dimensional observations can be extremely difficult, making it infeasible to directly apply RL to problems with challenging exploration.

Can the RL problem be made more tractable if the agent is provided with examples of successful outcomes instead of an uninformative reward function? Such examples are often easier to provide than, for example, entire demonstrations or a hand-designed reward function. However, they can still provide considerable guidance on how to successfully accomplish a task, potentially alleviating exploration challenges if the agent can successfully recognize similarities between visited states and the provided examples. In this paper, we study such a problem setting, where instead of a hand-designed reward function, the RL algorithm is provided with a set of *successful outcome examples*: states in which the desired task has been accomplished successfully. Prior work [97], [98] aims to solve tasks in this setting by estimating the distribution over these states and maximizing the probability of reaching states that are likely under this distribution. While this can work well in some domains, it has largely been limited to settings without significant exploration challenges. In our

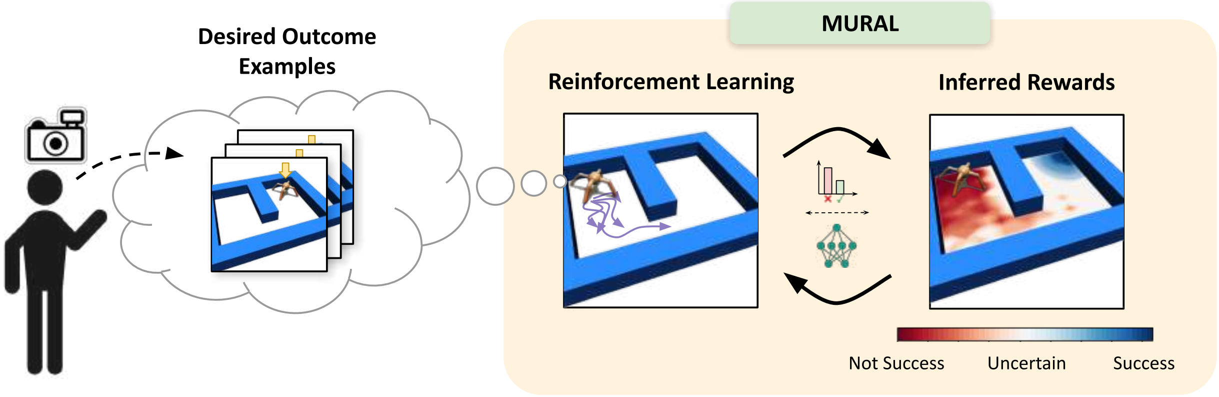


Figure 3.1: **MURAL:** Our method trains an uncertainty-aware classifier based on user-provided examples of successful outcomes. Appropriate uncertainty in the classifier, obtained via a meta-learning based estimator for the normalized maximum likelihood (NML) distribution, automatically incentivizes exploration and provides reward shaping for RL.

work, we focus on the potential for this mode of task specification to enable RL algorithms to solve more challenging tasks without the need for manual reward shaping. Intuitively, the availability of extra information in the form of explicit success examples can provide the algorithm more directed information for exploration, rather than having to simply rely on uninformed task agnostic exploration methods. This allows us to formulate a class of more tractable problems, which we refer to as outcome-driven RL.

However, in order to attain improved exploration, an outcome-driven RL agent must be able to estimate some notion of similarity between the visited states and successful outcomes, so as to utilize this similarity as a kind of automatic reward shaping. Our method addresses this challenge by training a classifier to distinguish successful states, provided by the user, from those generated by the current policy, analogously to generative adversarial networks [99] and previously proposed methods for inverse reinforcement learning [100]. In general, such a classifier may not provide effective reward shaping for learning the policy, since it does not explicitly quantify uncertainty about success probabilities and can be overly pessimistic in providing reward signal for learning. We discuss how Bayesian classifiers incorporating a particular form of uncertainty quantification based on the normalized maximum likelihood (NML) distribution can incentivize exploration in outcome-driven RL problems. To understand its benefits, we connect our approach to count-based exploration methods, while also showing that it improves significantly over such methods when the classifier exhibits good generalization properties, due to its ability to utilize success examples. Finally, we propose a practical algorithm to train NML-based success classifiers in a computationally efficient way using meta-learning, and show experimentally that our method can more effectively solve a range of challenging navigation and robotic manipulation tasks.

Concretely, this work illustrates the challenges of using standard success classifiers [97] for outcome-driven RL, and proposes a novel technique for training uncertainty aware classifiers with normalized maximum likelihood, which is able to both incentivize the exploration of novel states and provide reward shaping that guides exploration towards successful outcomes. We present a tractable algorithm for learning these uncertainty aware classifiers in practice by leveraging concepts from meta-learning. We analyze our proposed technique for reward inference experimentally across a number of navigation and robotic manipulation domains and show benefits over prior classifier-based RL methods as well as goal-reaching methods.

3.2 Relationship to Prior Work

While a number of methods have been proposed to improve exploration, it remains a challenging open problem in RL [101]. Standard exploration methods either add bonuses to the reward function that encourage a policy to visit novel states in a task-agnostic manner [102]–[111], or approximate Thompson sampling from a posterior over value functions [112]–[114]. Whereas these techniques are uninformed about the actual task, we consider a constrained, yet still widely applicable, set of problems where the desired outcome can be specified by success examples, allowing for more efficient task-directed exploration.

Designing well-shaped reward functions can also make exploration easier, but often requires significant domain knowledge [41], access to privileged information [31] or a human in the loop providing rewards [115], [116]. Prior work has considered specifying rewards by providing example demonstrations and inferring rewards with inverse RL [62], [66], [71], [100]. This requires expensive expert demonstrations to be provided to the agent. In contrast, our work has the minimal requirement of successful outcome states, which can be provided more cheaply and intuitively. This subclass of problems is also related to goal-conditioned RL [30], [39], [117]–[125] but is more general, since it allows for the notion of success to be more abstract than reaching a single state.

A core idea behind our method is using a Bayesian classifier to learn a suitable reward function. Bayesian inference with expressive models and high dimensional data can often be intractable, requiring strong assumptions on the form of the posterior [126]–[128]. In this work, we build on the concept of normalized maximum likelihood [129], [130], or NML, to learn Bayesian classifiers that can impose priors over the space of outcomes. Although NML is typically considered from the perspective of optimal coding [131], [132], we show how it can be used for success classifiers, and discuss connections to exploration in RL. We propose a novel technique for making NML computationally tractable based on meta-learning, which more directly optimizes for quick NML computation as compared to prior methods like [133] which learn an amortized posterior.

3.3 Preliminaries

In this section, we discuss background on RL using successful outcome examples as well as conditional normalized maximum likelihood.

Reinforcement Learning with Outcome Examples

We follow the framework proposed by [97] and assume that we are provided with a Markov decision process (MDP) *without* a reward function, given by $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \gamma, \mu_0)$, as well as successful outcome examples $\mathcal{S}_+ = \{s_+^k\}_{k=1}^K$, which is a set of states in which the desired task

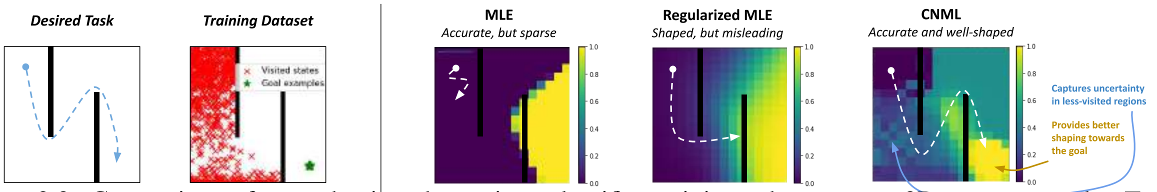


Figure 3.2: Comparison of rewards given by various classifier training schemes on a 2D maze example. Typical maximum likelihood (MLE) classifiers commonly suffer from either a lack of useful learning signal (if trained to convergence) or misleading local optima (if regularized using standard methods such as weight decay or early stopping), whereas CNML produces accurate and well-shaped rewards.

has been accomplished. This formalism is easiest to describe in terms of the control as inference framework [134]. The relevant graphical model (refer to [97]) consists of states and actions, as well as binary success variables $e_t \in \{0, 1\}$ that represent the occurrence of a particular event. The agent's objective is to cause this event to occur (e.g., a robot that is cleaning the floor must cause the "floor is clean" event to occur). Formally, we assume that the states in $\mathcal{S}_+$ are sampled from the distribution $p(s_t|e_t = 1)$ — that is, states where the desired event has taken place — and try to infer the distribution $p(e_t = 1|s_t)$ to use as a reward function. In this work, we focus on efficient methods for solving this reformulation of the RL problem by utilizing a novel uncertainty quantification method to represent $p(e_t|s_t)$.

In practice, prior methods that build on this formulation of the RL problem [97] derive an algorithm where the reward function in RL is produced by a classifier that estimates $p(e_t = 1|s_t)$. Following the derivation in [100], it is possible to show that the correct source of *negative* examples is the state distribution of the policy itself, $\pi(s)$. This insight results in a simple algorithm: at each iteration of the algorithm, the policy is updated to maximize the current reward, given by $\log p(e_t = 1|s_t)$, then samples from the policy are added to the set of negative examples $\mathcal{S}_-$, and the classifier is retrained on the original positive set $\mathcal{S}_+$ and the updated negative set $\mathcal{S}_-$. As noted in prior work [97], this process is closely connected to GANs and inverse reinforcement learning, where the classifier plays the role of the discriminator and the policy that of the generator. However, as we will discuss, this strategy can often face significant exploration challenges.

Conditional Normalized Maximum Likelihood

Our work utilizes the principle of conditional normalized maximum likelihood (CNML) [131], [132], [135], which we review briefly. CNML is a method for performing k-way classification, given a model class Θ and a dataset $\mathcal{D} = \{(x_0, y_0), (x_1, y_1), ..., (x_n, y_n)\}$, and has been shown to provide better calibrated predictions and uncertainty estimates with minimax regret guarantees [136]. More specifically, the CNML distribution can be shown to provably minimize worst-case regret against an oracle learner that has access to the true labels, but does not know which point it will be tested on. We refer the reader to [132], [133] for a more complete consideration of the theoretical properties of the CNML distribution.

To predict the class of a query point x_q, CNML constructs k augmented datasets by adding x_q with a different label in each dataset, which we write as $\mathcal{D} \cup (x_q, y = i), i \in (1, 2, ..., k)$. CNML then defines the class distribution by solving the maximum likelihood estimation problem at query time for each of these augmented datasets to convergence, and normalizes the likelihoods as follows:

$$p_{\text{CNML}}(y = i|x_q) = \frac{p_{\theta_i}(y = i|x_q)}{\sum_{j=1}^{k} p_{\theta_j}(y = j|x_q)} \tag{3.1}$$

$$\theta_i = arg \max_{\theta \in \Theta} \mathbb{E}_{(x,y)\sim\mathcal{D}\cup(x_q,y=i)}[\log p_\theta(y|x)] \tag{3.2}$$

If x_q is close to other datapoints in $\mathcal{D}$, the model will struggle to assign a high likelihood to labels that differ substantially from those of nearby points. However, if x_q is far from all datapoints in $\mathcal{D}$, then the different augmented maximum likelihood problems can easily classify x_q as any arbitrary class, providing us with likelihoods closer to uniform. We refer readers to [131] for an in-depth discussion of CNML and its connections to minimum description length and regret minimization. Intuitively, the CNML classifier provides a way to impose a uniform prior for uncertainty quantification, where we predict the uniform distribution on unseen inputs since they are maximally uncertain, and defer more to the maximum likelihood solution on frequently seen inputs since they are minimally uncertain.

3.4 Bayesian Success Classifiers for Reward Inference

As discussed in Section 3.3, a principled way of approaching outcome-driven RL is to train a classifier to determine whether a particular state is a successful outcome or not. However, while such a technique would eventually converge to the correct solution, it frequently suffers from uninformative or incorrect rewards during the learning process. For example, Figure 3.2 depicts a simple 2D maze scenario where the agent starts at the top left corner and the positive outcomes are at the bottom right corner of the environment. Without suitable regularization, the decision boundary may take on the form of a sharp boundary *anywhere* between the positive and negative examples in the early stages of training. As a result, the classifier might provide little to no reward signal for the policy, since it can assign arbitrarily small probabilities to the states sampled from the policy. Given that the classifier-based RL process is essentially equivalent to training a GAN (as described in Section 3.3), this issue is closely related to the challenges of GAN training as noted by [137], where an ideal maximum likelihood discriminator provides no gradient signal for training the generator.

We note that this issue is not pathological: our experiments in Section 13.7 show that this phenomenon of poor reward shaping happens in practice. In addition, introducing naïvely chosen forms of regularization such as weight decay, as is common in prior works, may actually provide *incorrect* reward shaping to the algorithm, making it more challenging to actually accomplish the task (as illustrated in Figure 3.2). This often limits classifier-based RL techniques to tasks with trivial exploration challenges. In this section, we will discuss how a simple change to the procedure for training a classifier, going from standard maximum likelihood estimation to an approach based on the principle of normalized maximum likelihood, allows for an appropriate consideration of uncertainty quantification that can solve problems with non-trivial exploration challenges.

Algorithm 1: RL with CNML-Based Success Classifiers

```
User provides success examples $\mathcal{S}_+$
Initialize policy $\pi$, replay buffer $\mathcal{S}_-$, and reward classifier parameters $\theta_{\mathcal{R}}$
for iteration $i = 1, 2, ...$ do
    Add on-policy examples to $\mathcal{S}_-$ by executing $\pi$.
    Sample $n_{\text{test}}$ points from $\mathcal{S}_+$ (label 1) and $n_{\text{test}}$ points from $\mathcal{S}_-$ (label 0) to construct a dataset $\mathcal{D}$
    Assign state rewards as $r(s) = p_{\text{CNML}}(e = 1|s, \mathcal{D})$
    Train $\pi$ with RL algorithm
end for
```

Regularized Success Classifiers via Normalized Maximum Likelihood

It is important to note that for effective exploration in reinforcement learning, the rewards should not just indicate whether a state is a successful outcome (since this will be 0 everywhere but successful outcomes), but should instead provide a sense of whether a particular state may be on the path to a successful outcome and should be explored further. The standard maximum likelihood classifier described in Section 15.2 is overly pessimistic in doing so, setting the likelihood of all intermediate states to 0 in the worst case, potentially mislabeling promising states to explore. To avoid this, we want to use a classification technique that minimizes this worst-case regret, maintaining some level of uncertainty about whether under-visited states are on the path to successful outcomes. As discussed in Section 3.3, the technique of conditional normalized maximum likelihood provides us a straightforward way to obtain such a classifier. CNML is particularly well suited to this problem since, as discussed in [138], it essentially imposes a uniform prior over the space of outcomes. It thus avoids pathological collapse of rewards by maintaining a measure of uncertainty over whether a state is potentially promising to explore further, rather than immediately bringing its likelihood to 0 as maximum likelihood solutions would.

To use CNML for reward inference, the procedure is similar to the one described in Section 15.2. We construct a dataset using the provided successful outcomes as positives and on-policy samples as negatives. However, the label probabilities for RL are instead produced by the CNML procedure to obtain rewards $r(s) = p_{\text{CNML}}(e = 1|s)$ as follows:

$$r(s) = \frac{p_{\theta_1}(e = 1|s)}{p_{\theta_1}(e = 1|s) + p_{\theta_0}(e = 0|s)} \tag{3.3}$$

$$\theta_0 = \arg\max_{\theta \in \Theta} \mathbb{E}_{(s_j, e_j) \sim \mathcal{D} \cup (s, e=0)} [\log p_\theta(e_j|s_j)] \tag{3.4}$$

$$\theta_1 = \arg\max_{\theta \in \Theta} \mathbb{E}_{(s_j, e_j) \sim \mathcal{D} \cup (s, e=1)} [\log p_\theta(e_j|s_j)] \tag{3.5}$$

This reward is then used to perform policy updates, new data is collected with the updated policy, and the process is repeated. A full description can be found in Algorithm 1.

To illustrate how this change affects reward assignment during learning, we visualize a potential assignment of rewards with a CNML-based classifier on the problem described earlier in Fig 3.2.

When the success classifier is trained with CNML instead of standard maximum likelihood, intermediate unseen states would receive non-zero rewards rather than simply having vanishing likelihoods like the maximum likelihood solution, thereby incentivizing exploration. In fact, the CNML likelihood has a strong connection to count-based exploration, as we show next. Additionally, we also see that CNML is able to provide more directed shaping towards the successful outcomes when generalization exists across states, as explained below.

Relationship to Count-Based Exploration

In this section we relate the success likelihoods obtained via CNML to commonly used exploration methods based on counts. Formally, we prove that the success classifier trained with CNML is equivalent to a version of count-based exploration in the absence of any generalization across states (i.e., a fully tabular setting).

Theorem 1. Suppose we are estimating success probabilities $p(e = 1|s)$ in the tabular setting, where we have an independent parameter for each state. Let $N(s)$ denote the number of times state s has been visited by the policy, and let $G(s)$ be the number of occurrences of state s in the set of positive examples. Then the CNML success probability $p_{\text{CNML}}(e = 1|s)$ is equal to $\frac{G(s)+1}{N(s)+G(s)+2}$. For states that are not represented in the positive examples, i.e. G(s) = 0, we then recover inverse counts $\frac{1}{N(s)+2}$.

Refer to Appendix A.1 for a full proof. While CNML has a strong connection with counts as described above, it is important to note two advantages. First, the rewards are estimated without an explicit generative model, simply by using a standard discriminative model trained via CNML. Second, in the presence of generalization via function approximation, the exploration behavior from CNML can be significantly more task directed, as described next.

In most problems, when the classifier is parameterized by a function approximator with non-trivial generalization, the structure of the state space actually provides more information to guide the agent towards the successful examples than simply using counts. In most environments [29], [139] states are not completely uncorrelated, but instead lie in a representation space where generalization correlates with the dynamics structure in the environment. For instance, states from which successful outcomes can be reached more easily (i.e., states that are “close” to successful outcomes) are likely to have similar representations. Since the uncertainty-aware classifier described in Section 3.4 is built on top of such features and is trained with knowledge of the desired successful outcomes, it is able to incentivize more *task-aware* directed exploration than simply using counts. This phenomenon is illustrated intuitively in Fig 3.2, and demonstrated empirically in our experimental analysis in Section 13.7.

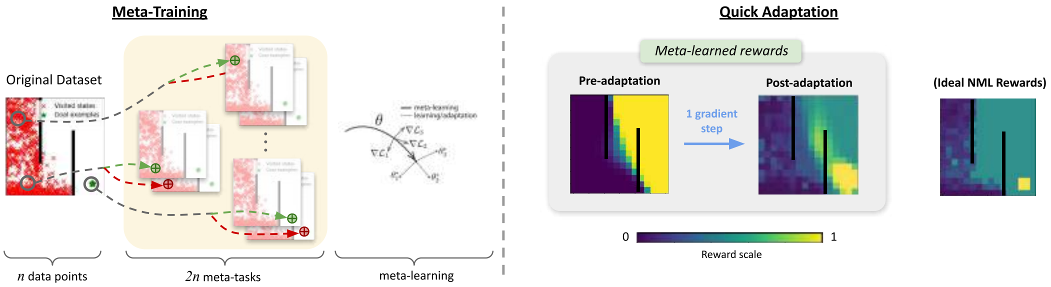


Figure 3.3: Diagram of using meta-NML to train a classifier. Meta-NML learns an initialization that can quickly adapt to new datapoints with arbitrary labels. At evaluation time, it approximates the NML probabilities (right) fairly well with a single gradient step.

3.5 MURAL: Training Uncertainty-Aware Success Classifiers for Outcome Driven RL via Meta-Learning and CNML

In Section 15.3, we discussed how success classifiers trained via CNML can incentivize exploration and provide reward shaping to guide RL. However, the reward inference technique via CNML described in Section 3.4 is in most cases computationally intractable, as it requires optimizing separate maximum likelihood estimation problems to convergence on every data point we want to query. In this section, we describe a novel approximation that allows us to apply this method in practice.

Meta-Learning for CNML

We adopt ideas from meta-learning to amortize the cost of obtaining the CNML distribution. As noted in Section 3.4, computing the CNML distribution involves repeatedly solving maximum likelihood problems. While computationally daunting, these problems share a significant amount of common structure, which we can exploit to estimate the CNML distribution more efficiently. Meta-learning uses a distribution of training problems to explicitly meta-train models that can quickly adapt to new problems and, as we show next, can be directly used to accelerate CNML.

To apply meta-learning to computing the CNML distribution, we can formulate each of the maximum likelihood problems described in Equation 3.2 as a separate task for meta-learning, and apply a standard meta-learning technique to obtain a model capable of few-shot adaptation to the maximum likelihood problems required for CNML. While any meta-learning algorithm is applicable, we found model agnostic meta-learning (MAML) [140] to be effective. MAML aims to meta-train a model that can quickly adapt to new tasks via a few steps of gradient descent by explicitly performing a bi-level optimization. We refer readers to [140] for a detailed overview.

The meta-training procedure to enable quick querying of CNML likelihoods can be described as follows. Given a dataset $\mathcal{D} = \{(\mathbf{x_0}, \mathbf{y_0}), (\mathbf{x_1}, \mathbf{y_1}), ..., (\mathbf{x_n}, \mathbf{y_n})\}$, we construct $2n$ different tasks τ_i, each corresponding to performing maximum likelihood estimation on the original dataset $\mathcal{D}$ combined with an additional point $(\mathbf{x_i}, \mathbf{y'})$, where y' is a proposed label of either 0 or 1 and $\mathbf{x_i}$

is a point from the dataset $\mathcal{D}$. Given these constructed tasks $\mathcal{S}(\tau)$, we perform meta-training as described by [140]:

$$\max_{\theta}\ \mathbb{E}_{\mathbf{x_i}\sim\mathcal{D},y'.\in\{0,1\}}[\mathcal{L}(\mathcal{D}\cup(\mathbf{x_i},\mathbf{y}'),\theta_i')], \tag{3.6}$$

$$s.t\ \ \theta_i' = \theta - \alpha\nabla_\theta\mathcal{L}(\mathcal{D}\cup(\mathbf{x_i},\mathbf{y}'),\theta). \tag{3.7}$$

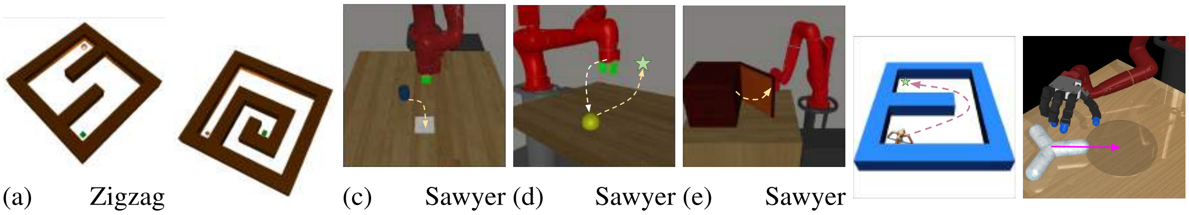
This training procedure produces a set of parameters θ that can then be quickly adapted to provide the CNML distribution with a step of gradient descent. The model can be queried for the CNML distribution by starting from the meta-learned θ and taking one step of gradient descent for the dataset augmented with the query point, each with a different potential label. These likelihoods are then normalized to provide the CNML distribution as follows:

$$p_{\text{meta-NML}}(y|x;\mathcal{D}) = \frac{p_{\theta_y}(y|x)}{\sum_{y\in\mathcal{Y}} p_{\theta_y}(y|x)} \tag{3.8}$$

$$\theta_y = \theta - \alpha\nabla_\theta\mathbb{E}_{(x_i,y_i)\sim\mathcal{D}\cup(x,y)}[\mathcal{L}(x_i,y_i,\theta)]. \tag{3.9}$$

This process is illustrated in Fig 3.3, which shows how the meta-NML procedure can be used to obtain approximate CNML likelihoods with just a single gradient step.

This scheme for amortizing the computational challenges of NML (which we call *meta-NML*) allows us to obtain normalized likelihood estimates without having to retrain maximum likelihood to convergence at every single query point. A complete description, runtime analysis and pseudocode of this algorithm are provided in Appendix A.2 and A.3. Crucially, we find that meta-NML is able to approximate the CNML outputs well with just one or a few gradient steps, making it several orders of magnitude faster than standard CNML.



(a) Zigzag Maze (b) Spiral Maze (c) Sawyer Push (d) Sawyer Pick (e) Sawyer Door (f) Locomotion (g) Dex. Hand

Figure 3.4: We evaluate on two mazes, three robotic arm manipulation tasks, one locomotion task and one dexterous manipulation task: (a) the agent must navigate around an S-shaped corridor, (b) the agent must navigate a spiral corridor, (c) the robot must push a puck to location, (d) the robot must raise a randomly placed tennis ball to location, (e) the robot must open the door a specified angle. (f) the quadruped ant must navigate the maze to a particular location (g) the dexterous robotic hand must reposition an object on the table.

Applying Meta-NML to Success Classification

We apply the meta-NML algorithm described previously to learning uncertainty-aware success classifiers for providing rewards for RL in our proposed algorithm, which we call MURAL. Similarly

Algorithm 2: MURAL: Meta-learning Uncertainty-aware Rewards for Automated Outcome-driven RL

```
User provides success examples $\mathcal{S}_+$
Initialize policy $\pi$, replay buffer $\mathcal{S}_-$, and reward classifier parameters $\theta_\mathcal{R}$
**for** iteration $i = 1, 2, ...$ **do**
  Add on-policy samples to $\mathcal{S}_-$ by executing $\pi$.
  **if** iteration $i \mod k$ == 0 **then**
    Sample $n_{\text{train}}$ states from $\mathcal{S}_-$ to create $2n_{\text{train}}$ meta-training tasks
    Sample $n_{\text{test}}$ total test points equally from $\mathcal{S}_+$ (label 1) and $\mathcal{S}_-$ (label 0)
    Meta-train $\theta_\mathcal{R}$ via meta-NML using Equation 3.7
  **end if**
  Assign state rewards via Equation 3.5
  Train $\pi$ with RL algorithm
**end for**
```

to [97], we can train our classifier by first constructing a dataset $\mathcal{D}$ for binary classification, using success examples as positives, and on-policy samples as negatives, balancing the number of sampled positives and negatives in the dataset. Given this dataset, the classifier parameters $\theta_\mathcal{R}$ can be trained via meta-NML as described in Equation 3.7. The classifier can then be used to directly and quickly assign rewards to a state s according to its probabilities $r(s) = p_{\text{meta-NML}}(e = 1|s)$, and perform standard reinforcement learning, as noted in Algorithm 2. Further details are in Appendix A.2.

3.6 Experimental Evaluation

In our experimental evaluation we aim to answer the following questions: (1) Can MURAL make effective use of successful outcome examples to solve challenging exploration tasks? (2) Does MURAL scale to dynamically complex tasks? (3) What are the impacts of different design decisions on the effectiveness of MURAL?

Further details, videos, and code can be found at `https://sites.google.com/view/mural-rl`

Experimental Setup

We first evaluate our method on maze navigation problems, which require avoiding several local optima. Then, we consider three robotic manipulation tasks that were previously covered in [116] with a Sawyer robot arm: door opening, tabletop object pushing, and 3D object picking. We also evaluate on a previously considered locomotion task [125] with a quadruped ant navigating to a target, as well as a dexterous manipulation problem with a robot repositioning an object with a multi-fingered hand. In the hand manipulation experiments, the classifier is provided with access to only the object position, while in the other tasks the classifier is provided the entire Markovian state. As we show in our results, exploration in these environments is challenging, and using naïvely chosen reward shaping often does not solve the problem at hand.

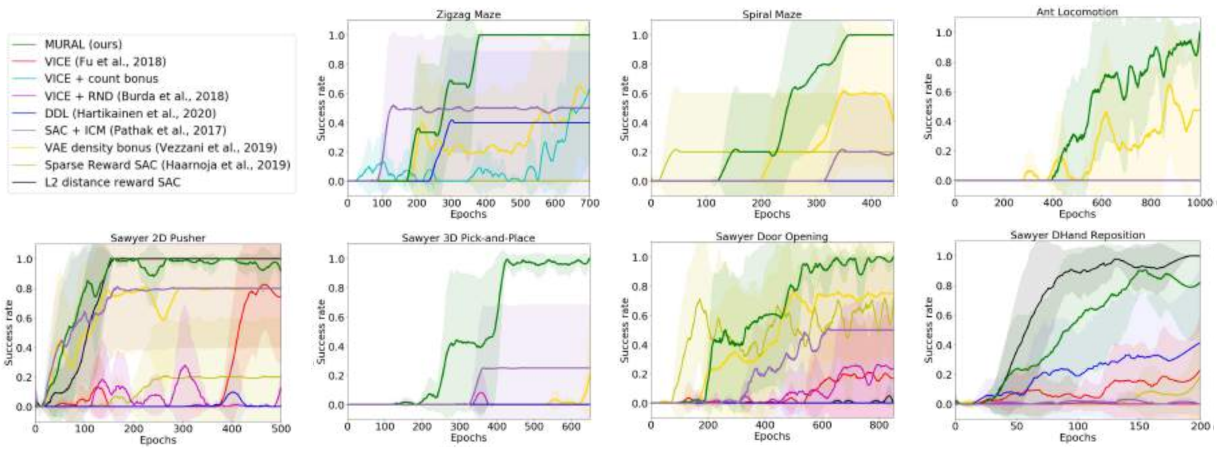


Figure 3.5: MURAL outperforms prior goal-reaching and exploration methods on all our evaluation environments, including ones with high-dimensional state and action spaces. MURAL also performs comparably to or better than a heuristically shaped hand-designed reward that uses Euclidean distance (black line), demonstrating that designing a well-shaped reward is not trivial in these domains. Shading indicates a standard deviation across 5 seeds. For details on the success metrics used, see Appendix A.4.

We compare with a number of prior algorithms. To provide a comparison with a previous method that uses standard success classifiers, we include the VICE algorithm [97]. Note that this algorithm is quite related to MURAL, but it uses a standard maximum likelihood classifier rather than a classifier trained with CNML and meta-learning. We also include a comparison with DDL, a technique for learning dynamical distances [141]. We additionally include comparisons to algorithms for task-agnostic exploration to show that MURAL performs more directed exploration and reward shaping. To provide a direct comparison, we use the same VICE method for training classifiers, but combine it with novelty-based exploration based on random network distillation [110] for the robotic manipulation tasks, and oracle inverse count bonuses for maze navigation. We also compare to prior task-agnostic exploration techniques which use intrinsic curiosity [106] and density estimates [142]. Finally, to demonstrate the importance of shaped rewards, we compare to running Soft Actor-Critic [23] with two naïve reward functions: a sparse reward, and a heuristic reward which uses L2 distance. More details are included in Appendix A.4 and A.6.

Comparisons with Prior Algorithms

We compare with prior algorithms on the domains described above. As we can see in Fig 3.5, MURAL is able to very quickly learn how to solve these challenging exploration tasks, often reaching better asymptotic performance than most prior methods, and doing so more efficiently than VICE [97] or DDL [141]. This suggests that MURAL is able to provide directed reward shaping and exploration that is substantially better than standard classifier-based methods. We provide a more detailed analysis of the shaping behavior of the learned reward in Section 3.6.

To isolate whether the benefits purely come from exploration or also from task-aware reward shaping, we compare with methods that only perform task-agnostic exploration. From these comparisons, it is clear that MURAL significantly outperforms methods that only use novelty-seeking exploration. We also compare to a heuristically-designed reward function based on Euclidean distance. MURAL generally outperforms simple manual shaping in terms of sample complexity and asymptotic performance, indicating that the learned shaping is non-trivial and adapted to the task. Of course, with sufficient domain knowledge, it is likely that this would improve. In addition, we find that MURAL scales up to tasks with challenging exploration in higher dimensional state and action spaces such as quadruped locomotion and dexterous manipulation, as seen in Fig 3.5.

Analysis of MURAL

MURAL and reward shaping. To better understand how MURAL provides reward shaping, we visualize the rewards for various slices along the z axis on the Sawyer Pick-and-Place task, an environment which presents a significant exploration challenge. In Fig 3.6 we see that the MURAL rewards clearly correlate with the distance to the mean object position in successful outcomes, shown as a white star, thus guiding the robot to raise the ball to the desired location even if it has never reached this before. In contrast, the maximum likelihood classifier has a sharp, poorly-shaped decision boundary.

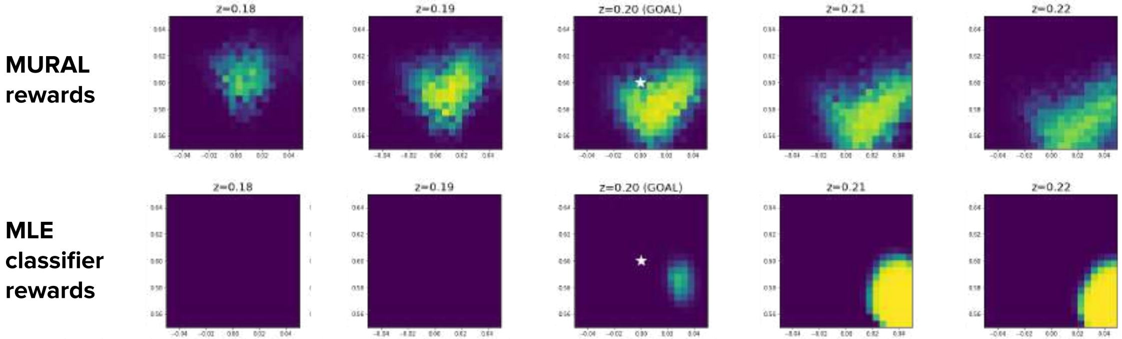


Figure 3.6: Visualization of reward shaping for 3D Pick-and-Place at various z values (heights). MURAL learns rewards that provide a smooth slope toward the successful outcomes, while the MLE classifier learns a sharp and poorly shaped decision boundary.

MURAL and exploration. Next, to illustrate the connection between MURAL and exploration, we compare the states visited by MURAL and by VICE [97] in Figure 3.7. We see that MURAL naturally incentivizes the agent to visit novel states, allowing it to navigate around local minima. In contrast, VICE learns a misleading reward function that prioritizes closeness to the success outcomes in xy space, causing the agent to get stuck.

Interestingly, despite incentivizing exploration, MURAL does not simply visit all possible states; at convergence, it has only covered around 70% of the state space. In fact, in Figure 3.7, MURAL prioritizes states that bring it closer to the success outcomes and ignores ones that don't, making use of the positive examples provided to it. This suggests that MURAL benefits from both novelty-seeking behavior and effective reward shaping.

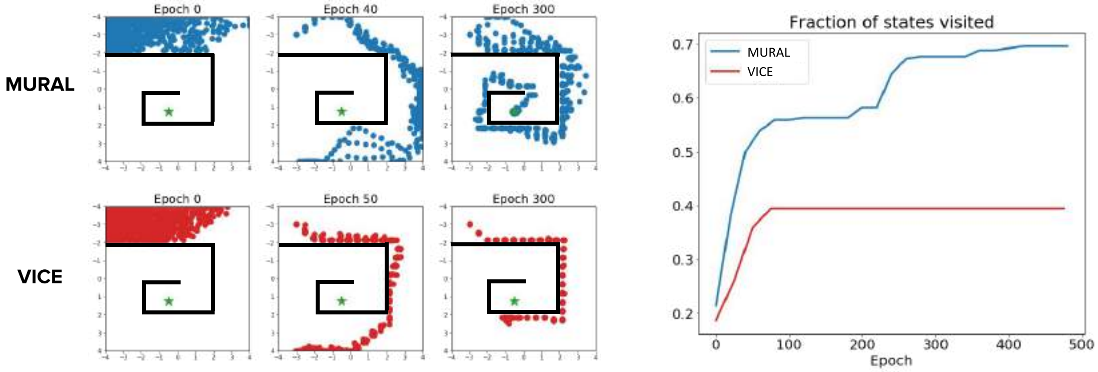


Figure 3.7: Plots of visitations and state coverage over time for MURAL vs. VICE. MURAL explores a significantly larger portion of the state space and is able to avoid local optima.

3.7 Discussion

In this work, we consider a subclass of RL problems where examples of successful outcomes specify the task. We analyze how solutions via standard success classifiers suffer from shortcomings, and training classifiers via CNML allows for better exploration to solve challenging problems. To make learning tractable, we propose a novel meta-learning approach to amortize the CNML process. While this work has shown the effectiveness of Bayesian classifiers for reward inference for tasks in simulation, it would be interesting to scale this solution to real world problems.

We will place this work into the broader context and discuss connections to other work post publication in Chapter 7.

Chapter 4

Supervision from Language Corrections

In the previous two chapters, we studied how we can infer supervision signal from raw human videos as well as examples of successful outcomes. As we move to more scalable learning techniques, it is prudent to wonder whether we can actually use very natural forms of supervision such as natural language feedback in order to actually guide the learning of behaviors in reinforcement learning. This chapter aims to study this question in detail.

4.1 Why Should We Use Language Feedback to Supervise RL algorithms?

Behavioral skills or policies for autonomous agents are typically specified in terms of reward functions (in the case of reinforcement learning) or demonstrations (in the case of imitation learning). However, both reward functions and demonstrations have downsides as mechanisms for communicating goals. Reward functions must be engineered manually, which can be challenging in real-world environments, especially when the learned policies operate directly on raw sensory perception. Sometimes, simply defining the goal of the task requires engineering the very perception system that end-to-end deep learning is supposed to acquire. Demonstrations sidestep this challenge, but require a human demonstrator to actually be able to perform the task, which can be cumbersome or even impossible. When humans must communicate goals to each other, we often use language. Considerable research has also focused on building autonomous agents that can follow instructions provided via language ([143], [144]). However, a single instruction may be insufficient to fully communicate the full intent of a desired behavior. For example, if we would like a robot to position an object on a table in a particular place, we might find it easier to guide it by telling it which way to move, rather than verbally defining a coordinate in space. Furthermore, an autonomous agent might be unable to deduce *how* to perform a task from a single instruction, even if it is very precise. In both cases, interactive and iterative corrections can help resolve confusion and ambiguity, and indeed humans often employ corrections when communicating task goals to each other.

In this paper, our goal is to enable an autonomous agent to accept instructions and then iteratively adjust its policy by incorporating interactive *corrections* (illustrated in Figure 13.1). This type of in-the-loop supervision can guide the learner out of local optima, provide fine-grained task definition, and is natural for humans to provide to the agent. As we discuss in Section 4.2, iterative language corrections can be substantially more informative than simpler forms of supervision, such as preferences, while being substantially easier and more natural to provide than reward functions or demonstrations.

In order to effectively use language corrections, the agent must be able to ground these corrections to concrete behavioral patterns. We propose an end-to-end algorithm for grounding iterative language corrections by using a multi-task setup to *meta-train* a model that can ingest its own past behavior and a correction, and then correct its behavior to produce better actions. During a meta-training phase, this model is iteratively retrained on its own behavior (and the corresponding correction) on a wide distribution of known tasks. The model learns to correct the types of mistakes that it actually tends to make in the world, by interpreting the language input. At meta-test time, this model can then generalize to new tasks, and learn those tasks quickly through iterative language corrections.

The main contributions of our work are the formulation of guided policies with language (GPL) via meta-learning, as well as a practical GPL meta-learning algorithm and model. We train on English sentences sampled from a hand-designed grammar as a first step towards real human-in-the-loop supervision. We evaluate our approach on two simulated tasks - multi-room object manipulation and robotic object relocation. The first domain involves navigating a complex world with partial observation, seeking out objects and delivering them to user-specified locations. The second domain involves controlling a robotic gripper in a continuous state and action space to move objects to precise locations in relation to other objects. This requires the policy to ground the corrections in terms of objects and places, and to control and correct complex behavior.

4.2 Relationship to Prior Work

Tasks for autonomous agents are most commonly specified by means of reward functions [27] or demonstrations [65]. Prior work has studied a wide range of different techniques for both imitation learning [69], [71] and reward specification, including methods that combine the two to extract reward functions and goals from user examples [97], [145] and demonstrations [100], [146]. Other works have proposed modalities such as preferences [147] or numerical scores [148]. Natural language presents a particularly appealing modality for task specification, since it enables humans to communicate task goals quickly and easily. Unlike demonstrations, language commands do not require being able to perform the task. Unlike reward functions, language commands do not require any manual engineering. Finally, in comparison to low-bandwidth supervision modalities, such as examples of successful outcomes or preferences, language commands can communicate substantially more information, both about the goals of the task and how it should be performed.

A considerable body of work has sought to ground natural language commands in meaningful behaviors. These works typically use a large supervised corpus in order to learn policies that are conditioned on natural language commands [143], [144], [149]–[157]. Other works consider using a known reward function in order to learn how to ground language into expert behaviors [154],

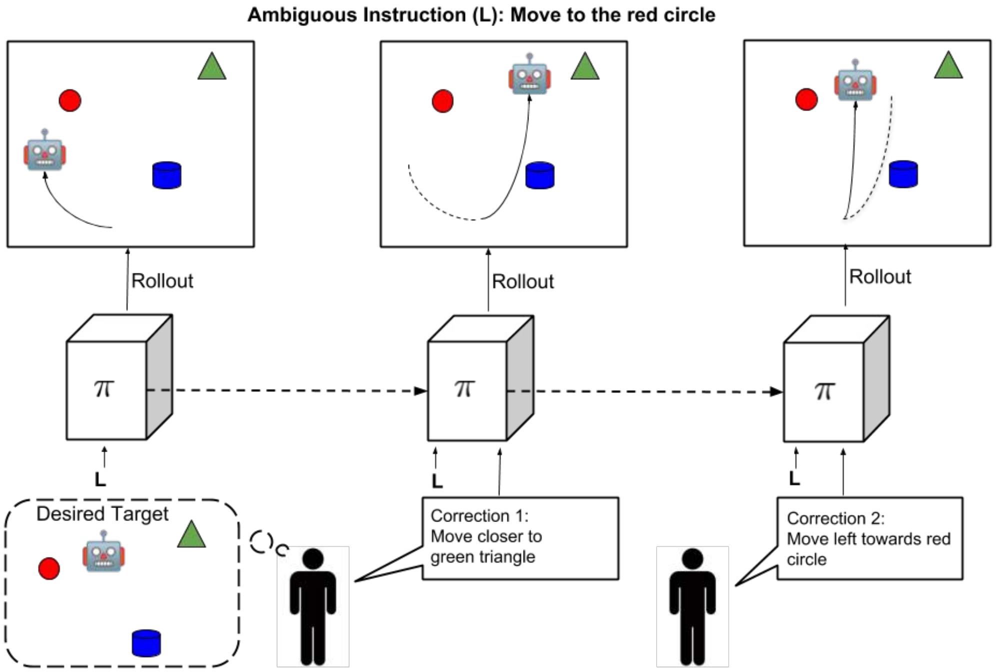

Ambiguous Instruction (L): Move to the red circle
Rollout
Rollout
Rollout
π
π
π
L
L
L
Desired Target
Correction 1: Move closer to green triangle
Correction 2: Move left towards red circle

Figure 4.1: An example where corrections disambiguate an instruction. The agent is unable to fully deduce the user's intent from the instruction alone and iterative language corrections guide the agent to the correct position. Our method is concerned with meta-learning policies that can ground language corrections in their environment and use them to improve through iterative feedback.

[158]. Most of these works consider the case of instruction following. However, tasks can often be quite difficult to specify with a single language description, and may require interactive guidance in order to be achieved. We focus on this setting in our work, where the agent improves its behavior via iterative language corrections.

While the focus in our work is on incorporating language corrections, several prior works have also studied reinforcement learning and related problems with in-the-loop feedback of other forms [148], [159]–[164]. In contrast, we study how to incorporate language corrections, which are more natural for humans to specify and can carry more information about the task. However, language corrections also present the challenge that the agent must learn how to ground them in behavior. To this end, we introduce an end-to-end algorithm that directly associates language with changes in behavior without intermediate supervision about object identities or word definitions.

Our approach to learning to learn from language corrections is based on meta-reinforcement learning. In meta-reinforcement learning, a meta-training procedure is used to learn a procedure (represented by initial network weights or a model that directly ingests past experience) [165] that can adapt to new tasks at meta-test time. However, while prior work has proposed meta-reinforcement learning for model-free RL [140], [163], [166], [167], model-based RL [168], a wide range of supervised tasks [169]–[172], as well as goal specification [173], [174], to our knowledge no prior work has proposed meta-training of policies that can acquire new tasks from iterative language corrections.

4.3 Problem Formulation

In this work, the agent's goal is specified by a language instruction L. This instruction describes what the general objective of the task is, but may be insufficient to fully communicate the intent of a desired behavior. The agent can attempt the task multiple times, and after each attempt, the agent is provided with a language *correction*. Each attempt results in a trajectory $\tau = (s_0, a_0, s_1, a_1, ..., s_T, a_T)$, the result of the agent executing its policy $\pi_\theta(a|s, L)$ in the environment. After each attempt, the user generates a correction according to some unknown stochastic function of the trajectory $c \sim \mathcal{F}(\tau)$. Here, cis a language phrase that indicates how to improve the current trajectory τ to bring it closer to accomplishing the goal. This process is repeated for multiple trials, and we will use τ_i to denote the trajectory on the i^{th} trial, and c_i to denote the corresponding correction. An effective model should be able to incorporate these corrections to come closer to achieving the goal. This process is illustrated in Figure 13.1.

In the next section, we will describe a model that can incorporate iterative corrections, and then describe a meta-training procedure that can train this model to incorporate corrections effectively.

4.4 The Language-Guided Policy Learning Model

As described in Section 4.3, our model for guiding policies with language (GPL) must take in an initial language instruction, and then iteratively incorporate corrections after each attempt at the task. This requires the model to ground the contents of the correction in the environment, and also interpret it in the context of its own previous trajectory so as to decide which actions to attempt next. To that end, we propose a deep neural network model, shown in Figure 5.1, that can accept the instruction, correction, previous trajectory, and state as input. The model consists of three modules: an instruction following module, a correction module, and a policy module.

The instruction following module interprets the initial language instruction. The instructions are provided as a sequence of words which is converted into a sequence of word-embeddings. A 1D CNN processes this sequence to generate an instruction embedding vector z_{im} which is fed into the policy module.

The correction module interprets the previous language corrections $\mathbf{c}_n = (c_0, \ldots, c_n)$ in the context of the previous trajectories $\boldsymbol{\tau}_n = (\tau_0. \ldots, \tau_n)$. Each previous trajectory is processed by a 1D CNN to generate a trajectory embedding. The correction c_i, similar to the language description, is converted into a sequence of word-embeddings which is then fed through a 1D CNN. The correction and trajectory embeddings are then concatenated and transformed by a MLP to form a single tensor. These tensors are then averaged to compute the full correction history tensor $z_{cm} = \frac{1}{n} \sum_{j=0}^{n} \text{MLP}(\text{1dCNN}(\tau_j), \text{1dCNN}(c_j))$.

The policy module has to integrate the high level description embedded into z_{im}, the actionable changes from the correction module z_{cm}, and the environment state s, to generate the right action. This module inputs z_{cm}, z_{im} and s and generates an action distribution $p(a|s)$ that determine how the agent should act. Specific architecture details are described in the appendix.

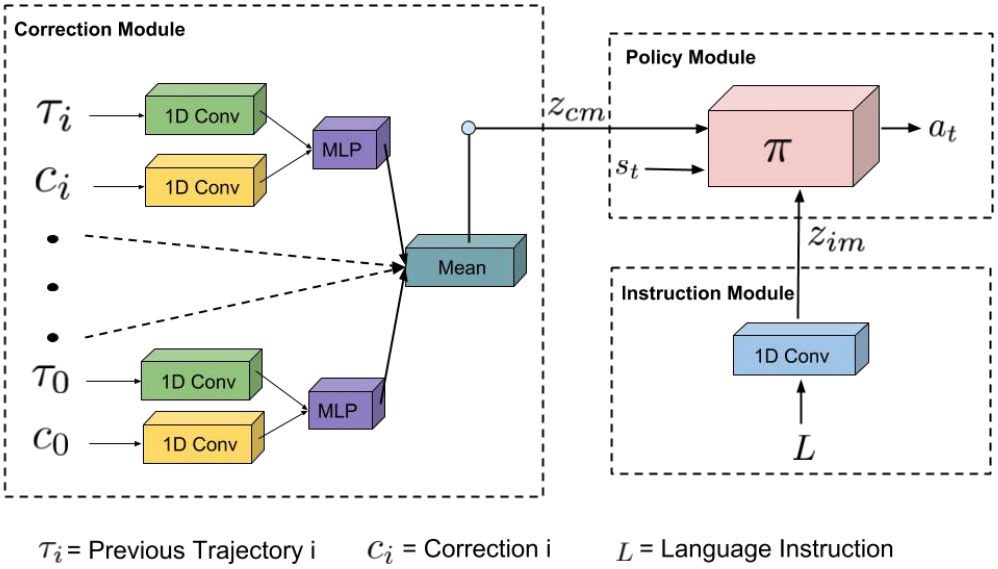


Figure 4.2: The architecture of our model. The instruction module embeds the initial instruction L, while the correction modules embed the trajectory τ_i and correction c_i from each previous trial. The features from these corrections are pooled and provided to the policy, together with the current state s and the embedded initial instruction.

Note that, by iteratively incorporating language corrections, such a model in effect implements a *learning algorithm*, analogously to meta-reinforcement learning recurrent models proposed in prior work that read in previous trajectories and rewards [163], [166]. However, in contrast to these methods, our model has to use the language correction to improve, essentially implementing an interactive, user-guided reinforcement learning algorithm. As we will demonstrate in our experiments, iterative corrections cause this model to progressively improve its performance on the task very quickly. In the next section, we will describe a meta-learning algorithm that can train this model such that it is able to adapt to iterative corrections effectively at meta-test time.

4.5 Meta-Training the GPL Model to Learn From Corrections

In order for the GPL model to be able to learn behaviors from corrections, it must be meta-trained to understand both instructions and corrections properly, put them in the context of its own previous trajectories, and associate them with objects and events in the world. For clarity of terminology, we will use the term “meta-training” to denote the process of training the model, and “meta-testing” to denote its use for solving a new task with language corrections.

During meta-training, we assume access to samples from a distribution of meta-training tasks $T \sim p(T)$. The tasks that the model will be asked to learn at meta-test time are distinct from the meta-training tasks, though we assume them to be drawn from the same distribution, which is analogous to the standard distribution assumption in supervised learning. The tasks have the same state space $\mathcal{S}$ and action space $\mathbf{A}$, but each task has a distinct goal, and each task T can be described

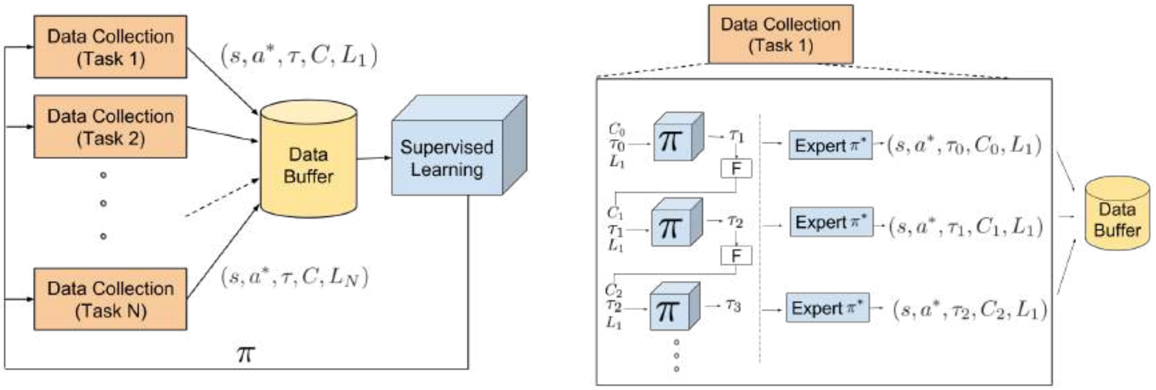


Figure 4.3: **Left:** Overall training procedure. We collect data for each task $[1, 2, \dots, N]$ using DAgger, storing it in the data buffer. This is used to train a GPL policy with supervised learning. The trained policy is then used to collect data for individual tasks again, repeating the process until convergence. **Right:** Individual data collection procedure for a single task. The GPL policy is initially executed to obtain a trajectory τ_1. This trajectory is corrected by an expert π^* to generate data to be added to the buffer. The trajectory is then used to generate a correction, which is fed back into π, along with τ_1 to generate a new trajectory τ_2. This repeats until a maximum number of corrections is given, adding data to the buffer at each step.

by a different language instruction L_T. In general, more than one instruction could describe a single task, and the instructions might contain ambiguity.

Each of the tasks during meta-training has a ground truth objective provided by a reward function. We can use the reward function with any existing reinforcement learning algorithm to train a near-optimal policy. Therefore, we derive the algorithm for the case where we have access to a near-optimal policy $\pi_T^*(a|s)$ for each task T. In our experiments, $\pi_T^*(a|s)$ is obtained via reinforcement learning from ground truth rewards. For each meta-training task, we also assume that we can sample from the corresponding correction function $\mathcal{F}_T(c|\tau)$, which generates a correction c for the trajectory τ in the context of task T. In practice, these corrections might be provided by a human annotator, though we use a computational proxy in our experiments. The key point is that we can use rewards to transform a collection of task-specific policies learned offline into a fast online learner during test time.

By using $\pi_T^*(a|s)$, L_T, and $\mathcal{F}_T(\tau)$, we can train our model for each task by using a variant of the DAgger algorithm [175], which was originally proposed for single-task imitation learning, where a learner policy is trained to mimic a near-optimal expert. We extend this approach to the setting of meta-learning, where we use it to meta-train the GPL model. Starting from an initialization where the previous trajectory τ_0 and correction c_0 are set to be empty sequences, we repeat the following process: first, we run the policy corresponding to the current learned model $\pi(a|s, L_T, \boldsymbol{\tau}_0, \mathbf{c}_0)$ to generate a new trajectory τ_1 for the task T. Every state along τ_1 is then labeled with near-optimal actions by using $\pi_T^*(a|s)$ to produce a set of training tuples $(L_T, \tau_0, c_0, s, a^*)$. These tuples are appended to the training set $\mathcal{D}$. Then, a correction c_1 is sampled from $\mathcal{F}_T(c|\tau_1)$, and a new trajectory is sampled from $\pi(a|s, L_T, \boldsymbol{\tau}_1, \mathbf{c}_1)$. This trajectory is again labeled by the expert and appended to the dataset. In the same way, we iteratively populate the training set $\mathcal{D}$ with the states, corrections, and prior trajectories observed by the model, all labeled with near-optimal actions. This process is repeated for a fixed number of corrections or until the task is solved, for each of the meta-training tasks. The model is then trained to maximum the likelihood of the samples in the dataset $\mathcal{D}$. Then, following the DAgger algorithm, the updated policy is used to again collect data for each of the tasks, which is appended to the dataset and used to train the policy again, until the process converges or a fixed number of iterations. This algorithm is summarized in Algorithm 3 and Figure 4.3.

4.6 Learning New Tasks with The GPL Model

Using a GPL model meta-trained as described in the previous section, we can solve new "meta-testing" tasks $T_{test} \sim p(T)$ drawn from the same distribution of tasks with interactive language corrections. An initial instruction L_T is first provided by the user, and the procedure for adapting with corrections follows the illustration in Figure 13.1. The learned policy is initially rolled out in the environment conditioned on L_T, and with the previous trajectory τ_0 and correction c_0 initialized to the empty sequence. Once this policy generates a trajectory τ_1, we can use the correction function $\mathcal{F}_T$ to generate a correction $c_1 = \mathcal{F}_T(\tau_1)$. The trajectory τ_1, along with the correction c_1 gives us a new improved policy which is conditioned on L_T, $\boldsymbol{\tau}_1$, and $\mathbf{c}_1$. This policy can be executed in the environment to generate a new trajectory τ_2, and the process repeats until convergence, thereby learning the new task. We provide the policy with the previous corrections as well but omit in the notation for clarity. This procedure is summarized in Algorithm 4.

This procedure is similar to meta-reinforcement learning [140], [166], but uses grounded natural language corrections in order to guide learning of new tasks with feedback provided in the loop. The advantage of such a procedure is that we can iteratively refine behaviors quickly for tasks that are hard to describe with high level descriptions. Additionally, providing language feedback iteratively in the loop may reduce the overall amount of supervision needed to learn new tasks. Using easily available natural language corrections in the loop can change behaviors much more quickly than scalar reward functions.

4.7 Experiments

Our experiments analyze GPL in a partially observed object manipulation environment and a block pushing environment. The first goal of our evaluation is to understand where GPL can benefit from iterative corrections – that is, does the policy's ability to succeed at the task improve as each new correction provided. We then evaluate our method comparatively, in order to understand whether iterative corrections provide an improvement over standard instruction-following methods, and also compare GPL to an oracle model that receives a much more detailed instruction, but without the iterative structure of interactive corrections. We perform further experiments including ablations to evaluate the importance of each module and additional experiments with more varied corrections. We also compare our method to state of the art instruction following methods, other baselines which use intermediate rewards instead of corrections, and pretraining with language. Our code and

Algorithm 3: GPL meta-training algorithm.

```
Initialize data buffer $\mathcal{D}$;
for iteration $j$ do
    for task $T$ do
        Initialize $\tau_0 = 0$ and $c_0 = 0$ ;
        for corr iter $i \in \{0, ..., c_{max}\}$ do
            Execute $\pi(a|s, L_T, \boldsymbol{\tau}_i, \mathbf{c}_i)$ on $T$ to collect $\tau_{i+1}$;
            Obtain $c_{i+1} \sim \mathcal{F}_T(\tau_{i+1})$ ;
            Label $a^\star \sim \pi^\star_T(a|s)$, $\forall\ s \in \tau_{i+1}$;
            Add all $(L_T, \tau_i, c_i, s, a^\star)$ to $D$;
        end for
    end for
    Train $\pi$ on $\mathcal{D}$.
end for
```

Algorithm 4: Meta-testing: learning new tasks with the GPL model.

```
Given new task $T_i$, with instruction $L_T$;
Initialize $\tau_0 = 0$ and $c_0 = 0$ ;
for corr iter $i \in \{0, ..., c_{max}\}$ do
    Execute $\pi(a|s, L_T, \boldsymbol{\tau}_i, \mathbf{c}_i)$ on $T$ to collect $\tau_{i+1}$ ;
    Obtain $c_{i+1} \sim \mathcal{F}_T(\tau_{i+1})$ ;
end for
```

supplementary material will be available at `https://sites.google.com/view/lgpl/home`

Experimental Setup

We describe the two experimental domains that we evaluate this method on. The first task is underspecified while the second task is ambiguous so each task lends itself to the use of corrections.

Multi-room Object Manipulation

Our first environment is discrete and represents the floor-plan of a building with six rooms (Figure 4.4), based on [176]. The task is to pickup a particular object from one room and bring it to a goal location in another room. Each room has a uniquely colored door that must be opened to enter the room, and rooms contain objects with different colors and shapes. Actions are discrete and allow for cardinal movement and picking up and dropping objects. The environment is partially observed: the policy only observes the contents of an ego-centric 7x7 region centered on the present location of the agent, and does not see through walls or closed doors. The contents of a room can only be observed by first opening its door.

This environment allows for the natural language instruction which specifies the task to be underspecified. The instruction is given as "Move <goal object color> <goal object shape> to <goal square color> square". Since the agent cannot see into closed rooms, it does not initially know the locations of the goal object or goal square. It must either explore or rely on external corrections which guide the agent to the appropriate rooms.

Environments are generated by sampling a goal object color, goal object shape, and goal square color which are placed at random locations in different random rooms. There are 6 possible colors and 3 possible object shapes. Decoy objects and goals are placed randomly among the six rooms. The only shared aspect between tasks are the color of the doors so the agent must learn to generalize across a variety of different objects across different locations.

To generate the corrections, we describe a task as a list of subgoals that the agent must complete. For example, the instruction in Figure 4.4 is "Move green triangle to green square", and the subgoals are "enter the blue room", "pick up the green triangle", "exit the blue room", "enter the purple room", and "go to the green goal". The correction for a given trajectory is then the first subgoal that the agent failed to complete. The multistep nature of this task also makes it challenging, as the agent must remember to solve previously completed subgoals while incorporating the corrections to solve the next subgoal.

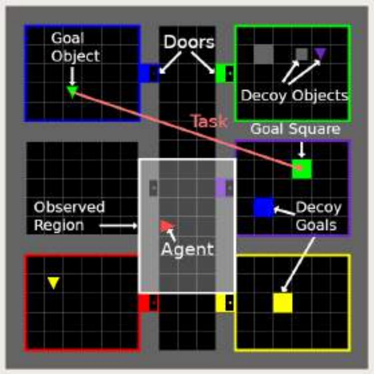


Figure 4.4: The multi-room object manipulation environment with labeled components.

The training and test tasks are generated such that for any test task, its list of all five subgoals does not exist in the training set. There are 3240 possible lists of all five subgoals. We train on 1700 of these environments and reserve a separate set for testing.

Robotic Object Relocation

The second environment is a robotic object relocation task built in Mujoco [91] shown in Figure 4.5. The task is to apply force on a gripper to push one of three blocks to a target location. Five immovable blocks scattered in the environment act as obstacles. States are continuous and include the agent and block positions. Actions are discrete and apply directional forces on the gripper.

The instruction is given as "Move <goal block color> close to <obstacle block color>," where the closest or 2nd closest obstacle block to the target location is randomly chosen. This instruction is ambiguous, as it does not describe the precise location of the target. Corrections will help guide the agent to the right location relative to the obstacles and previous actions of the agent. Environments are generated by sampling one of the three movable blocks to push and sampling a random target location. We generate 1000 of these environments and train on 750 of them. There are three types of corrections, and feedback is provided by stochastically choosing between these types. The correction types are directional ("Move a little left", "Move a lot down right"), relational ("Move left of the white block", "Move closer to the pink block"), or indicate the correct block to push ("Touch the red block").

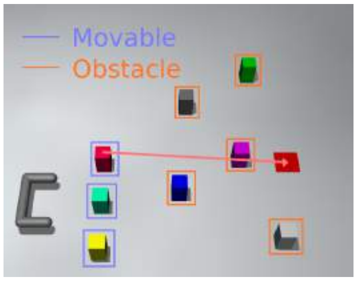


Figure 4.5: The robotic object relocation environment. The agent must push the red block right of the magenta block.

This environment was chosen because of its continuous state space and configuration of objects which allows for spatial relations between objects. It is more natural and convenient for a user to specify a goal with language, e.g., "close to the pink block," than using exact spatial coordinates. Furthermore, because the corrections can be relative to other blocks, the policy must learn to contextualize the correction in relation to other objects and its own previous behavior.

Comparisons

We compare our method to two types of alternative methods - 1) alternative methods for instruction following 2) methods using rewards for finetuning on new tasks.

We first compare our method to a standard instruction following method using our architecture, a method that receives full information, and to an instruction following architecture from [155] which we call MIVOA. We test MIVOA with both the instruction and full information. All methods are trained with DAgger. The instruction following methods only receive the instruction, which is ambiguous and does not contain the precise locations of objects or target positions. The full information methods receive the exact information the agent needs but for which a human may not always want to provide or have access to. For the multi-room environment the full information method receives *all* the subgoals that are needed to solve the task, but does not receive them interactively. The full information method for the robotic object relocation task receives which block to move and the exact coordinates of the goal and so is an oracle. We measure the performance of an agent on the task by computing the completion rate: the fraction of subgoals (max 5) that the agent has successfully completed. The completion rate for the robotic domain is $1 - \frac{\text{final block dist}}{\text{initial block dist}}$. We expect our model to perform better than the instruction following baseline as it can receive the missing information through the corrections and we expect it to come close to performing as well as the full information method. The results for this comparison is shown in Table 4.1.

In the second set of comparisons, we compare the sample complexity of learning new tasks with GPL against other baselines which utilize the reward function for learning new tasks (Figure 4.6). We compare against a reinforcement learning (RL) baseline that trains a separate policy per task using the reward. In addition, we run a baseline that does pretraining with DAgger on the training tasks and then finetunes on the test tasks with RL, thereby following a similar procedure as [156]. In both domains, the pretrained policy receives the instruction. For RL and finetuning we use the same algorithm [18] and reward function used to train the expert policies. Additionally, in order to evaluate if the language corrections provide more information than just a scalar correction we also run a version of our method called GPR (Guiding Policies with Reward) which replaces the language correction with the reward. The correction for a trajectory is the sum of rewards of that trajectory. The performance of all of these methods is shown in Figure 4.6.

Learning New Tasks Quickly with Language Corrections

As described above, we consider two domains - multi-room object manipulation and a robotic object relocation task. In the object relocation domain, the test tasks here consist of new configurations of objects and goals. For the robotic domain, the test tasks consist of pushing to new target locations. Details on the meta-training complexity of our method are in appendix 18.2.

In the first set of comparisons mentioned in Section 4.7, we measure the completion rate of our method for various numbers of corrections on the test tasks. The instruction baseline does not have enough information and is unable to effectively solve the task. As expected, we see increasing completion rates as the number of corrections increases and the agent incrementally gets further in the task. For the multi-room domain our method matches the full information baseline with 3 corrections and outperforms it with 4 or more corrections. Since the full information baseline receives all 5 subgoals, this means our method performs better with *less* information. The interactive nature of our method allows it to receive only the information it needs to solve the task. Furthermore, our model must learn to map corrections to changes in behavior which may be more modular, disentangled, and easier to generalize compared to mapping a long list of instructions to a single policy that can solve the task. For the robotic domain, our model exceeds the performance of the instruction baseline with just 1 correction. With more corrections it comes close to the full information method which receives the exact goal coordinates.

In the second set of comparisons mentioned in Section 4.7, we compare against a number of baselines that use the task reward (Fig 4.6). We see that our method can achieve high completion rate with very few test trajectories. While GPL only receives up to five trajectories on the test task, the RL baseline takes more than 1000 trajectories to reach similar levels of performance. The RL baseline is able to achieve better final performance but takes orders of magnitude more training examples and has access to the test task reward. The pretraining baseline has better sample complexity than the RL baseline but still needs more than 1000 test trajectories. The reward guided version of our method, GPR, performs poorly on the multi-room domain but obtains reasonable performance for the robotic domain. This may indicate that language corrections in the multi-room domain provide much more information than just scalar rewards. However, using scalar rewards or binary preferences may be an alternative to language corrections for continuous state space domains such as ours.

Env	Instruction	Full Info	MIVOA (Instr.)	MIVOA (Full Info)	$\mathbf{c}_0$	$\mathbf{c}_1$	$\mathbf{c}_2$	$\mathbf{c}_3$	$\mathbf{c}_4$	$\mathbf{c}_5$
Multi-room	0.075	0.73	0.067	0.63	0.066	0.46	0.65	0.73	0.77	**0.82**
Obj Relocation	0.64	**0.96**	0.65	-	0.65	0.80	0.84	0.85	0.88	0.90

Table 4.1: Mean completion rates on test tasks for baseline methods and ours across 5 random seeds. $\mathbf{c}_i$ denotes that the agent has received i corrections. GPL is able to quickly incorporate corrections to improve agent behavior over instruction following with fewer corrections than full information on the multi-room domain. MIVOA is an architecture from [155].

Analyzing the Behavior of GPL

To understand GPL better, we perform a number of different analyses - model ablations, extrapolation to more corrections, more varied corrections and generalization to out of distribution tasks.

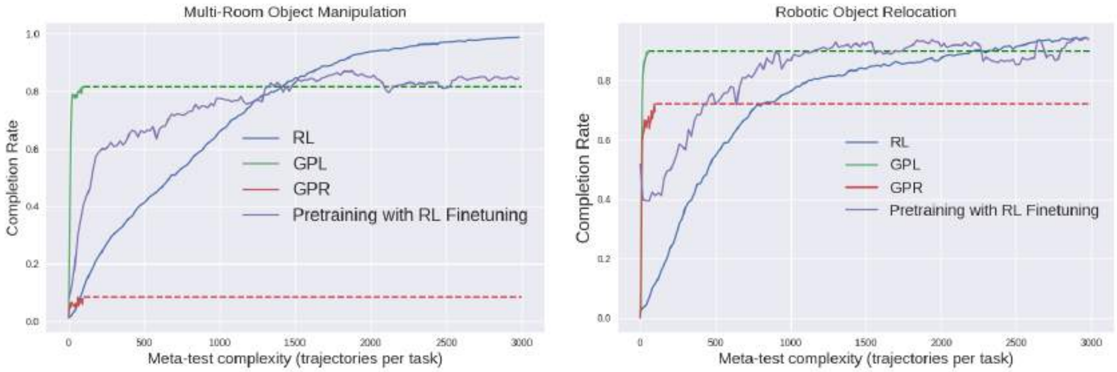


Figure 4.6: Sample complexity on test tasks. The mean completion rate is plotted against the number of trajectories using during training per task. Our method (GPL) is shown in green.

Ablations	$\mathbf{c}_0$	$\mathbf{c}_1$	$\mathbf{c}_2$	$\mathbf{c}_3$	$\mathbf{c}_4$	$\mathbf{c}_5$
Base	0.066	0.46	0.65	0.73	0.77	0.82
No instruction	0.059	0.45	0.62	0.72	0.78	0.79
No trajectory	0.077	0.44	0.62	0.70	0.76	0.77
Only immediate correction	0.067	0.49	0.44	0.58	0.59	0.63

Table 4.2: Ablation Experiments analyzing the importance of various components of the model on the multi-room env. We see that removing previous corrections (only c_i) performs the worst, while removing instruction L is less impactful.

We perform ablations on the multi-room domain to analyze the importance of each component of our model in Figure 18.50. For the three ablations, we remove the instruction L, remove the previous trajectory τ_i, and provide only the immediate previous correction c_i instead of all previous corrections. We find that removing the instruction hurts the performance the least. This makes sense because the model can receive the information contained in the instruction through the corrections. Removing the previous corrections hurts the performance the most. Qualitatively, the agent that does not have access to the previous tends to forget what it had done previously and erases the progress it made. This explains the dip in performance from c_1 to c_2 for the only immediate correction.

We also investigate if performance continues to increase if we provide more corrections at meta-testing time than seen during training. In Table 4.3, we provide up to 10 corrections during test time while only up to 5 corrections are seen during training. We see that completion rate increases from 0.82 to 0.86 for the multi-room domain and from 0.90 to 0.95 for the object relocation domain. These are small gains in performance and we expect to see diminishing returns as the number of corrections increases even further.

Env	$\mathbf{c}_5$	$\mathbf{c}_7$	$\mathbf{c}_{10}$
Multi-room	0.82	0.83	0.86
Obj Relocation	0.90	0.91	0.95

Table 4.3: Mean completion rates on test tasks for 5, 7, 10 corrections. Only up to 5 corrections are seen during training.

In Table 4.4 we investigate what happens if we give an agent more varied corrections in the multi-room environment. We add two new correction types. The first is directional and specifies which of the eight cardinal direction the next goal is in, e.g. "goal room is southwest." The second type is binary and consists of simple "yes/no" type information such as "you are in the wrong room".

Type	$\mathbf{c}_1$	$\mathbf{c}_2$	$\mathbf{c}_3$	$\mathbf{c}_4$	$\mathbf{c}_5$
Directional	0.242	0.343	0.43	0.51	0.56
Binary	0.073	0.078	0.08	0.089	0.09
All	0.236	0.363	0.44	0.538	0.606

Table 4.4: Experiments investigating different correction types and effect on performance (mean completion). The experiments agree with intuition that binary carries little information and results in a small increase of the completion rate. Directional corrections which gives an intermediate amount of information result in a fair increase in performance, but less than fully specified correction.

We also experiment with if our method can generalize to unseen objects in the multi-room domain. We holdout specific objects during training and test on these unseen objects. For example, the agent will not see green triangles but will see other green objects and non-green triangles during training and must generalize to the unseen combination at test time. In Table 4.5, we see that our method achieves a lower completion rate compared to when specific objects are not held out but is still able to achieve a high completion rate and outperforms the baselines.

Env	Full Info	MIVOA (Full Info)	$\mathbf{c}_0$	$\mathbf{c}_1$	$\mathbf{c}_2$	$\mathbf{c}_3$	$\mathbf{c}_4$	$\mathbf{c}_5$
Multi-room	0.57	0.62	0.073	0.44	0.58	0.65	0.73	0.75

Table 4.5: Experiments on holding out specific objects during training to see if our method can generalize to unseen objects at test time.

Discussion and Future Work

We presented meta-learning for guided policies with language (GPL), a framework for interactive learning of tasks with in-the-loop language corrections. In GPL , the policy attempts successive trials in the environment, and receives language corrections that suggest how to improve the next trial over the previous one. The GPL model is trained via meta-learning, using a dataset of other tasks to learn how to ground language corrections in terms of behaviors and objects. While our method currently uses fake language, future work could incorporate real language at training time.

Chapter 5

Learning Skills Without Reward Supervision

In this chapter, we discuss how we can actually move from supervising RL systems using things like raw video or natural language to actually learning skills without *any* task directed reward functions. In this chapter, we discuss a technique for unsupervised skill discovery which is able to propose and learn a set of tasks without any human provided reward functions, and we then discuss how this framework can be incorporated into a broader framework for pre-training reinforcement learning agents with unsupervised reinforcement learning using ideas from meta-reinforcement learning in the following chapter. In this chapter, we start by describing a procedure for unsupervised skill discovery.

5.1 Why Is Unsupervised Skill Discovery Important?

Deep reinforcement learning (RL) has been demonstrated to effectively learn a wide range of reward-driven skills, including playing games [11], [177], controlling robots [35], [178], and navigating complex environments [39], [179]. However, intelligent creatures can explore their environments and learn useful skills even without supervision, so that when they are later faced with specific goals, they can use those skills to satisfy the new goals quickly and efficiently.

Learning skills without reward has several practical applications. Environments with sparse rewards effectively have no reward until the agent randomly reaches a goal state. Learning useful skills without supervision may help address challenges in exploration in these environments. For long horizon tasks, skills discovered without reward can serve as primitives for hierarchical RL, effectively shortening the episode length. In many practical settings, interacting with the environment is essentially free, but evaluating the reward requires human feedback [147]. Unsupervised learning of skills may reduce the amount of supervision necessary to learn a task. While we can take the human out of the loop by designing a reward function, it is challenging to design a reward function that elicits the desired behaviors from the agent [180]. Finally, when given an unfamiliar environment, it is challenging to determine what tasks an agent should be able to learn.

Unsupervised skill discovery partially answers this question.[1]

Autonomous acquisition of useful skills without any reward signal is an exceedingly challenging problem. A *skill* is a latent-conditioned policy that alters the state of the environment in a consistent way. We consider the setting where the reward function is unknown, so we want to learn a set of skills by maximizing the utility of this set. Making progress on this problem requires specifying a learning objective that ensures that each skill individually is distinct and that the skills collectively explore large parts of the state space. In this paper, we show how a simple objective based on mutual information can enable RL agents to autonomously discover such skills. These skills are useful for a number of applications, including hierarchical reinforcement learning and imitation learning.

We propose a method for learning diverse skills with deep RL in the absence of any rewards. We hypothesize that in order to acquire skills that are useful, we must train the skills so that they maximize coverage over the set of possible behaviors. While one skill might perform a useless behavior like random dithering, other skills should perform behaviors that are distinguishable from random dithering, and therefore more useful. A key idea in our work is to use discriminability between skills as an objective. Further, skills that are distinguishable are not necessarily maximally diverse – a slight difference in states makes two skills distinguishable, but not necessarily diverse in a semantically meaningful way. To combat this problem, we want to learn skills that not only are distinguishable, but also are *as diverse as possible.* By learning distinguishable skills that are as random as possible, we can "push" the skills away from each other, making each skill robust to perturbations and effectively exploring the environment. By maximizing this objective, we can learn skills that run forward, do backflips, skip backwards, and perform face flops (see Figure 5.3).

Our paper makes five contributions. First, we propose a method for learning useful skills without any rewards. We formalize our discriminability goal as maximizing an information theoretic objective with a maximum entropy policy. Second, we show that this simple exploration objective results in the unsupervised emergence of diverse skills, such as running and jumping, on several simulated robotic tasks. In a number of RL benchmark environments, our method is able to solve the benchmark task despite never receiving the true task reward. In these environments, some of the learned skills correspond to solving the task, and each skill that solves the task does so in a distinct manner. Third, we propose a simple method for using learned skills for hierarchical RL and find this methods solves challenging tasks. Four, we demonstrate how skills discovered can be quickly adapted to solve a new task. Finally, we show how skills discovered can be used for imitation learning.

5.2 Related Work

Previous work on hierarchical RL has learned skills to maximize a single, known, reward function by jointly learning a set of skills and a meta-controller (e.g., [181]–[186]). One problem with joint training (also noted by [187]) is that the meta-policy does not select "bad" options, so these options do not receive any reward signal to improve. Our work prevents this degeneracy by using a random meta-policy during unsupervised skill-learning, such that neither the skills nor the meta-policy are

[1] See videos here: `https://sites.google.com/view/diayn/`

aiming to solve any single task. A second importance difference is that our approach learns skills *with no reward*. Eschewing a reward function not only avoids the difficult problem of reward design, but also allows our method to learn task-agnostic.

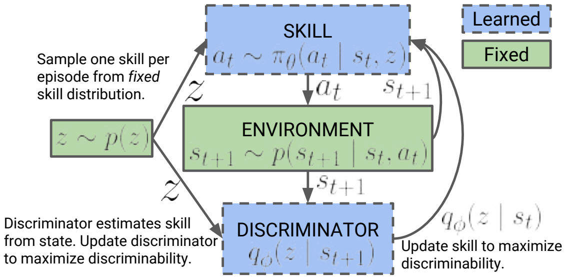
Related work has also examined connections between RL and information theory [71], [188]–[190] and developed maximum entropy algorithms with these ideas [23], [190]. Recent work has also applied tools from information theory to skill discovery. [191] and [192] use the mutual information between states and actions as a notion of empowerment for an intrinsically motivated agent. Our method maximizes the mutual information between states and *skills*, which can be interpreted as maximizing the empowerment of a *hierarchical agent* whose action space is the set of skills. [193], [186], and [194] showed that a discriminability objective is equivalent to maximizing the mutual information between the latent skill z and some aspect of the corresponding trajectory. [193] considered the setting with many tasks and reward functions and [186] considered the setting with a single task reward. Three important distinctions allow us to apply our method to tasks significantly more complex than the gridworlds in [194]. First, we use maximum entropy policies to force our skills to be diverse. Our theoretical analysis shows that including entropy maximization in the RL objective results in the mixture of skills being maximum entropy in aggregate. Second, we fix the prior distribution over skills, rather than learning it. Doing so prevents our method from collapsing to sampling only a handful of skills. Third, while the discriminator in [194] only looks at the final state, our discriminator looks at every state, which provides additional reward signal. These three crucial differences help explain how our method learns useful skills in complex environments.

Prior work in neuroevolution and evolutionary algorithms has studied how complex behaviors can be learned by directly maximizing diversity [195]–[201]. While this prior work uses diversity maximization to obtain better solutions, we aim to acquire complex skills with minimal supervision to improve efficiency (i.e., reduce the number of objective function queries) and as a stepping stone for imitation learning and hierarchical RL. We focus on deriving a general, information-theoretic objective that does not require manual design of distance metrics and can be applied to any RL task without additional engineering.

Previous work has studied intrinsic motivation in humans and learned agents. [106], [109], [202]–[206]. While these previous works use an intrinsic motivation objective to learn a *single* policy, we propose an objective for learning *many*, diverse policies. Concurrent work [207] draws ties between learning discriminable skills and variational autoencoders. We show that our method scales to more complex tasks, likely because of algorithmic design choices, such as our use of an off-policy RL algorithm and conditioning the discriminator on individual states.

5.3 Diversity is All You Need

We consider an unsupervised RL paradigm in this work, where the agent is allowed an unsupervised "exploration" stage followed by a supervised stage. In our work, the aim of the unsupervised stage is to learn skills that eventually will make it easier to maximize the task reward in the supervised stage. Conveniently, because skills are learned without a priori knowledge of the task, the learned skills can be used for many different tasks.


SKILL
$a_t \sim \pi_\theta(a_t \mid s_t, z)$
Learned
Fixed
Sample one skill per episode from *fixed* skill distribution.
z
a_t
s_{t+1}
$z \sim p(z)$
ENVIRONMENT
$s_{t+1} \sim p(s_{t+1} \mid s_t, a_t)$
z
s_{t+1}
$q_\phi(z \mid s_t)$
Discriminator estimates skill from state. Update discriminator to maximize discriminability.
DISCRIMINATOR
$q_\phi(z \mid s_{t+1})$
Update skill to maximize discriminability.

Algorithm 5: DIAYN

```
while not converged do
    Sample skill $z \sim p(z)$ and initial state $s_0 \sim p_0(s)$
    for $t \leftarrow 1$ to $steps\_per\_episode$ do
        Sample action $a_t \sim \pi_\theta(a_t \mid s_t, z)$ from skill.
        Step environment: $s_{t+1} \sim p(s_{t+1} \mid s_t, a_t)$.
        Compute $q_\phi(z \mid s_{t+1})$ with discriminator.
        Set skill reward $r_t = \log q_\phi(z \mid s_{t+1}) - \log p(z)$
        Update policy ($\theta$) to maximize $r_t$ with SAC.
        Update discriminator ($\phi$) with SGD.
```

Figure 5.1: **DIAYN Algorithm**: We update the discriminator to better predict the skill, and update the skill to visit diverse states that make it more discriminable.

How it Works

Our method for unsupervised skill discovery, DIAYN ("Diversity is All You Need"), builds off of three ideas. First, for skills to be useful, we want the skill to dictate the states that the agent visits. Different skills should visit different states, and hence be distinguishable. Second, we want to use states, not actions, to distinguish skills, because actions that do not affect the environment are not visible to an outside observer. For example, an outside observer cannot tell how much force a robotic arm applies when grasping a cup if the cup does not move. Finally, we encourage exploration and incentivize the skills to be as diverse as possible by learning skills that act as randomly as possible. Skills with high entropy that remain discriminable must explore a part of the state space far away from other skills, lest the randomness in its actions lead it to states where it cannot be distinguished.

We construct our objective using notation from information theory: S and A are random variables for states and actions, respectively; $Z \sim p(z)$ is a latent variable, on which we condition our policy; we refer to a the policy conditioned on a fixed Z as a "skill"; $I(\cdot; \cdot)$ and $\mathcal{H}[\cdot]$ refer to mutual information and Shannon entropy, both computed with base e. In our objective, we maximize the mutual information between skills and states, $I(S; Z)$, to encode the idea that the skill should control which states the agent visits. Conveniently, this mutual information dictates that we can infer the skill from the states visited. To ensure that states, not actions, are used to distinguish skills, we minimize the mutual information between skills and actions given the state, $I(A; Z \mid S)$. Viewing all skills together with $p(z)$ as a mixture of policies, we maximize the entropy $\mathcal{H}[A \mid S]$ of this mixture policy. In summary, we maximize the following objective with respect to our policy parameters, θ:

$$\begin{aligned}\mathcal{F}(\theta) &\triangleq I(S;Z) + \mathcal{H}[A \mid S] - I(A;Z \mid S) \qquad (5.1)\\ &= (\mathcal{H}[Z] - \mathcal{H}[Z \mid S]) + \mathcal{H}[A \mid S] - (\mathcal{H}[A \mid S] - \mathcal{H}[A \mid S, Z])\\ &= \mathcal{H}[Z] - \mathcal{H}[Z \mid S] + \mathcal{H}[A \mid S, Z] \qquad (5.2)\end{aligned}$$

We rearranged our objective in Equation 5.2 to give intuition on how we optimize it.[2] The first term encourages our prior distribution over $p(z)$ to have high entropy. We fix $p(z)$ to be uniform in our approach, guaranteeing that it has maximum entropy. The second term suggests that it should be easy to infer the skill z from the current state. The third term suggests that each skill should act as randomly as possible, which we achieve by using a maximum entropy policy to represent each skill. As we cannot integrate over all states and skills to compute $p(z \mid s)$ exactly, we approximate this posterior with a learned discriminator $q_\phi(z \mid s)$. Jensen's Inequality tells us that replacing $p(z \mid s)$ with $q_\phi(z \mid s)$ gives us a variational lower bound $\mathcal{G}(\theta, \phi)$ on our objective $\mathcal{F}(\theta)$ (see [208] for a detailed derivation):

$$\begin{aligned}\mathcal{F}(\theta) &= \mathcal{H}[A \mid S, Z] - \mathcal{H}[Z \mid S] + \mathcal{H}[Z]\\ &= \mathcal{H}[A \mid S, Z] + \mathbb{E}_{z\sim p(z), s\sim\pi(z)}[\log p(z \mid s)] - \mathbb{E}_{z\sim p(z)}[\log p(z)]\\ &\geq \mathcal{H}[A \mid S, Z] + \mathbb{E}_{z\sim p(z), s\sim\pi(z)}[\log q_\phi(z \mid s) - \log p(z)] \triangleq \mathcal{G}(\theta, \phi)\end{aligned}$$

Implementation

We implement DIAYN with soft actor critic (SAC) [23], learning a policy $\pi_\theta(a \mid s, z)$ that is conditioned on the latent variable z. Soft actor critic maximizes the policy's entropy over actions, which takes care of the entropy term in our objective $\mathcal{G}$. Following [23], we scale the entropy regularizer $\mathcal{H}[a \mid s, z]$ by α. We found empirically that an $\alpha = 0.1$ provided a good trade-off between exploration and discriminability. We maximize the expectation in $\mathcal{G}$ by replacing the task reward with the following pseudo-reward:

$$r_z(s, a) \triangleq \log q_\phi(z \mid s) - \log p(z) \qquad (5.3)$$

We use a categorical distribution for $p(z)$. During unsupervised learning, we sample a skill $z \sim p(z)$ at the start of each episode, and act according to that skill throughout the episode. The agent is rewarded for visiting states that are easy to discriminate, while the discriminator is updated to better infer the skill z from states visited. Entropy regularization occurs as part of the SAC update.

Stability

Unlike prior adversarial unsupervised RL methods (e.g., [209]), DIAYN forms a cooperative game, which avoids many of the instabilities of adversarial saddle-point formulations. On gridworlds,

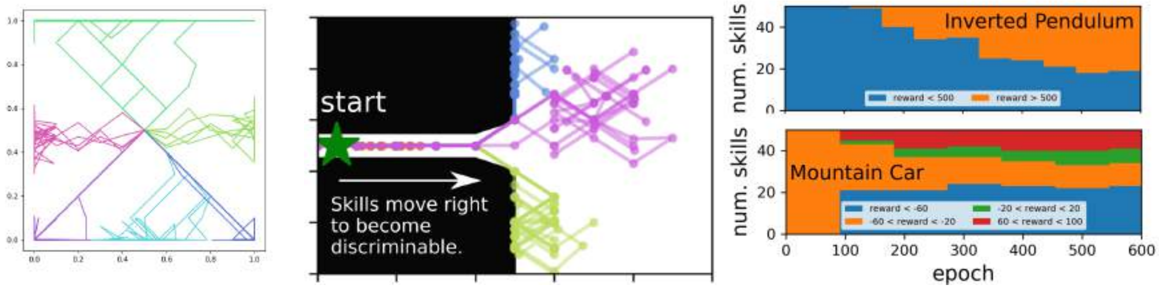
[2] While our method uses stochastic policies, note that for deterministic policies in continuous action spaces, $I(A; Z \mid S) = \mathcal{H}[A \mid S]$. Thus, for deterministic policies, Equation 5.2 reduces to maximizing $I(S; Z)$.

we can compute analytically that the unique optimum to the DIAYN optimization problem is to evenly partition the states between skills, with each skill assuming a uniform stationary distribution over its partition (proof in Appendix 18.3). In the continuous and approximate setting, convergence guarantees would be desirable, but this is a very tall order: even standard RL methods with function approximation (e.g., DQN) lack convergence guarantees, yet such techniques are still useful. Empirically, we find DIAYN to be robust to random seed; varying the random seed does not noticeably affect the skills learned, and has little effect on downstream tasks (see Fig.s 5.4, 5.5, and 18.19).

5.4 Experiments

In this section, we evaluate DIAYN and compare to prior work. First, we analyze the skills themselves, providing intuition for the types of skills learned, the training dynamics, and how we avoid problematic behavior in previous work. In the second half, we show how the skills can be used for downstream tasks, via policy initialization, hierarchy, imitation, outperforming competitive baselines on most tasks. We encourage readers to view videos[3] and code[4] for our experiments.

Analysis of Learned Skills



(a) 2D Navigation (b) Overlapping Skills (c) Training Dynamics

Figure 5.2: *(Left)* DIAYN skills in a simple navigation environment; *(Center)* skills can overlap if they eventually become distinguishable; *(Right)* diversity of the rewards increases throughout training.

Question 1. *What skills does DIAYN learn?*

We study the skills learned by DIAYN on tasks of increasing complexity, ranging from point navigation (2 dimensions) to ant locomotion (111 dimensions). We first applied DIAYN to a simple 2D navigation environment. The agent starts in the center of the box, and can take actions to directly move its (x, y) position. Figure 5.2a illustrates how the 6 skills learned for this task move away from each other to remain distinguishable. Next, we applied DIAYN to two classic control tasks,

[3] `https://sites.google.com/view/diayn/`

[4] `https://github.com/ben-eysenbach/sac/blob/master/DIAYN.md`

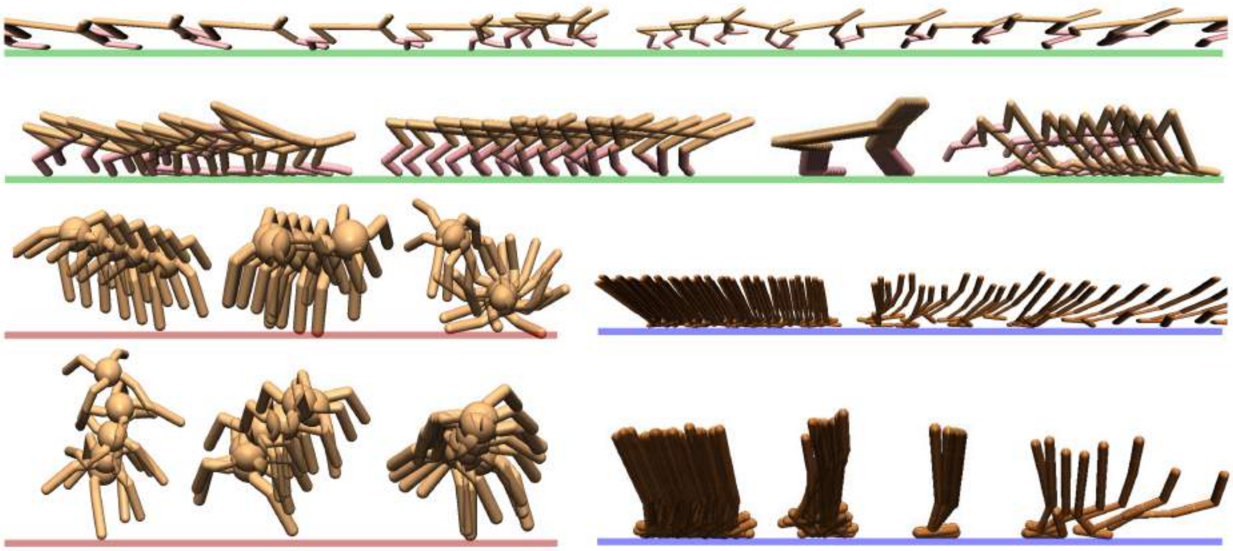


Figure 5.3: **Locomotion skills**: Without any reward, DIAYN discovers skills for running, walking, hopping, flipping, and gliding. It is challenging to craft reward functions that elicit these behaviors.

inverted pendulum and mountain car. Not only does our approach learn skills that solve the task without rewards, it learns multiple distinct skills for solving the task. (See Appendix 18.3 for further analysis.)

Finally, we applied DIAYN to three continuous control tasks [29]: half cheetah, hopper, and ant. As shown in Figure 5.3, we learn a diverse set of primitive behaviors for all tasks. For half cheetah, we learn skills for running forwards and backwards at various speeds, as well as skills for doing flips and falling over; ant learns skills for jumping and walking in many types of curved trajectories (though none walk in a straight line); hopper learns skills for balancing, hopping forward and backwards, and diving. See Appendix 18.3 for a comparison with VIME.

Question 2. *How does the distribution of skills change during training?*

While DIAYN learns skills without a reward function, as an outside observer, can we evaluate the skills throughout training to understand the training dynamics. Figure 5.2 shows how the skills for inverted pendulum and mountain car become increasingly diverse throughout training (Fig. 18.19 repeats this experiment for 5 random seeds, and shows that results are robust to initialization). Recall that our skills are learned with no reward, so it is natural that some skills correspond to small task reward while others correspond to large task reward.

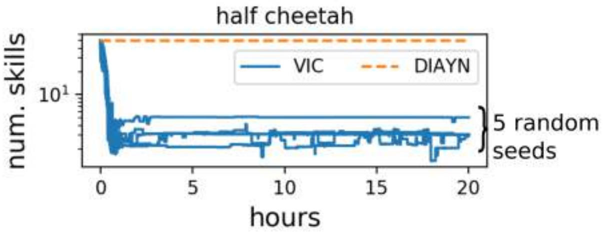


Figure 5.4: **Why use a fixed prior?** In contrast to prior work, DIAYN continues to sample all skills throughout training.

Question 3. *Does discriminating on single states restrict DIAYN to learn skills that visit disjoint sets of states?*

Our discriminator operates at the level of states, not trajectories. While DIAYN favors skills that do not overlap, our method is not limited to learning skills that visit entirely disjoint sets of states. Figure 5.2b shows a simple experiment illustrating this. The agent starts in a hallway (green star), and can move more freely once exiting the end of the hallway into a large room. Because RL agents are incentivized to maximize their cumulative reward, they may take actions that initially give no reward to reach states that eventually give high reward. In this environment, DIAYN learns skills that exit the hallway to make them mutually distinguishable.

Question 4. *How does DIAYN differ from Variational Intrinsic Control (VIC) [194]?*

The key difference from the most similar prior work on unsupervised skill discovery, VIC, is our decision to *not* learn the prior $p(z)$. We found that VIC suffers from the "Matthew Effect" [210]: VIC's learned prior $p(z)$ will sample the more diverse skills more frequently, and hence only those skills will receive training signal to improve. To study this, we evaluated DIAYN and VIC on the half-cheetah environment, and plotting the effective number of skills (measured as $\exp(\mathcal{H}[Z])$) throughout training (details and more figures in Appendix 18.3). The figure to the right shows how VIC quickly converges to a setting where it only samples a handful of skills. In contrast, DIAYN fixes the distribution over skills, which allows us to discover more diverse skills.

Harnessing Learned Skills

The perhaps surprising finding that we can discover diverse skills without a reward function creates a building block for many problems in RL. For example, to find a policy that achieves a high reward on a task, it is often sufficient to simply choose the skill with largest reward. Three less obvious applications are adapting skills to maximize a reward, hierarchical RL, and imitation learning.

Using Skills for Hierarchical RL

In theory, hierarchical RL should decompose a complex task into motion primitives, which may be reused for multiple tasks. In practice, algorithms for hierarchical RL can encounter many problems: (1) each motion primitive reduces to a single action [181], (2) the hierarchical policy only samples a single motion primitive [194], or (3) all motion primitives attempt to do the entire task. In contrast, DIAYN discovers diverse, *task-agnostic* skills, which hold the promise of acting as a building block for hierarchical RL.

Question 5. *Are skills discovered by DIAYN useful for hierarchical RL?*

We propose a simple extension to DIAYN for hierarchical RL, and find that simple algorithm outperforms competitive baselines on two challenging tasks. To use the discovered skills for hierarchical RL, we learn a meta-controller whose actions are to choose which skill to execute for the next k steps (100 for ant navigation, 10 for cheetah hurdle). The meta-controller has the same observation space as the skills.

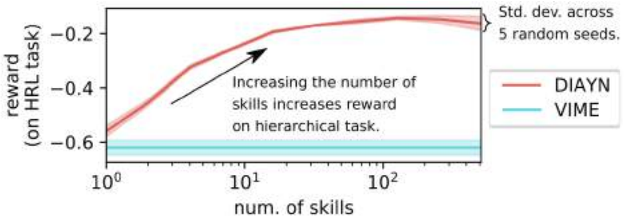


Figure 5.5: **Hierarchical RL**

As an initial test, we applied the hierarchical RL algorithm to a simple 2D point navigation task (details in Appendix 18.3). Figure 5.5 illustrates how the reward on this task increases with the number of skills; error bars show the standard deviation across 5 random seeds. To ensure that our goals were not cherry picked, we sampled 25 goals evenly from the state space, and evaluated each random seed on all goals. We also compared to Variational Information Maximizing Exploration (VIME) [105]. Note that even the best random seed from VIME significantly under-performs DIAYN. This is not surprising: whereas DIAYN learns a set of skills that effectively partition the state space, VIME attempts to learn a single policy that visits many states.

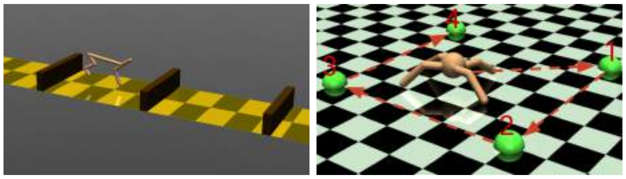


Figure 5.6: Challenging tasks for hierarchical RL: *(Left)* Cheetah Hurdle; *(Right)* Ant Navigation

Next, we applied the hierarchical algorithm to two challenging simulated robotics environment. On the cheetah hurdle task, the agent is rewarded for bounding up and over hurdles, while in the ant navigation task, the agent must walk to a set of 5 waypoints in a specific order, receiving only a sparse reward upon reaching each waypoint. The sparse reward and obstacles in these environments make them exceedingly difficult for non-hierarchical RL algorithms. Indeed, state of the art RL algorithms that do not use hierarchies perform poorly on these tasks. Figure 5.7 shows how DIAYN outperforms state of the art on-policy RL (TRPO [90]), off-policy RL (SAC [23]), and exploration bonuses (VIME). This experiment suggests that unsupervised skill learning provides an effective mechanism for combating challenges of exploration and sparse rewards in RL.

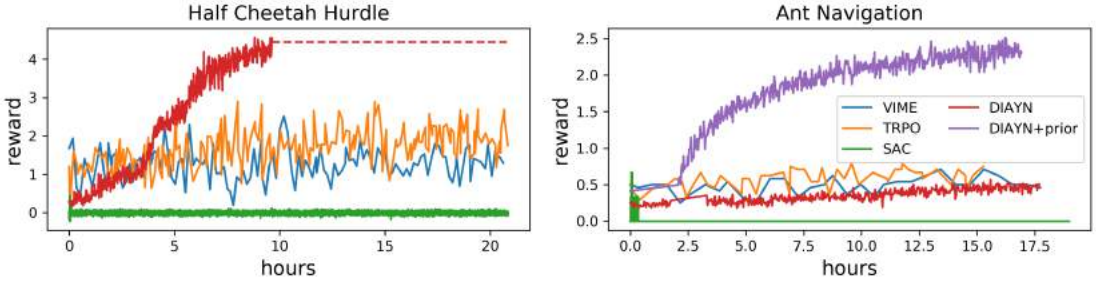


Figure 5.7: **DIAYN for Hierarchical RL**: By learning a meta-controller to compose skills learned by DIAYN, cheetah quickly learns to jump over hurdles and ant solves a sparse-reward navigation task.

Question 6. *How can DIAYN leverage prior knowledge about what skills will be useful?*

If the number of possible skills grows exponentially with the dimension of the task observation, one might imagine that DIAYN would fail to learn skills necessary to solve some tasks. While we found that DIAYN *does* scale to tasks with more than 100 dimensions (ant has 111), we can also use a simple modification to bias DIAYN towards discovering particular types of skills. We can condition the discriminator on only a subset of the observation space, or any other function of the observations. In this case, the discriminator maximizes $\mathbb{E}[\log q_\phi(z \mid f(s))]$. For example, in the ant navigation task, $f(s)$ could compute the agent's center of mass, and DIAYN would learn skills that correspond to changing the center of mass. The "DIAYN+prior" result in Figure 5.7 (right) shows how incorporating this prior knowledge can aid DIAYN in discovering useful skills and boost performance on the hierarchical task. (No other experiments or figures in this paper used this prior.) The key takeaway is that while DIAYN is primarily an unsupervised RL algorithm, there is a simple mechanism for incorporating supervision when it is available. Unsurprisingly, we perform better on hierarchical tasks when incorporating more supervision.

Imitating an Expert

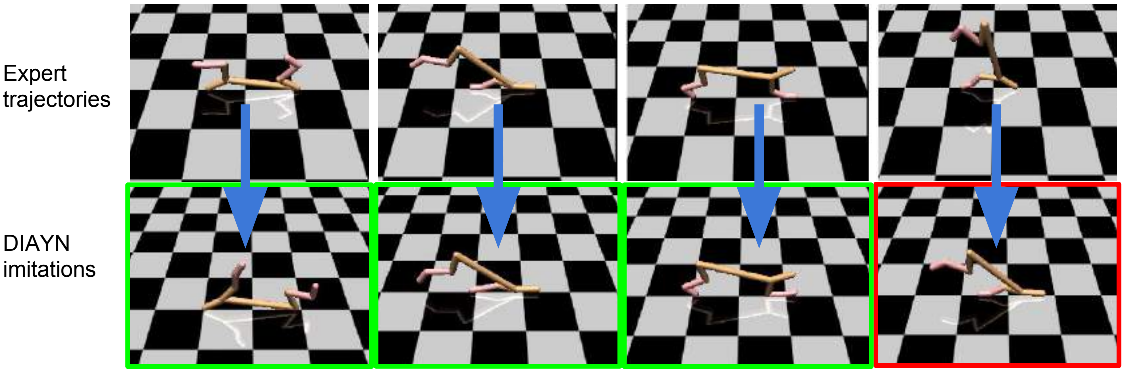


Figure 5.8: **Imitating an expert**: DIAYN imitates an expert standing upright, flipping, and faceplanting, but fails to imitate a handstand.

Question 7. *Can we use learned skills to imitate an expert?*

Aside from maximizing reward with finetuning and hierarchical RL, we can also use learned skills to follow expert demonstrations. One use-case is where a human manually controls the agent to complete a task that we would like to automate. Simply replaying the human's actions fails in stochastic environments, cases where closed-loop control is necessary. A second use-case involves an existing agent with a hard coded, manually designed policy. Imitation learning replaces the existing policy with a similar yet differentiable policy, which might be easier to update in response to new constraints or objectives. We consider the setting where we are given an expert trajectory consisting of states, without actions, defined as $\tau^* = \{(s_i)\}_{1 \le i \le N}$. Our goal is to obtain a feedback controller that will reach the same states. Given the expert trajectory, we use our learned discriminator to estimate which skill was most likely to have generated the trajectory. This optimization problem, which we solve for categorical z by enumeration, is equivalent to an M-projection [211]:

$$\hat{z} = \arg\max_z \Pi_{s_t \in \tau^*} q_\phi(z \mid s_t)$$

We qualitatively evaluate this approach to imitation learning on half cheetah. Figure 5.8 (left) shows four imitation tasks, three of which our method successfully imitates. We quantitatively evaluate this imitation method on classic control tasks in Appendix 18.3.

Accelerating Learning with Policy Initialization

Given that the skills learned by DIAYN are often meaningful and semantically interesting, it stands to reason that these skills can also help accelerate the learning of skills when a reward function *is* actually provided. In fact, we will show how DIAYN can be incorporated into a broader meta-learning framework for this very purpose in the following chapter, providing an effective way of pre-training RL agents for rapid skill acquisition.

5.5 Conclusion

In this chapter, we present DIAYN, a method for learning skills without reward functions. We show that DIAYN learns diverse skills for complex tasks, often solving benchmark tasks with one of the learned skills without actually receiving any task reward. We further proposed methods for using the learned skills (1) to solve complex tasks via hierarchical RL, and (2) to imitate an expert, and as we will show next, can also accelerate the learning of new tasks.

Chapter 6

Unsupervised Pre-Training for Quick Reinforcement Learning

In the previous chapter, we showed how we can learn skills without *any* reward functions, often resulting in semantically meaningful and interesting behaviors. However, these behaviors are still task-agnostic and not quite geared towards solving a specific task that a human may want to solve. In order to actually accomplish goal directed behaviors, the unsupervised pre-training resulting from a technique like DIAYN must be combined with a task directed finetuning scheme with rewards specified using a technique like the ones described in Section 3, 4. In this chapter, we show that the ideas from unsupervised skill discovery can be combined with the framework of meta-reinforcement learning to enable very quick learning of new tasks. In this way, we show that we can design a general purpose procedure for pre-training RL algorithms without any human defined reward functions. We start by motivating this idea from first principles.

6.1 Motivating a General Unsupervised Meta-RL Framework

Reusing past experience for faster learning of new tasks is a key challenge for machine learning. Meta-learning methods achieve this by using past experience to explicitly optimize for rapid adaptation [140], [163], [165], [169], [212]–[214]. In the context of reinforcement learning (RL), meta-reinforcement learning (meta-RL) algorithms can learn to solve new RL tasks more quickly through experience on past tasks [140], [166], [213]. Typical meta-RL algorithms assume the ability to sample from a pre-specified task distribution, and these algorithms learn to solve new tasks *drawn from this distribution* very quickly. However, specifying a task distribution is tedious and requires a significant amount of supervision [79], [166] that may be difficult to provide for large, real-world problem settings. The performance of meta-learning algorithms critically depends on the meta-training task distribution, and meta-learning algorithms generalize best to new tasks which are drawn from the same distribution as the meta-training tasks [140]. In effect, meta-RL offloads the design burden from algorithm design to task design. While meta-RL acquires representations for fast adaptation to the specified task distribution, specifying this task distribution is often tedious and challenging. Can we automate the process of task design, thereby doing away with human supervision entirely?

In this chapter, we take a step towards *unsupervised* meta-RL: meta-learning from a task distribution that is acquired automatically, rather than requiring manual design of the meta-training tasks. While unsupervised meta-RL does not make any assumptions about the reward functions on which it will be evaluated at test time, it does assume that the environment dynamics remain the same. This allows an unsupervised meta-RL agent to utilize environment interactions to meta-train a model that is optimized to be effective for learning from previously unseen reward functions in that environment at meta-test time. Our method can also be thought of as automatically acquiring an *environment-specific learning procedure* for deep neural network policies, somewhat related to data-driven initialization procedures explored in supervised learning [215], [216].

The primary contribution of our work is a framework for unsupervised meta-RL. We describe a family of unsupervised meta-RL algorithms and provide analysis to show that unsupervised meta-RL methods based on mutual information can be optimal, in a minimax sense. Our experiments shows that, for a variety of robotic control tasks, unsupervised meta-RL can effectively acquire RL procedures. These procedures not only learn faster than standard RL approaches that learn from scratch, but also outperform prior methods that do pure exploration and then fine-tuning at test time. Our results even approach the performance of an oracle method that relies on hand-designed task distributions.

6.2 Related Work

Our work lies at the intersection of meta-RL, goal generation, and unsupervised exploration. Meta-learning algorithms use data from multiple tasks to learn how to learn, acquiring rapid adaptation procedures from experience [140], [165], [170], [217]–[223]. These approaches have been extended into the setting of RL [140], [163], [166], [168], [172], [213], [224]–[227]. In practice, the performance of meta-learning algorithms depends on the user-specified meta-training task distribution. We aim to lift this limitation and provide a general recipe for avoiding manual task engineering for meta-RL. A handful of prior meta-learning methods have used self-proposed task distributions for learning supervised learning procedures [216], [228]–[230]. In contrast, our work deals with the RL setting, where the environment dynamics provides a rich inductive bias that our meta-learner can exploit. In the RL setting, task distributions can be obtained in a variety of ways, including adversarial goal generation [209], [231], information-theoretic methods [194], [207], [232], [233]. The most similar work is [234], which also considers the unsupervised application of meta-learning to RL tasks. We build upon this work by proving that an optimal meta-learner can be acquired using mutual information-based task proposal.

Exploration methods that seek out novel states are also closely related to goal generation methods [106], [108], [109], [235], but do not by themselves aim to generate new tasks or learn to adapt more quickly to new tasks, only to achieve wide coverage of the state space. Model-based RL methods [24], [26], [236]–[239] use unsupervised experience to learn a dynamics model but do not learn how to efficiently use this model to explore to solve new tasks.

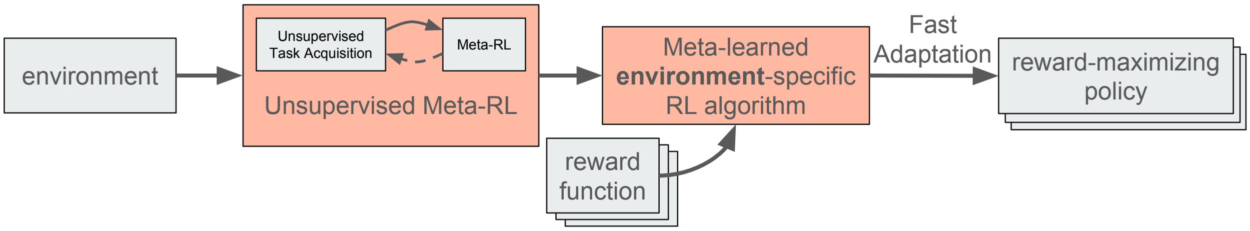


Figure 6.1: **Unsupervised meta-reinforcement learning**: Given an environment, unsupervised meta-RL produces an environment-specific learning algorithm that quickly acquire new policies that maximize any task reward function.

Goal-conditioned RL [118], [119], [240] is also related to our work, and our analysis will study this special case first before generalizing to the general case of arbitrary tasks. As we discuss in Section 6.3, goal-reaching itself is not enough, as goal-reaching agents are not optimized to efficiently explore to determine which goal they should reach, relying instead on a hand-specified goal parameterization that doesn't allow these algorithms to work with arbitrary reward functions.

6.3 Unsupervised Meta-RL

We consider the problem of *learning* a reinforcement learning algorithm that can quickly solve new tasks in a given environment. This meta-RL process could, for example, tune the hyperparameters of another RL algorithm, or could replace the RL update rule itself with a learned update rule. Unlike prior work, we aim to do so without depending on any human supervision or information about the tasks that will be provided for meta-testing. A task reward is provided at meta-test time, and the learned RL procedure should adapt to this task reward as quickly as possible. We assume that all test-time tasks have the same dynamics, and differ only in their reward functions. Our algorithm will therefore need to utilize unsupervised environment interaction to learn an RL algorithm. *In effect, the dynamics themselves will be the supervision for our learning algorithm.*

We formalize the meta-training setting as a controlled Markov process (CMP) – a Markov decision process without a reward function, $C = (S, A, P, \gamma, \rho)$, with state space S, action space A, transition dynamics P, discount factor γ and initial state distribution ρ. The CMP, along with a reward function r, produces a Markov decision processes $M = (S, A, P, \gamma, \rho, r)$. We define a learning algorithm $f : \mathcal{D} \to \pi$ as a function that takes as input a dataset of experience from the MDP, $\mathcal{D} = \{(s_i, a_i, r_i, s_i')\} \sim M$, and outputs a policy $\pi(a \mid s)$. Evaluation of the learning procedure f is carried out over a handful of episodes. In episode i, the learning procedure f observes all previous data $\{\tau_1, \cdots, \tau_{i-1}\}$ and outputs a policy to be used in iteration i. We evaluate the learning procedure f by summing its cumulative reward across iterations:

$$R(f, r_z) = \sum_i \mathbb{E}_{\substack{\pi = f(\{\tau_1, \cdots, \tau_{i-1}\}) \\ \tau \sim \pi}} \left[\sum_t r_z(s_t, a_t) \right]$$

Our aim is to take this CMP and produce an environment-specific learning algorithm f that can quickly learn an optimal policy $\pi_r^*(a \mid s)$ for *any* reward function r. We refer to this problem as *unsupervised meta-RL*, and illustrate the problem setting in Fig. 6.1.

We now sketch a recipe for unsupervised meta-RL, analyze when this recipe is optimal, and then instantiate a practical approximation to this theoretically-motivated approach by building upon known meta-learning algorithms and unsupervised exploration methods.

A General Recipe

To construct an unsupervised meta-RL algorithm, we leverage the insight that, to acquire a fast learning algorithm without task supervision, we can simply leverage standard meta-learning techniques, but with unsupervised task proposal mechanisms. Our unsupervised meta-RL framework therefore consists of a task proposal mechanism and a meta-learning method. For reasons that will become more apparent later, we will define the task distribution as a mapping from a latent variable $z \sim p(z)$ to a reward function $r_z(s, a) : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$[1]. That is, for each value of the random variable z, we have a different reward function $r_z(s, a)$. Under this formulation, learning a task distribution amounts to optimizing a parametric form for the reward function $r_z(s, a)$ that maps each $z \sim p(z)$ to a different reward function. The choice of this parametric form represents an important design decision for an unsupervised meta-learning method, and the resulting set of tasks is often referred to as a task or goal proposal procedure. In the following section, we will discuss a theoretical framework that allows us to make this choice in the following section so as to minimize worst case regret of the subsequently meta-learned learning algorithm f.

The second component is the meta-learning algorithm, which takes the family of reward functions induced by $p(z)$ and $r_z(s, a)$, along with the associated CMP, and meta-learns an RL algorithm f that can quickly adapt to any task from the task distribution defined by $p(z)$ and $r_z(s, a)$ in the given CMP. The meta-learned algorithm f can then learn new tasks quickly at meta-test time, when a user-specified reward function is actually provided. Fig. 6.1 summarizes this generic design for an unsupervised meta-RL algorithm.

The "no free lunch theorem" [241], [242] might lead us to expect that a truly generic approach to proposing a task distribution would not yield a learning procedure f that is effective on any real tasks. However, *the assumption that the dynamics remain the same across tasks affords us an inductive bias with which we pay for our lunch.* In the following sections, we will discuss how to formulate acquiring the optimal unsupervised learning procedure, which minimizes regret on new meta-test tasks in the absence of any prior knowledge. Since our analysis will focus on a restricted class of learning procedures, our results are lower bounds for the performance of general learning procedures. We first define an optimal meta-learner and then show how we can train one without requiring task distributions to be hand-specified.

Optimal Meta-Learners

We begin our analysis by considering the optimal learning procedure when the task distribution is known. For a task distribution $p(r_z)$, the optimal learning procedure f^* is given by

$$f^* \triangleq \arg\max_f \mathbb{E}_{p(r_z)} \left[R(f, r_z) \right].$$

Other learning procedures f may achieve lower reward, and we define the regret incurred by using a suboptimal learning procedure as the difference in expected reward, compared with the optimal learning procedure:

$$\text{REGRET}(f, p(r_z)) \triangleq \mathbb{E}_{p(r_z)} \left[R(f^*, r_z) \right] - \mathbb{E}_{p(r_z)} \left[R(f, r_z) \right].$$

[1] In most cases $p(z)$ is chosen to be a uniform categorical so it is not challenging to specify

Minimizing this regret is equivalent to maximizing the expected reward objective used by most meta-RL methods [140], [166]. Note that different task distributions $p(r_z)$ will have different optimal learning procedures f^*. For example, the optimal behavior for manipulation tasks involves moving a robot's arms, while the optimal behavior for locomotion tasks involves moving a robot's legs. Therefore, f^* depends on $p(r_z)$. We next define the notion of an optimal *unsupervised* meta-learner, which does not require prior knowledge of $p(r_z)$.

In unsupervised meta-reinforcement learning, the reward distribution $p(r_z)$ is unknown. In this setting, we evaluate a learning procedure f based on its regret against the worst-case task distribution for CMP C:

$$\text{R}\textsc{egret}_{\text{WC}}(f, C) = \max_{p(r_z)} \text{R}\textsc{egret}(f, p(r_z)). \tag{6.1}$$

For a CMP C, we define the optimal unsupervised learning procedure as follows:

Definition 1. The optimal unsupervised learning procedure f_C^* for a CMP C is defined as

$$f_C^* \triangleq \arg\min_f \text{R}\textsc{egret}_{\text{WC}}(f, C).$$

Note the optimal unsupervised learning procedure may be different for different CMPs. We can also define the optimal unsupervised *meta-learning* algorithm $\mathcal{F}*$, which takes as input a CMP C and returns the optimal unsupervised learning procedure f_C^* for that CMP:

Definition 2. The optimal unsupervised meta-learner $\mathcal{F}^*(C) = f_C^*$ is a function that takes as input a CMP C and outputs the corresponding optimal unsupervised learning procedure f_C^*:

$$\mathcal{F}^* \triangleq \arg\min_{\mathcal{F}} \text{R}\textsc{egret}_{\text{WC}}(\mathcal{F}(C), C)$$

Note that the optimal unsupervised meta-learner $\mathcal{F}^*$ is universal – it does not depend on any particular task distribution, or any particular CMP. The next sections discuss how to find the minimax learning procedure, which minimizes the worst-case regret (Eq. 6.1).

Special Case: Goal-Reaching Tasks

We start by deriving an optimal unsupervised meta-learner for the special case where all tasks are assumed to be goal state reaching tasks, and then generalize this approach to solve arbitrary tasks in Section 6.3. We restrict our analysis to CMPs with deterministic dynamics, and consider episodes with finite horizon T and a discount factor of $\gamma = 1$. Each tasks corresponds to reaching a goal states s_g at the last time step in the episode, so the reward function is

$$r_g(s_t) \triangleq \mathbb{1}(t = T) \cdot \mathbb{1}(s_t = g).$$

We first derive the optimal learning procedure for the case where $p(s_g)$ is known, and then derive the optimal procedure for the case where $p(s_g)$ is unknown.

The Optimal Learning Procedure for Known $p(s_g)$

In the case of goal reaching tasks, the optimal fast learning procedure f searches through potential goal states until it finds the goal and then navigates to that goal state in all subsequent episodes. Define f_π as the learning procedure that uses policy π to explore until the goal is found, and then always returns to the goal state. We will restrict our attention to the set of learning procedures $\mathcal{F}_\pi \triangleq \{f_\pi\}$ constructed in this fashion, so our theoretical results will be lower bound on the performance of arbitrary learning procedures. The learning procedure f_π incurs one unit of regret for each step before it has found the goal, and zero regret afterwards. The expected cumulative regret is therefore the expectation of the hitting time. To compute the expected hitting time, we define $\rho_\pi^T(s)$ as the probability that policy π visits state s at time step $t = T$. If s_g is the true goal, then the event that the policy π reaches s_g at the final step of an episode is a Bernoulli random variable with parameter $p = \rho_\pi^T(s_g)$. Thus, the expected hitting time of this goal state is

$$\textsc{HittingTime}_\pi(s_g) = \frac{1}{\rho_\pi^T(s_g)}.$$

The regret of the learning procedure f_π is

$$\textsc{Regret}(f_\pi, p(r_g)) \quad = \quad \int \textsc{HittingTime}_\pi(s_g)p(s_g)ds_g \quad = \quad \int \frac{p(s_g)}{\rho_\pi^T(s_g)}ds_g. \tag{6.2}$$

To now compute the *optimal* learning procedure f_π, we can minimize the regret in Equation 6.2 w.r.t. the marginal distribution ρ_π^T. Using the calculus of variations (for more details refer to Appendix C in [243]), the exploration policy for the optimal meta-learner, π^*, satisfies:

$$\rho_{\pi^*}^T(s_g) = \frac{\sqrt{p(s_g)}}{\int \sqrt{p(s_g')ds_g'}}. \tag{6.3}$$

Thus, when the goal sampling distribution $p(s_g)$ is known, the optimal learning procedure is obtained by finding π^* satisfying Eq. 6.3 and then using f_{π^*} as the learning procedure. The next section considers the case where $p(s_g)$ is not known.

The Optimal Unsupervised Learning Procedure for Goal Reaching Tasks

In the case of goal-reaching tasks where the goal distribution $p(s_g)$ is not known, the optimal unsupervised learning procedure can be constructed from a policy with a uniform marginal state distribution (proof in Appendix 18.4):

Lemma 2. Let π be a policy for which $\rho_\pi^T(s)$ is uniform. Then f_π is has lowest worst-case regret among learning procedures in $\mathcal{F}_\pi$.

One route for constructing this optimal unsupervised learning procedure is to first acquire a policy π for which $\rho_\pi^T(s)$ is uniform and then return f_π. However, finding such a policy π is challenging, especially in high-dimensional state spaces and in the absense of resets. Instead, we will take an alternate route, acquiring f_π directly without every computing π. In addition to sidestepping the requirement of computing π, this approach will also have the benefit of generalizing beyond goal-reaching tasks to arbitrary task distributions.

Our approach for directly computing the optimal unsupervised learning procedure hinges on the observation that the optimal unsupervised learning procedure is the optimal (supervised) learning procedure for goals proposed from a uniform distribution. Thus, the optimal unsupervised learning procedure will come not as a result of a careful construction, but rather as the output of the an optimization procedure (i.e., meta-learning). Thus, we can obtain the optimal unsupervised learning procedure by applying a meta-learning algorithm to a task distribution that samples goals uniformly. To ensure that the resulting learning procedure f lies within the set $\mathcal{F}_\pi$, we will only consider "memoryless" meta-learning algorithms that maintain no internal state before the true goal is found.[2] While sampling goals uniform is itself a challenging problem, we can use the same trick as before: instead of constructing this uniform goal distribution directly, we instead find an optimization problem for which the solution is to sample goals uniformly.

The optimization problem that we use will involve two latent variables, the final state s_T and an auxiliary latent variable z sampled from a prior $\mu(z)$. The optimization problem will be to find a conditional distribution $\mu(s_T \mid z)$ such that the mutual information between z and s_T is optimized:

$$\max_{\mu(s_T \mid z)} I_\mu(s_T; z) \tag{6.4}$$

The conditional distribution $\mu(s_T \mid z)$ that optimizes Equation 6.4 is one with a uniform marginal distribution over terminal states (proof in Appendix 18.4):

Lemma 3. Assume there exists a conditional distribution $\mu(s_T \mid z)$ satisfying the following two properties:

topsep=1pt,1temsep=1pt The marginal distribution over terminal states is uniform: $\mu(s_T) = \int \mu(s_T \mid z)\mu(z)dz = \text{UNIF}(\mathcal{S})$; and

topsep=2pt,2temsep=2pt The conditional distribution $\mu(s_T \mid z)$ is a Dirac: $\forall z, s_T\ \exists s_z$ s.t. $\mu(s_T \mid z) = \mathbb{1}(s_T = s_z)$.

Then any solution $\mu(s_T \mid z)$ to the mutual information objective (Eq. 6.4) satisfies the following:

$$\mu(s_T) = \text{UNIF}(\mathcal{S}) \quad \text{and} \quad \mu(s_T \mid z) = \mathbb{1}(s_T = s_z).$$

Optimizing Mutual Information

To optimize the above mutual information objective, we note that a conditional distribution $\mu(s_T \mid z)$ can be defined implicitly via a latent-conditioned policy $\mu(a \mid s, z)$. This policy is *not* a meta-learned

[2] MAML satisfies this requirement, as the internal parameters are updated by policy gradient, which is zero because the reward is zero before the true goal is found.

model, but rather will become part of the task proposal mechanism. For a given prior $\mu(z)$ and latent-conditioned policy $\mu(a \mid s, z)$, the joint likelihood is

$$\mu(\tau, z) = \mu(z)p(s_1)\prod_t p(s_{t+1} \mid s_t, a_t)\mu(a_t \mid s_t, z),$$

and the marginal likelihood is simply given by

$$\mu(s_T, z) = \int \mu(\tau, z) ds_1 a_1 \cdots a_{T-1}.$$

The purpose of our repeated indirection now becomes clear: prior work [207], [232] has proposed efficient algorithms for maximizing the mutual information objective (Eq. 6.4) when the conditional distribution $\mu(s_T \mid z)$ is defined implicitly in terms of a latent-conditioned policy. At this point, we finally can sample goals uniformly, by sampling $z \sim \mu(z)$ followed by $s_T \sim \mu(s_T \mid z)$.

Recall that we wanted to obtain a uniform goal distribution so that we could apply meta-learning to obtain the optimal learning procedure. However, the input to meta-learning procedures is not a distribution over goals but a distribution over reward functions. We then define our task proposal distribution $p(r_z)$ by sampling $z \sim p(z)$ and using the corresponding reward function $r_z(s_T, a_T) \triangleq \log p(s_T \mid z)$, resulting in a uniform distribution as described in Lemma 2.

General Case: Trajectory-Matching Tasks

To extend the analysis in the previous section to the general case, and thereby derive a framework for optimal unsupervised meta-learning, we will consider "trajectory-matching" tasks. These tasks are a trajectory-based generalization of goal reaching: while goal reaching tasks only provide a positive reward when the policy reaches the goal state, trajectory-matching tasks only provide a positive reward when the policy executes the optimal trajectory. The trajectory matching case is more general because, while trajectory matching can represent different goal-reaching tasks, it can also represent tasks that are not simply goal reaching, such as reaching a goal while avoiding a dangerous region or reaching a goal in a particular way. Moreover, the trajectory matching case is actually also a generalization of the typical reinforcement learning case with Markovian rewards, because any such task can be represented by a trajectory reaching objective as well. Please refer to Section 6.3 for a more complete discussion of the same.

As before, we will restrict our attention to CMPs with deterministic dynamics. These non-Markovian tasks essentially amount to a problem where an RL algorithm must "guess" the optimal policy, and only receives a reward if its behavior is perfectly consistent with that optimal policy.

We will show that optimizing the mutual information between z and *trajectories* to obtain a task proposal distribution, and subsequently optimizing a meta-learner for this distribution will give us the optimal unsupervised meta-learner for this class of reward functions. We subsequently show that unsupervised meta-learning for the trajectory-matching task is at least as hard as unsupervised meta-learning for general tasks. As before, let us begin within an analysis of optimal meta-learners in the case where the distribution over trajectory matching tasks $p(\tau^*)$ is known, and subsequently direct our attention to formulating an optimal unsupervised meta-learner.

Optimal meta-learner for known $p(\tau^*)$

Formally, we define a distribution of trajectory-matching tasks by a distribution over desired trajectories, $p(\tau^*)$. For each goal trajectory τ^*, the corresponding trajectory-level reward function is

$$r^*_\tau(\tau) \triangleq \mathbb{1}(\tau = \tau^*)$$

Analysis from Section 6.3 can be repurposed here. As before, restrict our attention to learning procedures $f_\pi \in \mathcal{F}_\pi$. After running the exploration policy to discover trajectories that obtain reward, the policy will deterministically keep executing the desired trajectory. We can define the hitting time as the expected number of episodes to match the target trajectory:

$$\textsc{HittingTime}_\pi(\tau^*) = \frac{1}{\pi(\tau^*)}$$

We then define regret as the expected hitting time:

$$\textsc{Regret}(f_\pi, p(r_\tau)) \quad = \quad \int \textsc{HittingTime}_\pi(\tau) p(\tau) d\tau \quad = \quad \int \frac{p(\tau)}{\pi(\tau)} d\tau. \quad (6.5)$$

This definition of regret allows us to optimize for an optimal learning procedure, and we obtain an exploration policy for the optimal learning procedure satisfying the requirement

$$\pi^*(\tau) = \frac{\sqrt{p(\tau)}}{\int \sqrt{p(\tau')} d\tau'}.$$

Optimal unsupervised learning procedure for trajectory-matching tasks

As described in Section 6.3, obtaining such a policy requires knowing the trajectory distribution $p(\tau)$, and we must resort to optimizing the worst-case regret. As argued in Lemma 1, the solution to this min-max optimization is a learning procedure which has an exploration policy that is uniform distribution over trajectories.

Lemma 4. Let π be a policy for which $\pi(\tau)$ is uniform. Then f_π has lowest worst-case regret among learning procedures in $\mathcal{F}_\pi$.

We can acquire an unsupervised meta-learner of this form by proposing and meta-learning on a task distribution that is uniform over trajectories. How might we actually propose a task distribution that is uniform over trajectories? As argued for the goal reaching case, we can do so by optimizing a *trajectory-level* mutual information objective:

$$I(\tau; z) = \mathcal{H}[\tau] - \mathcal{H}[\tau \mid z]$$

The optimal policy for this objective has a uniform distribution over trajectories that, conditioned on a particular latent z, deterministically produces a single trajectory in a deterministic CMP. The analysis for the case of stochastic dynamics is more involved and is left to future work. By optimizing a task proposal distribution that maximizes trajectory-level mutual information, and subsequently performing meta-learning on the proposed tasks, we can acquire the optimal unsupervised meta-learner for trajectory matching tasks, under the definition in Section 6.3.

Relationship to General Reward Maximizing Tasks

Now that we have derived the optimal meta-learner for trajectory-matching tasks, we observe that trajectory-matching is a super-set of the problem of optimizing any possible Markovian reward function at test-time. For a given initial state distribution, each reward function is optimized by a particular trajectory. However, trajectories produced by a non-Markovian policy (i.e., a policy with memory) are not necessarily the unique optimum for any Markovian reward function. Let R_τ denote the set of trajectory-level reward functions, and $R_{s,a}$ denote the set of all state-action level reward functions. Bounding the worst-case regret on R_τ minimizes an upper bound on the worst-case regret on $R_{s,a}$:

$$\min_{r_\tau \in R_\tau} \mathbb{E}_\pi \left[r_\tau(\tau) \right] \leq \min_{r \in R_{s,a}} \mathbb{E}_\pi \left[\sum_t r(s_t, a_t) \right] \quad \forall \pi.$$

This inequality holds for all policies π, including the policy that maximizes the LHS. While we aim to maximize the RHS, we only know how to maximize the LHS, which gives us a lower bound on the RHS. This inequality holds for all policies π, so it also holds for the policy that maximizes the LHS. In general, this bound is loose, because the set of all Markovian reward functions is smaller than the set of all trajectory-level reward functions (i.e., trajectory-matching tasks). However, this bound becomes tight when considering meta-learning on the set of all possible (non-Markovian) reward functions.

In the discussion of meta-learning thus far, we have restricted our attention to tasks where the reward is provided at the last time step T of each episode and to the set of learning procedures $\mathcal{F}_\pi$ that maintain no internal state before the true goal or trajectory is found. In this restricted setting case, the best that an optimal meta-learner can do is go directly to a goal or execute a particular trajectory at every episode according to the optimal exploration policy as discussed previously, essentially performing a version of posterior sampling. In the more general case with arbitrary reward functions and arbitrary learning procedures, intermediate rewards along a trajectory may be informative, and the optimal exploration strategy may be different from posterior sampling [163], [166], [244].

Nonetheless, the analysis presented in this section provides us insight into the behavior of optimal meta-learning algorithms and allows us to understand the qualities desirable for unsupervised task proposals. The general proposed scheme for unsupervised meta-learning has a significant benefit over standard universal value function and goal reaching style algorithms: it can be applied to arbitrary reward functions going beyond simple goal reaching, and doesn't require the goal to be known in a parametric form beforehand.

Summary of Analysis

Through our analysis, we introduced the notion of optimal meta-learners and analyze their exploration behavior and regret on a class of goal reaching problems. We showed that on these problems, when the test-time task distribution is unknown, the optimal meta-training task distribution for minimizing worst-case test-time regret is *uniform* over the space of goals. We also showed that this

Algorithm 6: Unsupervised Meta-RL Pseudocode

```
Input: $\mathcal{M} \setminus R$, an MDP without a reward function
$D_\phi \leftarrow$ DIAYN() or $D_\phi \leftarrow random$
while not converged do
    Sample latent task variables $z \sim p(z)$
    Define task reward $r_z(s)$ using $D_\phi(z|s)$
    Update $f$ using MAML with reward $r_z(s)$
end while
Return: a learning algorithm $f : D_\phi \to \pi$
```

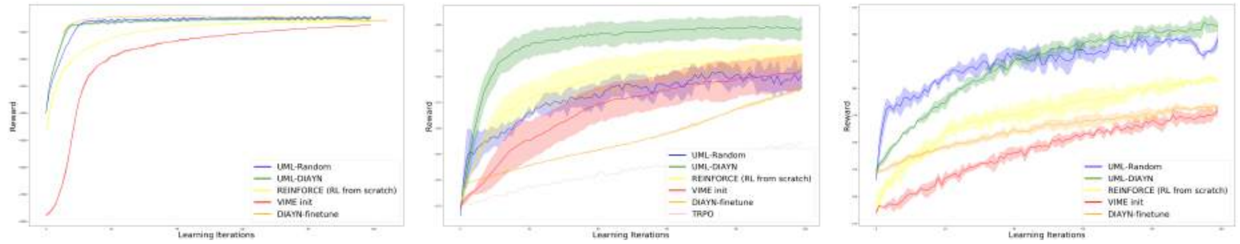
optimal task distribution can be acquired by a simple mutual information maximization scheme. We subsequently extend the analysis to the more general case of matching arbitrary trajectories, as a proxy for the more general class of arbitrary reward functions. In the following section, we will discuss how we can derive a practical algorithm for unsupervised meta-learning from this analysis.

A Practical Algorithm

Following the derivation in the previous section, we can instantiate a practical unsupervised meta-RL algorithm by constructing a task proposal mechanism based on a mutual information objective. A variety of different mutual information objectives can be formulated, including mutual information between single states and z (referred to as DIAYN) [232], pairs of start and end states and z [194], and entire trajectories and z [122], [207], [245]. We will use DIAYN and leave a full examination of possible mutual information objectives for future work.

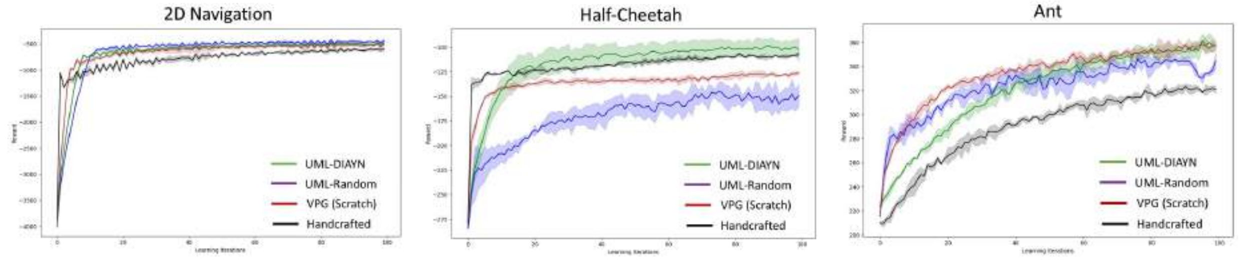
DIAYN optimizes mutual information by training a *discriminator* network $D_\phi(z|\cdot)$ that predicts which z was used to generate the states in a given rollout according to a *latent-conditioned policy* $\pi(a|s,z)$. Our task proposal distribution is thus defined by $r_z(s,a) = \log(D_\phi(z|s))$. The complete unsupervised meta-learning algorithm is as follows: first, we acquire $r_z(s,a)$ by running DIAYN, which learns $D_\phi(z|s)$ and a latent-conditioned policy $\pi(a|s,z)$ (which is discarded). Then, we use $z \sim p(z)$ to propose tasks $r_z(s,a)$ to a standard meta-RL algorithm. This meta-RL algorithm uses the proposed tasks to learn how to learn, acquiring a fast learn algorithm f which can then learn new tasks quickly. While, in principle, any meta-RL algorithm could be used, we use MAML [140] as our meta-learning algorithm. Note that the learning algorithm f returned by MAML is defined simply as running gradient descent using the initial parameters found by MAML as initialization, as discussed in prior work [246]. The method is summarized in Algorithm 6.

In addition to mutual information maximizing task proposals, we will also consider random task proposals, where we also use a discriminator as the reward, according to $r(s,z) = \log D_{\phi_{\text{rand}}}(z|s)$, but where the parameters ϕ_{rand} are chosen randomly (i.e., a random weight initialization for a neural network). While such random reward functions are not optimal, we find that they can surprisingly be used to acquire useful task distributions for simple tasks, though they are not as effective as the tasks become more complicated.



2D navigation Half-Cheetah Ant

Figure 6.2: **Unsupervised meta-learning accelerates learning**: After unsupervised meta-learning, our approach (UML-DIAYN and UML-RANDOM) quickly learns a new task significantly faster than learning from scratch, especially on complex tasks. Learning the task distribution with DIAYN helps more for complex tasks. Results are averaged across 20 evaluation tasks, and 3 random seeds for testing. UML-DIAYN and random also significantly outperform learning with DIAYN initialization or VIME.



2D Navigation Half-Cheetah Ant Navigation

Figure 6.3: **Comparison with handcrafted tasks**: Unsupervised meta-learning (UML-DIAYN) is competitive with meta-training on handcrafted reward functions (i.e., an oracle). A misspecified, handcrafted meta-training task distribution often performs worse, illustrating the benefits of learning the task distribution.

6.4 Experimental Evaluation

In our experiments, we aim to understand whether unsupervised meta-learning as described in Section 6.3 can provide us with an accelerated RL procedure on new tasks. Whereas standard meta-learning requires a hand-specified task distribution at meta-training time, unsupervised meta-learning learns the task distribution through unsupervised interaction with the environment. A fair baseline that likewise uses requires *no reward supervision* at training time, and only uses rewards at test time, is learning via RL from scratch without any meta-learning. As an upper bound, we include the *unfair* comparison to a standard meta-learning approach, where the meta-training distribution is manually designed. This method has access to a hand-specified task distribution that is not available to our method. We evaluate two variants of our approach: (a) task acquisition based on DIAYN followed by meta-learning using MAML, and (b) task acquisition using a randomly initialized discriminator followed by meta-learning using MAML.

Tasks and Implementation Details

Our experiments study three simulated environments of varying difficulty: 2D point navigation, 2D locomotion using the "HalfCheetah," and 3D locomotion using the "Ant," with the latter two environments are modifications of popular RL benchmarks [247]. While the 2D navigation environment allows for direct control of position, HalfCheetah and Ant can only control their center of mass via feedback control with high dimensional actions (6D for HalfCheetah, 8D for Ant) and observations (17D for HalfCheetah, 111D for Ant).

The evaluation tasks, shown in Figure 18.31, are similar to prior work [140], [240]: 2D navigation and ant require navigating to goal positions, while the half cheetah must run at different goal velocities. These tasks are not accessible to our algorithm during meta-training. Please refer to Appendix C for details about hyperparameters for both MAML and DIAYN.

Fast Adaptation after Unsupervised Meta RL

The comparison between the two variants of unsupervised meta-learning and learning from scratch is shown in Figure 6.2. We also add a comparison to VIME [105], a standard novelty-based exploration method, where we pretrain a policy with the VIME reward and then finetune it on the meta-test tasks. In all cases, the UML-DIAYN variant of unsupervised meta-learning produces an RL procedure that outperforms RL from scratch and VIME-init, suggesting that unsupervised interaction with the environment and meta-learning is effective in producing environment-specific but task-agnostic priors that accelerate learning on new, previously unseen tasks. The comparison with VIME shows that the speed of learning is not just about exploration but is indeed about fast adaptation. In our experiments thus far, UML-DIAYN always performs better than learning from scratch, although the benefit varies across tasks depending on the actual performance of DIAYN. We also perform significantly better than a baseline of simply initializing from a DIAYN trained contextual policy, and then finetuning the best skill with the actual task reward.

Interestingly, in many cases (in Figure 6.3) the performance of unsupervised meta-learning with DIAYN matches that of the hand-designed task distribution. We see that on the 2D navigation task, while handcrafted meta-learning is able to learn very quickly initially, it performs similarly after 100 steps. For the cheetah environment as well, handcrafted meta-learning is able to learn very quickly to start off, but is quickly matched by unsupervised meta-RL with DIAYN. On the ant task, we see that hand-crafted meta-learning does do better than UML-DIAYN, likely because the task distribution is challenging, and a better unsupervised task proposal algorithm would improve performance.

The comparison between the two unsupervised meta-learning variants is also illuminating: while the DIAYN-based variant of our method generally achieves the best performance, even the random discriminator is often able to provide a sufficient diversity of tasks to produce meaningful acceleration over learning from scratch in the case of 2D navigation and ant. This result has two interesting implications. First, it suggests that unsupervised meta-learning is an effective tool for learning an environment prior. Although the performance of unsupervised meta-learning can be improved with better coverage using DIAYN (as seen in Figure 6.2), even the random discriminator version provides competitive advantages over learning from scratch. Second, the comparison provides a clue for identifying the source of the structure learned through unsupervised meta-learning: though the particular task distribution has an effect on performance, simply interacting with the environment (without structured objectives, using a random discriminator) already allows meta-RL to learn effective adaptation strategies in a given environment.

6.5 Discussion and Future Work

We presented an unsupervised approach to meta-RL, where meta-learning is used to acquire an efficient RL procedure without requiring hand-specified task distributions. This approach accelerates RL without relying on the manual supervision required for conventional meta-learning algorithms. We provide a theoretical derivation that argues that task proposals based on mutual information maximization can provide a minimum worst-case regret meta-learner, under certain assumptions. Our experiments indicate unsupervised meta-RL can accelerate learning on a range of tasks.

Our approach also opens a number of questions about unsupervised meta-learning algorithms. One limitation of our analysis is that it only considers deterministic dynamics, and only considers task distributions where posterior sampling is optimal. Extending our analysis to stochastic dynamics and more realistic task distributions may allow unsupervised meta-RL to acquire learning algorithms that can more effectively solve real-world tasks.

Chapter 7

Relationship to Other Work on Supervision in Reinforcement Learning

Next we try and place this work in context of some related work both prior to and post publishing of our work. Reward inference in reinforcement learning is a well studied problem, and there have been a number of works in the space of inverse reinforcement learning, language conditioned RL and intrinsic motivation that have been studied in the past. We describe some of these below.

7.1 Connections to Prior Work

Imitation learning has largely been studied previously in the context of learning from low level state demonstrations obtained in the embodiment of the actual system being controlled. For instance, prior work [248], [249] provides demonstrations using a teleoperation system while other work [53], [250] uses a kinesthetic teaching system to provide these demonstrations. Given these demonstrations, then techniques for inverse RL [71], [72] or apprenticeship learning [66], [69] can be applied to learn behaviors. In contrast to these works, our work shows the ability to learn behaviors directly from raw video, without requiring *any* low level demonstrations or state tracking. This was one of the first works to show this kind of behavior from multiple viewpoints. Concurrently, work from other groups [61], [63], [77] showed the ability to also learn from multiple viewpoint aligned videos using a contrastive learning approach.

Learning from examples of outcomes is an easier form of supervision, however the fundamental problem is quite related to the problem of inverse reinforcement learning [71], [72]. While work in inverse RL aims to use entire demos, our work shows the ability to learn simply from outcome states rather than entire trajectories. We build our work in Section 4 on the framework of classifier based rewards, popularized in [97] and [37]. While these techniques train standard classifiers for reward function provision, we show how these can be greatly improved by modeling uncertainty appropriately. In many ways, this is also connected to works for exploration [109], [110], [203] but conducting exploration in a task directed way instead.

Moving from videos and outcome examples to actually learning from abstract supervision like language allows agents to learn with easy to provide and natural sources of supervision. While most work in the space of language and RL lies in the realm of instruction following [143], [144], [149]–[157] our work aims to develop a paradigm where an agent is not just following instructions but actually using human in the loop guidance through the form of language to improve behavior and learn how to solve new tasks. In this way, language is now not just a tool for contextualizing *which* task to solve, but also *how* to solve it.

Finally, moving from supervised forms of RL to an unsupervised form of RL provides an effective way of pre-training agents in an environment without task specific rewards. This work builds on significant prior work dealing with intrinsic motivation [194], [251], [252], curiosity [104], [106], empowerment [191], [253], [254] in order to provide RL agents with signal even in the absence of rewards. Building on these works, our work shows the ability to use a mutual information style objective, in conjunction with meta-learning as a provably good way of pre-training RL agents. Concurrently, [255] showed the ability of a very similar algorithm to learn unsupervised behaviors as well.

7.2 Related Work Subsequent to Publishing

Building on the work we published, several subsequent works studied the "Imitation from Observation" problem. In particular, [256] provides a useful overview of these techniques. Recent work [257] shows the ability to learn across various embodiments via image to image translation, [258] shows an adversarial framework for performing imitation from observation, [259] shows the ability to estimate observation conditional Q-functions from video, [260] shows theoretical results on state-only imitation learning, [261] extends these ideas into the realm of dexterous manipulation, [262] shows the ability to learn from first person ego-centric videos as well. This field is burgeoning and very much in its infancy and many exciting directions are to follow. Recent work has been considering the interaction between language and RL as a guidance tool more closely as well. A recent line of work [263], [264] has shown that language grounding and autonomous skill acquisition can be decoupled, while still retaining the benefits of both language and large scale autonomy. In a similar vein, [265] shows the efficacy of using language as an abstraction tool for hierarchical RL and relabeling. [266], [267] show the ability of language agents to cooperatively perform tasks using dialogue. And moving slightly away from language, [268] shows the ability for RL agents to interactively learn how to assist a human in performing its own tasks using a very related set of techniques.

Unsupervised reinforcement learning has seen a huge increase in attention in recent times. This has happened both from the lens of exploration [269] as well as representation learning [270]–[272] Recent work [245], [273] has shown the ability for agents to learn even more complex tasks unsupervised like the Humanoid, while also being efficient enough to apply to the real world. In a different vein, [274] has shown the ability of agents to learn how to set goals and accomplish them in a natural curriculum to solve complex robotics tasks with minimal human supervision. Unsupervised learning techniques [270]–[272] have also seen a huge uptick in enabling vision based RL in recent years.

Part II

Distributions

In Part I, we discussed how we can actually go about supervising real world RL systems, providing them reward functions through data driven reward inference. Once we actually know the right objectives to optimize, the next question becomes how do we actually go about exploring the environment to collect the right data needed to optimize the RL objective in an efficient, safe and directed manner. The exploration problem in RL has been studied in great detail and in its most general form is extremely difficult to tackle in practical time-scales. Moreover, exploration in the context of standard RL can be arbitrarily unsafe, expensive and damaging to the environment. This is reasonable in video games or simulated domains, but the same cannot be said about domains like a robot operating in someone's home. In these cases, data has a limiting cost, which requires us to think carefully about how to guide exploration to collect the right data in a safe, efficient and practical way.

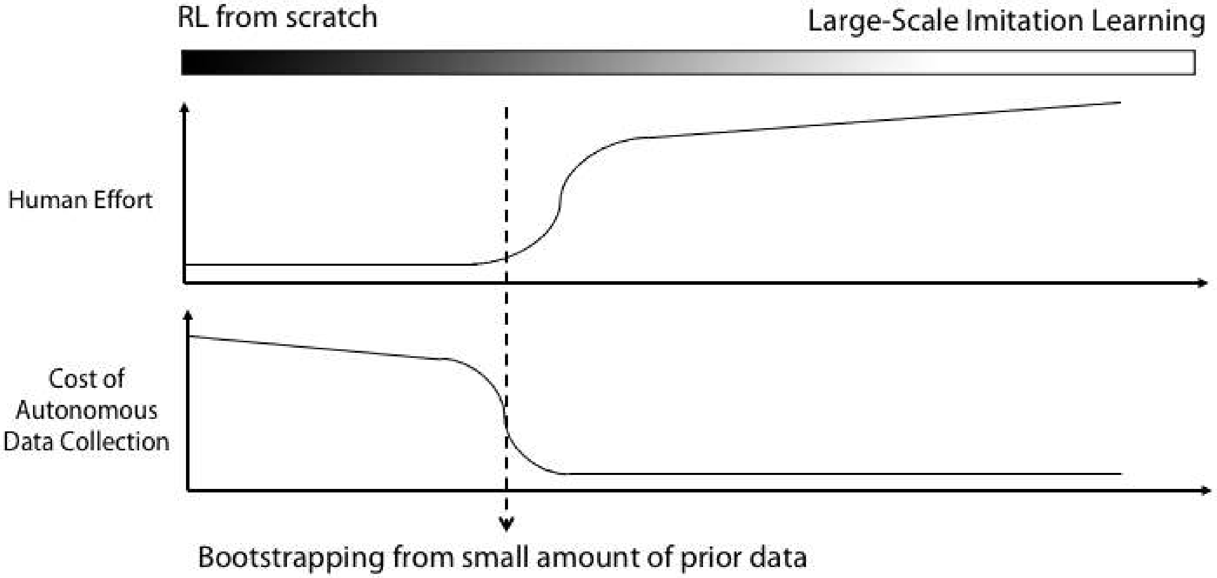

RL from scratch
Large-Scale Imitation Learning
Human Effort
Cost of Autonomous Data Collection
Bootstrapping from small amount of prior data

Figure 7.1: Schematic illustration of the tradeoff between human effort and the cost of autonomous data collection in RL

One way of moving from the hopelessly intractable, most general form of RL to a more tractable problem formulation is to consider how a small amount of human provided data can significantly bring down the cost of autonomous data collection. As seen from the schematic in Fig 7.1, a small amount of human provided data can significantly offset how much autonomous data collection costs, making it much more well directed, safe as well as efficient. In this part, we largely explore this problem statement - we hope to understand how a small amount of (potentially suboptimal) human provided data can bootstrap the process of reinforcement learning, encouraging safe, directed and efficient data collection. In doing so, we show that we can go significantly beyond the types of tasks that were possible before, and solve much more complicated problems in practical time frames. In particular, in Section 10 we discuss an on-policy RL algorithm to bootstrap RL with human demonstrations, followed by it's application to real world robotics problems in Section 11. In Section 12, we then discuss how to extend this paradigm to solve long horizon manipulation tasks. In Section 13, we show how this process can be made much more efficient using off-policy reinforcement learning algorithms and show how this results in efficient real world training as well.

Chapter 8

Bootstrapping On-Policy Reinforcement Learning with Human Demonstrations

As discussed in chapter 9, bootstrapping RL from prior data can provide more efficient, safer and more directed data collection for learning complex tasks. To this end, we develop a novel algorithm for bootstrapping RL from human provided demonstration data, and show this actually enables us to solve significantly more challenging dexterous manipulation tasks than was previously thought to be possible. While the discussion in this chapter is largely limited to simulation, in the following chapter we show how this algorithm can be implemented and trained on a real world hardware system.

8.1 Why Does Complex Dexterous Manipulation Require Demonstration Bootstrapped RL?

Multi-fingered dexterous manipulators are crucial for robots to function in human-centric environments, due to their versatility and potential to enable a large variety of contact-rich tasks, such as in-hand manipulation, complex grasping, and tool use. However, this versatility comes at the price of high dimensional observation and action spaces, complex and discontinuous contact patterns, and under-actuation during non-prehensile manipulation. This makes dexterous manipulation with multi-fingered hands a challenging problem.

Dexterous manipulation behaviors with multi-fingered hands have previously been obtained using model-based trajectory optimization methods [275], [276]. However, these methods typically rely on accurate dynamics models and state estimates, which are often difficult to obtain for contact rich manipulation tasks, especially in the real world. Reinforcement learning provides a model agnostic approach that circumvents these issues. Indeed, model-free methods have been used for acquiring manipulation skills [35], [277], but so far have been limited to simpler behaviors with 2-3 finger hands or whole-arm manipulators, which do not capture the challenges of high-dimensional multi-fingered hands.

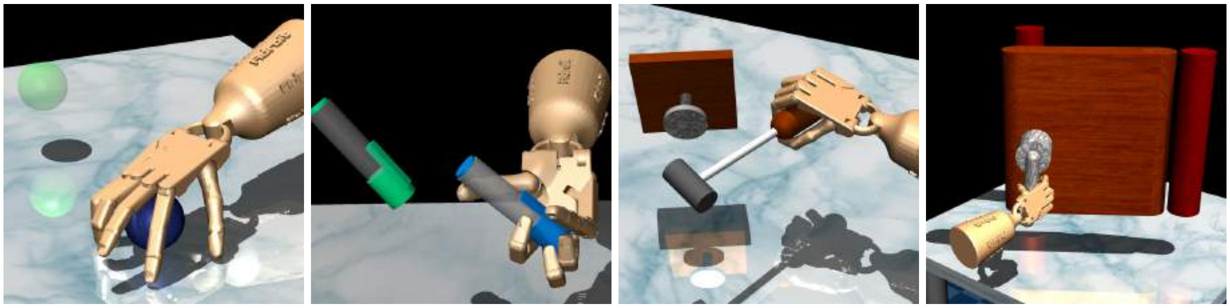


Figure 8.1: We demonstrate a wide range of dexterous manipulation skills such as object relocation, in-hand manipulation, tool use, and opening doors using DRL methods. By augmenting with human demonstrations, policies can be trained in the equivalent of a few real-world robot hours.

DRL research has made significant progress in improving performance on standardized benchmark tasks, such as the OpenAI gym benchmarks [29]. However, the current benchmarks are typically quite limited both in the dimensionality of the tasks and the complexity of the interactions. Indeed, recent work has shown that simple linear policies are capable of solving many of the widely studied benchmark tasks [20]. Thus, before we can develop DRL methods suitable for dexterous manipulation with robotic hands, we must set up a suite of manipulation tasks that exercise the properties that are most crucial for real-world hands: high dimensionality, rich interactions with objects and tools, and sufficient task variety. To that end, we begin by proposing a set of 4 dexterous manipulation tasks in simulation, which are illustrated in Figure 1. These tasks are representative of the type of tasks we expect robots to be proficient at: grasping and moving objects, in-hand manipulation, and tool usage among others. Using these representative tasks, we study how DRL can enable learning of dexterous manipulation skills. Code, models, and videos accompanying this work can be found at: `http://sites.google.com/view/deeprl-dexterous-manipulation`

We find that existing RL algorithms can indeed solve these dexterous manipulation tasks, but require significant manual effort in reward shaping. In addition, the sample complexity of these methods is very poor, thus making real world training infeasible, and the resulting policies exhibit idiosyncratic strategies and poor robustness. To overcome this challenge, we propose to augment the policy search process with a small number of human demonstrations collected in virtual reality (VR). In particular, we find that pre-training a policy with behavior cloning, and subsequent fine-tuning with policy gradient along with an augmented loss to stay close to the demonstrations, dramatically reduces the sample complexity, enabling training within the equivalent of a few real-world robot hours. The use of human demonstrations also provides additional desirable qualities such as human-like smooth behaviors and robustness to variations in the environment. Although success remains to be demonstrated on hardware, our results in this work indicate that DRL methods when augmented with demonstrations are a viable option for real-world learning of dexterous manipulation skills. Our contributions are:

- We demonstrate, in simulation, dexterous manipulation with high-dimensional human-like five-finger hands using model-free DRL. To our knowledge, this is the first empirical result that demonstrates model-free learning of tasks of this complexity.

- We show that with a small number of human demonstrations, the sample complexity can be reduced dramatically and brought to levels which can be executed on physical systems.

- We also find that policies trained with demonstrations are more human-like as well as robust to variations in the environment. We attribute this to human priors in the demonstrations which bias the learning towards more robust strategies.

- We propose a set of dexterous hand manipulation tasks, which would be of interest to researchers at the intersection of robotic manipulation and machine learning.

8.2 Related Work

Manipulation with dexterous hands represents one of the most complex and challenging motor control tasks carried out by humans, and replicating this behavior on robotic systems has proven extremely difficult. Aside from the technical challenges of constructing multi-fingered dexterous hands, control of these manipulators has turned out to be exceedingly challenging. Many of the past successes in dexterous manipulation have focused either on designing hands that mechanically simplify the control problem [278], [279], at the expense of reduced flexibility, or on acquiring controllers for relatively simple behaviors such as grasping [280] or rolling objects in the fingers [277], often with low degree of freedom manipulators. Our work explores how a combination of DRL and demonstration can enable dexterous manipulation with high-dimensional human-like five-finger hands [281], [282], controlled at the level of individual joints. We do not aim to simplify the morphology, and explore highly complex tasks that involve tool use and object manipulation. Although our experiments are in simulation, they suggest that DRL might in the future serve as a powerful tool to enable much more complex dexterous manipulation skills with complex hands.

Model-based trajectory optimization : Model-based trajectory optimization methods [275], [276], [283] have demonstrated impressive results in simulated domains, particularly when the dynamics can be adjusted or relaxed to make them more tractable (as, e.g., in computer animation). Unfortunately, such approaches struggle to translate to real-world manipulation since prespecifying or learning complex models on real world systems with significant contact dynamics is very difficult. Although our evaluation is also in simulation, our algorithms do not make any assumption about the structure of the dynamics model, requiring only the ability to generate sample trajectories. Our approach can be executed with minimal modification on real hardware, with the limitation primarily being the number of real-world samples required. As we will show, this can be reduced significantly with a small number of demonstrations, suggesting the possibility of performing learning directly in the real world.

Model-free reinforcement learning Model-free RL methods [284], [285] and versions with deep function approximators [90], [286] do not require a model of the dynamics, and instead optimize the policy directly. However, their primary drawback is the requirement for a large number of real-world samples. Some methods like PoWER overcome this limitation through the use of simple policy representations such as DMPs and demonstrate impressive results [287], [288]. However, more complex representations may be needed in general for more complex tasks and to incorporate rich sensory information. More recently, RL methods with rich neural network function approximators have been studied in the context of basic manipulation tasks with 7-10 DoF manipulators, and have proposed a variety of ways to deal with the sample complexity. Prior work has demonstrated model-free RL in the real world with simulated pre-training [289] and parallelized data collection [35] for lower-dimensional whole-arm manipulation tasks. Our work builds towards model-free DRL on anthropomorphic hands, by showing that we can reduce sample complexity of learning to practical levels with a small number of human demonstrations. Prior work has demonstrated learning of simpler manipulation tasks like twirling a cylinder with a similar morphology [290] using guided policy search [31], and extended this approach to also incorporate demonstrations [291]. However, the specific tasks we demonstrate are substantially more complex featuring a large number of contact points and tool use, as detailed in Section 9.3.

Imitation learning In imitation learning, demonstrations of successful behavior are used to train policies that imitate the expert providing these successful trajectories [64]. A simple approach to imitation learning is behavior cloning (BC), which learns a policy through supervised learning to mimic the demonstrations. Although BC has been applied successfully in some instances like autonomous driving [292], it suffers from problems related to distribution drift [175]. Furthermore, pure imitation learning methods cannot exceed the capabilities of the demonstrator since they lack a notion of task performance. In this work, we do not just perform imitation learning, but instead use imitation learning to bootstrap the process of reinforcement learning. The bootstrapping helps to overcome exploration challenges, while RL fine-tuning allows the policy to improve based on actual task objective.

Combining RL with demonstrations Methods based on dynamic movement primitives (DMPs) [67], [287], [293], [294] have been used to effectively combine demonstrations and RL to enable faster learning. Several of these methods use trajectory-centric policy representations, which although well suited for imitation, do not enable feedback on rich sensory inputs. Although such methods have been applied to some dexterous manipulation tasks [277], the tasks are comparatively simpler than those illustrated in our work. Using expressive function approximators allow for complex, nonlinear ways to use sensory feedback, making them well-suited to dexterous manipulation.

In recent work, demonstrations have been used for pre-training a Q-function by minimizing TD error [295]. Additionally demonstrations have been used to guide exploration through reward/policy shaping but these are often rule-based or work on discrete spaces making them difficult to apply to high dimensional dexterous manipulation [296]–[298]. The work most closely related to ours is DDPGfD [299], where demonstrations are incorporated into DDPG [286] by adding them to the replay buffer. This presents a natural and elegant way to combine demonstrations with an off-policy RL method. In concurrent work [300], this approach was combined with hindsight experience replay [119]. The method we propose in this work bootstraps the policy using behavior cloning, and combines demonstrations with an on-policy policy gradient method. Off-policy methods, when

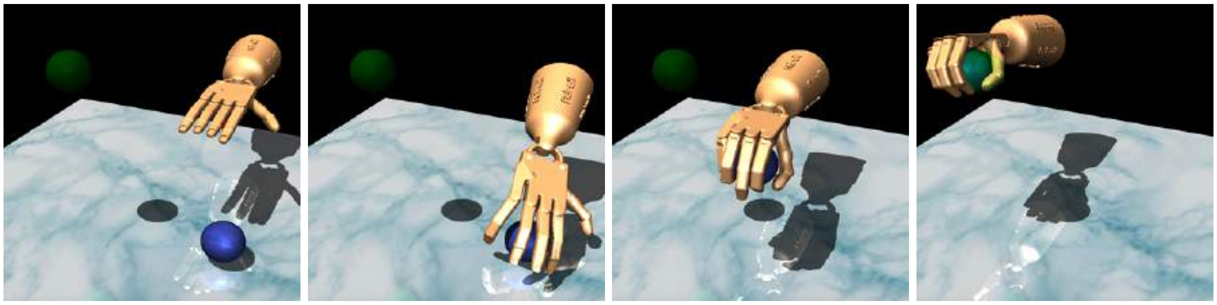


Figure 8.2: Object relocation – move the blue ball to the green target. Positions of the ball and target are randomized over the entire workspace. Task is considered successful when the object is within epsilon-ball of the target.

successful, tend to be more sample efficient, but are generally more unstable [247], [301]. On-policy methods on the other hands are more stable, and scale well to high dimensional spaces [247]. Our experimental results indicate that with the incorporation of demonstrations, the sample complexity of on-policy methods can be dramatically reduced, while retaining their stability and robustness. Indeed, the method we propose significantly outperforms DDPGfD. Concurrent works with this paper have also proposed to integrate demonstrations into reward functions [302], and there have been attempts to learn with imperfect demonstrations [303]. Overall, we note that the general idea of bootstrapping RL with supervised training is not new. However, the extent to which it helps with learning of complex dexterous manipulation skills is surprising and far from obvious.

8.3 Dexterous Manipulation Tasks

The real world presents a plethora of interesting and important manipulation tasks. While solving individual tasks via custom manipulators in a controlled setting has led to success in industrial automation, this is less feasible in an unstructured settings like the home. Our goal is to pick a minimal task-set that captures the technical challenges representative of the real world. We present four classes of tasks - object relocation, in-hand manipulation, tool use, and manipulating environmental props (such as doors). Each class exhibits distinctive technical challenges, and represent a large fraction of tasks required for proliferation of robot assistance in daily activities – thus being potentially interesting to researchers at the intersection of robotics and machine learning. All our task environments expose hand (joint angles), object (position and orientation), and target (position and orientation) details as observations, expect desired position of hand joints as actions, and provides an oracle to evaluate success. We now describe the four classes in light of the technical challenges they present.

Tasks

Object relocation (Figure 8.2)

Object relocation is a major class of problems in dexterous manipulation, where an object is picked up and moved to a target location. The principal challenge here from an RL perspective is exploration, since in order to achieve success, the hand has to reach the object, grasp it, and take it

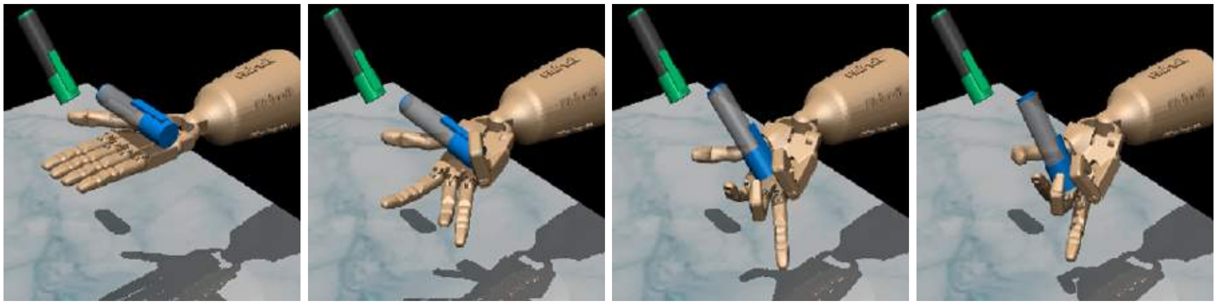


Figure 8.3: In-hand manipulation – reposition the blue pen to match the orientation of the green target. The base of the hand is fixed. The target is randomized to cover all configurations. Task is considered successful when the orientations match within tolerance.

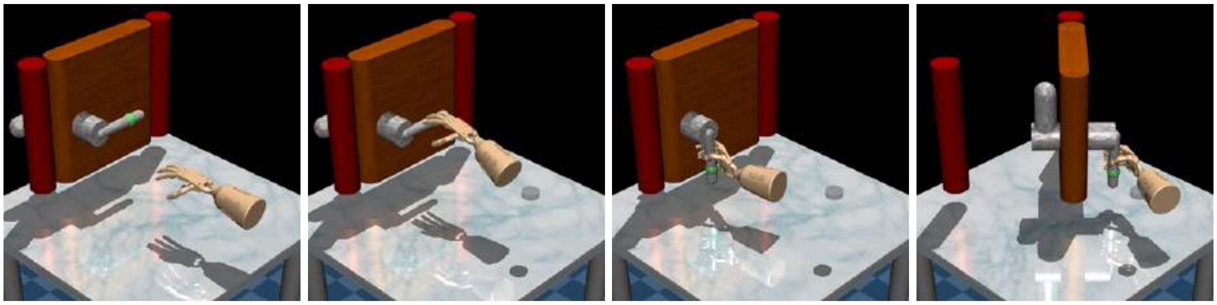


Figure 8.4: Door opening – undo the latch and swing the door open. The latch has significant dry friction and a bias torque that forces the door to stay closed. Agent leverages environmental interaction to develop the understanding of the latch as no information about the latch is explicitly provided. The position of the door is randomized. Task is considered complete when the door touches the door stopper at the other end.

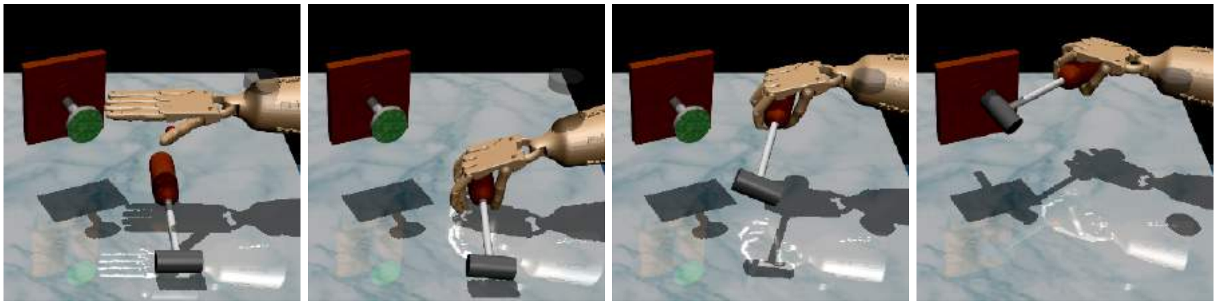


Figure 8.5: Tool use – pick up and hammer with significant force to drive the nail into the board. Nail position is randomized and has dry friction capable of absorbing up to 15N force. Task is successful when the entire length of the nail is inside the board.

to the target position – a feat that is very hard to accomplish without priors in the form of shaped rewards or demonstrations.

In-hand Manipulation – Repositioning a pen (Figure 8.3)

In hand-manipulation maneuvers like re-grasping, re-positioning, twirling objects etc. involve leveraging the dexterity of a high DOF manipulator to effectively navigate a difficult landscape filled with constraints and discontinuities imposed by joint limits and frequently changing contacts.

Due to the large number of contacts, conventional model-based approaches which rely on accurate estimates of gradients through the dynamics model struggle in these problem settings. The major challenge in these tasks is representing the complex solutions needed for different maneuvers. For these reason, sampling based DRL methods with rich neural network function approximators are particularly well suited for this class of problems. Previous work on in-hand manipulation with RL [290] has considered simpler tasks such as twirling a cylinder, but our tasks involve omni-directional repositioning which involves significantly more contact use. Collecting human demonstrations for this task was challenging due to lack of tactile feedback in VR. Instead, to illustrate the effectiveness of our proposed algorithms, we used a computational expert trained using RL on a well shaped reward for many iterations. This expert serves to give demonstrations which are used to speed up training from scratch.

Manipulating Environmental Props (Figure 8.4)

Real-world robotic agents will require constant interaction and manipulation in human-centric environments. Tasks in this class involve modification of the environment itself - opening drawers for fetching, moving furniture for cleaning, etc. The solution is often multi-step with hidden subgoals (e.g undo latch before opening doors), and lies on a narrow constrained manifold shaped primarily by the inertial properties and the under actuated dynamics of the environment.

Tool Use – Hammer (Figure 8.5)

Humans use tools such as hammers, levers, etc. to augment their capabilities. These tasks involve co-ordination between the fingers and the arm to apply the tool correctly. Unlike object relocation, the goal in this class of tasks is to use the tool as opposed to just relocating it. Not all successful grasp leads to effective tool use. Effective tool use requires multiple steps involving grasp reconfiguration and careful motor co-ordination in order to impart the required forces. In addition, effective strategies needs to exhibit robust behaviors in order to counter and recover from destabilizing responses from the environment.

Further details about all the tasks, including detailed shaped reward functions, physics parameters etc. are available in the project website: `http://sites.google.com/view/deeprl-dexterous-manipulation`.

Experimental setup

To accomplish the tasks laid out above, we use a high degree of freedom dexterous manipulator and a virtual reality demonstration system which we describe below.

ADROIT hand

We use a simulated analogue of a highly dexterous manipulator – ADROIT [281], which is a 24-DoF anthropomorphic platform designed for addressing challenges in dynamic and dexterous manipulation [276], [290]. The first, middle, and ring fingers have 4 DoF. Little finger and thumb have 5 DoF, while the wrist has 2 DoF. Each DoF is actuated using position control and is equipped with a joint angle sensor (Figure 8.6).

Simulator

Our experimental setup uses the MuJoCo physics simulator [91]. The stable contact dynamics of MuJoCo [304] makes it well suited for contact rich hand manipulation tasks. The kinematics, the dynamics, and the sensing details of the physical hardware were carefully modeled to encourage physical realism. In addition to dry friction in the joints, all hand-object contacts have planar friction. Object-fingertip contacts support torsion and rolling friction. Though the simulation supports tactile feedback, we do not use it in this work for simplicity, but expect that its use will likely improve the performance.

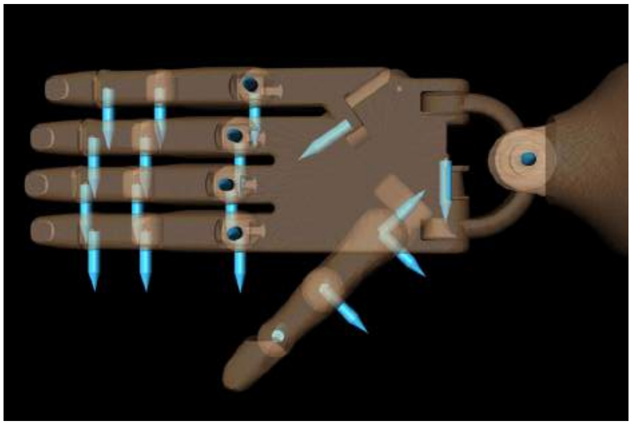


Figure 8.6: 24 degree of freedom ADROIT hand. The blue arrows mark the position of the joints and corresponding position actuator.

Demonstrations

Accurate demonstrations data are required to help accelerate various learning algorithms. Standard methods like kinesthetic teaching are impractical with complex systems like ones we study in this work. We use an updated version of the Mujoco HAPTIX system [305]. The system uses the CyberGlove III system for recording the fingers, HTC vive tracker for tracking the base of the hand and HTC vive headset for stereoscopic visualization. This moves the process of demonstration data collection from the real world to virtual reality, allowing for several high fidelity demonstrations for tasks involving large number of contacts and dynamic phenomena such as rolling, sliding, stick-slip, deformations and soft contacts. Since the demonstrations are provided in simulation, physically consistent details of the movements can be easily recorded. We gathered 25 successful demonstrations for all our tasks (with task randomization as outlined in captions of Figure 8.2, 8.3, 8.5, and 8.4), with each demonstration consisting of the state-action trajectories needed to perform the task in simulation. To combat distribution drift, a small amount of noise (uniform random $[-0.1, 0.1]$ radians) is added to the actuators per timestep so that the policy can better capture relevant statistics about the data.

8.4 Demo Augmented Policy Gradient (DAPG)

In this work, we use a combination of RL and imitation learning to solve complex dexterous manipulation problems. To reduce sample complexity and help with exploration, we collect a few expert demonstrations using the VR system described in Section 8.3, and incorporate these into the RL process. We first present the base RL algorithm we use for learning, and finally describe our procedure to incorporate demonstrations. The preliminaries and notation we use is the same one introduced in Chapter 1.

Natural Policy Gradient

In this work, we primarily consider policy gradient methods, which are a class of model-free RL methods. In policy gradient methods, the parameters of the policy are directly optimized to maximize the objective, $\eta(\theta)$, using local search methods such as gradient ascent. In particular, for this work we consider the NPG algorithm [20], [306], [307]. First, NPG computes the vanilla policy gradient, or REINFORCE [19] gradient:

$$g = \frac{1}{NT} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^i | s_t^i) \hat{A}^\pi(s_t^i, a_t^i, t). \tag{8.1}$$

Secondly, it pre-conditions this gradient with the (inverse of) Fisher Information Matrix [306], [308] computed as:

$$F_\theta = \frac{1}{NT} \sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(a_t^i | s_t^i) \nabla_\theta \log \pi_\theta(a_t^i | s_t^i)^T, \tag{8.2}$$

and finally makes the following normalized gradient ascent update [20], [90], [307]:

$$\theta_{k+1} = \theta_k + \sqrt{\frac{\delta}{g^T F_{\theta_k}^{-1} g}}\, F_{\theta_k}^{-1} g, \tag{8.3}$$

where δ is the step size choice. A number of pre-conditioned policy gradient methods have been developed in literature [18], [20], [90], [306], [307], [309], [310] and in principle any of them could be used. Our implementation of NPG for the experiments is based on Rajeswaran et al. [20].

Augmenting RL with demonstrations

Although NPG with an appropriately shaped reward can somewhat solve the tasks we consider, there are several challenges which necessitate the incorporations of demonstrations to improve RL:

1. RL is only able to solve the tasks we consider with careful, laborious task reward shaping.
2. While RL eventually solves the task with appropriate shaping, it requires an impractical number of samples to learn - in the order of a 100 hours for some tasks.
3. The behaviors learned by pure RL have unnatural appearance, are noisy and are not as robust to environmental variations.

Combining demonstrations with RL can help combat all of these issues. Demonstrations help alleviate the need for laborious reward shaping, help guide exploration and decrease sample complexity of RL, while also helping produce robust and natural looking behaviors. We propose the demonstration augmented policy gradient (DAPG) method which incorporates demonstrations into policy gradients in two ways:

Pretraining with behavior cloning

Policy gradient methods typically perform exploration by utilizing the stochasticity of the action distribution defined by the policy itself. If the policy is not initialized well, the learning process could be very slow with the algorithm exploring state-action spaces that are not task relevant. To combat this, we use behavior cloning (BC) [58], [292] to provide an informed policy initialization that efficiently guides exploration. Use of demonstrations circumvents the need for reward shaping often used to guide exploration. This idea of pretraining with demonstrations has been used successfully in prior work [67], and we show that this can dramatically reduce the sample complexity for dexterous manipulation tasks as well. BC corresponds to solving the following maximum-likelihood problem:

$$\underset{\theta}{\text{maximize}} \sum_{(s,a)\in\rho_D} \ln \pi_\theta(a|s). \tag{8.4}$$

The optimizer of the above objective, called the behavior cloned policy, attempts to mimic the actions taken in the demonstrations at states visited in the demonstrations. In practice, behavior cloning does not guarantee that the cloned policy will be effective, due to the distributional shift between the demonstrated states and the policy's own states [175]. Indeed, we observed experimentally that the cloned policies themselves were usually not successful.

RL fine-tuning with augmented loss

Though behavior cloning provides a good initialization for RL, it does not optimally use the information present in the demonstration data. Different parts of the demonstration data are useful in different stages of learning, especially for tasks involving a sequence of behaviors. For example, the hammering task requires behaviors such as reaching, grasping, and hammering. Behavior cloning by itself cannot learn a policy that exhibits all these behaviors in the correct sequence with limited data. The result is that behavior cloning produces a policy that can often pick up the hammer but seldom swing it close to the nail. The demonstration data contains valuable information on how to hit the nail, but is lost when the data is used only for initialization. Once RL has learned to pick up

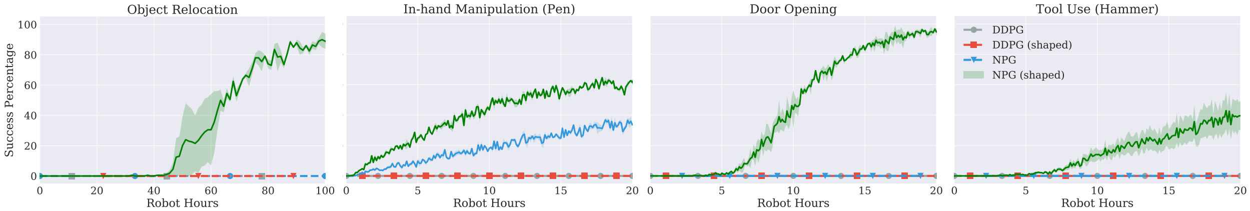


Figure 8.7: Performance of pure RL methods – NPG and DDPG, with sparse task completion reward and shaped reward. Sparse reward setting is primarily ineffective in solving our task set (expect in-hand manipulation). In corporation of human priors in-terms of reward shaping helps NPG get off the ground, but DDPG sill struggles to find success.

the hammer properly, we should use the demonstration data to provide guidance on how to hit the nail. To capture all information present in the demonstration data, we add an additional term to the gradient:

$$\begin{aligned} g_{aug} = \sum_{(s,a)\in\rho_\pi} & \nabla_\theta \ln \pi_\theta(a|s) A^\pi(s,a) + \\ \sum_{(s,a)\in\rho_D} & \nabla_\theta \ln \pi_\theta(a|s) w(s,a). \end{aligned} \tag{8.5}$$

Here ρ_π represents the dataset obtained by executing policy π on the MDP, and $w(s,a)$ is a weighting function. This augmented gradient is then used in eq. (4) to perform a co-variant update. If $w(s,a) = 0 \; \forall(s,a)$, then we recover the policy gradient in eq. (2). If $w(s,a) = c \; \forall(s,a)$, with sufficiently large c, it reduces to behavior cloning, as in eq. (5). However, we wish to use both imitation and reinforcement learning, so we require an alternate weighting function. The analysis in [311] suggests that eq. (2) is also valid for mixture trajectory distributions of the form $\rho = \alpha\rho_\pi + (1-\alpha)\rho_D$. Thus, a natural choice for the weighting function would be $w(s,a) = A^\pi(s,a) \; \forall(s,a) \in \rho_D$. However, it is not possible to compute this quantity without additional rollouts or assumptions [312]. Thus, we use the heuristic weighting scheme:

$$w(s,a) = \lambda_0 \lambda_1^k \max_{(s',a')\in\rho_\pi} A^\pi(s',a') \;\; \forall(s,a) \in \rho_D,$$

where λ_0 and λ_1 are hyperparameters, and k is the iteration counter. The decay of the weighting term via λ_1^k is motivated by the premise that initially the actions suggested by the demonstrations are at least as good as the actions produced by the policy. However, towards the end when the policy is comparable in performance to the demonstrations, we do not wish to bias the gradient. Thus, we asymptotically decay the auxiliary objective. We empirically find that the performance of the algorithm is not very sensitive to the choice of these hyperparameters. For all the experiments, $\lambda_0 = 0.1$ and $\lambda_1 = 0.95$ was used.

8.5 Results and Discussion

Our results study how RL methods can learn dexterous manipulation skills, comparing several recent algorithms and reward conditions. First, we evaluate the capabilities of RL algorithms to learn dexterous manipulation behaviors from scratch on the tasks outlined in Section III. Subsequently, we demonstrate the benefits of incorporating human demonstrations with regard to faster learning, increased robustness of trained policies, and ability to cope with sparse task completion rewards.

Reinforcement Learning from Scratch

We aim to address the following questions in this experimental evaluation:

1. Can existing RL methods cope with the challenges presented by the high dimensional dexterous manipulation tasks?
2. Do the resulting policies exhibit desirable properties like robustness to variations in the environment?
3. Are the resulting movements safe for execution on physical hardware, and are elegant/nimble/human-like?

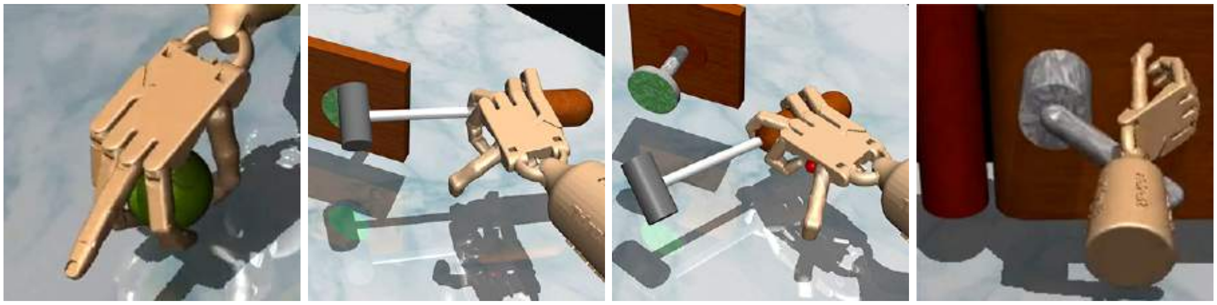


Figure 8.8: Unnatural movements observed in the execution trace of behavior trained with pure reinforcement leaning. From left to right: (a) unnatural, socially unacceptable, finger position during pick up. (b/c) unnatural grasp for hammer (d) unnatural use of wrist for unlatching the door.

In order to benchmark the capabilities of DRL with regard to the dexterous manipulation tasks outlined in Section 9.3, we evaluate the NPG algorithm described briefly in Section V, and the DDPG algorithm [286], which has recently been used in a number of robotic manipulation scenarios [35], [299], [300]. Both of these methods have demonstrated state of the art results in popular DRL continuous control benchmarks, and hence serve as a good representative set. We score the different methods based on the percentage of successful trajectories the trained policies can generate, using a sample size of 100 trajectories. We find that with sparse task completion reward signals, the policies with random exploration never experience success (except in the in-hand task) and hence do not learn.

In order to enable these algorithms to learn, we incorporate human priors on how to accomplish the task through careful *reward shaping*. With the shaped rewards, we find that NPG is indeed able to achieve high success percentage on these tasks (Figure 8.7), while DDPG was unable to learn successful policies despite considerable hyperparameter tuning. DDPG can be very sample efficient, but is known to be very sensitive to hyperparameters and random seeds [301], which may explain the difficulty of scaling it to complex, high-dimensional tasks like dexterous manipulation.

Table 8.1: Sample and robot time complexity of DAPG (ours) compared to RL (Natural Policy Gradient) from scratch with shaped (sh) and sparse task completion reward (sp). N is the number of RL iterations needed to achieve 90% success rate, Hours represent the robot hours needed to learn the task. Each iteration is 200 trajectories of length 2 seconds each.

Method	DAPG	(sp)	RL	(sh)	RL	(sp)
Task	N	Hours	N	Hours	N	Hours
Relocation	52	5.77	880	98	∞	∞
Hammer	55	6.1	448	50	∞	∞
Door	42	4.67	146	16.2	∞	∞
Pen	30	3.33	864	96	2900	322

Although incorporation of human knowledge via reward shaping is helpful, the resulting policies: (a) often exhibit unnatural looking behaviors, and (b) are too sample inefficient to be useful for training on the physical hardware. While it is hard to mathematically quantify the quality of generated behaviors, Figure 8.8 and the accompanying video clearly demonstrate that the learned policies produce behaviors that are erratic and not human-like. Such unnatural behaviors are indeed quite prevalent in the recent DRL results [313]. Furthermore, we take the additional step of analyzing the robustness of these policies to variations in environments that were not experienced during training. To do so, we take into account the policy trained for the object relocation task and vary the mass and size of the object that has to be relocated. We find that the policies tend to over-fit to the specific objects they were trained to manipulate and is unable to cope with variations in the environment as seen in Figure 9.16.

We attribute the artifacts and brittleness outlined above to the way in which human priors are incorporated into the policy search process. The mental models of solution strategies that humans have for these tasks are indeed quite robust. However it is challenging to distill this mental model or intuition into a mathematical reward function. As we will show in the next section, using a data driven approach to incorporate human priors, in the form of demonstrations, alleviates these issues to a significant extent.

Reinforcement Learning with Demonstrations

In this section we aim to study the following questions:

1. Does incorporating demonstrations via DAPG reduce the learning time to practical timescales?

2. How does DAPG compare to other model-free methods that incorporate demonstrations, such as DDPGfD [299]?

3. Does DAPG acquire robust and human-looking behaviors without reward shaping?

We employ the DAPG algorithm in Section 8.4 on the set of hand tasks and compare with the recently proposed DDPGfD method [299]. DDPGfD builds on top of the DDPG algorithm, and

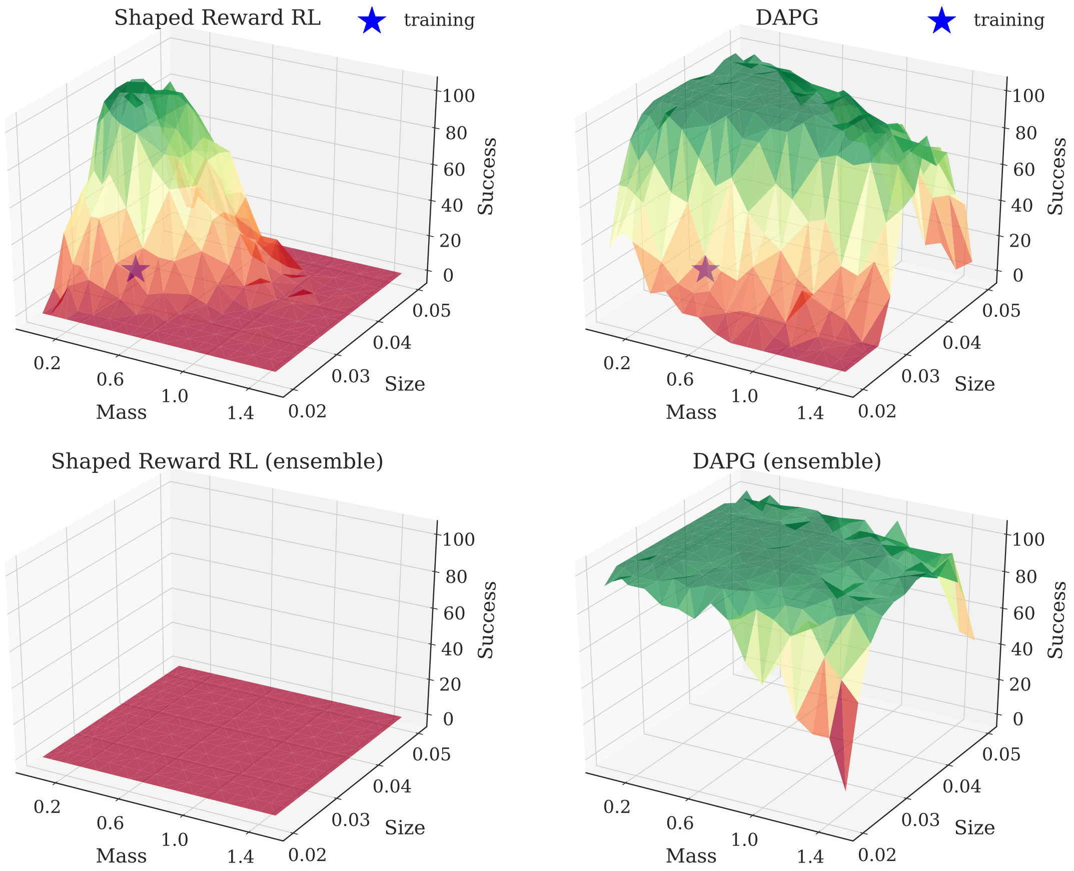


Figure 8.9: Robustness of trained policies to variations in the environment. The top two figures are trained on a single instance of the environment (indicated by the star) and then tested on different variants of the environment. The policy trained with DAPG is more robust, likely due to the intrinsic robustness of the human strategies which are captured through use of demonstrations. We also see that RL from scratch is unable to learn when the diversity of the environment is increased (bottom left) while DAPG is able to learn and produces even more robust policies.

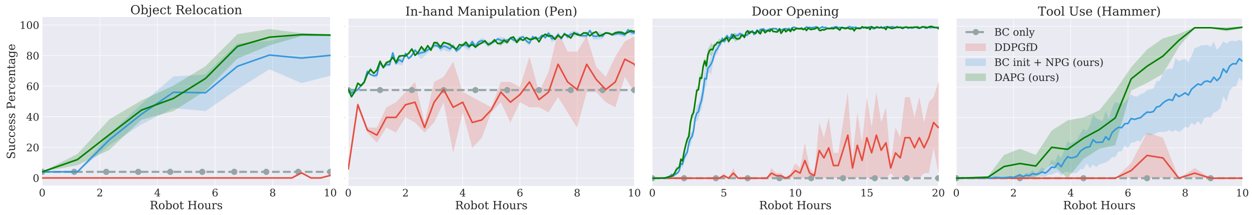


Figure 8.10: Performance of RL with demonstrations methods – DAPG(ours) and DDPGfD. DAPG significantly outperforms DDPGfD

incorporates demonstrations to bootstrap learning by: (1) Adding demonstrations to the replay buffer; (2) Using prioritzed experience replay; (3) Using n-step returns; (4) Adding regularizations to the policy and critic networks. Overall, DDPGfD has proven effective on arm manipulation tasks with sparse rewards in prior work, and we compare performance of DAPG against DDPGfD on our dexterous manipulation tasks.

For this section of the evaluation we use only sparse task completion rewards, since we are using demonstrations. With the use of demonstrations, we expect the algorithms to implicitly learn the human priors on how to accomplish the task. Figure 8.10 presents the comparison between the different algorithms. DAPG convincingly outperforms DDPGfD in all the tasks well before DDPGfD even starts showing signs of progress. Furthermore, DAPG is able to train policies for these complex tasks in under a few robot hours Table 8.1. In particular, for the object relocation task, DAPG is able to train policies almost 30 times faster compared to learning from scratch. This indicates that RL methods in conjunction with demonstrations, and in particular DAPG, are viable approaches for real world training of dexterous manipulation tasks.

When analyzing the robustness of trained policies to variations in the environment, we find that policies trained with DAPG are significantly more robust compared to the policies trained with shaped rewards (Figure 9.16). Furthermore, a detail that can be well appreciated in the accompanying video is the human-like motions generated by the policy. The policies trained with DAPG better capture the human priors from the demonstrations, and generate policies that are more robust and exhibit qualities that are inherently expected but hard to mathematically specify.

In our final experiment, we also explore how robustness can be further improved by training on a distribution (ensemble) of training environments, where each environment differs in terms of the physical properties of the manipulated object (its size and mass). By training policies to succeed on the entire ensemble, we can acquire policies that are explicitly trained for robustness, similar in spirit to previously proposed model ensemble methods [314], [315]. Interestingly, we observe that for difficult control problems like high dimensional dexterous manipulation, RL from scratch with shaped rewards is unable to learn a robust policy for a diverse ensemble of environments in a time frame comparable to the time it takes to master a single instance of the task. On the other hand, DAPG is still able to succeed in this setting and generates even more robust policies, as shown in Figure 9.16.

8.6 Conclusion

In this work, we developed a set of manipulation tasks representative of the types of tasks we encounter in everyday life. The tasks involve the control of a 24-DoF five-fingered hand. We also propose a method, DAPG, for incorporating demonstrations into policy gradient methods. Our empirical results compare model-free RL from scratch using two state-of-the-art DRL methods, DDPG and NPG, as well their demonstration-based counterparts, DDPGfD and our DAPG algorithm. We find that NPG is able to solve these tasks, but only after significant manual reward shaping. Furthermore, the policies learned under these shaped rewards are not robust, and produce idiosyncratic and unnatural motions. After incorporating human demonstrations, we find that our DAPG algorithm acquires policies that not only exhibit more human-like motion, but are also substantially more robust. Furthermore, we find that DAPG can be up to 30x more sample efficient than RL from scratch with shaped rewards. In the following chapter, we show how DAPG can be easily applied to train dexterous manipulation tasks on real hardware.

Chapter 9

Applying Bootstrapped On-Policy RL to Real World Robotic Systems

In the previous chapter, we introduced a novel on-policy reinforcement learning algorithm that is able to bootstrap from human demonstrations to solve much more complex tasks than previously possible in simulation, through a combination of reinforcement learning and imitation learning. In this chapter, we discuss how this algorithm can be applied to real world dexterous manipulation tasks, solving complex problems relatively quickly in the real world with a 3 fingered robotic hand.

9.1 Contributions

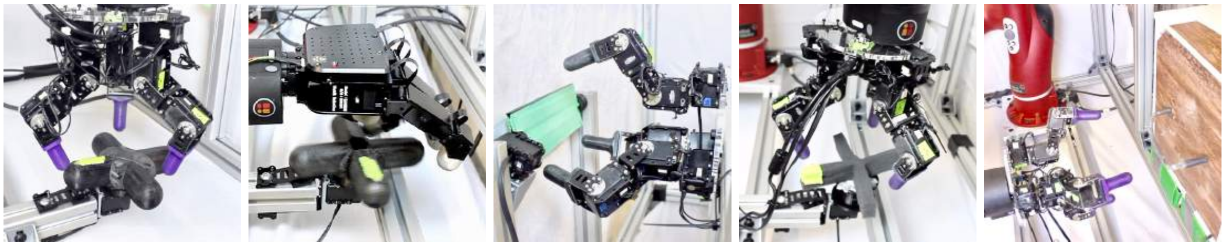


Figure 9.1: We demonstrate that DRL can learn a wide range of dexterous manipulation skills with multi-fingered hands, such as opening door with flexible handle, rotating a cross-shaped valve, and rotating the same valve but with a deformable foam handle, which presents an additional physical challenge, and box flipping.

In this work, we study how the model-free RL techniques we discussed in Chapter 8 can be scaled up to learn a variety of manipulation behaviors with multi-fingered hands directly in the real world, using general-purpose neural network policy representations and without manual controller or policy class design. We conduct our experiments with low cost multi-fingered manipulators, and show results on tasks such as rotating a valve, flipping a block vertically, and door opening. Somewhat surprisingly, we find that successful neural network controllers for each of these tasks can be trained directly in the real world in about 4-7 hours for most tasks.

We also show that we can further accelerate the learning process by incorporating a small number of human demonstrations, building on the recently proposed DAPG [95] algorithm. Using 20 demonstrations obtained through kinesthetic teaching, the learning time can be brought down to around $2 - 3$ hours which corresponds to a 2x speedup. Together, these results establish that model-free deep RL and demonstration-driven acceleration provide a viable approach for real-world learning of diverse manipulation skills.

The main contribution of this work is to demonstrate that model-free deep reinforcement learning can learn contact rich manipulation behaviors directly for low-cost multi-fingered hands directly in the real world. We demonstrate this on two different low cost robotic hands, manipulating both rigid and deformable objects, and performing three distinct tasks. We further demonstrate that a small number of demonstrations can accelerate learning, and analyze a number of design choices and performance parameters.

9.2 Hardware Setup

In order to demonstrate the generalizable nature of the model-free deep RL algorithms, we consider two different hardware platforms: a custom built 3 fingered hand, referred to as the Dynamixel claw (Dclaw), and a 4 fingered Allegro hand. Both hands are relatively cheap, especially the Dclaw, which costs under $\$2,500$ to build. For several experiments, we also mounted the hands on a Sawyer robot arm to allow for a larger workspace.

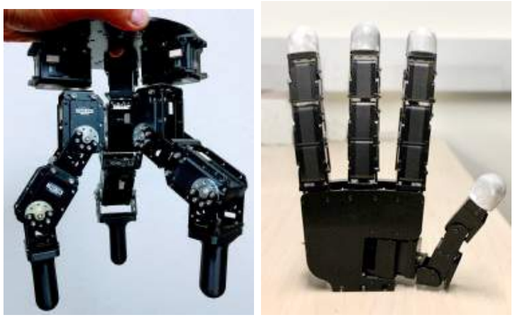


Figure 9.2: Left: 3 finger Dynamixel claw. Right: 4 finger anthropomorphic Allegro hand

Dynamixel Claw The Dynamixel claw (Dclaw) is custom built using Dynamixel servo motors. It is a powerful, low latency, position controlled 9 DoF manipulator which costs under $\$2,500$ to construct. Dclaw is robust and is able to run up to 24 hours without intervention or hardware damage.

Allegro Hand The Allegro hand is a 4 fingered anthropomorphic hand, with 16 degrees of freedom, and can handle payloads of up to 5 kg. This hand uses DC motors for actuation and can be either torque or position controlled using a low level PID. The Allegro hand costs on the order of $\$15,000$.

9.3 Tasks

While the approach we describe for learning dexterous manipulation is general and broadly applicable, we consider three distinct tasks in our experimental evaluation - valve rotation, box flipping, and door opening. These tasks involve challenging contact patterns and coordination, and are inspired by everyday hand manipulations. We approach these problems using reinforcement learning, modeling them as Markov decision processes (MDPs), which provide a generic mathematical abstraction to model sequential decision making problems. The goal in reinforcement learning is to learn a control policy which maximizes a user-provided reward function. This reward function is defined independently for each of our tasks as described below.

Valve Rotation This task involves turning a valve or faucet to a target position. The fingers must cooperatively push and move out of the way, posing an exploration challenge. Furthermore the contact forces with the valve complicate the dynamics. For our task, the valve must be rotated from $0°$ to $180°$.

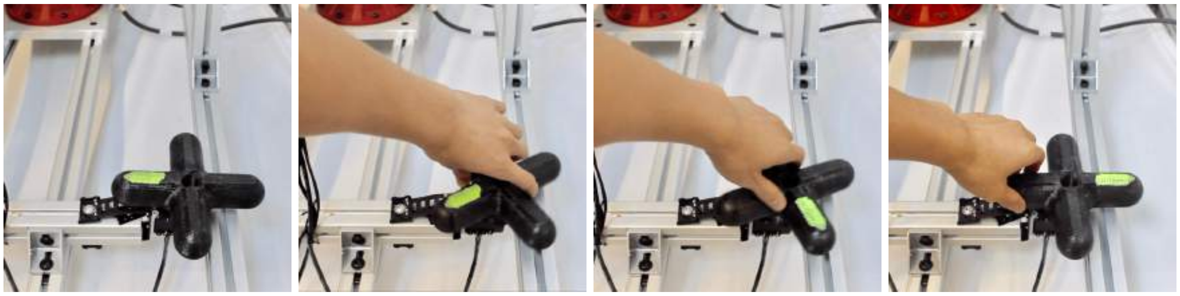


Figure 9.3: Illustration of valve rotation

The state space consists of all the joint angles of the hand, the current angle of rotation of the valve [θ_{valve}], the distance to the goal angle [$d\theta$], and the last action taken. The action space is joint angles of the hand and the reward function is

$$r = -|d\theta| + 10 * \mathbb{1}_{\{|d\theta|<0.1\}} + 50 * \mathbb{1}_{\{|d\theta|<0.05\}}$$
$$d\theta := \theta_{\text{valve}} - \theta_{\text{goal}}$$

We define a trajectory as a success if $|d\theta| < 20°$ for at least 20% of the trajectory.

Vertical box flipping This task involves rotating a rectangular box, which freely spins about its long axis, from $0°$ to $180°$. This task also involves learning alternating coordinated motions of the fingers such that while the top finger is pushing, the bottom two move out of the way, and vice versa.

The state space consists of all the joint angles of the hand, the current angle of rotation of the box [θ_{box}], the distance in angle to the goal, and the last action taken. The action space consists of

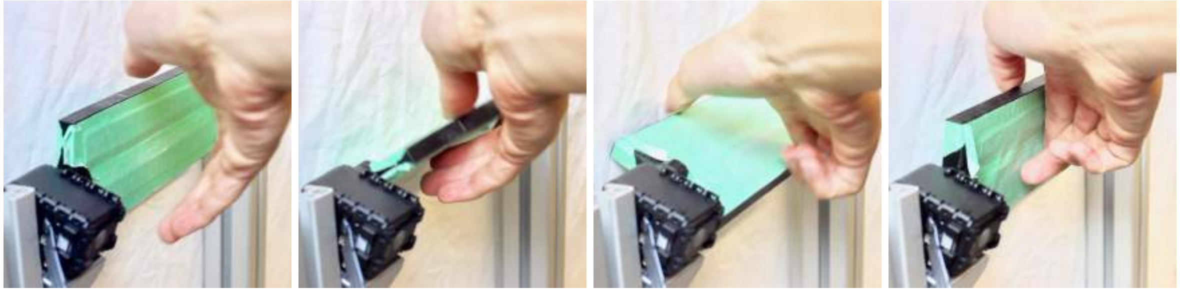


Figure 9.4: Illustration of box flipping

the joint angles of the hand and the reward function is

$$r = -|d\theta| + 10 * \mathbb{1}_{\{|d\theta|<0.1\}} + 50 * \mathbb{1}_{\{|d\theta|<0.05\}}$$
$$d\theta := \theta_{\text{box}} - \theta_{\text{goal}}$$

We define a trajectory as a success if $|d\theta| < 20°$ for at least 20% of the trajectory.

Door opening This task involves both the arm and the hand working in tandem to open a door. The robot must learn to approach the door, grip the handle, and then pull backwards. This task has more degrees of freedom given the additional arm, and involves the sequence of actions: going to the door, gripping the door, and then pulling away.

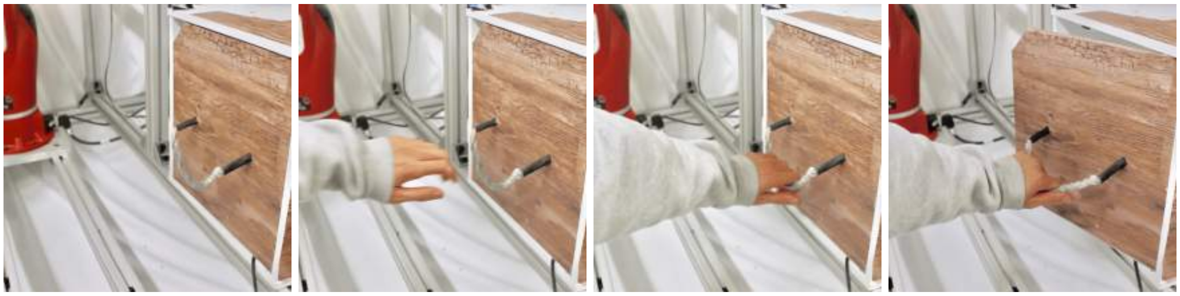


Figure 9.5: Opening door with flexible handle

The state space is all the joint angles of the hand, the Cartesian position of the arm, the current angle of the door, and last action taken. The action space is the position space of the hand and horizontal position of the wrist of the arm. The reward function is provided as

$$r = -(d\theta)^2 - (x_{\text{arm}} - x_{\text{door}})$$
$$d\theta := \theta_{\text{door}} - \theta_{\text{closed}}$$

We define a trajectory as a success if at any point $d\theta > 30°$.

Dynamixel Driven State Estimation

For these tasks, we solve both the problem of state estimation and resetting using a setup with objects augmented with Dynamixel servo motors. These motors serve the dual purpose of resetting objects (such as the valve, box, or door) to their original positions and measuring the state of the object (angle of rotation or position).

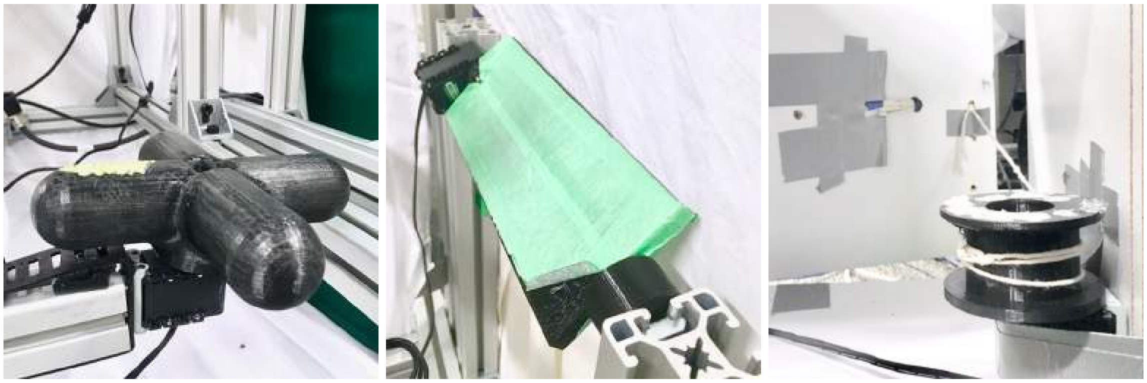


Figure 9.6: Dynamixel driven sensing and reset mechanisms. Left to Right: rigid valve, box, door.

9.4 Experimental Results and Analysis

The goal of our experiments is to empirically address the following research questions:

- Can model-free deep RL provide a practical means for learning various dexterous manipulation behaviors directly in the real world?
- Can model-free deep RL algorithms learn on different hardware platforms and on different physical setups?
- Can we accelerate the learning process using a small number of demonstrations obtained through kinesthetic teaching?
- How do particular design choices of the reward function and actuation space affect learning?

To do so, we utilize the tasks in Section 9.3. Additional details can be found at the supplementary website `https://sites.google.com/view/deeprl-handmanipulation`

Model-Free Deep RL

First, we explore the performance of model-free deep reinforcement learning on our suite of hardware tasks. The learning progress is depicted in Fig 9.10 and also in the accompanying video. We find that, somewhat surprisingly, model-free deep RL can acquire coherent manipulation skills in the time scales of a few hours (7 hours for turning a valve, 4 hours to flip a box, 16 hours for opening a door). These training times are evaluated once the deterministic policy achieves 100% success rate over 10 evaluation rollouts, according to the success metrics defined in Section 9.3 (Fig 9.15). The algorithm is robust and did not require extensive hyperparameter optimization.

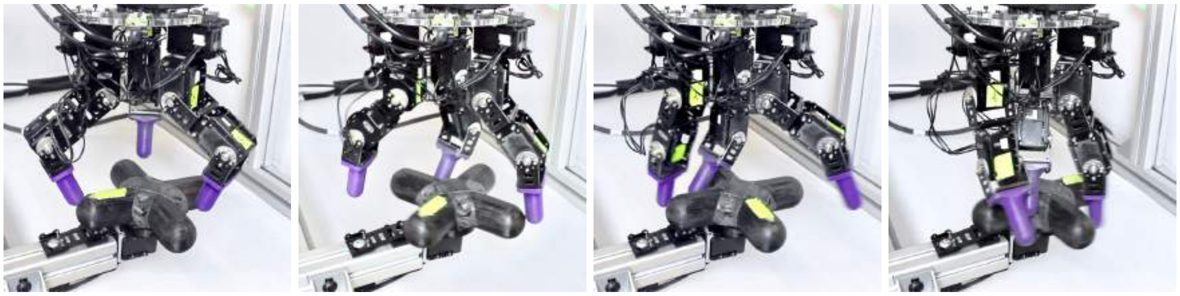


Figure 9.7: Visualization of Dclaw policy that has learned to turn a valve after 7 hours of training. The fingers learn to alternately move in and out to turn the valve.

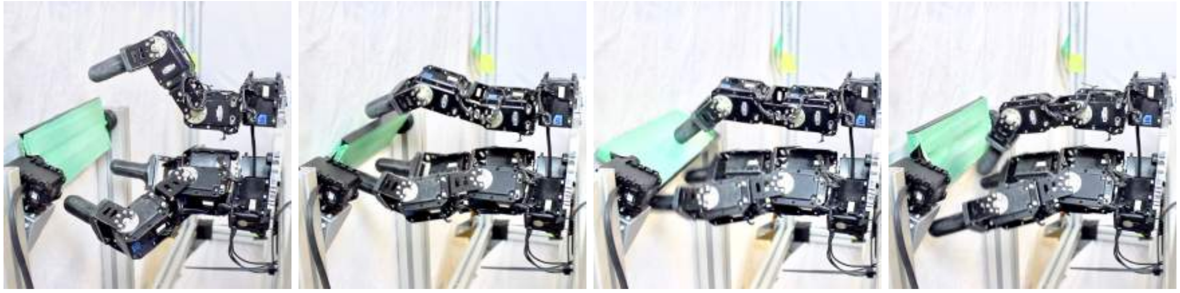


Figure 9.8: Visualization of Dclaw that has learned to flip a box after 4 hours of training. The Dclaw learns to extend its bottom two fingers and push its top finger forwards, then lower its bottom two fingers while pushing downwards with its top finger.

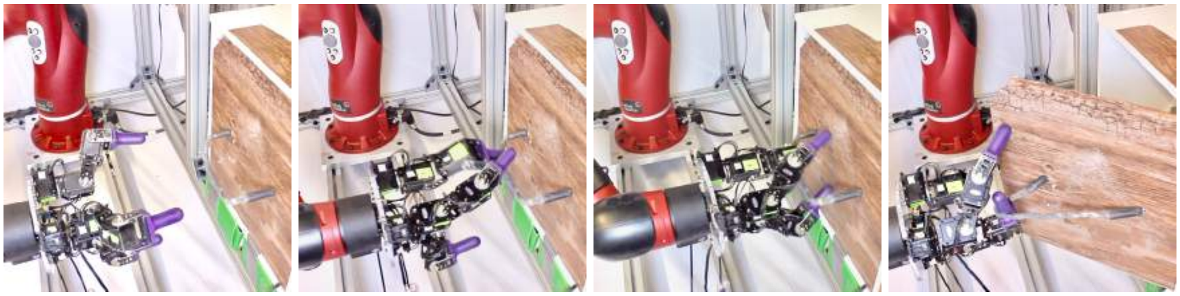


Figure 9.9: Visualization of a policy that has learned to open a door after 16 hours of training. The robot learns to move towards the door, grasp the deformable handle, and then pull the door open.

The only hyperparameter that was tuned was the initial variance of the policy for exploration. We analyze specific design choices in the reward function and actuation scheme in Section 9.4.

We find that for the valve and the box flipping, the learning is able to monotonically improve on the continuous reward signal, whereas for the door the learning is more challenging, given that reward is only obtained when the door is actually opened. The agent has to consistently pull open the door in order to see the reward, leading to the large spikes in learning as seen in Fig. 9.10.

The learned finger gaits that we observe do have interesting coordination patterns. The tasks require that the fingers move quickly and in a coordinated way so as to rotate the objects and then move out of the way. This behavior can be appreciated in the accompanying video on the supplementary website.

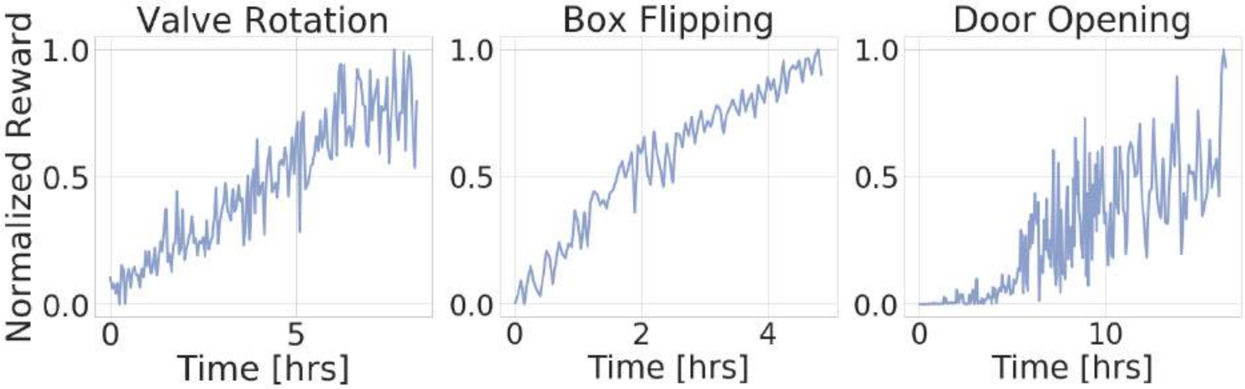


Figure 9.10: Learning progress with model-free RL from scratch using the NPG algorithm. The policies reach 100% average success rates after 7.4 hours for valve rotation and 4.1 hours for box flipping and 15.6 hours for door opening.

Learning on Different Hardware and Different Materials

To illustrate the ability of model-free RL algorithms to be applied easily to new scenarios and robots, we evaluated the valve task using the exact same deep RL algorithm with a different hand – the 4-fingered Allegro hand. This hand has 16 DoFs and is also anthropomorphic. We find that the Allegro hand was able to learn this task, as illustrated in Fig. 9.13, in comparable time as the Dclaw.

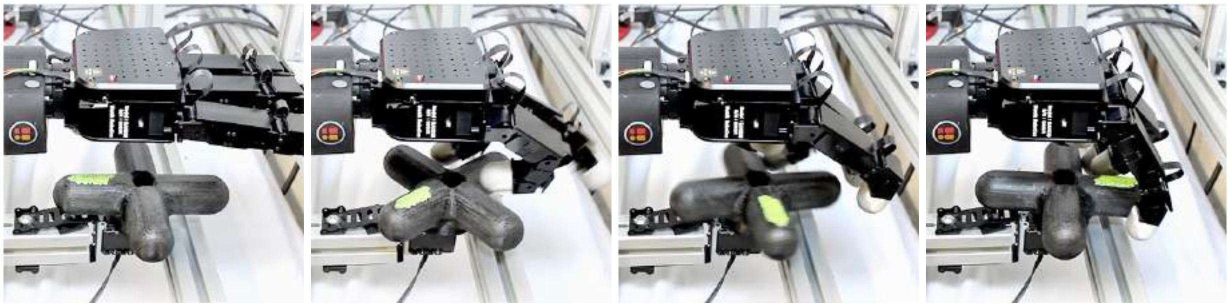


Figure 9.11: The Allegro hand learns to rotate a valve 180°.

Both systems are able to quickly learn the right behavior with the same hyperparameters. While morphology does indeed have an effect on rate of learning, we find that it is not a hindrance to eventually learning the task. The easy adaptability of these algorithms is extremely important, since we don't need to construct an accurate simulation or model of each new robot, and can simply run the same algorithm repeatedly to learn new behaviors.

Besides changing the robot's morphology, we can also modify the object that is manipulated. We evaluate whether model-free RL algorithms can be effective at learning with a different valve material, such as soft foam (Fig. 9.12). The contact dynamics with such a deformable material are hard to simulate and the hand can deform the valve in many different directions, making the actual manipulation task challenging. We see that model-free RL is able to learn manipulation with the foam valve effectively, even generating behaviors that exploit the deformable structure of the object.

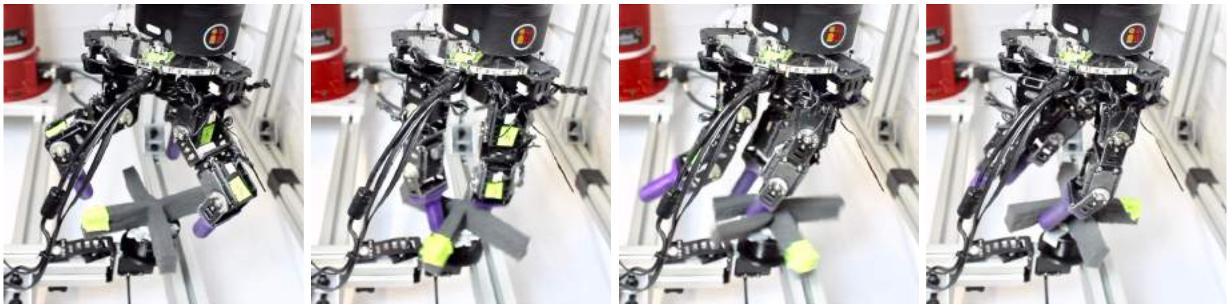


Figure 9.12: DClaw learns to rotate a foam valve despite its deformable structure. The claw learns to focus its manipulation on the center of the valve where there is more rigidity.

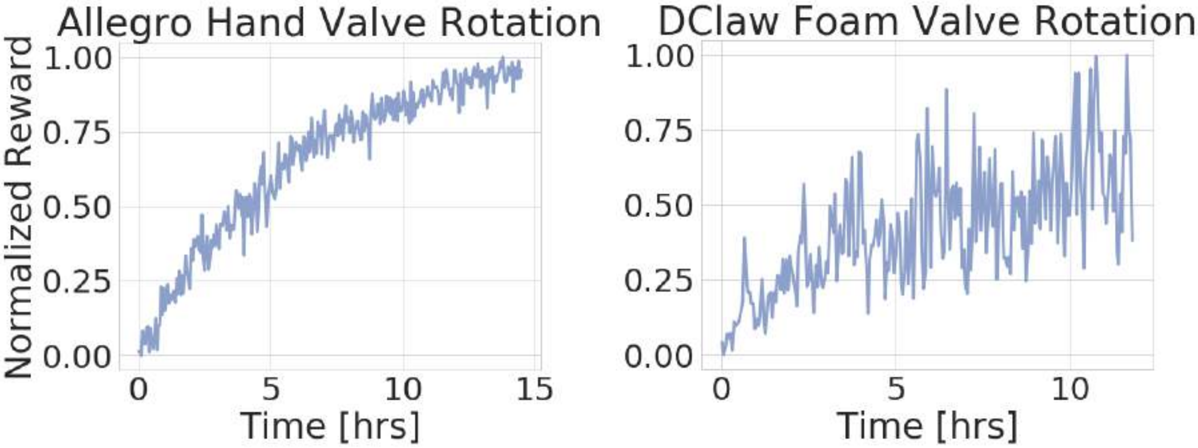


Figure 9.13: Left: learning progress of Allegro hand rotating rigid screw. Right: learning progress of Dclaw rotating the foam valve. The rewards have been normalized such that a random policy achieves score of 0 and the final trained policy achieves a score of 1.

Accelerating Learning with Demonstrations

While we find that model-free deep RL is generally practical in the real world, the number of samples can be further reduced by employing demonstrations. In order to understand the role of demonstrations, we collected 20 demonstrations for each task via kinesthetic teaching.

These demonstrations are slow and suboptimal, but can still provide guidance for exploration and help guide the learning process. With only a few demonstrations, we see that demonstration augmented policy gradient (DAPG) can speed up learning significantly, dropping learning time by 2x (Fig 9.14). We record training times across tasks using the previously defined success metrics in (Fig 9.15).

The efficiency of DAPG comes from the fact that the behavior cloning initialization gives the agent a rough idea of how to solve the task, and the augmented loss function guides learning through several iterations. The behaviors learned are also more gentle and legible to humans than behaviors learned via training from scratch, which is clearly displayed in the accompanying video.

To better understand the learned behaviors, we also evaluated the robustness of these behaviors to variations in the initial position of the valve, and to noise injected into actions and observations (Fig 9.16).

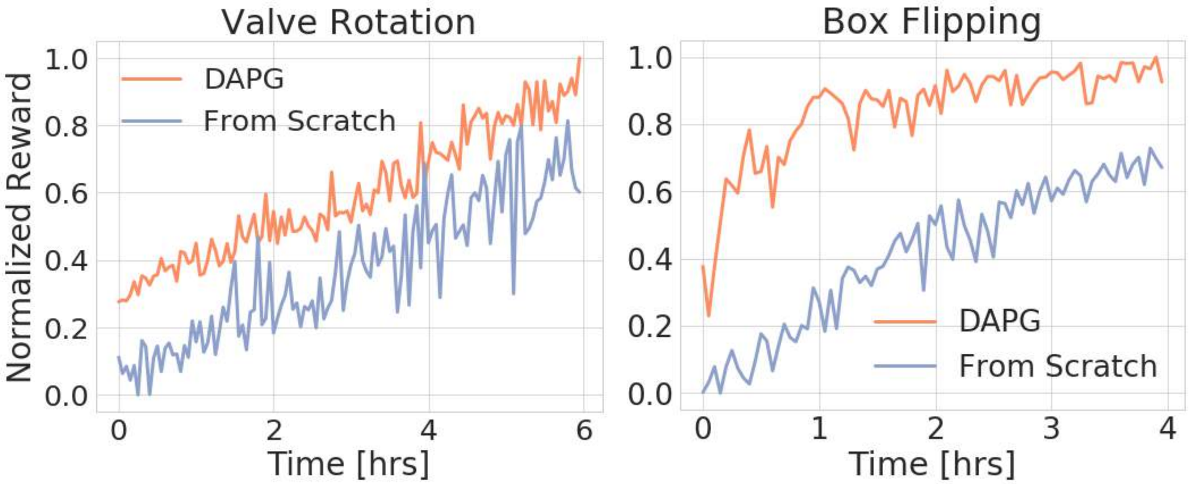


Figure 9.14: Learning progress with training from scratch using NPG and using DAPG. The performances have been normalized such that a random policy achieves a score of 0 and the best DAPG policy gets a score of 1.

Figure 9.15: Training Times Across Tasks [hrs]. Training time is determined using the success metrics defined in Section IV. A training run is complete once the deterministic policy achieves 100% success rate over 10 evaluation rollouts.

Task	From Scratch	DAPG
Valve	7.4	3.0
Box	4.1	1.5
Door	15.6	—–

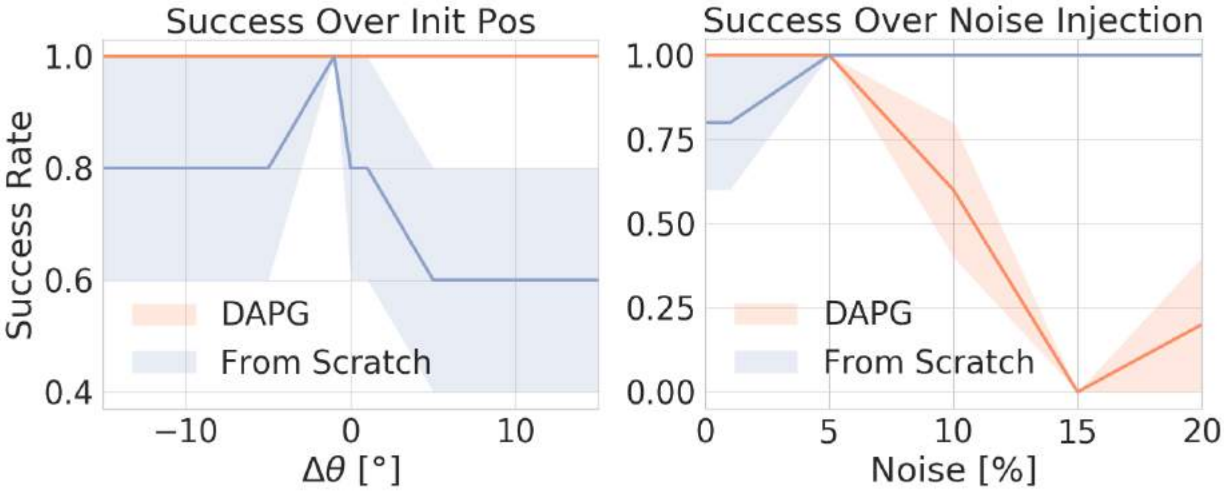


Figure 9.16: Plots showing robustness of DAPG vs learning from scratch for the valve rotation task. Left: Variation of success with change in valve initial position (degrees). Right: Variation of success with x% uniformly random noise injected into the observation and action space. DAPG is more robust with change in initial valve position, but is less robust as we add more noise.

We also considered collecting demonstrations from a wider range of initial configurations of the environment, and using this wider demo set for DAPG. The demonstrations were collected with initial valve positions in the range $[-\frac{\pi}{4}, \frac{\pi}{4}]$. This paradigm also works well (Fig 9.17). Unsurprisingly, it is not as effective as using a number of demonstrations in the same environment configuration, but is still able to learn well. (Fig 9.14).

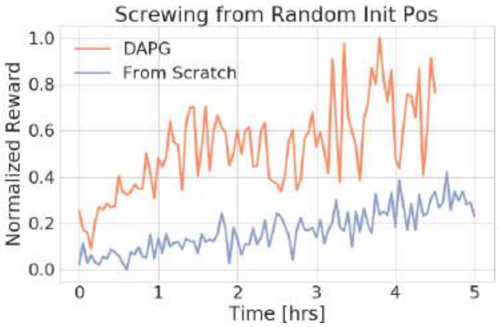


Figure 9.17: Training DClaw to turn a valve from a single randomly sampled initial position between $[-45°, 45°] to 180°$. DAPG was trained with 20 demos that turned the valve from initial positions sampled uniformly at random between $[-45°, 45°]$ to $180°$.

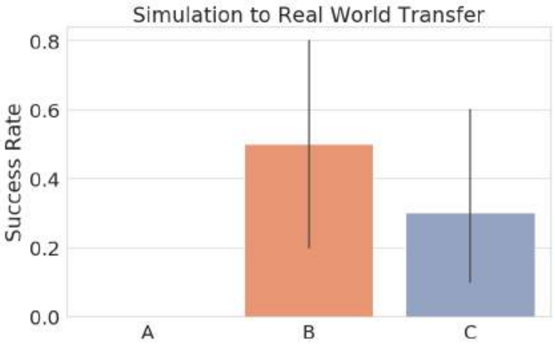


Figure 9.18: Success rates using different sim2real transfer strategies for valve turning with Dclaw. A: No domain randomization. B: Randomization of position control PID parameters and friction. C: Same as B, but also including the previous action as part of the state space.

Performance with Simulated Training

While the main goal of this work is to study how model-free RL can be used to learn complex policies directly on real hardware, we also evaluated training in simulation and transfer, employing randomization to allow for transfer [43], [315]. This requires modeling the task in a simulator and manually choosing the parameters to randomize.

We see in Fig. 9.18 that the randomization of PID parameters and friction is crucial for effective transfer.

While simulation to real transfer enabled by randomization is an appealing option, especially for fragile robots, it has a number of limitations. First, the resulting policies can end up being overly conservative due to the randomization, a phenomenon that has been widely observed in the field of robust control. Second, the particular choice of parameters to randomize is crucial for good results, and insights from one task or problem domain may not transfer to others. Third, increasing the amount of randomization results in more complex models tremendously increasing the training time and required computational resources, as discussed in Section 4.2. Directly training in the real world may be more efficient and lead to better policies. Finally, and perhaps most importantly, an accurate simulator must be constructed manually, with each new task modeled by hand in the simulation, which requires substantial time and expertise. For tasks such as valve rotation with the foam valve or door opening with a soft handle, creating the simulation itself is very challenging.

Design Choices

To understand the design choices needed to effectively train dexterous hand manipulation systems with model-free RL, we analyzed different factors which contribute to learning. We performed this analysis in simulation in order to choose the right schemes for real world training.

Choices of Action Space

The choice of actuation space often makes an impact on learning progress . For hand manipulation it also greatly affects the smoothness, and hence sustainability, of the hardware. In our results, we end up using position control since it induces the fewest vibrations and is easiest to learn with.

In order to better understand the rationale behind this, we consider a comparison between using controlling position and torque controllers as well as their higher order derivatives. We compare the vibrations induced by each of these control schemes by measuring the sum of the magnitudes of the highest Fourier coefficients of sample trajectories (joint angles) induced by random trajectories.

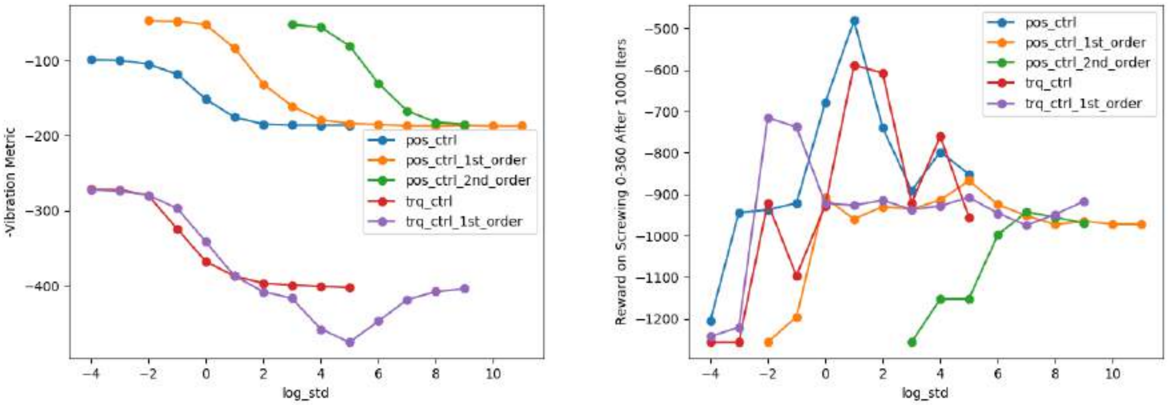


Figure 9.19: Left: Analysis of vibrations induced by different actuation schemes in simulation. Higher metric indicates lower vibrations. We find that position control is able to induce significantly lower vibrations than torque control, making it safer to run on hardware. Right: Analysis of rewards attained in simulation after training using the control scheme on Dclaw valve rotating task.

We see that position control has the lowest vibration amongst all the choices of control schemes, and is also able to achieve significantly better performance. This is likely because we are using a stabilizing PID control at the low level to do position control which reduces the load on the learning algorithm. We also see that it is harder to learn a policy when controlling higher order derivatives, and that it is easier to learn with position control than with torque control.

Impact of Reward Function

We also investigated the effect of the reward function on learning progress. To provide some intuition about the different choice of reward functions, we show a comparison between 3 different reward functions for learning. We evaluate learning progress for these three types of reward functions in simulation, as a means for choosing an appropriate form of reward for real world training.

1. $r_1 = -\|\theta - \theta_{goal}\|_2$
2. $r_2 = r_1 + 10 * \mathbb{1}_{\{r_1 < 0.1\}} + 50 * \mathbb{1}_{\{r_1 < 0.05\}}$
3. $r_3 = r_2 - \|v\|_2$

We find that learning is most effective with using either option 1 or 2, and is slower with a control cost. The control cost ensures smoother operation, but at the cost of efficiency in learning, since optimization of control penalties results in reduction of exploration. In real world experiments we found that training without a control cost still produced safe behaviors.

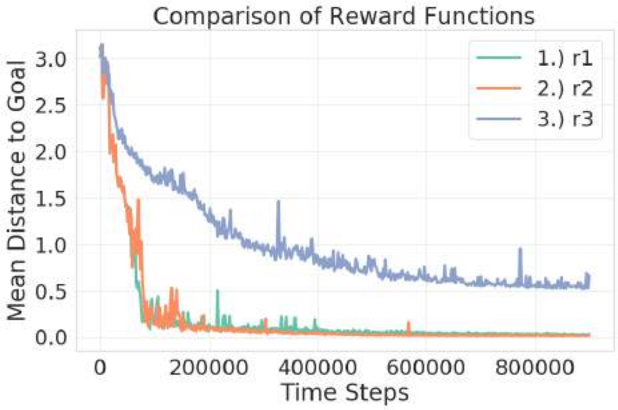


Figure 9.20: Analysis of learning progress in simulation with different reward functions.

9.5 Discussion and Future Work

In this work, we study real-world model-free reinforcement learning as a means to learn complex dexterous hand manipulation behaviors. We show that model-free deep RL algorithms can provide a practical, efficient, and general method for learning with high dimensional multi-fingered hands. Model-free RL algorithms can easily be applied to a variety of low-cost hands, and solve challenging tasks that are hard to simulate accurately. This kind of generality and minimal manual engineering may be a key ingredient in endowing robots with the kinds of large skill repertoires they need to be useful in open-world environments such as homes, offices, and hospitals. We also show that the sample complexity of model-free RL can be substantially reduced with suboptimal kinesthetic demonstrations, while also improving the resulting motion quality. In the following chapter, we show that the paradigm of DAPG can be extended with hierarchical policy representations to solve long horizon manipulation problems.

Chapter 10

Bootstrapping Hierarchical Reinforcement Learning with Human Demonstrations for Long Horizon Reasoning

In the previous two chapters, we introduced a novel algorithm for bootstrapping from human demonstrations and showed the ability of this algorithm to solve complex dexterous manipulation tasks both in simulation and in the real world. However, the tasks that are being solved are still relatively short horizon, atomic tasks. Several tasks that we expect agents to perform in the real world are long horizon tasks with multiple sub-steps, requiring temporally extended reasoning. In this chapter we show that human data can also be used to enable agents to perform long horizon behaviors, building on the ideas from DAPG with hierarchical policy representations and goal conditioned reinforcement learning. We discuss the structure of this algorithm and show it's ability to solve complex long horizon tasks.

10.1 How Can Demonstration Bootstrapped RL Solve Long Horizon Tasks?

Figure 10.1: RPL learns complex, long-horizon manipulation tasks

Recent years have seen reinforcement learning (RL) successfully applied to a number of robotics tasks such as in-hand manipulation [290], grasping [316] and door opening [35]. However, these applications have been largely constrained to relatively simple short-horizon skills. Hierarchical reinforcement learning (HRL) [317] has been proposed as a potential solution that should scale to challenging long-horizon problems, by explicitly introducing temporal abstraction. However, HRL methods have traditionally struggled due to various practical challenges such as exploration [318], skill segmentation [319] and reward definition [232]. We can simplify the above-mentioned problems by utilizing extra supervision in the form of unstructured human demonstrations, in which case the question becomes: how should we best use this kind of demonstration data to make it easier to solve long-horizon robotics tasks?

This question is one focus area of hierarchical imitation learning (HIL), where solutions [320], [321] typically try to achieve two goals: i) learn a temporal task abstraction, and ii) discover a meaningful segmentation of the demonstrations into subtasks. These methods have not traditionally been tailored to further RL fine-tuning, making it challenging to apply them to a long-horizon setting, where pure imitation is very likely to fail. To address this need, we devise a simple and universally-applicable two-phase approach that in the first phase pre-trains hierarchical policies using demonstrations such that they can be easily fine-tuned using RL during the second phase. In contrast to HRL methods, our method takes advantage of unstructured demonstrations to bootstrap further fine-tuning, and in contrast to conventional HIL methods, it does not focus on careful subtask segmentation, making the method simple, general and very amenable to further reinforcement fine-tuning. In particular, we show that we can develop an imitation and reinforcement learning approach that while not necessarily perfect at imitation learning, is very amenable to improvement via fine-tuning with reinforcement learning and that can be scaled to challenging long-horizon manipulation tasks.

What are the advantages of using such an algorithm? First, the approach is very general, in that it can be applied to any demonstration data, including easy to provide unsegmented, unstructured and undifferentiated demonstrations of meaningful behaviors. Second, our method does not require any explicit form of skill segmentation or subgoal definition, which otherwise would need to be learned or explicitly provided. Lastly, and most importantly, since our method ensures that every low-level trajectory is goal-conditioned (allowing for a simple reward specification) and of the same, limited length, it is very amenable to reinforcement fine-tuning, which allows for continuous policy improvement. We show that relay policy learning allows us to learn general, hierarchical, goal-conditioned policies that can solve long-horizon manipulation tasks in a challenging kitchen environment in simulation, while significantly outperforming hierarchical RL algorithms and imitation learning algorithms.

10.2 Relationship to Prior Work

Typical solutions for solving temporally extended tasks have been proposed under the HRL framework [317]. Solutions like the options framework [181], [319], HAM [322], max-Q [323], and feudal networks [183], [324] present promising algorithmic frameworks for HRL. A particularly promising approach was proposed in [325] and [326], using goal conditioned policies at multiple layers of hierarchy for RL. Nevertheless, these algorithms still suffer from challenges in exploration and optimization (as also seen in our experimental comparison with [325]), which have limited their application to general robotic problems. In this work, we tackle these problems by using additional supervision in the form of unstructured, unsegmented human demonstrations. Our work builds on goal-conditioned RL [118], [119], [240], [327], which has been explored in the context of reward-free learning [30], learning with sparse rewards [119], large scale generalizable imitation learning [249], and hierarchical RL [325]. We build on this principle to devise a general-purpose imitation and RL algorithm that uses data relabeling and bi-level goal conditioned policies to learn complex skills.

There has a been a number of hierarchical imitation learning (HIL) approaches [321], [328]–[331] that typically focus on extracting transition segments from the demonstrations. These methods aim to perform imitation learning by learning low-level primitives [321], [331] or latent conditioned policies [328] which meaningfully segment the demonstrations. Traditionally, these approaches do not aim to and are not amenable to improving the learned primitives with subsequent RL, which is necessary as we move towards multi-task, challenging long-horizon problems where pure imitation might be insufficient. In this work, we specifically focus on utilizing both imitation and RL, and devise a method that does not explicitly try to segment out individual primitives into meaningful subtasks, but instead splits the demonstration data into fixed-length segments, amenable to fine-tuning with reinforcement learning. This allows us to leverage relabeling across different goals [118], [119], [240], [327]. We introduce a novel form of goal relabeling and demonstrate its efficiency when applied to learning robust bi-level policies. A related idea is presented in [332], where the authors assume that an expert provides labelled and segmented demonstrations at both levels of the hierarchy, with an interactive expert for guiding RL. In contrast, we use a pool of unlabelled demonstrations and apply our method to learn a policy to achieve various desired goals, without needing interactive guidance or segmentation.

10.3 Relay Policy Learning

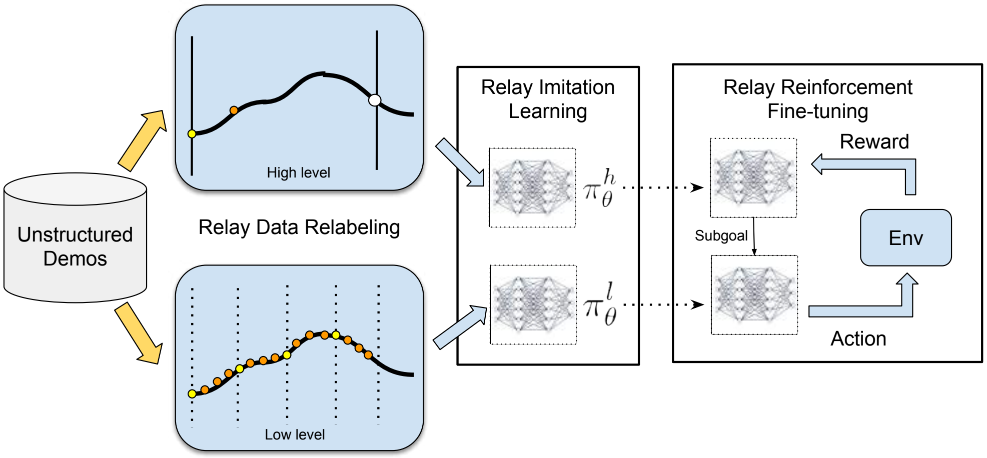


Figure 10.2: **Relay policy learning:** the algorithm starts with relabelling unstructured demonstrations at both the high and the low level of the hierarchical policy and then uses them to perform relay imitation learning. This provides a good policy initialization for subsequent relay reinforcement fine-tuning. We demonstrate that learning such simple goal-conditioned policies at both levels from demonstrations using relay data relabeling, combined with relay reinforcement fine-tuning allows us to learn complex manipulation tasks.

In this section, we describe our proposed relay policy learning (RPL) algorithm, which leverages unstructured demonstrations and reinforcement learning to solve challenging long-horizon tasks.

Our approach consists of two phases: relay imitation learning (RIL), followed by relay reinforcement fine-tuning (RRF) described in Sec. 10.3 and 10.3 respectively. While RIL by itself is not able to solve the most challenging tasks that we consider, it provides a very effective initialization for fine-tuning.

Relay Policy Architecture

We first introduce our bi-level hierarchical policy architecture (shown in Fig 10.3), which enables us to leverage temporal abstraction. This architecture consists of a high-level goal-setting policy and a low-level subgoal-conditioned policy, which together generate an environment action for a given state. The high-level policy $\pi_\theta^h(s_g^l|s_t, s_g^h)$ takes the current state s_t and a long-term high-level goal s_g^h and produces a subgoal $s_g^l \in \mathcal{S}$ which is then ingested by a low-level policy $\pi_\phi^l(a|s_t, s_g^l)$. The low-level policy takes the current state s_t, and the subgoal s_g^l commanded by the high-level policy and outputs an action a_t, which is executed in the environment.

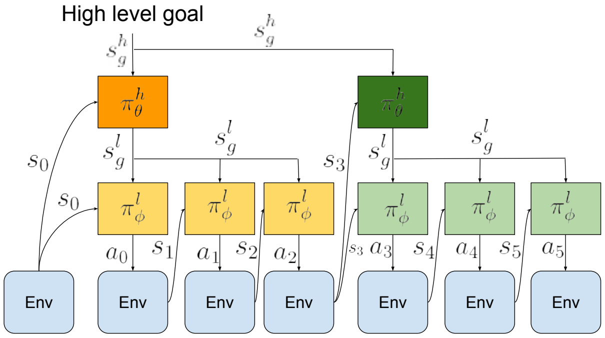


Figure 10.3: **Relay policy architecture:** A high level goal setter π_θ takes high level goal s_g^h and sets goals s_g^l for a lower level policy π_ϕ, which acts for a fixed time horizon before resampling s_g^l

Importantly, the goal setting policy π_θ^h makes a decision every H time steps (set to 30 in our experiments), with each of its subgoals being kept constant during that period for the low-level policy, while the low-level policy π_ϕ^l operates at every single time-step. This provides temporal abstraction, since the high level policy operates at a coarser resolution than the low-level policy. This policy architecture, while inspired by goal-conditioned HRL algorithms [325], requires a novel learning algorithm to be applicable in the context of imitation learning, which we describe in Sec. 10.3. Given a high-level goal s_g^h, π_θ^h samples a subgoal s_{g0}^l, which is passed to π_θ^l to generate action a_0. For the subsequent H steps, the goal produced by π_θ^h is kept fixed, while π_θ^l generates an action a_t at every time step.

Relay Imitation Learning

Our problem setting assumes access to a pool of unstructured, unlabeled “play" demonstrations ([249]) $\mathcal{D}$, corresponding to demonstrations of meaningful activities provided by the user, without any particular task in mind, e.g. opening cabinet doors, playing with different objects, or simply tidying up the scene. We do not assume that this demonstration data actually accomplishes any of the final task goals that we will need to solve at test-time, though we do need to assume that the test-time goals come from the same distribution of goals as those accomplished in the demonstration data. In order to take the most advantage of such data, we initialize our policy with our proposed relay imitation learning (RIL) algorithm. RIL is a simple imitation learning procedure that builds

Algorithm 7: Relay Policy Learning

```
Require: Unstructured pool of demonstrations
    $D = \{\tau_0, \tau_1, ...\tau_N\}$
Relabel goals in demonstration trajectories
    using Algorithm 8, 9 to extract $D_l, D_h$
Relay Imitation Learning: Train $\pi^h_\theta$ and $\pi^l_\phi$
    using Eqn 10.1
while not done do
    Collect on-policy experience with $\pi^h_\theta$ and $\pi^l_\phi$
        for high level goals different $s^h_g$
    [Optional] Relabel this experience
        (Sec. 10.3), and add to $D_l$, $D_h$
    Update the policy via policy gradient update
        using Eqn 10.2, 10.3.
end while
Distill fine-tuned policies into a single
    multi-goal policy
```

Algorithm 8: Relay data relabeling for RIL low level

```
Require: Demonstrations $D = \{\tau_0, \tau_1, ...\tau_N\}$
for $n = 1...N$ do
    for $t = 1...t_n$ do
        for $w = 1...W_l$ do
            Add $(s^n_t, a^n_t, s^n_{t+w})$ to $D_l$
        end for
    end for
end for
```

Algorithm 9: Relay data relabeling for RIL high level

```
Require: Demonstrations $D = \{\tau_0, \tau_1, ...\tau_N\}$
for $n = 1...N$ do
    for $t = 1...t_n$ do
        for $w = 1...W_h$ do
            Add $(s^n_t, s^n_{t+\min(w,W_l)}, s^n_{t+w})$ to $D_h$
        end for
    end for
end for
```

on the goal relabeling scheme described in [249] for the hierarchical setting, resulting in improved handling of multi-task generalization and compounding error. RIL assumes access to the pool of demonstrations consisting of N trajectories $\mathcal{D} = \{\tau^i, \tau^j, \tau^k, ...\}$, where each trajectory consists of state-action pairs $\tau^i = \{s^i_0, a^i_0, \ldots, s^i_T, a^i_T\}$. Importantly, these demonstrations can be attempting to reach a variety of different high level goals s^h_g, but we do not require these goals to be specified explicitly. To learn the relay policy from these demonstrations, we construct a low-level dataset $\mathcal{D}_l$, and a high-level dataset $\mathcal{D}_h$ from these demonstrations via "relay data relabeling", which is described below, and use them to learn π^h_θ and π^l_θ via supervised learning at multiple levels.

We construct the low-level dataset by iterating through the pool of demonstrations and relabeling them using our relay data relabelling algorithm. First, we choose a window size W_l and generate state-goal-action tuples for $\mathcal{D}_l$, (s, s^l_g, a) by goal-relabeling within a sliding window along the demonstrations, as described in detail below and in Algorithms 8, 9. The key idea behind relay data relabeling is to consider all states that are actually reached along a demonstration trajectory within W_l time steps from any state s_t to be goals reachable from the state s_t by executing action a_t. This allows us to label all states $s_{t+1}, ..., s_{t+W_l}$ along a valid demonstration trajectory as potential goals that are reached from state s_t, when taking action a_t. We repeat this process for all states s_t along all the demonstration trajectories being considered. This procedure ensures that the low-level policy is proficient at reaching a variety of goals from different states, which is crucial when the low-level policy is being commanded potentially different goals generated by the high-level policy.

We employ a similar procedure for the high level, generating the high-level state-goal-action dataset $\mathcal{D}_h$. However, the actions at the high level are subgoal states that are provided to the low-level policy, so they must be chosen as states along the demonstration trajectories. We start by choosing a high-level window size W_h, which encompasses the high-level goals we would like to eventually reach. We then generate state-goal-action tuples for $\mathcal{D}_h$, via relay data relabeling within the high-level window being considered, as described in Algorithm 8, 9. We also label all states $s_{t+1}, ..., s_{t+W_h}$ along a valid trajectory as potential high-level goals that are reached from state s_t by the high level policy, but we set the high-level action for a goal j steps ahead s_{t+j}, as $s_{t+\min(W_l,j)}$ choosing a sufficiently distant subgoal as the high-level action.

Given these relay-data-relabeled datasets, we train π_θ^l and π_θ^h by maximizing the likelihood of the actions taken given the corresponding states and goals:

$$\max_{\phi,\theta} \mathbb{E}_{(s,a,s_g^l)\sim D_l}[\log \pi_\phi(a|s, s_g^l)] + \mathbb{E}_{(s,s_g^l,s_g^h)\sim D_h}[\log \pi_\theta(s_g^l|s, s_g^h)]. \tag{10.1}$$

This procedure gives us an initialization for both the low-level and the high-level policies, without the requirement for any explicit goal labeling from a human demonstrator. As we show in our experiments, this bi-level initialization is significantly more amenable to RRF than learning the high level from scratch as described in [232], [249], [325], and allows us to avoid the expensive goal labeling that is required in [332]. Relay data relabeling not only allows us to learn hierarchical policies without explicit labels, but also provides algorithmic improvements to imitation learning: (i) it generates more data through the relay-data-relabelling augmentation, and (ii) it improves generalization since it is trained on a large variety of goals.

Relay Reinforcement Fine-tuning

The procedure described in Sec. 10.3 allows us to extract an effective policy initialization via relay imitation learning. However, this policy is often unable to perform well across all temporally extended tasks, due to the well-known compounding errors stemming from imitation learning [175]. Reinforcement learning provides a solution to this challenge, by enabling continuous improvement of the learned policy directly from experience. We can use RL to improve RIL policies via fine-tuning on different tasks. We employ a goal-conditioned HRL algorithm that is a variant of natural policy gradient (NPG) with adaptive step size [90], where both the high-level and the low-level goal-conditioned policies π_θ^h and π_ϕ^l are being trained with policy gradient in a decoupled optimization.

Given a low-level goal-reaching reward function $r_l(s_t, a_t, s_g^l)$, we can optimize the low-level policy by simply augmenting the state of the agent with the goal commanded by the high-level policy and then optimizing the policy to effectively reach the commanded goals by maximizing the sum of its rewards. For the high-level policy, given a high-level goal-reaching reward function $r_h(s_t, g_t, s_g^h)$, we can optimize it by running a similar goal-conditioned policy gradient optimization to maximize the sum of high-level rewards obtained by commanding the current low-level policy.

To effectively incorporate demonstrations into this reinforcement learning procedure, we leverage our method via: (1) initializing both π_θ^l and π_θ^h with the policies learned via RIL, and (2) encouraging policies at both levels to stay close to the behavior shown in the demonstrations. To incorporate (2), we augment the NPG objective with a max-likelihood objective that ensures that policies at both levels take actions that are consistent with the relabeled demonstration pools D_l and

D_h from Section 10.3, as described in Eqn 10.2 and 10.3:

$$\nabla_\phi J_l = \mathbb{E}[\nabla_\phi \log \pi^l_\phi(a|s, s^l_g) \sum_t r_l(s_t, a_t, s^l_g)] + \lambda_l \mathbb{E}_{(s,a,s^l_g)\sim\mathcal{D}_l}[\nabla_\phi \log \pi^l_\phi(a|s, s^l_g)] \tag{10.2}$$

$$\nabla_\theta J_h = \mathbb{E}[\nabla_\theta \log \pi^h_\theta(s^l_g|s, s^h_g) \sum_t r_h(s_t, s^l_g, s^h_g)] + \lambda_h \mathbb{E}_{(s,s^l_g,s^h_g)\sim\mathcal{D}_h}[\nabla_\theta \log \pi^h_\theta(s^l_g|s, s^h_g)]. \tag{10.3}$$

While a similar objective has been described in [95], [300], it is yet to be explored in the hierarchical, goal-conditioned scenarios, which makes a significant difference as indicated in our experiments.

In addition, since we are learning goal-conditioned policies at both the low and high level, we can leverage relay data relabeling as described in Sec. 10.3 to also enable the use of off-policy data for fine-tuning. Suppose that at a particular iteration i, we sampled N trajectories according to the scheme proposed in Sec. 10.3. While these trajectories did not necessarily reach the goals that were originally commanded, and therefore cannot be considered optimal for those goals, they do end up reaching the *actual* states visited along the trajectory. Thus, they can be considered as optimal when the goals that they were intended for are relabeled to states along the trajectory via relay data relabeling described in Algorithm 8, 9. This scheme generates a low-level dataset $\mathcal{D}^i_l$ and a high level dataset $\mathcal{D}^i_h$ by relabeling the trajectories sampled at iteration i. Since these are considered "optimal" for reaching goals along the trajectory, they can be added to the buffer of demonstrations D_l and D_h, thereby contributing to the objective described in Eqn 10.2 and Eqn 10.3 and allowing us to leverage off-policy data during RRF. We experiment with three variants of the fine-tuning update in our experimental evaluation: IRIL-RPL (fine-tuning with Eqn 10.2, 10.3 and iterative relay data relabeling to incorporate off-policy data as described above), DAPG-RPL (fine-tuning the policy with the update in Eqn 10.2, 10.3 without the off-policy addition) and NPG-RPL (fine-tuning the policy with the update in Eqn 10.2, 10.3, without the off-policy addition or the second maximum likelihood term). The overall method is described in Algorithm 7.

As described in [333], it is often difficult to learn multiple tasks together with on-policy policy gradient methods, because of high variance and conflicting gradients. To circumvent these challenges, we use RPL to fine-tune on a number of different high level goals individually, and then *distill* all of the learned behaviors into a single policy as described in [334]. This allows us to learn a single policy capable of achieving multiple high level goals, without dealing with the challenges of multi-task optimization.

10.4 Experimental Results

Our experiments aim to answer the following questions: (1) Does RIL improve imitation learning with unstructured and unlabelled demonstrations? (2) Is RIL more amenable to RL fine-tuning than its flat, non-hierarchical alternatives? (3) Can we use RPL to accomplish long-horizon manipulation tasks? Videos and further experimental details are available at `https://relay-policy-learning.githu`

Environment Setup To evaluate our algorithm, we utilize a challenging robotic manipulation environment modeled in MuJoCo, shown in Fig. 10.1. The environment consists of a 9 DoF position-controlled Franka robot interacting with a kitchen scene that includes an openable microwave, four turnable oven burners, an oven light switch, a freely movable kettle, two hinged cabinets, and a sliding cabinet door. We consider reaching different goals in the environment, as shown in Fig. 10.4, each of which may require manipulating many different components. For instance, in Fig. 10.4 (a), the robot must open the microwave, move the kettle, turn on the light, and slide open the cabinet. While the goals we consider are temporally extended, the setup is fully general. We collect a set of unstructured and unsegmented human demonstrations described in Sec. 10.3, using the PUPPET MuJoCo VR system [305]. We provide the algorithm with 400 sequences containing various unstructured demonstrations that each manipulate four different elements of the scene in sequence.

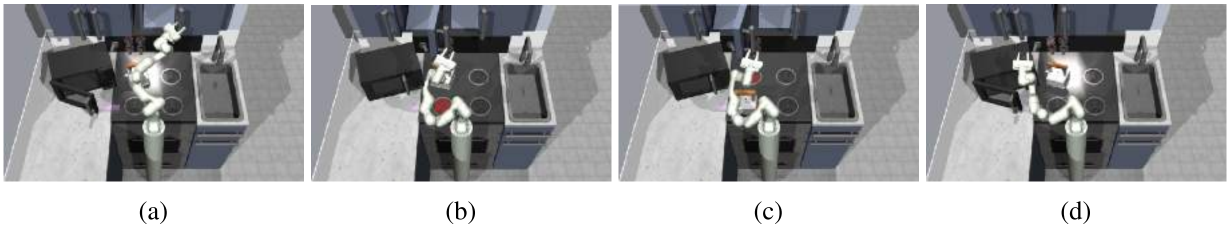


Figure 10.4: Examples of compound goals in the kitchen environment. Each goal has different elements manipulated, requiring multiple stages to solve: (a) microwave, kettle, light, slider, (b) kettle, burner, slider, cabinet, (c) burner, top burner, slide hinge, (d) kettle, microwave, top burner, lights

Evaluation and Comparisons Since each of our tasks consist of compound goals that involve manipulating four elements in the environment, we evaluate policies based on the number of steps that they complete out of four, which we refer to as step-completion score. A step is completed when the corresponding element in the scene is moved to within ϵ distance of its desired position.

We compare variants of our RPL algorithm to a number of ablations and baselines, including prior algorithms for imitation learning combined with RL and methods that learn from scratch. Among algorithms which utilize imitation learning combined with RL, we compare with several methods that utilize flat behavior cloning with additional finetuning. Specifically, we compare with (1) flat goal-conditioned behavior cloning followed by finetuning (*BC*), (2) flat goal-conditioned behavior cloning trained with data relabeling followed by finetuning (*GCBC*) [249], and variants of these algorithms that augment the *BC* and *GCBC* fine-tuning with losses as described in [95] - (3) *DAPG-BC* and (4) *DAPG-GCBC*. We also compare RPL to (5) hierarchical imitation learning + finetuning with an oracle segmentation scheme, which performs hierarchical goal conditioned imitation learning by using a hand-specified oracle to segment the demonstrations for imitation learning, followed by RRF style fine-tuning. Details of this scheme can be found in Appendix 3. For comparisons with methods that learn from scratch we compare with (6) an on-policy variant of *HIRO* [325] trained from scratch with natural policy gradient [90] instead of Q-learning and (7) a baseline (*Pre-train low level*) that learns low-level primitives from the demonstration data, and learns the high-level goal-setting policy from scratch with RL. The last baseline is representative of a class of HIL algorithms [328], [329], [331], which are difficult to fine-tune because it is not clear how to provide rewards for improving low-level primitives. Lastly, we compare RPL with a baseline (7) (*Nearest Neighbor*) which uses a nearest neighbor strategy to choose the demonstration which has the achieved goal closest to the commanded goal and subsequently executes actions open-loop.

Relay Imitation Learning from Unstructured Demonstrations

We start by aiming to understand whether RIL improves imitation learning over standard methods. We compare the step-wise completion scores averaged over 17 different compound goals with RIL as compared to flat BC variants. We find that, while none of the variants are able to achieve near-perfect completion scores via just imitation, the average stepwise completion score is higher for RIL as compared to both flat variants (see Table 10.1, first row). Additionally, we find that the flat policy with data augmentation via relabeling performs better than without relabeling. When we analyze the proportion of compound goals that are actually fully achieved (see Table 10.1, bottom row), RIL shows significant improvement over other methods. This indicates that, even for imitation learning, we see benefits from introducing the simple RIL scheme described in Sec. 10.3.

	RIL (ours)	GCBC relabeling	GCBC no relabeling
Success Rate (%)	**21.7**	8.8	7.6
Average Step Completion (of 4)	$\mathbf{2.4 \pm 1.13}$	2.2 ± 0.95	1.78 ± 1.0

Table 10.1: Comparison of RIL to goal-conditioned behavior cloning with and without relabeling in terms success and step-completion rate averaged across 17 tasks. RIL outperforms the non-hierarchical methods

Relay Reinforcement Fine-tuning of Imitation Learning Policies

Although pure RIL does succeed at times, its performance is still relatively poor. In this section, we study the degree to which RIL-based policies are amenable to further reinforcement fine-tuning. Performing reinforcement fine-tuning individually on 17 different compound goals seen in the demonstrations, we observe a significant improvement in the average success rate and stepwise completion scores over all the baselines when using any of the variants of RPL (see Fig. 10.5). In our experiments, we found that it was sufficient to fine-tune the low-level policy, although we could also fine-tune both levels, at the cost of more non-stationarity. Although the large majority of the benefit is from RRF, we find a slight additional improvement from the DAPG-RPL and IRIL-RPL schemes, indicating that including the effect of the demonstrations throughout the process helps.

When compared with HRL algorithms that learn from scratch (on-policy HIRO [325]), we observe that RPL is able to learn much faster and reach a much higher success rate, showing the benefit of demonstrations. Additionally, we notice better fine-tuning performance when we compare RPL with flat-policy fine-tuning. This can be attributed to the fact that the credit assignment and reward specification problems are much easier for the relay policies, as compared to fine-tuning flat policies, where a sparse reward is rarely obtained. The RPL method also outperforms the pre-train-low-level baseline, which we hypothesize is because we are not able to search very effectively

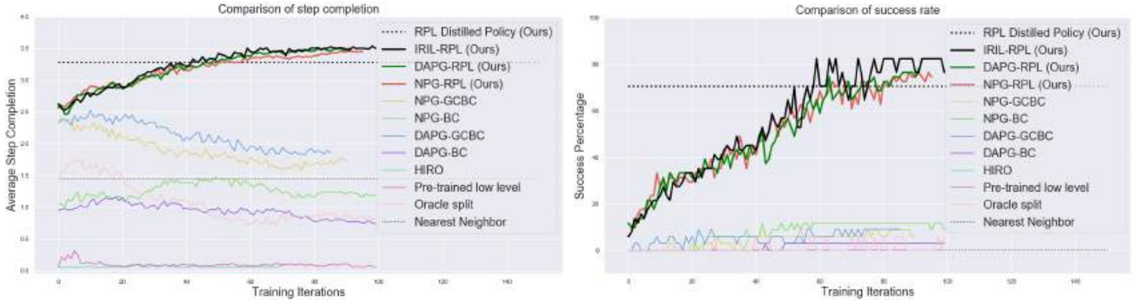


Figure 10.5: Comparison of the RPL algorithm with a number of baselines averaged over 17 compound goals and 2 (baseline methods) or 3 (our approach) random seeds. Fine-tuning with all three variants of our method outperforms fine-tuning using flat policies. RIL initialization at both levels improves the performance over HIRO [325] and over learning only the high-level policy from scratch. If we use policy distillation, we are able to get a successful, multi-task goal-conditioned policy.

in the goal space without further guidance. We also see a significant benefit over using the oracle scheme described in Appendix 3, since the segments become longer making the exploration problem more challenging. The comparison with the nearest neighbor baseline also suggests that there is a significant benefit from actually learning a *closed-loop* policy rather than using an open-loop policy. While plots in Fig. 10.5 show the average over various goals when fine-tuned individually, we can also distill the fine-tuned policies into a single, multi-task policy, as described in Sec. 10.5, that is able to solve almost all of the compound goals that were fine-tuned. While the success rate drops slightly, this gives us a single multi-task policy that can achieve multiple temporally-extended goals (Fig 10.5).

Ablations and Analysis

To understand design choices, we consider the role of using different window sizes for RPL as well as the role of reward functions during fine-tuning. In Fig 10.6 (left), we observe that the window size for RPL plays a major role in algorithm performance. As window size increases, both imitation learning and fine-tuning performance decreases since the behaviors are now more temporally extended.

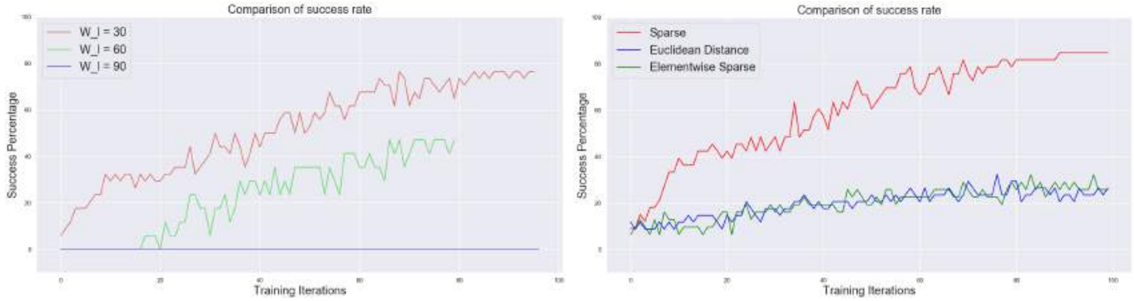


Figure 10.6: **Left:** Role of low level window size in RPL. As the window size increases, imitation learning and fine-tuning become less effective. **Right:** Role of fine-tuning reward function in RPL. We see that the sparse reward function is most effective once exploration is sufficiently directed.

Next, we consider the role of the chosen reward function in fine-tuning with RRF. We evaluate the relative performance of using different types of rewards for fine-tuning - sparse reward, euclidean distance, element-wise reward (refer to Appendix A for details). When each is used as a goal conditioned reward for fine-tuning the low-level, sparse reward works much better. This indicates that when exploration is sufficient, sparse reward functions are less prone to local optima than alternatives.

10.5 Conclusion and Future Work

We proposed relay policy learning, a method for solving long-horizon, multi-stage tasks by leveraging unstructured demonstrations to bootstrap a hierarchical learning procedure. We showed that we can learn a single policy capable of achieving multiple compound goals, each requiring temporally extended reasoning. In addition, we demonstrated that RPL significantly outperforms other baselines that utilize hierarchical RL from scratch, as well as imitation learning algorithms. While this paradigm can be quite effective in simulation, improving on challenging tasks through autonomous interaction, it is still quite expensive in terms of the number of samples required to learn. To really scale to real world settings, we need to move towards more efficient algorithms for bootstrapping. In the following chapter, we show how we can move from on-policy bootstrapped RL algorithms to *off-policy* bootstrapped RL algorithms, providing much more efficient learning as well as the ability to incorporate sub-optimal data. We then show that this paradigm can solve long horizon tasks in a real world kitchen in Section 18.

Chapter 11

Bootstrapping Off-Policy Reinforcement Learning with Offline Datasets and Online Finetuning

In the previous three chapters, we showed how to construct on-policy RL algorithms that can be bootstrapped with prior human data, and demonstrated the ability of these algorithms to solve both long horizon manipulation tasks and be applied to dexterous manipulation tasks in the real world. However, these algorithms are still on-policy, wasting a significant portion of the samples they collect since they can only use their latest batch of collected experience. In this chapter, we show that the paradigm of demonstration bootstrapped RL can actually be extended to off-policy RL algorithms, which are significantly more efficient. Additionally, these off-policy algorithms can bootstrap RL using not just optimal demonstration data but also sub-optimal data or data of mixed quality. We introduce the formalism of our off-policy bootstrapped RL algorithm and then show how we can apply it to complex tasks both in simulation and the real world, significantly outperforming the on-policy technique in chapter 10.

11.1 Why Should We Care About Bootstrapped Off-Policy RL?

Learning models that generalize effectively to complex open-world settings, from image recognition [12] to natural language processing [335], 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

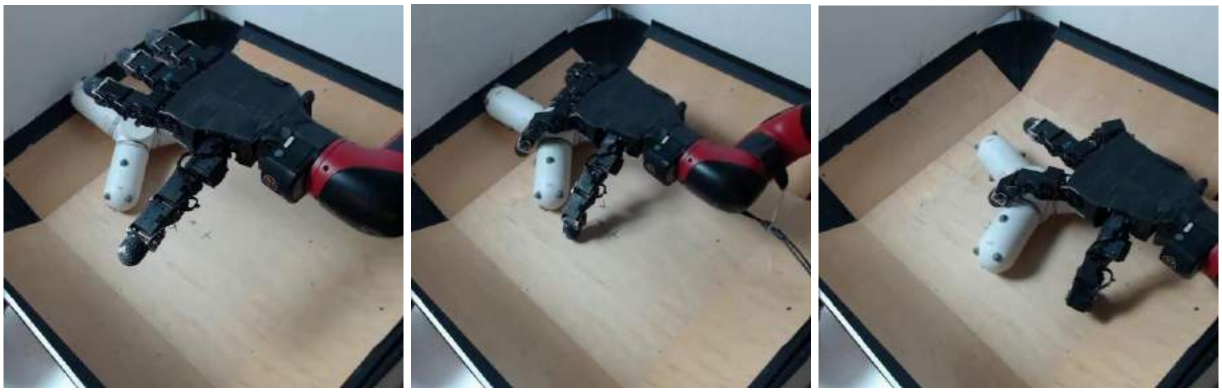


Figure 11.1: Utilizing prior data for online learning allows us to solve challenging real-world robotics tasks, such as this dexterous manipulation task where the learned policy must control a 4-fingered hand to reposition an object.

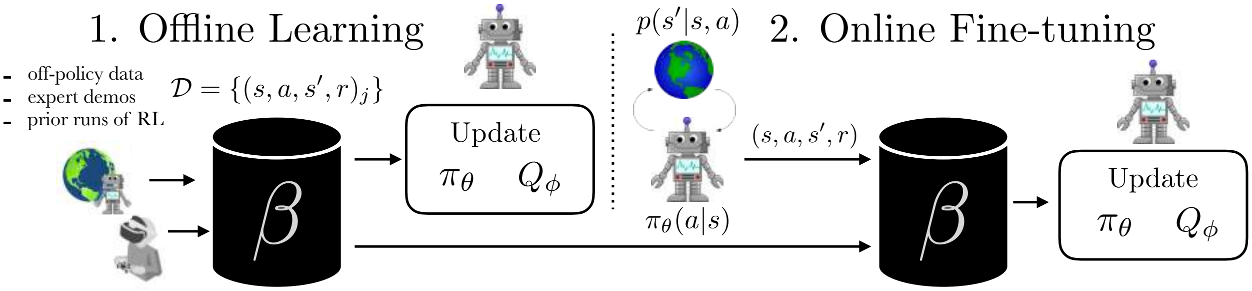


Figure 11.2: We study learning policies by offline learning on a prior dataset $\mathcal{D}$ and then fine-tuning with online interaction. The prior data could be obtained via prior runs of RL, expert demonstrations, or any other source of transitions. Our method, advantage weighted actor critic (AWAC) 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.

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 [336], [337], suboptimal policies [303], demonstrations for related tasks [338], 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.

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, prior methods that incorporate demonstrations into RL directly aim to mimic these demonstrations [300], 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 [339], [340], 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 [23], [340], [341] 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 data [339], [340], 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, 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.

11.2 Preliminaries

The optimal policy can be learned directly (e.g., using policy gradient to estimate $\nabla J(\pi)$ [19]), 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:

$$\begin{aligned} Q^\pi(\mathbf{s},\mathbf{a}) &= r(\mathbf{s},\mathbf{a}) + \gamma \mathbb{E}_{p(\mathbf{s}'|\mathbf{s},\mathbf{a})}[V^\pi(\mathbf{s}')] && (11.1) \\ &= r(\mathbf{s},\mathbf{a}) + \gamma \mathbb{E}_{p(\mathbf{s}'|\mathbf{s},\mathbf{a})}[\mathbb{E}_{\pi(\mathbf{a}'|\mathbf{s}')}[Q^\pi(\mathbf{s}',\mathbf{a}')]]. && (11.2) \end{aligned}$$

Instead of estimating policy gradients directly, actor-critic algorithms maximize returns by alternating between two phases [342]: 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 11.2, as defined below:

$$\mathcal{B}^\pi Q(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{p(\mathbf{s}'|\mathbf{s},\mathbf{a})}[\mathbb{E}_{\pi(\mathbf{a}'|\mathbf{s}')}[Q^\pi(\mathbf{s}', \mathbf{a}')]]. \tag{11.3}$$

By iterating according to $Q^{k+1} = \mathcal{B}^\pi Q^k$, Q^k converges to Q^π [343]. With function approximation, we cannot apply the Bellman operator exactly, and instead minimize the Bellman error with respect to Q-function parameters ϕ_k:

$$\phi_k = \arg\min_\phi \ \mathbb{E}_{\mathcal{D}}[(Q_\phi(\mathbf{s}, \mathbf{a}) - y)^2], \tag{11.4}$$

$$y = r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{\mathbf{s}',\mathbf{a}'}[Q_{\phi_{k-1}}(\mathbf{s}', \mathbf{a}')]. \tag{11.5}$$

During policy improvement, the actor π is typically updated based on the current estimate of Q^π. A commonly used technique [23], [286], [344] 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:

$$\theta_k = \arg\max_\theta \mathbb{E}_{\mathbf{s}\sim\mathcal{D}}[\mathbb{E}_{\pi_\theta(\mathbf{a}|\mathbf{s})}[Q_{\phi_k}(\mathbf{s}, \mathbf{a})]] \tag{11.6}$$

Actor-critic algorithms are widely used in deep RL [23], [286], [344], [345]. 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.

11.3 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.

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 11.2. 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

[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 11.4 are too hard to solve to analyze variants of different methods.

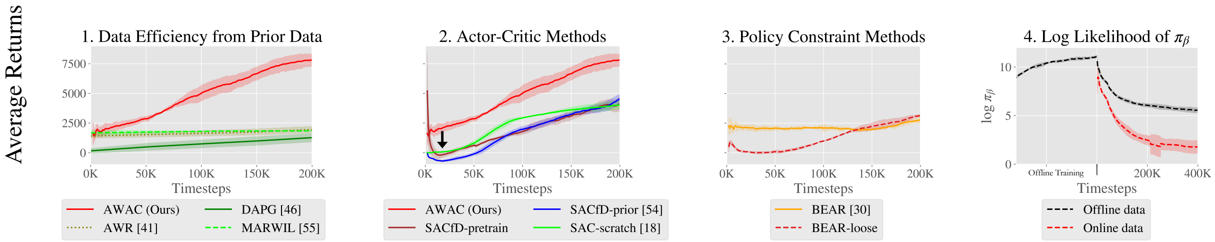


Figure 11.3: Analysis of prior methods on HalfCheetah-v2 using offline RL with online fine-tuning. (1) On-policy methods (DAPG, AWR, MARWIL) learn relatively slowly, even with access to prior data. We present our method, AWAC, as an example of how off-policy RL methods can learn much faster. (2) Variants 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 [340] 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.

suboptimal trajectories sampled from a behavioral clone of these demonstrations. All methods for the remainder of this paper incorporate the prior dataset, unless explicitly labeled "scratch".

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 [95], [346]. 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 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 [95], [346], or Monte-Carlo return estimation [347]–[349] 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 11.3 plot 1, where on-policy methods like DAPG [95] and Monte-Carlo return methods like AWR [349] and MARWIL [348] are an order of magnitude slower than off-policy actor-critic methods. Actor-critic methods, shown in Figure 11.3 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.

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 11.3: despite having a prior dataset in the replay buffer, these algorithms do not benefit significantly from offline training. We evaluate soft actor critic [23], 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 [339], [340], [343], [350], [351]. In actor-critic algorithms, the target value $Q(\mathrm{s}', \mathbf{a}')$, with $\mathbf{a}' \sim \pi$, is used to update $Q(\mathrm{s}, \mathbf{a})$. When $\mathbf{a}'$ is outside of the data distribution, $Q(\mathrm{s}', \mathbf{a}')$ will be inaccurate, leading to accumulation of error on static datasets.

Offline RL algorithms [339], [340], [350] propose to address this issue by explicitly adding constraints on the policy improvement update (Equation 11.6) to avoid bootstrapping on out-of-distribution actions, leading to a policy update of this form:

$$\arg\max_{\theta} \mathbb{E}_{\mathbf{s}\sim\mathcal{D}}[\mathbb{E}_{\pi_\theta(\mathbf{a}|\mathbf{s})}[Q_{\phi_k}(\mathbf{s}, \mathbf{a})]] \text{ s.t. } D(\pi_\theta, \pi_\beta) \leq \epsilon. \tag{11.7}$$

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 [339], [340], [352] 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 11.7 in various ways, including penalties on the policy update [340], [350] or architecture choices for sampling actions for policy training [339], [352]. As we will see next, the requirement for accurate estimation of $\hat{\pi}_\beta$ makes these methods difficult to use with online fine-tuning.

Excessively Conservative Online Learning

While offline RL algorithms with constraints [339], [340], [350] perform well offline, they struggle to improve with fine-tuning, as shown in the third plot in Figure 11.3. We see that the purely offline RL performance (at "0K" in Fig. 11.3) is much better than the standard off-policy methods shown in Section 11.3. However, with additional iterations of online fine-tuning, the performance increases very slowly (as seen from the slope of the BEAR curve in Fig 11.3). 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 [353], 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.

11.4 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 11.3. AWAC follows the design for actor-critic algorithms as described in Section 11.2, 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 11.3. 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 11.3, 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 11.2 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:

$$\pi_{k+1} = \underset{\pi \in \Pi}{\arg\max}\ \mathbb{E}_{\mathbf{a} \sim \pi(\cdot|\mathbf{s})}[A^{\pi_k}(\mathbf{s}, \mathbf{a})] \tag{11.8}$$

$$\text{s.t. } D_{\mathrm{KL}}(\pi(\cdot|\mathbf{s}) || \pi_{\mathcal{D}}(\cdot|\mathbf{s})) \leq \epsilon. \tag{11.9}$$

As we saw in Section 11.3, enforcing the constraint by incorporating an explicit learned behavior model [339], [340], [350], [352] 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 11.8 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 11.8 can be obtained by enforcing the KKT conditions [347], [349], [354]. The Lagrangian is:

$$\mathcal{L}(\pi, \lambda) = \mathbb{E}_{\mathbf{a} \sim \pi(\cdot|\mathbf{s})}[A^{\pi_k}(\mathbf{s}, \mathbf{a})] + \lambda(\epsilon - D_{\mathrm{KL}}(\pi(\cdot|\mathbf{s}) || \pi_\beta(\cdot|\mathbf{s}))), \tag{11.10}$$

and the closed form solution to this problem is $\pi^*(\mathbf{a}|\mathbf{s}) \propto \pi_{\mathcal{D}}(\mathbf{a}|\mathbf{s}) \exp\left(\frac{1}{\lambda} A^{\pi_k}(\mathbf{s},\mathbf{a})\right)$. 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 $\rho_{\pi_{\mathcal{D}}}(\mathbf{s})$:

$$\arg\min_\theta \mathop{\mathbb{E}}_{\rho_{\pi_{\mathcal{D}}}(\mathbf{s})} \left[D_{\mathrm{KL}}(\pi^*(\cdot|\mathbf{s}) || \pi_\theta(\cdot|\mathbf{s})) \right] \tag{11.11}$$

$$= \arg\min_\theta \mathop{\mathbb{E}}_{\rho_{\pi_{\mathcal{D}}}(\mathbf{s})} \left[\mathop{\mathbb{E}}_{\pi^*(\cdot|\mathbf{s})} [-\log \pi_\theta(\cdot|\mathbf{s})] \right] \tag{11.12}$$

Note that the parametric policy could be projected with either direction of KL divergence. Choosing the reverse KL results in explicit penalty methods [350] 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:

$$\theta_{k+1} = \arg\max_\theta \mathop{\mathbb{E}}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}} \left[\log \pi_\theta(\mathbf{a}|\mathbf{s}) \exp\left(\frac{1}{\lambda} A^{\pi_k}(\mathbf{s},\mathbf{a}) \right) \right]. \tag{11.13}$$

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 18.6 for a more detailed derivation and Appendix 18.6 for specific implementation details.

Avoiding explicit behavior modeling. Note that the update in Equation 11.13 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 13.7. This derivation is related to AWR [349], 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 11.6). The update also resembles ABM-MPO, but ABM-MPO *does* require modeling the behavior policy which, as discussed in Section 11.3, can lead to poor fine-tuning. In Section 11.6, 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 11.2. We utilize a temporal difference learning scheme for policy evaluation [23], [344], minimizing the Bellman error as described in Equation 11.3. 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 [349].

Algorithm 10: Advantage Weighted Actor Critic (AWAC)

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

Algorithm summary. The full AWAC algorithm for offline RL with online fine-tuning is summarized in Algorithm 12. In a practical implementation, we can parameterize the actor and the critic by neural networks and perform SGD updates from Eqn. 11.13 and Eqn. 11.4. Specific details are provided in Appendix 18.6. 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 [349], AWAC uses TD bootstrapping for significantly more efficient and even asymptotically better performance. As compared to offline RL techniques like ABM [352], MPO [341], BEAR [340] or BCQ [339] 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 11.4, when these seemingly ablations are made to AWAC, the algorithm performs significantly worse.

11.5 Related Work

Off-policy RL algorithms are designed to reuse off-policy data during training, and have been studied extensively [23], [310], [342], [344], [345], [355]–[359]. While standard off-policy methods are able to benefit from including data seen *during* a training run, as we show in Section 11.3 they struggle when training from previously collected offline data from other policies, due to error accumulation with distribution shift [339], [340]. Offline RL methods aim to address this issue, often by constraining the actor updates to avoid excessive deviation from the data distribution [339],

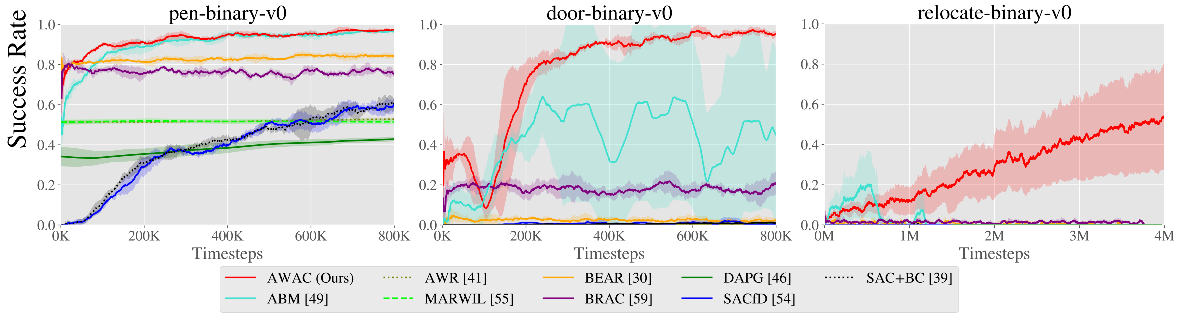


Figure 11.4: Comparative evaluation on the dexterous manipulation tasks. These tasks are difficult due to their high action dimensionality and reward sparsity. We see that AWAC is able to learn these tasks with little online data collection required (100K samples $\approx$ 16 minutes of equivalent real-world interaction time). Meanwhile, most prior methods are not able to solve the harder two tasks: door opening and object relocation.

[340], [351], [352], [360]–[368]. One class of these methods utilize importance sampling [355], [361], [364], [367]–[369]. Another class of methods perform offline reinforcement learning via dynamic programming, with an explicit constraint to prevent deviation from the data distribution [339], [340], [350], [360], [370]. While these algorithms perform well in the purely offline settings, we show in Section 11.3 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.

Our method builds on algorithms that implement a maximum likelihood objective for the actor, based on an expectation-maximization formulation of RL [285], [341], [347]–[349], [354], [371]. Most closely related to our method in this respect are the algorithms proposed by [349] (AWR) and [352] (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 11.3, 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 11.3. Closely to our work, [372] 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.

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 [287], [288], [373], initializing dynamic motion primitives [53], [373], [374] in the context of on-policy reinforcement learning algorithms [95], [375], inferring reward shaping [376] and even for inferring reward functions [69], [71]. Our work shows how we can generalize the idea

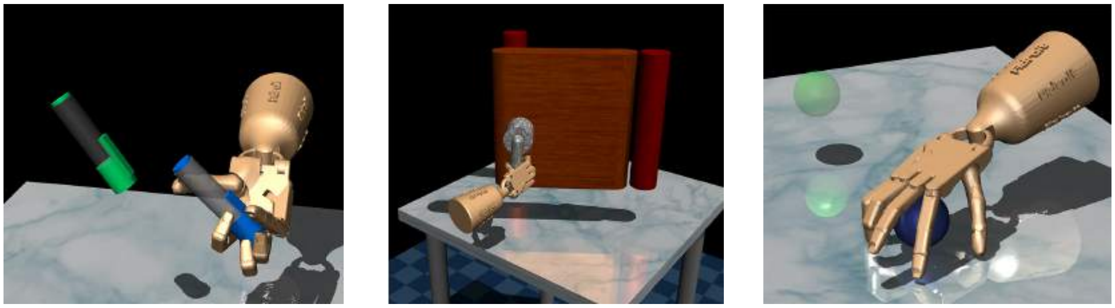


Figure 11.5: Illustration of dexterous manipulation tasks in simulation. These tasks exhibit sparse rewards, high-dimensional control, and complex contact physics.

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.

11.6 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*

Simulated Experiments

We study the first two questions in challenging simulation environments.

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 [95] in simulation. These tasks involve complex manipulation skills using a 28-DoF five-fingered hand in the MuJoCo simulator [91] shown in Figure 11.4: 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

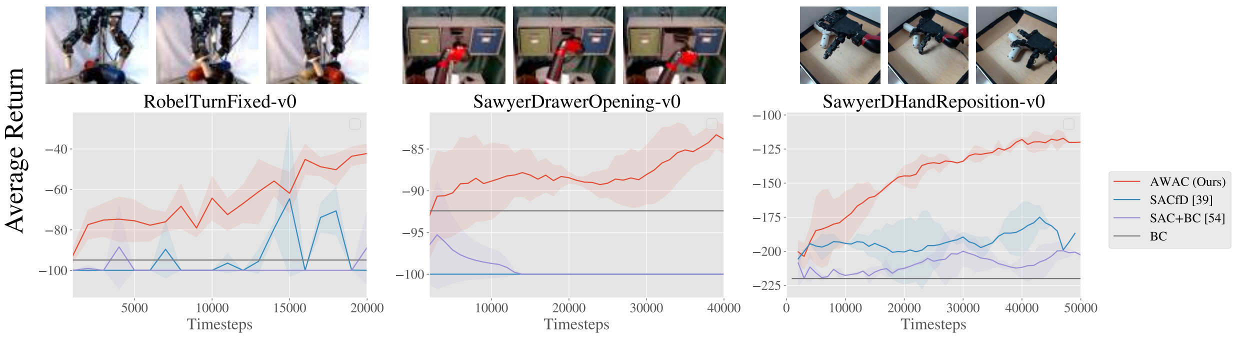


Figure 11.6: Algorithm comparison on three real-world robotic environments. Images of real world robotic tasks are pictured above. Left: a three fingered D'claw must rotate a valve 180°. Middle: a Sawyer robot must slide open a drawer using a hook attachment. Right: a dexterous hand attached to a Sawyer 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*

0-1 rewards for task completion. [2] [95] 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 18.6. The results are shown in Fig. 11.4. 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 [95]. 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 [349] while the implicit behavior modeling allows AWAC to significantly outperform ABM [352], although ABM does make some progress. [95] show that DAPG can solve variants of these tasks with more well-shaped rewards, but requires considerably more samples.

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 18.6 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

[2] [95] 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.

single prior method attains good performance on all tasks.

Fine-Tuning from Random Policy Data

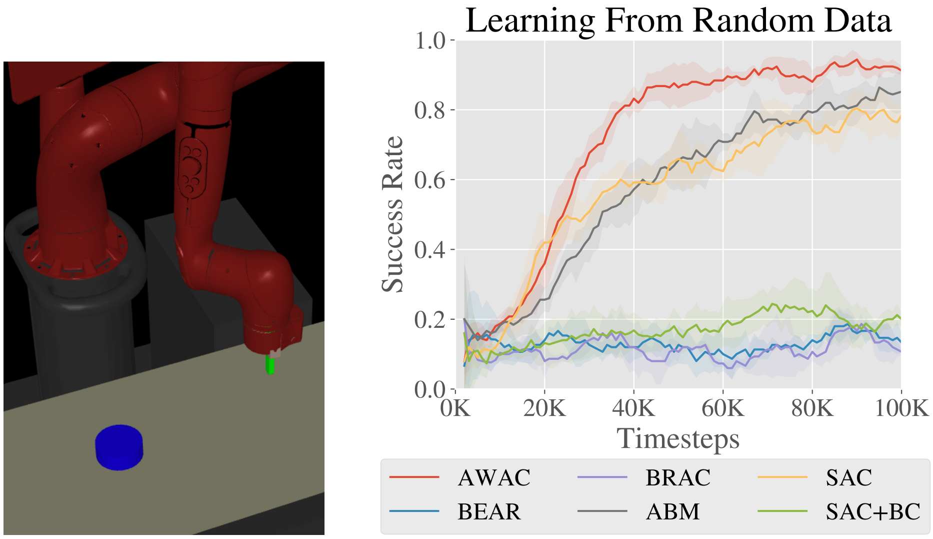


Figure 11.7: Comparison of fine-tuning from an initial dataset of suboptimal data on a Sawyer robot pushing task.

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 11.4 and described further in Appendix 18.6. 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 11.7. 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.

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 11.6. 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) [23], 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 [300] or an off-policy version of [95]. Further implementation details of these algorithms are provided in Appendix 18.6 in the supplementary materials.

Next, we describe the experimental setup for hardware experiments. Precise details of the hardware setup can be found in Appendix 18.6 in the supplementary materials.

Dexterous Manipulation with a 3-Fingered Claw. This task requires controlling a 3-fingered, 9 DoF robotic hand, introduced by [377], 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 [377]. 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 11.6, 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 [378]. 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 11.6. 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 finetuning). 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.

11.7 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. We end this part with a brief discussion of how this line of work connects to the broader space of related works and what the resulting implications are.

Chapter 12

Relationship to Other Work on Bootstrapping Reinforcement Learning

Next we try and place this work in context of some related work both prior to and post publishing of our work. Please refer to the detailed related works in each section for a more comprehensive consideration. Bootstrapping reinforcement learning from prior data is a well studied problem in the literature, and we build heavily on this literature. Prior to our work, several works based on dynamic motion primitives [67], [287], [293], [294] were used to enable faster learning but have been restricted to particular policy representations and somewhat simpler sets of problems. More recent work has shown the ability to incorporate demonstrations into off-policy algorithms by adding this data to the replay buffer [295], [299], [300]. However, as we showed in our work in chapter 13, this strategy can be prone to distribution shift and dips in performance. Our work on on-policy algorithms showed the ability to actually scale bootstrapped learning methods to complex dexterous manipulation tasks, while work on off-policy RL showed how we can make this efficient enough for real world application at scale. Since the publishing of our work, significant work has been done in the realm of dexterous manipulation, particularly in terms of simulation to reality transfer [41], [379], and extending these results to unique and different tasks [380].

Our work is closely connected to methods for offline RL [339], [340], [351], [381]. Offline RL algorithms have shown the ability to train from large offline datasets, requiring no online data collection. Most of these methods use a form of constrained optimization [340], [350] or conservative learning [381]–[383] as the subroutine for offline learning. Our work uses an implicit constrained optimization (similar to [349], [372]) to actually enable both offline training and online finetuning. We are amongst the first to show that this paradigm is able to scale to robotics problems and quick learning in the real world. Since our work has been published, many algorithms for offline RL have been studied [365], [384]–[386] and even in the finetuning case [387], [388]. [37] has shown the ability for these systems to scale up to large scale multi-task scenarios as well.

Part III

Continual Data Collection

In Part I we discussed how to actually supervise robotic learning systems, and in Part II we discussed how to get these algorithms to collect the right type of data efficiently and safely. However, as seen from the task setups in both Part I and II, most of these setups still require some amount of human instrumentation or intervention. While the techniques from Part I and II aim to alleviate some of these requirements, especially in the context of inferring reward functions, they largely require the involvement of human supervisors to ensure continual data collection and provide episodic resets very frequently.

The problem of continual data collection is important to ensure that learning systems are able to obtain data at the scale needed to actually leverage powerful deep learning techniques. If a human supervisor is constantly needed to babysit the learning process or construct elaborate instrumentation for providing resets, this becomes very hard to scale to collect large amounts of data across environments and tasks. In this part, we take a systems perspective on the problem and show how we can build largely instrumentation free, reset free learning systems that can collect large amounts of data with minimal human intervention to learn complex skills. In Section 17, we show how we can construct vision based reset free RL systems for simple dexterous manipulation tasks. In Section 18, we show how this paradigm can be extended to more complex tasks using multi-task RL and in Section 19 we show how this process can be made much more efficient by leveraging ideas from Part II to bootstrap from small amounts of human provided data. In doing so, we show that we can construct robotic learning systems that can learn complex behaviors, autonomously, collecting data at scale.

Chapter 13

Instrumentation Free Learning Systems for Real World Reinforcement Learning

In this chapter, we start our study of continual data collection systems by first discussing what ingredients are crucial for real world reinforcement learning at scale. We then instantiate a particular version of these ingredients that allows us to solve dexterous manipulation tasks with large scale, uninterrupted and uninstrumented real world training. In subsequent chapters, we discuss how this paradigm can be scaled to more complex tasks and manipulators.

13.1 Motivation

Reinforcement learning (RL) can in principle enable autonomous systems, such as robots, to acquire a large repertoire of skills automatically. Perhaps even more importantly, reinforcement learning can enable such systems to *continuously improve* the proficiency of their skills from experience. However, realizing this in reality has proven challenging: even with reinforcement learning methods that can acquire complex behaviors from high-dimensional low-level observations, such as images, the assumptions of the reinforcement learning problem setting do not fit cleanly into the constraints of the real world. For this reason, most successful robotic learning experiments have employed various kinds of environmental instrumentation in order to define reward functions, reset between trials, and obtain ground truth state [23], [31], [33], [41], [279], [290], [389], [390]. In order to practically and scalably deploy autonomous learning systems that improve continuously through real-world operation, we must lift these limitations and design algorithms that can learn under the constraints of real-world environments, as illustrated in Figure 13.2.

We propose that overcoming these challenges in a scalable way requires designing robotic systems that possess three capabilities: they are able to (1) learn from their own raw sensory inputs, (2) assign rewards to their own trials without hand-designed perception systems or instrumentation, and (3) learn continuously in non-episodic settings without requiring human intervention to manually reset the environment. A system with these capabilities can autonomously collect large amounts of real world data – typically crucial for effective generalization – without significant instrumentation

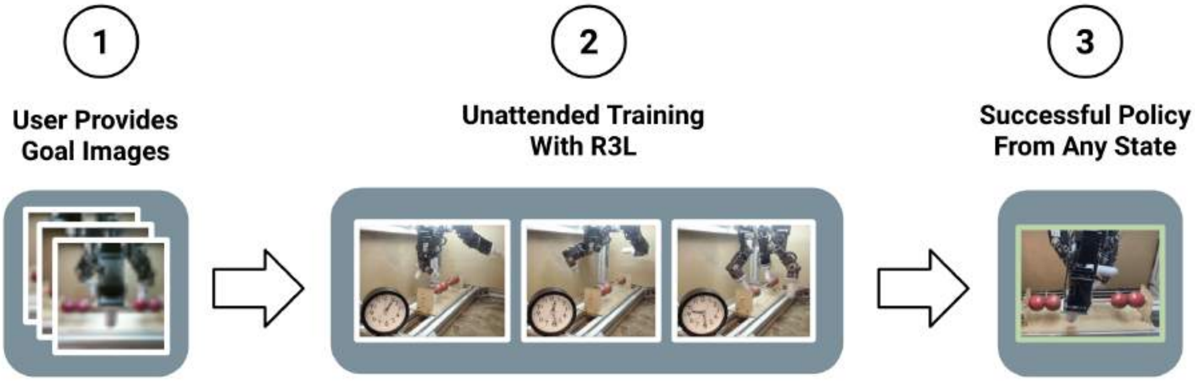

1
User Provides Goal Images
2
Unattended Training With R3L
3
Successful Policy From Any State

Figure 13.1: Illustration of our proposed instrumentation-free system requiring minimal human engineering. Human intervention is only required in the goal collection phase (1). The robot is left to train unattended (2) during the learning phase and can be evaluated from arbitrary initial states at the end of training (3). We show sample goal and intermediate images from the training process of a real hardware system

in each training environment, an example of which is shown in Figure 13.1. If successful, this would lift a major constraint that stands between current reinforcement learning algorithms and the ability to learn scalable, generalizable, and robust real-world behaviors. Such a system would also bring us significantly closer to the goal of embodied learning-based systems that improve continuously through their own real-world experience.

Having laid out these requirements, we propose a practical instantiation of such a learning system. While prior works have studied many of these issues in isolation, combining them into a complete real-world learning system presents a number of challenges, as we discuss in Section 13.3. We provide an empirical analysis of these issues, both in simulation and on a real-world robotic system, and propose a number of simple but effective solutions that together produce a complete robotic learning system. This system can autonomously learn from raw sensory inputs, learn reward functions from easily available supervision, and learn without manually designed reset mechanisms. We show that this system can learn dexterous robotic manipulation tasks in the real world, substantially outperforming ablations and prior work.

13.2 The Structure of a Real-World RL System

The standard reinforcement learning paradigm assumes that the controlled system is represented as a Markov decision process with a state space $\mathcal{S}$, action space $\mathcal{A}$, unknown transition dynamics $\mathcal{T}$, unknown reward function $\mathcal{R}$, and a (typically) episodic initial state distribution ρ. The goal is to learn a policy that maximizes the expected sum of rewards via interactions with the environment.

Although this formalism is simple and concise, it does not capture all of the complexities of real-world robotic learning problems. If a robotic system is to learn continuously and autonomously in the real world, we must ensure that it can learn under the actual conditions that are imposed by the real world. To move from the idealized MDP formulation to the real world, we require a system that has the following properties. **First**, all of the information necessary for learning must be obtained from the robot's own sensors. This includes information about the state and necessitates that the policy must be learned from high-dimensional and low-level sensory observations, such as camera images. **Second**, the robot must also obtain the *reward signal* itself from its own sensor readings. This is exceptionally difficult for all but the simplest tasks (e.g., reward functions that depend on interactions with specific objects require perceiving those objects explicitly). **Third**, we must be able to learn without access to episodic resets. A setup with explicit resets quickly becomes impractical in open-world settings, due to the requirement for significant human engineering of the environment, or direct human intervention during learning. While this list may not exhaustively enumerate all the components necessary for an effective real-world learning system, we posit that the properties listed here are fundamental to building such systems.

While some of the components discussed above can be tackled in isolation by current algorithms, there are considerable challenges inherent to assembling all these components into a complete learning system for real world robotic learning. In the rest of this section, we outline the challenges associated with each component, then discuss the challenges associated with combining these components in Section 13.3, and then proceed to address these challenges in Section 13.4.

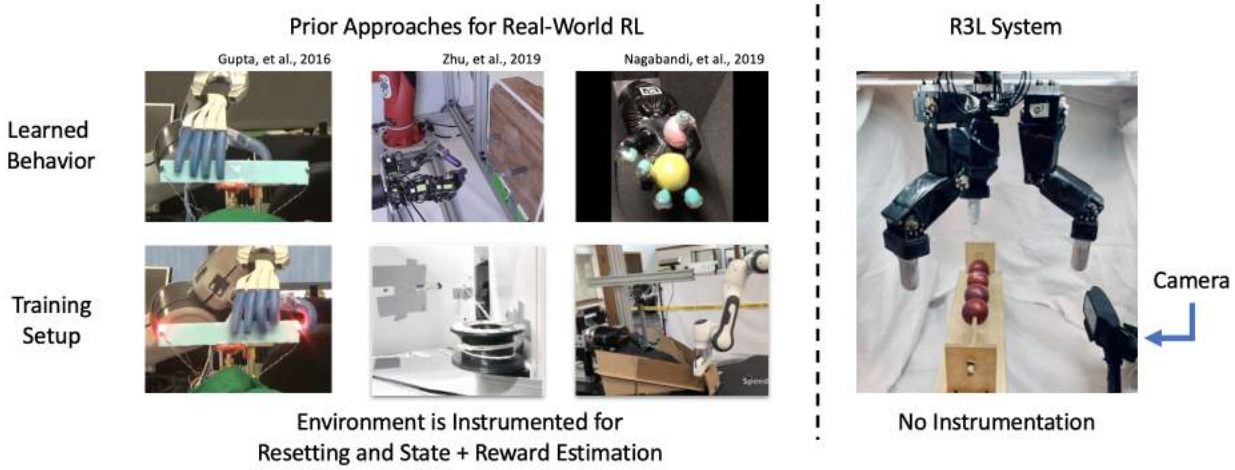


Figure 13.2: We draw a comparison between current real world learning systems which rely on instrumentation versus a system that learns in an environment more representative of the real world, free of instrumentation. While all three prior works utilize instrumentation for resets, state estimation, and reward, the motion capture system of [279], sensor attached to the door in [389], and auxiliary robot which picks up fallen balls in [33] are good examples of engineered state estimation, reward estimation, and reset mechanisms respectively.

Learning from Raw Sensory Input

To enable learning without complex state estimation systems or environment instrumentation, we require our robotic systems to be able to learn from their own raw sensory observations. Typically, these sensory observations are raw camera images from a camera mounted on the robot, as well as proprioceptive sensory inputs such as the joint angles. These observations do not directly provide the poses of the objects in the scene, which is the typical assumption in simulated robotic environments – any such information must be extracted by the learning system.

While in principle many RL frameworks can support learning from raw sensory inputs [21], [31], [90], [286], it is important to consider the practicalities of this approach. For instance, we can instantiate vision-based RL with policy gradient algorithms such as TRPO [90] and PPO [18], but these have high sample complexities which make them unsuited for real world robotic learning [23]. In our work, we consider adopting the general framework of off-policy actor-critic reinforcement learning, using a version of the soft actor critic (SAC) algorithm described by [391]. This algorithm effectively uses off-policy data and has been shown to learn some tasks directly from visual inputs. However, while SAC can learn directly from images, we find in our experiments that, as the task complexity increases, the efficiency of direct end-to-end learning (particularly without resets and with learned rewards) still degrades substantially. However, as we will discuss in Section 13.3, augmenting end-to-end learning with unsupervised representation learning substantially alleviates such challenges.

Reward Functions without Reward Engineering

Vision-based RL algorithms, such as SAC, rely on a reward function being provided to the system, which is typically manually programmed by a user. While this can be simple to do in simulation by using ground truth state information, it is significantly harder to implement in uninstrumented real world environments. In the real world, the robot must obtain the reward signal itself from its own sensor readings. A few options for tackling this challenge have been discussed in prior work: design complete computer vision systems to detect objects and extract the reward signals [33], [392], engineer reward functions that use various task-specific heuristics to obtain rewards from pixels [85], [393], or instrument every environment [32]. Many of these solutions are manual and tedious, and a more general approach is needed to scale real-world robotic learning gracefully.

Learning Without Resets

While the components described in Section 13.2 and 13.2 are essential to building continuously learning RL systems in the real world, they have often been implemented with the assumption of episodic learning [97], [391]. However, natural open-world settings do not provide any such reset mechanism, and in order to enable scalable and autonomous real-world learning we need systems that do not require an episodic formulation of the learning problem.

To devise a system that requires minimal human engineering for providing rewards, we must use algorithms that are able to assign *themselves* rewards, using learned models that operate on the same raw sensory inputs as the policy. One candidate is for a user to specify intended behavior beforehand through examples of desired outcomes (i.e., images). The algorithm can then assign itself rewards based on a measure of how well it is accomplishing the specified goals, with no additional human supervision. This approach can scale well in principle, since it requires minimal human engineering, and goal images are easy to provide.

In principle, algorithms such as SAC do not actually require episodic learning; however, in practice, most instantiations use explicit resets, even in simulation, and removing resets has resulted in failure to solve challenging tasks. In our experiments in Section 13.3 as well, we see that actor-critic methods applied naïvely to the reset free setting do not learn the intended behaviors. Introducing visual observations and classifier based rewards exacerbates these challenges.

We propose that these three components – vision-based RL with actor-critic algorithms, vision-based goal classifier for rewards, and reset-free learning – are the fundamental pieces that we need to build a real world robotic learning system. However, when we actually combine the individual components in Sections 13.3 and 13.7, we find that learning effective policies is quite challenging. We provide insight into these challenges in Section 13.3. Based on these insights, we propose simple but important changes in Section 13.4 to build a system, R3L , that can learn effectively and autonomously in the real world without human intervention.

13.3 The Challenges of Real World RL

The system design outlined in Section 13.2 in principle gives us a complete system to perform real world reinforcement learning without instrumentation. However, when utilized for robotic learning problems, we find this basic design to be largely ineffective. To study this, we present results for a simulated robotic manipulation task that requires repositioning a free-floating objects with a three-fingered robotic hand, shown in Fig 13.3. We use this task for our investigative analysis and show that the same insights extend to several other tasks, including real world tasks, in Section 13.7. The goal in this task is to reposition the object to a target pose from any initial pose in the arena. When the system is instantiated with vision-based soft actor-critic, rewards from goal images using VICE, and run without episodic resets, we see that the algorithm fails to make progress (Fig 13.4). Although it might appear that this setup fits within the assumptions of all of the components that are used, the complete system is ineffective. Which particular components of this problem make it so difficult?

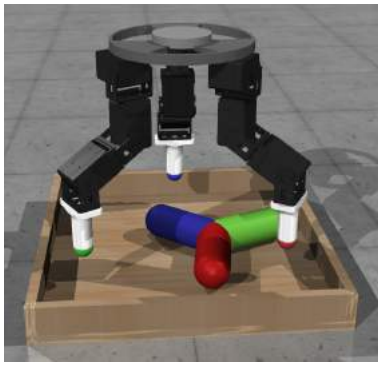
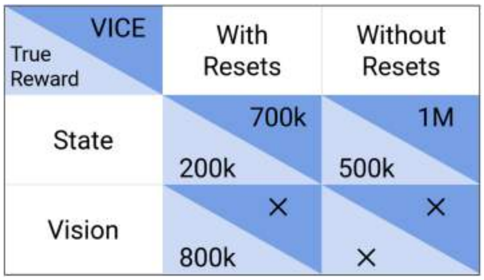


Figure 13.3: Our object repositioning task. The goal is to move the object from any starting configuration to a particular goal position and orientation.

To investigate this issue, we perform experiments investigating the combination of the three main ingredients: varying observation type (visual vs. low-dimensional state), reward structure (VICE vs. hand-defined rewards that utilize ground-truth object state), and the ability to reset (episodic resets vs. reset-free, non-episodic learning). We start by considering the training time reward under each combination of factors as shown in Fig 13.4, which reveals several trends. First, the results in Fig 13.4 show that learning with resets achieves high training time reward from both vision and state, while reset-free only achieves high training time reward with low-dimensional state. Second, we find that the policy is able to pass the threshold for training time reward in a non-episodic setting when learning from low-dimensional state, but it is not able to do the same using image observations. This suggests that combining the reset-free learning problem with visual observations makes it significantly more challenging.

However, the table in Fig 13.4 paints an incomplete picture. These numbers are related to



VICE True Reward	With Resets	Without Resets
State	700k 200k	1M 500k
Vision	× 800k	× ×

Figure 13.4: We report the approximate number of samples needed for a policy learned with a prior *off-policy RL algorithm (SAC)* to achieve average training performance of less than 0.15 in pose distance (defined in Appendix 18.7) across 3 seeds on the re-positioning task. We compare training performance after varying three axes: ground truth rewards vs. learned rewards, with vs. without episodic resets, low-level state vs. images as inputs. We observe learning without resets is harder than with resets and is much harder when combined with visual inputs.

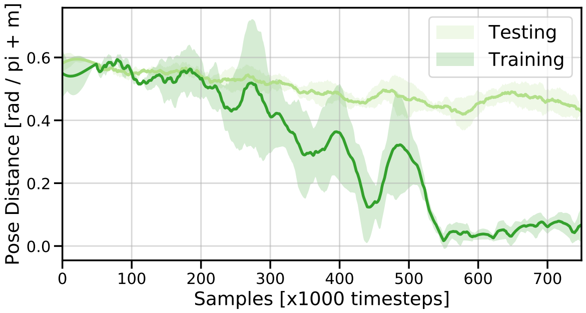


Figure 13.5: We observe that when training reset free to reach a single goal, while the pose distance at training time is quite low, the pose errors obtained at test-time with the learned policy are very high. This indicates that while the object is getting close to the goal at training time, the policies being learned are still not effective.

the performance of the policies at training time, not how effective the learned policies are when being evaluated. When we consider the test-time performance (Fig 13.5) of the policies that are learned under reset free conditions, we obtain a different set of conclusions. While learning from low-dimensional state in the reset free setting achieves high reward at training time, the test-time performance of the corresponding learned policies is very poor. This can likely be attributed to the fact that when the agent spends all its time stuck at the goal, and sees very little diversity of data in other parts of the state space, which significantly reduces the efficacy of the actual policies being learned. In a sense, the reset encodes some prior information about the task: it tells the policy about what types of states it might be required to succeed from at test time. Without this knowledge, performance is substantially worse. This makes it very challenging to learn policies with completely reset-free schemes, which has prompted prior work to consider schemes such as learning reset controllers [394]. As we discuss in the following section and in our experiments, these schemes are often insufficient for learning effective policies in the real world without *any* resets.

13.4 A Real-world Robotic Reinforcement Learning System

To address the challenges identified in Section 13.3, we present two improvements, which we found to be essential for uninstrumented real-world training: randomized perturbation controllers and unsupervised representation learning. Incorporating these components into the system in Section 13.2 results in a system that can learn successfully in uninstrumented environments, as we will show in Section 13.7, and attains good performance both at training time and at test time.

Randomized Perturbation Controller

Prior works in addressing the reset free setting problem have considered converting the problem into a more standard episodic problem, by learning a "reset controller," which is trained to reset the system to a particular *initial state* [394], [395]. This scheme has been thought to make the learning problem easier by reducing the variance of the resulting initial state distribution. However, as we will show in our experiments in Section 13.7, this still results in policies whose success depends heavily on a narrow range of initial states. Indeed, prior reset controller methods all reset to a *single* initial state [394], [395].

We take a different approach to learning in a reset-free setting. Rather than attributing the problem to the variance of the initial state distribution, we hypothesize that a major problem with reset-free learning is that the support of the distribution of states visited by the policy is too narrow, which makes the learning problem challenging and doesn't allow the agent to learn how to perform the desired task from *any* state it might find itself in. In this view, the goal should not be to *reduce* the variance of the initial state distribution, but instead to *increase* it.

To this end, we utilize what we call random perturbation controllers: controllers that introduce perturbations intermittently into the system through a policy that is trained to explore the environment. The standard actor $\pi(a|s)$ is executed for H time-steps, following which we executed the perturbation controller $\pi_p(a|s)$ for H steps, and repeat. The policy π is trained with the VICE-based rewards for reaching the desired goals, while the perturbation controller π_p is trained only with an intrinsic motivation objective that encourages visiting under-explored states. In our implementation, we use the random network distillation (RND) objective for training the perturbation controller [110], but any effective exploration method can be used for the same. This procedure is described in detail in Appendix 13.6, and is evaluated on the tasks we consider in Fig 18.11. The perturbation controller ensures that the support of the training distribution grows and as a result the policies can learn the desired behavior much more effectively, as shown in Fig 13.7.

Goal Classifier

To design a system that requires minimal human engineering for providing reward, we use a data-driven reward specification framework called variational inverse control with events (VICE) introduced by [97]. VICE learns rewards in a task-agnostic way: we provide the algorithm with success examples in the form of images where the task is accomplished, and learn a discriminator that is capable of distinguishing successes from failures. This discriminator can then be used to provide a learning signal to nudge the reinforcement learning agent towards success. This algorithm has been previously considered in the context of learning tasks from raw sensory observations in the real world by [116] but we show that it presents unique challenges when used in conjunction with learning without episodic resets. Details and specifics of the algorithms being considered are described in Appendix 13.6 and also discussed by [116].

Unsupervised Representation Learning

The perturbation controller discussed above allows us to learn policies that can succeed at the task from a variety of starting states. However, learning from visual observations still present a challenge. Our experiments in Fig 13.4 show that learning without resets from low-dimensional states is comparatively easier. We therefore aim to convert the vision-based learning problem into one that more closely resembles state-based learning, by training a variational autoencoder (VAE, [396]) and sharing the latent-variable representation across the actor and critic networks (refer to Appendix 18.7 for more details). Note that we use a VAE as an instantiation of representation learning techniques that works well in the domains we considered, but other more sophisticated density models proposed in prior work may also be substituted in place of the VAE [397]–[399].

While several works have also sought to incorporate unsupervised learning into reinforcement learning to make learning from images easier [30], [397], we note that this becomes especially critical in the vision-based, reset-free setting, as motivated by the experiments in Section 13.3, which indicate that it is precisely this combination of factors – vision and no resets – that presents the most difficult learning problem. Therefore, although the particular solution we use in our system has been studied in prior work, it is brought to bare to address a challenge that arises in real-world learning that we believe has not been explored in prior studies.

These two improvements – the perturbation controller and unsupervised learning – combined with the general system described above, give us a complete practical system for real world reinforcement learning. The overall method uses soft-actor critic for learning with visual observations and classifier based rewards with VICE, introduces auxiliary reconstruction objectives or pretrains encoders for unsupervised representation learning, and uses a perturbation controller during training to ensure that the support of visited states grows sufficiently. We term this full system for real-world robotic reinforcement learning R3L. Further implementation details can be found in Appendix 13.6.

13.5 Related Work

The primary contribution of this work is to propose a paradigm for continual instrumentation-free real world robotic learning and a practical instantiation of such a system. A number of prior works have studied reinforcement learning in the real world for acquiring robotic skills [31], [33], [35], [238], [290], [389], [391], [393]. Much of the focus in prior work has been on improving the efficiency and generalization of RL algorithms to make real-world training feasible, or else on utilizing simulation and transferring policies into the real world [94], [400]–[402]. The simulation-based methods typically require considerable effort in terms of both simulator design and overcoming the distributional shift between simulated and real-experience, while prior real-world training methods typically require additional instrumentation for either reward function evaluation [403] or resetting between trials [32], [35], [389], or both. In contrast, our work is primarily focused on lifting these requirements, rather than devising more efficient RL methods. We show that removing the need for instrumentation (i.e., for reward evaluation and resets) introduces additional challenges, which in turn require a careful set of design choices. Our complete R3L method is able to learn completely autonomously, without manual resets or reward design.

A key component of our system involves learning from raw visual inputs. This has proven to be difficult for policy gradient style algorithms [404] due to challenging representation learning problems. This has been made easier in simulated domains by using modified objectives, such as auxiliary losses [405], or by using more efficient algorithms [23]. We show that reinforcement learning on raw visual input, while possible in standard RL settings, becomes significantly more challenging when considered in conjunction with non-episodic, reset-free scenarios.

Reward function design is crucial for any RL system, and is non-trivial to provide in the real world. Prior works have considered instrumenting the environment with additional sensors to evaluate rewards [32], [35], [389], which is a highly manual process, using demonstrations, which require manual effort to collect [60], [146], [299], or using interactive supervision from the user [147]. In this work, we leverage the algorithm introduced by [97] to assign rewards based on the likelihood of a goal classifier. While prior work also applied this method to robotic tasks [116], this was done in a setting where manual resets were provided by hand, while we demonstrate that we can use learned rewards in a fully uninstrumented, reset-free setup.

Learning without resets has been considered in prior works [394], [395], although in different contexts – safe learning and learning compound controllers, respectively. [394] provide an algorithm to learn a reset controller with the goal of ensuring safe operation, but makes several assumptions that make it difficult to use in the real world: it assumes access to a ground truth reward function, it assumes access to an oracle function that can detect if an attempted reset by the reset policy was successful or not, and it assumes the ability to perform manual resets if the reset policy fails a certain number of times. In contrast, we propose an algorithm that allows for fully automated reinforcement learning in the real world. We compare to an ablation of our method that uses a reset controller similar to [394], and show that our method performs substantially better. Our perturbation controller also resembles the adversarial RL setup [209], [406]. However, while these prior methods explicitly aim to train policies that are robust to perturbations [406] or explore effectively [209], we are concerned with learning without access to resets.

While this line of work has connections to developmental robotics [407], [408] and its subfields, such as continual [409] and lifelong [410] learning, the goal of our work is to handle the practicalities of enabling reinforcement learning systems to learn in the real world without instrumentation or interruption, even for a single task setting. Though our work does not directly study continual lifelong learning, nor all facets of developmental robotics, it relates to continual learning [409], intrinsic motivation [235] and sensory-motor development involving proprioceptive manipulation [411].

13.6 Algorithm details

Algorithm 11: Real-World Robotic Reinforcement Learning (R3L)

```
$N \leftarrow$ number of training epochs
$H \leftarrow$ trajectory length (horizon)
$n_{\text{VICE}} \leftarrow$ number of VICE classifier training iterations per epoch
Initialize forward and perturbing policies $\pi_0, \pi_1$
Obtain goal states $s_i^E$ and initialize as a goal pool $\mathcal{G}$
Initialize RND target and predictor networks $f(s)$, $\hat{f}(s)$
Initialize VICE reward classifier $r_{\text{VICE}}(s)$
Initialize replay buffer $\mathcal{D}$
Collect initial exploration data and add to $\mathcal{D}$
for $i = 1$ to $2N$ do
    $k \leftarrow i \,\%\, 2$
    for $t = 1$ to $H$ do
        Sample $a_t \sim \pi_k(s_t)$
        Sample $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
        Sample batch from $\mathcal{D}$
        Update $\pi_k$ with batch according to SAC
        Update RND predictor network with batch
        Update running estimate of standard deviations of classifier and RND reward
    end for
    Add experience to the replay buffer with $\mathcal{D} \leftarrow \mathcal{D} \cup \tau_i$
    Sample an equal number of goal examples from $\mathcal{G}$ and negative examples from $\mathcal{D}$
    for $t = 1$ to $n_{\text{VICE}}$ do
        Train the VICE classifier on this batch with binary labels
    end for
end for
```

13.7 Experiments

In our experimental evaluation, we study how well the R3L system, described in Sections 13.2 and 13.4, can learn under realistic settings – visual observations, no hand-specified rewards, and no resets. We consider the following hypotheses:

1. Can we use R3L to learn complex robotic manipulation tasks without instrumentation? Does this system learn skills in both simulation and the real world?

2. Do the solutions proposed in Section 13.4 actually enable R3L to perform tasks without instrumentation that would not have been otherwise possible?

Experimental Setup

We consider the task of dexterous manipulation with a three fingered robotic hand, the D'Claw [377], [389], on a number of simulated and real world tasks. These tasks involve complex coordination of three fingers with 3 DoF each in order to manipulate objects. Prior works that used this robot utilized explicit resets and low-dimensional true state observations, while we consider settings with visual observations, no hand-specified rewards, and no resets.

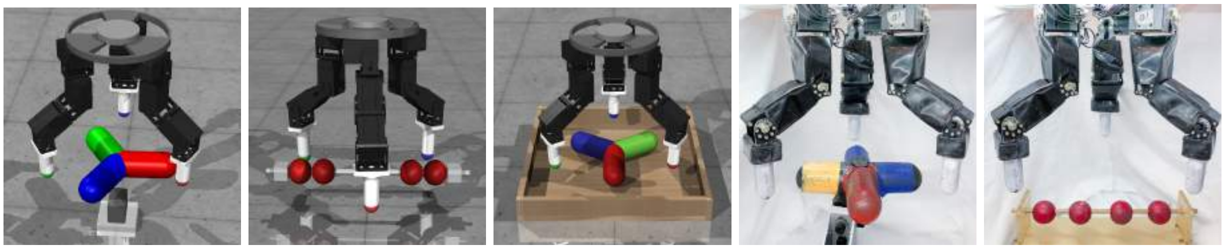


Figure 13.6: Visualizations of the goal configurations of the simulated and real world tasks being considered. From left to right we depict valve rotation, bead manipulation and free object repositioning in simulation, as well as valve rotation and bead manipulation manipulation in the real world.

The tasks in our experiments are shown in Fig 18.11: manipulating beads on an abacus row, valve rotation, and free object repositioning. These tasks represent a wide class of problems that robots might encounter in the real world. For each task, we consider the problem of reaching the depicted goal configuration: moving the abacus beads to a particular position, rotating the valve to a particular angle, and repositioning the free object to a particular goal position. For each task, policies are evaluated from a wide selection of initial configurations. Additional details about the tasks and evaluation procedures are provided in Appendix 18.7. Videos and additional details can be found at `https://sites.google.com/view/realworld-rl/`

Learning in Simulation without Instrumentation

We compare our entire proposed system implementation (Section 13.4) with a number of baselines and ablations. Importantly, all methods must operate under the same assumptions: none of the algorithms have access to system instrumentation for state estimation, reward specification, or episodic resets. Firstly, we compare the performance of R3L to a system which uses SAC for vision-based RL from raw pixels, VICE for providing rewards and running reset-free (denoted as “VICE"). This corresponds to the vanilla version of R3L (Section 13.2), with none of the proposed insights and changes. We then compare with prior reset-free RL algorithms [394] that explicitly learn a reset controller to alternate goals in the state space (“Reset Controller + VAE"). Lastly, we compare algorithm performance with two ablations: running R3L without the perturbation controller (“VICE + VAE") and without the unsupervised learning (“R3L w/o VAE"). This highlights the significance of each of the components of R3L .

From the experimental results in Fig 13.7, it is clear that R3L is able to reach the best performance across tasks, while none of the other methods are able to solve all of the tasks. Different prior methods and ablations fail for different reasons: (1) methods without the reconstruction objective struggle at parsing the high-dimensional input and are unable to solve the harder task; (2) methods without the perturbation controller are ineffective at learning how to reach the goal from

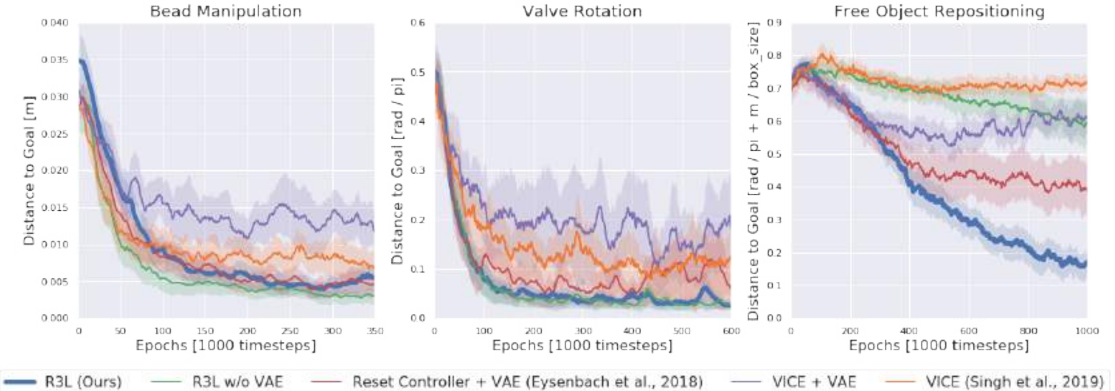


Figure 13.7: Quantitative evaluation of performance in simulation for bead manipulation, valve rotation and free object repositioning (left to right) run with 10 random seeds. The error bars show 95% bootstrap confidence intervals for average performance. While other variants are sufficient to get good evaluation performance on easier tasks, harder tasks like free object repositioning require random perturbations and unsupervised representation learning to learn skills reset-free. See Appendix 18.7 for details of evaluation procedures.

novel initialization positions for the more challenging object repositioning tasks, as discussed in Section 13.4.

We note that an explicit reset controller, which can be viewed as a softer version of our perturbation controller with goal-directedness, learns to solve the easier tasks due to the reset controller encouraging exploration of the state space. In our experiments for free object repositioning, performance was reported across 3 choices of reset states. The high variance in evaluation performance indicates that the performance of such a controller (or a goal conditioned variant of it) is highly sensitive to the choice of reset states. A poor choice of reset states, such as two that are very close together, may yield poor exploration leading to performance similar to the single goal VICE baseline. Furthermore, the choice of reset states is highly task dependent and it is often not clear what choice of goals will yield the best performance. On the contrary, our method does not require such task-specific knowledge and uses random perturbations to "reset" while training without any explicit reset states: this allows for a robust, instrumentation-free controller while also ensuring fast convergence.

Learning in the Real World without Instrumentation

Since the aim of R3L is to enable uninstrumented training in the real world, we next evaluate our method on a real-world robotic system, providing evidence that our insights generalize to the real world *without any instrumentation*. After providing the initial outcome examples for learning the reward function with VICE, we leave the robot unattended, and the algorithm learns the desired behavior through interaction. The experiments are performed on the D'Claw robotic hand with an RGB camera as the only sensory input. Intermediate policies are saved at regular intervals, and evaluations of all policies is performed after training has completed. For valve rotation, we declare an evaluation rollout a success if the final orientation is within within $15°$ of the goal. For bead manipulation, we declare success if all the beads are within 2cm of the goal state. Fig 13.8 compares the performance of our method without supervised learning ("R3L w/o VAE") in the real world against a baseline that uses SAC for vision-based RL from raw pixels, VICE for providing rewards, and running reset-free (denoted as “VICE"). We see that our method learns policies that succeed from nearly all the initial configurations, whereas VICE alone fails from most initial configurations. Fig 13.9 depicts sample evaluation rollouts of the policies learned using our method. For further details about real world experiments see Appendix 18.7.

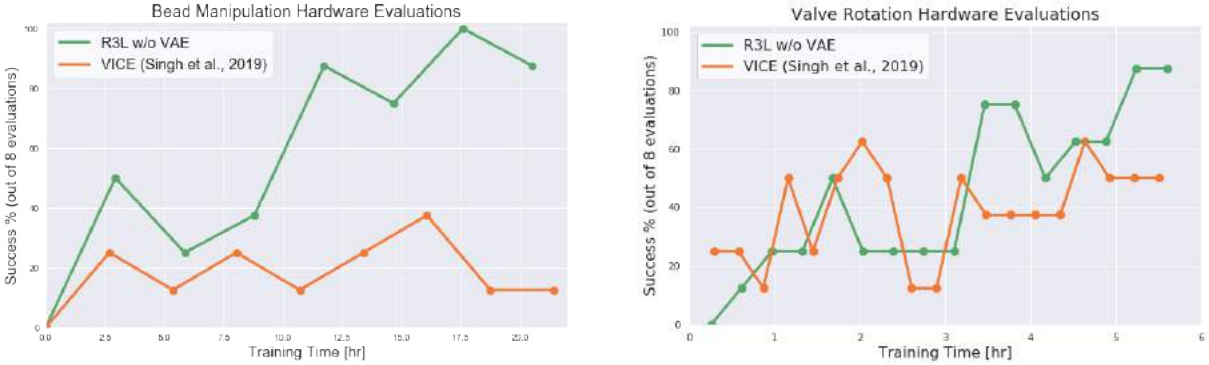


Figure 13.8: Quantitative evaluation of performance in real-world for valve rotation and bead manipulation. Policies trained with perturbation controllers have effectively learned behaviors after 17 and 5 hours of training, respectively. For more fine-grained reporting of results see Figs 18.45-18.48.

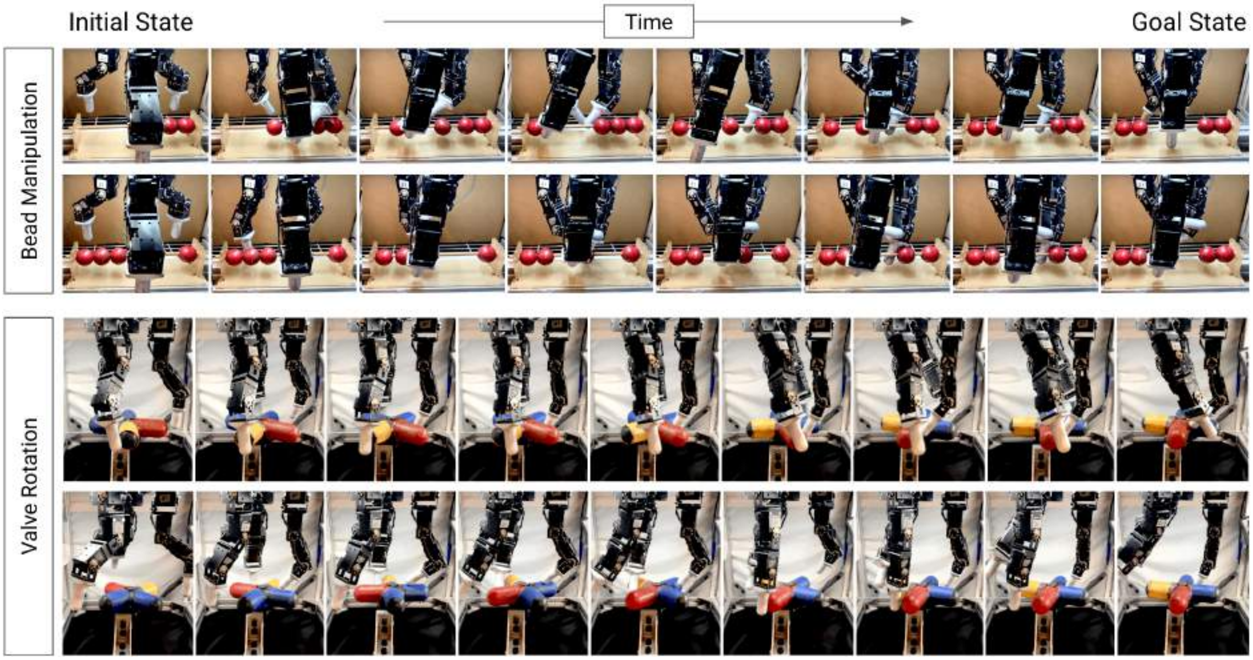


Figure 13.9: Evaluation rollouts of R3L on the real world tasks for policies trained without instrumentation. Successful evaluation rollouts for bead manipulation (top) and valve rotation (bottom) are shown here.

13.8 Discussion

We presented the design and instantiation of a system for real world reinforcement learning. We identify and investigate the various ingredients required for such a system to scale gracefully with minimal human engineering and supervision. We show that this system must be able to learn from raw sensory observations, learn from very easily specified reward functions without reward engineering, and learn without any episodic resets. We describe the basic elements that are required to construct such a system, and identify unexpected learning challenges that arise from interplay of these elements. We propose simple and scalable fixes to these challenges through introducing unsupervised representation learning and a randomized perturbation controller.

The ability to train robots directly in the real world with minimal instrumentation opens a number of exciting avenues for future research. Robots that can learn unattended, without resets or hand-designed reward functions, can in principle collect very large amounts of experience autonomously, which may enable very broad generalization in the future. However, there are also a number of additional challenges, including sample complexity, optimization and exploration difficulties on more complex tasks, safe operation, communication latency, sensing and actuation noise, and so forth, all of which would need to be addressed in future work in order to enable truly scalable real-world robotic learning. In the next chapters, we try to address some of these challenges by showing how to scale to more complex families of tasks via multi-task RL and bootstrapping from human demonstrations.

Chapter 14

Building Reset-Free Reinforcement Learning Algorithms via Multi-Task Learning

In the previous chapter, we showed how we can build a system for dexterous manipulation with uninterrupted and large scale data collection. However, the tasks that are actually being solved are still somewhat simplistic and to scale to a more complex set of problems becomes challenging with a simple perturbation controller. In this section, we show that the reset-free learning paradigm can actually scale particularly well when considered in a multi-task learning framework rather than solving single tasks in isolation. We show that this allows us to solve significantly more complex tasks in the real world than the previous section, while still requiring minimal human interventions.

14.1 Introduction

Reinforcement learning algorithms have shown promise in enabling robotic tasks in simulation [95], [302], and even some tasks in the real world [412], [413]. RL algorithms in principle can learn generalizable behaviors with minimal manual engineering, simply by collecting their own data via trial and error. This approach works particularly well in simulation, where data is cheap and abundant. Success in the real world has been restricted to settings where significant environment instrumentation and engineering is available to enable autonomous reward calculation and episodic resets [33], [41], [98], [316]. To fully realize the promise of robotic RL in the real world, we need to be able to learn even in the absence of environment instrumentation.

In this work, we focus specifically on reset-free learning for dexterous manipulation, which presents an especially clear lens on the reset-free learning problem. For instance, a dexterous hand performing in-hand manipulation, as shown in Figure 14.1 (right), must delicately balance the forces on the object to keep it in position. Early on in training, the policy will frequently drop the object, necessitating a particularly complex reset procedure. Prior work has addressed this manually by having a human involved in the training process [289], [290], [414], instrumenting a

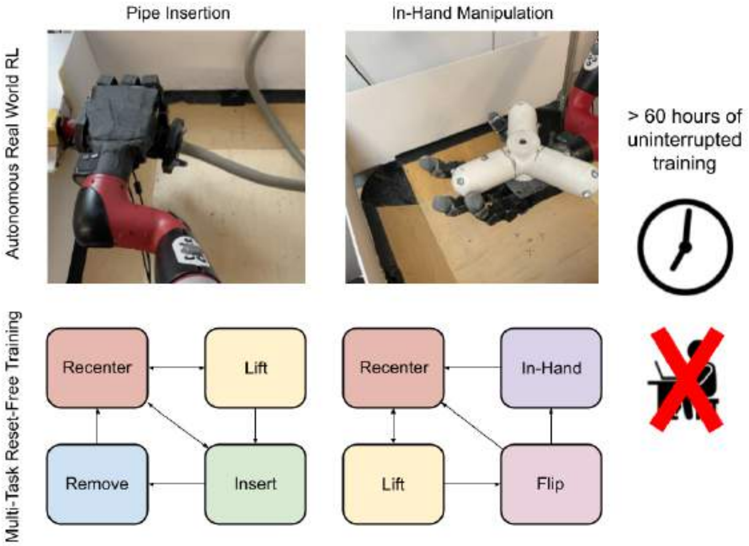


Figure 14.1: Reset-free learning of dexterous manipulation behaviors by leveraging multi-task learning. When multiple tasks are learned together, different tasks can serve to reset each other, allowing for uninterrupted continuous learning of all of the tasks. This allows for the learning of dexterous manipulation tasks like in-hand manipulation and pipe insertion with a 4-fingered robotic hand, without any human intervention, with over 60 hours of uninterrupted training

reset in the environment [389], [390], or even by programming a separate robot to place the object back in the hand [33]. Though some prior techniques have sought to learn some form of a "reset" controller [98], [209], [377], [394], [415], [416], none of these are able to successfully scale to solve a complex dexterous manipulation problem without hand-designed reset systems due to the challenge of learning robust reset behaviors.

However, general-purpose robots deployed in real-world settings will likely be tasked with performing many different behaviors. While multi-task learning algorithms in these settings have typically been studied in the context of improving sample efficiency and generalization [139], [334], [417]–[420], in this work we make the observation that multi-task algorithms naturally lend themselves to the reset-free learning problem. We hypothesize that the reset-free RL problem can be addressed by reformulating it as a multi-task problem, and appropriately sequencing the tasks commanded and learned during online reinforcement learning. As outlined in Figure 14.6, solving a collection of tasks simultaneously presents the possibility of using some tasks as a reset for others, thereby removing the need of explicit per task resets. For instance, if we consider the problem of learning a variety of dexterous hand manipulation behaviors, such as in-hand reorientation, then learning and executing behaviors such as recenter and pickup can naturally reset the other tasks in the event of failure (as we describe in Section 15.3 and Figure 14.6). We show that by learning multiple different tasks simultaneously and appropriately sequencing behavior across different tasks, we can learn *all* of the tasks without episodic resets required at all. This allows us to effectively learn a "network" of reset behaviors, each of which is easier than learning a complete reset controller, but together can execute and learn more complex behavior.

The main contribution of this work is to propose a learning system that can learn dexterous manipulation behaviors without the need for episodic resets. We do so by leveraging the paradigm of multi-task reinforcement learning to make the reset free problem less challenging. The system accepts a diverse set of tasks that are to be learned, and then trains reset-free, leveraging progress in some tasks to provide resets for other tasks. To validate this algorithm for reset-free robotic learning, we perform both simulated and hardware experiments on a dexterous manipulation system with a four fingered anthropomorphic robotic hand. To our knowledge, these results demonstrate the first instance of a combined hand-arm system learning dexterous in-hand manipulation with deep RL entirely in the real world with minimal human intervention during training, simultaneously acquiring both the in-hand manipulation skill and the skills needed to retry the task. We also show the ability of this system to learn other dexterous manipulation behaviors like pipe insertion via uninterrupted real world training, as well as several tasks in simulation.

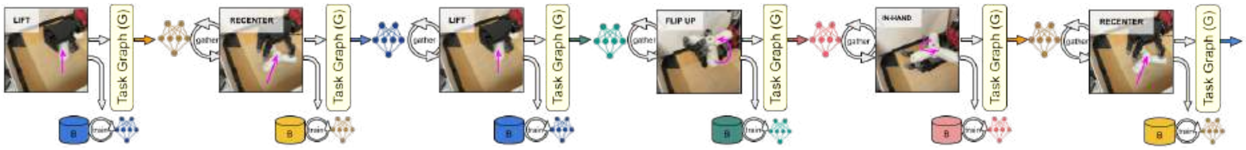


Figure 14.2: Depiction of some steps of reset-free training for the in-hand manipulation task family on hardware. Reset-free training uses the task graph to choose which policy to execute and train at every step. For executions that are not successful (e.g., the pickup in step 1), other tasks (recenter in step 2) serve to provide a reset so that pickup can be attempted again. Once pickup is successful, the next task (flip up) can be attempted. If the flip-up policy is successful, then the in-hand reorientation task can be attempted and if this drops the object then the re-centering task is activated to continue training.

14.2 Learning Dexterous Manipulation Behaviors Reset-Free via Multi-Task RL

One of the main advantages of dexterous robots is their ability to perform a wide range of *different* tasks. Indeed, we might imagine that a real-world dexterous robotic system deployed in a realistic environment, such as a home or office, would likely need to perform a repertoire of different behaviors, rather than just a single skill. While this may at first seem like it would only make the problem of learning without resets more difficult, the key observation we make in this work is that the multi-task setting can actually facilitate reset-free learning without manually provided instrumentation. When a large number of diverse tasks are being learned simultaneously, some tasks can naturally serve as resets for other tasks during learning. Learning each of the tasks individually without resets is made easier by appropriately learning and sequencing together other tasks in the right order. By doing so, we can replace the simple forward, reset behavior dichotomy with a more natural "network" of multiple tasks that can perform complex reset behaviors between each other.

Let us ground this intuition in a concrete example. Given a dexterous table-top manipulation task, shown in Fig 14.6 and Fig 14.2 , our reset-free RL procedure might look like this: let us say the robot starts with the object in the palm and is trying to learn how to manipulate it in-hand so that it is oriented in a particular direction (*in-hand reorient*). While doing so, it may end up dropping the object. When learning with resets, a person would need to pick up the object and place it back in the hand to continue training. However, since we would like the robot to learn without such manual interventions, the robot itself needs to retrieve the object and resume practicing. To do so, the robot must first *re-center* and the object so that it is suitable for grasping, and then actually *lift* and the *flip up* so it's in the palm to resume practicing. In case any of the intermediate tasks (say lifting the object) fails, the recenter task can be deployed to attempt picking up again, practicing these tasks themselves in the process. Appropriately sequencing the execution and learning of different tasks, allows for the autonomous practicing of in-hand manipulation behavior, without requiring any human or instrumented resets.

Algorithm Description

In this work, we directly leverage this insight to build a dexterous robotic system that learns in the absence of resets. We assume that we are provided with K different tasks that need to be learned together. These tasks each represent some distinct capability of the agent. As described above, in the dexterous manipulation domain the tasks might involve re-centering, picking up the object, and reorienting an object in-hand. Each of these K different tasks is provided with its own reward function $R_i(s_t, a_t)$, and at test-time is evaluated against its distinct initial state distribution μ_0^i.

Our proposed learning system, which we call Multi-Task learning for Reset-Free RL (*MTRF*), attempts to jointly learn K different policies π_i, one for each of the defined tasks, by leveraging off-policy RL and continuous data collection in the environment. The system is initialized by randomly sampling a task and state s_0 sampled from the task's initial state distribution. The robot collects a continuous stream of data *without any subsequent resets* in the environment by sequencing the K policies according to a meta-controller (referred to as a "task-graph") $G(s) : S \to \{0, 1, \dots, K-1\}$. Given the current state of the environment and the learning process, the task-graph makes a decision once every T time steps on which of the tasks should be executed and trained for the *next* T time steps. This task-graph decides what order the tasks should be learned and which of the policies should be used for data collection. The learning proceeds by iteratively collecting data with a policy π_i chosen by the task-graph for T time steps, after which the collected data is saved to a task-specific replay buffer $\mathcal{B}_i$ and the task-graph is queried again for which task to execute next, and the whole process repeats.

We assume that, for each task, successful outcomes for tasks that lead into that task according to the task graph (i.e., all incoming edges) will result in valid initial states for that task. This assumption is reasonable: intuitively, it states that an edge from task A to task B implies that successful outcomes of task A are valid initial states for task B. This means that, if task B is triggered after task A, it will learn to succeed from these valid initial states under μ_0^B. While this does not always guarantee that the downstream controller for task B will see *all* of the initial states from μ_0^B, since the upstream controller is not explicitly optimizing for coverage, in practice we find that this still performs very well. However, we expect that it would also be straightforward to introduce coverage into this method by utilizing state marginal matching methods [243]. We leave this for future work.

The individual policies can continue to be trained by leveraging the data collected in their individual replay buffers $\mathcal{B}_i$ via off-policy RL. As individual tasks become more and more successful, they can start to serve as effective resets for other tasks, forming a natural curriculum. The proposed framework is general and capable of learning a diverse collection of tasks reset-free when provided with a task graph that leverages the diversity of tasks. This leaves open the question of how to actually define the task-graph G to effectively sequence tasks. In this work, we assume that a task-graph defining the various tasks and the associated transitions is provided to the agent by the algorithm designer. In practice, providing such a graph is simple for a human user, although it could in principle be learned from experiential data. We leave this extension for future work.

Practical Instantiation

To instantiate the algorithmic idea described above as a deep reinforcement learning framework that is capable of solving dexterous manipulation tasks without resets, we can build on the framework of actor-critic algorithms. We learn separate policies π_i for each of the K provided tasks, with separate critics Q_i and replay buffers $\mathcal{B}_i$ for each of the tasks. Each of the policies π_i is a deep neural network Gaussian policy with parameters θ_i, which is trained using a standard actor-critic algorithm, such as soft actor-critic [23], using data sampled from its own replay buffer $\mathcal{B}_i$. The task graph G is represented as a user-provided state machine, as shown in Fig 14.6, and is queried every T steps to determine which task policy π_i to execute and update next. Training proceeds by starting execution from a particular state s_0 in the environment, querying the task-graph G to determine which policy $i = G(s_0)$ to execute, and then collecting T time-steps of data using the policy π_i, transitioning the environment to a new state s_T (Fig 14.6). The task-graph is then queried again and the process is repeated until all the tasks are learned.

Algorithm 12: *MTRF*

```
Given: $K$ tasks with rewards $R_i(s_t, a_t)$, along with a task graph mapping states to a task index
    $G(s) : S \to \{0, 1, \ldots, K-1\}$
Let $\hat{i}$ represent the task index associated with the forward task that is being learned.
Initialize $\pi_i$, $Q_i$, $\mathcal{B}_i$ $\forall i \in \{0, 1, \ldots, K-1\}$
Initialize the environment in task $\hat{i}$ with initial state $s_{\hat{i}} \sim \mu_{\hat{i}}(s_{\hat{i}})$
for iteration $n = 1, 2, ...$ do
    Obtain current task $i$ to execute by querying task graph at the current environment state
        $i = G(s_{\text{curr}})$
    for iteration $j = 1, 2, ..., T$ do
        Execute $\pi_i$ in environment, receiving task-specific rewards $R_i$ storing data in the buffer $\mathcal{B}_i$
        Train the current task's policy and value functions $\pi_i$, $Q_i$ by sampling a batch from the
            replay buffer containing this task's experience $\mathcal{B}_i$, according to SAC [23].
    end for
end for
```

14.3 Task and System Setup

To study *MTRF* in the context of challenging robotic tasks, such as dexterous manipulation, we designed an anthropomorphic manipulation platform in both simulation and hardware. Our system

(Fig 14.5) consists of a 22-DoF anthropomorphic hand-arm system. We use a self-designed and manufactured four-fingered, 16 DoF robot hand called the *D'Hand*, mounted on a 6 DoF Sawyer robotic arm to allow it to operate in an extended workspace in a table-top setting. We built this hardware to be particularly amenable to our problem setting due it's robustness and ease of long term operation. The *D'Hand* can operate for upwards of 100 hours in contact rich tasks without any breakages, whereas previous hand based systems are much more fragile. Given the modular nature of the hand, even if a particular joint malfunctions, it is quick to repair and continue training. In our experimental evaluation, we use two different sets of dexterous manipulation tasks in simulation and two different sets of tasks in the real world. Details can be found in Appendix A and at `https://sites.google.com/view/mtrf`

Simulation Domains

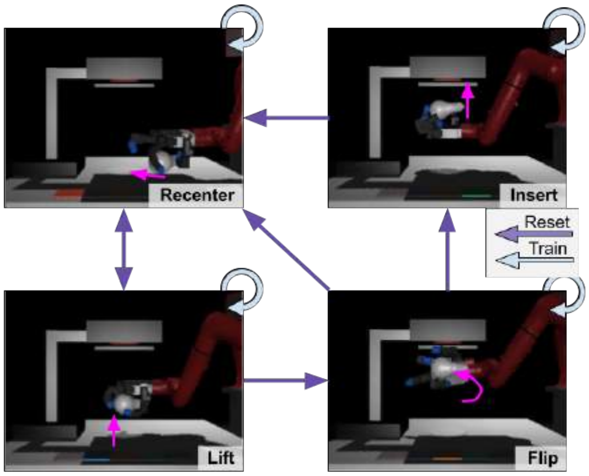


Figure 14.3: Tasks and transitions for lightbulb insertion in simulation. The goal is to recenter a lightbulb, lift it, flip it over, and then insert it into a lamp.

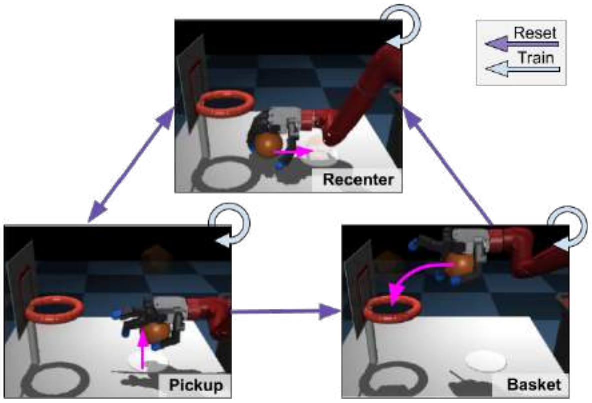


Figure 14.4: Tasks and transitions for basketball domain in simulation. The goal here is to reposition a basketball object, pick it up and then dunk it in a hoop.

Lightbulb insertion tasks. The first family of tasks involves inserting a lightbulb into a lamp in simulation with the dexterous hand-arm system. The tasks consist of centering the object on the table, pickup, in-hand reorientation, and insertion into the lamp. The multi-task transition task graph is shown in Fig 14.3. These tasks all involve coordinated finger and arm motion and require precise movement to insert the lightbulb.

Basketball tasks. The second family of tasks involves dunking a basketball into a hoop. This consists of repositioning the ball, picking it up, positioning the hand over the basket, and dunking the ball. This task has a natural cyclic nature, and allows tasks to reset each other as shown in Fig 14.4, while requiring fine-grained behavior to manipulate the ball midair.

Hardware Tasks

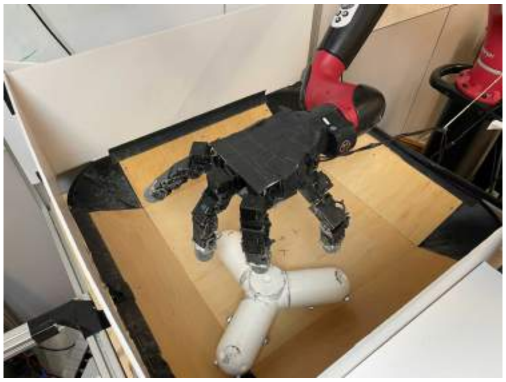


Figure 14.5: Real-world hand-arm manipulation platform. The system comprises a 16 DoF hand mounted on a 6 DoF Sawyer arm. The goal of the task is to perform in-hand reorientation, as illustrated in Fig 14.10 or pipe insertion as shown in Fig 14.11

We also evaluate *MTRF* on the real-world hand-arm robotic system, training a set of tasks in the real world, without any simulation or special instrumentation. We considered 2 different task families - in hand manipulation of a 3 pronged valve object, as well as pipe insertion of a cylindrical pipe into a hose attachment mounted on the wall. We describe each of these setups in detail below:

In-Hand Manipulation: For the first task on hardware, we use a variant of the in-hand reorienting task, where the goal is to pick up an object and reorient it in the palm into a desired configuration, as shown in Fig 14.6. This task not only requires mastering the contacts required for a successful pickup, but also fine-grained finger movements to reorient the object in the palm, while at the same time balancing it so as to avoid dropping. The task graph corresponding to this domain is shown in Fig 14.6. A frequent challenge in this domain stems from dropping the object during the in-hand reorientation, which ordinarily would require some sort of reset mechanism (as seen in prior work [33]).

However, *MTRF* enables the robot to utilize such “failures” as an opportunity to practice the tabletop re-centering, pickup, and flip-up tasks, which serve to “reset” the object into a pose where the reorientation can be attempted again.[1] The configuration of the 22-DoF hand-arm system mirrors that in simulation. The object is tracked using a motion capture system. Our policy directly controls each joint of the hand and the position of the end-effector. The system is set up to allow for extended uninterrupted operation, allowing for over 60 hours of training without any human intervention. We show how our proposed technique allows the robot to learn this task in the following section.

Pipe insertion: For the second task on hardware, we set up a pipe insertion task, where the goal is to pick up a cylindrical pipe object and insert it into a hose attachment on the wall, as shown in Fig 14.7. This task not only requires mastering the contacts required for a successful pickup, but also accurate and fine-grained arm motion to insert the pipe into the attachment in the wall. The task graph corresponding to this domain is shown in Fig 14.7. In this domain, the agent learns to pickup the object and then insert it into the attachment in the wall. If the object is dropped, it is then re-centered and picked up again to allow for another attempt at insertion. [2] As in the previous domain, our policy directly controls each joint of the hand and the position of the end-effector. The system is set up to allow for extended uninterrupted operation, allowing for over 30 hours of training without any human intervention.

[1] For the pickup task, the position of the arm’s end-effector is scripted and only D’Hand controls are learned to reduce training time.

[2] For the pickup task, the position of the arm’s end-effector is scripted and only D’Hand controls are learned to reduce training time. For the insertion task, the fingers are frozen since it is largely involving accurate motion of the arm.

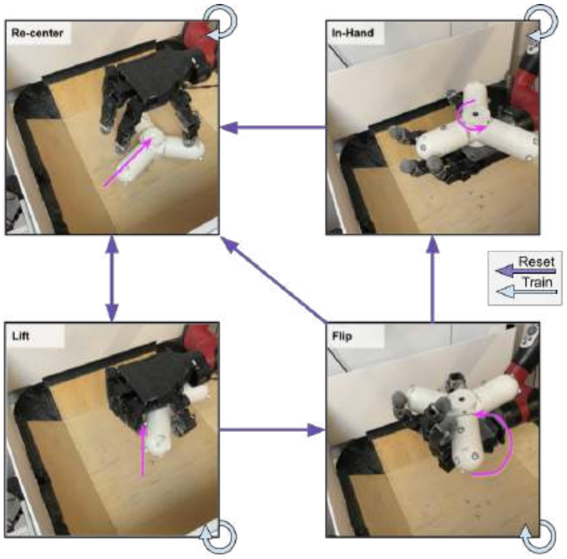


Figure 14.6: Tasks and transitions for the in-hand manipulation domain on hardware. The goal here is to rotate a 3 pronged valve object to a particular orientation in the palm of the hand, picking it up if it falls down to continue practicing.

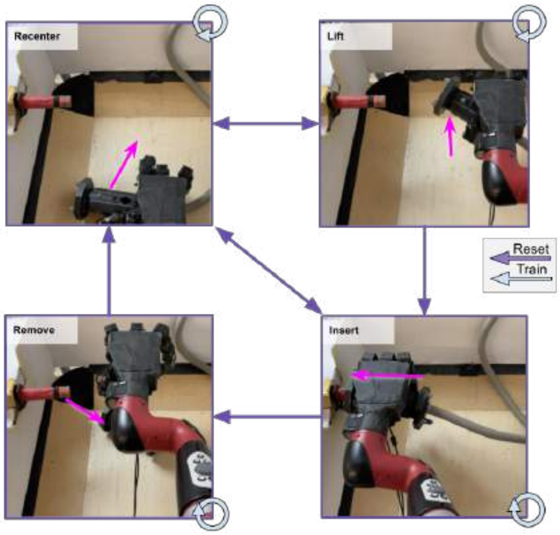


Figure 14.7: Tasks and transitions for pipe insertion domain on hardware. The goal here is to reposition a cylindrical pipe object, pick it up and then insert it into a hose attachment on the wall.

14.4 Experimental Evaluation

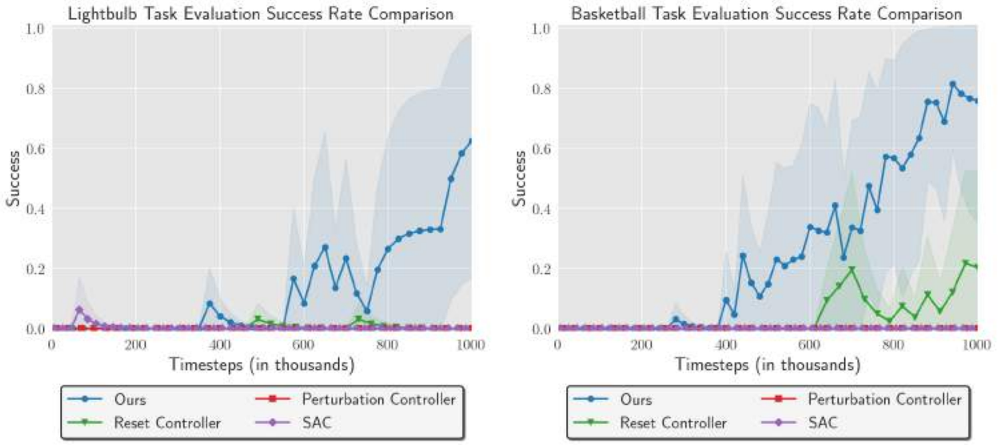


Figure 14.8: Comparison of *MTRF* with baseline methods in simulation when run without resets. In comparison to the prior reset-free RL methods, *MTRF* is able to learn the tasks more quickly and with higher average success rates, even in cases where none of the prior methods can master the full task set. *MTRF* is able to solve all of the tasks without requiring any explicit resets.

We focus our experiment on the following questions:

1. Are existing off-policy RL algorithms effective when deployed under reset-free settings to solve dexterous manipulation problems?

2. Does simultaneously learning a collection of tasks under the proposed multi-task formulation with *MTRF* alleviate the need for resets when solving dexterous manipulation tasks?

3. Does learning multiple tasks simultaneously allow for reset-free learning of more complex tasks than previous reset free algorithms?

4. Does *MTRF* enable real-world reinforcement learning without resets or human interventions?

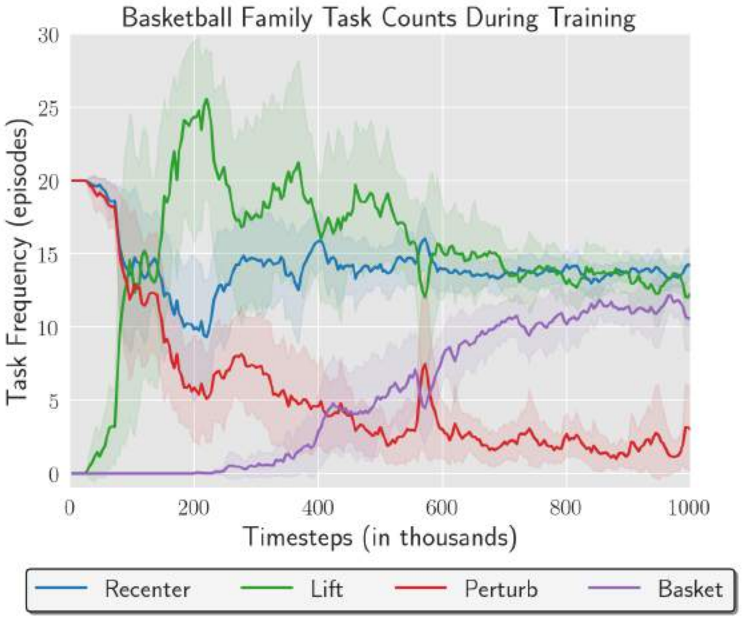


Figure 14.9: Visualization of task frequency in the basketball task Family. While initially recentering and pickup are common, as these get better they are able to provide resets for other tasks.

Baselines and Prior Methods

We compare *MTRF* (Section 15.3) to three prior baseline algorithms. Our first comparison is to a state-of-the-art off-policy RL algorithm, soft actor-critic [23] (labeled as *SAC*). The actor is executed continuously and reset-free in the environment, and the experienced data is stored in a replay pool. This algorithm is representative of efficient off-policy RL algorithms. We next compare to a version of a reset controller [394] (labeled as *Reset Controller*), which trains a forward controller to perform the task and a reset controller to reset the state back to the initial state. Lastly, we compare with the perturbation controller technique [98] introduced in prior work, which alternates between learning and executing a forward task-directed policy and a perturbation controller trained purely with novelty bonuses [110] (labeled as *Perturbation Controller*). For all the experiments we used the same RL algorithm, soft actor-critic [23], with default hyperparameters. To evaluate a task, we roll out its final policy starting from states randomly sampled from the distribution induced by all the tasks that can transition to the task under evaluation, and report performance in terms of their success in solving the task. Additional details, videos of all tasks and hyperparameters can be found at [`https://sites.google.com/view/mtrf`].

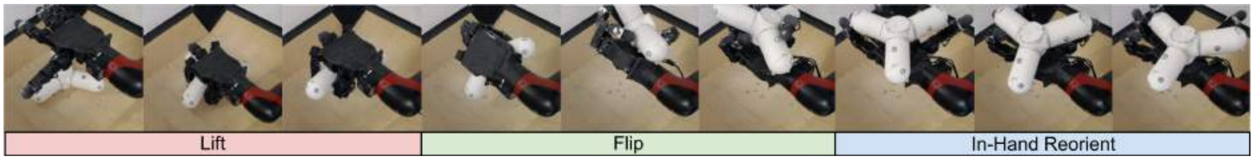


Figure 14.10: Film strip illustrating partial training trajectory of hardware system for in-hand manipulation of the valve object. This shows various behaviors encountered during the training - picking up the object, flipping it over in the hand and then in-hand manipulation to get it to a particular orientation. As seen here, MTRF is able to successfully learn how to perform in-hand manipulation without any human intervention.

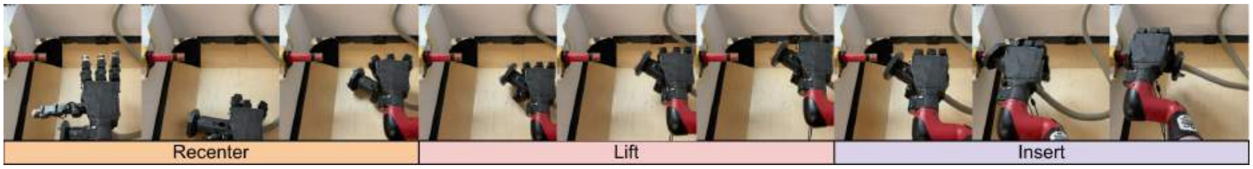


Figure 14.11: Film strip illustrating partial training trajectory of hardware system for pipe insertion. This shows various behaviors encountered during the training - repositioning the object, picking it up, and then inserting it into the wall attachment. As seen here, MTRF is able to successfully learn how to do pipe insertion without any human intervention.

Reset-Free Learning Comparisons in Simulation

We present results for reset-free learning, using our algorithm and prior methods, in Fig 14.8, corresponding to each of the tasks in simulation in Section 14.3. We see that *MTRF* is able to successfully learn all of the tasks jointly, as evidenced by Fig 14.8. We measure evaluation performance after training by loading saved policies and running the policy corresponding to the "forward" task for each of the task families (i.e. lightbulb insertion and basketball dunking). This indicates that we can solve all the tasks, and as a result can learn reset free more effectively than prior algorithms.

In comparison, we see that the prior algorithms for off-policy RL – the reset controller [394] and perturbation controller [98] – are not able to learn the more complex of the tasks as effectively as our method. While these methods are able to make some amount of progress on tasks that are constructed to be very easy such as the pincer task family shown in Appendix B, we see that they struggle to scale well to the more challenging tasks (Fig 14.8). Only *MTRF* is able to learn these tasks, while the other methods never actually reach the later tasks in the task graph.

To understand how the tasks are being sequenced during the learning process, we show task transitions experienced during training for the basketball task in Fig 14.9. We observe that early in training the transitions are mostly between the recenter and perturbation tasks. As *MTRF* improves, the transitions add in pickup and then basketball dunking, cycling between re-centering, pickup and basketball placement in the hoop.

Learning Real-World Dexterous Manipulation Skills

Next, we evaluate the performance of MTRF on the real-world robotic system described in Section 14.3, studying the dexterous manipulation tasks described in Section 14.3.

In-Hand Manipulation Let us start by considering the in-hand manipulation tasks shown in Fig 14.6. This task is challenging because it requires delicate handling of finger-object contacts during training, the object is easy to drop during the flip-up and in-hand manipulation, and the reorientation requires a coordinated finger gait to rotate the object. In fact, most prior work that aims to learn similar in-hand manipulation behaviors either utilizes simulation or employs a hand-designed reset procedure, potentially involving human interventions [33], [290], [389]. To the best of our knowledge, our work is the first to show a real-world robotic system learning such a task entirely in the real world and without any manually provided or hand-designed reset mechanism. We visualize a sequential execution of the tasks (after training) in Fig 14.10, and encourage the reader to watch a video of this task, as well as the training process, on the project website: [`https://sites.google.com/view/mtrf`]. Over the course of training, the robot must first learn to recenter the object on the table, then learn to pick it up (which requires learning an appropriate grasp and delicate control of the fingers to maintain grip), then learn to flip up the object so that it rests in the palm, and finally learn to perform the orientation. Dropping the object at any point in this process requires going back to the beginning of the sequence, and initially most of the training time is spent on re-centering, which provides resets for the pickup. The entire training process takes about 60 hours of real time, learning all of the tasks simultaneously. Although this time requirement is considerable, it is entirely autonomous, making this approach scalable even without any simulation or manual instrumentation. The user only needs to position the objects for training, and switch on the robot.

For a quantitative evaluation, we plot the success rate of sub-tasks including re-centering, lifting, flipping over, and in-hand reorientation. For lifting and flipping over, success is defined as lifting the object to a particular height above the table, and for reorient success is defined by the difference between the current object orientation and the target orientation of the object. As shown in Fig 14.12, *MTRF* is able to autonomously learn all tasks in the task graph in 60 hours, and achieves an 70% success rate for the in-hand reorient task. This experiment illustrates how *MTRF* can enable a complex real-world robotic system to learn an in-hand manipulation behavior while at the same time autonomously retrying the task during a lengthy unattended training run, without any simulation, special instrumentation, or manual interventions. This experiment suggests that, when *MTRF* is provided with an appropriate set of tasks, learning of complex manipulation skills can be carried out entirely autonomously in the real world, even for highly complex robotic manipulators such as multi-fingered hands.

Pipe Insertion We also considered the second task variant which involves manipulating a cylindrical pipe to insert it into a hose attachment on the wall as shown in Fig 14.7. This task is challenging because it requires accurate grasping and repositioning of the object during training in order to accurately insert it into the hose attachment, requiring coordination of both the arm and the hand.

Training this task without resets requires a combination of repositioning, lifting, insertion and removal to continually keep training and improving. We visualize a sequential execution of the tasks (after training) in Fig 14.11, and encourage the reader to watch a video of this task, as well as the training process, on the project website: [`https://sites.google.com/view/mtrf`]. Over the course of training, the robot must first learn to recenter the object on the table, then learn to pick it up (which requires learning an appropriate grasp), then learn to actually move the arm accurately to insert the pipe to the attachment, and finally learn to remove it to continue training and practicing. Initially most of the training time is spent on re-centering, which provides resets for the pickup, which then provides resets for the insertion and removal. The entire training process takes about 25 hours of real time, learning all of the tasks simultaneously.

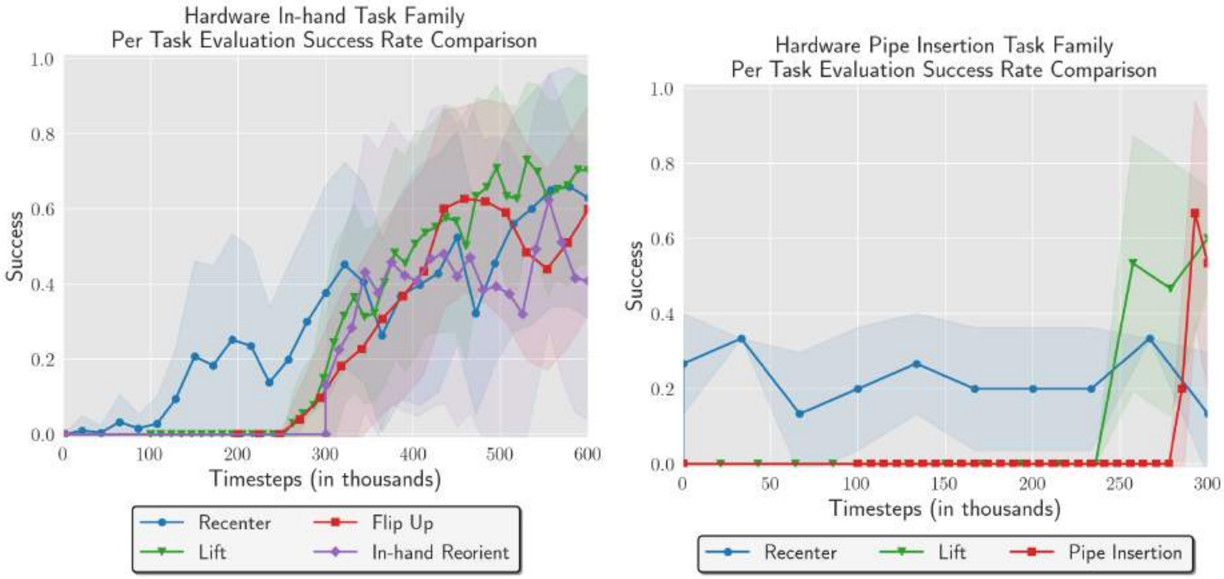


Figure 14.12: Success rate of various tasks on dexterous manipulation task families on hardware. **Left: In-hand manipulation** We can see that all of the tasks are able to successfully learn with more than 70% success rate. **Right: Pipe insertion** We can see that the final pipe insertion task is able to successfully learn with more than 60% success rate.

14.5 Discussion

In this work, we introduced a technique for learning dexterous manipulation behaviors reset-free, without the need for any human intervention during training. This was done by leveraging the multi-task RL setting for reset-free learning. When learning multiple tasks simultaneously, different tasks serve to reset each other and assist in uninterrupted learning. This algorithm allows a dexterous manipulation system to learn manipulation behaviors uninterrupted. Our experiments show that this approach can enable a real-world hand-arm robotic system to learn an in-hand reorientation task, including pickup and repositioning as well as a pipe insertion task without human intervention or special instrumentation. One major drawback of the approach as stated is the fact that it still requires human specification of how the different tasks are sequenced one after another through the task graph. This gets hard to scale as the number of tasks increases and the interaction between them gets more complex. In the following section, we show that this task graph can be inferred from data by leveraging a small amount of human data.

Chapter 15

Bootstrapping Reset-Free Reinforcement Learning Algorithms with Human Data

As alluded to in the previous section, multi-task RL can provide a good way to bootstrap continual autonomy. However, the question when learning in a multi-task framework, without human intervention becomes —how do we know how to sequence the tasks to learn continually, with minimal human intervention. In this section, we show that with a small amount of human data provided upfront, we can actually learn not just how to bootstrap low level policies but also the autonomous practicing scheme itself. In doing so, we show that we can actually practice for extended periods of time with directed data collection, learning how to perform long horizon tasks in a realistic kitchen.

15.1 Introduction

Teaser FINAL.png

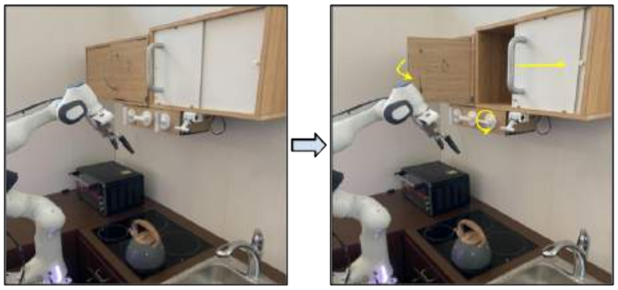


Figure 15.1: Robot setup for real world training in a kitchen. The agent needs to manipulate multiple different elements to accomplish complex goals.

Reinforcement learning (RL) algorithms are a promising tool for robotic learning, in principle providing a straightforward technique for acquiring complex behaviors, simply through trial and error interaction. The framework of RL is particularly appealing for robotics in that it allows an agent to collect its own experience, improving autonomously in new environments. However, many current real-world examples of RL are quite far from this promise of autonomy, often requiring considerable amounts of human effort and engineering to enable the learning process to proceed. One of such requirements – a resettable initial state distribution, while easy to satisfy in simulation, do not actually exist in the real world, i.e. providing episodic resets can often be tedious and time-consuming and requires a constant supervision of a human operator. Alleviating some of these requirements would allow us to explore the full potential of RL methods to build plug-and-play learning systems where agents are dropped into environments and continue improving.

To illustrate this more concretely, imagine a robot that is deployed in a previously unseen kitchen, such as the one shown in Fig. 15.1. Since this is an unfamiliar environment, in which a robot could damage its surroundings, the agent could prompt a user to provide a set of easy-to-collect, continuous demonstrations, illustrating example behaviors and goal states that the robot should be able to acquire and robustly achieve. Given a small number of such examples, the robot should be able to extract all the necessary information required to continually practice a wide range of tasks in the kitchen so that it can perfect them entirely on its own, with minimal human intervention.

Although there are a number of challenges that are preventing us from making this scenario a reality such as access to reward functions, difficulties with safe exploration or the resettability of the environment, in this paper we focus on the latter two. While several prior works have studied continual reset-free training by utilizing auxiliary reset controllers [98] and multiple scaffolding tasks [378] to enable a skill to be learned autonomously from scratch, our focus in this work is on enabling easy-to-specify autonomous reset-free training of multi-skill repertoires by utilizing the multi-task nature of the problem and a small amount of human-provided data. We show that, in exchange for a small amount of supervision, we can lift two major constraints on real-world unattended learning: bootstrapping exploration and selecting which tasks to practice so as to enable reset-free training. More concretely, let us again consider a robotic arm operating in a kitchen as shown in Fig. 15.1. In this environment, we can specify different tasks by the ability to reach different goal states that can be extracted from human-provided data, such as the two example images in Fig. 15.1 (cabinet and slider closed or cabinet and slider open). Given this definition of a task, reset-free demonstration-augmented multi-task RL can obtain initial behaviors by utilizing the human-provided demonstrations with offline reinforcement learning. Moreover, it can utilize the same data to learn how to sequence different tasks one after the other so as to practice all of them together with minimal number of resets, with some tasks providing resets for others, in the process improving behavior on all the tasks. For instance, if the robot is trying to practice opening and closing the cabinet, it may start in a state where the cabinet is closed and command the task to open it. Even if this task does not succeed and the cabinet is partially open, the agent can continue practicing any of the remaining tasks including closing or opening the cabinet so as to minimize the requirement for resets, naturally resetting the environment by just solving different tasks. One of our observations is that the mechanism needed to *reset* the environment to facilitate practicing during unattended training is the same as the mechanism we need to sequence multiple subgoals for completing long-horizon tasks. Informed by this observation, we also show how this system is able to accomplish long-horizon behaviors by sequencing multiple tasks.

In this paper, we present a novel robotic learning system, which we call demonstration bootstrapped autonomous practicing (DBAP), that is able to incorporate a small amount of human data provided up front to bootstrap long periods of autonomous learning with just a few human interventions, using some tasks to reset others. We show that just a few hours of easily provided "play" style [249], [346] multi-task demonstrations can make the entire learning process more efficient, bootstrapping both policy learning and autonomous practicing of behaviors. Moreover, we demonstrate that the provided data can also aid in the acquisition of goal-directed high-level controllers that can solve multi-step long horizon problems at evaluation time by sequencing individual tasks. Our experiments indicate that this paradigm allows us to efficiently solve complex multi-task robotics problems both in simulation and in the real world, with an order of magnitude less human intervention needed during learning.

15.2 Preliminaries and Problem Statement

In this work, we focus on a special case of multi-task RL, where the task can be represented as a state goal s_g that the policy has to reach to obtain the reward, which is often referred to as goal-conditioned RL. We assume to be given a set of N goal states that are of particular interest: $S_{g_i} = \{s_g^i\}_{i=1}^N$ that the agent is expected to be able to reach from any other state of interest, that is, the agent has the following objective: $\mathbb{E}_{s_0=s_g^i, a_t\sim\pi(a_t|s_t,s_g^j)} \sum_{t=1}^T r(s_t, a_t, s_g^j) \quad \forall s_g^i \in S_{g_i}, s_g^j \in S_{g_i}$. For example, given two goal states of interest: s_g^1 corresponding to the state where both a microwave and a cabinet are closed and and s_g^2 where the microwave and cabinet are open, our goal is to learn a policy that transitions between these two goal states by manipulating the cabinet and the microwave. We often refer to reaching these goals of interests as *tasks* since reaching such states usually corresponds to accomplishing a meaningful task such as opening a cabinet.

In addition, since we are operating in the real world, it is cumbersome for a human supervisor to provide resets after every episode. Instead, to allow for more autonomous and scalable learning, we assume that a reset can be provided every n episodes. Our objective is to operate in this non-stationary setting and learn how to practice tasks mostly autonomously with only occasional resets.

15.3 Demonstration Augmented Autonomous Practicing for Multi-Task Reinforcement Learning

Our method, Demonstration-Bootstrapped Autonomous Practicing (DBAP), leverages the human-provided demonstration data in two ways: to bootstrap a multi-task policy that eventually learns to perform the full repertoire of the demonstrated behaviors, and to enable a graph-based sequencing mechanism, which chooses which tasks to command in order to attain uniform coverage over possible behaviors and thus practice the full repertoire of tasks during autonomous reset-free reinforcement learning. Furthermore, this same sequencing mechanism can also enable the robot to accomplish long-horizon tasks at test-time, by sequencing the subgoals needed to perform those tasks. An overview is presented in Figure 15.2. Next, we describe our reset-free reinforcement learning procedure, followed by a discussion of how tasks can be sequenced via graph search for both autonomous practicing and performing long-horizon tasks.

Bootstrapping Reset-Free Multi-Task Learning with Prior Data

Our goal is to use multi-task learning to enable reset-free learning, bootstrapping from human provided prior data. We learn a single goal-conditioned policy for all tasks, denoted $\pi_\theta(a|s, s_g)$, where s_g is a goal state. To learn this goal-conditioned policy, we instantiate a goal-conditioned version of

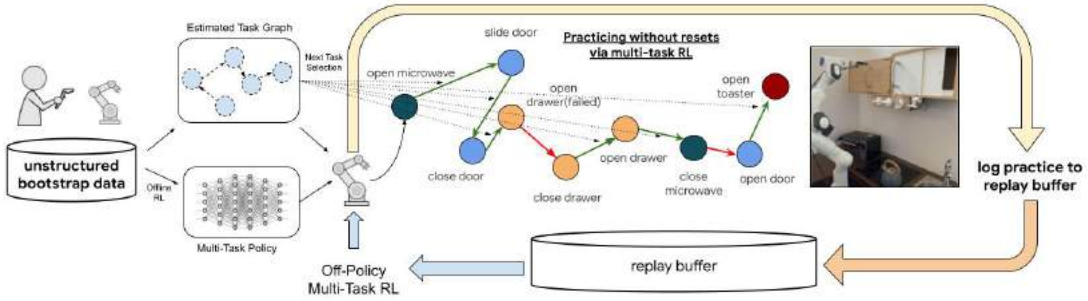


Figure 15.2: Overview of our method. Given a set of human provided unstructured demonstrations, the system bootstraps a multi-task RL policy via offline RL and builds a task graph that models transitions between different tasks. The system then practices autonomously with small number of resets by leveraging multi-task RL, using the learned task graph to command the appropriate next task. The resulting multi-task policy and estimated task graph can then be used to solve long-horizon problems at test-time.

AWAC [421], where the policy is provided with the state goal s_g alongside the current state s, yielding the following policy update: $\theta_{k+1} = \arg\max_\theta \mathbb{E}_{s,a,s_g} \left[\log \pi_\theta(a|s, s_g) \exp\left(\frac{1}{\lambda} A^{\pi_\theta}(s, a, s_g)\right)\right]$, where A^{π_θ} is an advantage function for the current, goal-conditioned policy π_θ.

To enable autonomous practicing to collect the data needed to improve the goal-conditioned policy $\pi_\theta(a|s, s_g)$ in a reset-free manner, we also require a task-sequencing policy that determines how the goals are sequenced one after another. To accomplish that, we introduce a task-sequencing policy $q(s_g|s, s_g^{\text{desired}})$ that decides which goal of interest s_g to command next from the current state. On closer inspection, we see that this resembles high-level policies used in hierarchical RL that are commonly used to sequence individual subgoals for accomplishing long-horizon goals. However, in this scenario, $q(s_g|s, s_g^{\text{desired}})$ is not just used to accomplish long-horizon goals but also to enable autonomous practicing. In the following sections we will often refer to the multi-task policy $\pi_\theta(a|s, s_g)$ as the low-level policy, and the task sequencer $q(s_g|s, s_g^{\text{desired}})$ as the high-level policy since it commands what task should be executed next for autonomous practicing or long-horizon goal reaching. Next, we describe how we can instantiate and utilize $q(s_g|s, s_g^{\text{desired}})$ for autonomous learning.

Task Sequencing via Graph Search

To learn a task sequencer policy q that would allow for autonomous practicing of the low-level multi-task policy π_θ and facilitate long-horizon goal reaching, we propose a simple model-based graph search algorithm. The key idea in our approach is to leverage prior data to learn which low-level task transitions are *possible* and can be sequenced, and then use this knowledge to optimize for autonomous practicing and goal sequencing. In particular, we utilize the provided data to build a directed task graph $\mathcal{G}$, with vertices as different goal states of interest s_g^i, s_g^j, and an adjacency matrix A with $A(i, j) = 1$ if there exists a transition between particular states of interest s_g^i and s_g^j in the demonstration data, and $A(i, j) = 0$ otherwise. This graph can be thought of as a discrete high-level model, which represents how different goal states of interest are connected to each other. Given this graph G acquired from the prior data, we can then utilize it for both autonomous practicing of

Algorithm 13: PracticeGoalSelect: High Level Task Selection via Graph Search

```
Input: task-graph $\mathcal{G}$, goal-states $\{s_g^0, .., s_g^N\}$, density over goals $\rho(s_g^i)$, current state $s$
Output: Next goal $s_g^{\text{next}}$ to command

  // Initialize maximum entropy value
  $\mathcal{H}_{\text{max}} = -\infty$
  for $s_g^i \in \{s_g^1, \dots, s_g^N\}$ do
      $\tau \leftarrow$ Dijkstra$(s, s_g^i, \mathcal{G})$
      $\rho' = \rho + \tau$ // Compute updated density
      if $\mathcal{H}(\rho') \geq \mathcal{H}_{\text{max}}$ then
          $\mathcal{H}_{\text{max}} = \mathcal{H}(\rho')$
          $s_g^{\text{next}} \leftarrow \tau[1]$ // First step of path to $s_g^i$
      end if
  end for
  return $s_g^{\text{next}}$
```

Algorithm 14: Overview of DBAP

```
Input: Human provided multi-task demonstrations $\mathcal{D}$
Output: Multi-task policy $\pi_\theta(a|s, s_g)$ and task-graph $\mathcal{G}$

  Estimate task graph $\mathcal{G}$ via counting
  Initialize $\pi_\theta(a|s, s_g)$ via AWAC [421]
  Initialize current density $\rho = 0$
  for t = 0, . . . , $N$ steps do
      // Select next goal via graph search
      $s_g^{\text{next}} \leftarrow \text{PracticeGoalSelect}(s, \rho)$
      Rollout $\pi_\theta(a|s, s_g^{\text{next}})$
      Add collected data to replay buffer $\beta$
      Update $\pi_\theta$ via AWAC.
  end for
  return $\pi_\theta(a|s, s_g)$, $\mathcal{G}$
```

low-level tasks and for sequencing multiple low-level tasks to achieve multi-step goals.

Autonomous practicing. The task graph $\mathcal{G}$ can be used to direct autonomous practicing by explicitly optimizing to command goals that bring the overall coverage over states close to the uniform distribution, when operating with minimal resets. More concretely, for goal selection during practicing, the algorithm iterates through all possible goal states of interest, determines the shortest path to the goal state via Dijkstra's algorithm [422] from the current state, and computes the resulting densities over goal states if that path was chosen. The path that results in bringing the density closest to uniform (maximum entropy) is then picked, and the first step in the path is commanded as the next goal state. This is done to ensure that the algorithm maintains roughly uniform coverage over different goal states of interest in the environment, so that all tasks can be practiced equally. This process repeats at the next step, thereby performing receding horizon control to maintain the density over potential goal states of interest to be as close to uniform as possible. Formally, the objective being optimized to select which goal to sequence next is given by: $\max_{s_g^i \in S_g^i} \mathcal{H}(\mathcal{U}, \rho + \text{Djikstra}(s, s_g^i))$, where ρ is the current marginal distribution (represented as a categorical distribution) over goal states of interest, Dijkstra(s, s_g^i) computes the shortest distances between current state s and s_g^i and the goal is to bring the updated density as close to uniform $\mathcal{U}$ as possible[1]. A detailed description of the task sequencing algorithm via graph search can be found in Algorithm 13.

Task sequencing for multi-step tasks. While the scheme for autonomous practicing aims to choose goals that maintain uniform coverage over states so as to improve performance on *all* tasks,

[1] We overload the $+$ operator here to denote an update to the density ρ when accounting for new states

it can also be used to accomplish long-horizon goals. When the agent is tasked with reaching a particularly distant goal at evaluation time, which standard RL algorithms tend to struggle with, we note that the same graph can be reused to sequence appropriate sub-goals to reach the specific goal state.

In particular, given a target goal of interest s_g^j, when the agent is at a particular state s, we can use the estimated graph $\mathcal{G}$ to compute the shortest path between goal states of interest $\tau = \left[s, s_g^1, s_g^2, \ldots, s_g^j\right]$ to the goal via Dijkstra's algorithm. The next goal state of interest in the path s_g^1 is then chosen as the next goal commanded to the multi task policy $\pi(a|s, s_g^1)$ and executed for a single episode till the agent reaches s_1. This procedure is then repeated till the agent accomplishes s_g^j. This procedure allows the agent to not just practice short horizon problems but also sequence multiple short-horizon goals to solve long-horizon ones. Further details on these procedures can be found in Algorithm 13 and Algorithm 14.

15.4 System Description

We evaluate our method on a set of manipulation tasks in both a simulated and real world kitchen environment. In this section, we describe both these setups.

Real World Environment

To evaluate DBAP in the real world, we built a physical kitchen environment based on the kitchen manipulation benchmark described by [346].

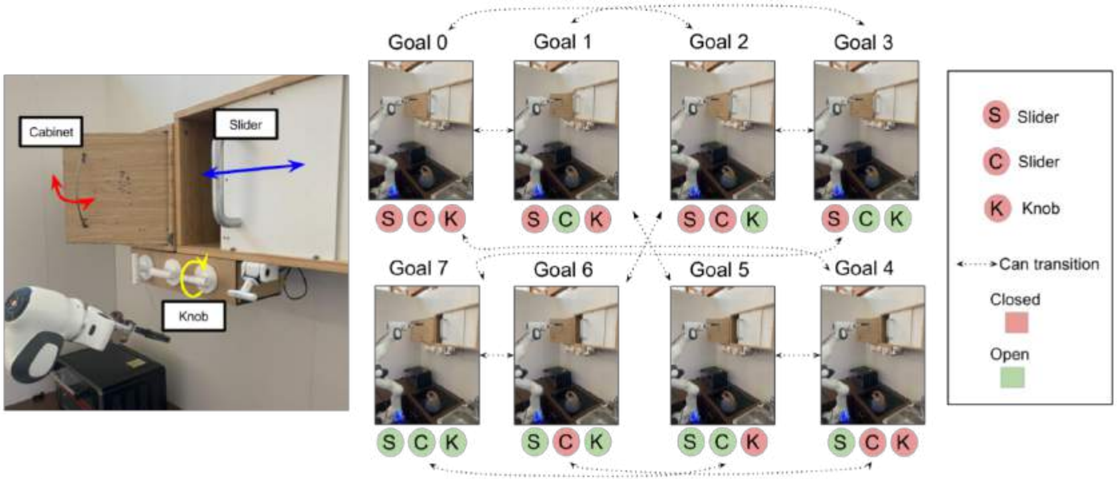


Figure 15.3: Elements, tasks and goal states in the real world kitchen environment. The agent is manipulating the cabinet, slider and knob to accomplish particular configurations of the environment as shown from goal 0 to 7 here. The dotted lines represent individual transitions that are feasible within the length of an episode, toggling one element at a time between it's extreme positions. The goal of the agent is to learn a policy and a graph controller that is able to transition between goal states.

Tasks. In this environment, we task a 7 DoF Franka Emika robotic arm with manipulating three distinct elements: a sliding door that can be slid open or closed, a cabinet door that can be open or closed and a knob that can rotate to control the stove burners, as shown in Fig 15.3. The arm is controlled with low-level end-effector pose commands at 5 Hz. At each time step, the policy chooses a position and orientation for the end effector, as well as a command for opening and closing the gripper. These three elements represent distinct types of motion, each of which require different control strategies. The goal states of interest s_g^i are defined as various combinations of the elements being opened or closed (or in the case of the knob, turned by $0 or 90$), resulting in $2^3 = 8$ goal states based on various combinations of the elements being open or closed (Fig 15.3). As described in Sections 15.2, 15.3, the agent must practice reaching the goals of interest autonomously in the environment to master going from any combination of element configurations to any other. The agent is said to be in one of the goal states of interest if the current state of the elements in the scene are within ϵ distance of the particular goal state of interest.

Data collection. We make use of a teleoperation system to provide a continuous sequence of multi-task demonstrations. We collect "play-style" [249], [346] demonstrations, where different tasks are attempted successfully one after the other, indicating both how tasks are solved and how they may be sequenced. While prior work [249], [346] assumes that we are provided with unsegmented demonstrations, we make a simple change to the data collection procedure where the user indicates when a particular goal state of interest is completed before transitioning over to demonstrating a different goal. This allows the algorithm to easily determine the goals of interest as the transition points between these human-provided demonstrations. We provide around 500 demonstrations in the real world, requiring 2.5 hours of data collection time.

Tasks.png

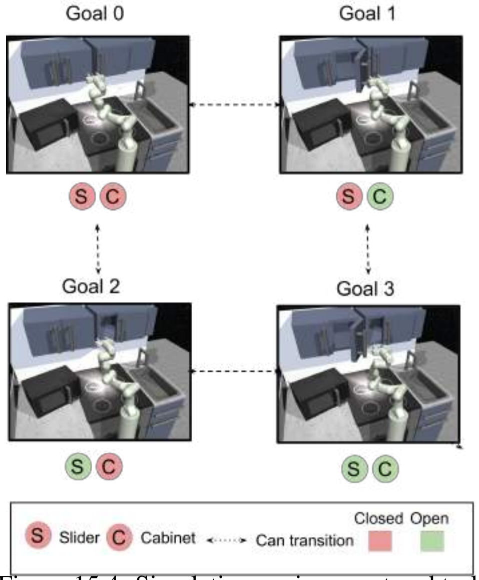


Figure 15.4: Simulation environment and tasks. 2 elements —the slider and the cabinet are being manipulated

Simulation Environment

To provide thorough quantitative comparisons, we also evaluate our method on a simulated version of the above task, based on the MuJoCo kitchen manipulation environment described out by [346]. The purpose of this evaluation is to study the behavior of different components of the system in more detail and more systematically run comparisons. In particular, in simulation we consider tasks with 2 elements (Fig 15.4): the cabinet and the slider. The goal states correspond to combinations of the cabinet and slider being open and closed.

15.5 Experimental Evaluation

In our experimental evaluation, we aim to answer the following questions: **(1)** Does DBAP enable improvement with a small amount of human intervention, and how does it compare to prior methods for reset-free or demonstration-augmented RL? **(2)** Is DBAP able to bootstrap effectively from prior data both at the low-level policy execution and at the high level practicing behavior? **(3)** Does DBAP allow agents to learn how to perform long horizon behaviors in the real world?

To understand the decisions made in DBAP , we compare to a number of baselines in simulation:

Non-pretrained, graph search task selection: This is a version of our algorithm, where the human provided data is not used to bootstrap low level policies, but only the high level graph. This is roughly related to ideas from [378], where multiple policies are learned from scratch, but unlike that work this baselne uses a learned high level graph.

Pretrained low-level, random high level controller: This is a multi-task analogue to the perturbation controller [98]. The low-level policy is pre-trained from human demonstrations, but the high-level practicing chooses randomly which task to execute.

Pretrained low-level, BC task selection: This is a version of the method by [346], modified for the reset-free setting. Rather than using random task selection for practicing, tasks are sequenced using a high level policy trained with supervised learning (behavior cloning) on the collected data to command tasks.

Pretrained low-level, reset controller: This baseline is similar to a reset controller [394], [395], and alternates between commanding a random task and commanding a return to a single fixed start state during practicing. In this case, the high-level policy is simply a random controller.

For real world experiments, we compare the performance of DBAP to standard imitation learning (behavior cloning) without any finetuning (labeled as “Imitation" in Table 15.2), standard offline RL with AWAC without any finetuninig (labeled as “Offline RL" in Table 15.2) and the no pre-training baseline described above. The policies and models used in this work operate on a low-level state and they are represented with multi-layer perceptrons using 3 layers of 256 units each. We use the standard set of hyperparameters available in the open-source AWAC [421] implementation. More details can be found in the supplementary materials and on the website `https://sites.google.com/view/dbap/`

Evaluation metrics. We explicitly evaluate on 2 different metrics. The first metric captures the ability to solve short horizon single-step tasks, using the low-level multi-task policy $\pi(a|s, s_g)$ to reach a specific goal s_g. For instance, in Fig 15.3, we can evaluate the success rate on reaching goal 0 from goal 1 or goal 1 from goal 3 and so on, each of which involves manipulating a single element in the environment. We report the average success rate across all the possible single-step tasks, and refer to this as “short horizon success”. Secondly, we evaluate the ability of the task sequencer q to chain multiple low-level goal reaching behaviors to accomplish a long-horizon goal. For instance in Fig 15.3, we can measure the success of the task sequencer at reaching goal 7 from goal 0, goal 6 from goal 1, and so on. This requires manipulating multiple elements. We report the average success rate across all the possible 3-step tasks in the real environment as the “long horizon success” rate.

Autonomous Demonstration-Bootstrapped Practicing in Simulation

First, we evaluate the ability of DBAP to learn how to perform tasks in the environment in simulation as described in Section 15.4, with a minimal amounts of automatically provided resets. In this work, we assume access to one human reset every 10 episodes, reducing the amount of human

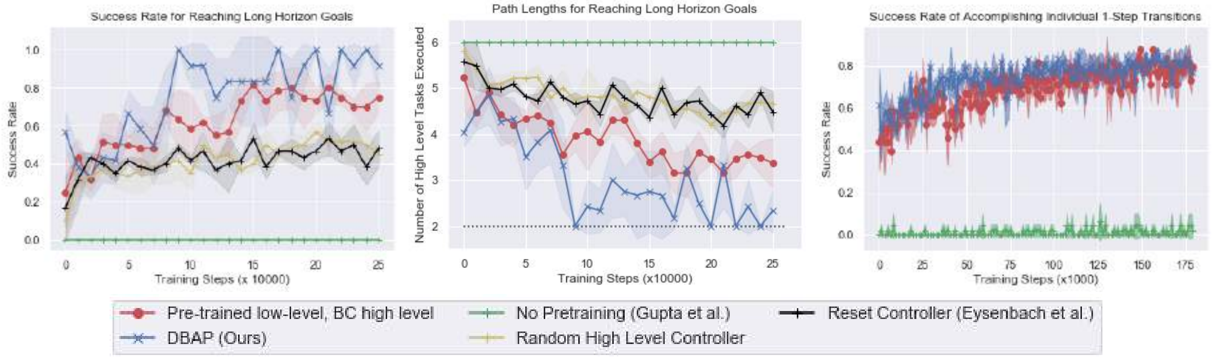


Figure 15.5: Performance of DBAP on the simulated environment in Fig 15.4 **(Left)** Success rate averaged across 3 seeds each on long horizon goal reaching **(Middle)** Average number of sub-goals commanded to reach a long horizon goal **(Right)** Success rate on short horizon goal reaching. We can see that DBAP performs better than several prior methods improving with minimal amounts of human intervention to solve multi-step problems.

supervision by an order of magnitude as compared to standard RL. We can see from Fig 15.5 that autonomous practicing via DBAP is able to improve the short horizon success rate from 50% to 85% in simulation. This performance is significantly better than not using the prior data for bootstrapping. This suggests that the important part for improving the performance of single step tasks is that the practicing, especially in its initial phases, is bootstrapped from human-provided data as described in Section 15.3.

In the next experiment, we study how well a robotic agent is able to accomplish multi-step goal reaching. As we find in Fig 15.5 (left), our method is able to successfully reach a variety of multi-step goals and shows improvement over time with minimal human involvement, improving from a success rate of 50% to around 90% on the long horizon success rate. While other techniques are able to occasionally succeed, they take significantly longer to get to the goal, often resulting in roundabout paths to achieve the goal. This leads to the extended path length of trajectories in Fig 15.5 (middle). In particular, training a high level policy via goal relabeling and behavior cloning (the "pretrained low-level, BC task selection" baseline in Fig 15.5. can inherit the biases of the human provided data. If the play data demonstrates sub-optimal paths to reach long horizon goals, this is also reflected in the performance of the high-level behavioral cloning. In contrast, our graph-based search method is able to find and practice shorter paths than the baselines. Importantly, this improvement is acquired with a minimal requirement for human effort in the process.

Autonomous Demonstration-Bootstrapped Practicing in the Real World

	Success Rate	**Path Length**
Offline RL	0.83 ± 0.058	3.5 ± 0.17
DBAP (Ours)	$\mathbf{0.95 \pm 0.05}$	$\mathbf{3.37 \pm 0.3}$
Imitation	$0.62 \pm 0.$	4.3 ± 0.15
No Pretraining	0.0 ± 0.0	6.0 ± 0.0

Table 15.1: Success rates and path lengths (in number of steps) in the real world for reaching long horizon goals.

	Success Rate
Offline RL	0.708 ± 0.058
DBAP (Ours)	$\mathbf{0.94 \pm 0.019}$
Imitation	0.763 ± 0.051
No Pretraining	0.0 ± 0.0

Table 15.2: Success rates for reaching short horizon goals in the real world.

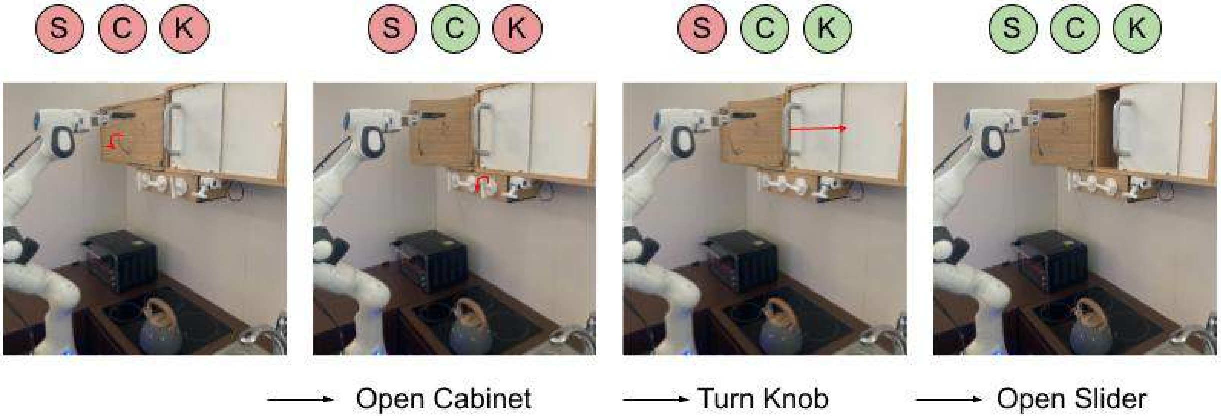


Figure 15.6: Film strip of multi-step behavior in the kitchen environment. We see a successful execution of multi-step behavior transitioning between having all elements in the kitchen closed to all open, by first having the graph search command the agent to open the cabinet, then turn the knob and finally open the slider.

Next, we evaluate the ability of DBAP to learn multiple goal-reaching tasks in the real world. Bootstrapped from the human-provided trajectories, we trained the robot for over 25 hours to learn the tasks described in Section 15.4. As seen from the evaluation performance of offline RL on long horizon success in Table 15.1, DBAP starts off doing well even during pre-training, achieving a 83% success rate, but improves significantly during autonomous practicing to around a 95% success rate, indicating the importance of online finetuning. In Fig. 15.6, we show an example where the robot must reach a goal with slider, cabinet, and microwave door all open, from a state where all three are closed. The graph search automatically commands the low-level policy to reach each of the subgoals for the complete task: opening the cabinet, turning the knob, and opening the slider.

In comparison, we see that the no-pretraining baseline that trains from scratch does very poorly, resulting in 0% success rate. DBAP also significantly outperforms standard imitation learning in terms of long horizon goal-reaching performance. We additionally compare the number of tasks commanded to accomplish the multi-step goals and we can observe that the average path length is significantly lower for our method than the alternatives. We can see that a similar trend is also reflected in the success percentages of short horizon goals in Table 15.2.

15.6 Discussion

We described the design principles behind a novel system for learning behaviors autonomously using RL. We leverage small amounts of human-provided data to bootstrap a multi-task RL system, using some tasks to provide resets for others when learning autonomously. We show that our scheme allows for a robot to not only use prior data to bootstrap low-level policies but also facilitate the autonomous practicing behavior itself. This allows for improvement on individual tasks as well as sequencing of these tasks to reach multi-step, long-horizon goals. We demonstrated this algorithm on both simulated and real-world kitchen manipulation problems. We finally place this work in the context of the broader space of related work and show how this ties into the bigger picture of real world robotic learning.

Chapter 16

Relationship to Other Work on Continual Data Collection in Reinforcement Learning

Next we try and place this work in context of some related work both prior to and post publishing of our work.

Building large scale continual data collection systems with RL is a challenging problem both from a systems and an algorithms standpoint. Some examples of excellent large scale RL systems that we used as inspiration include [34], [35], [38], [316], [413], [415], [423], [424], which show the ability for real world grasping and manipulation tasks to scale up to many many hours of real world learning. Our work aims to build systems that can do a broader set of dexterous manipulation tasks, with a similar mindset of large scale data collection. Our work builds on prior work on reset-free RL as well [394], [395]. These works try to learn explicit reset controllers to reset the environment to a particular state so that a task can be attempted again. We show that this notion can be generalized, both through the lens of the perturbation controller, which resets to arbitrarily random states rather than a single state, and the idea of multi-task RL as a solution to reset-free RL. Our line of work is closely connected to that shown by [425] in the context of locomotion, solving tasks with minimal effort using multi-task RL. Since the publishing of our work, recent work has shown that the ideas of reset-free RL can be extended to a framework for adversarial reset task selection [426], and for automatic curriculum generation [427]. We hope that this work provides some design principles for the construction of other large scale RL systems in the future.

Chapter 17

Conclusion

Through the course of this thesis, we studied the mismatch between the assumptions made by most reinforcement learning algorithms and what is actually available in the real world. By analyzing this through the lens of data, we showed that the challenge can be broken down into that of obtaining the right supervision, that of ensuring safe and efficient collection of data from the right distribution and the challenge of ensuring large scale, continual data collection with minimal human intervention. We showed that through a combination of data driven reward inference, bootstrapping from small amounts of human provided data and multi-task algorithms for reset-free reinforcement learning, we were able to move towards systems that can actually learn and continually improve in the real world. We demonstrated these techniques on a variety of real world platforms, ranging from dexterous manipulation to kitchen manipulation to tabletop manipulation and even to tasks in mobile manipulation. In doing so, we showed the versatility of the classes of approaches we introduced and the potential to scale to the robots of the future in our homes, hospitals and shopping malls.

Besides the work covered in detail in this thesis, I was fortunate to be able to explore several other directions that are likely to be crucial when deploying robots into truly unstructured environments. One direction that is particularly compelling is that of multi-task RL and meta-learning to transfer policies [224], [419] or exploration strategies [213] across different tasks and environments. The future is likely to move towards generalist agents, and being able to actually train agents to solve multiple tasks, using shared data, policies and representations is likely to be crucial. In the spirit of transfer learning, we also explored a series of work on actually transferring behavior across different robot embodiments [89], [428]. The future of robotics is likely to be widely heterogenous, so learning behaviors that can transfer across robots and tasks is likely to be very important.

While the work we discussed in this thesis represents important steps towards real world deployment of agents, it is important to recognize that the agents are still largely being trained in a lab setting and not the truly “real" world, like someone’s home. The training setups that were actually used had to be somewhat carefully constructed and restricted, and these algorithms are unlikely to be directly applicable when a robot is deployed directly in the “wild". In order to actually scale up the current techniques to deal with the diversity and variability of the real world, a number of fundamentally new research directions would be exciting to explore going forward:

1. **Human-in-the-loop RL:** As agents get deployed around human centric environments, the ability to learn from, about and around humans will become increasingly important. Building teachable agents that infer explicit beliefs about human intent and explore safely around humans, even leveraging them actively to guide their own learning process will be very important. This area of research has a number of rich problems - how to build safely exploring RL agents, how to build RL agents that display a common sense understanding of what human supervisors are specifying, how to build communication interfaces between human supervisors and learning agents that enable the communication of maximal information with minimal human effort, just to name a few. The goal of building human in the loop RL systems is to construct systems that can naturally operate in the presence of human supervisors, leveraging them as tools and collaborators in the learning process rather than impediments.

2. **Continual, multi-task RL:** The unstructured real world is not cleanly divided into multiple different tasks experienced in isolation, but is instead a continuous stream of ever changing tasks performed sequentially. For instance a robot deployed in someone's living room may have to continually deal with new appliances and objects that are introduced into a living room, necessitating the learning of many different skills. These tasks may potentially be quite different from each other, even being drawn from different distributions, as opposed to the typical assumptions that most ML methods make that data is drawn IID from a fixed training distribution. For building flexible and scalable RL systems, it is important for us to start building agents that are robust to these types of distributional shifts, quickly and robustly adapting to new scenarios, collecting the appropriate data to do so. This may involve building on the framework of online learning, building algorithms that are both able to deal with catastrophic forgetting problems but more importantly becomes better and better at dealing with unknown scenarios as they collect more data. Specific problems that may be of interest in this direction would include building better calibrated models for uncertainty, developing RL algorithms that adapt their exploration space as they continue to explore new tasks, and developing RL algorithms that are explicitly trained to be robust or adaptive to distribution shift, rather than just to be good on the training MDP.

3. **Generalization in RL:** Lastly, much of the work in this thesis has dealt with a single object, scene or task. For truly harnessing the power of learning, we need to build algorithms that widely generalize across different scenarios. Studying how appropriate inductive biases, causal mechanisms and choice of representations affect generalization in RL will be crucial in getting general purpose robotic agents for the real world. First and foremost, this requires us to develop a proper formalism and definition for generalization in RL, answering the question of “what" an agent should be generalizing over. Given this formalism, a number of problems are open and fascinating to explore. For instance it would be very interesting to understand how function approximation and generalization plays a role in the optimization process of model based and off-policy RL algorithms, even without considering out-of-support generalization. Thinking about extrapolation further, it is also exciting to consider what the right medium is for human supervisors to provide inductive biases, and developing formalism

to solicit the right types of inductive bias from the algorithm designer. Another direction that is particularly exciting is understanding whether the broader class of robotic learning problems share common structure that can be encoded as inductive bias into these models, allowing for broader generalization just as convolutions did for image based learning.

While there are a number of really exciting future directions that we just described, there are also a number of important lessons that were learned over the course of the work in this Ph.D. Here are a select few:

1. There is a huge delta between systems that learn in simulation and systems that learn in the real world. When training at scale in the real world, any crutch that is leveraged while training in simulation becomes a limiting factor. Often our intuition on how to actually build these systems is not actually correct, and building robotic systems early and with future iterations in mind is important.

2. Algorithms benefit very significantly from the use of prior data, and this can often make the difference between a system being useful versus being impractically slow. Thinking about where this data actually comes from and how to enable data scaling is likely to be very important going forward.

3. Often the focus of the reinforcement learning community has been on sample efficiency and the so-called “small-data" regime. However, I believe that the really interesting questions in robotic learning actually lie beyond that frontier, when we operate in the medium to large data regime. The challenges of generalization, distributions and optimization become much more important in this regime, rather than a skewed focus on sample efficiency. In future work, I hope researchers consider this regime of robotic learning more deeply.

* * *

We hope the work presented in this thesis serves as a springboard for continuing work on building robotic learning systems with RL in the real world. Even if not the exact techniques being proposed, we hope the design principles outlined here have an impact on the construction of robotic learning systems in the future.

Bibliography

[1] D. J. Braun and M. Goldfarb, "A control approach for actuated dynamic walking in biped robots," *IEEE Trans. Robotics*, vol. 25, no. 6, pp. 1292–1303, 2009. DOI: 10.1109/TRO.2009.2028762. [Online]. Available: https://doi.org/10.1109/TRO.2009.2028762.

[2] N. J. Kong, G. Council, and A. M. Johnson, "Ilqr for piecewise-smooth hybrid dynamical systems," *CoRR*, vol. abs/2103.14584, 2021. arXiv: 2103.14584. [Online]. Available: https://arxiv.org/abs/2103.14584.

[3] H. Delavari, R. Ghaderi, N. A. Ranjbar, S. H. HosseinNia, and S. Momani, "Adaptive fractional PID controller for robot manipulator," *CoRR*, vol. abs/1206.2027, 2012. arXiv: 1206.2027. [Online]. Available: http://arxiv.org/abs/1206.2027.

[4] J. Scholz and M. Stilman, "Combining motion planning and optimization for flexible robot manipulation," in *10th IEEE-RAS International Conference on Humanoid Robots, Humanoids 2010, Nashville, TN, USA, December 6-8, 2010*, IEEE, 2010, pp. 80–85. DOI: 10.1109/ICHR.2010.5686849. [Online]. Available: https://doi.org/10.1109/ICHR.2010.5686849.

[5] M. Saha and P. Isto, "Motion planning for robotic manipulation of deformable linear objects," in *Proceedings of the 2006 IEEE International Conference on Robotics and Automation, ICRA 2006, May 15-19, 2006, Orlando, Florida, USA*, IEEE, 2006, pp. 2478–2484. DOI: 10.1109/ROBOT.2006.1642074. [Online]. Available: https://doi.org/10.1109/ROBOT.2006.1642074.

[6] J. Schulman, Y. Duan, J. Ho, A. X. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," *Int. J. Robotics Res.*, vol. 33, no. 9, pp. 1251–1270, 2014. DOI: 10.1177/0278364914528132. [Online]. Available: https://doi.org/10.1177/0278364914528132.

[7] A. Mori, K. Hiramatsu, F. Naya, and N. Osato, "A robot-controlling agent description with finite state machines," in *Distributed Autonomous Robotic Systems 3, Proceedings of the 4th International Symposium on Distributed Autonomous Robotic Systems, DARS 1998, Karslruhe, Germany, 1998*, T. C. Lueth, R. Dillmann, P. Dario, and H. Wörn, Eds., Springer, 1998, pp. 225–234. DOI: 10.1007/978-3-642-72198-4_22. [Online]. Available: https://doi.org/10.1007/978-3-642-72198-4_22.

[8] P. Allgeuer and S. Behnke, "Hierarchical and state-based architectures for robot behavior planning and control," *CoRR*, vol. abs/1809.11067, 2018. arXiv: `1809.11067`. [Online]. Available: `http://arxiv.org/abs/1809.11067`.

[9] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined Task and Motion Planning Through an Extensible Planner-Independent Interface Layer," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2014. [Online]. Available: `https://people.eecs.berkeley.edu/{~}russell/papers/icra14-planrob.pdf`.

[10] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. Berg, and F. Li, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision (IJCV)*, 2015.

[11] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2012, pp. 1097–1105.

[13] D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," in *International Conference on Learning Representations (ICLR)*, 2015. [Online]. Available: `https://arxiv.org/pdf/1409.0473.pdf`.

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention Is All You Need," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. [Online]. Available: `https://arxiv.org/pdf/1706.03762.pdf`.

[15] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization.," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: `http://dblp.uni-trier.de/db/journals/corr/corr1412.html#KingmaB14`.

[16] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," *Neural Networks*, vol. 113, pp. 54–71, 2019. DOI: `10.1016/j.neunet.2019.01.012`. [Online]. Available: `https://doi.org/10.1016/j.neunet.2019.01.012`.

[17] D. Rao, F. Visin, A. A. Rusu, R. Pascanu, Y. W. Teh, and R. Hadsell, "Continual unsupervised representation learning," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 7645–7655. [Online]. Available: `https://proceedings.neurips.cc/paper/2019/hash/861578d797aeb0634f77aff3f488cca2-Abstract.html`.

[18] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. arXiv: `1707.06347`.

[19] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3-4, pp. 229–256, May 1992, ISSN: 0885-6125. DOI: `10.1007/BF00992696`. [Online]. Available: `http://dx.doi.org/10.1007/BF00992696`.

[20] A. Rajeswaran, K. Lowrey, E. Todorov, and S. Kakade, "Towards Generalization and Simplicity in Continuous Control," in *NIPS*, 2017.

[21] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, 2015.

[22] H. Van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-learning," in *Association for the Advancement of Artificial Intelligence (AAAI)*, 2016, ISBN: 1509.06461v3. arXiv: `1509.06461v3`. [Online]. Available: `www.aaai.org`.

[23] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *arXiv preprint arXiv:1801.01290*, 2018.

[24] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 7559–7566.

[25] N. Heess, G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa, "Learning continuous control policies by stochastic value gradients," in *NIPS*, 2015.

[26] M. Deisenroth and C. E. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *Proceedings of the 28th International Conference on machine learning (ICML-11)*, 2011, pp. 465–472.

[27] R. Sutton and A. Barto, *Introduction to Reinforcement Learning*. MIT Press, 1998, vol. 1.

[28] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," in *NIPS Workshop on Deep Learning*, 2013, pp. 1–9, ISBN: 1476-4687 (Electronic) 0028-0836 (Linking). DOI: `10.1038/nature14236`. arXiv: `1312.5602`. [Online]. Available: `https://arxiv.org/pdf/1312.5602.pdfhttp://arxiv.org/abs/1312.5602https://www.cs.toronto.edu/{~}vmnih/docs/dqn.pdf`.

[29] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *CoRR*, 2016. arXiv: `1606.01540`.

[30] A. V. Nair, V. Pong, M. Dalal, S. Bahl, S. Lin, and S. Levine, "Visual reinforcement learning with imagined goals," in *Advances in Neural Information Processing Systems*, 2018, pp. 9191–9200.

[31] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[32] Y. Chebotar, K. Hausman, M. Zhang, G. S. Sukhatme, S. Schaal, and S. Levine, "Combining model-based and model-free updates for trajectory-centric reinforcement learning," *CoRR*, vol. abs/1703.03078, 2017. [Online]. Available: `http://arxiv.org/abs/1703.03078`.

[33] A. Nagabandi, K. Konoglie, S. Levine, and V. Kumar, "Deep Dynamics Models for Learning Dexterous Manipulation," in *Conference on Robot Learning (CoRL)*, 2019.

[34] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *CoRR*, vol. abs/1603.02199, 2016. arXiv: `1603.02199`. [Online]. Available: `http://arxiv.org/abs/1603.02199`.

[35] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, IEEE, 2017, pp. 3389–3396.

[36] M. Riedmiller, R. Hafner, T. Lampe, M. Neunert, J. Degrave, T. Wiele, V. Mnih, N. Heess, and J. T. Springenberg, "Learning by playing solving sparse reward tasks from scratch," in *International Conference on Machine Learning*, PMLR, 2018, pp. 4344–4353.

[37] D. Kalashnikov, J. Varley, Y. Chebotar, B. Swanson, R. Jonschkowski, C. Finn, S. Levine, and K. Hausman, "Mt-opt: Continuous multi-task robotic reinforcement learning at scale," *arXiv preprint arXiv:2104.08212*, 2021.

[38] G. Kahn, P. Abbeel, and S. Levine, "Badgr: An autonomous self-supervised learning-based navigation system," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1312–1319, 2021.

[39] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[40] H. Surmann, C. Jestel, R. Marchel, F. Musberg, H. Elhadj, and M. Ardani, "Deep reinforcement learning for real autonomous mobile robot navigation in indoor environments," *CoRR*, vol. abs/2005.13857, 2020. arXiv: `2005.13857`. [Online]. Available: `https://arxiv.org/abs/2005.13857`.

[41] M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, *et al.*, "Learning dexterous in-hand manipulation," *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, 2020.

[42] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," *ArXiv e-prints*, 2017. arXiv: `1703.06907`.

[43] F. Sadeghi and S. Levine, “Cad2rl: Real single-image flight without a single real image,” *arXiv preprint arXiv:1611.04201*, 2016.

[44] A. Kadian, J. Truong, A. Gokaslan, A. Clegg, E. Wijmans, S. Lee, M. Savva, S. Chernova, and D. Batra, “Are we making real progress in simulated environments? measuring the sim2real gap in embodied visual navigation,” *CoRR*, vol. abs/1912.06321, 2019. arXiv: 1912.06321. [Online]. Available: http://arxiv.org/abs/1912.06321.

[45] Y. Du, O. Watkins, T. Darrell, P. Abbeel, and D. Pathak, “Auto-tuned sim-to-real transfer,” *CoRR*, vol. abs/2104.07662, 2021. arXiv: 2104.07662. [Online]. Available: https://arxiv.org/abs/2104.07662.

[46] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, “Closing the Sim-to-Real Loop: Adapting Simulation Randomization with Real World Experience,” in *International Conference on Robotics and Automation (ICRA)*, 2019. arXiv: 1810.05687v4. [Online]. Available: https://arxiv.org/pdf/1810.05687.pdf.

[47] D. Shah, B. Eysenbach, G. Kahn, N. Rhinehart, and S. Levine, “Ving: Learning open-world navigation with visual goals,” *CoRR*, vol. abs/2012.09812, 2020. arXiv: 2012.09812. [Online]. Available: https://arxiv.org/abs/2012.09812.

[48] X. Lin, Y. Wang, J. Olkin, and D. Held, “Softgym: Benchmarking deep reinforcement learning for deformable object manipulation,” *CoRR*, vol. abs/2011.07215, 2020. arXiv: 2011.07215. [Online]. Available: https://arxiv.org/abs/2011.07215.

[49] T. D. Kulkarni, W. F. Whitney, P. Kohli, and J. B. Tenenbaum, “Deep convolutional inverse graphics network,” in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., 2015, pp. 2539–2547. [Online]. Available: https://proceedings.neurips.cc/paper/2015/hash/ced556cd9f9c0c8315cfbe0744a3baf0-Abstract.html.

[50] J. Tremblay, T. To, B. Sundaralingam, Y. Xiang, D. Fox, and S. Birchfield, “Deep object pose estimation for semantic robotic grasping of household objects,” in *2nd Annual Conference on Robot Learning, CoRL 2018, Zürich, Switzerland, 29-31 October 2018, Proceedings*, ser. Proceedings of Machine Learning Research, vol. 87, PMLR, 2018, pp. 306–316. [Online]. Available: http://proceedings.mlr.press/v87/tremblay18a.html.

[51] L. Manuelli, Y. Li, P. R. Florence, and R. Tedrake, “Keypoints into the future: Self-supervised correspondence in model-based reinforcement learning,” *CoRR*, vol. abs/2009.05085, 2020. arXiv: 2009.05085. [Online]. Available: https://arxiv.org/abs/2009.05085.

[52] T. Schmidt, R. A. Newcombe, and D. Fox, "Self-supervised visual descriptor learning for dense correspondence," *IEEE Robotics Autom. Lett.*, vol. 2, no. 2, pp. 420–427, 2017. DOI: `10.1109/LRA.2016.2634089`. [Online]. Available: `https://doi.org/10.1109/LRA.2016.2634089`.

[53] K. Mülling, J. Kober, O. Kroemer, and J. Peters, "Learning to select and generalize striking movements in robot table tennis," *Int. J. Robotics Res.*, vol. 32, no. 3, pp. 263–279, 2013. DOI: `10.1177/0278364912472380`. [Online]. Available: `https://doi.org/10.1177/0278364912472380`.

[54] A. Murali, A. Mousavian, C. Eppner, C. Paxton, and D. Fox, "6-dof grasping for target-driven object manipulation in clutter," in *2020 IEEE International Conference on Robotics and Automation, ICRA 2020, Paris, France, May 31 - August 31, 2020*, IEEE, 2020, pp. 6232–6238. DOI: `10.1109/ICRA40945.2020.9197318`. [Online]. Available: `https://doi.org/10.1109/ICRA40945.2020.9197318`.

[55] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J. Crespo, and D. Dennison, "Hidden technical debt in machine learning systems," in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., 2015, pp. 2503–2511. [Online]. Available: `https://proceedings.neurips.cc/paper/2015/hash/86df7dcfd896fcaf2674f757a2463eba-Abstract.html`.

[56] B. Goldberg, K. Goldberg, and A. Chase, "How to train your robot," 2019.

[57] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert, "Learning monocular reactive UAV control in cluttered natural environments," in *2013 IEEE International Conference on Robotics and Automation*, 2013. DOI: `10.1109/ICRA.2013.6630809`.

[58] D. Pomerleau, "ALVINN: an autonomous land vehicle in a neural network," in *NIPS*, 1988.

[59] A. Edwards, C. Isbell, and A. Takanishi, "Perceptual reward functions," *arXiv preprint arXiv: 1608.03824*, 2016.

[60] A. Y. Ng and S. J. Russell, "Algorithms for inverse reinforcement learning," in *Proceedings of the Seventeenth International Conference on Machine Learning*, ser. ICML '00, 2000, ISBN: 1-55860-707-2.

[61] B. C. Stadie, P. Abbeel, and I. Sutskever, "Third-person imitation learning," in *Proceedings of the International Conference on Learning Representations, ICLR 2017.*

[62] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Advances in Neural Information Processing Systems 29*, 2016.

[63] P. Sermanet, K. Xu, and S. Levine, "Unsupervised perceptual rewards for imitation learning," in *Robotics: Science and Systems (RSS) 2017.*

[64] S. Schaal, “Is imitation learning the route to humanoid robots?” *Trends in Cognitive Sciences*, vol. 3, no. 6, pp. 233–242, 1999.

[65] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, “A survey of robot learning from demonstration,” *Robot. Auton. Syst.*, vol. 57, no. 5, pp. 469–483, May 2009, ISSN: 0921-8890. DOI: `10.1016/j.robot.2008.10.024`.

[66] P. Abbeel, A. Coates, and A. Y. Ng, “Autonomous helicopter aerobatics through apprenticeship learning,” *I. J. Robotics Res.*, vol. 29, no. 13, pp. 1608–1639, 2010. DOI: `10.1177/0278364910371999`.

[67] J. Kober and J. Peters, “Learning motor primitives for robotics,” in *2009 IEEE International Conference on Robotics and Automation, ICRA 2009, Kobe, Japan, May 12-17, 2009*, IEEE, 2009, pp. 2112–2118. DOI: `10.1109/ROBOT.2009.5152577`. [Online]. Available: `https://doi.org/10.1109/ROBOT.2009.5152577`.

[68] A. Billard and M. J. Mataric, “Learning human arm movements by imitation: Evaluation of a biologically inspired connectionist architecture,” *Robotics and Autonomous Systems*, vol. 37, no. 2-3, pp. 145–160, 2001. DOI: `10.1016/S0921-8890(01)00155-5`.

[69] P. Abbeel and A. Y. Ng, “Apprenticeship learning via inverse reinforcement learning,” in *Proceedings of the Twenty-first International Conference on Machine Learning*, ser. ICML ’04, Banff, Alberta, Canada: ACM, 2004. DOI: `10.1145/1015330.1015430`.

[70] S. Levine, Z. Popovic, and V. Koltun, “Nonlinear inverse reinforcement learning with gaussian processes,” in *Advances in Neural Information Processing Systems 24*, 2011.

[71] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, “Maximum entropy inverse reinforcement learning.,” in *AAAI*, AAAI Press, 2008, ISBN: 978-1-57735-368-3.

[72] N. D. Ratliff, J. A. Bagnell, and M. Zinkevich, “Maximum margin planning,” in *Machine Learning, Proceedings of the Twenty-Third International Conference ICML*, 2006. DOI: `10.1145/1143844.1143936`.

[73] D. Ramachandran and E. Amir, “Bayesian inverse reinforcement learning,” in *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence*, 2007.

[74] C. Finn, S. Levine, and P. Abbeel, “Guided cost learning: Deep inverse optimal control via policy optimization,” in *Proceedings of the 33nd International Conference on Machine Learning, ICML*, 2016.

[75] M. Kalakrishnan, P. Pastor, L. Righetti, and S. Schaal, “Learning objective functions for manipulation,” in *2013 IEEE International Conference on Robotics and Automation*, 2013. DOI: `10.1109/ICRA.2013.6630743`.

[76] A. Boularias, J. Kober, and J. Peters, “Relative entropy inverse reinforcement learning,” in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS*, 2011.

[77] P. Sermanet, C. Lynch, Y. Chebotar, J. Hsu, E. Jang, S. Schaal, S. Levine, and G. Brain, "Time-Contrastive Networks: Self-Supervised Learning from Video," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 1134–1141, 2018, ISSN: 10504729. DOI: `10.1109/ICRA.2018.8462891`. arXiv: `1704.06888`.

[78] J. Lee and M. S. Ryoo, "Learning robot activities from first-person human videos using convolutional future regression," *arXiv preprint arXiv:1703.01040*, 2017.

[79] Y. Duan, M. Andrychowicz, B. C. Stadie, J. Ho, J. Schneider, I. Sutskever, P. Abbeel, and W. Zaremba, "One-shot imitation learning," in *NIPS*, 2017.

[80] J. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *ICCV*, 2017.

[81] K. Bousmalis, N. Silberman, D. Dohan, D. Erhan, and D. Krishnan, "Unsupervised pixel-level domain adaptation with generative adversarial networks," in *CVPR*, 2017.

[82] P. Isola, J. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," *2017 IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[83] L. A. Gatys, A. S. Ecker, and M. Bethge, "A neural algorithm of artistic style," *arXiv preprint arXiv:1508.06576*, 2015.

[84] Z. Xu and M. Cakmak, "Enhanced robotic cleaning with a low-cost tool attachment," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2014. DOI: `10.1109/IROS.2014.6942916`.

[85] C. Schenck and D. Fox, "Visual closed-loop control for pouring liquids," in *2017 IEEE International Conference on Robotics and Automation, ICRA*, IEEE, 2017. DOI: `10.1109/ICRA.2017.7989307`.

[86] I. Lenz, R. A. Knepper, and A. Saxena, "Deepmpc: Learning deep latent features for model predictive control," in *Robotics: Science and Systems XI*, 2015.

[87] J. Sung, S. H. Jin, I. Lenz, and A. Saxena, "Robobarista: Learning to manipulate novel objects via deep multimodal embedding," *In International Symposium on Robotics Research (ISRR), 2015*,

[88] A. Byravan and D. Fox, "Se3-nets: Learning rigid body motion using deep neural networks," in *ICRA*, 2017.

[89] A. Gupta, C. Devin, Y. Liu, P. Abbeel, and S. Levine, "Learning invariant feature spaces to transfer skills with reinforcement learning," in *Proceedings of the International Conference on Learning Representations, ICLR*, 2017.

[90] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust Region Policy Optimization," in *International Conference on Machine Learning (ICML)*, 2015, ISBN: 0375-9687. DOI: `10.1063/1.4927398`. arXiv: `1502.05477`. [Online]. Available: `https://arxiv.org/pdf/1502.05477.pdfhttp://arxiv.org/abs/1502.05477`.

[91] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *IROS*, 2012. DOI: `10.1109/IROS.2012.6386109`.

[92] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2016. DOI: `10.1109/CVPR.2016.308`.

[93] Y. Li, J. Song, and S. Ermon, "Inferring the latent structure of human decision-making from raw visual inputs," *arXiv preprint arXiv:1703.08840*, 2017.

[94] E. Tzeng, C. Devin, J. Hoffman, C. Finn, P. Abbeel, S. Levine, K. Saenko, and T. Darrell, "Adapting deep visuomotor representations with weak pairwise constraints," in *Workshop on Algorithmic Robotics*, 2016.

[95] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine, *Learning complex dexterous manipulation with deep reinforcement learning and demonstrations*, 2018. arXiv: `1709.10087 [cs.LG]`.

[96] A. Y. Ng, D. Harada, and S. J. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *Proceedings of the Sixteenth International Conference on Machine Learning (ICML 1999), Bled, Slovenia, June 27 - 30, 1999*, I. Bratko and S. Dzeroski, Eds., Morgan Kaufmann, 1999, pp. 278–287.

[97] J. Fu, A. Singh, D. Ghosh, L. Yang, and S. Levine, "Variational inverse control with events: A general framework for data-driven reward definition," in *Advances in Neural Information Processing Systems*, 2018, pp. 8547–8556.

[98] H. Zhu, J. Yu, A. Gupta, D. Shah, K. Hartikainen, A. Singh, V. Kumar, and S. Levine, "The ingredients of real-world robotic reinforcement learning," *arXiv preprint arXiv:2004.12570*, 2020.

[99] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Nets," in *Advances in Neural Information Processing Systems (NIPS)*, 2014. [Online]. Available: `https://arxiv.org/pdf/1406.2661.pdf`.

[100] J. Fu, K. Luo, and S. Levine, "Learning robust rewards with adverserial inverse reinforcement learning," in *International Conference on Learning Representations*, 2018.

[101] D. Misra, M. Henaff, A. Krishnamurthy, and J. Langford, "Kinematic state abstraction and provably efficient rich-observation reinforcement learning," *CoRR*, vol. abs/1911.05815, 2019. arXiv: `1911.05815`. [Online]. Available: `http://arxiv.org/abs/1911.05815`.

[102] M. Wiering and J. Schmidhuber, "Efficient model-based exploration," in *Proceedings of the Sixth International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, MIT Press Cambridge, MA, 1998.

[103] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, 2002.

[104] T. Schaul, Y. Sun, D. Wierstra, F. Gomez, and J. Schmidhuber, "Curiosity-driven optimization," in *2011 IEEE Congress of Evolutionary Computation (CEC)*, 2011.

[105] R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel, "Variational Information Maximizing Exploration," in *Advances in Neural Information Processing Systems (NIPS)*, 2016. arXiv: `1605.09674`.

[106] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, pp. 16–17.

[107] H. Tang, R. Houthooft, D. Foote, A. Stooke, O. X. Chen, Y. Duan, J. Schulman, F. DeTurck, and P. Abbeel, "# exploration: A study of count-based exploration for deep reinforcement learning," in *Advances in neural information processing systems*, 2017, pp. 2753–2762.

[108] B. C. Stadie, S. Levine, and P. Abbeel, "Incentivizing exploration in reinforcement learning with deep predictive models," *arXiv preprint arXiv:1507.00814*, 2015.

[109] M. G. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying count-based exploration and intrinsic motivation," *CoRR*, vol. abs/1606.01868, 2016. arXiv: `1606.01868`. [Online]. Available: `http://arxiv.org/abs/1606.01868`.

[110] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by random network distillation," in *International Conference on Learning Representations*, 2018.

[111] B. O'Donoghue, "Variational bayesian reinforcement learning with regret bounds," *arXiv preprint arXiv:1807.09647*, 2018.

[112] M. Strens, "A bayesian framework for reinforcement learning," in *ICML*, 2000.

[113] I. Osband, D. Russo, and B. Van Roy, "(more) efficient reinforcement learning via posterior sampling," in *Advances in Neural Information Processing Systems*, 2013, pp. 3003–3011.

[114] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep exploration via bootstrapped dqn," in *Advances in neural information processing systems*, 2016.

[115] W. B. Knox and P. Stone, "Interactively shaping agents via human reinforcement: The tamer framework," in *Proceedings of the fifth international conference on Knowledge capture*, 2009, pp. 9–16.

[116] A. Singh, L. Yang, C. Finn, and S. Levine, "End-to-end robotic reinforcement learning without reward engineering," in *Robotics: Science and Systems XV, University of Freiburg, Freiburg im Breisgau, Germany, June 22-26, 2019*, A. Bicchi, H. Kress-Gazit, and S. Hutchinson, Eds., 2019.

[117] L. P. Kaelbling, "Learning to achieve goals," in *IJCAI*, 1993.

[118] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal value function approximators," in *International conference on machine learning*, 2015, pp. 1312–1320.

[119] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *Advances in neural information processing systems*, 2017.

[120] V. Veeriah, J. Oh, and S. Singh, "Many-goals reinforcement learning," *arXiv preprint arXiv:1806.09605*, 2018.

[121] P. Rauber, A. Ummadisingu, F. Mutz, and J. Schmidhuber, "Hindsight policy gradients," in *International Conference on Learning Representations*, 2018.

[122] D. Warde-Farley, T. Van de Wiele, T. Kulkarni, C. Ionescu, S. Hansen, and V. Mnih, "Unsupervised control through non-parametric discriminative rewards," in *International Conference on Learning Representations*, 2018.

[123] C. Colas, P. Fournier, M. Chetouani, O. Sigaud, and P.-Y. Oudeyer, "Curious: Intrinsically motivated modular multi-goal reinforcement learning," in *International conference on machine learning*, 2019, pp. 1331–1340.

[124] D. Ghosh, A. Gupta, and S. Levine, "Learning actionable representations with goal conditioned policies," in *International Conference on Learning Representations*, 2019.

[125] V. H. Pong, M. Dalal, S. Lin, A. Nair, S. Bahl, and S. Levine, "Skew-fit: State-covering self-supervised reinforcement learning," in *International Conference on Machine Learning (ICML)*, 2020.

[126] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley, "Stochastic variational inference," *The Journal of Machine Learning Research*, vol. 14, no. 1, pp. 1303–1347, 2013.

[127] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural network," in *International Conference on Machine Learning*, 2015, pp. 1613–1622.

[128] W. J. Maddox, P. Izmailov, T. Garipov, D. P. Vetrov, and A. G. Wilson, "A simple baseline for bayesian uncertainty in deep learning," in *Advances in Neural Information Processing Systems*, 2019, pp. 13 153–13 164.

[129] J. J. Rissanen, "Fisher information and stochastic complexity," *IEEE transactions on information theory*, vol. 42, no. 1, pp. 40–47, 1996.

[130] Y. M. Shtar'kov, "Universal sequential coding of single messages," *Problemy Peredachi Informatsii*, vol. 23, no. 3, pp. 3–17, 1987.

[131] P. D. Grünwald, *The minimum description length principle*. MIT press, 2007.

[132] Y. Fogel and M. Feder, "Universal batch learning with log-loss," in *2018 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2018, pp. 21–25.

[133] A. Zhou and S. Levine, "Amortized conditional normalized maximum likelihood," *CoRR*, vol. abs/2011.02696, 2020. arXiv: `2011.02696`.

[134] S. Levine, "Reinforcement learning and control as probabilistic inference: Tutorial and review," *CoRR*, vol. abs/1805.00909, 2018. arXiv: `1805.00909`.

[135] J. Rissanen and T. Roos, "Conditional NML universal models," in *2007 Information Theory and Applications Workshop*, 2007, pp. 337–341.

[136] K. Bibas, Y. Fogel, and M. Feder, "Deep pnml: Predictive normalized maximum likelihood for deep neural networks," *CoRR*, vol. abs/1904.12286, 2019. arXiv: `1904.12286`.

[137] M. Arjovsky and L. Bottou, "Towards principled methods for training generative adversarial networks," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, OpenReview.net, 2017.

[138] J. Zhang, "Model selection with informative normalized maximum likelihood: Data prior and model prior," 2011.

[139] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine, *Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning*, 2019. arXiv: `1910.10897 [cs.LG]`.

[140] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International Conference on Machine Learning (ICML)*, 2017.

[141] K. Hartikainen, X. Geng, T. Haarnoja, and S. Levine, "Dynamical distance learning for unsupervised and semi-supervised skill discovery," *CoRR*, vol. abs/1907.08225, 2019. arXiv: `1907.08225`.

[142] G. Vezzani, A. Gupta, L. Natale, and P. Abbeel, "Learning latent state representation for speeding up exploration," *arXiv preprint arXiv:1905.12621*, 2019.

[143] S. Branavan, H. Chen, L. Zettlemoyer, and R. Barzilay, "Reinforcement learning for mapping instructions to actions," in *ACL*, 2009.

[144] D. Chen and R. Mooney, "Learning to interpret natural language navigation instructions from observations," in *AAAI*, 2011.

[145] A. L. Thomaz, G. Hoffman, and C. Breazeal, "Reinforcement learning with human teachers: Understanding how people want to teach robots," in *RO-MAN*, 2006.

[146] Y. Liu, A. Gupta, P. Abbeel, and S. Levine, "Imitation from observation: Learning to imitate behaviors from raw video via context translation," *CoRR*, vol. abs/1707.03374, 2017. arXiv: `1707.03374`.

[147] P. F. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," in *NIPS*, 2017.

[148] G. Warnell, N. R. Waytowich, V. Lawhern, and P. Stone, "Deep TAMER: interactive agent shaping in high-dimensional state spaces," *CoRR*, vol. abs/1709.10163, 2017. arXiv: `1709.10163`.

[149] M. MacMahon, B. Stankiewicz, and B. Kuipers, "Walk the talk: Connecting language, knowledge, and action in route instructions," in *AAAI*, 2006.

[150] A. Vogel and D. Jurafsky, "Learning to follow navigational directions," in *ACL*, 2010.

[151] S. Tellex, T. Kollar, S. Dickerson, M. Walter, A. Banerjee, S. Teller, and N. Roy, "Understanding natural language commands for robotic navigation and mobile manipulation," in *AAAI*, 2011.

[152] Y. Artzi and L. Zettlemoyer, "Weakly supervised learning of semantic parsers for mapping instructions to actions," in *TACL*, 2013.

[153] J. Kim and R. Mooney, “Adapting discriminative reranking to grounded language learning,” in *ACL*, 2013.

[154] J. Andreas and D. Klein, “Alignment-based compositional semantics for instruction following,” in *EMNLP*, 2015.

[155] D. Misra, J. Langford, and Y. Artzi, “Mapping instructions and visual observations to actions with reinforcement learning,” in *EMNLP*, 2017.

[156] J. Andreas, D. Klein, and S. Levine, “Learning with latent language,” in *NAACL*, 2018.

[157] J. Oh, S. P. Singh, H. Lee, and P. Kohli, “Zero-shot task generalization with multi-task deep reinforcement learning,” *CoRR*, vol. abs/1706.05064, 2017. arXiv: `1706.05064`.

[158] M. Janner, K. Narasimhan, and R. Barzilay, “Representation learning for grounded spatial reasoning,” in *ACL*, 2018.

[159] R. Akrour, M. Schoenauer, and M. Sebag, “Preference-based policy learning,” in *ECML/PKDD*, 2011.

[160] P. Pilarski, M. Dawson, T. Degris, F. Fahimi, J. Carey, and R. Sutton, “Online human training of a myoelectric prosthesis controller via actor-critic reinforcement learning,” in *IEEE ICORR*, 2011.

[161] R. Akrour, M. Schoenauer, and M. Sebag, “APRIL: active preference-learning based reinforcement learning,” *CoRR*, vol. abs/1208.0984, 2012. arXiv: `1208.0984`.

[162] L. El Asri, J. He, and K. Suleman, “A sequence-to-sequence model for user simulation in spoken dialogue systems,” in *arXiv preprint arXiv:1607.00070*, 2016.

[163] J. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick, “Learning to reinforcement learn,” *arXiv preprint arXiv:1611.05763*, 2016.

[164] S. Reddy, S. Levine, and A. D. Dragan, “Shared autonomy via deep reinforcement learning,” *CoRR*, vol. abs/1802.01744, 2018. arXiv: `1802.01744`. [Online]. Available: `http://arxiv.org/abs/1802.01744`.

[165] J. Schmidhuber, “Evolutionary principles in self-referential learning,” *Diploma thesis, Institut f. Informatik, Tech. Univ. Munich*, 1987.

[166] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, “Rl^2: Fast reinforcement learning via slow reinforcement learning,” *CoRR*, vol. abs/1611.02779, 2016. arXiv: `1611.02779`.

[167] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel, “Meta-learning with temporal convolutions,” *CoRR*, vol. abs/1707.03141, 2017. arXiv: `1707.03141`.

[168] A. Nagabandi, I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn, “Learning to adapt in dynamic, real-world environments through meta-reinforcement learning,” *arXiv preprint arXiv:1803.11347*, 2018.

[169] J. Snell, K. Swersky, and R. S. Zemel, "Prototypical networks for few-shot learning," *CoRR*, vol. abs/1703.05175, 2017. arXiv: `1703.05175`.

[170] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "Meta-learning with memory-augmented neural networks," in *International Conference on Machine Learning (ICML)*, 2016.

[171] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra, "Matching networks for one shot learning," in *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[172] F. Sung, L. Zhang, T. Xiang, T. M. Hospedales, and Y. Yang, "Learning to learn: Meta-critic networks for sample efficient learning," *CoRR*, vol. abs/1706.09529, 2017. arXiv: `1706.09529`.

[173] K. Xu, E. Ratner, A. D. Dragan, S. Levine, and C. Finn, "Learning a prior over intent via meta-inverse reinforcement learning," *CoRR*, vol. abs/1805.12573, 2018.

[174] A. Xie, A. Singh, S. Levine, and C. Finn, "Few-shot goal inference for visuomotor learning and planning," in *CoRL*, 2018.

[175] S. Ross, G. J. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, 2011, pp. 627–635. [Online]. Available: `http://www.jmlr.org/proceedings/papers/v15/ross11a/ross11a.pdf`.

[176] M. Chevalier-Boisvert and L. Willems, *Minimalistic gridworld environment for openai gym*, `https://github.com/maximecb/gym-minigrid`, 2018.

[177] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[178] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[179] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, *et al.*, "Learning to navigate in complex environments," *arXiv preprint arXiv:1611.03673*, 2016.

[180] D. Hadfield-Menell, S. Milli, P. Abbeel, S. J. Russell, and A. Dragan, "Inverse reward design," in *Advances in Neural Information Processing Systems*, 2017, pp. 6768–6777.

[181] P. Bacon, J. Harb, and D. Precup, "The option-critic architecture," *CoRR*, vol. abs/1609.05140, 2016. arXiv: `1609.05140`. [Online]. Available: `http://arxiv.org/abs/1609.05140`.

[182] N. Heess, G. Wayne, Y. Tassa, T. Lillicrap, M. Riedmiller, and D. Silver, "Learning and transfer of modulated locomotor controllers," *arXiv preprint arXiv:1610.05182*, 2016.

[183] P. Dayan and G. E. Hinton, "Feudal reinforcement learning," in *Advances in neural information processing systems*, 1993, pp. 271–278.

[184] K. Frans, J. Ho, X. Chen, P. Abbeel, and J. Schulman, "Meta learning shared hierarchies," *arXiv preprint arXiv:1710.09767*, 2017.

[185] S. Krishnan, R. Fox, I. Stoica, and K. Goldberg, "Ddco: Discovery of deep continuous options for robot learning from demonstrations," in *Conference on Robot Learning*, 2017, pp. 418–437.

[186] C. Florensa, Y. Duan, and P. Abbeel, "Stochastic neural networks for hierarchical reinforcement learning," *arXiv preprint arXiv:1704.03012*, 2017.

[187] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," *arXiv preprint arXiv:1701.06538*, 2017.

[188] J. Schulman, P. Abbeel, and X. Chen, "Equivalence between policy gradients and soft q-learning," *arXiv preprint arXiv:1704.06440*, 2017.

[189] O. Nachum, M. Norouzi, K. Xu, and D. Schuurmans, "Bridging the gap between value and policy based reinforcement learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 2772–2782.

[190] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement learning with deep energy-based policies," *arXiv preprint arXiv:1702.08165*, 2017.

[191] S. Mohamed and D. J. Rezende, "Variational information maximisation for intrinsically motivated reinforcement learning," in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., 2015, pp. 2125–2133. [Online]. Available: `https://proceedings.neurips.cc/paper/2015/hash/e00406144c1e7e35240afed70f34166a-Abstract.html`.

[192] T. Jung, D. Polani, and P. Stone, "Empowerment for continuous agent—environment systems," *Adaptive Behavior*, vol. 19, no. 1, pp. 16–39, 2011.

[193] K. Hausman, J. T. Springenberg, Z. Wang, N. Heess, and M. Riedmiller, "Learning an embedding space for transferable robot skills," *International Conference on Learning Representations*, 2018. [Online]. Available: `https://openreview.net/forum?id=rk07ZXZRb`.

[194] K. Gregor, D. J. Rezende, and D. Wierstra, "Variational intrinsic control," *arXiv preprint arXiv:1611.07507*, 2016.

[195] J. Lehman and K. O. Stanley, "Abandoning objectives: Evolution through the search for novelty alone," *Evolutionary computation*, vol. 19, no. 2, pp. 189–223, 2011.

[196] ——, "Evolving a diversity of virtual creatures through novelty search and local competition," 2011.

[197] B. G. Woolley and K. O. Stanley, "On the deleterious effects of a priori objectives on evolution and representation," in *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, ACM, 2011, pp. 957–964.

[198] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[199] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, "Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning," *arXiv preprint arXiv:1712.06567*, 2017.

[200] J. K. Pugh, L. B. Soros, and K. O. Stanley, "Quality diversity: A new frontier for evolutionary computation," *Frontiers in Robotics and AI*, vol. 3, p. 40, 2016.

[201] J.-B. Mouret and S. Doncieux, "Overcoming the bootstrap problem in evolutionary robotics using behavioral diversity," in *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, IEEE, 2009, pp. 1161–1168.

[202] R. M. Ryan and E. L. Deci, "Intrinsic and extrinsic motivations: Classic definitions and new directions," *Contemporary educational psychology*, vol. 25, no. 1, pp. 54–67, 2000.

[203] J. Fu, J. Co-Reyes, and S. Levine, "Ex2: Exploration with exemplar models for deep reinforcement learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 2574–2584.

[204] J. Schmidhuber, "Formal theory of creativity, fun, and intrinsic motivation," *IEEE Transactions on Autonomous Mental Development*, vol. 2, no. 3, pp. 230–247, 2010.

[205] P.-Y. Oudeyer, F. Kaplan, and V. V. Hafner, "Intrinsic motivation systems for autonomous mental development," *IEEE transactions on evolutionary computation*, vol. 11, no. 2, pp. 265–286, 2007.

[206] A. Baranes and P.-Y. Oudeyer, "Active learning of inverse models with intrinsically motivated goal exploration in robots," *Robotics and Autonomous Systems*, vol. 61, no. 1, pp. 49–73, 2013.

[207] J. Achiam, H. Edwards, D. Amodei, and P. Abbeel, "Variational autoencoding learning of options by reinforcement," *NIPS Deep Reinforcement Learning Symposium*, 2017.

[208] D. B. F. Agakov, "The im algorithm: A variational approach to information maximization," *Advances in Neural Information Processing Systems*, vol. 16, p. 201, 2004.

[209] S. Sukhbaatar, Z. Lin, I. Kostrikov, G. Synnaeve, A. Szlam, and R. Fergus, "Intrinsic motivation and automatic curricula via asymmetric self-play," *arXiv preprint arXiv:1703.05407*, 2017.

[210] R. K. Merton, "The matthew effect in science: The reward and communication systems of science are considered," *Science*, vol. 159, no. 3810, pp. 56–63, 1968.

[211] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer-Verlag New York, 2016.

[212] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel, “A simple neural attentive meta-learner,” in *NIPS 2017 Workshop on Meta-Learning*, 2017.

[213] A. Gupta, R. Mendonca, Y. Liu, P. Abbeel, and S. Levine, “Meta-reinforcement learning of structured exploration strategies,” *arXiv preprint arXiv:1802.07245*, 2018.

[214] M. Al-Shedivat, T. Bansal, Y. Burda, I. Sutskever, I. Mordatch, and P. Abbeel, “Continuous adaptation via meta-learning in nonstationary and competitive environments,” *arXiv preprint arXiv:1710.03641*, 2017.

[215] P. Krähenbühl, C. Doersch, J. Donahue, and T. Darrell, “Data-dependent initializations of convolutional neural networks,” *arXiv preprint arXiv:1511.06856*, 2015.

[216] K. Hsu, S. Levine, and C. Finn, “Unsupervised learning via meta-learning,” in *International Conference on Learning Representations (ICLR)*, 2019.

[217] D. K. Naik and R. Mammone, “Meta-neural networks that learn by learning,” in *International Joint Conference on Neural Netowrks (IJCNN)*, 1992.

[218] S. Thrun and L. Pratt, *Learning to Learn*. Springer Science & Business Media, 1998.

[219] S. Bengio, Y. Bengio, J. Cloutier, and J. Gecsei, “On the optimization of a synaptic learning rule,” in *Optimality in Artificial and Biological Neural Networks*, 1992.

[220] S. Hochreiter, A. Younger, and P. Conwell, “Learning to learn using gradient descent,” in *International Conference on Artificial Neural Networks (ICANN)*, 2001.

[221] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, and N. de Freitas, “Learning to learn by gradient descent by gradient descent,” in *Neural Information Processing Systems (NIPS)*, 2016.

[222] S. Ravi and H. Larochelle, “Optimization as a model for few-shot learning,” in *International Conference on Learning Representations (ICLR)*, 2017.

[223] T. Munkhdalai and H. Yu, “Meta networks,” *International Conference on Machine Learning (ICML)*, 2017.

[224] R. Mendonca, A. Gupta, R. Kralev, P. Abbeel, S. Levine, and C. Finn, “Guided meta-policy search,” *CoRR*, vol. abs/1904.00956, 2019.

[225] R. Houthooft, R. Y. Chen, P. Isola, B. C. Stadie, F. Wolski, J. Ho, and P. Abbeel, “Evolved policy gradients,” *arXiv preprint arXiv:1802.04821*, 2018.

[226] B. C. Stadie, G. Yang, R. Houthooft, X. Chen, Y. Duan, Y. Wu, P. Abbeel, and I. Sutskever, “Some considerations on learning to explore via meta-reinforcement learning,” *CoRR*, vol. abs/1803.01118, 2018. arXiv: `1803.01118`. [Online]. Available: `http://arxiv.org/abs/1803.01118`.

[227] K. Rakelly, A. Zhou, D. Quillen, C. Finn, and S. Levine, “Efficient off-policy meta-reinforcement learning via probabilistic context variables,” *arXiv preprint arXiv:1903.08254*, 2019.

[228] A. Antoniou and A. Storkey, “Assume, augment and learn: Unsupervised few-shot meta-learning via random labels and data augmentation,” *arXiv preprint arXiv:1902.09884*, 2019.

[229] J. Lin, Y. Wang, Y. Xia, T. He, and Z. Chen, “Learning to transfer: Unsupervised meta domain translation,” *arXiv preprint arXiv:1906.00181*, 2019.

[230] Z. Ji, X. Zou, T. Huang, and S. Wu, “Unsupervised few-shot learning via self-supervised training,” *arXiv preprint arXiv:1912.12178*, 2019.

[231] D. Held, X. Geng, C. Florensa, and P. Abbeel, “Automatic goal generation for reinforcement learning agents,” *arXiv preprint arXiv:1705.06366*, 2017.

[232] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine, “Diversity is all you need: Learning skills without a reward function,” *arXiv preprint arXiv:1802.06070*, 2018.

[233] J. D. Co-Reyes, Y. Liu, A. Gupta, B. Eysenbach, P. Abbeel, and S. Levine, “Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings,” *arXiv preprint arXiv:1806.02813*, 2018.

[234] A. Jabri, K. Hsu, A. Gupta, B. Eysenbach, S. Levine, and C. Finn, “Unsupervised curricula for visual meta-reinforcement learning,” in *Advances in Neural Information Processing Systems*, 2019, pp. 10 519–10 530.

[235] J. Schmidhuber, “Driven by compression progress: A simple principle explains essential aspects of subjective beauty, novelty, surprise, interestingness, attention, curiosity, creativity, art, science, music, jokes,” in *Computational Creativity: An Interdisciplinary Approach, 12.07. - 17.07.2009*, 2009. [Online]. Available: `http://drops.dagstuhl.de/opus/volltexte/2009/2197/`.

[236] K. Chua, R. Calandra, R. McAllister, and S. Levine, “Deep reinforcement learning in a handful of trials using probabilistic dynamics models,” in *Advances in Neural Information Processing Systems*, 2018, pp. 4754–4765.

[237] A. Srinivas, A. Jabri, P. Abbeel, S. Levine, and C. Finn, “Universal planning networks,” *arXiv preprint arXiv:1804.00645*, 2018.

[238] C. Finn and S. Levine, “Deep visual foresight for planning robot motion,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2017, pp. 2786–2793.

[239] C. G. Atkeson and J. C. Santamaria, “A comparison of direct and model-based reinforcement learning,” in *Proceedings of International Conference on Robotics and Automation*, IEEE, vol. 4, 1997, pp. 3557–3564.

[240] V. Pong, S. Gu, M. Dalal, and S. Levine, “Temporal difference models: Model-free deep rl for model-based control,” *arXiv preprint arXiv:1802.09081*, 2018.

[241] D. H. Wolpert, W. G. Macready, *et al.*, “No free lunch theorems for search,” Technical Report SFI-TR-95-02-010, Santa Fe Institute, Tech. Rep., 1995.

[242] D. Whitley and J. P. Watson, *Complexity theory and the no free lunch theorem*, 2005.

[243] L. Lee, B. Eysenbach, E. Parisotto, E. P. Xing, S. Levine, and R. Salakhutdinov, "Efficient exploration via state marginal matching," *CoRR*, vol. abs/1906.05274, 2019. arXiv: `1906.05274`. [Online]. Available: `http://arxiv.org/abs/1906.05274`.

[244] J. Rothfuss, D. Lee, I. Clavera, T. Asfour, and P. Abbeel, "Promp: Proximal meta-policy search," in *International Conference on Learning Representations, ICLR*, 2019.

[245] A. Sharma, S. Gu, S. Levine, V. Kumar, and K. Hausman, "Dynamics-aware unsupervised discovery of skills," *arXiv preprint arXiv:1907.01657*, 2019.

[246] C. Finn and S. Levine, "Meta-learning and universality: Deep representations and gradient descent can approximate any learning algorithm," *CoRR*, vol. abs/1710.11622, 2017. arXiv: `1710.11622`. [Online]. Available: `http://arxiv.org/abs/1710.11622`.

[247] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *International Conference on Machine Learning*, 2016, pp. 1329–1338.

[248] T. Zhang, Z. McCarthy, O. Jow, D. Lee, X. Chen, K. Goldberg, and P. Abbeel, "Deep imitation learning for complex manipulation tasks from virtual reality teleoperation," in *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*, IEEE, 2018, pp. 1–8. DOI: `10.1109/ICRA.2018.8461249`. [Online]. Available: `https://doi.org/10.1109/ICRA.2018.8461249`.

[249] C. Lynch, M. Khansari, T. Xiao, V. Kumar, J. Tompson, S. Levine, and P. Sermanet, "Learning latent plans from play," *CoRR*, vol. abs/1903.01973, 2019. arXiv: `1903.01973`.

[250] B. Akgün, M. Cakmak, J. W. Yoo, and A. L. Thomaz, "Trajectories and keyframes for kinesthetic teaching: A human-robot interaction perspective," in *International Conference on Human-Robot Interaction, HRI'12, Boston, MA, USA - March 05 - 08, 2012*, H. A. Yanco, A. Steinfeld, V. Evers, and O. C. Jenkins, Eds., ACM, 2012, pp. 391–398. DOI: `10.1145/2157689.2157815`. [Online]. Available: `https://doi.org/10.1145/2157689.2157815`.

[251] S. P. Singh, R. L. Lewis, A. G. Barto, and J. Sorg, "Intrinsically motivated reinforcement learning: An evolutionary perspective," *IEEE Trans. Auton. Ment. Dev.*, vol. 2, no. 2, pp. 70–82, 2010. DOI: `10.1109/TAMD.2010.2051031`. [Online]. Available: `https://doi.org/10.1109/TAMD.2010.2051031`.

[252] K. Baumli, D. Warde-Farley, S. Hansen, and V. Mnih, "Relative variational intrinsic control," in *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, AAAI Press, 2021, pp. 6732–6740. [Online]. Available: `https://ojs.aaai.org/index.php/AAAI/article/view/16832`.

[253] R. Zhao, K. Lu, P. Abbeel, and S. Tiomkin, "Efficient empowerment estimation for unsupervised stabilization," in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, OpenReview.net, 2021. [Online]. Available: `https://openreview.net/forum?id=u2YNJPcQlwq`.

[254] A. S. Klyubin, D. Polani, and C. L. Nehaniv, "Empowerment: A universal agent-centric measure of control," in *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2005, 2-4 September 2005, Edinburgh, UK*, IEEE, 2005, pp. 128–135. DOI: `10.1109/CEC.2005.1554676`. [Online]. Available: `https://doi.org/10.1109/CEC.2005.1554676`.

[255] J. Achiam, H. Edwards, D. Amodei, and P. Abbeel, "Variational option discovery algorithms," *arXiv preprint arXiv:1807.10299*, 2018.

[256] F. Torabi, G. Warnell, and P. Stone, "Recent advances in imitation learning from observation," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, S. Kraus, Ed., ijcai.org, 2019, pp. 6325–6331. DOI: `10.24963/ijcai.2019/882`. [Online]. Available: `https://doi.org/10.24963/ijcai.2019/882`.

[257] L. Smith, N. Dhawan, M. Zhang, P. Abbeel, and S. Levine, "AVID: learning multi-stage tasks via pixel-level translation of human videos," *CoRR*, vol. abs/1912.04443, 2019. arXiv: `1912.04443`. [Online]. Available: `http://arxiv.org/abs/1912.04443`.

[258] F. Torabi, G. Warnell, and P. Stone, "Generative adversarial imitation from observation," *CoRR*, vol. abs/1807.06158, 2018. arXiv: `1807.06158`. [Online]. Available: `http://arxiv.org/abs/1807.06158`.

[259] A. D. Edwards, H. Sahni, R. Liu, J. Hung, A. Jain, R. Wang, A. Ecoffet, T. Miconi, C. Isbell, and J. Yosinski, "Estimating q(s,s') with deep deterministic dynamics gradients," in *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, ser. Proceedings of Machine Learning Research, vol. 119, PMLR, 2020, pp. 2825–2835. [Online]. Available: `http://proceedings.mlr.press/v119/edwards20a.html`.

[260] W. Sun, A. Vemula, B. Boots, and D. Bagnell, "Provably efficient imitation learning from observation alone," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, K. Chaudhuri and R. Salakhutdinov, Eds., ser. Proceedings of Machine Learning Research, vol. 97, PMLR, 2019, pp. 6036–6045. [Online]. Available: `http://proceedings.mlr.press/v97/sun19b.html`.

[261] I. Radosavovic, X. Wang, L. Pinto, and J. Malik, "State-only imitation learning for dexterous manipulation," *CoRR*, vol. abs/2004.04650, 2020. arXiv: `2004.04650`. [Online]. Available: `https://arxiv.org/abs/2004.04650`.

[262] X. Pan, T. Zhang, B. Ichter, A. Faust, J. Tan, and S. Ha, "Zero-shot imitation learning from demonstrations for legged robot visual navigation," in *2020 IEEE International Conference on Robotics and Automation, ICRA 2020, Paris, France, May 31 - August 31, 2020*, IEEE, 2020, pp. 679–685. DOI: `10.1109/ICRA40945.2020.9196602`. [Online]. Available: `https://doi.org/10.1109/ICRA40945.2020.9196602`.

[263] A. Akakzia, C. Colas, P. Oudeyer, M. Chetouani, and O. Sigaud, "Grounding language to autonomously-acquired skills via goal generation," in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, OpenReview.net, 2021. [Online]. Available: `https://openreview.net/forum?id=chPj\_I5KMHG`.

[264] C. Colas, A. Akakzia, P. Oudeyer, M. Chetouani, and O. Sigaud, "Language-conditioned goal generation: A new approach to language grounding for RL," *CoRR*, vol. abs/2006.07043, 2020. arXiv: `2006.07043`. [Online]. Available: `https://arxiv.org/abs/2006.07043`.

[265] Y. Jiang, S. Gu, K. Murphy, and C. Finn, "Language as an abstraction for hierarchical deep reinforcement learning," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 9414–9426. [Online]. Available: `https://proceedings.neurips.cc/paper/2019/hash/0af787945872196b42c9f73ead2565c8-Abstract.html`.

[266] J. Thomason, M. Murray, M. Cakmak, and L. Zettlemoyer, "Vision-and-dialog navigation," in *Conference on Robot Learning (CoRL)*, 2019. [Online]. Available: `https://arxiv.org/abs/1907.04957`.

[267] J. Thomason, A. Padmakumar, J. Sinapov, N. Walker, Y. Jiang, H. Yedidsion, J. Hart, P. Stone, and R. J. Mooney, "Improving grounded natural language understanding through human-robot dialog," in *International Conference on Robotics and Automation (ICRA)*, 2019. [Online]. Available: `https://arxiv.org/abs/1903.00122`.

[268] M. Woodward, C. Finn, and K. Hausman, "Learning to interactively learn and assist," in *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, AAAI Press, 2020, pp. 2535–2543. [Online]. Available: `https://aaai.org/ojs/index.php/AAAI/article/view/5636`.

[269] E. Hazan, S. M. Kakade, K. Singh, and A. V. Soest, "Provably efficient maximum entropy exploration," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, K. Chaudhuri and R. Salakhutdinov, Eds., ser. Proceedings of Machine Learning Research, vol. 97, PMLR, 2019, pp. 2681–2691. [Online]. Available: `http://proceedings.mlr.press/v97/hazan19a.html`.

[270] M. Laskin, A. Srinivas, and P. Abbeel, "CURL: contrastive unsupervised representations for reinforcement learning," in *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, ser. Proceedings of Machine Learning Research, vol. 119, PMLR, 2020, pp. 5639–5650. [Online]. Available: `http://proceedings.mlr.press/v119/laskin20a.html`.

[271] M. Laskin, K. Lee, A. Stooke, L. Pinto, P. Abbeel, and A. Srinivas, "Reinforcement learning with augmented data," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020. [Online]. Available: `https://proceedings.neurips.cc/paper/2020/hash/e615c82aba461681ade82da2da38004a-Abstract.html`.

[272] T. Lesort, M. Seurin, X. Li, N. D. Rodríguez, and D. Filliat, "Unsupervised state representation learning with robotic priors: A robustness benchmark," *CoRR*, vol. abs/1709.05185, 2017. arXiv: `1709.05185`. [Online]. Available: `http://arxiv.org/abs/1709.05185`.

[273] A. Sharma, M. Ahn, S. Levine, V. Kumar, K. Hausman, and S. Gu, "Emergent real-world robotic skills via unsupervised off-policy reinforcement learning," *CoRR*, vol. abs/2004.12974, 2020. arXiv: `2004.12974`. [Online]. Available: `https://arxiv.org/abs/2004.12974`.

[274] OpenAI, M. Plappert, R. Sampedro, T. Xu, I. Akkaya, V. Kosaraju, P. Welinder, R. D'Sa, A. Petron, H. P. de Oliveira Pinto, A. Paino, H. Noh, L. Weng, Q. Yuan, C. Chu, and W. Zaremba, "Asymmetric self-play for automatic goal discovery in robotic manipulation," *CoRR*, vol. abs/2101.04882, 2021. arXiv: `2101.04882`. [Online]. Available: `https://arxiv.org/abs/2101.04882`.

[275] I. Mordatch, Z. Popović, and E. Todorov, "Contact-invariant optimization for hand manipulation," in *Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation*, Eurographics Association, 2012.

[276] V. Kumar, Y. Tassa, T. Erez, and E. Todorov, "Real-time behaviour synthesis for dynamic hand-manipulation," in *ICRA*, 2014.

[277] H. van Hoof, T. Hermans, G. Neumann, and J. Peters, "Learning robot in-hand manipulation with tactile features," in *Humanoid Robots (Humanoids)*, IEEE, 2015.

[278] R. Deimel and O. Brock, "A novel type of compliant and underactuated robotic hand for dexterous grasping," *I. J. Robotics Res.*, vol. 35, no. 1-3, pp. 161–185, 2016. DOI: `10.1177/0278364915592961`. [Online]. Available: `https://doi.org/10.1177/0278364915592961`.

[279] A. Gupta, C. Eppner, S. Levine, and P. Abbeel, "Learning dexterous manipulation for a soft robotic hand from human demonstrations," in *IROS*, 2016.

[280] H. B. Amor, O. Kroemer, U. Hillenbrand, G. Neumann, and J. Peters, "Generalization of human grasping for multi-fingered robot hands," in *IROS 2012*.

[281] V. Kumar, Z. Xu, and E. Todorov, “Fast, strong and compliant pneumatic actuation for dexterous tendon-driven hands,” in *ICRA*, 2013.

[282] Z. Xu, V. Kumar, and E. Todorov, “A low-cost and modular, 20-dof anthropomorphic robotic hand: Design, actuation and modeling,” in *Humanoids*, 2013.

[283] M. Posa, C. Cantu, and R. Tedrake, “A direct method for trajectory optimization of rigid bodies through contact,” *I. J. Robotics Res.*, vol. 33, no. 1, pp. 69–81, 2014. DOI: 10.1177/0278364913506757. [Online]. Available: https://doi.org/10.1177/0278364913506757.

[284] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, 1, vol. 1.

[285] E. Theodorou, J. Buchli, and S. Schaal, “A generalized path integral control approach to reinforcement learning,” *Journal of Machine Learning Research*, vol. 11, pp. 3137–3181, 2010.

[286] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *arXiv preprint arXiv:1509.02971*, 2015.

[287] J. Peters and S. Schaal, “Reinforcement learning of motor skills with policy gradients,” *Neural Networks*, 2008.

[288] J. Kober and J. Peters, “Policy search for motor primitives in robotics,” *Machine Learning*, vol. 84, no. 1-2, pp. 171–203, 2011. DOI: 10.1007/s10994-010-5223-6. [Online]. Available: https://doi.org/10.1007/s10994-010-5223-6.

[289] A. Ghadirzadeh, A. Maki, D. Kragic, and M. Björkman, “Deep predictive policy training using reinforcement learning,” *CoRR*, vol. abs/1703.00727, 2017. [Online]. Available: http://arxiv.org/abs/1703.00727.

[290] V. Kumar, E. Todorov, and S. Levine, “Optimal control with learned local models: Application to dexterous manipulation,” *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 378–383, 2016.

[291] V. Kumar, A. Gupta, E. Todorov, and S. Levine, “Learning Dexterous Manipulation Policies from Experience and Imitation,” *CoRR*, vol. abs/1611.05095, 2016.

[292] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, “End to end learning for self-driving cars,” *CoRR*, vol. abs/1604.07316, 2016.

[293] E. Theodorou, J. Buchli, and S. Schaal, “Reinforcement learning of motor skills in high dimensions: A path integral approach,” in *ICRA*, 2010.

[294] A. J. Ijspeert, J. Nakanishi, and S. Schaal, “Movement imitation with nonlinear dynamical systems in humanoid robots,” in *ICRA*, 2002.

[295] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband, G. Dulac-Arnold, J. Agapiou, J. Z. Leibo, and A. Gruslys, "Deep q-learning from demonstrations," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018*, 2018. [Online]. Available: `https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16976`.

[296] M. E. Taylor, H. B. Suay, and S. Chernova, "Integrating reinforcement learning with human demonstrations of varying ability," in *AAMAS*, 2011.

[297] T. Brys, A. Harutyunyan, H. B. Suay, S. Chernova, M. E. Taylor, and A. Nowé, "Reinforcement learning from demonstration through shaping," in *IJCAI*, 2015.

[298] K. Subramanian, C. L. I. Jr., and A. L. Thomaz, "Exploration from demonstration for interactive reinforcement learning," in *AAMAS*, ACM, 2016, pp. 447–456.

[299] M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. A. Riedmiller, "Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards," *CoRR*, vol. abs/1707.08817, 2017. [Online]. Available: `http://arxiv.org/abs/1707.08817`.

[300] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Overcoming exploration in reinforcement learning with demonstrations," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 6292–6299.

[301] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[302] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne, "Deepmimic: Example-guided deep reinforcement learning of physics-based character skills," in *ACM SIGGRAPH*, 2018.

[303] Y. Gao, H. Xu, J. Lin, F. Yu, S. Levine, and T. Darrell, "Reinforcement learning from imperfect demonstrations," *CoRR*, vol. abs/1802.05313, 2018. [Online]. Available: `http://arxiv.org/abs/1802.05313`.

[304] T. Erez, Y. Tassa, and E. Todorov, "Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx," in *ICRA*, 2015.

[305] V. Kumar and E. Todorov, "Mujoco haptix: A virtual reality system for hand manipulation," in *Humanoids*, 2015.

[306] S. Kakade, "A natural policy gradient," in *NIPS*, 2001.

[307] J. Peters, "Machine learning of motor skills for robotics," *PhD Dissertation, University of Southern California*, 2007.

[308] S. Amari, "Natural gradient works efficiently in learning," *Neural Computation*, vol. 10, pp. 251–276, 1998.

[309] J. A. Bagnell and J. G. Schneider, "Covariant policy search," in *IJCAI*, 2003.

[310] J. Peters and S. Schaal, “Natural actor-critic,” *Neurocomputing*, vol. 71,

[311] S. Kakade and J. Langford, “Approximately optimal approximate reinforcement learning,” in *ICML*, vol. 2, 2002, pp. 267–274.

[312] W. Sun, A. Venkatraman, G. J. Gordon, B. Boots, and J. A. Bagnell, “Deeply aggrevated: Differentiable imitation learning for sequential prediction,” in *ICML*, 2017.

[313] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. M. A. Eslami, M. A. Riedmiller, and D. Silver, “Emergence of locomotion behaviours in rich environments,” *CoRR*, vol. abs/1707.02286, 2017. [Online]. Available: `http://arxiv.org/abs/1707.02286`.

[314] I. Mordatch, K. Lowrey, G. Andrew, Z. Popovic, and E. Todorov, “Interactive Control of Diverse Complex Characters with Neural Networks,” in *NIPS*, 2015.

[315] A. Rajeswaran, S. Ghotra, B. Ravindran, and S. Levine, “EPOpt: Learning Robust Neural Network Policies Using Model Ensembles,” in *ICLR*, 2017.

[316] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, “QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation,” in *Conference on Robot Learning (CoRL)*, 2018. arXiv: `1806.10293v3`. [Online]. Available: `https://goo.gl/ykQn6g.`.

[317] A. G. Barto and S. Mahadevan, “Recent advances in hierarchical reinforcement learning,” *Discrete Event Dynamic Systems*, 2003.

[318] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, “Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation,” in *NeurIPS 2016*.

[319] R. S. Sutton, D. Precup, and S. P. Singh, “Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning,” *Artif. Intell.*, 1999.

[320] S. Krishnan, A. Garg, R. Liaw, B. Thananjeyan, L. Miller, F. T. Pokorny, and K. Goldberg, “SWIRL: A sequential windowed inverse reinforcement learning algorithm for robot tasks with delayed rewards,” *I. J. Robotics Res.*, vol. 38, no. 2-3, 2019. DOI: `10.1177/0278364918784350`. [Online]. Available: `https://doi.org/10.1177/0278364918784350`.

[321] R. Fox, S. Krishnan, I. Stoica, and K. Goldberg, “Multi-level discovery of deep options,” *CoRR*, vol. abs/1703.08294, 2017. arXiv: `1703.08294`. [Online]. Available: `http://arxiv.org/abs/1703.08294`.

[322] R. Parr and S. J. Russell, “Reinforcement learning with hierarchies of machines,” in *Advances in Neural Information Processing Systems 10, [NIPS Conference, Denver, Colorado, USA, 1997]*, 1997, pp. 1043–1049. [Online]. Available: `http://papers.nips.cc/paper/1384-reinforcement-learning-with-hierarchies-of-machines`.

[323] T. G. Dietterich, "Hierarchical reinforcement learning with the MAXQ value function decomposition," *J. Artif. Intell. Res.*, vol. 13, pp. 227–303, 2000. DOI: 10.1613/jair.639. [Online]. Available: https://doi.org/10.1613/jair.639.

[324] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu, "FeUdal Networks for Hierarchical Reinforcement Learning," in *International Conference on Machine Learning (ICML)*, 2017. arXiv: 1703.01161. [Online]. Available: https://arxiv.org/pdf/1703.01161.pdfhttp://arxiv.org/abs/1703.01161.

[325] O. Nachum, S. S. Gu, H. Lee, and S. Levine, "Data-Efficient Hierarchical Reinforcement Learning," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2018. arXiv: arXiv:1805.08296v2. [Online]. Available: https://sites.google.com/view/efficient-hrl.

[326] A. Levy, R. Platt, and K. Saenko, "Hierarchical Actor-Critic," *arXiv preprint arXiv:1712.00948*, 2017.

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

[328] K. Hausman, Y. Chebotar, S. Schaal, G. S. Sukhatme, and J. J. Lim, "Multi-modal imitation learning from unstructured demonstrations using generative adversarial nets," in *NeurIPS 2017*.

[329] P. Henderson, W. Chang, P. Bacon, D. Meger, J. Pineau, and D. Precup, "Optiongan: Learning joint reward-policy options using generative adversarial inverse reinforcement learning," in *AAAI 2018*.

[330] A. Sharma, M. Sharma, N. Rhinehart, and K. M. Kitani, "Directed-info GAIL: learning hierarchical policies from unsegmented demonstrations using directed information," *CoRR*, vol. abs/1810.01266, 2018. arXiv: 1810.01266.

[331] T. Kipf, Y. Li, H. Dai, V. F. Zambaldi, A. Sanchez-Gonzalez, E. Grefenstette, P. Kohli, and P. Battaglia, "Compile: Compositional imitation learning and execution," in *ICML 2019*.

[332] H. M. Le, N. Jiang, A. Agarwal, M. Dudík, Y. Yue, and H. D. III, "Hierarchical imitation and reinforcement learning,"

[333] D. Ghosh, A. Singh, A. Rajeswaran, V. Kumar, and S. Levine, "Divide-and-conquer reinforcement learning," *CoRR*, vol. abs/1711.09874, 2017. arXiv: 1711.09874. [Online]. Available: http://arxiv.org/abs/1711.09874.

[334] A. A. Rusu, S. G. Colmenarejo, Ç. Gülçehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell, "Policy distillation," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016. [Online]. Available: http://arxiv.org/abs/1511.06295.

[335] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Association for Compuational Linguistics (ACL)*, 2019. arXiv: `1810.04805`. [Online]. Available: `http://arxiv.org/abs/1810.04805`.

[336] S. Schaal, "Learning from demonstration," in *Advances in Neural Information Processing Systems (NeurIPS)*, 1997, pp. 1040–1046, ISBN: 1558604863. DOI: `10.1016/j.robot.2004.03.001`. [Online]. Available: `http://www.cc.gatech.edulfachttp://wwwiaim.ira.uka.de/users/rogalla/WebOrdnerMaterial/ml-robotlearning.pdfhttp://www.cc.gatech.edulfac/Stefan.Schaal`.

[337] C. G. Atkeson and S. Schaal, "Robot Learning From Demonstration," in *International Conference on Machine Learning (ICML)*, 1997. [Online]. Available: `http://www.cc.gatech.edu/fac/fChris..`

[338] A. Zhou, E. Jang, D. Kappler, A. Herzog, M. Khansari, P. Wohlhart, Y. Bai, M. Kalakrishnan, S. Levine, and C. Finn, "Watch, try, learn: Meta-learning from demonstrations and reward," *CoRR*, vol. abs/1906.03352, 2019. arXiv: `1906.03352`. [Online]. Available: `http://arxiv.org/abs/1906.03352`.

[339] S. Fujimoto, D. Meger, and D. Precup, "Off-Policy Deep Reinforcement Learning without Exploration," in *International Conference on Machine Learning (ICML)*, 2019. arXiv: `1812.02900`. [Online]. Available: `http://arxiv.org/abs/1812.02900`.

[340] A. Kumar, J. Fu, G. Tucker, and S. Levine, "Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction," in *Neural Information Processing Systems (NeurIPS)*, 2019. arXiv: `1906.00949`. [Online]. Available: `http://arxiv.org/abs/1906.00949`.

[341] A. Abdolmaleki, J. T. Springenberg, Y. Tassa, R. Munos, N. Heess, and M. Riedmiller, "Maximum a Posteriori Policy Optimisation," in *International Conference on Learning Representations (ICLR)*, 2018, pp. 1–19.

[342] V. R. Konda and J. N. Tsitsiklis, "Actor-Critic Algorithms," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2000.

[343] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. 1998. [Online]. Available: `http://incompleteideas.net/sutton/book/bookdraft2016sep.pdfhttps://webdocs.cs.ualberta.ca/{~}sutton/book/bookdraft2016sep.pdf`.

[344] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods," *International Conference on Machine Learning (ICML)*, 2018.

[345] V. Mnih, A. Puigdomènech Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," in *International Conference on Machine Learning (ICML)*, 2016. [Online]. Available: `https://arxiv.org/pdf/1602.01783.pdf`.

[346] A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman, "Relay policy learning: Solving long horizon tasks via imitation and reinforcement learning," *Conference on Robot Learning (CoRL)*, 2019.

[347] J. Peters and S. Schaal, "Reinforcement Learning by Reward-weighted Regression for Operational Space Control," in *International Conference on Machine Learning*, 2007. [Online]. Available: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.79.6266{\&}rep=rep1{\&}type=pdf`.

[348] Q. Wang, J. Xiong, L. Han, P. Sun, H. Liu, and T. Zhang, "Exponentially Weighted Imitation Learning for Batched Historical Data," in *Neural Information Processing Systems (NeurIPS)*, 2018.

[349] X. B. Peng, A. Kumar, G. Zhang, and S. Levine, "Advantage-Weighted Regression: Simple and Scalable Off-Policy Reinforcement Learning," 2019. arXiv: `1910.00177`. [Online]. Available: `http://arxiv.org/abs/1910.00177`.

[350] Y. Wu, G. Tucker, and O. Nachum, "Behavior Regularized Offline Reinforcement Learning," 2020. arXiv: `1911.11361`. [Online]. Available: `http://arxiv.org/abs/1911.11361`.

[351] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems," Tech. Rep., 2020. arXiv: `2005.01643v1`.

[352] N. Y. Siegel, J. T. Springenberg, F. Berkenkamp, A. Abdolmaleki, M. Neunert, T. Lampe, R. Hafner, N. Heess, and M. Riedmiller, *Keep doing what worked: Behavioral modelling priors for offline reinforcement learning*, 2020. arXiv: `2002.08396 [cs.LG]`.

[353] J. Ramapuram, M. Gregorova, and A. Kalousis, "Lifelong Generative Modeling," *Neurocomputing*, 2017. arXiv: `1705.09847`. [Online]. Available: `http://arxiv.org/abs/1705.09847`.

[354] J. Peters, K. Mülling, and Y. Altün, "Relative Entropy Policy Search," in *AAAI Conference on Artificial Intelligence*, 2010, pp. 1607–1612. [Online]. Available: `https://pdfs.semanticscholar.org/ff47/526838ce85d77a50197a0c5f6ee5095156aa.pdfhttp://www-clmc.usc.edu/publications/P/Peters{\_}POTTNCOAIPGAT{\_}2010.pdf`.

[355] T. Degris, M. White, and R. S. Sutton, "Off-Policy Actor-Critic," in *International Conference on Machine Learning (ICML)*, 2012. arXiv: `1205.4839`. [Online]. Available: `http://arxiv.org/abs/1205.4839`.

[356] S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, and M. Lee, "Natural actor-critic algorithms," *Autom.*, vol. 45, no. 11, pp. 2471–2482, 2009. DOI: `10.1016/j.automatica.2009.07.008`. [Online]. Available: `https://doi.org/10.1016/j.automatica.2009.07.008`.

[357] S. Zhang, W. Boehmer, and S. Whiteson, “Generalized off-policy actor-critic,” in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d´ Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 2001–2011. [Online]. Available: `http://papers.nips.cc/paper/8474-generalized-off-policy-actor-critic.pdf`.

[358] P. Wawrzynski, “Real-time reinforcement learning by sequential actor-critics and experience replay,” *Neural Networks*, vol. 22, no. 10, pp. 1484–1497, 2009. DOI: `10.1016/j.neunet.2009.05.011`. [Online]. Available: `https://doi.org/10.1016/j.neunet.2009.05.011`.

[359] D. Balduzzi and M. Ghifary, “Compatible value gradients for reinforcement learning of continuous deep policies,” *CoRR*, vol. abs/1509.03005, 2015. arXiv: `1509.03005`. [Online]. Available: `http://arxiv.org/abs/1509.03005`.

[360] S. Lange, T. Gabel, and M. A. Riedmiller, “Batch reinforcement learning,” in *Reinforcement Learning*, ser. Adaptation, Learning, and Optimization, M. Wiering and M. van Otterlo, Eds., vol. 12, Springer, 2012, pp. 45–73. DOI: `10.1007/978-3-642-27645-3\_2`. [Online]. Available: `https://doi.org/10.1007/978-3-642-27645-3\_2`.

[361] P. S. Thomas and E. Brunskill, “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*, M. Balcan and K. Q. Weinberger, Eds., ser. JMLR Workshop and Conference Proceedings, vol. 48, JMLR.org, 2016, pp. 2139–2148. [Online]. Available: `http://proceedings.mlr.press/v48/thomasa16.html`.

[362] A. Hallak, F. Schnitzler, T. Mann, and S. Mannor, “Off-policy Model-based Learning under Unknown Factored Dynamics,” in *International Conference on Machine Learning (ICML)*, 2015.

[363] A. Hallak, A. Tamar, R. Munos, and S. Mannor, “Generalized Emphatic Temporal Difference Learning: Bias-Variance Analysis,” in *Association for the Advancement of Artificial Intelligence (AAAI)*, 2016. arXiv: `1509.05172v2`.

[364] A. Hallak and S. Mannor, “Consistent On-Line Off-Policy Evaluation,” in *International Conference on Machine Learning (ICML)*, 2017. arXiv: `1702.07121v1`.

[365] R. Agarwal, D. Schuurmans, and M. Norouzi, “An Optimistic Perspective on Offline Reinforcement Learning,” in *International Conference on Machine Learning (ICML)*, 2019. arXiv: `1907.04543v2`.

[366] R. Fakoor, P. Chaudhari, and A. J. Smola, “P3O: Policy-on Policy-off Policy Optimization,” in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2019. arXiv: `1905.01756v2`. [Online]. Available: `https://github.com/rasoolfa/P3O.`.

[367] O. Nachum, Y. Chow, B. Dai, and L. Li, “DualDICE: Behavior-Agnostic Estimation of Discounted Stationary Distribution Corrections,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. arXiv: `1906.04733`. [Online]. Available: `http://arxiv.org/abs/1906.04733`.

[368] R. Zhang, B. Dai, L. Li, and D. Schuurmans, “GenDICE: Generalized Offline Estimation of Stationary Values,” in *International Conference on Learning Representations (ICLR)*, 2020. arXiv: `2002.09072`. [Online]. Available: `http://arxiv.org/abs/2002.09072`.

[369] N. Jiang and L. Li, “Doubly Robust Off-policy Value Evaluation for Reinforcement Learning,” in *International Conference on Machine Learning (ICML)*, 2016. arXiv: `1511.03722v3`.

[370] N. Jaques, A. Ghandeharioun, J. H. Shen, C. Ferguson, À. Lapedriza, N. Jones, S. Gu, and R. W. Picard, “Way off-policy batch deep reinforcement learning of implicit human preferences in dialog,” *CoRR*, vol. abs/1907.00456, 2019. arXiv: `1907.00456`. [Online]. Available: `http://arxiv.org/abs/1907.00456`.

[371] G. Neumann and J. Peters, “Fitted Q-iteration by Advantage Weighted Regression,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2008.

[372] Z. Wang, A. Novikov, K. Zołna, J. T. Springenberg, S. Reed, B. Shahriari, N. Siegel, J. Merel, C. Gulcehre, N. Heess, and N. De Freitas, “Critic Regularized Regression,” 2020. arXiv: `2006.15134v1`.

[373] P. Kormushev, S. Calinon, and D. G. Caldwell, “Robot motor skill coordination with em-based reinforcement learning,” in *IROS*, IEEE, 2010.

[374] D. C. Bentivegna, G. Cheng, and C. G. Atkeson, “Learning from observation and from practice using behavioral primitives,” in *Robotics Research, The Eleventh International Symposium, ISRR, October 19-22, 2003, Siena, Italy*, P. Dario and R. Chatila, Eds., ser. Springer Tracts in Advanced Robotics, vol. 15, Springer, 2003, pp. 551–560. DOI: `10.1007/11008941\_59`. [Online]. Available: `https://doi.org/10.1007/11008941\_59`.

[375] H. Zhu, A. Gupta, A. Rajeswaran, S. Levine, and V. Kumar, “Dexterous Manipulation with Deep Reinforcement Learning: Efficient, General, and Low-Cost,” in *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2019-May, Institute of Electrical and Electronics Engineers Inc., 2019, pp. 3651–3657. arXiv: `1810.06045`. [Online]. Available: `http://arxiv.org/abs/1810.06045`.

[376] Y. Wu, M. Mozifian, and F. Shkurti, *Shaping rewards for reinforcement learning with imperfect demonstrations using generative models*, 2020. arXiv: `2011.01298 [cs.RO]`.

[377] M. Ahn, H. Zhu, K. Hartikainen, H. Ponte, A. Gupta, S. Levine, and V. Kumar, “ROBEL: Robotics Benchmarks for Learning with Low-Cost Robots,” in *Conference on Robot Learning (CoRL)*, arXiv, 2019. arXiv: `1909.11639`. [Online]. Available: `http://arxiv.org/abs/1909.11639`.

[378] A. Gupta, J. Yu, T. Z. Zhao, V. Kumar, A. Rovinsky, K. Xu, T. Devlin, and S. Levine, "Reset-free reinforcement learning via multi-task learning: Learning dexterous manipulation behaviors without human intervention," *arXiv preprint arXiv:2104.11203*, 2021.

[379] P. Mandikal and K. Grauman, *Dexterous robotic grasping with object-centric visual affordances*, 2020. arXiv: `2009.01439 [cs.RO]`.

[380] H. Xu, Y. Luo, S. Wang, T. Darrell, and R. Calandra, "Towards learning to play piano with dexterous hands and touch," *CoRR*, vol. abs/2106.02040, 2021. arXiv: `2106.02040`. [Online]. Available: `https://arxiv.org/abs/2106.02040`.

[381] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative Q-Learning for Offline Reinforcement Learning," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. arXiv: `2006.04779v1`.

[382] T. Yu, G. Thomas, L. Yu, S. Ermon, J. Y. Zou, S. Levine, C. Finn, and T. Ma, "MOPO: model-based offline policy optimization," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020. [Online]. Available: `https://proceedings.neurips.cc/paper/2020/hash/a322852ce0df73e204b7e67cbbef0d0a-Abstract.html`.

[383] R. Kidambi, A. Rajeswaran, P. Netrapalli, and T. Joachims, "Morel: Model-based offline reinforcement learning," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020. [Online]. Available: `https://proceedings.neurips.cc/paper/2020/hash/f7efa4f864ae9b88d43527f4b14f750f-Abstract.html`.

[384] Y. Chebotar, K. Hausman, Y. Lu, T. Xiao, D. Kalashnikov, J. Varley, A. Irpan, B. Eysenbach, R. Julian, C. Finn, and S. Levine, "Actionable models: Unsupervised offline reinforcement learning of robotic skills," *CoRR*, vol. abs/2104.07749, 2021. arXiv: `2104.07749`. [Online]. Available: `https://arxiv.org/abs/2104.07749`.

[385] X. Chen, Z. Zhou, Z. Wang, C. Wang, Y. Wu, and K. Ross, "BAIL: best-action imitation learning for batch deep reinforcement learning," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020. [Online]. Available: `https://proceedings.neurips.cc/paper/2020/hash/d55cbf210f175f4a37916eafe6c04f0d-Abstract.html`.

[386] Y. Liu, A. Swaminathan, A. Agarwal, and E. Brunskill, "Provably good batch off-policy reinforcement learning without great exploration," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020. [Online]. Available: `https://proceedings.`

neurips.cc/paper/2020/hash/0dc23b6a0e4abc39904388dd3ffadcd1-Abstract.html.

[387] A. Singh, A. Yu, J. Yang, J. Zhang, A. Kumar, and S. Levine, "Cog: Connecting new skills to past experience with offline reinforcement learning," *arXiv preprint arXiv:2010.14500*, 2020.

[388] T. Matsushima, H. Furuta, Y. Matsuo, O. Nachum, and S. Gu, "Deployment-efficient reinforcement learning via model-based offline optimization," *CoRR*, vol. abs/2006.03647, 2020. arXiv: 2006.03647. [Online]. Available: https://arxiv.org/abs/2006.03647.

[389] H. Zhu, A. Gupta, A. Rajeswaran, S. Levine, and V. Kumar, "Dexterous manipulation with deep reinforcement learning: Efficient, general, and low-cost," in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 3651–3657.

[390] Y. Chebotar, M. Kalakrishnan, A. Yahya, A. Li, S. Schaal, and S. Levine, "Path integral guided policy search," *CoRR*, vol. abs/1610.00529, 2016.

[391] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.

[392] C. Devin, P. Abbeel, T. Darrell, and S. Levine, "Deep object-centric representations for generalizable robot learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 7111–7118.

[393] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, "Scalable deep reinforcement learning for vision-based robotic manipulation," in *Proceedings of The 2nd Conference on Robot Learning*, A. Billard, A. Dragan, J. Peters, and J. Morimoto, Eds., ser. Proceedings of Machine Learning Research, vol. 87, PMLR, 2018. [Online]. Available: http://proceedings.mlr.press/v87/kalashnikov18a.html.

[394] B. Eysenbach, S. Gu, J. Ibarz, and S. Levine, "Leave no trace: Learning to reset for safe and autonomous reinforcement learning," in *International Conference on Learning Representations (ICLR)*, 2018.

[395] W. Han, S. Levine, and P. Abbeel, "Learning compound multi-step controllers under unknown dynamics," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2015, pp. 6435–6442.

[396] D. Kingma and M. Welling, "Auto-encoding variational Bayes," in *International Conference on Learning Representations (ICLR)*, 2014.

[397] A. X. Lee, A. Nagabandi, P. Abbeel, and S. Levine, "Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model," *CoRR*, vol. abs/1907.00953, 2019.

[398] R. D. Hjelm, A. Fedorov, S. Lavoie-Marchildon, K. Grewal, P. Bachman, A. Trischler, and Y. Bengio, "Learning deep representations by mutual information estimation and maximization," in *International Conference on Learning Representations*, 2019. [Online]. Available: `https://openreview.net/forum?id=Bklr3j0cKX`.

[399] A. Anand, E. Racah, S. Ozair, Y. Bengio, M. Côté, and R. D. Hjelm, "Unsupervised state representation learning in atari," *arXiv*, 2019. eprint: `1906.08226`.

[400] J. Tobin, W. Zaremba, and P. Abbeel, "Domain randomization and generative models for robotic grasping," *arXiv preprint arXiv:1710.06425*, 2017.

[401] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-Real Transfer of Robotic Control with Dynamics Randomization," in *International Conference on Robotics and Automation (ICRA)*, 2018, ISBN: 9781538630815. [Online]. Available: `https://xbpeng.github.io/projects/SimToReal/2018{\_}SimToReal.pdf`.

[402] K. Kang, S. Belkhale, G. Kahn, P. Abbeel, and S. Levine, "Generalization through Simulation: Integrating Simulated and Real Data into Deep Reinforcement Learning for Vision-Based Autonomous Flight," *arXiv e-prints*, arXiv:1902.03701, arXiv:1902.03701, 2019. arXiv: `1902.03701 [cs.LG]`.

[403] S. Levine and V. Koltun, "Guided policy search," in *Proceedings of The 30th International Conference on Machine Learning*, 2013.

[404] L. Pinto, M. Andrychowicz, P. Welinder, W. Zaremba, and P. Abbeel, "Asymmetric actor critic for image-based robot learning," *arXiv preprint arXiv:1710.06542*, 2017.

[405] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks," *arXiv preprint arXiv:1611.05397*, 2016.

[406] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta, "Robust Adversarial Reinforcement Learning," in *International Conference on Machine Learning (ICML)*, 2017. [Online]. Available: `https://arxiv.org/pdf/1703.02702.pdf`.

[407] M. Lungarella, G. Metta, R. Pfeifer, and G. Sandini, "Developmental robotics: A survey," *Connection Science*, vol. 15, no. 4, pp. 151–190, 2003. DOI: `10.1080/09540090310001655110`.

[408] M. Asada, K. Hosoda, Y. Kuniyoshi, H. Ishiguro, T. Inui, Y. Yoshikawa, M. Ogino, and C. Yoshida, "Cognitive developmental robotics: A survey," *IEEE Trans. Autonomous Mental Development*, vol. 1, no. 1, pp. 12–34, 2009. DOI: `10.1109/TAMD.2009.2021702`. [Online]. Available: `https://doi.org/10.1109/TAMD.2009.2021702`.

[409] T. Lesort, V. Lomonaco, A. Stoian, D. Maltoni, D. Filliat, and N. Díaz-Rodríguez, "Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges," *Information Fusion*, vol. 58, pp. 52–68, 2020.

[410] S. Thrun and T. M. Mitchell, "Lifelong robot learning," *Robotics and Autonomous Systems*, vol. 15, no. 1-2, pp. 25–46, 1995. DOI: `10.1016/0921-8890(95)00004-Y`. [Online]. Available: `https://doi.org/10.1016/0921-8890(95)00004-Y`.

[411] A. Stoica, "Robot fostering techniques for sensory-motor development of humanoid robots," *Robotics and Autonomous Systems*, vol. 37, no. 2-3, pp. 127–143, 2001. DOI: 10.1016/S0921-8890(01)00154-3. [Online]. Available: https://doi.org/10.1016/S0921-8890(01)00154-3.

[412] S. Levine, N. Wagener, and P. Abbeel, "Learning contact-rich manipulation skills with guided policy search," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, IEEE, 2015.

[413] L. Pinto and A. Gupta, "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours," in *ICRA*, 2016.

[414] K. Ploeger, M. Lutter, and J. Peters, *High acceleration reinforcement learning for real-world juggling with binary rewards*, 2020. arXiv: 2010.13483 [cs.RO].

[415] P. Agrawal, A. Nair, P. Abbeel, J. Malik, and S. Levine, "Learning to Poke by Poking: Experiential Learning of Intuitive Physics," in *Advances in Neural Information Processing Systems (NIPS)*, 2016. arXiv: 1606.07419. [Online]. Available: http://arxiv.org/abs/1606.07419.

[416] C. Richter and N. Roy, "Safe visual navigation via deep learning and novelty detection," in *Robotics: Science and Systems XIII, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, July 12-16, 2017*, N. M. Amato, S. S. Srinivasa, N. Ayanian, and S. Kuindersma, Eds., 2017. DOI: 10.15607/RSS.2017.XIII.064. [Online]. Available: http://www.roboticsproceedings.org/rss13/p64.html.

[417] Y. W. Teh, V. Bapst, W. M. Czarnecki, J. Quan, J. Kirkpatrick, R. Hadsell, N. Heess, and R. Pascanu, "Distral: Robust multitask reinforcement learning," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, Eds., 2017, pp. 4496–4506. [Online]. Available: http://papers.nips.cc/paper/7036-distral-robust-multitask-reinforcement-learning.

[418] E. Parisotto, J. L. Ba, and R. Salakhutdinov, "Actor-mimic: Deep multitask and transfer reinforcement learning," in *Proc. International Conference on Learning Representations*, 2016.

[419] T. Yu, S. Kumar, A. Gupta, S. Levine, K. Hausman, and C. Finn, "Gradient surgery for multi-task learning," *CoRR*, vol. abs/2001.06782, 2020. arXiv: 2001.06782. [Online]. Available: https://arxiv.org/abs/2001.06782.

[420] O. Sener and V. Koltun, "Multi-task learning as multi-objective optimization," *CoRR*, vol. abs/1810.04650, 2018. arXiv: 1810.04650. [Online]. Available: http://arxiv.org/abs/1810.04650.

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

[422] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[423] D. Gandhi, L. Pinto, and A. Gupta, "Learning to Fly by Crashing," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017. [Online]. Available: `https://arxiv.org/pdf/1704.05588.pdf`.

[424] C. Finn, I. Goodfellow, and S. Levine, "Unsupervised Learning for Physical Interaction through Video Prediction," in *Advances in Neural Information Processing Systems (NIPS)*, 2016. arXiv: `1605.07157`. [Online]. Available: `http://arxiv.org/abs/1605.07157`.

[425] S. Ha, P. Xu, Z. Tan, S. Levine, and J. Tan, "Learning to walk in the real world with minimal human effort," *arXiv preprint arXiv:2002.08550*, 2020.

[426] K. Xu, S. Verma, C. Finn, and S. Levine, "Continual learning of control primitives : Skill discovery via reset-games," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020. [Online]. Available: `https://proceedings.neurips.cc/paper/2020/hash/3472ab80b6dff70c54758fd6dfc800c2-Abstract.html`.

[427] A. Sharma, A. Gupta, K. Hausman, S. Levine, and C. Finn, "Persistent reinforcement learning via subgoal curricula," *ICLR Workshop on Never Ending RL*, 2021.

[428] C. Devin, A. Gupta, T. Darrell, P. Abbeel, and S. Levine, "Learning modular neural network policies for multi-task and multi-robot transfer," *arXiv preprint arXiv:1609.07088*, 2016.

[429] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

[430] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine, "D4RL: Datasets for Deep Data-Driven Reinforcement Learning," 2020. arXiv: `2004.07219`. [Online]. Available: `http://arxiv.org/abs/2004.07219`.

[431] H. Zhang, M. Cissé, Y. N. Dauphin, and D. Lopez-Paz, "Mixup: Beyond empirical risk minimization," in *International Conference on Learning Representations*, 2018.

Chapter 18

Appendices

18.1 Appendix A: Appendix for Chapter 3

Proof of Theorem 1 connecting NML and inverse counts

We provide the proof of Theorem 1 here for completeness.

Theorem 5. Suppose we are estimating success probabilities $p(e = 1|s)$ in the tabular setting, where we have a separate parameter independently for each state. Let $N(s)$ denote the number of times state s has been visited by the policy, and let $G(s)$ be the number of occurrences of state s in the successful outcomes. Then the CNML probability $p_{\text{CNML}}(e = 1|s)$ is equal to $\frac{G(s)+1}{N(s)+G(s)+2}$. For states that are never observed to be successful, we then recover inverse counts $\frac{1}{N(s)+2}$.

Proof. In the fully tabular setting, our MLE estimates for $p(O|s)$ are simply given by finding the best parameter p_s for each state. The proof then proceeds by simple calculation.

For a state with $n = N(s)$ negative occurrences and $g = G(s)$ positive occurrences, the MLE estimate is simply given by $\frac{g}{n+g}$.

Now for evaluating CNML, we consider appending another instance for each class. The new parameter after appending a negative example is then $\frac{g}{n+g+1}$, which then assigns probability $\frac{n+1}{n+g+1}$ to the negative class. Similarly, after appending a positive example, the new parameter is $\frac{g+1}{n+g+1}$, so we try to assign probability $\frac{g+1}{n+g+1}$ to the positive class. Normalizing, we have

$$p_{\text{CNML}}(O = 1|s) = \frac{g+1}{n+g+2}. \tag{18.1}$$

When considering states that have only been visited on-policy, and are not included in the set of successful outcomes, then the likelihood reduces to

$$p_{\text{CNML}}(O = 1|s) = \frac{1}{n+2}. \tag{18.2}$$

□

Detailed Description of Meta-NML

We provide a detailed description of the meta-NML algorithm described in Section 3.5, and the details of the practical algorithm.

Given a dataset $\mathcal{D} = \{(x_0, y_0), (x_1, y_1), .., (x_n, y_n)\}$, the meta-NML procedure proceeds by first constructing $k * n$ tasks from these data points, for a k shot classification problem. We will keep $k = 2$ for simplicity in this description, in accordance with the setup of binary success classifiers in RL. Each task τ_i is constructed by augmenting the dataset with a negative label $\mathcal{D} \cup (x_i, y = 0)$ or a positive label $\mathcal{D} \cup (x_i, y = 1)$. Now that each task consists of solving the maximum likelihood problem for its augmented dataset, we can directly apply standard meta-learning algorithms to this setting. Building off the ideas in MAML [140], we can then train a set of model parameters θ such

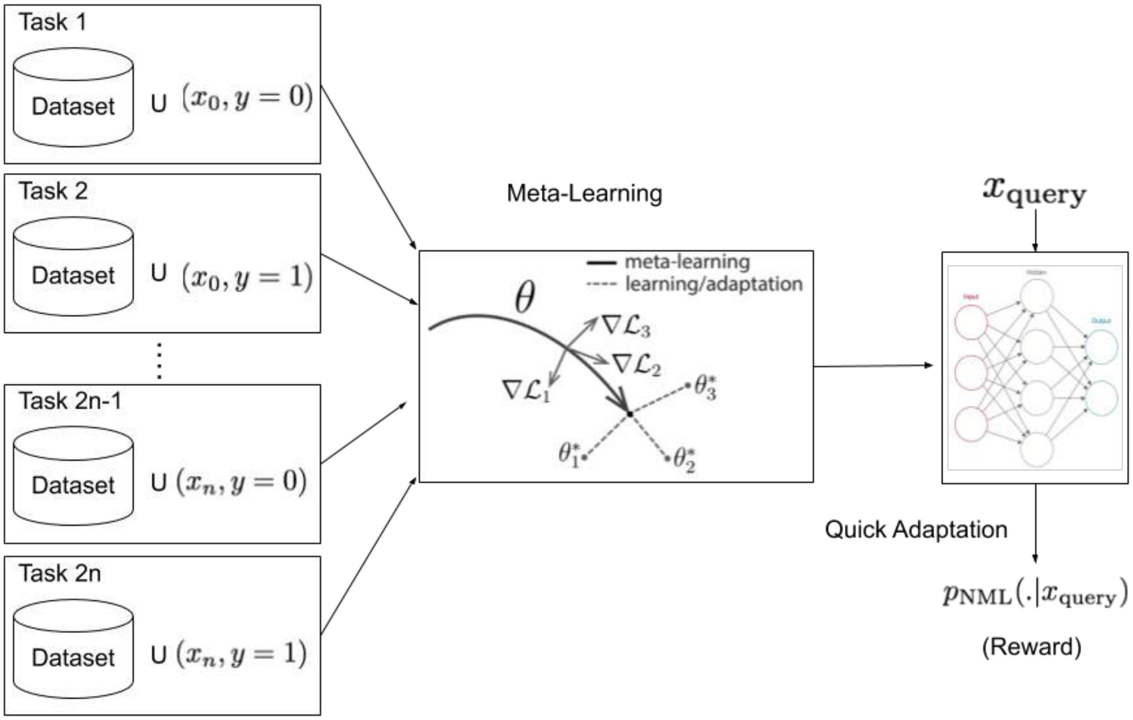


Figure 18.1: Figure illustrating the meta-training procedure for meta-NML.

that after a single step of gradient descent it can quickly adapt to the optimal solution for the MLE problem on any of the augmented datasets. This is more formally written as

$$\max_{\theta} \; \mathbb{E}_{\tau \sim \mathcal{S}(\tau)}[\mathcal{L}(\tau, \theta')], \quad s.t \;\; \theta' = \theta - \alpha \nabla_\theta \mathcal{L}(\tau, \theta) \tag{18.3}$$

where $\mathcal{L}$ represents a standard classification loss function, α is the learning rate, and the distribution of tasks $p(\tau)$ is constructed as described above. For a new query point x, these initial parameters can then quickly be adapted to provide the CNML distribution by taking a gradient step on each augmented dataset to obtain the approximately optimal MLE solution, and normalizing these as follows:

$$p_{\text{meta-NML}}(y|x; \mathcal{D}) = \frac{p_{\theta_y}(y|x)}{\sum_{y \in \mathcal{Y}} p_{\theta_y}(y|x)}$$

$$\theta_y = \theta - \alpha \nabla_\theta \mathbb{E}_{(x_i, y_i) \sim \mathcal{D} \cup (x,y)}[\mathcal{L}(x_i, y_i, \theta)]$$

This algorithm in principle can be optimized using any standard stochastic optimization method such as SGD, as described in [140], backpropagating through the inner loop gradient update. For the specific problem setting that we consider, we additionally employ some optimization tricks in order to enable learning:

Importance Weighting on Query Point

Since only one datapoint is augmented to the training set at query time for CNML, stochastic gradient descent can ignore this datapoint with increasing dataset sizes. For example, if we train on an augmented dataset of size 2048 by cycling through it in batch sizes of 32, then only 1 in 64 batches would include the query point itself and allow the model to adapt to the proposed label, while the others would lead to noise in the optimization process, potentially worsening the model's prediction on the query point.

In order to make sure the optimization considers the query point, we include the query point and proposed label (x_q, y) in every minibatch that is sampled, but downweight the loss computed on that point such that the overall objective remains unbiased. This is simply doing importance weighting, with the query point downweighted by a factor of $\lceil \frac{b-1}{N} \rceil$ where b is the desired batch size and N is the total number of points in the original dataset.

To see why the optimization objective remains the same, we can consider the overall loss over the dataset. Let f_θ be our classifier, $\mathcal{L}$ be our loss function, $\mathcal{D}' = \{(x_i, y_i)\}_{i=1}^N \cup (x_q, y)$ be our augmented dataset, and $\mathcal{B}_k$ be the kth batch seen during training. Using standard SGD training that cycles through batches in the dataset, the overall loss on the augmented dataset would be:

$$\mathcal{L}(\mathcal{D}') = \left(\sum_{i=0}^{N} \mathcal{L}(f_\theta(x_i), y_i) \right) + \mathcal{L}(f_\theta(x_q), y)$$

If we instead included the downweighted query point in every batch, the overall loss would be:

$$\begin{aligned}
\mathcal{L}(\mathcal{D}') &= \sum_{k=0}^{\lceil \frac{b-1}{N} \rceil} \sum_{(x_i, y_i) \in \mathcal{B}_k} \left(\mathcal{L}(f_\theta(x_i), y_i) + \frac{1}{\lceil \frac{b-1}{N} \rceil} \mathcal{L}(f_\theta(x_q), y) \right) \\
&= \left(\sum_{k=0}^{\lceil \frac{b-1}{N} \rceil} \sum_{(x_i, y_i) \in \mathcal{B}_k} \mathcal{L}(f_\theta(x_i), y_i) \right) + \\
&\lceil \frac{b-1}{N} \rceil \frac{1}{\lceil \frac{b-1}{N} \rceil} \mathcal{L}(f_\theta(x_q), y) \\
&= \left(\sum_{i=0}^{N} \mathcal{L}(f_\theta(x_i), y_i) \right) + \mathcal{L}(f_\theta(x_q), y)
\end{aligned}$$

which is the same objective as before.

This trick has the effect of still optimizing the same maximum likelihood problem required by CNML, but significantly reducing the variance of the query point predictions as we take additional gradient steps at query time. As a concrete example, consider querying a meta-CNML classifier on the input shown in Figure 18.2. If we adapt to the augmented dataset without including the query point in every batch (i.e. without importance weighting), we see that the query point loss is significantly more unstable, requiring us to take more gradient steps to converge.

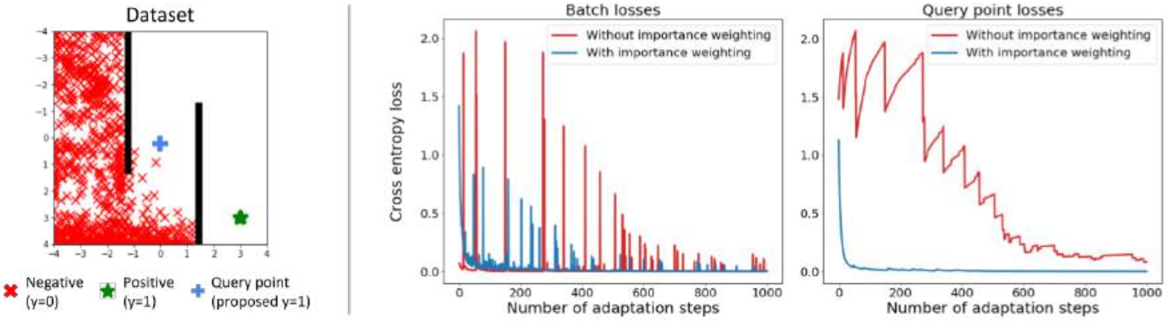


Figure 18.2: Comparison of adapting to a query point (pictured on left with the original dataset) at test time for CNML with and without importance weighting. The version without importance weighting is more unstable both in terms of overall batch loss and the individual query point loss, and thus takes longer to converge. The spikes in the red lines occur when that particular batch happens to include the query point, since that point's proposed label ($y = 1$) is different than those of nearby points ($y = 0$). The version with importance weighting does not suffer from this problem because it accounts for the query point in each gradient step, while keeping the optimization objective the same.

Kernel Weighted Training Loss

The augmented dataset consists of points from the original dataset $\mathcal{D}$ and one augmented point (x_q, y). Given that we mostly care about having the proper likelihood on the query point, with an imperfect optimization process, the meta-training can yield solutions that are not very accurately representing true likelihoods on the query point. To counter this, we introduce a kernel weighting into the loss function in Equation 18.3 during meta-training and subsequently meta-testing. The kernel weighting modifies the training loss function as:

$$\max_{\theta} \mathbb{E}_{\tau \sim \mathcal{S}(\tau)}[\mathbb{E}_{(x,y)\sim\tau}\mathcal{K}(x, x_\tau)\mathcal{L}(x, y, \theta')]$$
$$s.t \;\; \theta' = \theta - \alpha\nabla_\theta \mathbb{E}_{(x,y)\sim\tau}\mathcal{K}(x, x_\tau)\mathcal{L}(x, y, \theta)$$

where x_τ is the query point for task τ and $\mathcal{K}$ is a choice of kernel. We typically choose exponential kernels centered around x_τ. Intuitively, this allows the meta-optimization to mainly consider the datapoints that are copies of the query point in the dataset, or are similar to the query point, and ensures that they have the correct likelihoods, instead of receiving interfering gradient signals from the many other points in the dataset. To make hyperparameter selection intuitive, we designate the strength of the exponential kernel by a parameter λ_{dist}, which is the Euclidean distance away from the query point at which the weight becomes 0.1. Formally, the weight of a point x in the loss function for query point x_τ is computed as:

$$K(x, x_\tau) = \exp\{-\frac{2.3}{\lambda_{dist}}||x - x_\tau||_2\} \tag{18.4}$$

Meta-Training at Fixed Intervals

While in principle meta-NML would retrain with every new datapoint, in practice we retrain meta-NML once every k epochs. (In all of our experiments we set $k = 1$, but we could optionally increase k if we do not expect the meta-task distribution to change much between epochs.) We warm-start the meta-learner parameters from the previous iteration of meta-learning, so every instance of meta-training only requires a few steps. We find that this periodic training is a reasonable enough approximation, as evidenced by the strong performance of MURAL in our experimental results in Section 13.7.

Meta-NML Visualizations

Meta-NML with Additional Gradient Steps

Below, we show a more detailed visualization of meta-NML outputs on data from the Zigzag Maze task, and how these outputs change with additional gradient steps. For comparison, we also include the idealized NML rewards, which come from a discrete count-based classifier.

Meta-NML is able to resemble the ideal NML rewards fairly well with just 1 gradient step, providing both an approximation of a count-based exploration bonus and better shaping towards the goal due to generalization. By taking additional gradient steps, meta-NML can get arbitrarily close to the true NML outputs, which themselves correspond to inverse counts of $\frac{1}{n+2}$ as explained in Theorem 4.1. While this would give us more accurate NML estimates, in practice we found that taking one gradient step was sufficient to achieve good performance on our RL tasks.

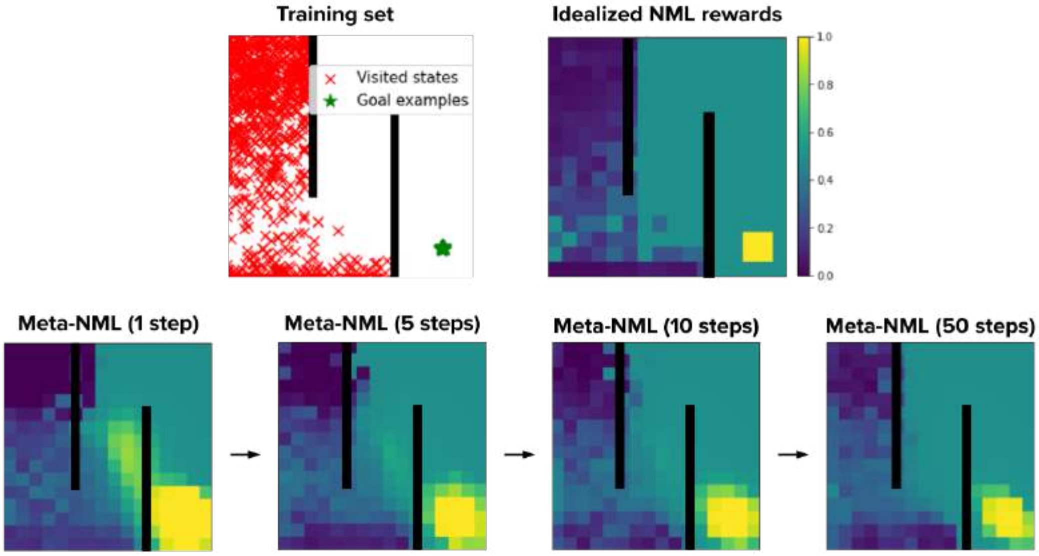


Figure 18.3: Comparison of idealized (discrete) NML and meta-NML rewards on data from the Zigzag Maze Task. Meta-NML approximates NML reasonably well with just one gradient step at test time, and converges to the true values with additional steps.

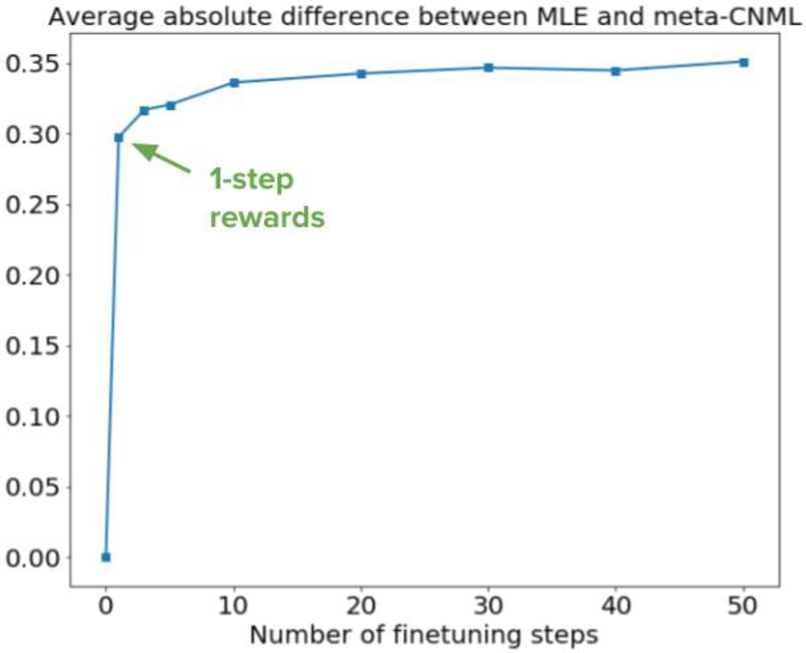


Figure 18.4: Average absolute difference between MLE and meta-NML goal probabilities across the entire maze state space from Figure 18.3 above. We see that meta-NML learns a model initialization whose parameters can change significantly in a small number of gradient steps. Additionally, most of this change comes from the first gradient step (indicated by the green arrow), which justifies our choice to use only a single gradient step when evaluating meta-NML probabilities for MURAL.

Comparison of Reward Classifiers

In Fig 18.5, we show the comparison between different types of reward. classifiers in the 2D maze navigation problem.

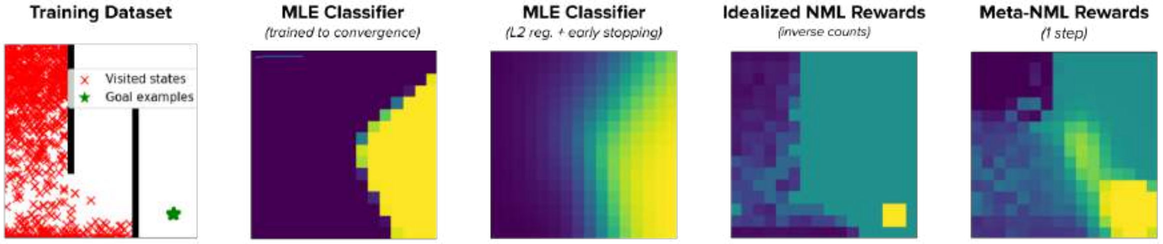


Figure 18.5: A comparison of the rewards given by various classifier training schemes on the 2D Zigzag maze. From left to right: (1) An MLE classifier when trained to convergence reduces to an uninformative sparse reward; (2) An MLE classifier trained with regularization and early stopping has smoother contours, but does not accurately identify the goal; (3) The idealized NML rewards correspond to inverse counts, thus providing a natural exploration objective in the absence of generalization; (4) The meta-NML rewards approximate the idealized rewards well in visited regions, while also benefitting from better shaping towards the goal due to generalization.

Runtime Comparisons

We provide the runtimes for feedforward inference, naive CNML, and meta-NML on each of our evaluation domains. We list both the runtimes for evaluating a single input (Table 18.1), and for completing a full epoch of training during RL (Table 18.2).

These benchmarks were performed on an NVIDIA Titan X Pascal GPU. Per-input runtimes are averaged across 100 samples, and per-epoch runtimes are averaged across 10 epochs.

	Feedforward	**Meta-NML**	**Naive CNML**
Mazes (zigzag, spiral)	0.0004s	0.0090s	15.19s
Sawyer 2D Pusher	0.0004s	0.0092s	20.64s
Sawyer Door	0.0004s	0.0094s	20.68s
Sawyer 3D Pick	0.0005s	0.0089s	20.68s
Ant Locomotion	0.0004s	0.0083s	17.26s
Dexterous Manipulation	0.0004s	0.0081s	17.58s

Table 18.1: Runtimes for evaluating a single input point using feedforward, meta-NML, and naive CNML classifiers. Meta-NML provides anywhere between a 1600x and 2300x speedup compared to naive CNML, which is crucial to making our NML-based reward classifier scheme feasible on RL problems.

	Feedforward	**Meta-NML**	**Naive CNML**
Mazes (zigzag, spiral)	23.50s	39.05s	4hr 13min 34s
Sawyer 2D Pusher	24.91s	43.81	5hr 44min 25s
Sawyer Door	19.77s	38.52s	5hr 45min 00s
Sawyer 3D Pick	20.24s	40.73s	5hr 45min 00s
Ant Locomotion	37.15s	73.72s	4hr 47min 40s
Dexterous Hand Manipulation	48.37s	69.97s	4hr 53min 00s

Table 18.2: Runtimes for completing a single epoch of RL according to Algorithm 2. We collect 1000 samples in the environment with the current policy for each epoch of training. The naive CNML runtimes are extrapolated based on the per-input runtime in the previous table, while the feedforward and meta-NML runtimes are averaged over 10 actual epochs of RL. These times indicate that naive CNML would be computationally infeasible to run in an RL algorithm, whereas meta-NML is able to achieve performance much closer to that of an ordinary feedforward classifier and make learning possible.

Experimental Details

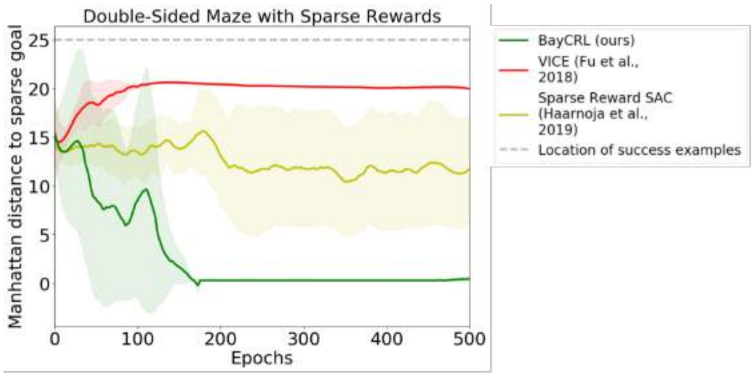


Figure 18.6: Performance of MURAL, VICE, and SAC with sparse rewards on a double-sided maze where some sparse reward states are not provided as goal examples. MURAL is still able to find the sparse rewards, thus receiving higher overall reward, whereas ordinary classifier methods (i.e. VICE) move only towards the provided examples and thus are never able to find the additional rewards. Standard SAC with sparse rewards, also included for comparison, is generally unable to find the goals. The dashed gray line represents the location of the goal examples initially provided to both MURAL and VICE.

Environments

Zigzag Maze and Spiral Maze: These two navigation tasks require moving through long corridors and avoiding several local optima in order to reach the goal. For example, on Spiral Maze, the agent must not get stuck on the other side of the inner wall, even though that position would be close in L2 distance to the desired goal. On these tasks, a sparse reward is not informative enough for learning, while ordinary classifier methods get stuck in local optima due to poor shaping near the goal.

Both of these environments have a continuous state space consisting of the (x, y) coordinates of the agent, ranging from $(-4, -4)$ to $(4, 4)$ inclusive. The action space is the desired velocity in the x and y directions, each ranging from -1 to 1 inclusive.

Sawyer 2D Pusher: This task involves using a Sawyer arm, constrained to move only in the xy plane, to push a randomly initialized puck to a fixed location on a table. The state space consists of the (x, y, z) coordinates of the robot end effector and the (x, y) coordinates of the puck. The action space is the desired x and y velocities of the arm.

Sawyer Door Opening: In this task, the Sawyer arm is attached to a hook, which it must use to open a door to a desired angle of 45 degrees. The door is randomly initialized each time to be at a starting angle of between 0 and 15 degrees. The state space consists of the (x, y, z) coordinates of the end effector and the door angle (in radians); the action space consists of (x, y, z) velocities.

Sawyer 3D Pick and Place: The Sawyer robot must pick up a ball, which is randomly placed somewhere on the table each time, and raise it to a fixed (x, y, z) location high above the table. This represents the biggest exploration challenge out of all the manipulation tasks, as the state space is large and the agent would normally not receive any learning signal unless it happened to pick up the ball and raise it, which is unlikely without careful reward shaping.

The state space consists of the (x, y, z) coordinates of the end effector, the (x, y, z) coordinates of the ball, and the tightness of the gripper (a continuous value between 0 and 1). The robot can control its (x, y, z) arm velocity as well as the gripper value.

Ant Locomotion: In this task, the quadruped ant robot has to navigate from one end of a maze to the other. This represents a high dimensional action space of 8 dimensions, and a high dimensional state space of 15 dimensions as well. The state space consists of the center of mass of the object as well as the positions of the various joints of the ant, and the action space controls the torques on all the joints.

Hand Manipulation: In this task, a 16 DoF robotic hand is mounted on a robot arm and has to reposition an object on a table. The task is challenging due to high dimensionality of the state and action spaces. The state space consists of the arm position, hand joint positions and object positions. In this task, we allow the classifier privileged access to the object position only, but provide the full state space as input to the policy. All the other baseline techniques are provided this same information as well (e.g. the classifier for VICE receives the object position as input).

Ground Truth Distance Metrics

In addition to the success rate plots in Figure 3.5, we provide plots of each algorithm's distance to the goal over time according to environment-specific distance metrics. The distance metrics and success thresholds, which were used to compute the success rates in Figure 3.5, are listed in the table on the next page.

Environment	**Distance Metric Used**	**Success Threshold**
Zigzag Maze	Maze distance to goal	0.5
Spiral Maze	Maze distance to goal	0.5
Sawyer 2D Pusher	Puck L2 distance to goal	0.05
Sawyer Door Opening	Angle difference to goal (radians)	0.035
Sawyer 3D Pick-and-Place	Ball L2 distance to goal	0.06
Ant Locomotion	Maze distance to goal	5
Dexterous Manipulation	Object L2 distance to goal	0.06

Additional Ablations

Learning in a Discrete, Randomized Environment

In practice, many continuous RL environments such as the ones we consider in section 13.7 have state spaces that are correlated at least roughly with the dynamics. For instance, states that are

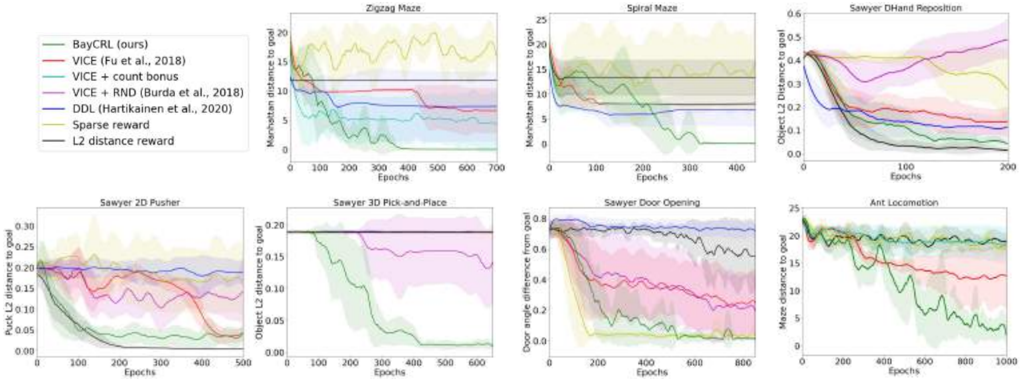


Figure 18.7: Performance of MURAL compared to other algorithms according to ground truth distance metrics. We note that while other algorithms seem to be making progress according to these distances, they are often actually getting stuck in local minima, as indicated by the success rates in Figure 3.5 and the visitation plots in Figure 3.7.

closer together dynamically are also typically closer in the metric space defined by the states. This correlation does not need to be perfect, but as long as it exists, MURAL can in principle learn a smoothly shaped reward towards the goal.

However, even in the case where states are unstructured and completely lack identity, such as in a discrete gridworld environment, the CNML classifier would still reduce to providing an exploration-centric reward bonus, as indicated by Theorem 1, ensuring reasonable worst-case performance.

To demonstrate this, we evaluate MURAL on a variant of the Zigzag Maze task where states are first discretized to a 16×16 grid, then "shuffled" so that the xy representation of a state does not correspond to its true coordinates and the states are not correlated dynamically. MURAL manages to solve the task, while a standard classifier method (VICE) does not. Still, MURAL is more effective in the original state space where generalization is possible, suggesting that both the exploration and reward shaping abilities of the CNML classifier are crucial to its overall performance.

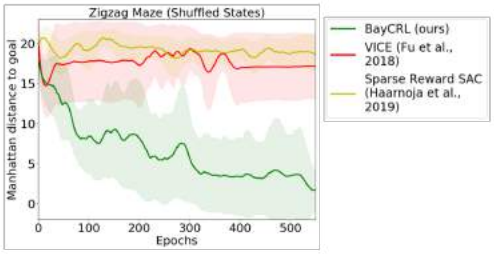


Figure 18.8: Comparison of MURAL, VICE, and SAC with sparse rewards on a discrete, randomized variant of the Zigzag Maze task. MURAL is still able to solve the task on a majority of runs due to its connection to a count-based exploration bonus, whereas ordinary classifier methods (i.e. VICE) experience significantly degraded performance in the absence of any generalization across states.

Finding "Hidden" Rewards Not Indicated by Success Examples

The intended setup for MURAL (and classifier-based RL algorithms in general) is to provide a set of success examples to learn from, thus removing the need for a manually specified reward function.

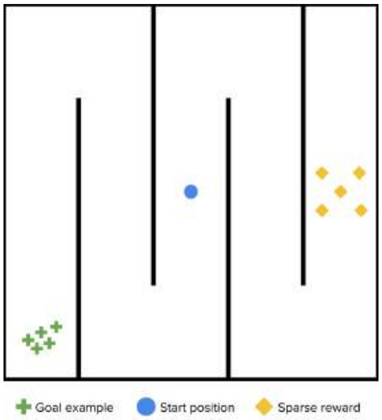


Figure 18.9: Visualization of the Double-Sided Maze environment. Only the goal examples in the bottom left corner are provided to the algorithm.

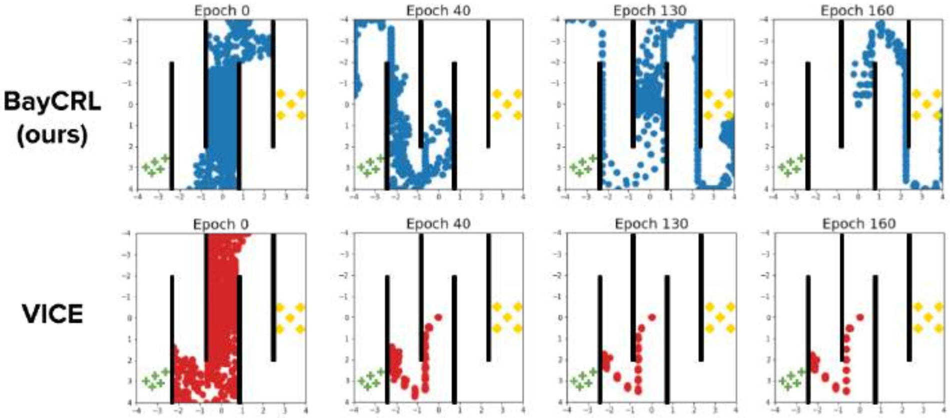


Figure 18.10: Plot of visitations for MURAL vs. VICE on the double-sided maze task. MURAL is initially guided towards the provided goals in the bottom left corner as expected, but continues to explore in both directions, thus allowing it to find the hidden sparse rewards as well. Once this happens, it focuses on the right side of the maze instead because those rewards are easier to reach. In contrast, VICE moves only towards the (incomplete) set of provided goals on the left, ignoring the right half of the maze entirely and quickly getting stuck in a local optima.

However, here we instead consider the case where a ground truth reward function exists which we do not fully know, and can only query through interaction with the environment. In this case, because the human expert has limited knowledge, the provided success examples may not cover all regions of the state space with high reward.

An additional advantage of MURAL is that it is still capable of finding these "unspecified" goals because of its built-in exploration behavior, whereas other classifier methods would operate solely based on the goal examples provided. To see this, we evaluate our algorithm on a two-sided variant of the Zigzag Maze with multiple goals, visualized in Figure 18.9 to the right. The agent starts in the middle and is provided with 5 goal examples on the far left side of the maze; unknown to it, the right side contains 5 sparse reward regions which are actually closer from its initial position.

As shown in Figures 18.6 and 18.10, MURAL manages to find the sparse rewards while other methods do not. MURAL, although initially guided towards the provided goal examples on the left, continues to explore in both directions and eventually finds the "hidden" rewards on the right. Meanwhile, VICE focuses solely on the provided goals, and gets stuck in a local optima near the bottom left corner.

Hyperparameter and Implementation Details

We describe the hyperparameter choices and implementation details for our experiments here. We first list the general hyperparameters that were shared across runs, then provide tables of additional hyperparameters we tuned over for each domain and algorithm.

Goal Examples: For the classifier-based methods in our experiments (VICE and MURAL), we provide 150 goal examples for each environment at the start of training. These are used as the pool of positive examples when training the success classifier.

DDL Reward: We use the version of DDL proposed in [141] where we provide the algorithm with the ground truth goal state $\mathbf{g}$, then run SAC with a reward function of $r(\mathbf{s}) = -d^{\pi}(\mathbf{s}, \mathbf{g})$, where d^{π} is the learned dynamical distance function.

General Hyperparameters

SAC	
Learning Rate	3×10^{-4}
Discount Factor γ	0.99
Policy Type	Gaussian
Policy Hidden Sizes	$(512, 512)$
Policy Hidden Activation	ReLU
RL Batch Size	1024
Reward Scaling	1
Replay Buffer Size	$500,000$
Q Hidden Sizes	$(512, 512)$
Q Hidden Activation	ReLU
Q Learning Rate	3×10^{-4}
Target Network τ	5×10^{-3}
MURAL	
Adaptation batch size	64
Meta-training tasks per epoch	128
Meta-test set size	2048
VICE	
Classifier Learning Rate	1×10^{-4}
Classifier Batch Size	128
RND	
Hidden Layer Sizes	$(256, 256)$
Output Units	512

Table 18.3: General hyperparameters used across all domains.

Zigzag Maze Hyperparameters

MURAL	
Classifier Hidden Layers	[(512, 512), **(2048, 2048)**]
λ_{dist}	$[\mathbf{0.5}, 1]$
k_{query}	**1**
VICE	
n_{VICE}	$[1, \mathbf{2}, 10]$
Mixup α	$[0, \mathbf{1}]$
Weight Decay λ	$[0, \mathbf{5 \times 10^{-3}}]$
VICE+Count Bonus	
n_{VICE}	$[1, \mathbf{2}, 10]$
Mixup α	$[0, \mathbf{1}]$
Classifier reward scale	$[\mathbf{0.25}, 0.5, 1]$
Weight Decay λ	$[\mathbf{0}, 5 \times 10^{-3}]$
DDL	
N_d	$[\mathbf{2}, 4]$
Training frequency (every n steps)	$[16, \mathbf{64}]$

Table 18.4: Hyperparameters we tuned for the Zigzag Maze task. Bolded values are what we use for the final runs in Section 6.

Spiral Maze Hyperparameters

MURAL	
Classifier Hidden Layers	[(512, 512), **(2048, 2048)**]
λ_{dist}	$[\mathbf{0.5}, 1]$
k_{query}	**1**
VICE	
n_{VICE}	$[1, \mathbf{2}, 10]$
Mixup α	$[0, \mathbf{1}]$
Weight Decay λ	$[0, \mathbf{5 \times 10^{-3}}]$
VICE+Count Bonus	
n_{VICE}	$[1, \mathbf{2}, 10]$
Mixup α	$[0, \mathbf{1}]$
Classifier reward scale	$[\mathbf{0.25}, 0.5, 1]$
Weight Decay λ	$[\mathbf{0}, 5 \times 10^{-3}]$
DDL	
N_d	$[\mathbf{2}, 4]$
Training frequency (every n steps)	$[16, \mathbf{64}]$

Table 18.5: Hyperparameters we tuned for the Spiral Maze task. Bolded values are what we use for the final runs in Section 6.

Ant Locomotion Hyperparameters

MURAL	
Classifier Hidden Layers	[(512, 512), **(2048, 2048)**]
λ_{dist}	$[0.5, \mathbf{1}, 1.5, 2]$
k_{query}	**1**
VICE	
n_{VICE}	$[1, \mathbf{2}, 10]$
Mixup α	$[0, \mathbf{1}]$
Weight Decay λ	$[0, \mathbf{5 \times 10^{-3}}]$
VICE+Count Bonus	
n_{VICE}	$[1, \mathbf{2}, 10]$
Mixup α	$[0, \mathbf{1}]$
Classifier reward scale	[**0.25**, $0.5, 1$]
Weight Decay λ	$\mathbf{5 \times 10^{-3}}$
DDL	
N_d	$[2, \mathbf{4}]$
Training frequency (every n steps)	$[\mathbf{16}, 64]$

Table 18.6: Hyperparameters we tuned for the Ant Locomotion task. Bolded values are what we use for the final runs in Section 6.

Sawyer Push Hyperparameters

MURAL	
Classifier Hidden Layers	[(512, 512), **(2048, 2048)**]
λ_{dist}	$[0.2, \mathbf{0.6}, 1]$
k_{query}	**1**
VICE	
n_{VICE}	$[1, 2, \mathbf{10}]$
Mixup α	$[0, \mathbf{1}]$
Weight Decay λ	$[\mathbf{0}, 5 \times 10^{-3}]$
VICE + RND	
n_{VICE}	$[1, 2, \mathbf{10}]$
Mixup α	$[0, \mathbf{1}]$
RND reward scale	$[\mathbf{1}, 5, 10]$
DDL	
N_d	$[\mathbf{4}, 10]$
Training frequency (every n steps)	$[\mathbf{16}, 64]$

Table 18.7: Hyperparameters we tuned for the Sawyer Push task. Bolded values are what we use for the final runs in Section 6.

Sawyer Pick-and-Place Hyperparameters

MURAL	
Classifier Hidden Layers	[**(512, 512)**, (2048, 2048)]
λ_{dist}	$[0.2, \mathbf{0.6}, 1]$
k_{query}	**1**
VICE	
n_{VICE}	$[1, \mathbf{2}, 10]$
Mixup α	$[0, \mathbf{1}]$
Weight Decay λ	$[\mathbf{0}, 5 \times 10^{-3}]$
VICE + RND	
n_{VICE}	$[1, \mathbf{2}, 10]$
Mixup α	$[0, \mathbf{1}]$
RND reward scale	$[\mathbf{1}, 5, 10]$
DDL	
N_d	$[4, \mathbf{10}]$
Training frequency (every n steps)	$[\mathbf{16}, 64]$

Table 18.8: Hyperparameters we tuned for the Sawyer Pick-and-Place task. Bolded values are what we use for the final runs in Section 6.

Sawyer Door Opening Hyperparameters

MURAL	
Classifier Hidden Layers	[**(512, 512)**, (2048, 2048)]
λ_{dist}	$[0.05, 0.1, \mathbf{0.25}]$
k_{query}	$[1, \mathbf{2}]$
VICE	
n_{VICE}	$[1, \mathbf{5}, 10]$
Mixup α	$[\mathbf{0}, 1]$
Weight Decay λ	$[\mathbf{0}, 5 \times 10^{-3}]$
VICE + RND	
n_{VICE}	$[1, \mathbf{5}, 10]$
Mixup α	$[\mathbf{0}, 1]$
RND reward scale	$[1, \mathbf{5}, 10]$
DDL	
N_d	$[4, \mathbf{10}]$
Training frequency (every n steps)	$[\mathbf{16}, 64]$

Table 18.9: Hyperparameters we tuned for the Sawyer Door Opening task. Bolded values are what we use for the final runs in Section 6.

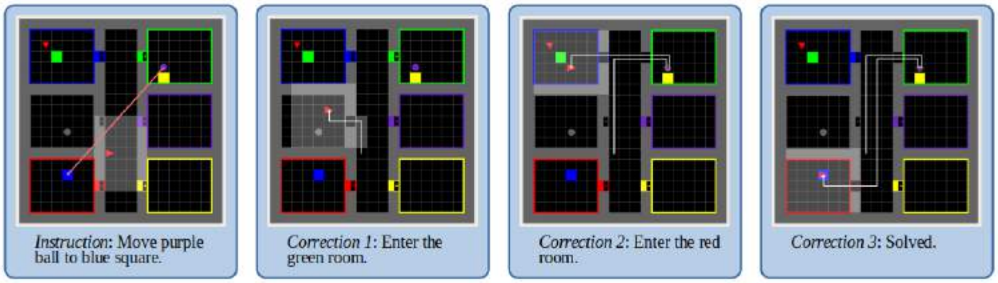
Dexterous Hand Repositioning Hyperparameters

MURAL	
Classifier Hidden Layers	[(512, 512), **(2048, 2048)**]
λ_{dist}	$[0.2, \mathbf{0.5}, 1]$
k_{query}	**1**
VICE	
n_{VICE}	$[1, 2, \mathbf{10}]$
Mixup α	$[\mathbf{0}, 1]$
Weight Decay λ	$[0, \mathbf{5 \times 10^{-3}}]$
VICE + RND	
n_{VICE}	$[1, \mathbf{2}, 10]$
Mixup α	$[\mathbf{0}, 1]$
RND reward scale	$[\mathbf{1}, 5, 10]$
DDL	
N_d	$[\mathbf{4}, 10]$
Training frequency (every n steps)	$[\mathbf{16}, 64]$

Table 18.10: Hyperparameters we tuned for the Dexterous Hand Repositioning task. Bolded values are what we use for the final runs in Section 6.

18.2 Appendix B: Appendix for Chapter 4

Visualizing Behavior: Multi-Room Object Manipulation Environment



Instruction: Move purple ball to blue square.

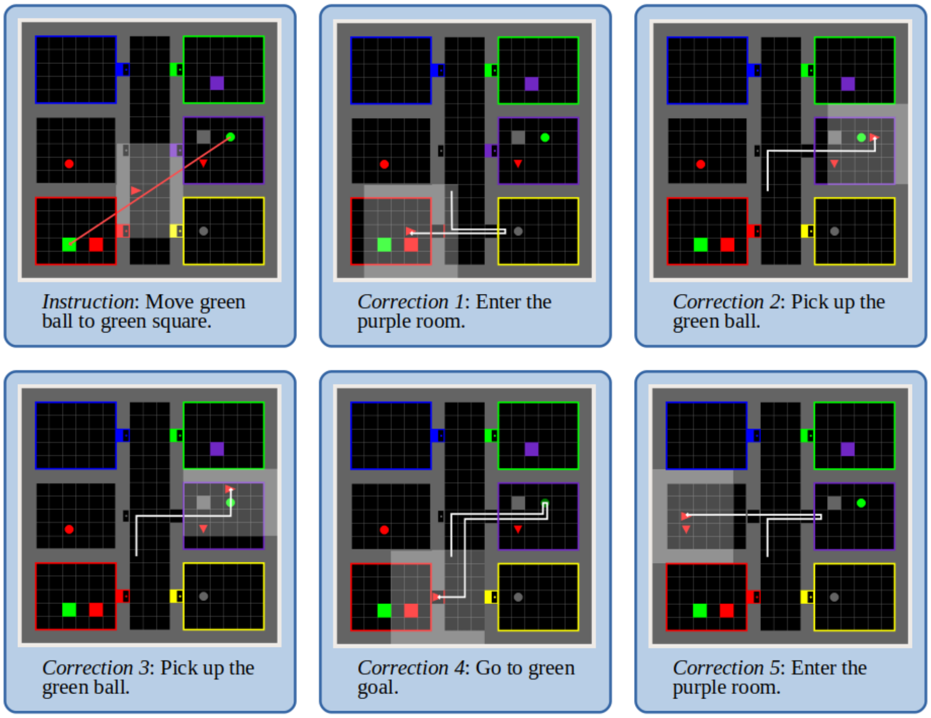
Correction 1: Enter the green room.

Correction 2: Enter the red room.

Correction 3: Solved.

Figure 18.11: Example task with corrections. Instruction: The agent receives the initial instruction. Correction 1: The agent mistakenly goes into the gray door, so it receives the correction to enter the green room, where the purple ball is located. Correction 2: The agent successfully picks up the ball, but then mistakenly enters the blue room, so it receives the correction to enter the red room, where the goal is located. Correction 3: The agent brings the object to the goal and solves the task.

An example task with corrections in show in Figure 18.11.



Instruction: Move green ball to green square.

Correction 1: Enter the purple room.

Correction 2: Pick up the green ball.

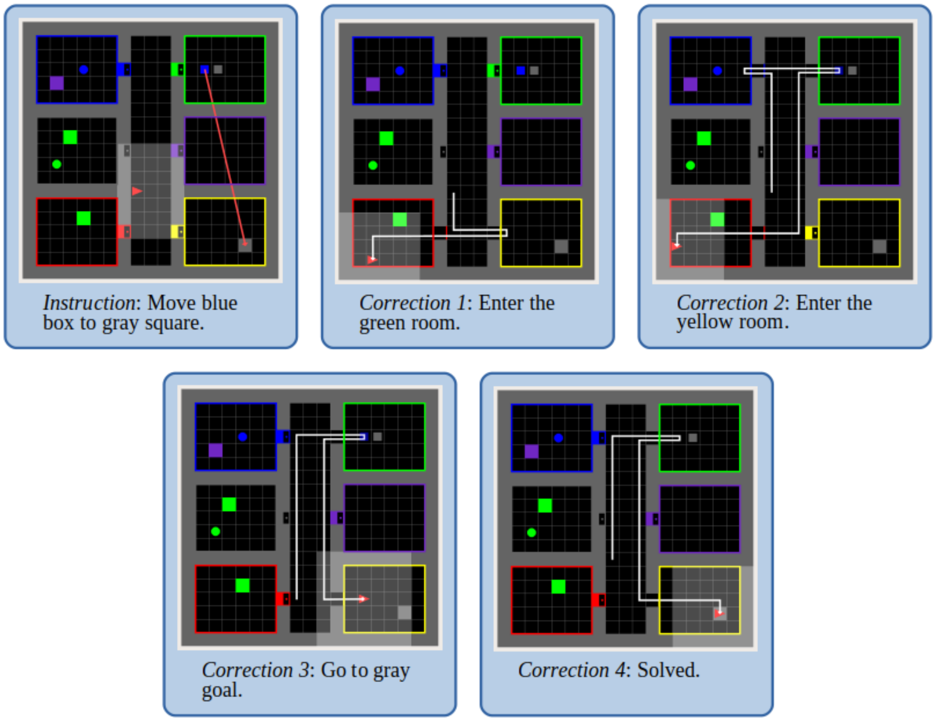
Correction 3: Pick up the green ball.

Correction 4: Go to green goal.

Correction 5: Enter the purple room.

Figure 18.12: Failure example. The orange arrow shows the task, the white arrows show the net trajectory.

It is possible to visualize failure cases, which illuminate the behavior of the algorithm on challenging tasks. In the failure case in Figure 18.12, we note that the agent is able to successfully enter the purple room, pickup the green ball, and exit. However, after it receives the fourth correction telling it to go to the green goal, it forgets to pick up the green ball. This behavior can likely be improved by varying corrections more at training time, and providing different corrections if an agent is unable to comprehend the first one.



Instruction: Move blue box to gray square.

Correction 1: Enter the green room.

Correction 2: Enter the yellow room.

Correction 3: Go to gray goal.

Correction 4: Solved.

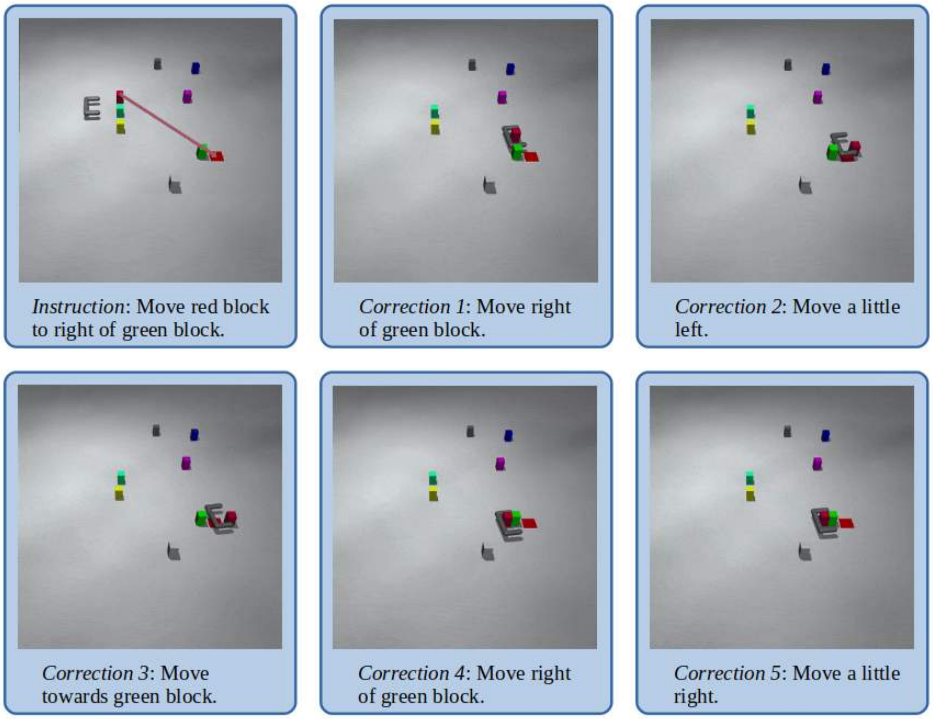
Figure 18.13: Success example. The orange arrow shows the task.

Additionally, we present a success case in Figure 18.13, where the agent successfully learns to solve the task through iterative corrections, making further progress in each frame.

Visualizing Behavior: Robotic Object Relocation Environment

Similarly, we can again visualize failure cases for the robotic object relocation environment. In the failure case in Figure 18.14, the agent pushes the correct object, and very nearly solves the task after the first and second corrections. However, after it receives the third and fourth corrections telling it to go near the green block, its trajectory changes such that its path to the goal is blocked by the green block itself. This case is particularly challenging, as the combination of the bulky agent body, angle of approach, and the proximity of the goal location to the obstacle near it make the goal location inaccessible. Were the goal location a little further from the green block, or the agent of a form lending itself to nimbler motion, this task may have been solved by the second correction.

We also present a success case in Figure 18.15, where the agent starts off pushing the wrong block, but successfully learns to solve the task through iterative corrections, making further progress in each frame.



Instruction: Move red block to right of green block.

Correction 1: Move right of green block.

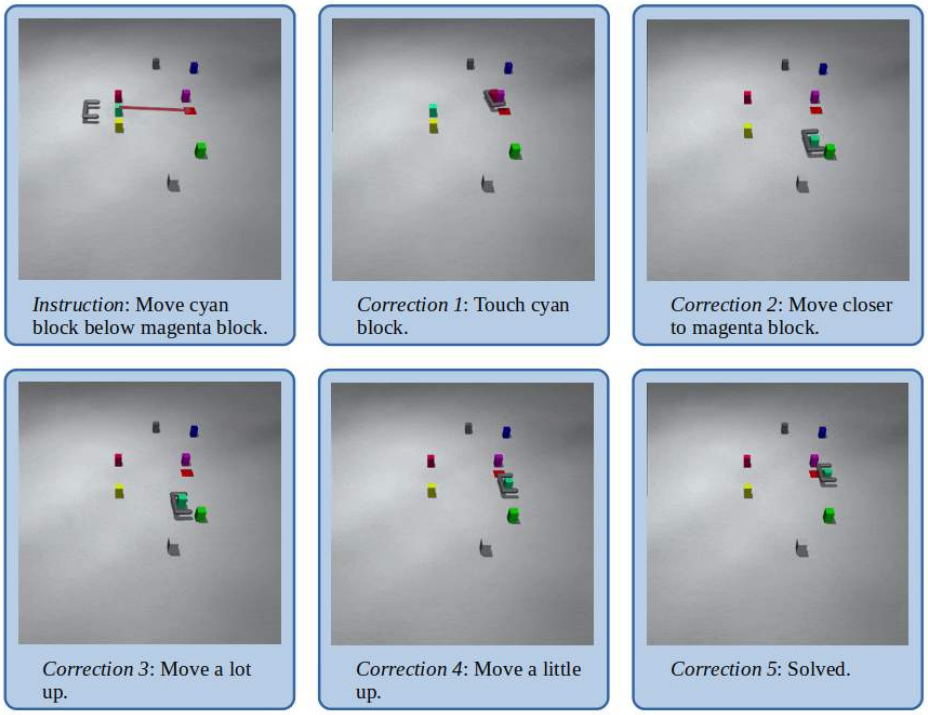
Correction 2: Move a little left.

Correction 3: Move towards green block.

Correction 4: Move right of green block.

Correction 5: Move a little right.

Figure 18.14: Failure example. The orange arrow shows the task, the white arrows show the net trajectory.



Instruction: Move cyan block below magenta block.

Correction 1: Touch cyan block.

Correction 2: Move closer to magenta block.

Correction 3: Move a lot up.

Correction 4: Move a little up.

Correction 5: Solved.

Figure 18.15: Success example. The orange arrow shows the task.

Training and Architecture Details

We detail the training and architecture details for each environment. We use Adam for optimization with a learning rate of 0.001. The data buffer size is $1e6$. We train on the whole data buffer for five epochs each time before adding more data to the buffer. Unless otherwise stated, we use ReLU activations. MLP(32, 32) specifies a multilayer-perceptron with 2 layers each of size 32. CNN((4, 2x2, 1), (4, 2x2, 1)) specifies a 2 layer convolutional neural network where each layer has 4 filters, 2x2 kernels, and 1 stride.

To train the expert policies we use Proximal Policy Optimization with a dense reward function. For the multi-room domain we keep track of the next subgoal the agent has to complete. The reward is $-0.01*$ Euclidian distance to the next subgoal where a grid is a distance of 1. If the agent completes a previously uncompleted subgoal it gets a onetime reward of 100. For the robotic object manipulation domain the reward is -(Euclidian distance of the gripper to the block + $5*$ Euclidian distance of the block to the target location).

For the multi-room domain we meta-train on 1700 environments. Our method converges in 6 DAgger steps so it takes 30 corrections per environment for a total of 51,000 corrections. For the robotic object relocation domain, we train on 750 environments. Our method converges in 9 DAgger steps so it takes 45 corrections per environment for a total of 33,750 corrections.

Multi-room object manipulation

The observation is a 7x7x4 grid centered on the agent. The 1st channel encode object types (empty, wall, closed door, locked door, triangle, square, circle, goal). The 2nd channel encodes colors (none, blue, green, gray, purple, red, yellow). The 3rd and 4th channels also encode object types and colors respectively but only for objects the agent is currently holding. The observation also includes a binary indicatory if the agent is currently holding an object. The action space is of size 6 and consists of (move up, move left, move right, move down, pickup object, drop object). The length of a trajectory is 100.

Correction Module Each previous trajectory is first subsampled every 25th state. Each observation is then processed by a 2D CNN((4, 2x2, 1), (4, 2x2, 1)) followed by a MLP(32, 32). This trajectory of state embeddings is then processed by a 1D CNN(4, 2, 1) followed by a MLP(16, 4).

The language correction is converted into word embeddings of size 16 and processed by a 1D CNN(4, 2, 1) followed by a MLP (16, 4). For a given correction iteration, the trajectory embedding and the correction embedding are then concatenated and fed through a MLP(32 ,32). These tensors are then averaged across the number of correction iterations.

Instruction Module The instruction is converted into word embeddings (the same embeddings as the correction) of size 16 and processed by a 1D CNN(4, 2, 1) followed by a MLP(16, 32).

Policy Module The observation is processed by a 2D CNN((8, 2x2, 1), (8, 2x2, 1)) followed by a MLP(32, 32). This state embedding, the correction module tensor, and the instruction module tensor are concatenated and fed through a MLP (64, 64) to output an action distribution.

Robotic object relocation

The observation is of size 19 and includes the (x, y, orientation) of the gripper and the (x, y) coordinates of the 3 movable blocks and 5 fixed blocks. The action space is of size 4 and consists of applying force on the gripper in the cardinal directions. The length of a trajectory is 350.

Correction Module Each previous trajectory is first subsampled every 70th state. The trajectory is then processed by a 1D CNN(8, 2, 1) followed by a MLP(16, 8).

The language correction is converted into word embeddings of size 16 and processed by a 1D CNN(4, 2, 1) followed by a MLP(16, 16). For a given correction iteration, the trajectory embedding and the correction embedding are then concatenated and fed through a MLP(32 ,32). These tensors are then averaged across the number of correction iterations.

Instruction Module The instruction is converted into word embeddings (the same embeddings as the correction) of size 16 and processed by a 1D CNN(4, 2, 1) followed by a MLP(16, 16).

Policy Module The observation, the correction module tensor, and the instruction module tensor are concatenated and fed through a MLP(256, 256, 256) to output an action distribution.

18.3 Appendix C: Appendix for Chapter 5

Pseudo-Reward

The $\log p(z)$ term in Equation 5.3 is a baseline that does not depend on the policy parameters θ, so one might be tempted to remove it from the objective. We provide a two justifications for keeping it. First, assume that episodes never terminate, but all skills eventually converge to some absorbing state (e.g., with all sensors broken). At this state, the discriminator cannot distinguish the skills, so its estimate is $\log q(z \mid s) = \log(1/N)$, where N is the number of skills. For practical reasons, we want to restart the episode after the agent reaches the absorbing state. Subtracting $\log(z)$ from the pseudo-reward at every time step in our finite length episodes is equivalent to pretending that episodes never terminate and the agent gets reward $\log(z)$ after our "artificial" termination. Second, assuming our discriminator q_ϕ is better than chance, we see that $q_\phi(z \mid s) \geq p(z)$. Thus, subtracting the $\log p(z)$ baseline ensures our reward function is always non-negative, encouraging the agent to stay alive. Without this baseline, an optimal agent would end the episode as soon as possible.[1]

Optimum for Gridworlds

For simple environments, we can compute an analytic solution to the DIAYN objective. For example, consider a $N \times N$ gridworld, where actions are to move up/down/left/right. Any action can be taken in any state, but the agent will stay in place if it attempts to move out of the gridworld. We use (x, y) to refer to states, where $x, y \in \{1, 2, \cdots, N\}$.

For simplicity, we assume that, for every skill, the distribution of states visited exactly equals that skill's stationary distribution over states. To clarify, we will use π_z to refer to the policy for skill z. We use ρ_{π_z} to indicate skill z's stationary distribution over states, and $\hat{\rho}_{\pi_z}$ as the empirical distribution over states within a single episode. Our assumption is equivalent to saying

$$\rho_{\pi_z}(s) = \hat{\rho}_{\pi_z}(s) \qquad \forall s \in \mathcal{S}$$

One way to ensure this is to assume infinite-length episodes.

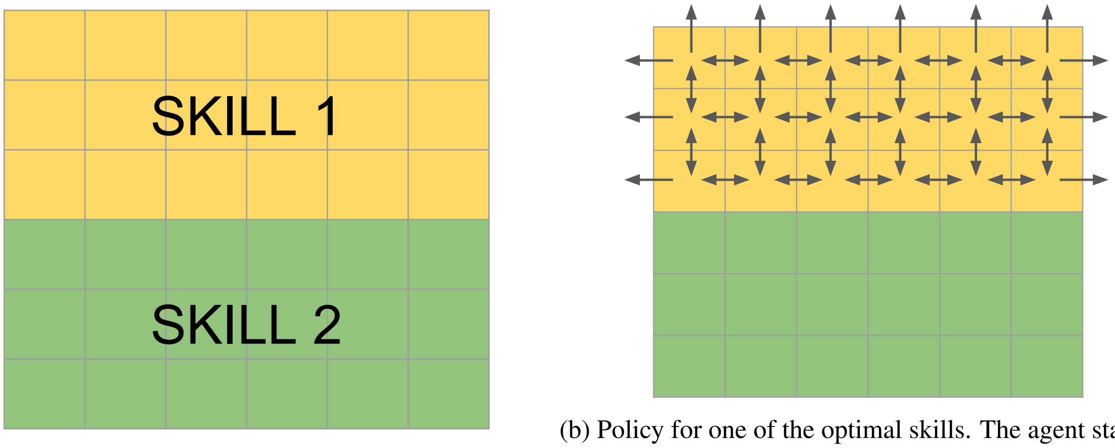
We want to show that a set of skills that evenly partitions the state space is the optimum of the DIAYN objective for this task. While we will show this only for the 2-skill case, the 4 skill case is analogous.

The optimum policies for a set of two skills are those which evenly partition the state space. We will show that a top/bottom partition is one such (global) optima. The left/right case is analogous.

Lemma 6. A pair of skills with state distributions given below (and shown in Figure 18.16) are an optimum for the DIAYN objective with no entropy regularization ($\alpha = 0$).

$$\rho_{\pi_1}(x, y) = \frac{2}{N^2}\delta(y \leq N/2) \quad \text{and} \quad \rho_{\pi_2}(x, y) = \frac{2}{N^2}\delta(y > N/2) \tag{18.5}$$

[1] In some environments, such as mountain car, it is desirable for the agent to end the episode as quickly as possible. For these types of environments, the $\log p(z)$ baseline can be removed.



(a) Optimum Skills for Gridworld with 2 Skills

(b) Policy for one of the optimal skills. The agent stays in place when it attempts to leave the gridworld.

Figure 18.16: **Optimum for Gridworlds:** For gridworld environments, we can compute an analytic solution to the DIAYN objective.

Before proving Lemma 6, we note that there exist policies that achieve these stationary distributions. Figure 18.16b shows one such policy, were each arrow indicates a transition with probability $\frac{1}{4}$. Note that when the agent is in the bottom row of yellow states, it does not transition to the green states, and instead stays in place with probability $\frac{1}{4}$. Note that the distribution in Equation 18.5 satisfies the detailed balance equations [429].

Proof. Recall that the DIAYN objective with no entropy regularization is:

$$-\mathcal{H}[Z \mid S] + \mathcal{H}[Z]$$

Because the skills partition the states, we can always infer the skill from the state, so $\mathcal{H}[Z \mid S] = 0$. By construction, the prior distribution over $\mathcal{H}[Z]$ is uniform, so $\mathcal{H}[Z] = \log(2)$ is maximized. Thus, a set of two skills that partition the state space maximizes the un-regularized DIAYN objective. □

Next, we consider the regularized objective. In this case, we will show that while an even partition is not perfectly optimal, it is "close" to optimal, and its "distance" from optimal goes to zero as the gridworld grows in size. This analysis will give us additional insight into the skills preferred by the DIAYN objective.

Lemma 7. A pair of skills with state distributions given given in Equation 18.5 achieve an DIAYN objective within a factor of $O(1/N)$ of the optimum, where N is the gridworld size.

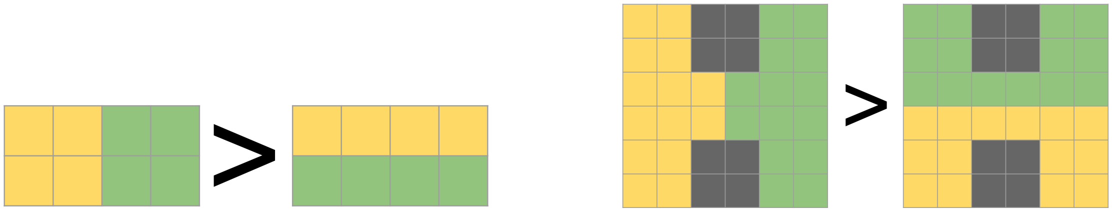
Proof. Recall that the DIAYN objective with no entropy regularization is:

$$\mathcal{H}[A \mid S, Z] - \mathcal{H}[Z \mid S] + \mathcal{H}[Z]$$

We have already computed the second two terms in the previous proof: $\mathcal{H}[Z \mid S] = 0$ and $\mathcal{H}[Z] = \log(2)$. For computing the first term, it is helpful to define the set of "border states" for a particular skill as those that do not neighbor another skill. For skill 1 defined in Figure 18.16 (colored yellow), the border states are: $\{(x, y) \mid y = 4\}$. Now, computing the first term is straightforward:

$$\begin{aligned}\mathcal{H}[A \mid S, Z] &= \frac{2}{N^2}\left(\underbrace{(N/2-1)N}_{\text{non-border states}}\log(4) + \underbrace{N}_{\text{border states}}\frac{3}{4}\log(4)\right) \\ &= \frac{2\log(4)}{N^2}\left(\frac{1}{2}N^2 - \frac{1}{4}N\right) \\ &= \log(4)(1 - \frac{1}{2N})\end{aligned}$$

Thus, the overall objective is within $\frac{\log(4)}{2N}$ of optimum. □



(a) (b)

Figure 18.17: The DIAYN objective prefers skills that *(Left)* partition states into sets with short borders and *(Right)* which correspond to bottleneck states.

Note that the term for maximum entropy over actions ($\mathcal{H}[A \mid S, Z]$) comes into conflict with the term for discriminability ($-\mathcal{H}[Z \mid S]$) at states along the border between two skills. Everything else being equal, this conflict encourages DIAYN to produce skills that have small borders, as shown in Figure 18.17. For example, in a gridworld with dimensions $N < M$, a pair of skills that split along the first dimension (producing partitions of size $(N, M/2)$) would achieve a larger (better) objective than skills that split along the second dimension. This same intuition that DIAYN seeks to minimize the border length between skills results in DIAYN preferring partitions that correspond to bottleneck states (see Figure 18.17b).

Experimental Details

In our experiments, we use the same hyperparameters as those in [23], with one notable exception. For the Q function, value function, and policy, we use neural networks with 300 hidden units instead of 128 units. We found that increasing the model capacity was necessary to learn many diverse skills. When comparing the "skill initialization" to the "random initialization" in Section 5.4, we use the same model architecture for both methods. To pass skill z to the Q function, value function, and policy, we simply concatenate z to the current state s_t. As in [23], epochs are 1000 episodes long. For all environments, episodes are at most 1000 steps long, but may be shorter. For example, the standard benchmark hopper environment terminates the episode once it falls over. Figures 5.2 shows up to 1000 epochs, which corresponds to at most 1 million steps. We found that learning was most stable when we scaled the maximum entropy objective ($\mathcal{H}[A \mid S, Z]$ in Eq. 5.1) by $\alpha = 0.1$. We use this scaling for all experiments.

Environments

Most of our experiments used the following, standard RL environments [29]: HalfCheetah-v1, Ant-v1, Hopper-v1, MountainCarContinuous-v0, and InvertedPendulum-v1. The simple 2D navigation task used in Figures 5.2a and 5.5 was constructed as follows. The agent starts in the center of the unit box. Observations $s \in [0, 1]^2$ are the agent's position. Actions $a \in [-0.1, 0.1]^2$ directly change the agent's position. If the agent takes an action to leave the box, it is projected to the closest point inside the box.

The cheetah hurdle environment is a modification of HalfCheetah-v1, where we added boxes with shape $H = 0.25m, W = 0.1m, D = 1.0m$, where the width dimension is along the same axis as the cheetah's forward movement. We placed the boxes ever 3 meters, start at $x = -1m$.

The ant navigation environment is a modification of Ant-v1. To improve stability, we follow [240] and lower the gear ratio of all joints to 30. The goals are the corners of a square, centered at the origin, with side length of 4 meters: $[(2, 2), (2, -2), (-2, -2), (-2, 2), (2, 2)]$. The ant starts at the origin, and receives a reward of +1 when its center of mass is within 0.5 meters of the correct next goal. Each reward can only be received once, so the maximum possible reward is +5.

Hierarchical RL Experiment

For the 2D navigation experiment shown in Figure 5.5, we first learned a set of skills on the point environment. Next, we introduced a reward function $r_g(s) = -\|s - g\|_2^2$ penalizing the distance from the agent's state to some goal, and applied the hierarchical algorithm above. In this task, the DIAYN skills provided sufficient coverage of the state space that the hierarchical policy only needed to take a single action (i.e., choose a single skill) to complete the task.

More Analysis of DIAYN Skills

Training Objectives

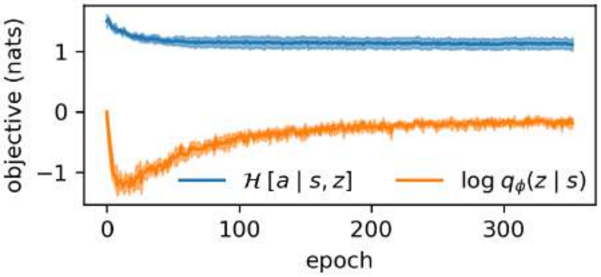


Figure 18.18: **Objectives**: We plot the two terms from our objective (Eq. 1) throughout training. While the entropy regularizer (blue) quickly plateaus, the discriminability term (orange) term continues to increase, indicating that our skills become increasingly diverse without collapsing to deterministic policies. This plot shows the mean and standard deviation across 5 seeds for learning 20 skills in half cheetah environment. Note that $\log_2(1/20) \approx -3$, setting a lower bound for $\log q_\phi(z \mid s)$.

To provide further intuition into our approach, Figure 18.18 plots the two terms in our objective throughout training. Our skills become increasingly diverse throughout training without converging to deterministic policies.

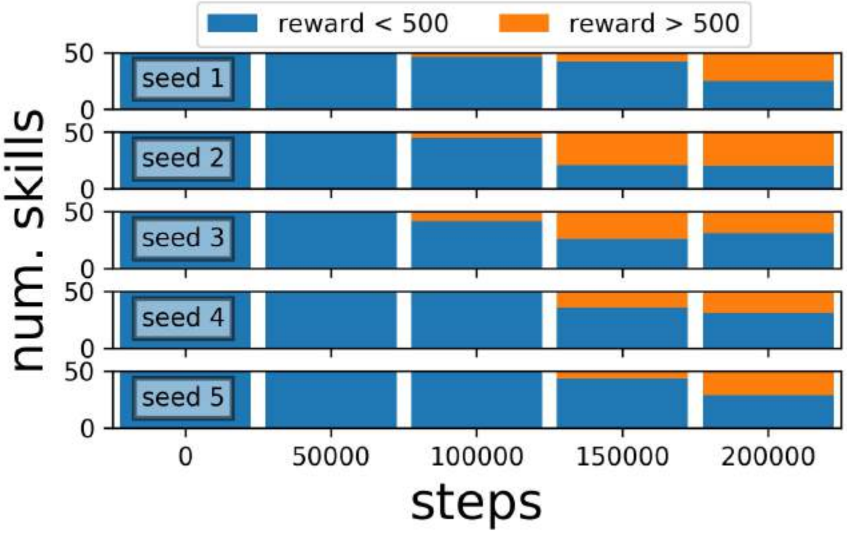
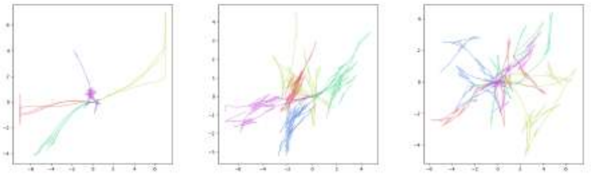


Figure 18.19: We repeated the experiment from Figure 5.2 with 5 random seeds to illustrate the robustness of our method to random seed.

To illustrate the stability of DIAYN to random seed, we repeated the experiment in Figure 5.2 for 5 random seeds. Figure 18.19 illustrates that the random seed has little effect on the training dynamics.

Effect of Entropy Regularization

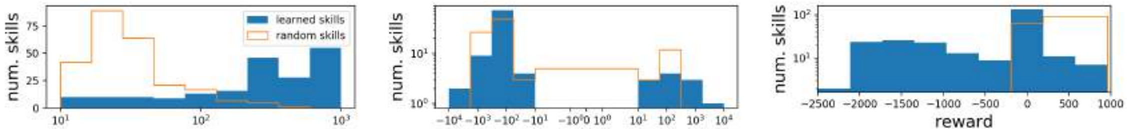
Question 8. *Does entropy regularization lead to more diverse skills?*



$\alpha = 0.01$ $\alpha = 1$ $\alpha = 10$

To answer this question, we apply our method to a 2D point mass. The agent controls the orientation and forward velocity of the point, with is confined within a 2D box. We vary the entropy regularization α, with larger values of α corresponding to policies with more stochastic actions. With small α, we learn skills that move large distances in different directions but fail to explore large parts of the state space. Increasing α makes the skills visit a more diverse set of states, which may help with exploration in complex state spaces. It is difficult to discriminate skills when α is further increased.

Distribution over Task Reward



(a) Hopper (b) Half Cheetah (c) Ant

Figure 18.21: **Task reward of skills learned without reward**: While our skills are learned without the task reward function, we evaluate each with the task reward function for analysis. The wide range of rewards shows the diversity of the learned skills. In the hopper and half cheetah tasks, many skills achieve large task reward, despite not observing the task reward during training. As discussed in prior work [247], [301], standard model-free algorithms trained directly on the task reward converge to scores of 1000 - 3000 on hopper, 1000 - 5000 on cheetah, and 700 - 2000 on ant.

In Figure 18.21, we take the skills learned without any rewards, and evaluate each of them on the standard benchmark reward function. We compare to random (untrained) skills. The wide distribution over rewards is evidence that the skills learned are diverse. For hopper, some skills hop or stand for the entire episode, receiving a reward of at least 1000. Other skills aggressively hop forwards or dive backwards, and receive rewards between 100 and 1000. Other skills fall over immediately and receive rewards of less than 100. The benchmark half cheetah reward includes a control penalty for taking actions. Unlike random skills, learned skills rarely have task reward near zero, indicating that all take actions to become distinguishable. Skills that run in place, flop on their nose, or do backflips receive reward of -100. Skills that receive substantially smaller reward correspond to running quickly backwards, while skills that receive substantially larger reward correspond to running forward. Similarly, the benchmark ant task reward includes both a control penalty and a survival bonus, so random skills that do nothing receive a task reward near 1000. While no single learned skill learns to run directly forward and obtain a task reward greater than 1000, our learned skills run in different patterns to become discriminable, resulting in a lower task reward.

Exploration

Question 9. *Does DIAYN explore effectively in complex environments?*

We apply DIAYN to three standard RL benchmark environments: half-cheetah, hopper, and ant. In all environments, we learn diverse locomotion primitives, as shown in Figure 5.3. Despite never receiving any reward, the half cheetah and hopper learn skills that move forward and achieve large task reward on the corresponding RL benchmarks, which all require them to move forward at a fast pace. Half cheetah and hopper also learn skills that move backwards, corresponding to receiving a task reward much smaller than what a random policy would receive. Unlike hopper and half cheetah, the ant is free to move in the XY plane. While it learns skills that move in different directions, most skills move in arcs rather than straight lines, meaning that we rarely learn a single skill that achieves large task reward on the typical task of running forward. In the appendix, we visualize the objective throughout training.

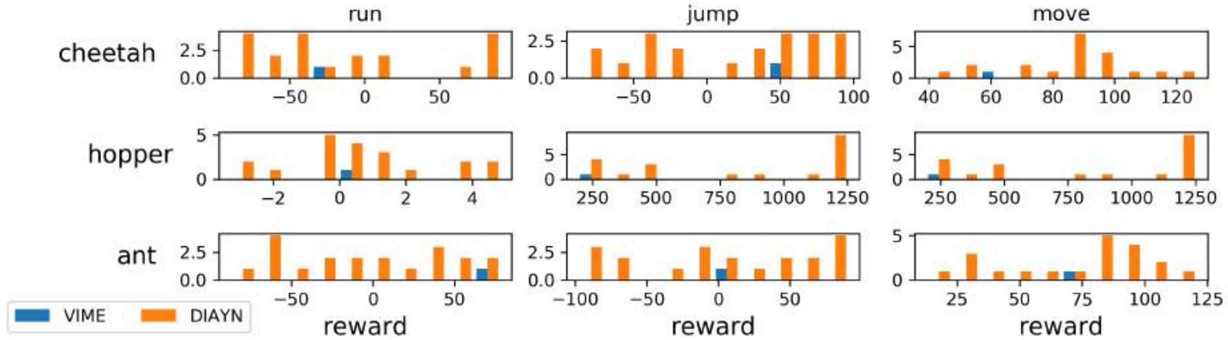


Figure 18.22: **Exploration**: We take DIAYN skills learned without a reward function, and evaluate on three natural reward functions: running, jumping, and moving away from the origin. For all tasks, DIAYN learns some skills that perform well. In contrast, a single policy that maximizes an exploration bonus (VIME) performs poorly on all tasks.

In Figure 18.22, we evaluate all skills on three reward functions: running (maximize X coordinate), jumping (maximize Z coordinate) and moving (maximize L2 distance from origin). For each skill, DIAYN learns some skills that achieve high reward. We compare to single policy trained with a pure exploration objective (VIME [105]). Whereas previous work (e.g., [105], [106], [109]) finds a single policy that explores well, DIAYN optimizes a *collection* of policies, which enables more diverse exploration.

Learning $p(z)$

We used our method as a starting point when comparing to VIC [194] in Section 5.4. While $p(z)$ is fixed in our method, we implement VIC by learning $p(z)$. In this section, we describe how we learned $p(z)$, and show the effect of learning $p(z)$ rather than leaving it fixed.

How to Learn $p(z)$

We choose $p(z)$ to optimize the following objective, where $p_z(s)$ is the distribution over states induced by skill s:

$$\begin{aligned}
\mathcal{H}[S, Z] &= \mathcal{H}[Z] - \mathcal{H}[Z \mid S] \\
&= \sum_z -p(z) \log p(z) + \sum_z \mathbb{E}_{s \sim p_z(s)} \left[\log p(z \mid s) \right] \\
&= \sum_z p(z) \left(\mathbb{E}_{s \sim p_z(s)} \left[\log p(z \mid s) \right] - \log p(z) \right)
\end{aligned}$$

For clarity, we define $p_z^t(s)$ as the distribution over states induced by skill z at epoch t, and define $\ell_t(z)$ as an approximation of $\mathbb{E}[\log p(z \mid s)]$ using the policy and discriminator from epoch t:

$$\ell_t(z) \triangleq \mathbb{E}_{s \sim p_z^t(s)}[\log q_t(z \mid s)]$$

Noting that $p(z)$ is constrained to sum to 1, we can optimize this objective using the method of Lagrange multipliers. The corresponding Lagrangian is

$$\mathcal{L}(p) = \sum_z p(z) \left(\ell_t(z) - \log p(z) \right) + \lambda \left(\sum_z p(z) - 1 \right)$$

whose derivative is

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial p(z)} &= \cancel{p(z)} \left(\frac{-1}{\cancel{p(z)}} \right) + \ell_t(z) - \log p(z) + \lambda \\
&= \ell_t(z) - \log p(z) + \lambda - 1
\end{aligned}$$

Setting the derivative equal to zero, we get

$$\log p(z) = \ell_t(z) + \lambda - 1$$

and finally arrive at

$$p(z) \propto e^{\ell_t(z)}$$

Effect of Learning $p(z)$

In this section, we briefly discuss the effect of learning $p(z)$ rather than leaving it fixed. To study the effect of learning $p(z)$, we compared the entropy of $p(z)$ throughout training. When $p(z)$ is fixed, the entropy is a constant ($\log(50) \approx 3.9$). To convert nats to a more interpretable quantity, we compute the effective number of skills by exponentiation the entropy:

$$\text{effective num. skills} \triangleq e^{\mathcal{H}[Z]}$$

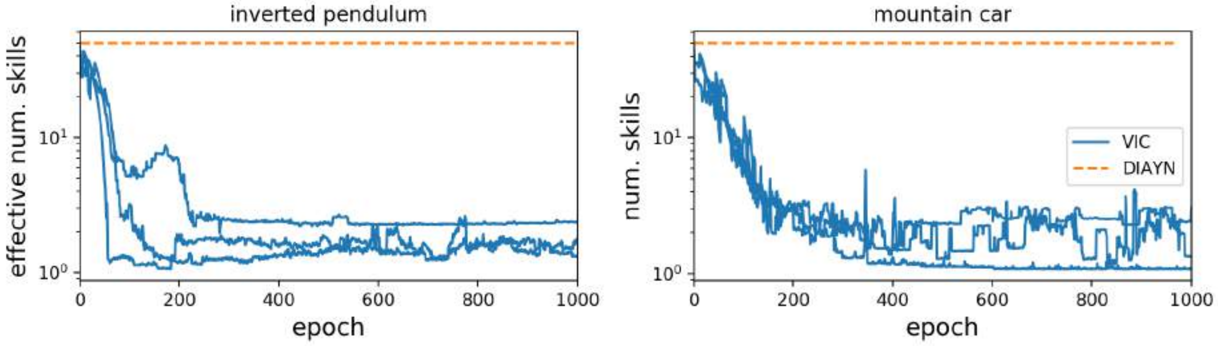


Figure 18.23: **Effect of learning** $p(z)$: We plot the effective number of skills that are sampled from the skill distribution $p(z)$ throughout training. Note how learning $p(z)$ greatly reduces the effective number on inverted pendulum and mountain car. We show results from 3 random seeds for each environment.

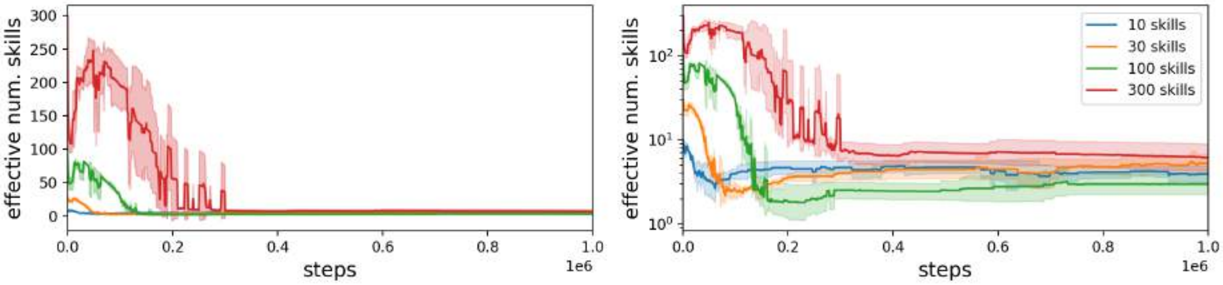


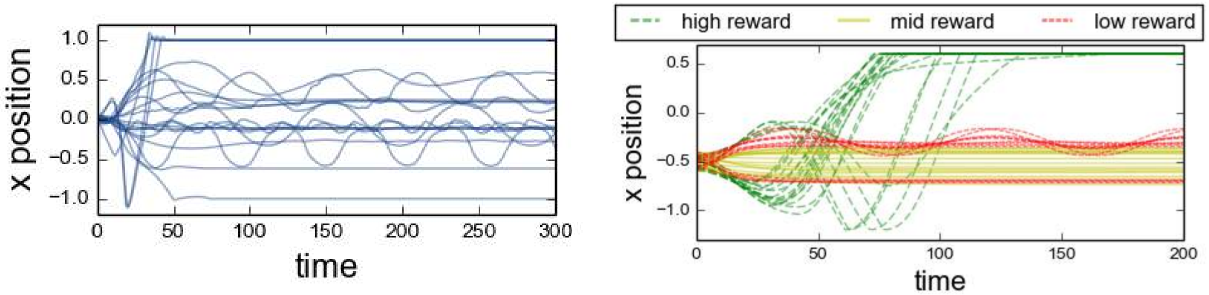
Figure 18.24: **Learning** $p(z)$ **with varying number of skills**: We repeat the experiment in Figure 5.4 for varying sizes of z. Regardless of the size of z, learning $p(z)$ causes the effective number of skills to drop to less than 10. The two subplots show the same data *(Left)* on a linear scale and *(Right)* logarithmic scale. We plot the mean and standard deviation across 3 random seeds.

Figure 18.23 shows the effective number of skills for half cheetah, inverted pendulum, and mountain car. Note how the effective number of skills drops by a factor of 10x when we learn $p(z)$. This observation supports our claim that learning $p(z)$ results in learning fewer diverse skills. Figure 18.24 is a repeat of the experiment in Figure 18.23, where we varying the dimension of z. Note that the dimension of z equals the maximum number of skills that the agent could learn. We observe that the effective number of skills plummets throughout training, even when using a high-dimensional vector for z.

Visualizing Learned Skills

Classic Control Tasks

In this section, we visualize the skills learned for inverted pendulum and mountain car without a reward. Not only does our approach learn skills that solve the task without rewards, it learns multiple distinct skills for solving the task. Figure 18.25 shows the X position of the agent across time, within one episode. For inverted pendulum (Fig. 18.25a), we plot only skills that solve the task. Horizontal lines with different X coordinates correspond to skills balancing the pendulum



(a) Inverted Pendulum (b) Mountain Car

Figure 18.25: **Visualizing Skills**: For every skill, we collect one trajectory and plot the agent's X coordinate across time. For inverted pendulum (top), we only plot skills that balance the pendulum. Note that among balancing skills, there is a wide diversity of balancing positions, control frequencies, and control magnitudes. For mountain car (bottom), we show skills that achieve larger reward (complete the task), skills with near-zero reward, and skills with very negative reward. Note that skills that solve the task (green) employ varying strategies.

at different positions along the track. The periodic lines correspond to skills that oscillate back and forth while balancing the pendulum. Note that skills that oscillate have different X positions, amplitudes, and periods. For mountain car (Fig. 18.25b), skills that climb the mountain employ a variety of strategies for to do so. Most start moving backwards to gather enough speed to summit the mountain, while others start forwards, then go backwards, and then turn around to summit the mountain. Additionally, note that skills differ in when the turn around and in their velocity (slope of the green lines).

Simulated Robot Tasks

Figures 18.26, 18.27, and 18.28 show more skills learned *without reward*.

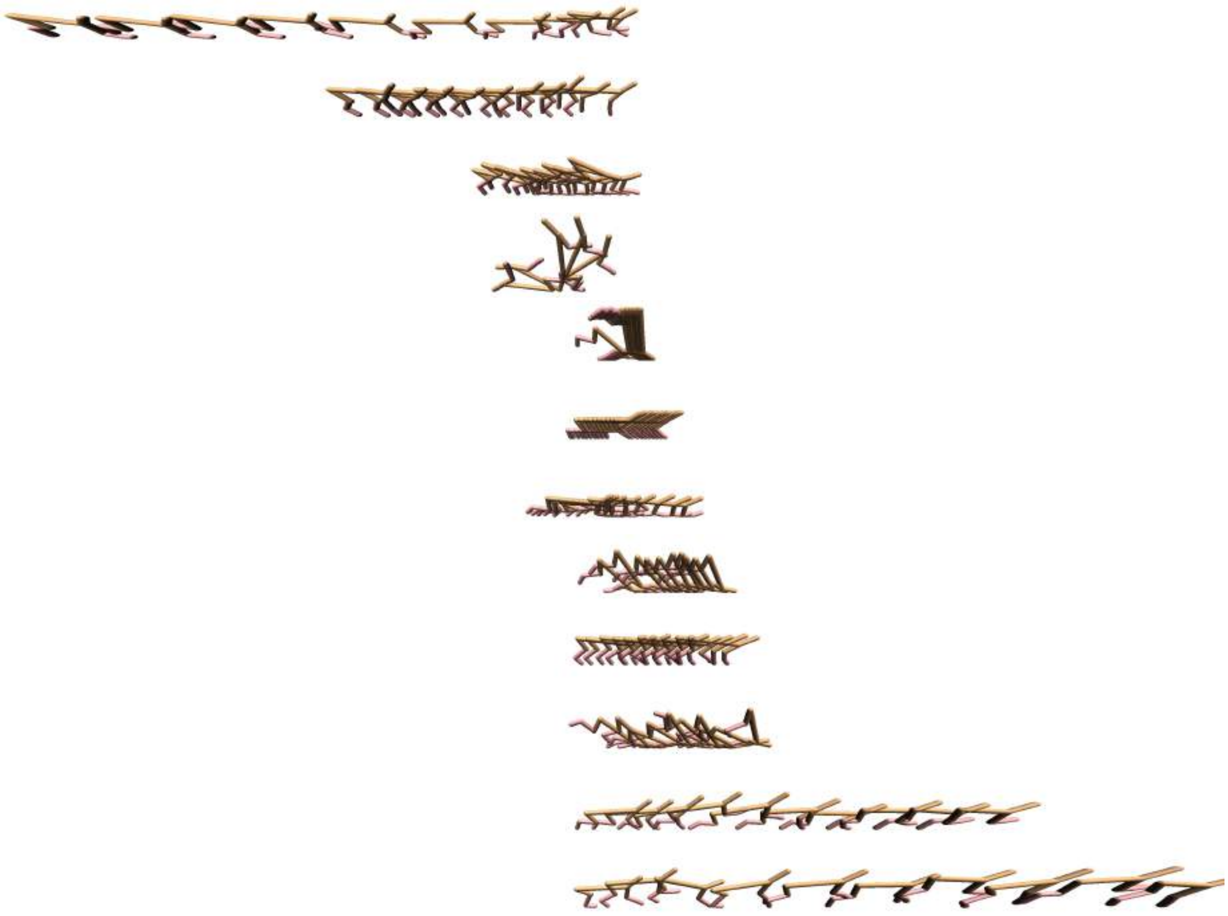


Figure 18.26: **Half cheetah skills**: We show skills learned by half-cheetah with no reward.

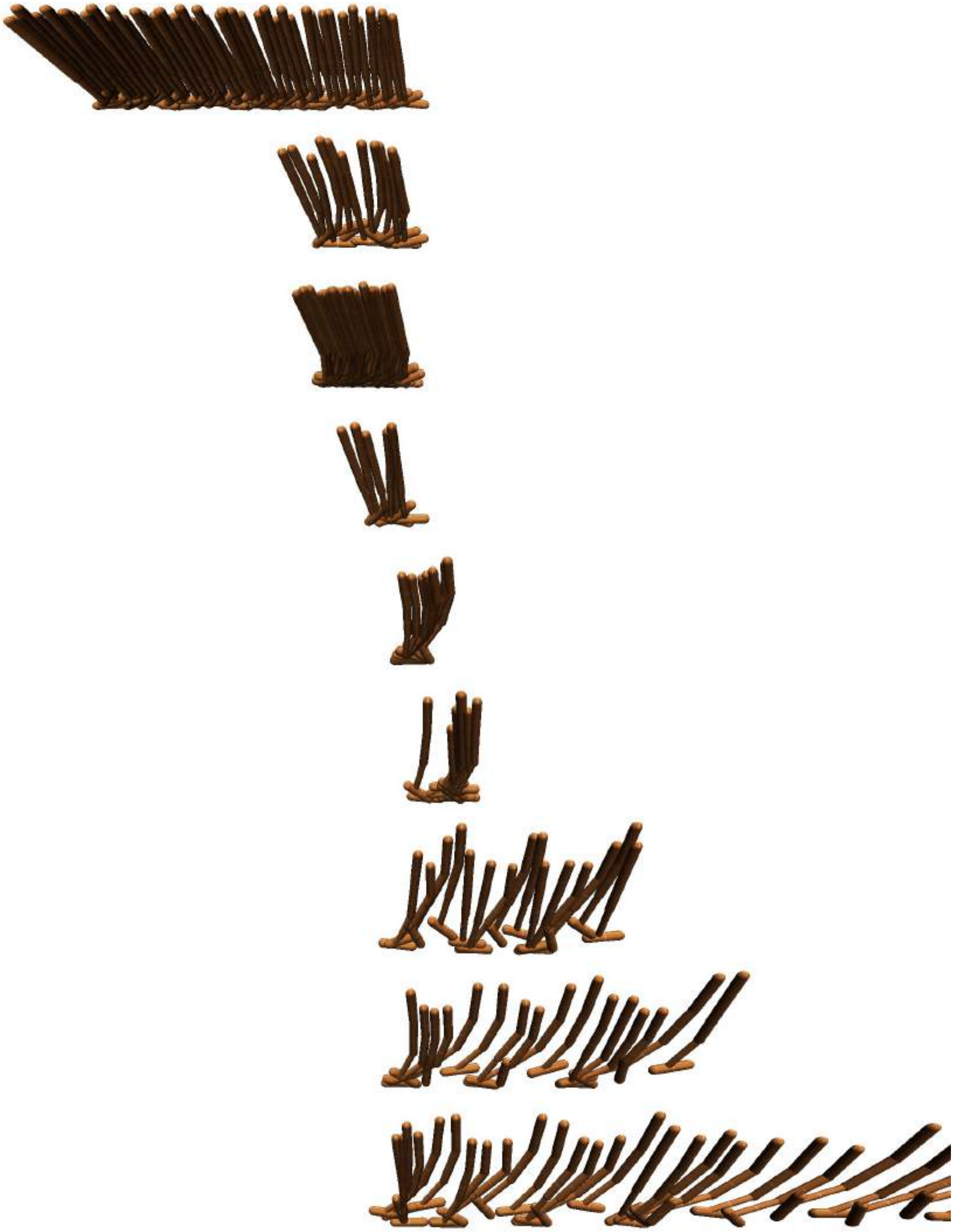


Figure 18.27: **Hopper Skills**: We show skills learned by hopper with no reward.

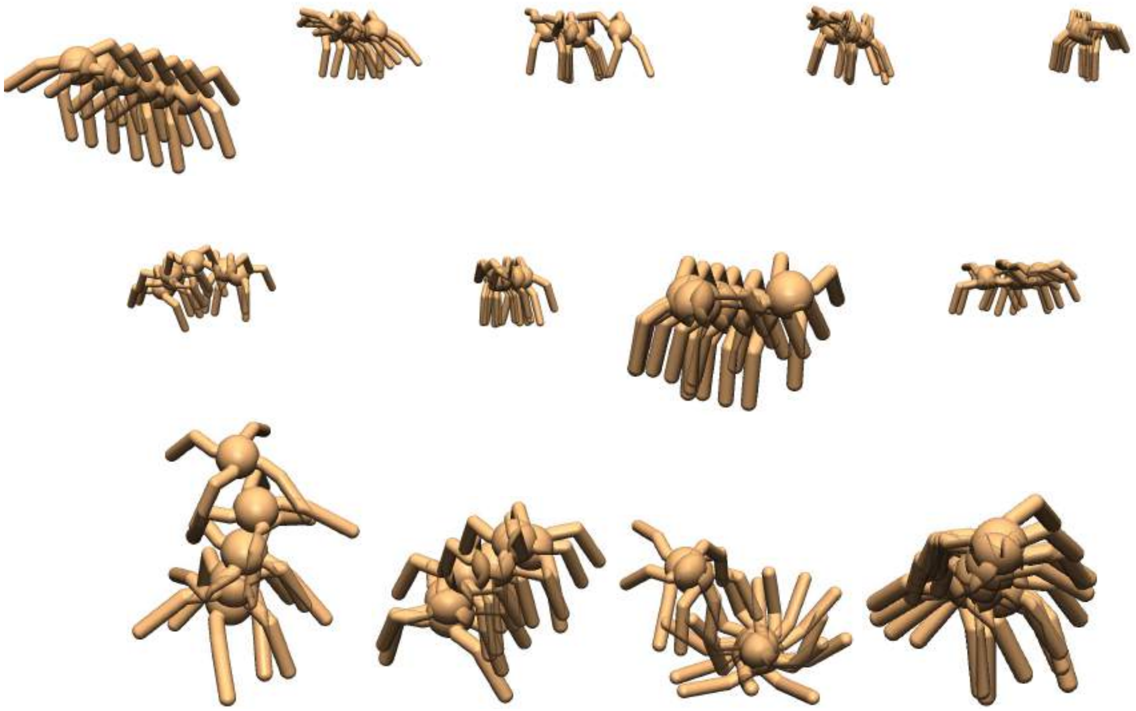


Figure 18.28: **Ant skills**: We show skills the ant learns without any supervision. Ant learns *(top row)* to move right, *(middle row)* to move left, *(bottom row, left to right)* to move up, to move down, to flip on its back, and to rotate in place.

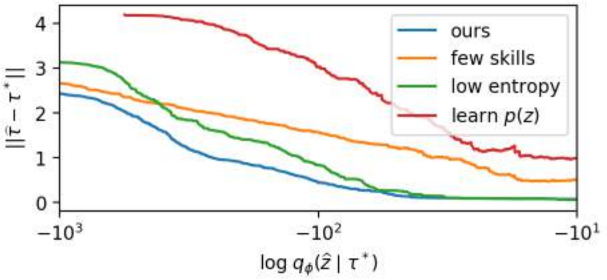


Figure 18.29: **Imitating an expert**: Across 600 imitation tasks, we find our method more closely matches the expert than all baselines.

Imitation Learning

Given the expert trajectory, we use our learned discriminator to estimate which skill was most likely to have generated the trajectory:

$$\hat{z} = \arg\max_z \Pi_{s_t \in \tau^*} q_\phi(z \mid s_t)$$

As motivation for this optimization problem, note that each skill induces a distribution over states, $p^z \triangleq p(s \mid z)$. We use p^* to denote the distribution over states for the expert policy. With a fixed prior distribution $p(z)$ and a perfect discriminator $q_\phi(z \mid s) = p(z \mid s)$, we have $p(s \mid z) \propto q_\phi(z \mid s)$ as a function of z. Thus, Equation 18.3 is an M-projection of the expert distribution over states onto the family of distributions over states, $\mathcal{P} = \{p^z\}$:

$$\arg\min_{p^z \in \mathcal{P}} D(p^* \mid\mid p^z) \tag{18.6}$$

For clarity, we omit a constant that depends only on p^*. Note that the use of an M-projection, rather than an I-projection, helps guarantee that the retrieved skill will visit all states that the expert visits [211]. In our experiments, we solve Equation 18.6 by simply iterating over skills.

Imitation Learning Experiments

The "expert" trajectories are actually generated synthetically in these experiments, by running a different random seed of our algorithm. A different seed is used to ensure that the trajectories are not actually produced by any of the currently available skills. Of course, in practice, the expert trajectories might be provided by any other means, including a human. For each expert trajectory, we retrieve the closest DIAYN skill $\hat{z}$ using Equation 5.4. Evaluating $q_\phi(\hat{z} \mid \tau^*)$ gives us an estimate of the probability that the imitation will match the expert (e.g., for a safety critical setting). This quantity is useful for predicting how accurately our method will imitate an expert before executing the imitation policy. In a safety critical setting, a user may avoid attempting tasks where this score is low. We compare our method to three baselines. The "low entropy" baseline is a variant on our method with lower entropy regularization. The "learned $p(z)$" baseline learns the distribution over skills. Note that Variational Intrinsic Control [194] is a combination of the "low entropy" baseline and the "learned $p(z)$" baseline. Finally, the "few skills" baseline learns only 5 skills, whereas all other methods learn 50. Figure 18.29 shows the results aggregated across 600 imitation tasks. The X-axis shows the discriminator score, our estimate for how well the imitation policy will match the expert. The Y-axis shows the true distance between the trajectories, as measured by L2 distance in state space. For all methods, the distance between the expert and the imitation decreases as the discriminator's score increases, indicating that the discriminator's score is a good predictor of task performance. Our method consistently achieves the lowest trajectory distance among all methods. The "low entropy" baseline is slightly worse, motivating our decision to learn maximum entropy skills. When imitating tasks using the "few skills" baseline, the imitation trajectories are even further from the expert trajectory. This is expected – by learning more skills, we obtain a better "coverage" over the space of skills. A "learn $p(z)$" baseline that learns the distribution over skills also performs poorly. Recalling that [194] is a combination of the "low entropy" baseline and the "learn $p(z)$" baseline, this plot provides evidence that using maximum entropy policies and fixing the distribution for $p(z)$ are two factors that enabled our method to scale to more complex tasks.

18.4 Appenix D: Appendix for Chapter 6

Proofs

Lemma 1 Let π be a policy for which $\rho_\pi^T(s)$ is uniform. Then π has lowest worst-case regret.

Proof of Lemma 2. To begin, we note that all goal distributions $p(s_g)$ have equal regret for policies where $\rho_\pi^T(s) = 1/|\mathcal{S}|$ is uniform:

$$\textsc{Regret}_p(\pi) = \int \frac{p(s_g)}{\rho_\pi^T(s_g)} ds_g = \int \frac{p(s_g)}{1/|\mathcal{S}|} ds_g = |\mathcal{S}|$$

Now, consider a policy π' for which $\rho_\pi^T(s)$ is not uniform. For simplicity, we will assume that the argmin is unique, though the proof holds for non-unique argmins as well. The worst-case goal distribution will choose the state s^- where that the policy is least likely to visit:

$$p^-(s_g) \triangleq \mathbb{1}(s_g = \arg\min_s \rho_\pi^T(s))$$

Thus, the worst-case regret for policy π' is strictly greater than the regret for a uniform π:

$$\begin{aligned} \max_p \textsc{Regret}_p(\pi) &= \textsc{Regret}_{p^-}(\pi) \\ &= \int \frac{\mathbb{1}(s_g = \arg\min_s \rho_\pi^T(s))}{\rho_\pi^T(s_g)} ds_g \\ &= \frac{1}{\min_s \rho_{\pi'}^T(s)} > |\mathcal{S}| \end{aligned} \tag{18.7}$$

Thus, a policy π' for which ρ_π^T is non-uniform cannot be minimax, so the optimal policy has a uniform marginal ρ_π^T. □

Lemma 2: Mutual information $I(s_T; z)$ is maximized by a task distribution $p(s_g)$ which is uniform over goal states.

Proof of Lemma 3. We define a latent variable model, where we sample a latent variable z from a uniform prior $p(z)$ and sample goals from a conditional distribution $p(s_T \mid z)$. To begin, note that the mutual information can be written as a difference of entropies:

$$I_p(s_T; z) = \mathcal{H}_p[s_T] - \mathcal{H}_p[s_T \mid z]$$

The conditional entropy $\mathcal{H}_p[s_T \mid z]$ attains the smallest possible value (zero) when each latent variable z corresponds to exactly one final state, s_z. In contrast, the marginal entropy $\mathcal{H}_p[s_T]$ attains the largest possible value ($\log |\mathcal{S}|$) when the marginal distribution $p(s_T) = \int p(s_T \mid z)p(z)dz$ is uniform. Thus, a task uniform distribution $p(s_g)$ maximizes $I(s_T; z)$. Note that for any non-uniform task distribution $q(s_T)$, we have $\mathcal{H}_q[s_T] < \mathcal{H}_p[s_T]$. Since the conditional entropy $\mathcal{H}_p[s_T \mid z]$ is zero, no distribution can achieve a smaller conditional entropy. This, for all non-uniform task distributions q, we have $I_q(s_T; z) < I_p(s_T; z)$. Thus, the optimal task distribution must be uniform. □

Ablations

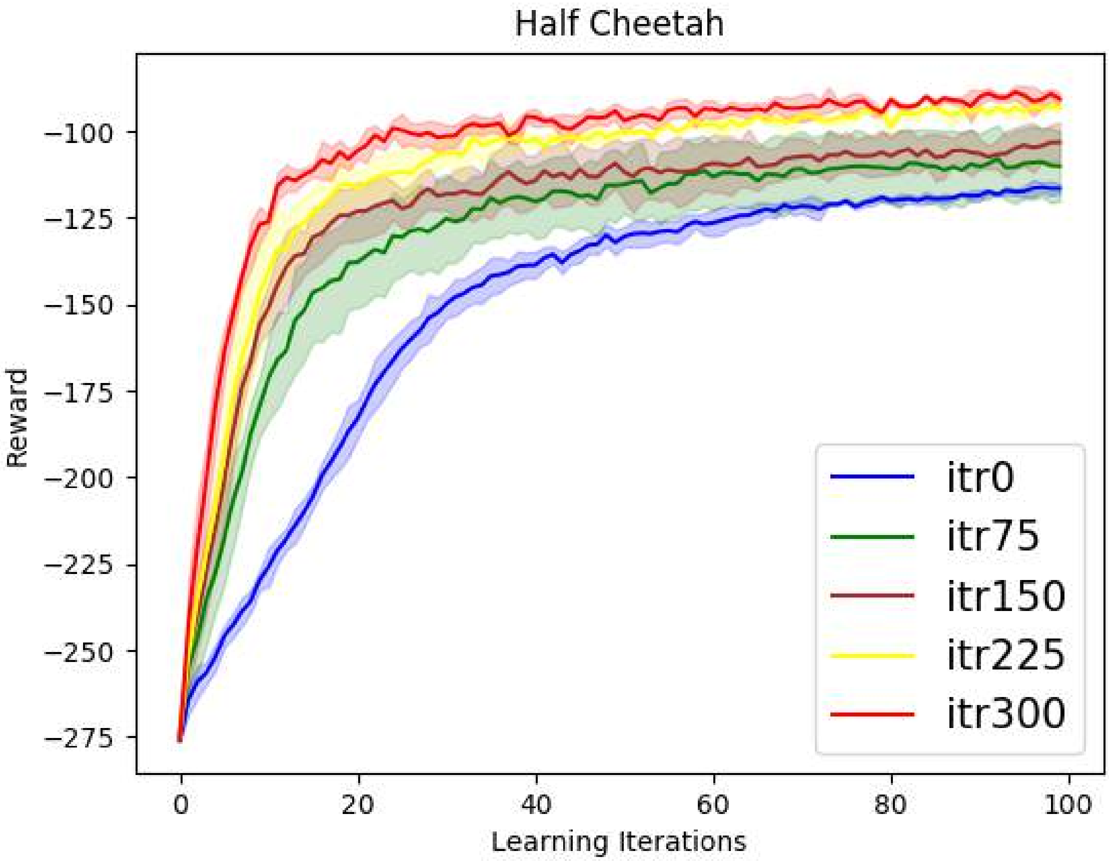
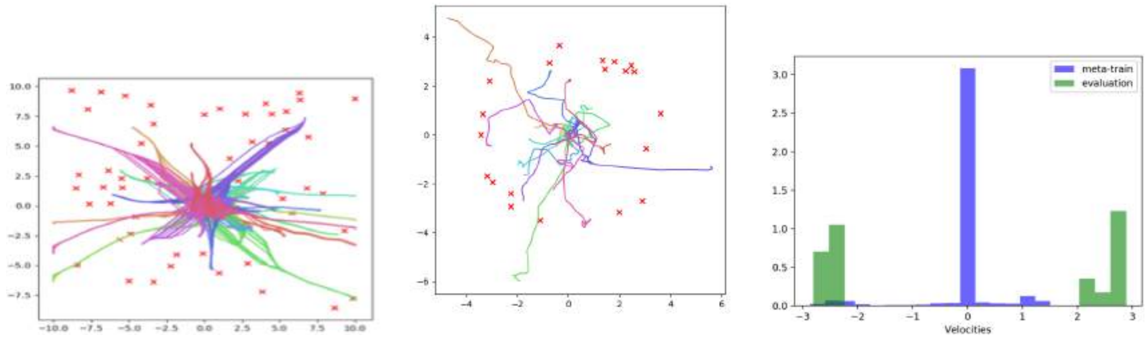


Figure 18.30: Analysis of effect of additional meta-training on meta-test time learning of new tasks. For larger iterations of meta-trained policies, we have improved test time performance, showing that additional meta-training is beneficial.

To understand the method performance more clearly, we also add an ablation study where we compare the meta-test performance of policies at different iterations along meta-training. This shows the effect that additional meta-training has on the fast learning performance for new tasks. This comparison is shown in Figure 18.30. As can be seen here, at iteration 0 of meta-training the policy is not a very good initialization for learning new tasks. As we move further along the meta-training process, we see that the meta-learned initialization becomes more and more effective at learning new tasks. This shows a clear correlation between additional meta-training and improved meta test-time performance.



2D navigation Ant Half-Cheetah

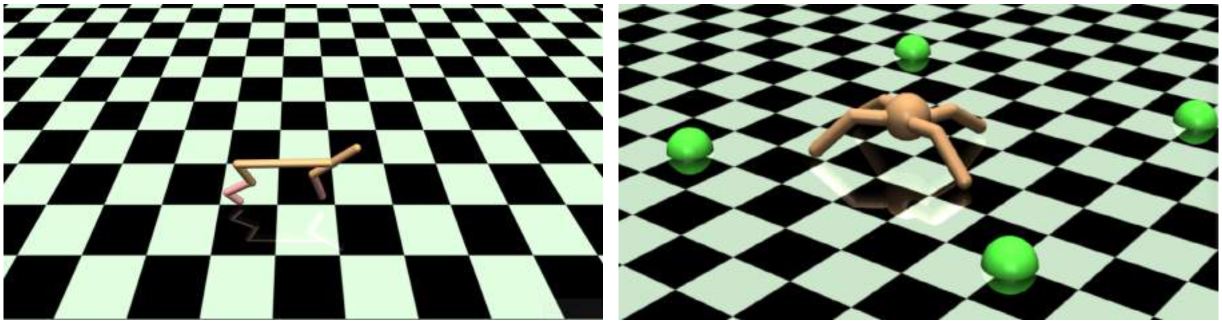
Figure 18.31: **Learned meta-training task distribution and evaluation tasks**: We plot the center of mass for various skills discovered by point mass and ant using DIAYN, and a blue histogram of goal velocities for cheetah. Evaluation tasks, which are not provided to the algorithm during meta-training, are plotted as red 'x' for ant and pointmass, and as a green histogram for cheetah. While the meta-training distribution is broad, it does not fully cover the evaluation tasks. Nonetheless, meta-learning on this *learned* task distribution enables efficient learning on a test task distribution.

Analysis of Learned Task Distributions

We can analyze the tasks discovered through unsupervised exploration and compare them to tasks we evaluate on at meta-test time. Figure 18.31 illustrates these distributions using scatter plots for 2D navigation and the Ant, and a histogram for the HalfCheetah. Note that we visualize dimensions of the state that are relevant for the evaluation tasks – positions and velocities – but these dimensions are *not* specified in any way during unsupervised task acquisition, which operates on the entire state space. Although the tasks proposed via unsupervised exploration provide fairly broad coverage, they are clearly quite distinct from the meta-test tasks, suggesting the approach can tolerate considerable distributional shift. Qualitatively, many of the tasks proposed via unsupervised exploration such as jumping and falling that are not relevant for the evaluation tasks. Our choice of the evaluation tasks was largely based on prior work, and therefore not tailored to this exploration procedure. The results for unsupervised meta-RL therefore suggest quite strongly that unsupervised task acquisition can provide an effective meta-training set, at least for MAML, even when evaluating on tasks that do not closely match the discovered task distribution.

Hyperparameter Details

For all our experiments, we used DIAYN to acquire the task proposals using 20 skills for half-cheetah and for ant and 50 skills for the 2D navigation. We illustrate these half cheetah and ant in Fig. 18.32. We ran the domains using the standard DIAYN hyperparameters described in `https://github.com/ben-eysenbach/sac` to acquire task proposals. These proposals were then fed into the MAML algorithm `https://github.com/cbfinn/maml_rl`, with inner learning rate 0.1, meta learning rate 0.01, inner batch size 40, outer batch size, path length 100, using 2 layer networks with 300 units each with ReLu nonlinearities. We vary the meta-batch size according to the number of skills: 50 for pointmass, 20 for cheetah, and 20 ant. The test time learning is done with the same parameters for the UMRL variants, and done using REINFORCE



Half-Cheetah

Ant

Figure 18.32: **Environments**: *(Left)* Half-Cheetah and *(Right)* Ant

with the Adam optimizer for the comparison with learning from scratch. We swept over learning rates for learning from scratch via vanilla policy gradient, and found that using ADAM with adaptive step size is the most stable and quick at learning.

18.5 Appendix E: Appendix for Chapter 10

Experimental Details

We use feed-forward MLPs for all our policies, with two layer neural networks with 256 units each and ReLu nonlinearities used for both the high-level policy π_θ and the low-level policy π_ϕ in all methods. Flat baselines use the same architecture as well and additional experimentation with the architecture did not yield substantially different results. We train all imitation learning algorithms with the ADAM optimizer using a batch size of 128 and a learning rate of 0.005. We choose W_l to be 30 and W_h to be 260 in all experiments. Our ablations suggest that the larger the window, the harder the learning problem becomes for both imitation and RL fine-tuning.

For reinforcement learning, we utilize a variant of Trust Region Policy Optimization (TRPO). We fine-tune on 17 different compound goals individually, with a path length of 280 for every compound goal, and the low-level horizon set to 30. We use 100 trajectories in each iteration of on-policy fine-tuning, with a discount of 0.995. When using variants of augmenting the policy gradient objective with demonstrations, we experimented with different weights λ_h and λ_l, but we found 0.0001 to work well. We use a batch size of a 100 trajectories per iteration, and fairly standard parameters for truncated natural policy gradient based on `https://github.com/aravindr93/mjrl`

The simulation environment has a 30-dimensional state space which consists of positions of the arm and the objects in the scene. The action space is 9 dimensional with 7 DoF for the arm and 2 DoF for the gripper. The actions are represented as the joint velocity.

Reward Function Details

For the comparisons detailed in Section 5.3, the reward functions used for sparse, euclidean and element-wise sparse reward functions are detailed below, with ϵ set to 0.3. For all our experimental results in Fig 5, we use the sparse reward variant as the reward function for fine-tuning.

$$R_{\text{sparse}}(s, g) = \mathbb{1}(\|s - g\|_2 < \epsilon) \tag{18.8}$$

$$R_{\text{euclidean}}(s, g) = -\|s - g\|_2 \tag{18.9}$$

$$R_{\text{elementwise sparse}}(s, g) = \sum_{\text{idx}\in\text{element indices}} \mathbb{1}(\|s[\text{idx}] - g[\text{idx}]\|_2 < \epsilon) \tag{18.10}$$

In the element-wise sparse reward case, idx is selected to be the indices of state corresponding to different distinct elements of the scene such as the microwave, stove burners, light switch, sliding cabinet, hinge cabinets and so on. The robot arm is excluded from these indices.

Oracle Baseline Details

For the oracle comparison described in Section 5, a hand-designed scheme is used to segment the demonstration into segments corresponding to semantically meaningful components, thereby generating variable sized windows rather than fixed length ones. Specifically, we split a segment any time one of [microwave, kettle, light switch, burners, slide cabinet, hinge cabinet] is moved more than $\epsilon = 0.3$. This leads to a variable segment generation scheme, which generates splits that is shown in Fig 18.33.

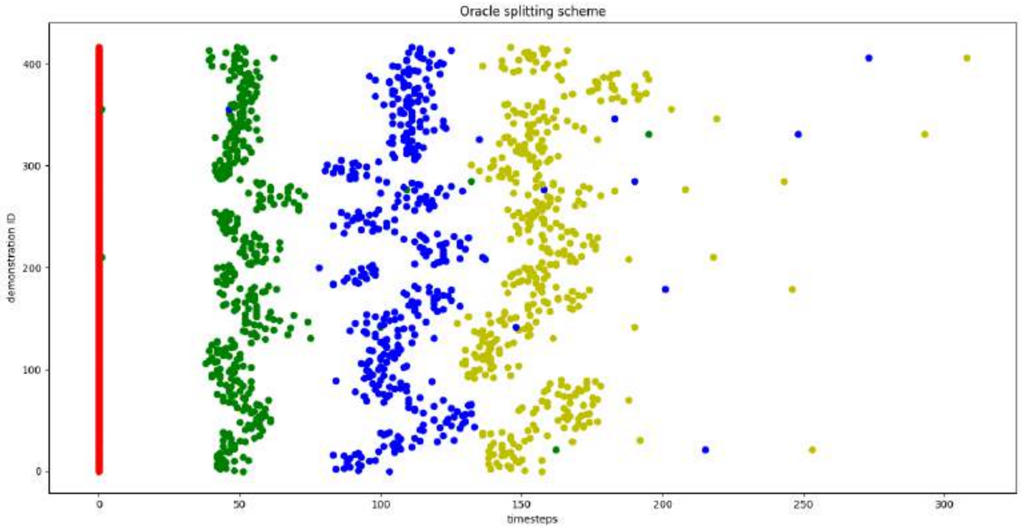


Figure 18.33: Splits generated by the oracle segmentation scheme. Each color corresponds to a different split and different demonstrations as plotted as different rows along the y-axis, with time-steps along the x-axis. We see that the split of demonstrations is fairly variable in time-steps. This makes the imitation learning and fine-tuning quite challenging.

Segments generated in this fashion can then be used for imitation learning both the low-level and high-level policies. Specifically, the actions for the high level policies are chosen to be the states at which the segments are broken, and the low level is trained via goal conditioned behavior cloning with those states set as goals.

Visualization of Learned Behaviors

We show example visualizations of several successful learned behaviors for compound tasks, and some failed behaviors to better understand the the method. These can be best appreciated by viewing the accompanying videos on the supplementary website `https://relay-policy-learning.github.io`

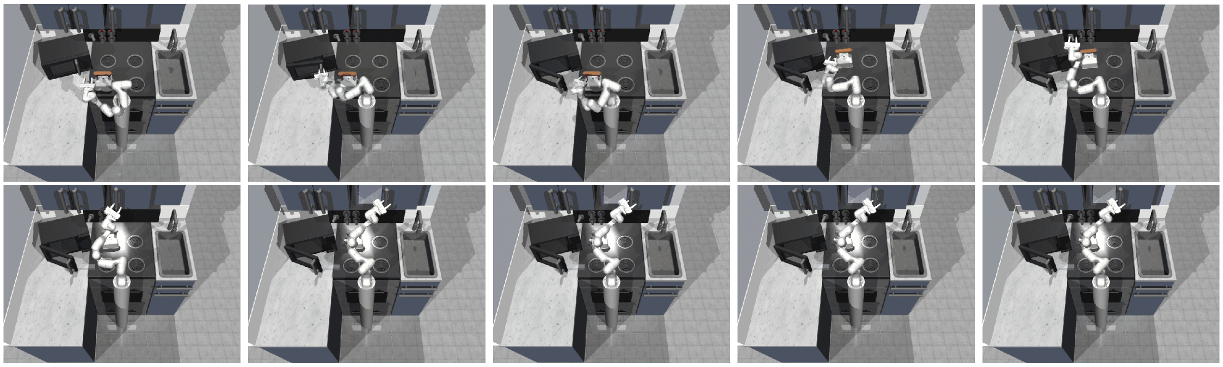


Figure 18.34: Visualization of successful learned behavior for opening microwave, moving kettle, turning on light switch, sliding the slider

Successful cases

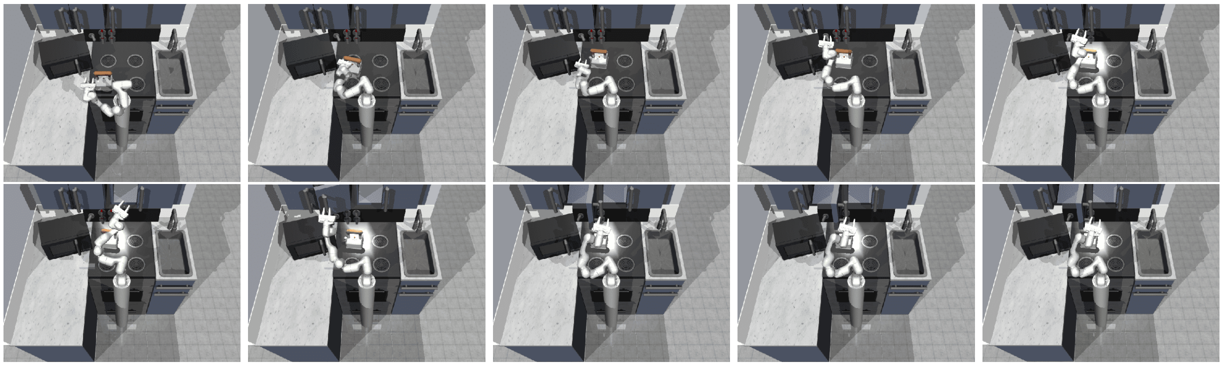


Figure 18.35: Visualization of successful learned behavior for moving kettle, turning top knob, sliding the slider and opening the hinge cabinet

Failure Cases

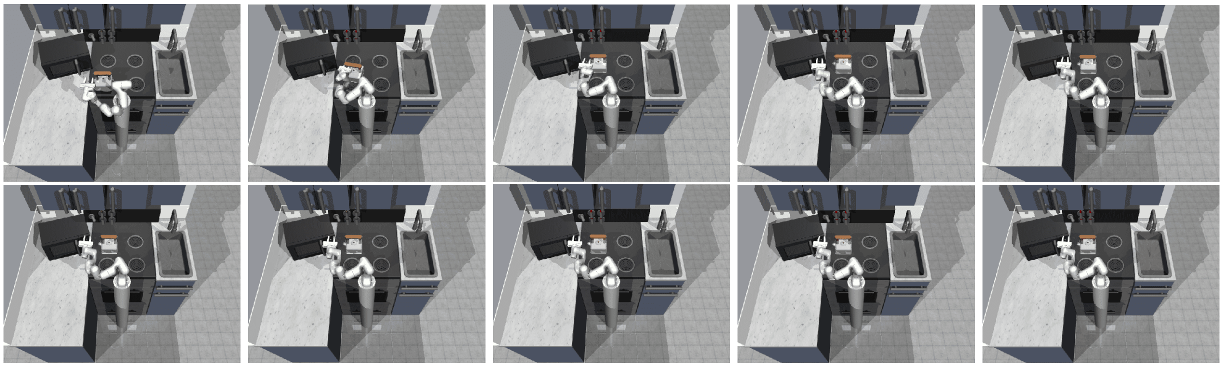


Figure 18.36: Visualization of failing learned behavior for moving kettle, turning the bottom knob, moving the slider and turning on the oven light

18.6 Appendix F: Appendix for Chapter 11

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:

$$
\begin{aligned}
\pi_{k+1} = \ & \underset{\pi\in\Pi}{\arg\max}\ \mathbb{E}_{\mathbf{a}\sim\pi(\cdot|\mathbf{s})}[A^{\pi_k}(\mathbf{s},\mathbf{a})] && (18.11)\\
& \text{s.t. } D_{\mathrm{KL}}(\pi(\cdot|\mathbf{s})||\pi_{\mathcal{D}}(\cdot|\mathbf{s})) \leq \epsilon && (18.12)\\
& \int_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s})d\mathbf{a} = 1. && (18.13)
\end{aligned}
$$

Our derivation follows [354] and [349]. The analytic solution for the constrained optimization problem above can be obtained by enforcing the KKT conditions. The Lagrangian is:

$$
\begin{aligned}
\mathcal{L}(\pi,\lambda,\alpha) = & \mathbb{E}_{\mathbf{a}\sim\pi(\cdot|\mathbf{s})}[A^{\pi_k}(\mathbf{s},\mathbf{a})] && (18.14)\\
& + \lambda(\epsilon - D_{\mathrm{KL}}(\pi(\cdot|\mathbf{s})||\pi_\beta(\cdot|\mathbf{s}))) && (18.15)\\
& + \alpha(1 - \int_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s})d\mathbf{a}). && (18.16)
\end{aligned}
$$

Differentiating with respect to π gives:

$$
\frac{\partial\mathcal{L}}{\partial\pi} = A^{\pi_k}(\mathbf{s},\mathbf{a}) - \lambda\log\pi_\beta(\mathbf{a}|\mathbf{s}) + \lambda\log\pi(\mathbf{a}|\mathbf{s}) + \lambda - \alpha. \tag{18.17}
$$

Setting $\frac{\partial\mathcal{L}}{\partial\pi}$ to zero and solving for π gives the closed form solution to this problem:

$$
\pi^*(\mathbf{a}|\mathbf{s}) = \frac{1}{Z(\mathbf{s})}\pi_{\mathcal{D}}(\mathbf{a}|\mathbf{s})\exp\left(\frac{1}{\lambda}A^{\pi_k}(\mathbf{s},\mathbf{a})\right), \tag{18.18}
$$

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 $\rho_{\pi_{\mathcal{D}}}(\mathbf{s})$:

$$
\begin{aligned}
& \underset{\theta}{\arg\min} \underset{\rho_{\pi_{\mathcal{D}}}(\mathbf{s})}{\mathbb{E}}\left[D_{\mathrm{KL}}(\pi^*(\cdot|\mathbf{s})||\pi_\theta(\cdot|\mathbf{s}))\right] && (18.19)\\
= & \underset{\theta}{\arg\min} \underset{\rho_{\pi_{\mathcal{D}}}(\mathbf{s})}{\mathbb{E}}\left[\underset{\pi^*(\cdot|\mathbf{s})}{\mathbb{E}}[-\log\pi_\theta(\cdot|\mathbf{s})]\right] && (18.20)
\end{aligned}
$$

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 $s, a \sim \mathcal{D}$, 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 $\pi_\theta \geq \alpha_\theta$. Then the upper bound:

$$D_{\mathrm{KL}}(\pi^*||\pi_\theta) \leq \frac{2}{\alpha_\theta} D_{\mathrm{TV}}(\pi^*, \pi_\theta)^2 \tag{18.21}$$

$$\leq \frac{1}{\alpha_\theta} D_{\mathrm{KL}}(\pi_\theta||\pi^*) \tag{18.22}$$

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.

Implementation Details

We implement the algorithm building on top of twin soft actor-critic [23], which incorporates the twin Q-function architecture from twin delayed deep deterministic policy gradient (TD3) from [344]. All off-policy algorithm comparisons (SAC, BRAC, MPO, ABM, BEAR) are implemented from the same skeleton. The base hyperparameters are given in Table 18.12. The policy update is replaced with:

$$\theta_{k+1} = \arg\max_\theta \mathop{\mathbb{E}}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}} \left[\log \pi_\theta(\mathbf{a}|\mathbf{s}) \frac{1}{Z(\mathbf{s})} \exp\left(\frac{1}{\lambda} A^{\pi_k}(\mathbf{s},\mathbf{a}) \right) \right]. \tag{18.23}$$

Env	Use $Z(\mathbf{s})$	Omit $Z(\mathbf{s})$
pen	84%	98%
door	0%	95%
relocate	0%	54%

Table 18.11: 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 [349] and other prior work [348], [352], [371], 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 18.11.

While prior work [348], [349], [371] 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, $Z(s) = \int_{\mathbf{a}} f(\mathbf{a})g(\mathbf{a})d\mathbf{a} \leq \sqrt{\int_{\mathbf{a}} f(\mathbf{a})^2 d\mathbf{a} \int_{\mathbf{a}} g(\mathbf{a})^2 d\mathbf{a}} = C_1$. 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 $Z(\mathbf{s}) \geq 2(\sqrt{\frac{M_f M_g}{m_f m_g}} + \frac{m_f m_g}{M_f M_g})^{-1} C_1 = C_2$. We therefore have $C_1 \leq Z(\mathbf{s}) \leq C_2$, 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.

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.

Dexterous Manipulation Environments

These environments are modified from those proposed by [95].

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 $r = \mathbb{1}_{|x_p - d_p| \leq 0.075} \mathbb{1}_{|x_o \cdot d_o| \leq 0.95}$ - 1. In [95], 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.

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 18.12: Hyper-parameters used for RL experiments.

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 and d_p denote the desired position. The reward is $r = \mathbb{1}_{|x_p - d_p| \leq 0.1}$ - 1.

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 $\sigma = 0.3$.

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 18.13. 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 [95].

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 18.13: 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	$D_{\mathrm{KL}}(\pi_\theta \| \bar{Q})$	No	None
SAC + BC	Q^π	Mixed	No	None
BCQ	Q^π	$D_{\mathrm{KL}}(\pi_\theta \| \bar{Q})$	Yes	Support (ℓ^∞)
BEAR	Q^π	$D_{\mathrm{KL}}(\pi_\theta \| \bar{Q})$	Yes	Support (MMD)
AWR	Q^β	$D_{\mathrm{KL}}(\bar{Q} \| \pi_\theta)$	No	Implicit
MPO	Q^π	$D_{\mathrm{KL}}(\bar{Q} \| \pi_\theta)$	Yes*	Prior
ABM-MPO	Q^π	$D_{\mathrm{KL}}(\bar{Q} \| \pi_\theta)$	Yes	Learned Prior
DAPG	-	$J(\pi_\theta)$	No	None
BRAC	Q^π	$D_{\mathrm{KL}}(\pi_\theta \| \bar{Q})$	Yes	Explicit KL penalty
AWAC (Ours)	Q^π	$D_{\mathrm{KL}}(\bar{Q} \| \pi_\theta)$	No	Implicit

Figure 18.37: Comparison of prior algorithms that can incorporate prior datasets. See section 18.6 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.

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 18.37.

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 [23], 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 [350] 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 [349], we add all prior data to the replay buffer before online training. We use the implementation provided by [349], 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 [348] 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 [341]. 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 AWAC to be:

$$\begin{aligned}\theta_i \longleftarrow \arg\max_{\theta_i} \quad & \mathbb{E}_{s\sim\mathcal{D},a\sim\pi(a|s)} \\ & \left[\log \pi_{\theta_i}(a|s)\exp(\frac{1}{\beta}Q^{\pi_\beta}(s,a))\right]. \end{aligned} \tag{18.24}$$

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 [352]. 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 $\pi_{\theta_{\text{abm}}}(a|s)$:

$$\begin{aligned}\theta_{\text{abm}} = \arg\max_{\theta_i} \quad & \mathbb{E}_{\tau\sim\mathcal{D}} \\ & \left[\sum_{t=1}^{|\tau|} \log \pi_{\theta_{\text{abm}}}(a_t|s_t) f(R(\tau_{t:N}) - \hat{V}(s))\right]. \end{aligned} \tag{18.25}$$

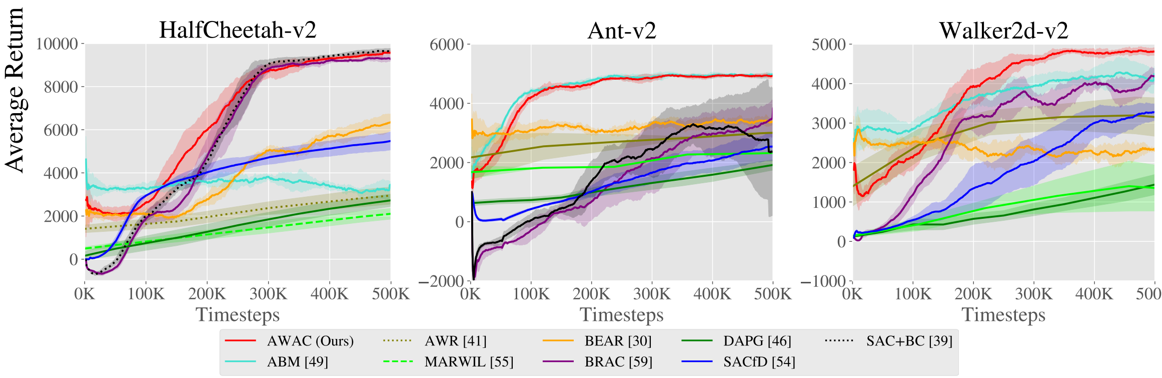


Figure 18.38: Comparison of our method and prior methods on standard MuJoCo benchmark tasks. These tasks are much easier than the dexterous manipulation tasks, and allow us to better inspect the performance of methods in the setting of offline pretraining followed by online fine-tuning. SAC+BC and BRAC 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.

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:

$$\theta_i \longleftarrow \arg\max_{\theta_i} \quad \mathbb{E}_{s\sim\mathcal{D},a\sim\pi_{\text{abm}}(a|s)} \left[\log \pi_{\theta_i}(a|s) \exp\left(\frac{1}{\lambda}(Q^{\pi_i}(s,a) - V^{\pi_i}(s))\right)\right]. \tag{18.26}$$

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 [95] 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.

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 18.38. 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 11.3.

Data efficiency. The three methods that do not estimate Q^π are DAPG [95], AWR [349], and MARWIL [348]. 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 [23], 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 [299]. SAC+BC reproduces [300] but uses SAC instead of DDPG [286] 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 [352], BEAR [340], and BRAC [350]. 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.

Extra Baseline Comparisons (CQL, AlgaeDICE)

In this section, we add comparisons to constrained Q-learning (CQL) [381] and AlgaeDICE [367]. 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 18.39.

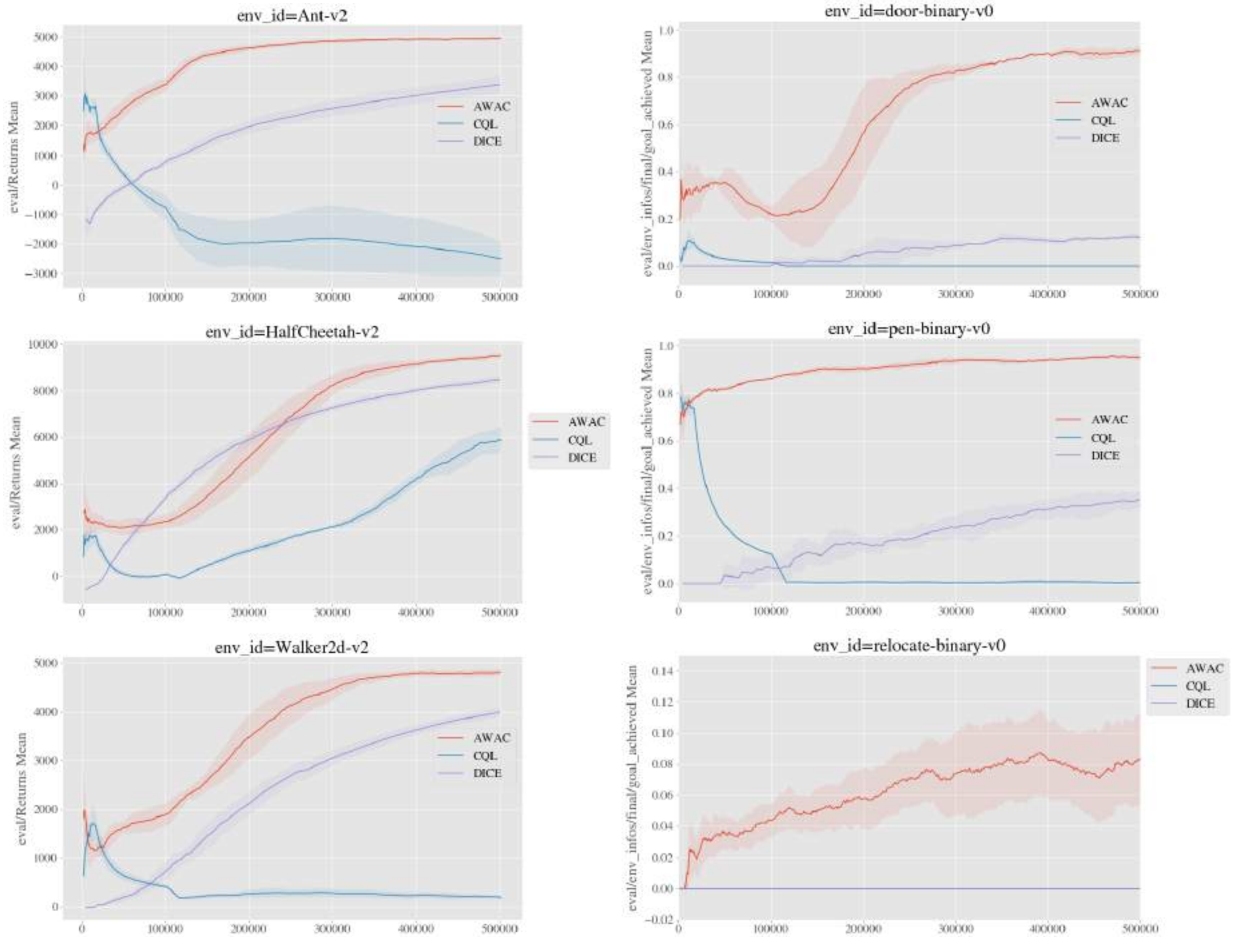


Figure 18.39: Comparison of our method (AWAC) with CQL and AlgaeDICE. CQL and AWAC perform similarly offline, but CQL does not improve when fine-tuning online. AlgaeDICE does not perform well for offline pretraining.

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 [430], 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 [340] 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 18.40. 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 18.40: Comparison of our method (AWAC) fine-tuning on varying data quality datasets in D4RL [430]. AWAC is able to improve its offline performance by further fine-tuning online.

Hardware Experimental Setup

Here, we provide further details of the hardware experimental setups, which are pictured in Fig 18.41.

Dexterous Manipulation with a 3 Fingered Claw.

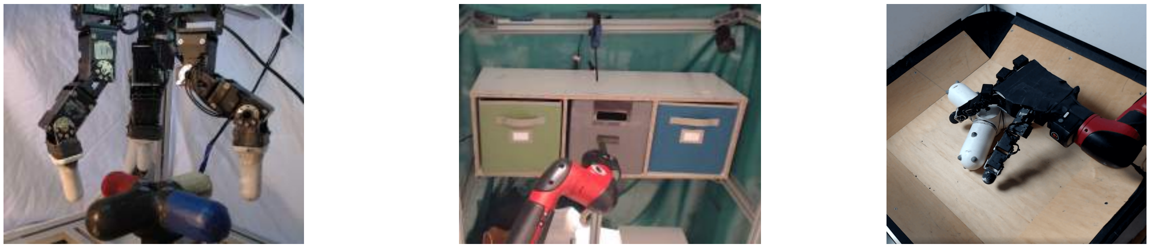


Figure 18.41: Full views of the robot hardware setups. Videos are available at *awacrl.github.io*

- 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 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.

18.7 Appendix G: Appendix for Chapter 13

Training details

Hyperparameters

General	
Standard deviation update coefficient	**0.99**
Image Sizes	[(16, 16, 3), **(32, 32, 3)**, (64, 64, 3)]
SAC	
Learning Rate	**3e-4**
γ	**0.99**
Batch Size	**256**
Convnet Filters	[**(64, 64, 64)**, (16, 32, 64)]
Stride	**(2, 2)**
Kernel Sizes	**(3, 3)**
Pooling	[MaxPool2D, **None**]
Actor/Critic FC Layers	[**(512, 512)**, (256, 256, 256)]
VICE	
n_{VICE}	[1, **5**, 10]
Batch Size	**128**
Learning Rate	**1e-4**
Mixup α	**Uniform(0, 1)**
Convnet Filters	[**(64, 64, 64)**, (16, 32, 64)]
Stride	**(2, 2)**
Kernel Sizes	**(3, 3)**
Pooling	[MaxPool2D, **None**]
FC Layers	[**(512, 512)**, (256, 256, 256)]

RND	
Learning Rate	**3e-4**
Batch Size	**256**
Convnet Filters	**(16, 32, 64)**
Stride	**(2, 2)**
Kernel Sizes	**(3, 3)**
Pooling	[MaxPool2D, **None**]
FC Layers	[**(512, 512)**, (256, 256, 256)]
VAE	
Learning Rate	**1e-4**
Batch Size	**256**
Encoder (Convnet) Filters	**(64, 64, 32)**
Latent Dimension	[8, **16, 32**, 64]
β	[1e-3, 0.1, **0.5**, 1, 10]
Stride	**(2, 2)**
Kernel Sizes	**(3, 3)**
Pooling	[MaxPool2D, **None**]

The ranges of values listed above represent the hyperparameters we searched over, and the bolded values are what we use in the Section 13.7 experiments.

VICE

We use a variant of VICE which defines the reward as the logits of the classifier, notably omitting the $-log(\pi(a|s))$ term. We also regularize our classifier with mixup [431]. We train all of our experiments using 200 goal images, which takes under an hour to collect in the real world for each task.

Random Network Distillation (RND)

We found it important to normalize the predictor errors, just as [110] did.

VAE

We train a standard beta-VAE to maximize the evidence lower bound, given by:

$$\mathbb{E}_{z \sim q_\phi(z|x)}[p_\theta(x|z)] - \beta D_{\text{KL}}(q_\phi(z|x) \,||\, p_\theta(z))$$

To collect training data, we sampled random states in the observation space. In the real world, this sampling can be replaced with training an exploratory policy (i.e. using the RND reward as the policy's only objective). The learned weights of the encoder of the VAE are frozen, and the latent input is used to train the policy for reset-free RL.

Task details

Simulated Tasks

We evaluated our system across three tasks in simulation: bead manipulation, valve rotation, free object repositioning.

Bead Manipulation The bead manipulation task involves an abacus rod with four beads that can slide freely. The goal is to position two beads on each end from any initial configuration of beads. This can take the form of sliding one bead over (if three beads start on one side), two beads over (if all four beads start on one side), splitting beads apart (all four beads start in the middle), or some intermediate combination of those. The true reward is defined as the mean goal distance of all four beads. Policies are evaluated starting from the 8 initial configurations depicted in Fig 18.42. Evaluation performance reported in Section 13.7 for this task is defined as the final reward averaged across the 8 evaluation rollouts.

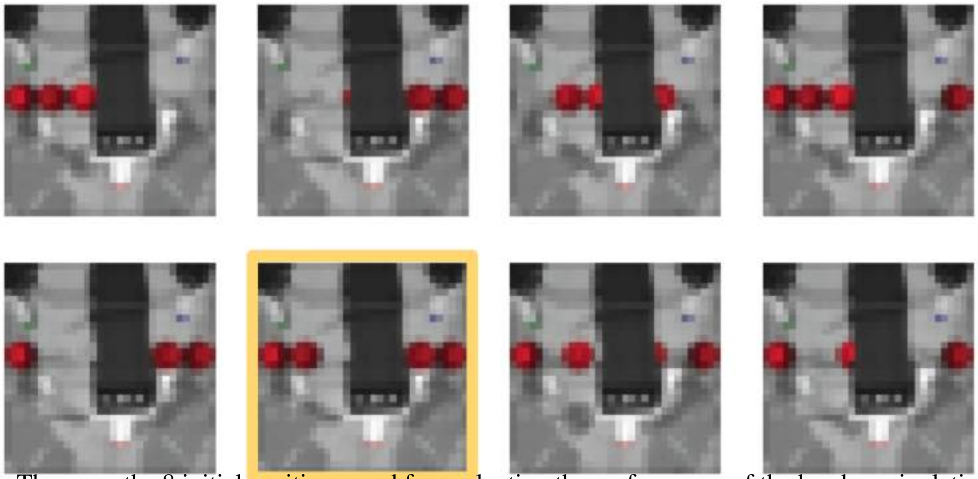


Figure 18.42: These are the 8 initial positions used for evaluating the performance of the bead manipulation policy. The goal configuration (which is also an initial evaluation position) is highlighted in yellow.

Valve Rotation The claw is positioned above a three pronged valve (15 cm in diameter). The objective is to turn the valve to a given orientation from any initial orientation. The "true reward" is $r = -\log(|\theta_{state} - \theta_{goal}|)$. Policies are evaluated starting from the 8 initial configurations depicted in Fig 18.43. Evaluation performance reported in Section 13.7 for this task is defined as the final orientation distance averaged across the 8 evaluation rollouts.

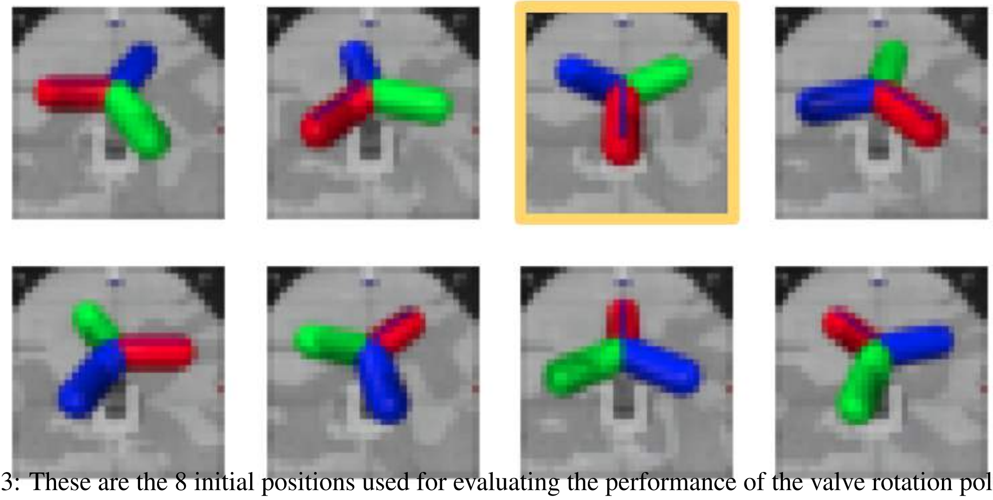


Figure 18.43: These are the 8 initial positions used for evaluating the performance of the valve rotation policy. The goal configuration (which is also an initial evaluation position) is highlighted in yellow.

Free Object Repositioning The claw is positioned atop a free (6 DoF) three pronged object (15cm in diameter), which can translate and rotate within a $30\text{cm}x30\text{cm}$ box. The goal is specified by a xy-position as well as a z-angle, where the xy-plane is the plane of the arena. The true reward is defined as the weighted sum of the angular and translational distances, $r = -2\log(||[x_{state}, y_{state}] - [x_{goal}, y_{goal}]||_2) - \log(|\theta_{state} - \theta_{goal}|)$. In our experiments, $(x, y, \theta)_{goal} = (0, 0, -\frac{\pi}{2})$, where the origin is centered in the arena. Policies are evaluated starting from the 15 initial configurations depicted in Figure 18.44. Evaluation performance reported in Section 13.7 for this task is defined as the final pose distance ($\frac{||[x_{final}, y_{final}] - [x_{goal}, y_{goal}]||_2}{0.25\text{ m}} + \frac{|\theta_{final} - \theta_{goal}|}{\pi\text{ rad}}$) averaged across the 15 evaluation rollouts.

In our reset controller experiments, we averaged evaluation performance over three different choices of reset states, where the first reset state is always the goal:

1. $(x, y, \theta)_{reset,1} = (x, y, \theta)_{goal}, (x, y, \theta)_{reset,2} = (0.05, 0.05, \frac{\pi}{2})$
2. $(x, y, \theta)_{reset,1} = (x, y, \theta)_{goal}, (x, y, \theta)_{reset,2} = (0, 0, -\frac{\pi}{6})$
3. $(x, y, \theta)_{reset,1} = (x, y, \theta)_{goal}, (x, y, \theta)_{reset,2} = (-0.04, -0.04, -\frac{\pi}{2})$

Real World Tasks

For each setup we use an RGB camera to get images. We execute actions on the DClaw at 10Hz. In order to operate at such a high frequency while also training from images we sample and train

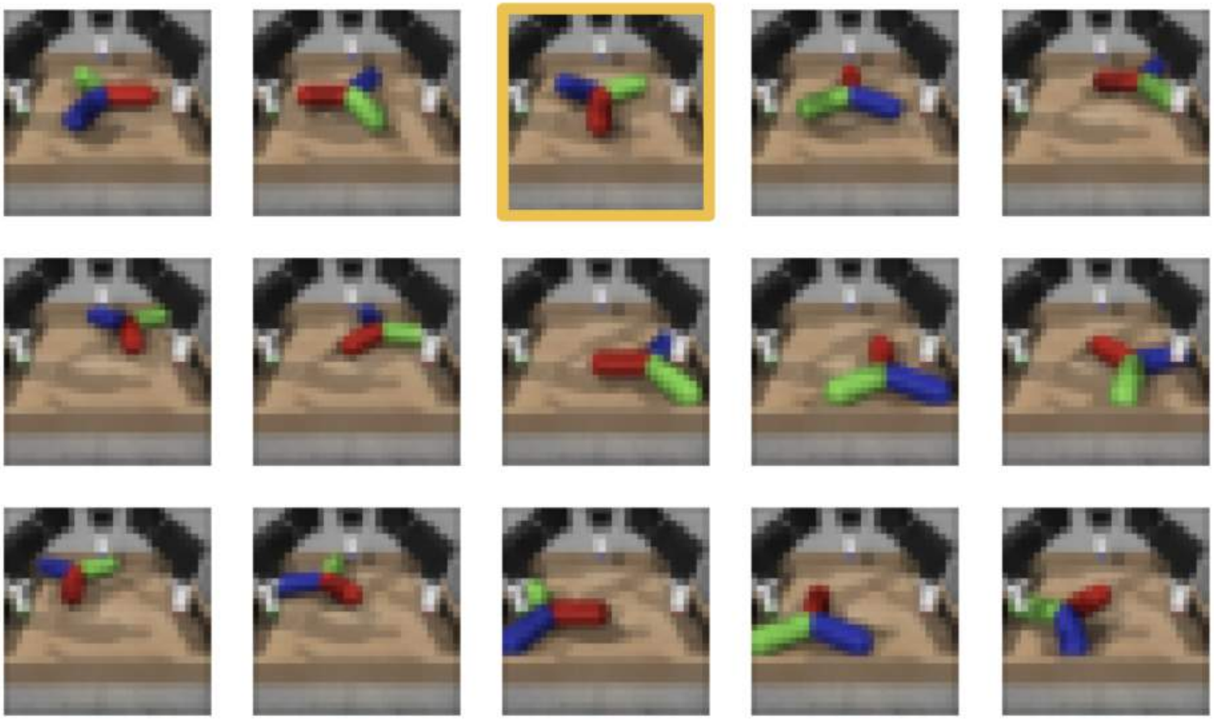
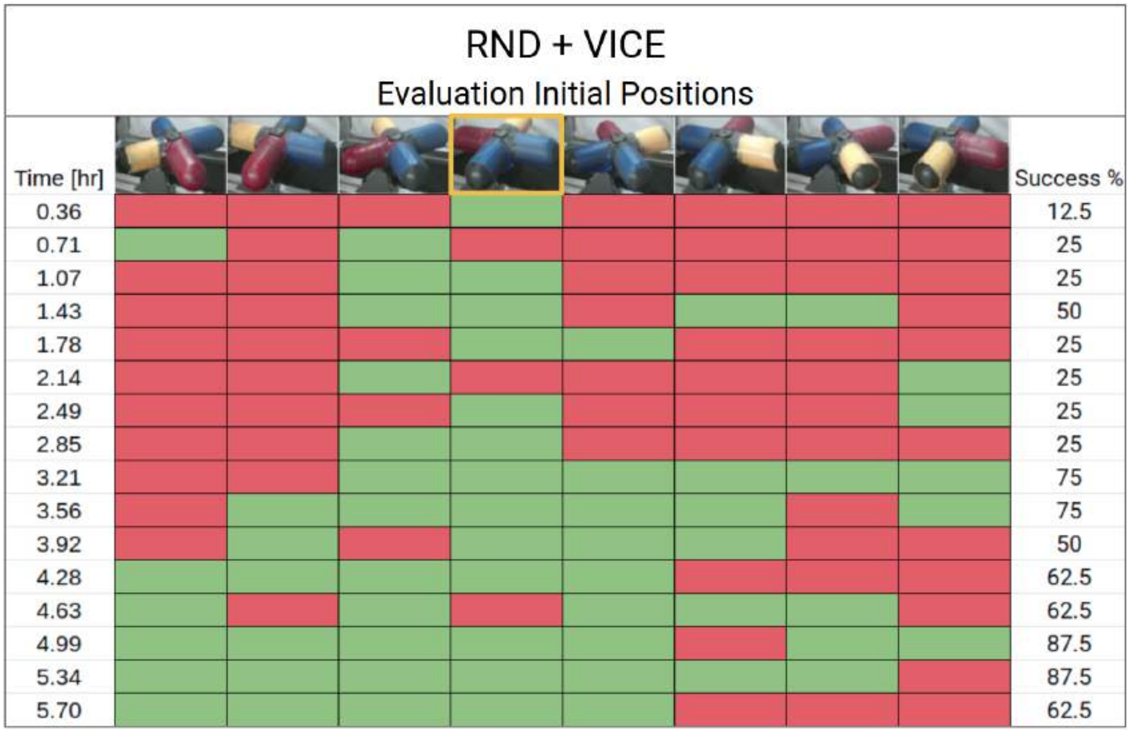


Figure 18.44: These are the 15 initial positions used for evaluating the performance of the free object repositioning policy. The goal configuration $(x, y, \theta)_{goal}$ which is also an initial evaluation position is highlighted in yellow.

asynchronously, but limit training to not exceed two gradient steps per transition sampled in the real world. Since direct performance metrics cannot be measured during training due to the lack of object instrumentation, evaluations of performance are done post-training.

Valve Rotation The task is identical to the one in simulation. Evaluations were done post-training. An evaluation trajectory was defined as a success if at the last step, the valve was within 15 degrees of the goal. Each policy was evaluated over 8 rollouts, with initial configurations evenly spaced out between at increments of 45 degrees. Results are reported in Figures 18.45, 18.46.

Bead Manipulation The rod is 22cm in length, and each bead measures 3.5cm in diameter. Evaluations were done post-training, using the following procedure: the environment was manually reset to each of the 8 specified configurations shown in Figures 18.47 and 18.48 (which cover a full range of the state space) at the start of each evaluation rollout. An evaluation trajectory was defined as a success if at the last time step, all beads were within 2cm of their goal positions. Performance was evaluated at around 20 hours, at which point the policy achieved greater than 80% success on the 10 evaluation rollouts (a random policy achieved a success rate of 10%). Results are reported in Figs 18.47, 18.48.

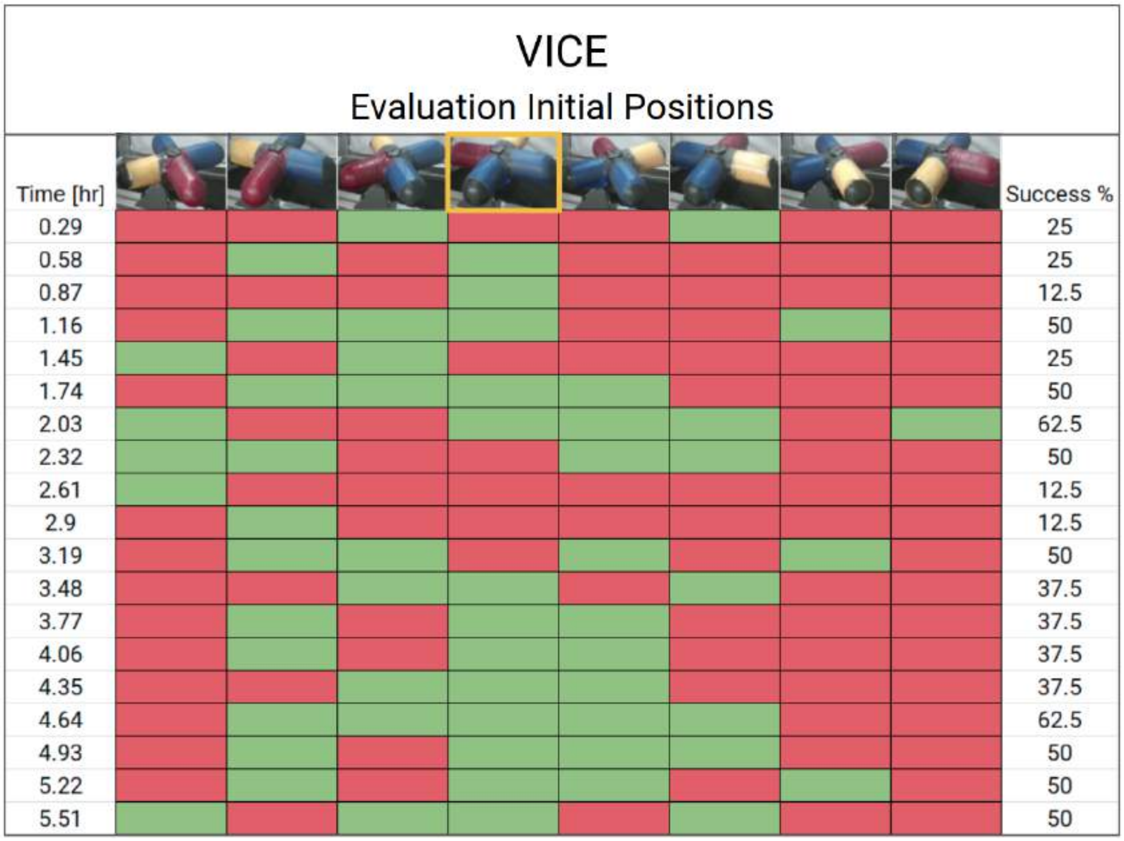


RND + VICE

Evaluation Initial Positions

Time [hr]									Success %
0.36	🟥	🟥	🟥	🟩	🟥	🟥	🟥	🟥	12.5
0.71	🟩	🟥	🟩	🟥	🟥	🟥	🟥	🟥	25
1.07	🟥	🟥	🟩	🟩	🟥	🟥	🟥	🟥	25
1.43	🟥	🟥	🟩	🟩	🟥	🟩	🟩	🟥	50
1.78	🟥	🟥	🟥	🟩	🟩	🟥	🟥	🟥	25
2.14	🟥	🟥	🟩	🟥	🟥	🟥	🟥	🟩	25
2.49	🟥	🟥	🟥	🟩	🟥	🟥	🟥	🟩	25
2.85	🟥	🟥	🟩	🟩	🟥	🟥	🟥	🟥	25
3.21	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	75
3.56	🟥	🟩	🟩	🟩	🟩	🟩	🟥	🟩	75
3.92	🟥	🟩	🟥	🟩	🟩	🟩	🟥	🟥	50
4.28	🟩	🟩	🟩	🟩	🟩	🟥	🟥	🟥	62.5
4.63	🟩	🟥	🟩	🟥	🟩	🟩	🟩	🟥	62.5
4.99	🟩	🟩	🟩	🟩	🟩	🟥	🟩	🟩	87.5
5.34	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟥	87.5
5.70	🟩	🟩	🟩	🟩	🟩	🟥	🟥	🟥	62.5

Figure 18.45: These are the results of the evaluation rollouts on the valve rotation task in the real world using our method (without the VAE). Trained policies were saved at regular intervals and evaluated post-training. Each row is a different policy, and each column an evaluation rollout from a different initial configuration. The goal is highlighted in yellow. Our method is able to achieve high success rates after 5 hours of training.

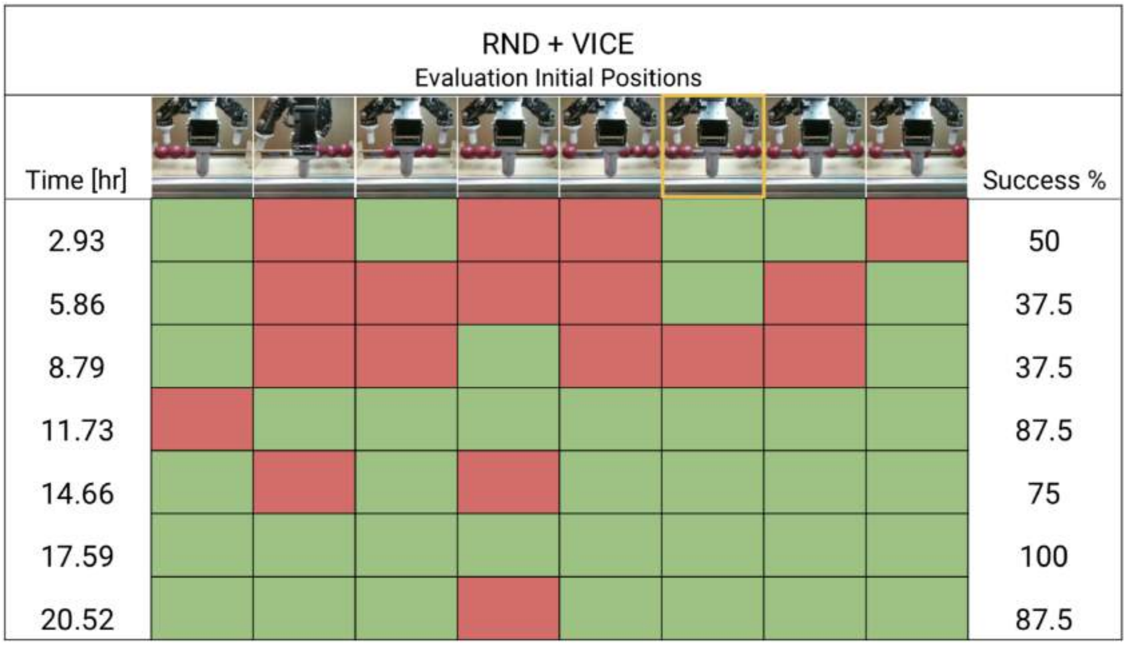


VICE

Evaluation Initial Positions

Time [hr]									Success %
0.29	🟥	🟥	🟩	🟥	🟥	🟩	🟥	🟥	25
0.58	🟥	🟩	🟥	🟩	🟥	🟥	🟥	🟥	25
0.87	🟥	🟥	🟥	🟩	🟥	🟥	🟥	🟥	12.5
1.16	🟥	🟩	🟩	🟩	🟥	🟥	🟩	🟥	50
1.45	🟩	🟥	🟩	🟥	🟥	🟥	🟥	🟥	25
1.74	🟥	🟩	🟩	🟩	🟩	🟥	🟥	🟥	50
2.03	🟩	🟥	🟥	🟩	🟩	🟩	🟥	🟩	62.5
2.32	🟩	🟩	🟥	🟥	🟩	🟩	🟥	🟥	50
2.61	🟩	🟥	🟥	🟥	🟥	🟥	🟥	🟥	12.5
2.9	🟥	🟩	🟥	🟥	🟥	🟥	🟥	🟥	12.5
3.19	🟥	🟩	🟩	🟥	🟩	🟥	🟩	🟥	50
3.48	🟥	🟥	🟩	🟩	🟥	🟩	🟥	🟥	37.5
3.77	🟥	🟩	🟥	🟩	🟩	🟥	🟥	🟥	37.5
4.06	🟥	🟩	🟥	🟩	🟩	🟥	🟥	🟥	37.5
4.35	🟥	🟥	🟩	🟩	🟩	🟥	🟥	🟥	37.5
4.64	🟥	🟩	🟩	🟩	🟩	🟩	🟥	🟥	62.5
4.93	🟥	🟩	🟥	🟩	🟩	🟩	🟥	🟥	50
5.22	🟥	🟩	🟥	🟩	🟩	🟥	🟩	🟥	50
5.51	🟩	🟥	🟩	🟩	🟥	🟩	🟥	🟥	50

Figure 18.46: These are the results of evaluation rollouts on the valve rotation task in the real world using the VICE single goal baseline. The policies fail to evaluate well, especially from initial positions far from the goal position.

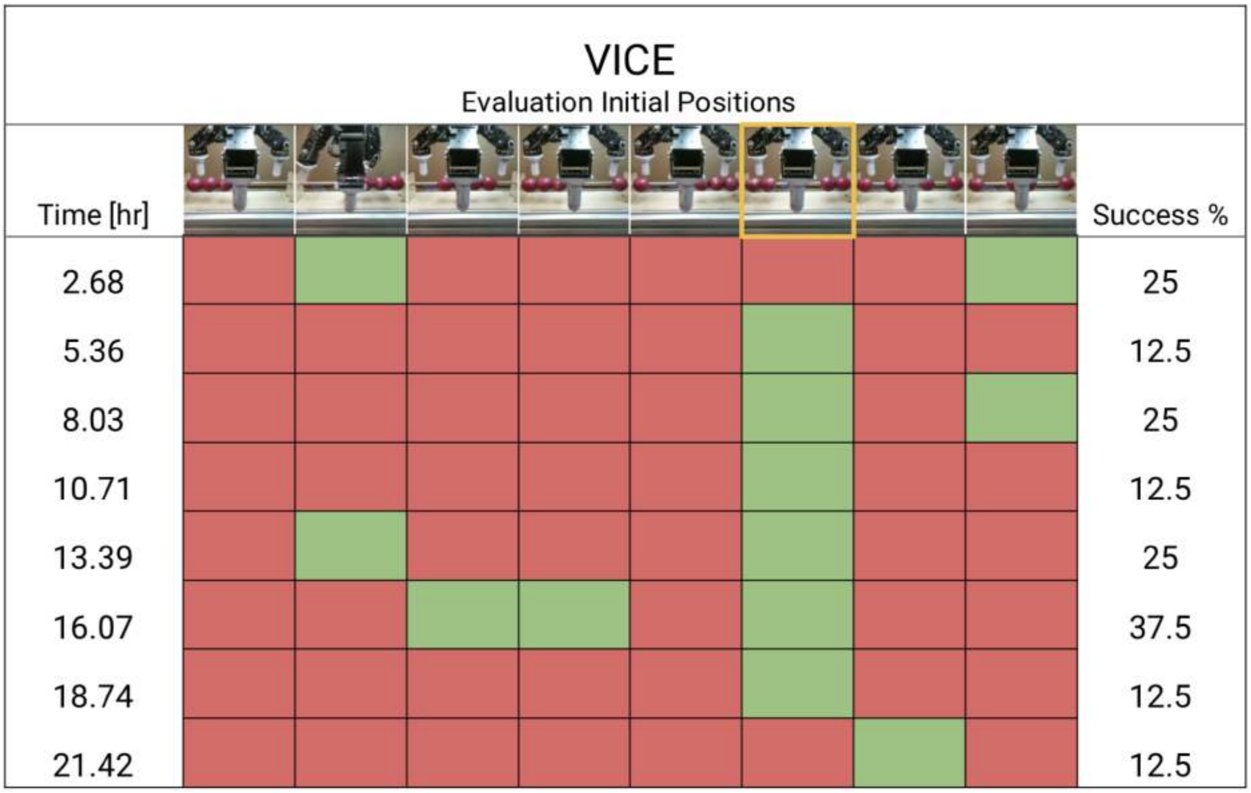


RND + VICE

Evaluation Initial Positions

Time [hr]									Success %
2.93	🟩	🟥	🟩	🟥	🟥	🟩	🟩	🟥	50
5.86	🟩	🟥	🟥	🟥	🟥	🟩	🟥	🟩	37.5
8.79	🟩	🟥	🟥	🟩	🟥	🟥	🟥	🟩	37.5
11.73	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	87.5
14.66	🟩	🟥	🟩	🟥	🟩	🟩	🟩	🟩	75
17.59	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	100
20.52	🟩	🟩	🟩	🟥	🟩	🟩	🟩	🟩	87.5

Figure 18.47: These are the results of the evaluation rollouts on the valve rotation task in the real world using our method (without the VAE). Trained policies were saved at regular intervals and evaluated post-training. Each row is a different policy, and each column an evaluation rollout from a different initial configuration. The goal is highlighted in yellow. Our method is able to achieve high success rates after 17 hours of training.



VICE

Evaluation Initial Positions

Time [hr]									Success %
2.68	🟥	🟩	🟥	🟥	🟥	🟥	🟥	🟩	25
5.36	🟥	🟥	🟥	🟥	🟥	🟩	🟥	🟥	12.5
8.03	🟥	🟥	🟥	🟥	🟥	🟩	🟥	🟩	25
10.71	🟥	🟥	🟥	🟥	🟥	🟩	🟥	🟥	12.5
13.39	🟥	🟩	🟥	🟥	🟥	🟩	🟥	🟥	25
16.07	🟥	🟥	🟩	🟩	🟥	🟩	🟥	🟥	37.5
18.74	🟥	🟥	🟥	🟥	🟥	🟩	🟥	🟥	12.5
21.42	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟥	12.5

Figure 18.48: These are the results of evaluation rollouts on the valve rotation task in the real world using the VICE single goal baseline. The policies fail to evaluate consistently, except when the initial configuration matches the goal configuration.

18.8 Appendix H: Appendix for Chapter 14

Appendix A. Reward Functions and Additional Task Details

In-Hand Manipulation Tasks on Hardware

The rewards for this family of tasks are defined as follows. θ^x represent the Sawyer end-effector's wrist euler angle, x, y, z represent the object's 3D position, and $\hat{\theta}^z$ represents the object's z euler angle. x_{goal} and others represent the task's goal position or angle. The threshold for determining whether the object has been lifted is subjected to the real world arena size. We set it to 0.1m in our experiment.

$$R_{recenter} = -3|| \begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} x_{goal} \\ y_{goal} \end{bmatrix} || - || \begin{bmatrix} x \\ y \\ z \end{bmatrix} - \begin{bmatrix} x_{hand} \\ y_{hand} \\ z_{hand} \end{bmatrix} ||$$

$$R_{lift} = -|z - z_{goal}|$$

$$\begin{aligned} R_{flipup} = & -5|\theta^x - \theta^x_{goal}| - 50(\mathbb{1}\Big\{z < \text{threshold}\Big\}) + \\ & 10(\mathbb{1}\Big\{|\theta^x - \theta^x_{goal}| < 0.15 \text{ AND } z > \text{threshold}\Big\}) \end{aligned}$$

$$R_{reorient} = -|\hat{\theta}^z - \hat{\theta}^z_{goal}|$$

A rollout is considered a success (as reported in the figures) if it reaches a state where the valve is in-hand and flipped facing up:

$$z > \text{threshold AND } |\theta^x - \theta^x_{goal}| < 0.15$$

Pipe Insertion Tasks on Hardware

The rewards for this family of tasks are defined as follows. x, y, z represent the object's 3D position, and q represent the joint positions of the D'Hand. x_{goal} and others represent the task's goal position or angle. The threshold for determining whether the object has been lifted is subjected to the real world arena size. We set it to 0.1m in our experiment. To reduce collision in real world experiments, we have two tasks for insertion. One approaches the peg and the other attempts insertion.

$$R_{recenter} = -3|| \begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} x_{goal} \\ y_{goal} \end{bmatrix} || - || \begin{bmatrix} x \\ y \\ z \end{bmatrix} - \begin{bmatrix} x_{hand} \\ y_{hand} \\ z_{hand} \end{bmatrix} ||$$

$$R_{lift} = -2|z - z_{goal}| - 2|q - q_{goal}|$$

$$R_{Insert1} = -d_1 + 10(\mathbb{1}\Big\{d_1 < 0.1\Big\})$$

$$R_{Insert2} = -d_2 + 10(\mathbb{1}\Big\{d_2 < 0.1\Big\})$$

$where$

$$d_1 = || \begin{bmatrix} x \\ y \\ z \end{bmatrix} - \begin{bmatrix} x_{goal1} \\ y_{goal1} \\ z_{goal1} \end{bmatrix} ||$$

$$d_2 = || \begin{bmatrix} x \\ y \\ z \end{bmatrix} - \begin{bmatrix} x_{goal2} \\ y_{goal2} \\ z_{goal2} \end{bmatrix} ||$$

$$R_{Remove} = -|| \begin{bmatrix} x \\ y \\ z \end{bmatrix} - \begin{bmatrix} x_{arena\_center} \\ y_{arena\_center} \\ z_{arena\_center} \end{bmatrix} ||$$

A rollout is considered a success (as reported in the figures) if it reaches a state where the valve is in-hand and pipe is inserted:

$$z > \text{threshold AND } d_2 < 0.05$$

Lightbulb Insertion Tasks in Simulation

The rewards for this family of tasks include $R_{recenter}, R_{pickup}, R_{flipup}$ defined in the previous section, as well as the following bulb insertion reward:

$$
\begin{aligned}
R_{bulb} = &- || \begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} x_{goal} \\ y_{goal} \end{bmatrix} || - 2(|z - z_{goal}|) + \\
&\nVdash \Big\{ || \begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} x_{goal} \\ y_{goal} \end{bmatrix} || < 0.1 \Big\} \\
&+ \\
&10(\nVdash \Big\{ || \begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} x_{goal} \\ y_{goal} \end{bmatrix} || < 0.1 \text{ AND } |z - z_{goal}| < 0.1 \Big\}) - \\
&\nVdash \Big\{ z < \text{threshold} \Big\}
\end{aligned}
$$

A rollout is considered a success (as reported in the figures) if it reaches a state where the bulb is positioned very close to the goal position in the lamp:

$$
|| \begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} x_{goal} \\ y_{goal} \end{bmatrix} || < 0.1 \text{ AND } |z - z_{goal}| < 0.1
$$

Basketball Tasks in Simulation

The rewards for this family of tasks include $R_{recenter}, R_{pickup}$ defined in the previous section, as well as the following basket dunking reward:

$$
\begin{aligned}
R_{basket} = -\,|| \begin{bmatrix} x \\ y \\ z \end{bmatrix} - \begin{bmatrix} x_{goal} \\ y_{goal} \\ z_{goal} \end{bmatrix} ||+ \\
20(\nVdash\Big\{|| \begin{bmatrix} x \\ y \\ z \end{bmatrix} - \begin{bmatrix} x_{goal} \\ y_{goal} \\ z_{goal} \end{bmatrix} || < 0.2\Big\}) \\
+\,50(\nVdash\Big\{|| \begin{bmatrix} x \\ y \\ z \end{bmatrix} - \begin{bmatrix} x_{goal} \\ y_{goal} \\ z_{goal} \end{bmatrix} || < 0.1\Big\})- \\
\nVdash\Big\{z < \text{threshold}\Big\}
\end{aligned}
$$

A rollout is considered a success (as reported in the figures) if it reaches a state where the ball is positioned very close to the goal position above the basket:

$$
|| \begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} x_{goal} \\ y_{goal} \end{bmatrix} || < 0.1 \text{ AND } |z - z_{goal}| < 0.15
$$

Appendix B. Additional Domains

In addition to the test domains described in Section 14.3, we also tested our method in simulation on simpler tasks such as a 2D “pincer” and a simplified lifting task on the Sawyer and “D’Hand” setup. The pincer task is described in Figure 18.49. Figure 18.49 shows the performance of our method as well as baseline comparisons.

Appendix C. Hyperparameter Details

Appendix D. Task Graph Details

We provide some details of the task graphs for every domain below.

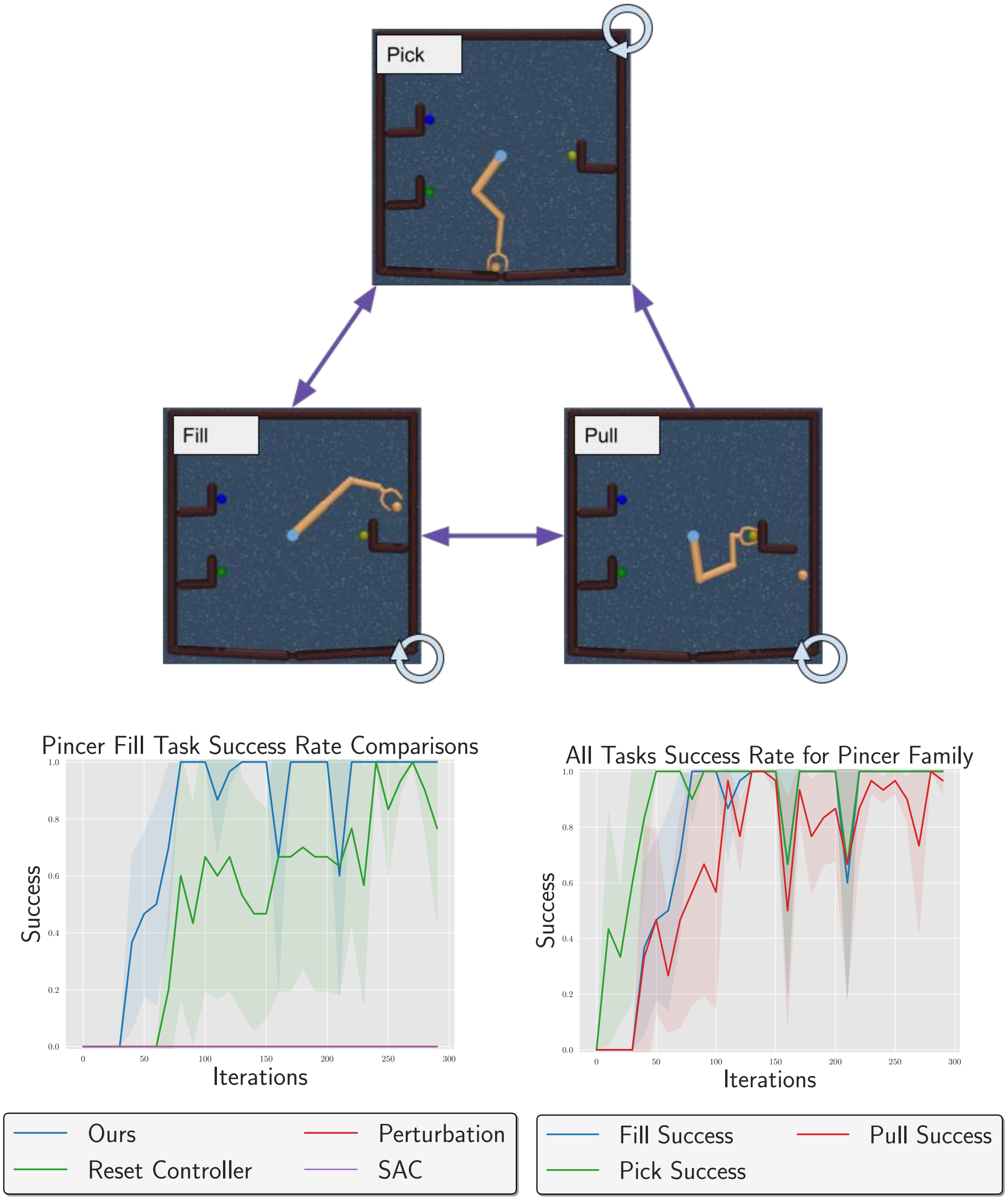


Figure 18.49: Pincer domain - object grasping, filling the drawer with the object, pulling open the drawer. These tasks naturally form a cycle - once an object is picked up, it can be filled in the drawer, following which the drawer can be pulled open and grasping and filling can be practiced again.

SAC	
Learning Rate	3×10^{-4}
Discount Factor γ	0.99
Policy Type	Gaussian
Policy Hidden Sizes	$(512, 512)$
RL Batch Size	1024
Reward Scaling	1
Replay Buffer Size	$500,000$
Q Hidden Sizes	$(512, 512)$
Q Hidden Activation	ReLU
Q Weight Decay	0
Q Learning Rate	3×10^{-4}
Target Network τ	5×10^{-3}

Table 18.14: Hyperparameters used across all domains.

Algorithm 15: n-Hand Manipulation Task Graph (Hardware)

```
Require: Euclidean coordinates of object $q$, Sawyer wrist angle $\theta$, previous task $\phi$
$is\_lifted = q_z > 0.15$
$is\_upright = |\theta - \theta_{upright}| < 0.1$
$not\_centered = |q - q_{center}| > 0.1$
if $is\_upright$ and $is\_lifted$ then
    return $Inhand$
else if $is\_lifted$ then
    return $Flipup$
else if $not\_centered$ and $\phi = Recenter$ then
    return $Perturb$
else if $not\_centered$ then
    return $Recenter$
else
    return $Lift$
end if
```

Algorithm 16: Pipe Insertion Task Graph (Hardware)

```
Require: Euclidean coordinates of object $q$, a waypoint close to peg $q_{waypoint}$, previous task $\phi$
$is\_lifted = q_z > 0.15$
$is\_inserted = |q - q_{inserted}| < 0.05$
$close\_to\_waypoint = |q - q_{waypoint}| < 0.05$
$not\_centered = |q - q_{center}| > 0.1$
if $is\_inserted$ then
    return $Remove$
else if $close\_to\_waypoint$ then
    return $Insert2$
else if $is\_lifted$ then
    return $Insert1$
else if $not\_centered$ and $\phi = Recenter$ then
    return $Perturb$
else if $not\_centered$ then
    return $Recenter$
else
    return $Lift$
end if
```

Algorithm 17: Lightbulb Insertion Task Graph (Simulation)

```
Require: Object position $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$, Sawyer wrist angle (its $x$ Euler angle) $\theta^x$, previous task $\phi$
Let $\begin{bmatrix} x_{center} \\ y_{center} \end{bmatrix}$ be the center coordinates of the arena (relative to the Sawyer base).
Let $z_{threshold}$ be the height (in meters) above the arena that we consider the object to be “picked up.”
$is\_centered = || \begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} x_{center} \\ y_{center} \end{bmatrix} || < 0.1$
$is\_lifted = z > z_{threshold}$
$is\_facing\_up = |\theta^x - \theta^x_{upright}| < 0.1$
if NOT $is\_centered$ and NOT $is\_lifted$ then
    if $\phi =$ Recenter then
        return Perturb
    else
        return Recenter
    end if
else if $is\_centered$ and NOT $is\_lifted$ then
    return Lift
else if $is\_lifted$ and NOT $is\_facing\_up$ then
    return Flip Up
else if $is\_lifted$ and $is\_facing\_up$ then
    return Lightbulb Insertion
end if
```

Algorithm 18: Basketball Task Graph (Simulation)

```
Require: Object position $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$, previous task $\phi$
Let $\begin{bmatrix} x_{center} \\ y_{center} \end{bmatrix}$ be the coordinates of the arena where we want to pick up the ball, such that it is out of the way of the hoop (relative to the Sawyer base).
Let $\theta^x_{upright}$ be the wrist angle (in radians) that we want the hand to be facing. ($\theta^x_{upright} = \pi$ in our instantiation).
Let $z_{threshold}$ be the height (in meters) above the arena that we consider the object to be “picked up.”
$is\_centered = || \begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} x_{center} \\ y_{center} \end{bmatrix} || < 0.1$
$is\_lifted = z > z_{threshold}$
if NOT $is\_centered$ and NOT $is\_lifted$ then
    if $\phi =$ Recenter then
        return Perturb
    else
        return Recenter
    end if
else if $is\_centered$ and NOT $is\_lifted$ then
    return Lift
else if $is\_lifted$ then
    return Basketball Dunking
end if
```

18.9 Appendix I: Appendix for Chapter 15

Hyperparameters and Model Architectures

Here we list a set of general hyperparameters and details of the model training. If indiciated as a list, we performed a grid search over those values and the underlined value is the chosen one to report in the paper.

As mentioned above, we use a standard 2 hidden layer MLP to represent policies and value functions. The state space consists of the position of the end-effector, euler rotation of the end effector, position of the various elements in the scene and a representation of the goal. In the hardware experiments, the representation of the goals of interest is a continuous vector of the position of the end-effector, euler rotation of the end effector, position of the various elements in the scene for a particular goal of interest. In the simulation environments, this is a discrete one-hot vector representing the ID of the particular goal of interest being commanded. Beyond the experiments mentioned in the main paper, we also ran experiments resetting every 50 episodes instead of 10, and achieved 96% success rate as well.

Reward Functions

We used a generic and simple reward function for different goals s_g, as follows:

$$r(s, a, s_g) = -20 * \|x_{\text{ee}} - x_{\text{element}}\|_2 - 20 * \|\theta_{\text{element}} - \theta_{\text{goal}}\|_2 \tag{18.27}$$

Here x_{ee} corresponds to the position of the end effector, x_{element} corresponds to the position of the particular element (cabinet, slider, knob) that has to be moved to accomplish the particular goal s_g, θ_{element} is the current position of the above-mentioned element and θ_{goal} is the goal position of the element being manipulated. This reward function essentially encourages the arm to move towards the element of interest and then manipulate it towards it's goal position. This reward function is suitable for articulated objects, but may have to be replaced by a more involved reward function for scenes with more free objects.

Baseline Details

We provide some further details of the baseline methods below:

Non-pretrained, graph search task selection This baseline simply uses the exact same graph search algorithm at the high level but starts the low level policy completely from scratch and trains the algorithm exactly the same way as DBAP , using exactly the same hyperparameters.

Pretrained low-level, random high level controller This baseline uses completely random goal selection both during practicing and long horizon task accomplishment. It simply samples a goal of interest from the set of all possible goals of interest

Pretrained low-level, BC task selection This baseline uses a behavior cloned high level model to select goals, where the high level task selector $q(s_g|s, s_g^{\text{desired}})$ is trained using behavior cloning

AWAC	
Learning Rate	3×10^{-4}
Discount Factor γ	0.99
Policy Type	Gaussian
Policy Hidden Sizes	$(256, 256)$
Policy Hidden Activation	ReLU
Policy Weight Decay	10^{-4}
Policy Learning Rate	3×10^{-4}
RL Batch Size	1024
Reward Scaling	1
Replay Buffer Size	$500,000$
Q Hidden Sizes	$(512, 512)$
Q Hidden Activation	ReLU
Q Weight Decay	0
Q Learning Rate	3×10^{-4}
Target Network τ	5×10^{-3}
Temperature (β)	$30.$
Epoch Length	2000
Path Length	200
Max High Level Steps	6
Standard Deviation Lower Bound	10^{-5}
Pretraining Steps	30000
Behavior Cloning	
Hidden Layer Sizes	$(256, 256)$
Training Steps	30000

Table 18.15: Hyperparameters used for experiments

on human provided data, and goals are selected using this task selector. In particular, we relabel sequences of goals of interest visited in the data within a particular window (as outlined in [346]) to generate $(s, s_g, s_g^{\text{desired}})$ which can then be used to train $q(s_g|s, s_g^{\text{desired}})$ via supervised learning. During autonomous practicing, goals are chosen by choosing a s_g^{desired} at random and then sampling a next goal from the behavior cloned $q(s_g|s, s_g^{\text{desired}})$ conditioned on the chosen s_g^{desired}. At long horizon execution time, the agent simply sets s_g^{desired} to the desired long horizon goal and sample from $q(s_g|s, s_g^{\text{desired}})$.

Pretrained low-level, reset controller This baseline alternates between setting the first goal of interest as the goal and sampling a goal of interest at random and setting this at the goal at the high level.

All of these baselines were implemented using exactly the same underlying algorithm and the same hyperparameters as mentioned above.

Visualizations and Ablations

To better understand the behavior of our method, we next visualize the sequence of tasks proposed by our graph search algorithm during evaluation time as compared to the behavior cloning baseline mentioned above. The behavior cloning baseline trains the high level with goal conditioned behavior cloning while graph search simply builds a graph of feasible edges and performs search. We find that when we run an idealized experiment, where the multi-task policies are assumed to be perfect, the effective path length obtained by BC is significantly higher than graph search. This suggests that doing high level learning with relabeled BC as suggested in [249], [346] is prone to an issue of cycles where it is not trained to take the shortest path if the training data is cyclic (as is common in play data). Graph search on the other hand, avoids these issues and is able to find a shortest path to the goal easily as seen in Fig 18.50.

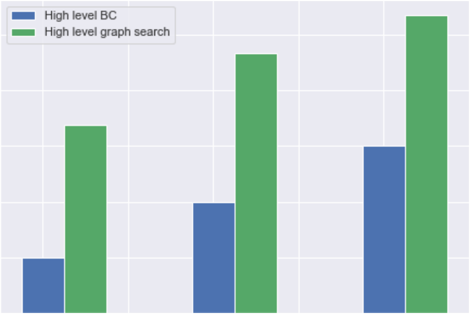


Figure 18.50: Comparison of path length of BC vs graph search for a simulated problem assuming perfect low level. We see that high level BC can often be ineffective at learning short paths to a goal

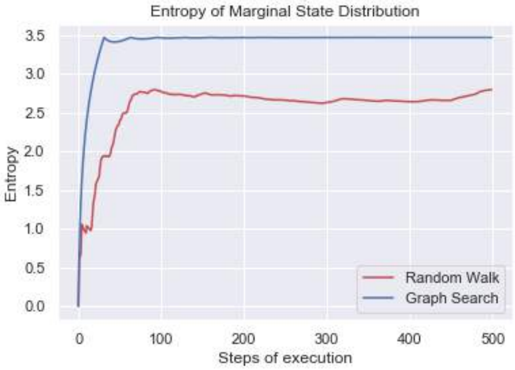


Figure 18.51: Visualization of entropy on a 32 state chain MDP. We see that the entropy of the marginal distribution over states is much higher with graph search than a random walk.

To further understand the behavior of graph search based practicing autonomously, we ran an isolated analysis experiment on a very simple chain MDP environment to understand the performance of the graph search algorithm in terms of task visitations and entropy over the distribution of tasks, as compared to simply performing a random walk on tasks. We visualize these results in

Fig 18.51, where entropy of the marginal distribution of states of interest is plotted against steps of the training process. We find that while both have increasing entropy, graph based search has much higher entropy as it maintains a uniform likelihood over states, providing the coverage needed to achieve good performance in the evaluations in Fig 18.51.