Title
Exploration and Safety in Deep Reinforcement Learning

Author
Achiam, Joshua

Publication Date
2021

Peer reviewed|Thesis/dissertation

Exploration and Safety in Deep Reinforcement Learning

by

Joshua S Achiam

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering - Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Pieter Abbeel, Co-Chair
Professor Shankar Sastry, Co-Chair
Professor Jack Gallant

Spring 2021

Abstract

Exploration and Safety in Deep Reinforcement Learning

by

Joshua S Achiam

Doctor of Philosophy in Engineering - Electrical Engineering and Computer Science

University of California, Berkeley

Professor Pieter Abbeel, Co-Chair

Professor Shankar Sastry, Co-Chair

Reinforcement learning (RL) agents need to explore their environments in order to learn optimal policies by trial and error. However, exploration is challenging when reward signals are sparse, or when safety is a critical concern and certain errors are unacceptable. In this thesis, we address these challenges in the deep reinforcement learning setting by modifying the underlying optimization problem that agents solve, incentivizing them to explore in safer or more-efficient ways.

In the first part of this thesis, we develop methods for intrinsic motivation to make progress on problems where rewards are sparse or absent. Our first approach uses an intrinsic reward to incentivize agents to visit states considered surprising under a learned dynamics model, and we show that this technique performs favorably compared to naive exploration. Our second approach uses an objective based on variational inference to endow agents with multiple skills that are distinct from each other, without the use of task-specific rewards. We show that this approach, which we call variational option discovery, can be used to learn locomotion behaviors in simulated robot environments.

In the second part of this thesis, we focus on problems in safe exploration. Building on a wide range of prior work on safe reinforcement learning, we propose to standardize constrained RL as the main formalism for safe exploration; we then proceed to develop algorithms and benchmarks for constrained RL. Our presentation of material tells a story in chronological order: we begin by presenting Constrained Policy Optimization (CPO), the first algorithm for constrained deep RL with guarantees of near-constraint satisfaction at each iteration. Next, we develop the Safety Gym benchmark, which allows us to find the limits of CPO and inspires us to press in a different direction. Finally, we develop PID Lagrangian methods, where we find that a small modification to the Lagrangian primal-dual gradient baseline approach results in significantly improved stability and robustness in solving constrained RL tasks in Safety Gym.

Contents

List of Figures

List of Tables

Acknowledgements

To my advisors, Pieter and Shankar: thank you for the opportunity to have worked with you and learned from you, and for the advice, support, and research insights.

To my research collaborators, John Schulman, David Held, Aviv Tamar, Harri Edwards, Dario Amodei, Alex Ray, Christy Dennison, Ethan Knight, Adam Stooke, Yuhao Wan, and Tyna Eloundou—thank you all for the privilege of working together. John, your suggestion to look for equivalences between seemingly-distinct algorithms led to many wonderful insights about hidden structures; that gift has not stopped giving. Dave, Aviv, I found your guidance throughout the CPO project invaluable. Harri, your taste in problems is excellent and I am grateful for the discussions we had about agent communications protocols that led to VALOR. Dario, you challenged me to favor occam's razor, discard broken ideas, and embrace truth revealed by experiment; you taught me a kind of intellectual humility that helps me see the world more clearly and honestly. Alex, Christy, Ethan, Adam—I'm proud of the work we did together on the Safe Exploration team at OpenAI. Words don't describe what those experiences meant to me. Yuhao, Tyna, it was a joy to work with you in the OpenAI Scholars program, and I am excited to see your future research.

I am grateful to my Berkeley family, Maggie Payne, Joe Menke, Christie Dierk, Kevin Ryan, for all of the love and adventures. To old friends who helped me get where I am today, Michael Bifalco, Dallas Haugh, Thomas Haugh, Pat Komoda, Mike Mulet, Scotty Nicks, Brandi Reece, Brian Schaefer, Jared Simmons, Matt Stolpe. To teachers and mentors who encouraged me to pursue science and research, Chap Percival, Jay Skipper, Anil Rao, Yoonseok Lee. To my first research mentor, Steve Richardson. To colleagues in my undergraduate research experiences who helped me chart a path into science, especially Mark Knight. To friends and colleagues at Berkeley who enriched my grad school experience, Margaret Chapman, Jaime Fisac, Carlos Florensa, Abhishek Gupta, Dylan Hadfield-Menell, Rein Houthooft, David McPherson, Anusha Nagabandi, Kamil Nar, Vicenç Rubies Royo, Dexter Scobee, Rohin Shah, Aravind Srinivas, and many others across Pieter and Shankar's labs. To Laurent El Ghaoui, for being an incredibly effective optimization teacher. To friends, colleagues, and mentors I have worked with at OpenAI who taught me something or inspired me to focus on real problems of consequence, Jack Clark, Mira Murati, Sandhini Agarwal, Marcin Andrychowicz, Amanda Askell, Greg Brockman, Miles Brundage, Paul Christiano, Ariel Herbert-Voss, Geoffrey Irving, Fraser Kelton, Heidy Khlaaf, Gretchen Krueger, Jan Leike, Ryan Lowe, Katie Meyer, Luke Miller, Ashley Pilipiszyn, Matthias Plappert, Alec Radford, Irene Solaiman, Peter Welinder and too many others to name here—I think if I were to name everyone at OpenAI who taught me something important, or whose friendship and collaboration I valued, I would reproduce more than half of the org chart. Thank you all. Finally, I'd like to gratefully acknowledge the support of both UC Berkeley and OpenAI as institutions in hosting and shaping my research agenda. I am humbled and honored to be a part of these vibrant, rigorous research communities.

Thank you Andreana, for your love, support, and patience.

Finally, thank you to my family, and especially to my grandma Ricki, my parents Debbie and Kobbi, and my sister Jenny. This is for you. I love you dearly.

Chapter 1

Introduction

Many of life's foundational challenges involve exploration and trade-offs within exploration. What do you choose when your options have unknown value? When risk and reward trade off against each other, how do you balance them?

When uncertainty and risk seem too great, the easiest option is not to explore at all—to rely on whatever is familiar and comfortable. But stagnation in quality of life is not always desirable, and besides that, sometimes an unforgiving environment removes this choice entirely. For example, as long as all human life is concentrated on a single planet, we are vulnerable to extinction from errant asteroids and other threats. Consequently, we have to figure out how to leave the cradle of Earth and become a spacefaring civilization in order to assure our long-range survival as a species. The exploration problems present themselves immediately: in the vastness of space we have to find those rare few planets that can sustain life, and we have to conduct that exploration across a space environment that is harsh and inhospitable. Yet we cannot avoid trying our luck.

Reinforcement learning (RL) is a discipline of machine learning where agents situated in environments learn to solve tasks from reward signals by trial and error. The trial and error nature of RL means that in order to learn optimal behaviors, agents must fruitfully explore the space of behaviors and outcomes. Consequently, the questions in exploration we have raised—how to explore in the absence of clear direction, and how to explore safely while learning—are among the central questions in the study of RL. These questions form the motivational core of this thesis, and in what follows, we will seek to formalize and address them for RL.

We will focus our efforts on deep reinforcement learning (deep RL), where deep neural networks are used to represent agent behaviors and other functions. Deep reinforcement learning has driven progress in many domains, including robotics [Levine et al., 2016, OpenAI et al., 2018, 2019, Peng et al., 2020], strategy games [Silver et al., 2016, 2017a,b, OpenAI, 2019, DeepMind, 2019], and serving content on social media [Gauci et al., 2018]. It is especially promising because of its flexibility—the framework and methods can be applied to any high-dimensional sequential decision-making problem—and its unique success on problems not currently solvable by any other class of techniques, for example in attaining superhuman

mastery of the game of Go. Yet challenges remain before deep RL-based technologies can be ubiquitously applied to real world problems, including challenges relating to exploration and safety. In this thesis, we will show how to improve the efficiency and safety of exploration in deep RL by modifying the underlying optimization problem that agents try to solve.

We begin the thesis with a review chapter that covers the basics of deep RL and lays the conceptual foundations for our approach. This chapter also introduces "Spinning Up in Deep RL" [Achiam, 2018], an educational resource intended to help new researchers quickly learn about the field and develop practical skills. Following our review, we proceed into two Parts that contain the main body of work.

Part I, **Intrinsically-Motivated Exploration**, centers on improving the efficiency of exploration when rewards are sparse (only rarely provided) or unavailable.

- In chapter 3, "Surprise-Based Intrinsic Motivation," we investigate an approach to exploration where agents are intrinsically rewarded for finding environment states that are surprising relative to a learned model of the environment. This simple, scalable approach helps agents in sparse reward environments learn to solve tasks faster than baseline exploration approaches based on ϵ-greedy action selection or Gaussian noise. This work has previously been presented as Achiam and Sastry [2016].
- In chapter 4, "Variational Option Discovery," we consider the problem of learning in the total absence of task rewards. We introduce an approach to discovering skills in the absence of rewards based on a connection to variational inference. We treat the problem of skill discovery as a variational autoencoding problem, where initially-undefined context vectors sampled from a noise distribution map to trajectories and are then recovered by a decoder. The optimization problem we set up incentivizes the agent to learn an invertible map from contexts to behaviors without any skill-specific rewards. We show how this allows simulated robotics agents to learn motion primitives, though we also describe limitations. This work has previously been presented as Achiam et al. [2017a] and Achiam et al. [2018].

Part II, **Safe Exploration**, addresses the question of minimizing harms resulting from errors during exploration.

- In chapter 5, "Constrained Reinforcement Learning," we describe the safe exploration problem and the formalism of constrained RL, where agents must maximize reward while simultaneously keeping auxiliary costs below thresholds. We take the position that constrained RL is the right framework for making progress on safe exploration, and we set the stage for developing constrained RL methods and benchmarks in subsequent chapters.
- In chapter 6, "Constrained Policy Optimization" (CPO), we introduce the first general-purpose policy search algorithm for constrained deep RL with guarantees for near constraint satisfaction at each iteration. This work is driven by a contribution to RL theory—an improved bound on the relative difference in performance between policies—resulting in a practical algorithm that has better safety characteristics than fixed penalty and Lagrangian baselines in a set of simulated locomotion environments.

We show that a key drawback of the Lagrangian approach is that it can't adapt quickly enough to prevent constraint violations, whereas CPO is designed to adapt rapidly. This work has previously been published as Achiam et al. [2017b].

- In chapter 7, "Benchmarking Safe Exploration," we introduce a new benchmark for constrained reinforcement learning algorithms: Safety Gym. Using the new benchmark environments in Safety Gym, we revisit the comparisons between CPO and Lagrangian baselines and find surprising results that conflict with our earlier experiments: Lagrangian methods achieve the safety goals more reliably than CPO on this more-comprehensive set of tasks. We find that approximation errors in CPO prevent it from satisfying constraints when the tasks are harder. This work has previously been presented as Ray et al. [2019].
- In chapter 8, "PID Lagrangian Methods," we seek an approach that gets the best of both worlds: a method that has the cross-task robustness of Lagrangian methods and the fast adaptation of CPO, without issues from approximation error. To find such an approach, we first reinterpret constrained RL as a control problem where the penalty coefficient for constraint violation is the control. With this insight, we identify the standard Lagrangian approach as integral control, and then develop a proportional-integral-derivative (PID) control for the penalty coefficient. The resulting PID Lagrangian method is a pleasingly-simple approach that meets the design desiderata and is robust to hyperparameter variations. This work was previously published as Stooke et al. [2020].

We conclude in chapter 9, "Epilogues," with a discussion of progress and horizons in exploration and safety.

Chapter 2

Foundations

In this chapter, we will lay the conceptional and notational foundations for the thesis. We begin with brief reviews of core concepts from reinforcement learning and deep learning. Then we stitch the pieces together into deep reinforcement learning, explaining how deep neural networks are used to approximate policies and value functions; we illustrate with a concrete example of how one standard approach (policy optimization) links the concepts into practical algorithms. This puts us in a position to describe the exploration and safety problems in RL in ways that make them tractable to technical approaches. Finally, we frame the core technical approach of this thesis by describing how changes in the RL optimization problem can induce changes in exploration behavior, resulting in faster or safer learning.

An additional section in this chapter is devoted to introducing an educational web resource called "Spinning Up in Deep RL," which is intended to help newcomers quickly learn about deep RL research and practice.

2.1 Reinforcement Learning

So far we have described RL in general terms—agents learning by trial and error from reward signals in environments—and we will now expand on that picture. An RL agent, situated in an environment, experiences a sequential decision-making task. At every step of interaction with its environment, the agent sees a (possibly partial) observation of the state of the environment, and then decides on an action to take. The environment changes when the agent acts on it, but may also change on its own. The agent also perceives a *reward* signal from the environment, a number that tells it how good or bad the current world state is. The goal of the agent is to maximize its *return*, the cumulative reward over many timesteps of interaction. Reinforcement learning methods are used to learn *policies*—rules used by agents for deciding what actions to take—that maximize return. In this section, we'll develop concepts and notation to precisely state the problem of finding an optimal policy.

MDPs and POMDPs: An environment is typically formalized with a Markov Decision Process (MDP), a 5-tuple $(\mathcal{S}, \mathcal{A}, R, P, \mu)$, where:

- $\mathcal{S}$ is a set of states that the environment can take on,
- $\mathcal{A}$ is a set of actions that the agent can take,
- $R(s, a, s')$ is the reward function, mapping state-action-next-state tuples to real-valued rewards,
- $P(s'|s, a)$ is the probability of transitioning to state s' given that the previous state was s and the agent took action a in s,
- and $\mu(s)$ is the probability of starting in state s.

When the environment is partially observed, the appropriate formalism is the Partially Observed Markov Decision Process (POMDP), which includes a set of observations $\mathcal{O}$ and a conditional probability distribution, $P(o|s)$, for what observation is seen in what state. For the purposes of this thesis, we will typically frame things as MDPs rather than POMDPs, though POMDPs are far more common in the real world where perfect observation is impossible.

Trajectories and Policies: A *trajectory*—also called an *episode* or *rollout*—is a sequence of states and actions experienced by an agent acting in an MDP. We will denote by τ a trajectory $(s_0, a_0, s_1, a_1, ...)$. A policy π gives a probability distribution over actions that an agent might take, conditioned in general on a history of observations of the environment. However, we will usually restrict our attention to *stationary* policies that depend only on the most recent state. The probability distribution over trajectories depends on both the environment and the policy—assuming a stationary policy $\pi(a|s)$ and a finite horizon trajectory that ends at time T, we have

$$P(\tau|\pi) = \mu(s_0) \prod_{t=0}^{T-1} \pi(a_t|s_t) P\left(s_{t+1}|s_t, a_t\right). \tag{2.1}$$

Return: There are two typical formulations for the return of a trajectory. The first formulation is for the setting where trajectories are assumed to have infinite length, and there is a discount factor $\gamma \in [0, 1)$ that makes farther-off rewards less valuable than earlier ones. Here, the return of a trajectory $R(\tau)$ is given by:

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}).$$

The discount factor is mathematically useful because it ensures that, as long as rewards are bounded $(\forall s, a, s',\ |R(s, a, s')| \leq R_{\max} < \infty)$, the infinite sum converges. The second formulation is for the setting where trajectories have finite length, and no discount factor is needed:

$$R(\tau) = \sum_{t=0}^{T-1} R(s_t, a_t, s_{t-1}).$$

The RL Optimization Problem: For either formulation of return, we may express the expected return over trajectories by

$$J(\pi) = \mathop{\mathrm{E}}_{\tau\sim\pi}\left[R(\tau)\right],$$

with $\tau \sim \pi$ indicating that the distribution over τ is given by Eq 2.1. The archetypical optimization problem we aim to solve, then, is to search through a space of policies Π for a policy π^* that maximizes $J(\pi)$:

$$\pi^* = \arg\max_{\pi \in \Pi} J(\pi). \tag{2.2}$$

Note: although we write this as an equality for conceptual clarity, the optimal policy is not always unique.

Value Functions: It's often useful to know the value of a state, or state-action pair. By value, we mean the expected return if you start in that state or state-action pair, and then act according to a particular policy forever after. Value functions are used, one way or another, in almost every RL algorithm. We'll describe two value functions that arise frequently in this thesis, given for the infinite horizon discounted return setting.

The *on-policy value function*, $V^\pi(s)$, gives the expected return if you start in state s and always act according to policy π:

$$V^\pi(s) = \mathop{\mathrm{E}}_{\tau \sim \pi} \left[R(\tau) \,|s_0 = s \right].$$

The *on-policy action-value function*, $Q^\pi(s,a)$, gives the expected return if you start in state s, take an arbitrary action a (which may not have come from the policy), and then forever after act according to policy π:

$$Q^\pi(s,a) = \mathop{\mathrm{E}}_{\tau \sim \pi} \left[R(\tau) \,|s_0 = s, a_0 = a \right].$$

These two value functions are connected through the relationships:

$$\begin{aligned} V^\pi(s) &= \mathop{\mathrm{E}}_{a \sim \pi} \left[Q^\pi(s,a) \right], \\ Q^\pi(s,a) &= \mathop{\mathrm{E}}_{s' \sim P} \left[R(s,a,s') + \gamma V^\pi(s') \right], \end{aligned}$$

where $a \sim \pi$ is shorthand for $a \sim \pi(\cdot|s)$, and $s' \sim P$ is shorthand for $s' \sim P(\cdot|s,a)$.

The Advantage Function: The last piece of RL notation we will define here is $A^\pi(s,a)$, the on-policy advantage function. It describes the relative advantage or disadvantage of taking action a by comparison to a baseline of drawing an action from π. It is given by the expression:

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s).$$

We will later find it useful in both theory and practice for deep reinforcement learning.

2.2 Deep Learning

Many key advances in machine learning over the past decade have been driven by *deep learning*, methods centered on the use of deep neural networks as function approximators

[Goodfellow et al., 2016]. Neural networks are versatile function approximators that compose *layers* of parameterized transformations, hence networks with more layers are comparatively deeper. Deep learning techniques have been successful at advancing the state of the art across a diverse array of domains, including image classification [Mahajan et al., 2018], speech recognition [He et al., 2018], natural language processing [Brown et al., 2020], and neural machine translation [Wu et al., 2016], among others.

Neural Networks: What is a neural network, and what makes one deep? The prototypical example of a deep neural network is the multilayer perceptron (MLP), a kind of network that maps from vector-valued inputs to vector-valued outputs through a sequence of parameterized layers with the form:

$$a^{(l)} = g\left(W^{(l)}a^{(l-1)} + b^{(l)}\right).$$

Here,

- $a^{(l-1)}$ is the vector-valued input to layer $l \in \{1, ..., N\}$, with $a^{(0)}$ taken to be the function input x,
- the layer parameters are the *weights* matrix $W^{(l)}$ and the *bias* vector $b^{(l)}$,
- and g is a threshold-like activation function differentiable almost everywhere (eg the sigmoid, tanh, or relu function) applied elementwise.

The layer parameters are learnable: they are adjusted by an algorithm (in a process interchangeably referred to as *learning* or *training*) until the network approximates a target function at an acceptable level of fidelity.

In this thesis, we will usually denote the full set of learnable parameters in a function approximator by a lowercase greek letter, eg θ, and then denote the function approximator itself by a symbol with a subscript referring to its parameters. For example, we would refer to the parameters of this MLP with $\theta = \{W^{(1)}, ..., W^{(N)}, b^{(1)}, ..., b^{(N)}\}$, where N is the number of layers, and write f_θ for the MLP itself.

While the MLP is the prototypical example of a neural network, neural network architectures—the functional forms of layers and their connectivity patterns—vary significantly based on application area. Other common architecture elements include convolutional layers [Bengio and Lecun, 1997], residual connections [He et al., 2016], recurrence [Hochreiter and Schmidhuber, 1997], and attention [Vaswani et al., 2017].

Training Neural Networks: Neural network architectures may vary across application areas, but the prevailing methods for training neural networks are almost universally based on *gradient descent*.

First, a suitable *loss function* L is determined that takes in the network f_θ and any other needed inputs, and returns a scalar value measuring how well or poorly the network represents the target function (a lower loss is better). The parameters θ are randomly initialized and then iteratively updated by stepping in the negative direction of the gradient of L, with the update at iteration k given by:

$$\theta_{k+1} = \theta_k - \alpha\nabla_{\theta_k}L.$$

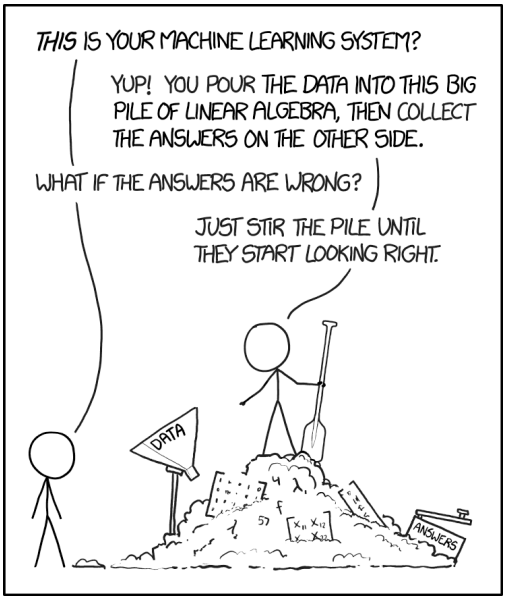


Figure 2.1: How training deep neural networks is done in practice. [Munroe, 2017]

Here, α is a *learning rate*. If α is small, θ_{k+1} is close to θ_k and L is approximately linear around θ_k; this step then produces an improvement in the loss of approximately:

$$L(\theta_{k+1}) \approx L(\theta_k) - \alpha \left\| \nabla_{\theta_k} L \right\|^2.$$

Common variants of gradient descent include momentum gradient descent [Qian, 1999] and Adam [Kingma and Ba, 2015]. If the goal is to maximize, rather than minimize, an objective function, parameters are updated in the direction of the gradient and the procedure is called gradient *ascent*.

Optimizing Expectations: Objective functions in deep learning are usually specified in terms of expectations over per-sample loss functions, where samples come from some data distribution. In practice, the expectations are not directly computable, since the distributions are often over uncountable sets—for example, the distribution over all possible photographs that people might take—so they are replaced by sample-based approximations. Concretely, we would have

$$L(\theta) = \mathop{\mathrm{E}}_{x \sim p} \left[\ell(\theta, x) \right] \approx \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \ell(\theta, x),$$

where x is a data point, p is a data distribution, $\mathcal{D}$ is a dataset sampled from p, and ℓ is a per-sample loss function.

Tuning is Key: Gradient descent and its variants empirically tend to produce monotonic improvement in the loss function while training neural networks when everything is well-"tuned"—that is, when hyperparameters like the learning rate are well-selected, the loss function is appropriately regularized, and the network architecture is favorably-designed. Design details are critical, and despite significant interest and exploratory work (in both theory and practice), there is often considerable uncertainty about why certain choices

perform as well as they do. In Figure 2.1, we provide an illustration from Munroe [2017] demonstrating an accurate account of how tuning is usually performed in the training of deep neural networks. The importance of tuning to performance creates special challenges for scholarship in deep learning [Lipton and Steinhardt, 2018]; often the difference in reported performance between methods is due to subtle and undeclared differences in tuning.

2.3 Deep Reinforcement Learning

Reinforcement learning is an appropriate way to formalize sequential decision-making problems where the optimal behavior is difficult to specify in advance by other means, but where evaluating behaviors is tractable—that is, where it's easier to design a reward function than a policy. Deep learning is the appropriate toolkit for intelligence tasks involving high-dimensional data like natural language or images and video, where simpler ML models that have fewer parameters or that lack compositionality might fail. Therefore deep reinforcement learning, the combination of deep learning with reinforcement learning, is appropriate for problems at the intersection: sequential decision-making tasks where agent inputs and outputs (observations and actions) involve high-dimensional data.

In deep reinforcement learning, deep neural networks are used to represent policies, value functions, and sometimes other functions. The canonical approach to training an agent in deep RL involves an interleaved process of *exploration* and *updating*. During exploration, the agent interacts with the environment to produce new interaction data. Usually, the agent's actions during exploration are based on the most recent version of the policy being learned. In updating, the environment interaction data is used to formulate sample-based objective functions, and the policy and value approximators—the neural networks—are updated by gradient descent. The relationship between exploration and updating is highly symbiotic: the results from each update depend on data acquired through exploration, and the exploration results depend on how updates change agent behavior.

We will make this concrete by using the *vanilla policy gradient* (VPG) algorithm as an example. Pseudocode for VPG is given as Algorithm 1. The central idea in VPG is to solve the RL optimization problem Eq 2.2 by gradient ascent in a space of parameterized stochastic policies $\Pi_\theta = \{\pi_\theta | \theta \in \mathbb{R}^N\}$, where π_θ is a function approximator (in deep RL, a neural network) with N parameters θ. VPG proceeds by alternating phases of exploration and updating: during exploration, trajectories are generated by sampling actions from the current policy, and during updating, the interaction data is used to estimate the gradient of policy performance for gradient ascent. The gradient ascent step increases the likelihood of actions that lead to high return, and decreases the likelihood of actions that lead to low return. The success of VPG depends on the trajectories discovered during the exploration phase. If a batch of trajectories includes high-return trajectories, then those are reinforced by the gradient ascent step and become more likely under the post-update policy. The nature of exploration depends symbiotically on the policy update, since the policy directly drives exploration.

The link between exploration and updating is a key target for the development of improved RL

algorithms. For example, a common failure mode in RL is premature convergence to a nearly-deterministic policy that obtains sub-optimal rewards. One way that this has been addressed is through the use of entropy regularization in the policy update, ensuring that agents maintain some minimum amount of randomness in their exploration behavior. Empirically this has been shown to help in simulated and real robotics environments [Haarnoja et al., 2018a,b]. This involves no direct change to how actions are sampled during exploration—in this approach the exploration behavior is changed solely by adjusting the underlying RL optimization problem to incentivize different behavior.

Algorithm 1 Vanilla Policy Gradient Algorithm

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0
2: **for** $k = 0, 1, 2, ...$ **do**
3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4: Compute rewards-to-go $\hat{R}_t$.
5: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
6: Estimate policy gradient as

$$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k} \hat{A}_t.$$

7: Compute policy update, either using standard gradient ascent,

$$\theta_{k+1} = \theta_k + \alpha_k \hat{g}_k,$$

or via another gradient ascent algorithm like Adam.
8: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.
9: **end for**

2.4 Exploration and Safety

We are now in a position to describe problems in exploration and safety in concrete terms, and to give a brief overview of our technical approach. The link between exploration and optimization we have just discussed is central to the developments in this thesis: we will propose a variety of such adjustments to the RL optimization problem, in order to modify exploration favorably in problems where rewards are sparse or where safety is a concern.

Intrinsically-Motivated Exploration. In RL, the problem of exploring in the absence of

clear direction is the problem of exploring when rewards are sparse (zero at most timesteps) or absent. We can get a sense of how this impacts learning by considering the example of VPG. If the rewards are zero along the trajectories collected during exploration in VPG, the value function should be zero everywhere, the advantage should be zero everywhere, and the policy gradient will correspondingly be zero. That is: when rewards are absent, there is nothing to reinforce and the policy will not change.

Since reward functions are (usually) designed by humans, it's reasonable to ask why we might ever find ourselves in this situation—why would we choose to design a reward function susceptible to this failure mode? In a nutshell, reward design is hard and sometimes sparse rewards are sensible for a given problem, especially when ideal end states are known but there is no easy way to specify preferences over intermediate states and actions. For example, there is no obvious reward function for describing *how* a household robot might set a table for dinner, fold laundry, or clean up. But it is comparatively easy to create a binary evaluation for determining if the house is in a suitable terminal state. This leads to a natural sparse reward task: a task where the rewards are zero everywhere except at "solved" states where the reward is some positive value.

An RL algorithm like VPG can only make progress in these sparse learning tasks when exploration has resulted in the agent discovering at least one reward state, and even then, in practice this is usually insufficient—reward states have to be visited many times in order for the agent to learn something meaningful. But the default approach to stochastic exploration will exhibit random walk behavior, and the likelihood of visiting reward states will be vanishingly small if they are many steps away from the starting point. Some other approach is needed to make learning progress in these tasks.

In this thesis, we will consider approaches to exploration in these kinds of problems that make use of *intrinsic motivation* [Oudeyer and Kaplan, 2008]. Intrinsic motivation incentivizes agents away from random walk behavior and towards purposeful, directed exploration, without any task-specific information. This is accomplished through modifying the policy objective function to include additional terms: in chapter 3, we will look at adding an intrinsic reward function based on surprise, and in chapter 4, we will look at an objective function meant to distinguish between different behaviors.

Safe Exploration. The safe exploration problem arises naturally from the trial and error nature of RL: sometimes agents will try actions that lead to unacceptable errors. To make it tractable, we will formalize it through *constrained* Markov Decision Processes (CMDPs), where environments have auxiliary cost functions in addition to the reward function, and agents try to maximize the reward subject to constraints on the costs. We will elaborate on this framework and the basis for preferring it in chapter 5.

In chapter 6, we will develop an algorithm called Constrained Policy Optimization (CPO) that approximately enforces constraints at each policy update in order to achieve safe exploration. In chapter 7, we develop benchmark environments that help us find the limits of CPO and enable further research on algorithms. In chapter 8, we start from the max-min problem equivalent to the constained problem, and develop a new technique for updating that improves the robustness of constraint satisfaction throughout training.

2.5 Spinning Up in Deep Reinforcement Learning

The introduction to deep reinforcement learning we have given in this chapter is cursory, omitting rich material on sub-fields, algorithmic developments, and the kind of hands-on knowledge that a practitioner needs in order to be productive. For this, we refer the interested reader to an educational resource created during the course of work in this thesis: "Spinning Up in Deep Reinforcement Learning" [Achiam, 2018]. "Spinning Up" consists of several core components:

- A short introduction to RL terminology, kinds of algorithms, and basic theory.
- An essay about how to grow into an RL research role.
- A curated list of important papers organized by topic.
- A well-documented code repository of short, standalone implementations of: Vanilla Policy Gradient (VPG) [Duan et al., 2016], Trust Region Policy Optimization (TRPO) [Schulman et al., 2015], Proximal Policy Optimization (PPO) [Schulman et al., 2017], Deep Deterministic Policy Gradient (DDPG) [Lillicrap et al., 2016], Twin Delayed DDPG (TD3) [Fujimoto et al., 2018], and Soft Actor-Critic (SAC) [Haarnoja et al., 2018a]. Algorithms are implemented in Tensorflow (v1) [Abadi et al., 2016] and in PyTorch [Paszke et al., 2019].
- Code implementation exercises to serve as warm-ups.

"Spinning Up" is located at `https://spinningup.openai.com/en/latest/`, and the source code is available at `https://github.com/openai/spinningup`. At the time of publication of this thesis, "Spinning Up" has more than 5500 stars on Github and more than 1200 forks.

We include a reproduction of the essay on growing as an RL researcher in Appendix A.1.

Part I

Intrinsically-Motivated Exploration

Chapter 3

Surprise-Based Intrinsic Motivation

Exploration in complex domains is a key challenge in reinforcement learning, especially for tasks with very sparse rewards. Recent successes in deep reinforcement learning have been achieved mostly using simple heuristic exploration strategies such as ϵ-greedy action selection or Gaussian control noise, but there are many tasks where these methods are insufficient to make any learning progress. Here, we consider more complex heuristics: efficient and scalable exploration strategies that maximize a notion of an agent's surprise about its experiences via intrinsic motivation. We propose to learn a model of the MDP transition probabilities concurrently with the policy, and to form intrinsic rewards that approximate the KL-divergence of the true transition probabilities from the learned model. One of our approximations results in using surprisal as intrinsic motivation, while the other gives the k-step learning progress. We show that our incentives enable agents to succeed in a wide range of environments with high-dimensional state spaces and very sparse rewards, including continuous control tasks and games in the Atari RAM domain, outperforming several other heuristic exploration techniques.

3.1 Introduction

A model-free reinforcement learning agent uses experiences obtained from interacting with an initially-unknown environment to learn a behavior that maximizes a reward signal. The optimality of the learned behavior is strongly dependent on how the agent approaches the exploration/exploitation trade-off. If it explores poorly or too little, it may never find rewards it can learn from and its behavior will never improve. If it does find rewards but exploits them too intensely, it may prematurely converge to suboptimal behaviors. The problem of optimal exploration may be framed as seeking exploration strategies that result in the fastest possible convergence to optimal behavior, avoiding both of these pitfalls. For environments with finite state and action spaces, this problem is largely addressed by algorithms that have theoretical performance guarantees, like R-Max [Brafman and Tennenholtz, 2002]. However, for continuous or high-dimensional environments—the regime where deep reinforcement

learning methods are needed—these theoretically-justified algorithms admit no obvious generalization or are prohibitively impractical to implement.

The default approach to exploration in deep reinforcement learning is to inject noise during action selection, increasing the diversity of trajectories seen by the agent. For algorithms based on Q-learning in discrete-action environments, like Deep Q-Networks (DQN) [Mnih et al., 2015], this is done by "ϵ-greedy" action selection: the action that maximizes the current Q^*-approximator is taken with probability $1 - \epsilon$, otherwise a random action is taken. For policy optimization algorithms, actions are sampled from the current stochastic policy; in continuous control environments, this usually means sampling Gaussian noise to add to a deterministic base policy. These noise-based exploration strategies tend to be sufficient when the environment provides dense rewards for the task—that is, when rewards are nonzero at most time steps—though they are inadequate when rewards are sparse. For example, DQN with ϵ-greedy exploration achieves superhuman performance on many Atari games with dense rewards. However, on games with sparse rewards, like Montezuma's Revenge, DQN and its variants fail to achieve scores even at the level of a novice human due to inadequate exploration [van Hasselt et al., 2016, Wang et al., 2016, Mnih et al., 2016, Nair et al., 2015]. Similarly, Duan et al. [2016] found that exploration through Gaussian control noise enabled policy optimization algorithms to succeed on simulated robotics tasks with dense rewards (like rewards proportional to the forward velocity of the robot). Yet, in tasks with sparse rewards—where the agent would only see nonzero rewards after first figuring out complex motion primitives—exploration via Gaussian control noise was inadequate, and none of the tested policy optimization algorithms could attain scores better than random agents.

One approach to encourage better exploration in sparse reward settings is via intrinsic motivation, where an agent has a task-independent intrinsic reward function which it seeks to maximize in addition to the reward from the environment. Examples of intrinsic motivation include empowerment, which measures the level of control the agent has over its future; surprise, where the agent is excited to see outcomes that run contrary to its understanding of the world; and novelty, where the agent is excited to see new states (which is tightly connected to surprise, as shown by Bellemare et al. [2016]). For in-depth reviews of the different types of intrinsic motivation, we direct the reader to Barto et al. [2013] and Oudeyer and Kaplan [2008].

Recently, several applications of intrinsic motivation to the deep reinforcement learning setting (such as Bellemare et al. [2016], Houthooft et al. [2016], Stadie et al. [2015]) have found promising success. In this work, we build on that success by exploring scalable measures of surprise for intrinsic motivation in deep reinforcement learning. We formulate surprise as the KL-divergence of the true transition probability distribution from a transition model which is learned concurrently with the policy, and consider two approximations to this divergence which are easy to compute in practice. One of these approximations results in using the surprisal of a transition as an intrinsic reward; the other results in using a measure of learning progress which is closer to a Bayesian concept of surprise. Our contributions are as follows:

1. we investigate surprisal and learning progress as intrinsic rewards across a wide range of environments in the deep reinforcement learning setting, and demonstrate empirically

that the incentives (especially surprisal) result in efficient exploration,

2. we evaluate the difficulty of the slate of sparse reward continuous control tasks introduced by Houthooft et al. [2016] to benchmark exploration incentives, and introduce a new task to complement the slate,

3. and we present an efficient method for learning the dynamics model (transition probabilities) concurrently with a policy.

We distinguish our work from prior work in a number of implementation details: unlike Bellemare et al. [2016], we learn a transition model as opposed to a state-action occupancy density; unlike Stadie et al. [2015], our formulation naturally encompasses environments with stochastic dynamics; unlike Houthooft et al. [2016], we avoid the overhead of maintaining a distribution over possible dynamics models, and learn a single deep dynamics model.

In our empirical evaluations, we compare the performance of our proposed intrinsic rewards with other heuristic intrinsic reward schemes and to recent results from the literature. In particular, we compare to Variational Information Maximizing Exploration (VIME) [Houthooft et al., 2016], a method which approximately maximizes Bayesian surprise and outperforms standard baselines on continuous control with sparse rewards. We show that our incentives can perform on the level of VIME at a lower computational cost.

3.2 Surprise Incentives

To train an agent with surprise-based exploration, we alternate between making an update step to a dynamics model (an approximator of the MDP's transition probability function), and making a policy update step that maximizes a trade-off between policy performance and a surprise measure.

The dynamics model step makes progress on the optimization problem

$$\min_{\phi} -\frac{1}{|D|} \sum_{(s,a,s') \in D} \log P_\phi(s'|s,a) + \alpha f(\phi), \tag{3.1}$$

where D is is a dataset of transition tuples from the environment, P_ϕ is the model we are learning, f is a regularization function, and $\alpha > 0$ is a regularization trade-off coefficient. The policy update step makes progress on an approximation to the optimization problem

$$\max_{\pi} J(\pi) + \eta \underset{s,a \sim \pi}{E} \left[D_{KL}(P||P_\phi)[s,a] \right], \tag{3.2}$$

where $\eta > 0$ is an explore-exploit trade-off coefficient, and $J(\pi)$ is the policy performance objective (in this chapter, the finite horizon undiscounted return). The exploration incentive in (3.2), which we select to be the on-policy average KL-divergence of P_ϕ from P, is intended to capture the agent's surprise about its experience. The dynamics model P_ϕ should only be close to P on regions of the transition state space that the agent has already visited (because those transitions will appear in D and thus the model will be fit to them), and as a result,

the KL divergence of P_ϕ and P will be higher in unfamiliar places. Essentially, this exploits the generalization in the model to encourage the agent to go where it has not gone before. The surprise incentive in (3.2) gives the net effect of performing a reward shaping of the form

$$r'(s,a,s') = r(s,a,s') + \eta \left(\log P(s'|s,a) - \log P_\phi(s'|s,a)\right), \tag{3.3}$$

where $r(s,a,s')$ is the original reward and $r'(s,a,s')$ is the transformed reward, so ideally we could solve (3.2) by applying any reinforcement learning algorithm with these reshaped rewards. In practice, we cannot directly implement this reward reshaping because P is unknown. Instead, we consider two ways of finding an approximate solution to (3.2).

In one method, we approximate the KL-divergence by the cross-entropy, which is reasonable when $H(P)$ is finite (and small) and P_ϕ is sufficiently far from P[1]; that is, denoting the cross-entropy by $H(P,P_\phi)[s,a] \doteq E_{s'\sim P(\cdot|s,a)}[-\log P_\phi(s'|s,a)]$, we assume

$$\begin{aligned} D_{KL}(P||P_\phi)[s,a] &= H(P,P_\phi)[s,a] - H(P)[s,a] \\ &\approx H(P,P_\phi)[s,a]. \end{aligned} \tag{3.4}$$

This approximation results in a reward shaping of the form

$$r'(s,a,s') = r(s,a,s') - \eta \log P_\phi(s'|s,a); \tag{3.5}$$

here, the intrinsic reward is the surprisal of s' given the model P_ϕ and the context (s,a).

In the other method, we maximize a lower bound on the objective in (3.2) by lower bounding the surprise term:

$$\begin{aligned} D_{KL}(P||P_\phi)[s,a] &= D_{KL}(P||P_{\phi'})[s,a] + \mathop{E}_{s'\sim P}\left[\log \frac{P_{\phi'}(s'|s,a)}{P_\phi(s'|s,a)}\right] \\ &\geq \mathop{E}_{s'\sim P}\left[\log \frac{P_{\phi'}(s'|s,a)}{P_\phi(s'|s,a)}\right]. \end{aligned} \tag{3.6}$$

The bound (3.6) results in a reward shaping of the form

$$r'(s,a,s') = r(s,a,s') + \eta \left(\log P_{\phi'}(s'|s,a) - \log P_\phi(s'|s,a)\right), \tag{3.7}$$

which requires a choice of ϕ'. From (3.6), we can see that the bound becomes tighter by minimizing $D_{KL}(P||P_{\phi'})$. As a result, we choose ϕ' to be the parameters of the dynamics model after k updates based on (3.1), and ϕ to be the parameters from before the updates. Thus, at iteration t, the reshaped rewards are

$$r'(s,a,s') = r(s,a,s') + \eta \left(\log P_{\phi_t}(s'|s,a) - \log P_{\phi_{t-k}}(s'|s,a)\right); \tag{3.8}$$

here, the intrinsic reward is the k-step learning progress at (s,a,s'). It also bears a resemblance to Bayesian surprise; we expand on this similarity in the next section.

In our experiments, we investigate both the surprisal bonus (3.5) and the k-step learning progress bonus (3.8) (with varying values of k).

[1]On the other hand, if $H(P)[s,a]$ is non-finite everywhere—for instance if the MDP has continuous states and deterministic transitions—then as long as it has the same sign everywhere, $\mathrm{E}_{s,a\sim\pi}[H(P)[s,a]]$ is a constant with respect to π and we can drop it from the optimization problem anyway.

3.2.1 Discussion

Ideally, we would like the intrinsic rewards to vanish in the limit as $P_\phi \to P$, because in this case, the agent should have sufficiently explored the state space, and should primarily learn from extrinsic rewards. For the proposed intrinsic reward in (3.5), this is not the case, and it may result in poor performance in that limit. The thinking goes that when $P_\phi = P$, the agent will be incentivized to seek out states with the noisiest transitions. However, we argue that this may not be an issue, because the intrinsic motivation seems mostly useful long before the dynamics model is fully learned. As long as the agent is able to find the extrinsic rewards before the intrinsic reward is just the entropy in P, the pathological noise-seeking behavior should not happen. On the other hand, the intrinsic reward in (3.8) should not suffer from this pathology, because in the limit, as the dynamics model converges, we should have $P_{\phi_t} \approx P_{\phi_{t-k}}$. Then the intrinsic reward will vanish as desired.

Next, we relate (3.8) to Bayesian surprise. The Bayesian surprise associated with a transition is the reduction in uncertainty over possibly dynamics models from observing it [Barto et al., 2013, Itti and Baldi, 2009]:

$$D_{KL}\left(P(\phi|h_t, a_t, s_{t+1})||P(\phi|h_t)\right).$$

Here, $P(\phi|h_t)$ is meant to represent a distribution over possible dynamics models parametrized by ϕ given the preceding history of observed states and actions h_t (so h_t includes s_t), and $P(\phi|h_t, a_t, s_{t+1})$ is the posterior distribution over dynamics models after observing (a_t, s_{t+1}). By Bayes' rule, the dynamics prior and posterior are related to the model-based transition probabilities by

$$P(\phi|h_t, a_t, s_{t+1}) = \frac{P(\phi|h_t)P(s_{t+1}|h_t, a_t, \phi)}{\mathrm{E}_{\phi\sim P(\cdot|h_t)}\left[P(s_{t+1}|h_t, a_t, \phi)\right]},$$

so the Bayesian surprise can be expressed as

$$\mathop{\mathrm{E}}_{\phi\sim P_{t+1}}\left[\log P(s_{t+1}|h_t, a_t, \phi)\right] - \log \mathop{\mathrm{E}}_{\phi\sim P_t}\left[P(s_{t+1}|h_t, a_t, \phi)\right], \tag{3.9}$$

where $P_{t+1} = P(\cdot|h_t, a_t, s_{t+1})$ is the posterior and $P_t = P(\cdot|h_t)$ is the prior. In this form, the resemblance between (3.9) and (3.8) is clarified. Although the update from ϕ_{t-k} to ϕ_t is not Bayesian—and is performed in batch, instead of per transition sample—we speculate (3.8) might contain similar information to (3.9).

3.2.2 Implementation Details

Our implementation uses L_2 regularization in the dynamics model fitting, and we impose an additional constraint to keep model iterates close in the KL-divergence sense. Denoting the average divergence as

$$\bar{D}_{KL}(P_{\phi'}||P_\phi) = \frac{1}{|D|}\sum_{(s,a)\in D} D_{KL}(P_{\phi'}||P_\phi)[s, a], \tag{3.10}$$

our dynamics model update is

$$\phi_{i+1} = \arg\min_{\phi} -\frac{1}{|D|} \sum_{(s,a,s') \in D} \log P_\phi(s'|s,a) + \alpha \|\phi\|_2^2 \ : \ \bar{D}_{KL}(P_\phi || P_{\phi_i}) \leq \kappa. \tag{3.11}$$

The constraint value κ is a hyper-parameter of the algorithm. We solve this optimization problem approximately using a single second-order step with a line search, as described by Schulman et al. [2015]; full details are given in supplementary material. D is a FIFO replay memory, and at each iteration, instead of using the entirety of D for the update step we sub-sample a batch $d \subset D$. Also, similarly to Houthooft et al. [2016], we adjust the bonus coefficient η at each iteration, to keep the average bonus magnitude upper-bounded (and usually fixed). Let η_0 denote the desired average bonus, and $r_+(s,a,s')$ denote the intrinsic reward; then, at each iteration, we set

$$\eta = \frac{\eta_0}{\max\left(1, \frac{1}{|B|}\left|\sum_{(s,a,s') \in B} r_+(s,a,s')\right|\right)},$$

where B is the batch of data used for the policy update step. This normalization improves the stability of the algorithm by keeping the scale of the bonuses fixed with respect to the scale of the extrinsic rewards. Also, in environments where the agent can die, we avoid the possibility of the intrinsic rewards becoming a living cost by translating all bonuses so that the mean is nonnegative. The basic outline of the algorithm is given as Algorithm 2. In all experiments, we use fully-factored Gaussian distributions for the dynamics models, where the means and variances are the outputs of neural networks.

Algorithm 2 Reinforcement Learning with Surprise Incentive

Input: Initial policy π_0, dynamics model P_{ϕ_0}
repeat
  collect rollouts on current policy π_i
  add rollout (s,a,s') tuples to replay memory D
  compute reshaped rewards using (3.5) or (3.8) with dynamics model P_{ϕ_i}
  normalize η by the average intrinsic reward of the current batch of data
  update policy to π_{i+1} using any RL algorithm with the reshaped rewards
  update the dynamics model to $P_{\phi_{i+1}}$ according to (3.11)
until training is completed

3.3 Experiments

We evaluate our proposed surprise incentives on a wide range of benchmarks that are challenging for naive exploration methods, including continuous control and discrete control tasks. Our continuous control tasks include the slate of sparse reward tasks introduced by Houthooft et al. [2016]: sparse MountainCar, sparse CartPoleSwingup, and sparse HalfCheetah, as well as a new sparse reward task that we introduce here: sparse Swimmer. (We refer to these

environments with the prefix 'sparse' to differentiate them from other versions which appear in the literature, where agents receive non-sparse reward signals.) Additionally, we evaluate performance on a highly-challenging hierarchical sparse reward task introduced by Duan et al. [2016], SwimmerGather. The discrete action tasks are several games from the Atari RAM domain of the OpenAI Gym [Brockman et al., 2016]: Pong, BankHeist, Freeway, and Venture.

Environments with deterministic and stochastic dynamics are represented in our benchmarks: the continuous control domains have deterministic dynamics, while the Gym Atari RAM games have stochastic dynamics. (In the Atari games, actions are repeated for a random number of frames.)

We use Trust Region Policy Optimization (TRPO) [Schulman et al., 2015], a natural policy gradient method, as our base reinforcement learning algorithm throughout our experiments, and we use the rllab implementations of TRPO and the continuous control tasks [Duan et al., 2016]. Full details for the experimental set-up are included in sections after the conclusion.

On all tasks, we compare against TRPO without intrinsic rewards, which we refer to as using naive exploration (in contrast to intrinsically motivated exploration). For the continuous control tasks, we also compare against intrinsic motivation using the L_2 model prediction error,

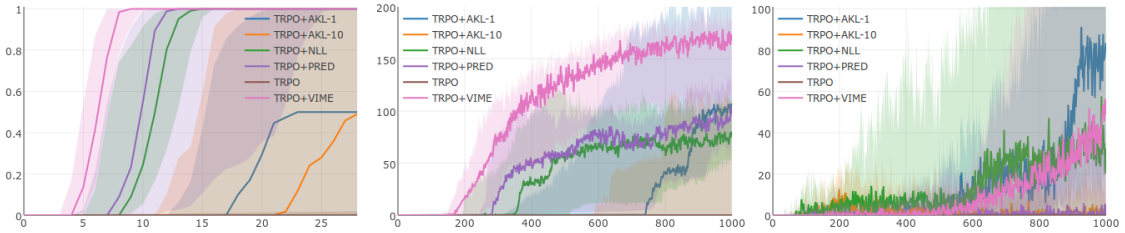
$$r_+(s, a, s') = \|s' - \mu_\phi(s, a)\|_2, \tag{3.12}$$

where μ_ϕ is the mean of the learned Gaussian distribution P_ϕ. The model prediction error was investigated as intrinsic motivation for deep reinforcement learning by Stadie et al. [2015], although they used a different method for learning the model μ_ϕ. This comparison helps us verify whether or not our proposed form of surprise, as a KL-divergence from the true dynamics model, is useful. Additionally, we compare our performance against the performance reported by Houthooft et al. [2016] for Variational Information Maximizing Exploration (VIME), a strong baseline method where the intrinsic reward associated with a transition approximates its Bayesian surprise using variational methods.

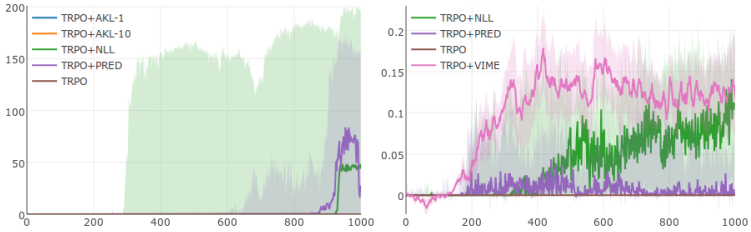
As a final check for the continuous control tasks, we benchmark the tasks themselves, by measuring the performance of the surprisal bonus without any dynamics learning: $r_+(s, a, s') = -\log P_{\phi_0}(s'|s, a)$, where ϕ_0 are the original random parameters of P_ϕ. This allows us to verify whether our benchmark tasks actually require surprise to solve at all, or if random exploration strategies successfully solve them.

3.3.1 Continuous Control Results

Median performance curves are shown in Figure 3.1 with interquartile ranges shown in shaded areas. Note that TRPO without intrinsic motivation failed on all tasks: the median score and upper quartile range for naive exploration were zero everywhere. Also note that TRPO with random exploration bonuses failed on most tasks, as shown separately in Figure 3.2.



(a) MountainCar (b) CartpoleSwingup (c) HalfCheetah



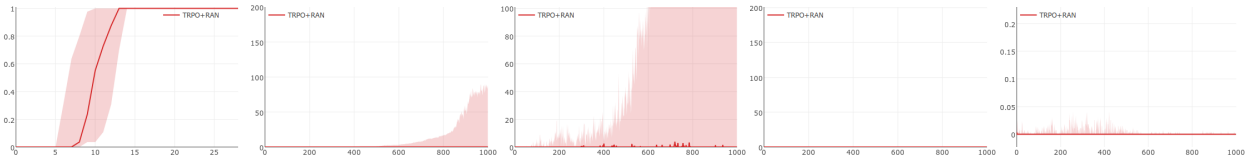
(d) Swimmer (e) SwimmerGather

Figure 3.1: Median performance for the continuous control tasks over 10 runs with a fixed set of seeds, with interquartile ranges shown in shaded areas. The x-axis is iterations of training; the y-axis is average undiscounted return. AKL-k refers to learning progress (3.8), NLL to surprisal (3.5), and PRED to (3.12). For the first four tasks, $\eta_0 = 0.001$; for SwimmerGather, $\eta_0 = 0.0001$. Results for VIME are from Houthooft et al. [2016], reproduced here with permission. We note that the performance curve for VIME in the SwimmerGather environment represents only 2 random seeds, not 10.

We found that surprise was not needed to solve MountainCar, but was necessary to perform well on the other tasks.

The surprisal bonus was especially robust across tasks, achieving good results in all domains and substantially exceeding the other baselines on the more challenging ones. The learning progress bonus for $k = 1$ was successful on CartpoleSwingup and HalfCheetah but it faltered in the others. Its weak performance in MountainCar was due to premature convergence of the dynamics model, which resulted in the agent receiving intrinsic rewards that were identically zero. (Given the simplicity of the environment, it is not surprising that the dynamics model converged so quickly.) In Swimmer, however, it seems that the learning progress bonuses did not inspire sufficient exploration. Because the Swimmer environment is effectively a stepping stone to the harder SwimmerGather, where the agent has to learn a motion primitive *and* collect target pellets, on SwimmerGather, we only evaluated the intrinsic rewards that had been successful on Swimmer.

Both surprisal and learning progress (with $k = 1$) exceeded the reported performance of VIME on HalfCheetah by learning to solve the task more quickly. On CartpoleSwingup, however, both were more susceptible to getting stuck in locally optimal policies, resulting in lower median scores than VIME. Surprisal performed comparably to VIME on SwimmerGather, the hardest task in the slate—in the sense that after 1000 iterations, they both reached approximately the same median score—although with greater variance than VIME.



(a) MountainCar (b) CartpoleSwing (c) HalfCheetah (d) Swimmer (e) SwimmerGather

Figure 3.2: Benchmarking the benchmarks: median performance for the continuous control tasks over 10 runs with a fixed set of seeds, with interquartile ranges shown in shaded areas, using the surprisal without learning bonus. RAN refers to the fact that this is essentially a random exploration bonus.

Our results suggest that surprisal is a viable alternative to VIME in terms of performance, and is highly favorable in terms of computational cost. In VIME, a backwards pass through the dynamics model must be computed for every transition tuple separately to compute the intrinsic rewards, whereas our surprisal bonus only requires forward passes through the dynamics model for intrinsic reward computation. (Limitations of current deep learning tool kits make it difficult to efficiently compute separate backwards passes, whereas almost all of them support highly parallel forward computations.) Furthermore, our dynamics model is substantially simpler than the Bayesian neural network dynamics model of VIME. To illustrate this point, in Figure 3.3 we show the results of a speed comparison making use of the open-source VIME code [Houthooft, 2016], with the settings described in the VIME paper. In our speed test, our bonus had a per-iteration speedup of a factor of 3 over VIME.[2] We give a full analysis of the potential speedup in Section 3.8.

3.3.2 Atari RAM Domain Results

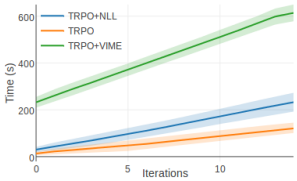
Median performance curves are shown in Figure 3.4, with tasks arranged from (a) to (d) roughly in order of increasing difficulty.

In Pong, naive exploration naturally succeeds, so we are not surprised to see that intrinsic motivation does not improve performance. However, this serves as a sanity check to verify that our intrinsic rewards do not degrade performance. (As an aside, we note that the performance here falls short of the standard score of 20 for this domain because we truncate play at 5000 timesteps.)

In BankHeist, we find that intrinsic motivation accelerates the learning significantly. The agents with surprisal incentives reached high levels of performance (scores > 1000) 10%

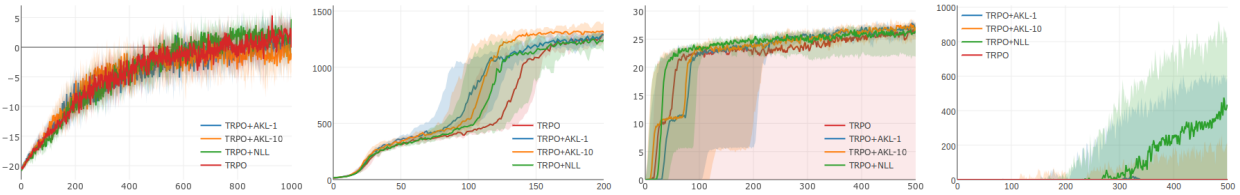
[2]We compute this by comparing the marginal time cost incurred just by the bonus in each case: that is, if T_{vime}, $T_{surprisal}$, and $T_{nobonus}$ denote the times to 15 iterations, we obtain the speedup as

$$\frac{T_{vime} - T_{nobonus}}{T_{surprisal} - T_{nobonus}}.$$



	VIME	Surprisal	No Bonus
Avg. Initialization Time	3 min, 52 s	0 min, 30 s	0 min, 13 s
Avg. Time to 15 Iterations	6 min, 21 s	3 min, 23 s	1 min, 51 s

Figure 3.3: Speed test: comparing the performance of VIME against our proposed intrinsic reward schemes, average compute time over 5 random runs. Tests were run on a Thinkpad T440p with four physical Intel i7-4700MQ cores, in the sparse HalfCheetah environment. VIME's greater initialization time, which is primarily spent in computation graph compilation, reflects the complexity of the Bayesian neural network model.



(a) Pong-RAM (b) BankHeist-RAM (c) Freeway-RAM (d) Venture-RAM

Figure 3.4: Median performance for the Atari RAM tasks over 10 runs with a fixed set of seeds, with interquartile ranges shown in shaded areas. The x-axis is iterations of training; the y-axis is average undiscounted return. AKL-k refers to learning progress (3.8), and NLL to surprisal (3.5).

sooner than naive exploration, while agents with learning progress incentives reached high levels almost 20% sooner.

In Freeway, the median performance for TRPO without intrinsic motivation was adequate, but the lower quartile range was quite poor—only 6 out of 10 runs ever found rewards. With the learning progress incentives, 8 out of 10 runs found rewards; with the surprisal incentive, all 10 did. Freeway is a game with very sparse rewards, where the agent effectively has to cross a long hallway before it can score a point, so naive exploration tends to exhibit random walk behavior and only rarely reaches the reward state. The intrinsic motivation helps the agent explore more purposefully.

In Venture, we obtain our strongest results in the Atari domain. Venture is extremely difficult because the agent has to navigate a large map to find very sparse rewards, and the agent can be killed by enemies interspersed throughout. We found that our intrinsic rewards were able to substantially improve performance over naive exploration in this challenging environment. Here, the best performance was again obtained by the surprisal incentive, which usually inspired the agent to reach scores greater than 500.

3.3.3 Comparing Incentives

Among our proposed incentives, we found that surprisal worked the best overall, achieving the most consistent performance across tasks. The learning progress-based incentives worked well on some domains, but generally not as well as surprisal. Interestingly, learning progress with $k = 10$ performed much worse on the continuous control tasks than with $k = 1$, but we observed virtually no difference in their performance on the Atari games; it is unclear why this should be the case.

Surprisal strongly outperformed the L_2 error based incentive on the harder continuous control tasks, learning to solve them more quickly and without forgetting. Because we used fully-factored Gaussians for all of our dyanmics models, the surprisal had the form

$$-\log P_\phi(s'|s,a) = \sum_{i=1}^{n} \left(\frac{(s'_i - \mu_{\phi,i}(s,a))^2}{2\sigma^2_{\phi,i}(s,a)} + \log \sigma_{\phi,i}(s,a) \right) + \frac{k}{2} \log 2\pi,$$

which essentially includes the L_2-squared error norm as a sub-expression. The relative difference in performance suggests that the variance terms confer additional useful information about the novelty of a state-action pair.

3.4 Related Work

Substantial theoretical work has been done on optimal exploration in finite MDPs, resulting in algorithms such as E^3 [Kearns and Singh, 1998], R-max [Brafman and Tennenholtz, 2002], and UCRL [Jaksch et al., 2010], which scale polynomially with MDP size. However, these works do not permit obvious generalizations to MDPs with continuous state and action

spaces. C-PACE [Pazis and Parr, 2013] provides a theoretical foundation for PAC-optimal exploration in MDPs with continuous state spaces, but it requires a metric on state spaces. Lopes et al. [2012] investigated exploration driven by learning progress and proved theoretical guarantees for their approach in the finite MDP case, but they did not address the question of scaling their approach to continuous or high-dimensional MDPs. Also, although they formulated learning progress in the same way as (3.8), they formed intrinsic rewards differently. Conceptually and mathematically, our work is closest to prior work on curiosity and surprise [Itti and Baldi, 2009, Schmidhuber, 1991, Storck et al., 1995, Sun et al., 2011], although these works focus mainly on small finite MDPs.

Recently, several intrinsic motivation strategies that deal specifically with deep reinforcement learning have been proposed. Stadie et al. [2015] learn deterministic dynamics models by minimizing Euclidean loss—whereas in our work, we learn stochastic dynamics with cross entropy loss—and use L_2 prediction errors for intrinsic motivation. Houthooft et al. [2016] train Bayesian neural networks to approximate posterior distributions over dynamics models given observed data, by maximizing a variational lower bound; they then use second-order approximations of the Bayesian surprise as intrinsic motivation. Bellemare et al. [2016] derived pseudo-counts from CTS density models over states and used those to form intrinsic rewards, notably resulting in dramatic performance improvement on Montezuma's Revenge, one of the hardest games in the Atari domain. Mohamed and Rezende [2015] developed a scalable method of approximating empowerment, the mutual information between an agent's actions and the future state of the environment, using variational methods. Oh et al. [2015] estimated state visit frequency using Gaussian kernels to compare against a replay memory, and used these estimates for directed exploration.

3.5 Conclusions

In this work, we formulated surprise for intrinsic motivation as the KL-divergence of the true transition probabilities from learned model probabilities, and derived two approximations—surprisal and k-step learning progress—that are scalable, computationally inexpensive, and suitable for application to high-dimensional and continuous control tasks. We showed that empirically, motivation by surprisal and 1-step learning progress resulted in efficient exploration on several hard deep reinforcement learning benchmarks. In particular, we found that surprisal was a robust and effective intrinsic motivator, outperforming other heuristics on a wide range of tasks, and competitive with the current state-of-the-art for intrinsic motivation in continuous control.

3.6 Single Step Second-Order Optimization

In our experiments, we approximately solve several optimization problems by using a single second-order step with a line search. This section will describe the exact methodology, which was originally given by Schulman et al. [2015].

We consider the optimization problem

$$p^* = \max_\theta L(\theta) \ : \ D(\theta) \leq \delta, \tag{3.13}$$

where $\theta \in R^n$, and for some θ_{old} we have $D(\theta_{old}) = 0$, $\nabla_\theta D(\theta_{old}) = 0$, and $\nabla^2_\theta D(\theta_{old}) \succeq 0$; also, $\forall \theta, D(\theta) \geq 0$.

We suppose that δ is small, so the optimal point will be close to θ_{old}. We also suppose that the curvature of the constraint is much greater than the curvature of the objective. As a result, we feel justified in approximating the objective to linear order and the constraint to quadratic order:

$$\begin{aligned} L(\theta) &\approx L(\theta_{old}) + g^T(\theta - \theta_{old}) & g &\doteq \nabla_\theta L(\theta_{old}) \\ D(\theta) &\approx \frac{1}{2}(\theta - \theta_{old})^T A(\theta - \theta_{old}) & A &\doteq \nabla^2_\theta D(\theta_{old}). \end{aligned}$$

We now consider the approximate optimization problem,

$$p^* \approx \max_\theta g^T(\theta - \theta_{old}) \ : \ \frac{1}{2}(\theta - \theta_{old})^T A(\theta - \theta_{old}) \leq \delta.$$

This optimization problem is convex as long as $A \succeq 0$, which is an assumption that we make. (If this assumption seems to be empirically invalid, then we repair the issue by using the substitution $A \to A + \epsilon I$, where I is the identity matrix, and $\epsilon > 0$ is a small constant chosen so that we usually have $A + \epsilon I \succeq 0$.) This problem can be solved analytically by applying methods of duality, and its optimal point is

$$\theta^* = \theta_{old} + \sqrt{\frac{2\delta}{g^T A^{-1} g}} A^{-1} g. \tag{3.14}$$

It is possible that the parameter update step given by (3.14) may not exactly solve the original optimization problem (3.13)—in fact, it may not even satisfy the constraint—so we perform a line search between θ_{old} and θ^*. Our update with the line search included is given by

$$\theta = \theta_{old} + s^k \sqrt{\frac{2\delta}{g^T A^{-1} g}} A^{-1} g, \tag{3.15}$$

where $s \in (0, 1)$ is a backtracking coefficient, and k is the smallest integer for which $L(\theta) \geq L(\theta_{old})$ and $D(\theta) \leq \delta$. We select k by checking each of $k = 1, 2, ..., K$, where K is the maximum number of backtracks. If there is no value of k in that range which satisfies the conditions, no update is performed.

Because the optimization problems we solve with this method tend to involve thousands of parameters, inverting A is prohibitively computationally expensive. Thus in the implementation of this algorithm that we use, the search direction $x = A^{-1}g$ is found by using the conjugate gradient method to solve $Ax = g$; this avoids the need to invert A.

When A and g are sample averages meant to stand in for expectations, we employ an additional trick to reduce the total number of computations necessary to solve $Ax = g$. The computation of A is more expensive than g, and so we use a smaller fraction of the population to estimate it quickly. Concretely, suppose that the original optimization problem's objective is $E_{z\sim P}[L(\theta, z)]$, and the constraint is $E_{z\sim P}[D(\theta, z)] \leq \delta$, where z is some random variable and P is its distribution; furthermore, suppose that we have a dataset of samples $D = \{z_i\}_{i=1,...,N}$ drawn on P, and we form an approximate optimization problem using these samples. Defining $g(z) \doteq \nabla_\theta L(\theta_{old}, z)$ and $A(z) \doteq \nabla^2_\theta D(\theta_{old}, z)$, we would need to solve

$$\left(\frac{1}{|D|}\sum_{z\in D} A(z)\right) x = \frac{1}{|D|}\sum_{z\in D} g(z)$$

to obtain the search direction x. However, because the computation of the average Hessian is expensive, we sub-sample a batch $b \subset D$ to form it. As long as b is a large enough set, then the approximation

$$\frac{1}{|b|}\sum_{z\in b} A(z) \approx \frac{1}{|D|}\sum_{z\in D} A(z) \approx \underset{z\sim P}{E}[A(z)]$$

is good, and the search direction we obtain by solving

$$\left(\frac{1}{|b|}\sum_{z\in b} A(z)\right) x = \frac{1}{|D|}\sum_{z\in D} g(z)$$

is reasonable. The sub-sample ratio $|b|/|D|$ is a hyperparameter of the algorithm.

3.7 Experiment Details

3.7.1 Environments

The environments have the following state and action spaces: for the sparse MountainCar environment, $S \subseteq \mathbb{R}^2, A \subseteq \mathbb{R}^1$; for the sparse CartpoleSwingup task, $S \subseteq \mathbb{R}^4, A \subseteq \mathbb{R}^1$; for the sparse HalfCheetah task, $S \subset \mathbb{R}^{20}, A \subseteq \mathbb{R}^6$; for the sparse Swimmer task, $S \subseteq \mathbb{R}^{13}, A \subseteq \mathbb{R}^2$; for the SwimmerGather task, $S \subseteq \mathbb{R}^{33}, A \subseteq \mathbb{R}^2$; for the Atari RAM domain, $S \subseteq \mathbb{R}^{128}, A \subseteq \{1, ..., 18\}$.

For the sparse MountainCar task, the agent receives a reward of 1 only when it escapes the valley. For the sparse CartpoleSwingup task, the agent receives a reward of 1 only when $\cos(\beta) > 0.8$, with β the pole angle. For the sparse HalfCheetah task, the agent receives a reward of 1 when $x_{body} \geq 5$. For the sparse Swimmer task, the agent receives a reward of $1 + |v_{body}|$ when $|x_{body}| \geq 2$.

Atari RAM states, by default, take on values from 0 to 256 in integer intervals. We use a simple preprocessing step to map them onto values in $(-1/3, 1/3)$. Let x denote the raw RAM state, and s the preprocessed RAM state:

$$s = \frac{1}{3}\left(\frac{x}{128} - 1\right).$$

3.7.2 Policy and Value Functions

For all continuous control tasks we used fully-factored Gaussian policies, where the means of the action distributions were the outputs of neural networks, and the variances were separate trainable parameters. For the sparse MountainCar and sparse CartpoleSwingup tasks, the policy mean networks had a single hidden layer of 32 units. For sparse HalfCheetah, sparse Swimmer, and SwimmerGather, the policy mean networks were of size $(64, 32)$. For the Atari RAM tasks, we used categorical distributions over actions, produced by neural networks of size $(64, 32)$.

The value functions used for the sparse MountainCar and sparse CartpoleSwingup tasks were neural networks with a single hidden layer of 32 units. For sparse HalfCheetah, sparse Swimmer, and SwimmerGather, time-varying linear value functions were used, as described by Duan et al. [2016]. For the Atari RAM tasks, the value functions were neural networks of size $(64, 32)$. The neural network value functions were learned via single second-order step optimization; the linear baselines were obtained by least-squares fit at each iteration.

All neural networks were feed-forward, fully-connected networks with tanh activation units.

3.7.3 TRPO Hyperparameters

For all tasks, the MDP discount factor γ was fixed to 0.995, and generalized advantage estimators (GAE) [Schulman et al., 2016] were used, with the GAE λ parameter fixed to 0.95.

In the table below, we show several other TRPO hyperparameters. Batch size refers to steps of experience collected at each iteration. The sub-sample factor is for the second-order optimization step, as detailed in Section 3.6.

Environments	Batch Size	Sub-Sample	Max Rollout Length	δ_{KL}
Mountaincar, Cartpole Swingup	5000	1	500	0.01
HalfCheetah, Swimmer	5000	1	500	0.05
SwimmerGather	$50,000$	0.1	500	0.01
Pong	$10,000$	1	5000	0.01
Bankheist, Freeway	$13,500$	1	5000	0.01
Venture	$50,000$	0.2	7000	0.01

Table 3.1: TRPO hyperparameters for our experiments.

3.7.4 Exploration Hyperparameters

For all tasks, fully-factored Gaussian distributions were used as dynamics models, where the means and variances of the distributions were the outputs of neural networks.

For the sparse MountainCar and sparse CartpoleSwingup tasks, the means and variances were parametrized by single hidden layer neural networks with 32 units. For all other tasks, the means and variances were parametrized by neural networks with two hidden layers of size 64 units each. All networks used tanh activation functions.

For all continuous control tasks except SwimmerGather, we used replay memories of size $5,000,000$, and a KL-divergence step size of $\kappa = 0.001$. For SwimmerGather, the replay memory was the same size, but we set the KL-divergence size to $\kappa = 0.005$. For the Atari RAM domain tasks, we used replay memories of size $1,000,000$, and a KL-divergence step size of $\kappa = 0.01$.

For all tasks except SwimmerGather and Venture, 5000 time steps of experience were sampled from the replay memory at each iteration of dynamics model learning to take a stochastic step on (3.11), and a sub-sample factor of 1 was used in the second-order step optimizer. For SwimmerGather and Venture, $10,000$ time steps of experience were sampled at each iteration, and a sub-sample factor of 0.5 was used in the optimizer.

For all continuous control tasks, the L_2 penalty coefficient was set to $\alpha = 1$. For the Atari RAM tasks except for Venture, it was set to $\alpha = 0.01$. For Venture, it was set to $\alpha = 0.1$.

For all continuous control tasks except SwimmerGather, $\eta_0 = 0.001$. For SwimmerGather, $\eta_0 = 0.0001$. For the Atari RAM tasks, $\eta_0 = 0.005$.

3.8 Analysis of Speedup Compared to VIME

In this section, we provide an analysis of the time cost incurred by using VIME or our bonuses, and derive the potential magnitude of speedup attained by our bonuses versus VIME.

At each iteration, bonuses based on learned dynamics models incur two primary costs:

- the time cost of fitting the dynamics model,
- and the time cost of computing the rewards.

We denote the dynamics fitting costs for VIME and our methods as T^{fit}_{vime} and T^{fit}_{ours}. Although the Bayesian neural network dynamics model for VIME is more complex than our model, the fit times can work out to be similar depending on the choice of fitting algorithm. In our speed test, the fit times were nearly equivalent, but used different algorithms.

For the time cost of computing rewards, we first introduce the following quantities:

- n: the number of CPU threads available,
- t_f: time for a forward pass through the model,
- t_b: time for a backward pass through the model,
- N: batch size (number of samples per iteration),
- k: the number of forward passes that can be performed simultaneously.

For our method, the time cost of computing rewards is

$$T_{ours}^{rew} = \frac{Nt_f}{kn}.$$

For VIME, things are more complex. Each reward requires the computation of a *gradient* through its model, which necessitates a forward and a backward pass. Because gradient calculations cannot be efficiently parallelized by any deep learning toolkits currently available, each (s, a, s') tuple requires its own forward/backward pass. As a result, the time cost of computing rewards for VIME is:

$$T_{vime}^{rew} = \frac{N(t_f + t_b)}{n}.$$

The speedup of our method over VIME is therefore

$$\frac{T_{vime}^{fit} + \frac{N(t_f+t_b)}{n}}{T_{ours}^{fit} + \frac{Nt_f}{kn}}.$$

In the limit of large N, and with the approximation that $t_f \approx t_b$, the speedup is a factor of $\sim 2k$.

Chapter 4

Variational Option Discovery

We explore methods for option discovery based on variational inference and make two algorithmic contributions. First: we highlight a tight connection between variational option discovery methods and variational autoencoders, and introduce Variational Autoencoding Learning of Options by Reinforcement (VALOR), a new method derived from the connection. In VALOR, the policy encodes contexts from a noise distribution into trajectories, and the decoder recovers the contexts from the complete trajectories. Second: we propose a curriculum learning approach where the number of contexts seen by the agent increases whenever the agent's performance is strong enough (as measured by the decoder) on the current set of contexts. We show that this simple trick stabilizes training for VALOR and prior variational option discovery methods, allowing a single agent to learn many more modes of behavior than it could with a fixed context distribution. Finally, we investigate other topics related to variational option discovery, including fundamental limitations of the general approach and the applicability of learned options to downstream tasks.

4.1 Introduction

Humans are innately driven to explore new ways of interacting with their environments. This can accelerate the process of discovering skills for downstream tasks and can also be viewed as a primary objective in its own right. This drive serves as an inspiration for reward-free option discovery in reinforcement learning (based on the options framework of Sutton et al. [1999], Precup [2000]), where an agent tries to learn skills by interacting with its environment without trying to maximize cumulative reward for a particular task.

In this work, we explore variational option discovery, the space of methods for option discovery based on variational inference. We highlight a tight connection between prior work on variational option discovery and variational autoencoders (Kingma and Welling [2013]), and derive a new method based on the connection. In our analogy, a policy acts as an encoder, translating contexts from a noise distribution into trajectories; a decoder attempts to recover the contexts from the trajectories, and rewards the policies for making contexts

easy to distinguish. Contexts are random vectors which have no intrinsic meaning prior to training, but they become associated with trajectories as a result of training; each context vector thus corresponds to a distinct option. Therefore this approach learns a set of options which are as diverse as possible, in the sense of being as easy to distinguish from each other as possible. We show that Variational Intrinsic Control (VIC) (Gregor et al. [2016]) and the recently-proposed Diversity is All You Need (DIAYN) (Eysenbach et al. [2018b]) are specific instances of this template which decode from states instead of complete trajectories.

We make two main algorithmic contributions:

1. We introduce Variational Autoencoding Learning of Options by Reinforcement (VALOR), a new method which decodes from trajectories. The VALOR objective is designed to encourage learning dynamical modes instead of goal-attaining modes, e.g. 'move in a circle' instead of 'go to X'.

2. We propose a curriculum learning approach where the number of contexts seen by the agent increases whenever the agent's performance is strong enough (as measured by the decoder) on the current set of contexts.

We perform a comparison analysis of VALOR, VIC, and DIAYN with and without the curriculum trick, evaluating them in various robotics environments (point mass, cheetah, swimmer, ant).[1] We show that, to the extent that our metrics can measure, all three of them perform similarly, except that VALOR can attain qualitatively different behavior because of its trajectory-centric approach, and DIAYN learns more quickly because of its denser reward signal. We show that our curriculum trick stabilizes and speeds up learning for all three methods, and can allow a single agent to learn up to hundreds of modes. Beyond our core comparison, we also explore applications of variational option discovery in two interesting spotlight environments: a simulated robot hand and a simulated humanoid. Variational option discovery finds naturalistic finger-flexing behaviors in the hand environment, but performs poorly on the humanoid, in the sense that it does not discover natural crawling or walking gaits. We consider this evidence that pure information-theoretic objectives can do a poor job of capturing human priors on useful behavior in complex environments. Lastly, we try a proof-of-concept for applicability to downstream tasks in a variant of ant-maze by using a (particularly good) pretrained VALOR policy as the lower level of a hierarchy. In this experiment, we find that the VALOR policy is more useful than a random network as a lower level, and equivalently as useful as learning a lower level from scratch in the environment.

4.2 Related Work

Option Discovery: Substantial prior work exists on option discovery (Sutton et al. [1999], Precup [2000]); here we will restrict our attention to relevant recent work in the deep RL setting. Bacon et al. [2017] and Fox et al. [2017] derive policy gradient methods for

[1] Videos of learned behaviors are available at `varoptdisc.github.io`.

learning options: Bacon et al. [2017] learn options concurrently with solving a particular task, while Fox et al. [2017] learn options from demonstrations to accelerate specific-task learning. Vezhnevets et al. [2017] propose an architecture and training algorithm which can be interpreted as implicitly learning options. Thomas et al. [2017] find options as controllable factors in the environment. Machado et al. [2017a], Machado et al. [2017b], and Liu et al. [2017] learn *eigenoptions*, options derived from the graph Laplacian associated with the MDP. Several approaches for option discovery are primarily information-theoretic: Gregor et al. [2016], Eysenbach et al. [2018b], and Florensa et al. [2017] train policies to maximize mutual information between options and states or quantities derived from states; by contrast, we maximize information between options and whole trajectories. Hausman et al. [2018] learn skill embeddings by optimizing a variational bound on the entropy of the policy; the final objective function is closely connected with that of Florensa et al. [2017].

Universal Policies: Variational option discovery algorithms learn universal policies (goal- or instruction- conditioned policies), like universal value function approximators (Schaul et al. [2015]) and hindsight experience replay (Andrychowicz et al. [2017]). However, these other approaches require extrinsic reward signals and a hand-crafted instruction space. By contrast, variational option discovery is unsupervised and finds its own instruction space.

Intrinsic Motivation: Many recent works have incorporated intrinsic motivation (especially curiosity) into deep RL agents (Stadie et al. [2015], Houthooft et al. [2016], Bellemare et al. [2016], Achiam and Sastry [2016], Fu et al. [2017], Pathak et al. [2017], Ostrovski et al. [2017], Burda et al. [2018a]). However, none of these approaches were combined with learning universal policies, and so suffer from a problem of knowledge fade: when states cease to be interesting to the intrinsic reward signal (usually when they are no longer novel), unless they coincide with extrinsic rewards or are on a direct path to the next-most novel state, the agent will forget how to visit them.

Variational Autoencoders: Variational autoencoders (VAEs) (Kingma and Welling [2013]) learn a probabilistic encoder $q_\phi(z|x)$ and decoder $p_\theta(x|z)$ which map between data x and latent variables z by optimizing the evidence lower bound (ELBO) on the marginal distribution $p_\theta(x)$, assuming a prior $p(z)$ over latent variables. Higgins et al. [2017] extended the VAE approach by including a parameter β to control the capacity of z and improve the ability of VAEs to learn disentangled representations of high-dimensional data. The β-VAE optimization problem is

$$\max_{\phi,\theta} \mathop{\mathrm{E}}_{x\sim\mathcal{D}} \left[\mathop{\mathrm{E}}_{z\sim q_\phi(\cdot|x)} \left[\log p_\theta(x|z)\right] - \beta D_{KL}\left(q_\phi(z|x)||p(z)\right) \right], \tag{4.1}$$

and when $\beta = 1$, it reduces to the standard VAE of Kingma and Welling [2013].

Novelty Search: Option discovery algorithms based on the diversity of learned behaviors can be viewed as similar in spirit to novelty search (Lehman [2012]), an evolutionary algorithm which finds behaviors which are diverse with respect to a characterization function which is usually pre-designed but sometimes learned (as in Meyerson et al. [2016]).

4.3 Variational Option Discovery Algorithms

Our aim is to learn a policy π where action distributions are conditioned on both the current state s_t and a *context* c which is sampled at the start of an episode and kept fixed throughout. The context should uniquely specify a particular mode of behavior (also called a skill). But instead of using reward functions to ground contexts to trajectories, we want the meaning of a context to be arbitrarily assigned ('discovered') during training.

We formulate a learning approach as follows. A context c is sampled from a noise distribution G, and then encoded into a trajectory $\tau = (s_0, a_0, ..., s_T)$ by a policy $\pi(\cdot|s_t, c)$; afterwards c is decoded from τ with a probabilistic decoder D. If the trajectory τ is unique to c, the decoder will place a high probability on c, and the policy should be correspondingly reinforced. Supervised learning can be applied to the decoder (because for each τ, we know the ground truth c). To encourage exploration, we include an entropy regularization term with coefficient β. The full optimization problem is thus

$$\max_{\pi,D} \mathop{\mathrm{E}}_{c\sim G}\left[\mathop{\mathrm{E}}_{\tau\sim\pi,c}[\log P_D(c|\tau)] + \beta\mathcal{H}(\pi|c)\right], \tag{4.2}$$

where P_D is the distribution over contexts from the decoder, and the entropy term is $\mathcal{H}(\pi|c) \doteq \mathrm{E}_{\tau\sim\pi,c}\left[\sum_t H(\pi(\cdot|s_t, c))\right]$. We give a generic template for option discovery based on Eq. 4.2 as Algorithm 3. Observe that the objective in Eq. 4.2 has a one-to-one correspondence with the β-VAE objective in Eq. 4.1: the context c maps to the data x, the trajectory τ maps to the latent representation z, the policy π and the MDP together form the encoder q_ϕ, the decoder D maps to the decoder p_θ, and the entropy regularization $\mathcal{H}(\pi|c)$ maps to the KL-divergence of the encoder distribution from a prior where trajectories are generated by a uniform random policy (proof in Section 4.7). Based on this connection, we call algorithms for solving Eq. 4.2 variational option discovery methods.

Algorithm 3 Template for Variational Option Discovery with Autoencoding Objective

Generate initial policy π_{θ_0}, decoder D_{ϕ_0}
for $k = 0, 1, 2, ...$ **do**
  Sample context-trajectory pairs $\mathcal{D} = \{(c^i, \tau^i)\}_{i=1,...,N}$, by first sampling a context $c \sim G$ and then rolling out a trajectory in the environment, $\tau \sim \pi_{\theta_k}(\cdot|\cdot, c)$.
  Update policy with any reinforcement learning algorithm to maximize Eq. 4.2, using batch $\mathcal{D}$
  Update decoder by supervised learning to maximize $\mathrm{E}\left[\log P_D(c|\tau)\right]$, using batch $\mathcal{D}$
end for

4.3.1 Connections to Prior Work

Variational Intrinsic Control: Variational Intrinsic Control[2] (VIC) (Gregor et al. [2016]) is an option discovery technique based on optimizing a variational lower bound on the mutual

[2]Specifically, the algorithm presented as 'Intrinsic Control with Explicit Options' in Gregor et al. [2016].

information between the context and the final state in a trajectory, conditioned on the initial state. Gregor et al. [2016] give the optimization problem as

$$\max_{G,\pi,D} \mathop{\mathrm{E}}_{s_0\sim\mu} \left[\mathop{\mathrm{E}}_{\substack{c\sim G(\cdot|s_0) \\ \tau\sim\pi,c}} [\log P_D(c|s_0, s_T)] + H(G(\cdot|s_0)) \right], \tag{4.3}$$

where μ is the starting state distribution for the MDP. This differs from Eq. 4.2 in several ways: the context distribution G can be optimized, G depends on the initial state s_0, G is entropy-regularized, entropy regularization for the policy π is omitted, and the decoder only looks at the first and last state of the trajectory instead of the entire thing. However, they also propose to keep G fixed and state-independent, and do this in their experiments; additionally, their experiments use decoders which are conditioned on the final state only. This reduces Eq. 4.3 to Eq. 4.2 with $\beta = 0$ and $\log P_D(c|\tau) = \log P_D(c|s_T)$. We treat this as the canonical form of VIC and implement it this way for our comparison study.

Diversity is All You Need: Diversity is All You Need (DIAYN) (Eysenbach et al. [2018b]) performs option discovery by optimizing a variational lower bound for an objective function designed to maximize mutual information between context and *every* state in a trajectory, while minimizing mutual information between actions and contexts conditioned on states, and maximizing entropy of the mixture policy over contexts. The exact optimization problem is

$$\max_{\pi,D} \mathop{\mathrm{E}}_{c\sim G} \left[\mathop{\mathrm{E}}_{\tau\sim\pi,c} \left[\sum_{t=0}^{T} (\log P_D(c|s_t) - \log G(c)) \right] + \beta\mathcal{H}(\pi|c) \right]. \tag{4.4}$$

In DIAYN, G is kept fixed (as in canonical VIC), so the term $\log G(c)$ is constant and may be removed from the optimization problem. Thus Eq. 4.4 is a special case of Eq. 4.2 with $\log P_D(c|\tau) = \sum_{t=0}^{T} \log P_D(c|s_t)$.

4.3.2 VALOR

In this section, we propose Variational Autoencoding Learning of Options by Reinforcement (VALOR), a variational option discovery method which directly optimizes Eq. 4.2 with two key decisions about the decoder:

- The decoder never sees actions. Our conception of 'interesting' behaviors requires that the agent attempt to interact with the environment to achieve some change in state. If the decoder was permitted to see raw actions, the agent could signal the context directly through its actions and ignore the environment. Limiting the decoder in this way forces the agent to manipulate the environment to communicate with the decoder.
- Unlike in DIAYN, the decoder does *not* decompose as a sum of per-timestep computations. That is, $\log P_D(c|\tau) \neq \sum_{t=0}^{T} f(s_t, c)$. We choose against this decomposition because it could limit the ability of the decoder to correctly distinguish between behaviors which share some states, or behaviors which share all states but reach them in different orders.

We implement VALOR with a recurrent architecture for the decoder (Fig. 4.1), using a bidirectional LSTM to make sure that both the beginning and end of a trajectory are equally important. We only use $N = 11$ equally spaced observations from the trajectory as inputs, for two reasons: 1) computational efficiency, and 2) to encode a heuristic that we are only interested in low-frequency behaviors (as opposed to information-dense high-frequency jitters). Lastly, taking inspiration from Vezhnevets et al. [2017], we only decode from the k-step *transitions* (deltas) in state space between the N observations. Intuitively, this corresponds to a prior that agents should move, as any two modes where the agent stands still in different poses will be indistinguishable to the decoder (because the deltas will be identically zero). We do not decode from transitions in VIC or DIAYN, although we note it would be possible and might be interesting future work.

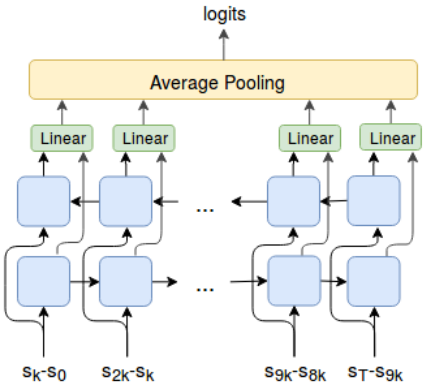


Figure 4.1: Bidirectional LSTM architecture for VALOR decoder. Blue blocks are LSTM cells.

4.3.3 Curriculum Approach

The standard approach for context distributions, used in VIC and DIAYN, is to have K discrete contexts with a uniform distribution: $c \sim \text{Uniform}(K)$. In our experiments, we found that this worked poorly for large K across all three algorithms we compared. Even with very large batches (to ensure that each context was sampled often enough to get a low-variance contribution to the gradient), training was challenging. We found a simple trick to resolve this issue: start training with small K (where learning is easy), and gradually increase it over time as the decoder gets stronger. Whenever $\mathrm{E}\left[\log P_D(c|\tau)\right]$ is high enough (we pick a fairly arbitrary threshold of $P_D(c|\tau) \approx 0.86$), we increase K according to

$$K \leftarrow \min\left(\text{int}\left(1.5 \times K + 1\right), K_{max}\right), \tag{4.5}$$

where K_{max} is a hyperparameter. As our experiments show, this curriculum leads to faster and more stable convergence.

4.4 Experiments

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

- What are best practices for training agents with variational option discovery algorithms (VALOR, VIC, DIAYN)? Does the curriculum learning approach help?
- What are the qualitative results from running variational option discovery algorithms? Are the learned behaviors recognizably distinct to a human? Are there substantial differences between algorithms?

- Are the learned behaviors useful for downstream control tasks?

Test environments: Our core comparison experiments is on a slate of locomotion environments: a custom 2D point agent, the HalfCheetah and Swimmer robots from the OpenAI Gym [Brockman et al., 2016], and a customized version of Ant from Gym where contact forces are omitted from the observations. We also tried running variational option discovery on two other interesting simulated robots: a dextrous hand (with $\mathcal{S} \in \mathbb{R}^{48}$ and $\mathcal{A} \in \mathbb{R}^{20}$, based on Plappert et al. [2018]), and a new complex humanoid environment we call 'toddler' (with $\mathcal{S} \in \mathbb{R}^{335}$ and $\mathcal{A} \in \mathbb{R}^{35}$). Lastly, we investigated applicability to downstream tasks in a modified version of Ant-Maze (Frans et al. [2018]).

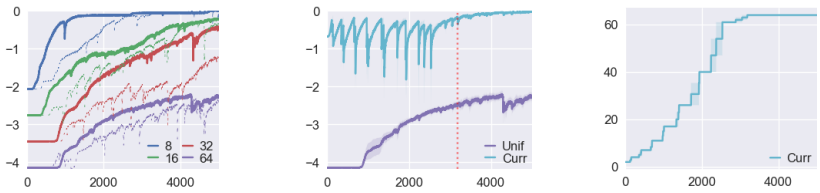
Implementation: We implement VALOR, VIC, and DIAYN with vanilla policy gradient as the RL algorithm (described in Section 4.8.1). We note that VIC and DIAYN were originally implemented with different RL algorithms: Gregor et al. [2016] implemented VIC with tabular Q learning (Watkins and Dayan [1992]), and Eysenbach et al. [2018b] implemented DIAYN with soft actor-critic (Haarnoja et al. [2018a]). Also unlike prior work, we use recurrent neural network policy architectures. Because there is not a final objective function to measure whether an algorithm has achieved qualitative diversity of behaviors, our hyperparameters are based on what resulted in stable training, and kept constant across algorithms. Because the design space for these algorithms is very large and evaluation is to some degree subjective, we caution that our results should not necessarily be viewed as definitive.

Training techniques: We investigated two specific techniques for training: curriculum generation via Eq. 4.5, and context embeddings. On context embeddings: a natural approach for providing the integer context as input to a neural network policy is to convert the context to a one-hot vector and concatenate it with the state, as in Eysenbach et al. [2018b]. Instead, we consider whether training is improved by allowing the agent to learn its own embedding vector for each context.

4.5 Results

Exploring Optimization Techniques: We present partial findings for our investigation of training techniques in Fig. 4.2 (showing results for just VALOR), with complete findings in Section 4.9. In Fig. 4.2a, we compare performance with and without embeddings, using a uniform context distribution, for several choices of K (the number of contexts). We find that using embeddings consistently improves the speed and stability of training. Fig. 4.2a also illustrates that training with a uniform distribution becomes more challenging as K increases. In Figs. 4.2b and 4.2c, we show that agents with the curriculum trick and embeddings achieve mastery on $K_{max} = 64$ contexts substantially faster than the agents trained with uniform context distributions in Fig. 4.2a. As shown in Section 4.9, these results are consistent across algorithms.

Comparison Study of Qualitative Results: In our comparison, we tried to assess whether variational option discovery algorithms learn an interesting set of behaviors. This is subjective and hard to measure, so we restricted ourselves to testing for behaviors which are easy to



(a) Uniform, for various K (b) Uniform vs Curriculum (c) Curriculum, current K

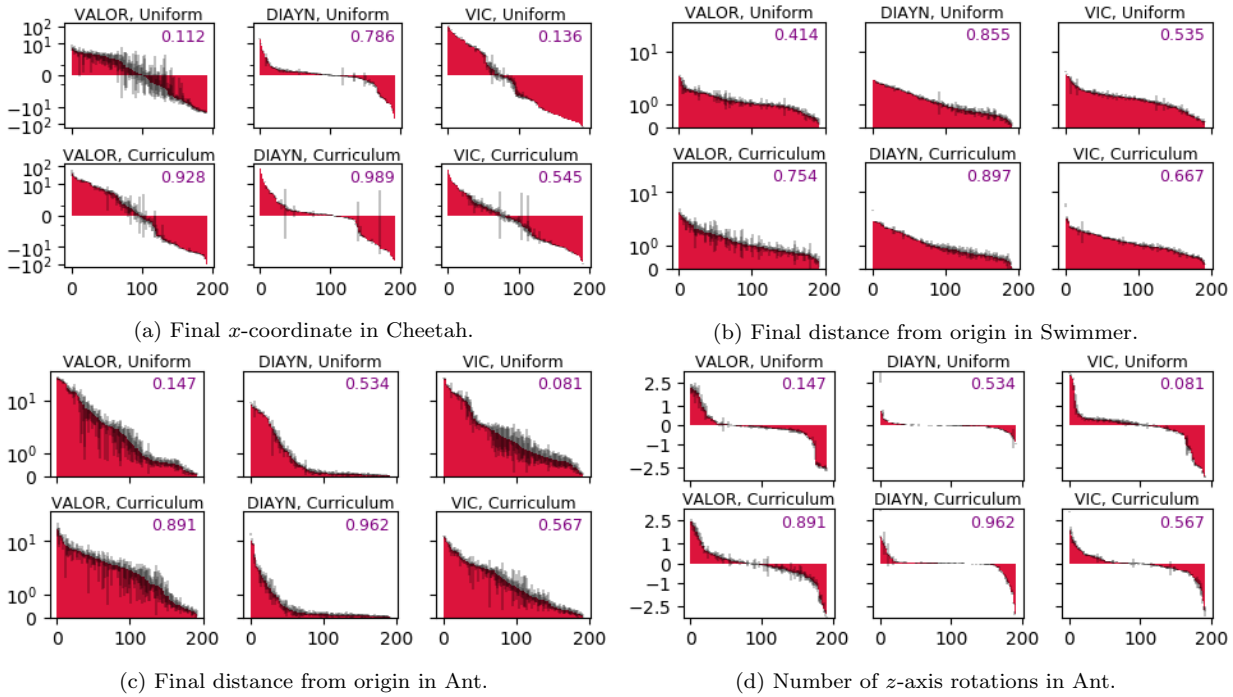
Figure 4.2: Studying optimization techniques with VALOR in HalfCheetah, showing performance—in (a) and (b), $\mathrm{E}[\log P_D(c|\tau)]$; in (c), the value of K throughout the curriculum—vs training iteration. (a) compares learning curves with and without context embeddings (solid vs dotted, resp.), for $K \in \{8, 16, 32, 64\}$, with uniform context distributions. (b) compares curriculum (with $K_{max} = 64$) to uniform (with $K = 64$) context distributions, using embeddings for both. The dips for the curriculum curve indicate when K changes via Eq. 4.5; values of K are shown in (c). The dashed red line shows when $K = K_{max}$ for the curriculum; after it, the curves for Uniform and Curriculum can be fairly compared. All curves are averaged over three random seeds.

quantify or observe; we note that there is substantial room in this space for developing performance metrics, and consider this an important avenue for future research.

We trained agents by VALOR, VIC, and DIAYN, with embeddings and $K = 64$ contexts, with and without the curriculum trick. We evaluated the learned behaviors by measuring the following quantities: final x-coordinate for Cheetah, final distance from origin for Swimmer, final distance from origin for Ant, and number of z-axis rotations for Ant[3]. We present partial findings in Fig. 4.3 and complete results in Section 4.10. Our results confirm findings from prior work, including Eysenbach et al. [2018b] and Florensa et al. [2017]: variational option discovery methods, in some MuJoCo environments, are able to find locomotion gaits that travel in a variety of speeds and directions. Results in Cheetah and Ant are particularly good by this measure; in Swimmer, fairly few behaviors actually travel any meaningful distance from the origin (> 3 units), but it happens non-negligibly often. All three algorithms produce similar results in the locomotion domains, although we do find slight differences: particularly, DIAYN is more prone than VALOR and VIC to learn behaviors like 'attain target state,' where the target state is fixed and unmoving. Our DIAYN behaviors are overall less mobile than the results reported by Eysenbach et al. [2018b]; we believe that this is due to qualitative differences in how entropy is maximized by the underlying RL algorithms (soft actor-critic vs. entropy-regularized policy gradients).

We find that the curriculum approach does not appear to change the diversity of behaviors discovered in any large or consistent way. It appears to slightly increase the ranges for Cheetah x-coorindate, while slightly decreasing the ranges for Ant final distance. Scrutinizing the X-Y traces for all learned modes, it seems (subjectively) that the curriculum approach causes agents to move more erratically (see Appendices D.11—D.14). We do observe a particularly interesting effect for robustness: the curriculum approach makes the distribution

[3]Approximately the number of complete circles walked by the agent around the ground-fixed z-axis (but not necessarily around the origin).



(a) Final x-coordinate in Cheetah.

(b) Final distance from origin in Swimmer.

(c) Final distance from origin in Ant.

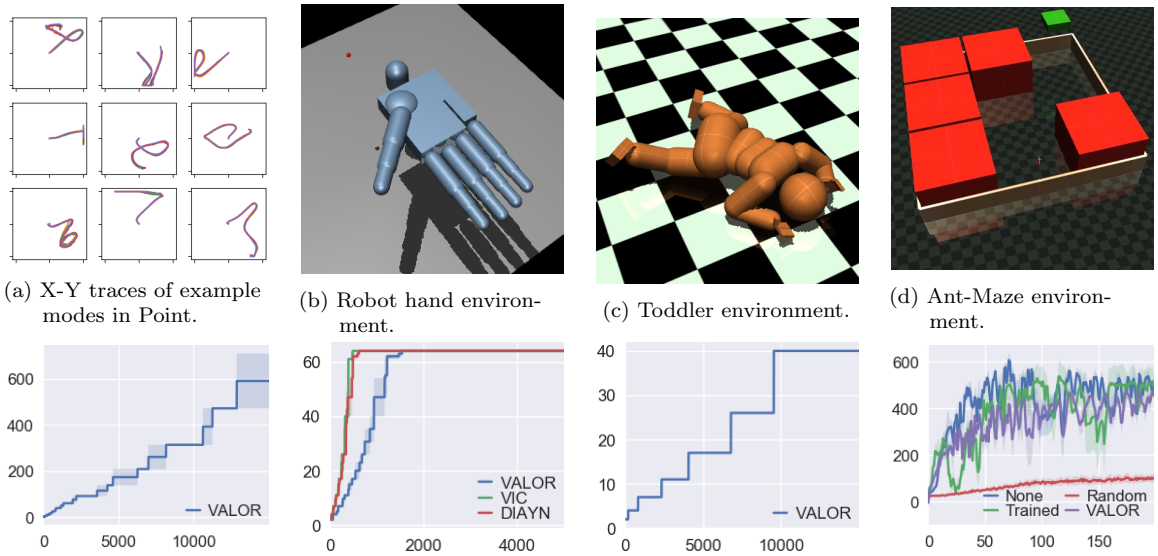
(d) Number of z-axis rotations in Ant.

Figure 4.3: Bar charts illustrating scores for behaviors in Cheetah, Swimmer, and Ant, with x-axis showing behavior ID and y-axis showing the score in log scale. Each red bar (width 1 on the x-axis) gives the average score for 5 trajectories conditioned on a single context; each chart is a composite from three random seeds, each of which was run with $K = 64$ contexts, for a total of 192 behaviors represented per chart. Behaviors were sorted in descending order by average score. Black bars show the standard deviation in score for a given behavior (context), and the upper-right corner of each chart shows the average decoder probability $\mathrm{E}[P_D(\tau|c)]$.

of scores more consistent between random seeds (for performances of all seeds separately, see Appendices D.3—D.10).

We also attempted to perform a baseline comparison of all three variational option discovery methods against an approach where we used random reward functions in place of a learned decoder; however, we encountered substantial difficulties in optimizing with random rewards. The details of these experiments are given in Section 4.11.

Hand and Toddler Environments: Optimizing in the Hand environment (Fig. 4.4f) was fairly easy and usually produced some naturalistic behaviors (eg pointing, bringing thumb and forefinger together, and one common rude gesture) as well as various unnatural behaviors (hand splayed out in what would be painful poses). Optimizing in the Toddler environment (Fig. 4.4g) was highly challenging; the agent frequently struggled to learn more than a handful of behaviors. The behaviors which the agent did learn were extremely unnatural. We believe that this is because of a fundamental limitation of purely information-theoretic RL objectives: humans have strong priors on what constitutes natural behavior, but for sufficiently complex systems, those behaviors form a set of measure zero in the space of all possible behaviors;



(a) X-Y traces of example modes in Point.

(b) Robot hand environment.

(c) Toddler environment.

(d) Ant-Maze environment.

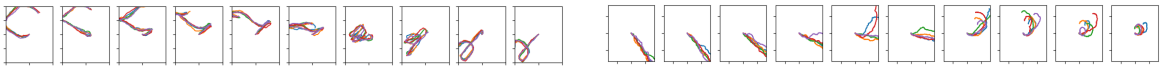
(e) Point, current K.

(f) Hand, current K.

(g) Toddler, current K.

(h) Ant-Maze return.

Figure 4.4: Various figures for spotlight experiments. Figs. 4.4a and 4.4e show results from learning hundreds of behaviors in the Point env, with $K_{max} = 1024$. Fig. 4.4f shows that optimizing Eq. 4.2 in the Hand environment is quite easy with the curriculum approach; all agents master the $K_{max} = 64$ contexts in < 2000 iterations. Fig. 4.4g illustrates the challenge for variational option discovery in Toddler: after 15000 iterations, only $K = 40$ behaviors have been learned. Fig. 4.4d shows the Ant-Maze environment, where red obstacles prevent the ant from reaching the green goal. Fig. 4.4h shows performance in Ant-Maze for different choices of a low-level policy in a hierarchy; in the Random and VALOR experiments, the low-level policy receives no gradient updates.



(a) Interpolating behavior in the point environment.

(b) Interpolating behavior in the ant environment.

Figure 4.5: Plots on the far left and far right show X-Y traces for behaviors learned by VALOR; in-between plots show the X-Y traces conditioned on interpolated contexts.

when a purely information-theoretic objective function is used, it will give no preference to the behaviors humans consider natural.

Learning Hundreds of Behaviors: Via the curriculum approach, we are able to train agents in the Point environment to learn hundreds of behaviors which are distinct according to the decoder (Fig. 4.4e). We caution that this does not necessarily expand the space of behaviors which are learnable—it may merely allow for increasingly fine-grained binning of already-learned behaviors into contexts. From various experiments prior to our final results, we developed an intuition that it was important to carefully consider the capacity of the decoder here: the greater the decoder's capacity, the more easily it would overfit to undetectably-small differences in trajectories.

Mode Interpolation: We experimented with interpolating between context embeddings for point and ant policies to see if we could obtain interpolated behaviors. As shown in Fig. 4.5, we found that some reasonably smooth interpolations were possible. This suggests that even though only a discrete number of behaviors are trained, the training procedure learns general-purpose universal policies.

Downstream Tasks: We investigated whether behaviors learned by variational option discovery could be used for a downstream task by taking a policy trained with VALOR on the Ant robot (Uniform distribution, seed 10; see Appendix D.7), and using it as the lower level of a two-level hierarchical policy in Ant-Maze. We held the VALOR policy fixed throughout downstream training, and only trained the upper level policy, using A2C as the RL algorithm (with reinforcement occuring only at the lower level—the upper level actions were trained by signals backpropagated through the lower level). Results are shown in Fig. 4.4h. We compared the performance of the VALOR-based agent to three baselines: a hierarchical agent with the same architecture trained from scratch on Ant-Maze ('Trained' in Fig. 4.4h), a hierarchical agent with a fixed random network as the lower level ('Random' in Fig. 4.4h), and a non-hierarchical agent with the same architecture as the upper level in the hierarchical agents (an MLP with one hidden layer, 'None' in Fig. 4.4h). We found that the VALOR agent worked as well as the hierarchy trained from scratch and the non-hierarchical policy, with qualitatively similar learning curves for all three; the fixed random network performed quite poorly by comparison. This indicates that the space of options learned by (the particular run of) VALOR was at least as expressive as primitive actions, for the purposes of the task, and that VALOR options were more expressive than random networks here.

4.6 Conclusions

We performed a thorough empirical examination of variational option discovery techniques, and found they produce interesting behaviors in a variety of environments (such as Cheetah, Ant, and Hand), but can struggle in very high-dimensional control, as shown in the Toddler environment. From our mode interpolation and hierarchy experiments, we found evidence that the learned policies are universal in meaningful ways; however, we did not find clear evidence that hierarchies built on variational option discovery would outperform task-specific policies learned from scratch.

We found that with purely information-theoretic objectives, agents in complex environments will discover behaviors that encode the context in trivial ways—eg through tiling a narrow volume of the state space with contexts. Thus a key challenge for future variational option discovery algorithms is to make the decoder distinguish between trajectories in a way which corresponds with human intuition about meaningful differences.

4.7 VAE-Equivalence Proof

The KL-divergence of $P(\tau|\pi, c)$ from $P(\tau|\pi_0)$ is

$$\begin{aligned} D_{KL}\left(P(\tau|\pi,c)||P(\tau|\pi_0)\right) &= \mathop{\mathrm{E}}_{\tau\sim\pi,c}\left[\log\frac{P(\tau|\pi,c)}{P(\tau|\pi_0)}\right] \\ &= \mathop{\mathrm{E}}_{\tau\sim\pi,c}\left[\log\frac{\mu(s_0)\prod_{t=0}^{T-1}P(s_{t+1}|s_t,a_t)\pi(a_t|s_t,c)}{\mu(s_0)\prod_{t=0}^{T-1}P(s_{t+1}|s_t,a_t)\pi_0(a_t|s_t)}\right] \\ &= \mathop{\mathrm{E}}_{\tau\sim\pi,c}\left[\sum_{t=0}^{T-1}\log\pi(a_t|s_t,c)-\log\pi_0(a_t|s_t)\right] \\ &= -\mathcal{H}(\pi,c)-\mathop{\mathrm{E}}_{\tau\sim\pi,c}\left[\sum_{t=0}^{T-1}\log\pi_0(a_t|s_t)\right]. \end{aligned}$$

The first term is our entropy regularization term. The second term, for a uniform random policy π_0, is a constant independent of π (as long as T is the same for all episodes) and can thus be removed from the objective function without changing the optimization problem.

4.8 Experiment Details

4.8.1 Policy Optimization Algorithm

In this section, we will describe how we performed policy optimization for our experiments. We used vanilla policy gradient to optimize the reinforcement objective for all three variational

option discovery algorithms,

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathrm{E}}_{\substack{c\sim G \\ \tau\sim\pi,c}}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t,c)\hat{A}_t\right],$$

although details varied slightly between algorithms and environments. The variation between environments was due to the presence or absence of extrinsic rewards. In all environments except for Ant, there were no extrinsic rewards; however, in Ant, a small penalty was applied for falling over (as opposed to terminating the episode when the agent falls over, as in Eysenbach et al. [2018b]).

- For VALOR and VIC, the advantage function was:

  $$\hat{A}_t = \text{normalize}\left(\log P_D(c|\tau)\right) + \text{normalize}\left(\sum_{t'=t}^{T}\left(\gamma^{t'-t} r_{t'} - V_\psi(s_t,c)\right)\right),$$

  where the normalize function subtracts out the batch mean and divides by the batch standard deviation, and V_ψ was a learned value function baseline. $V_\psi(s_t,c)$ was learned by taking one gradient descent step on

  $$\min_\psi \sum_{(s_t,c)\in\mathcal{D}}\left(\gamma^{t'-t} r_{t'} - V_\psi(s_t,c)\right)^2$$

  per iteration.

- For DIAYN, the advantage function was:

  $$\hat{A}_t = \text{normalize}\left(\sum_{t'=t}^{T}\left(\gamma^{t'-t}\left(\log P_D(c|s_{t'}) + r_{t'}\right) - V_\psi(s_t,c)\right)\right)$$

  where $V_\psi(s_t,c)$ was learned by descending on

  $$\min_\psi \sum_{(s_t,c)\in\mathcal{D}}\left(\gamma^{t'-t}\left(\log P_D(c|s_{t'}) + r_{t'}\right) - V_\psi(s_t,c)\right)^2.$$

When computing the gradient of the entropy term, we made an approximation that ignored the role of π in the distribution over trajectories:

$$\begin{aligned}\nabla_\theta \mathcal{H}(\pi,c) &= \nabla_\theta \sum_{t=0}^{T-1} \mathop{\mathrm{E}}_{s_t\sim\pi,c}\left[H(\pi(\cdot|s_t,c))\right] \\ &\approx \sum_{t=0}^{T-1} \mathop{\mathrm{E}}_{s_t\sim\pi,c}\left[\nabla_\theta H(\pi(\cdot|s_t,c))\right],\end{aligned}$$

resulting in the same entropy regularization as in Mnih et al. [2016]. Following practices for vanilla policy gradient established in Duan et al. [2016], we use the Adam optimizer Kingma and Ba [2015].

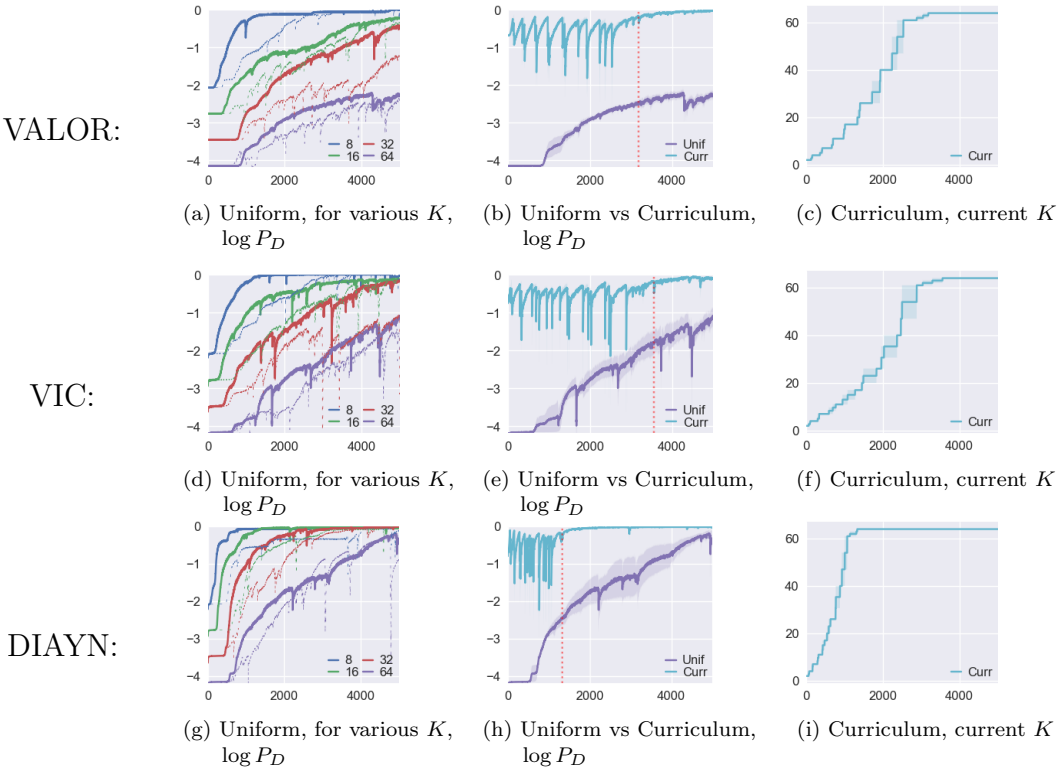
4.8.2 Hyperparameters

For all variational option discovery algorithms, we used:

- 1000 paths per epoch for the policy gradient batch
- $\gamma = 0.97$ as the discount factor
- $\beta = 1e^{-3}$ as the entropy regularization coefficient, where applicable (omitted for VIC)
- $1e^{-3}$ as the Adam learning rate
- LSTM(64) followed by MLP(32) with tanh activations as the policy architecture
- 32 as the context embedding dimension (when using context embeddings)

For VALOR, the decoder was a bidirectional LSTM where the cell for each direction was of size 64. For VIC and DIAYN, the decoder was an MLP of size $(180, 180)$.

4.9 Additional Analysis for Best Practices



(a) Uniform, for various K, $\log P_D$ (b) Uniform vs Curriculum, $\log P_D$ (c) Curriculum, current K

(d) Uniform, for various K, $\log P_D$ (e) Uniform vs Curriculum, $\log P_D$ (f) Curriculum, current K

(g) Uniform, for various K, $\log P_D$ (h) Uniform vs Curriculum, $\log P_D$ (i) Curriculum, current K

Figure 4.6: Analysis for understanding best training practices for various algorithms with HalfCheetah as the environment. The x-axis is number of training iterations, and in (a) and (b), the y-axis is $\mathrm{E}[\log P_D(c|\tau)]$; in (c), the y-axis gives the current value of K in the curriculum. (a) shows a direct comparison between learning curves with (dark) and without (dotted) context embeddings, for $K \in \{8, 16, 32, 64\}$. (b) shows learning performance for the curriculum approach with $K_{max} = 64$, compared against the uniform distribution approach with $K = 64$: the spikes and dips for the curriculum curve are characteristic of points when K changes according to Eq. 4.5. The dashed red line shows when $K = K_{max}$ for the curriculum approach; prior to it, the curves for Uniform and Curriculum are not directly comparable, but after it, they are. (c) shows K for the curriculum approach throughout the runs from (b). All curves are averaged over three random seeds.

4.10 Complete Experimental Results for Comparison Study

4.10.1 Guide to Reading This Section

In this section we present the results from our core comparison of {VALOR, VIC, DIAYN} × {Uniform, Curriculum}. Because these algorithms perform unsupervised behavior discovery, analyzing our results is highly-challenging: there is no single, quantitative measure by which to compare the algorithms. We choose to examine our results in a variety of ways:

- Learning curves for the optimization objective.
- Bar charts and histograms to show scores for the learned behaviors. Particularly, we evaluate final x-coordinate in the Cheetah environment, final distance traveled in the Swimmer environment, final distance traveled in the Ant environment, and number of z-axis rotations in the Ant environment. Scores are evaluated on trajectories of length $T = 1000$ steps, even though agents are trained on trajectories with $T = 250$; we find that using longer horizons at test time clarifies the differences between behaviors.
- X-Y traces for agent trajectories in the Point and Ant environments. (X-Y traces for the center-of-mass in Swimmer are not very insightful: Swimmer behavior is highly oscillatory and so it is difficult to discern what is happening.)

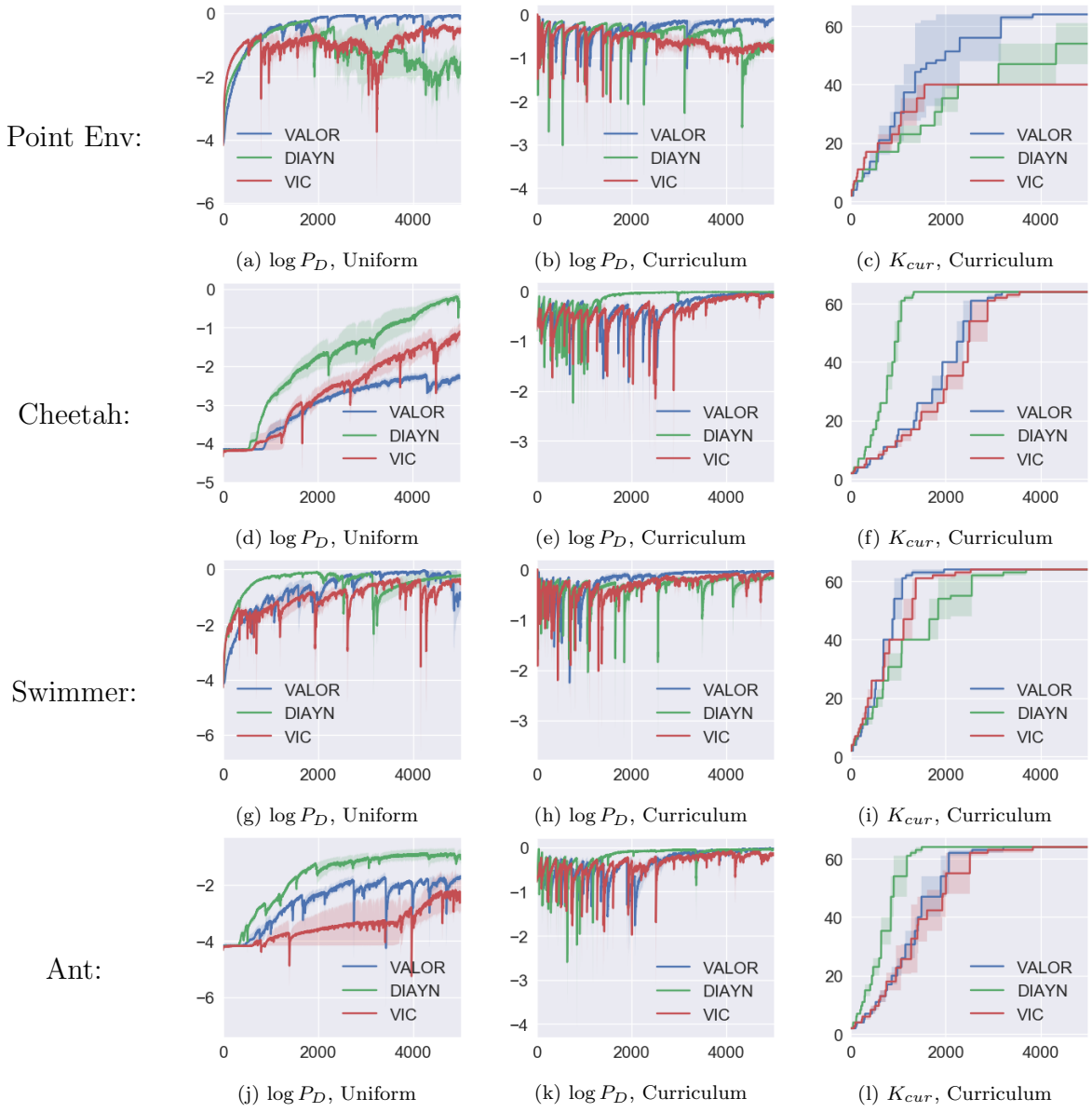
Regarding the bar charts and histograms in subsections 4.10.3—4.10.6:

- The bar charts are arranged in nearly the same way as the charts in 4.3: the x-axis is behavior ID, and the y-axis shows score in log scale for that behavior. The black bars show standard deviations for behavior scores.
- The histograms show score on the x-axis, and number of behaviors that fall into a given bin on the y-axis in log scale.
- The charts for 'all' show the composite bars for all behaviors from seeds 0, 10, and 20. The 's0', 's10', and 's20' charts show behaviors from particular random seeds. Each single seed corresponds to a single policy with $K = 64$ behaviors.

Regarding the X-Y traces in subsections 4.10.7—4.10.10:

- In the Point traces, the ranges for x and y are $x \in [-1.3, 1.3]$ and $y \in [-1.3, 1.3]$.
- In the Ant traces, the ranges for x and y are $x \in [-15, 15]$ and $y \in [-15, 15]$.
- For the Point environment, traces are taken from trajectories with the same time horizon as training ($T = 65$); for the Ant environment, we use the $T = 1000$ trajectories.

4.10.2 Learning Curves



(a) $\log P_D$, Uniform (b) $\log P_D$, Curriculum (c) K_{cur}, Curriculum

(d) $\log P_D$, Uniform (e) $\log P_D$, Curriculum (f) K_{cur}, Curriculum

(g) $\log P_D$, Uniform (h) $\log P_D$, Curriculum (i) K_{cur}, Curriculum

(j) $\log P_D$, Uniform (k) $\log P_D$, Curriculum (l) K_{cur}, Curriculum

Figure 4.7: Learning curves for all algorithms and environments in our core comparison, for number of contexts $K = 64$. The curriculum trick generally tends to speed up and stabilize performance, except for DIAYN and VIC in the point environment.

4.10.3 Evaluating Learned Behaviors: Cheetah

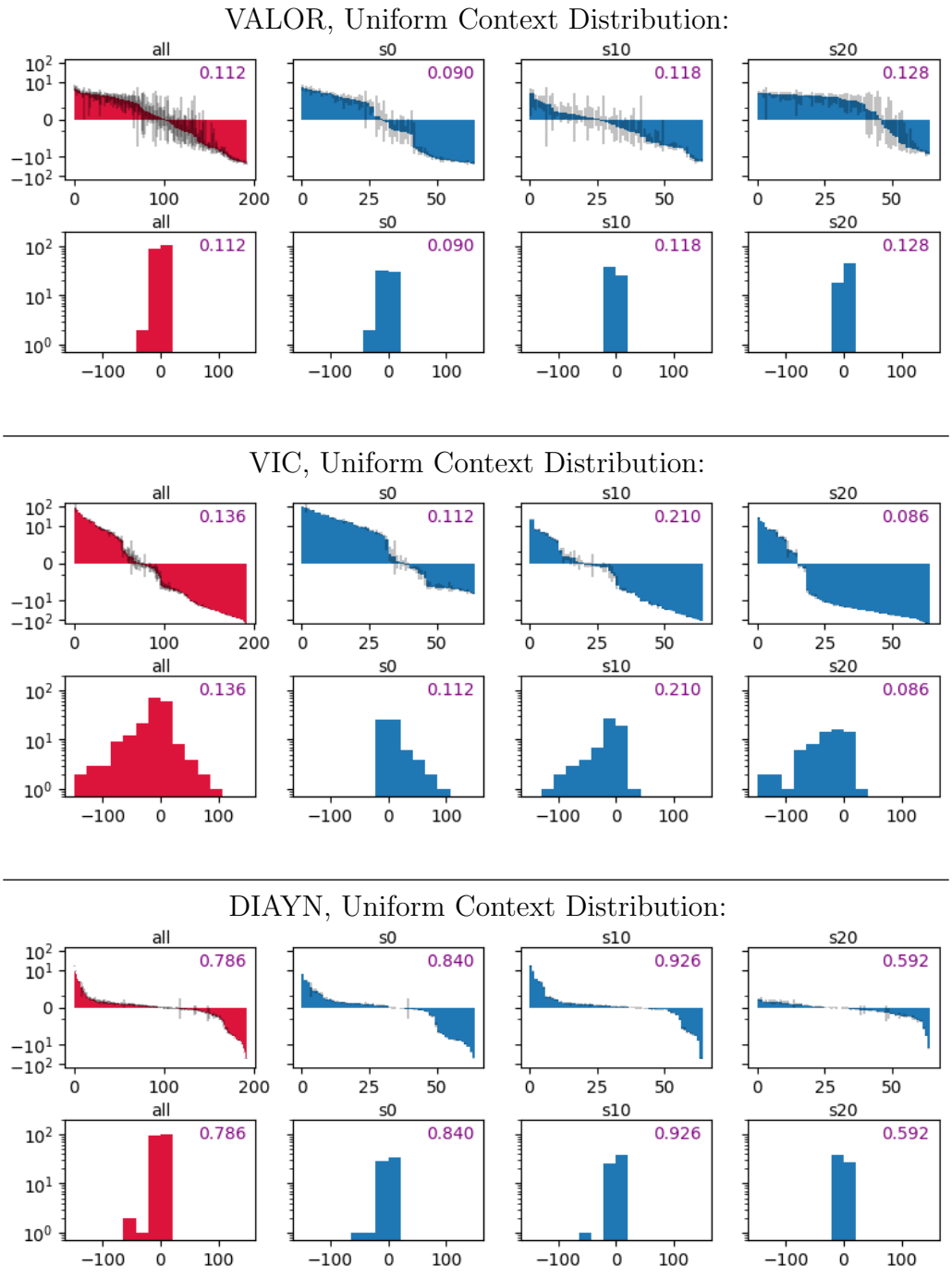


Figure 4.8: Final x-coordinate in the Cheetah environment.

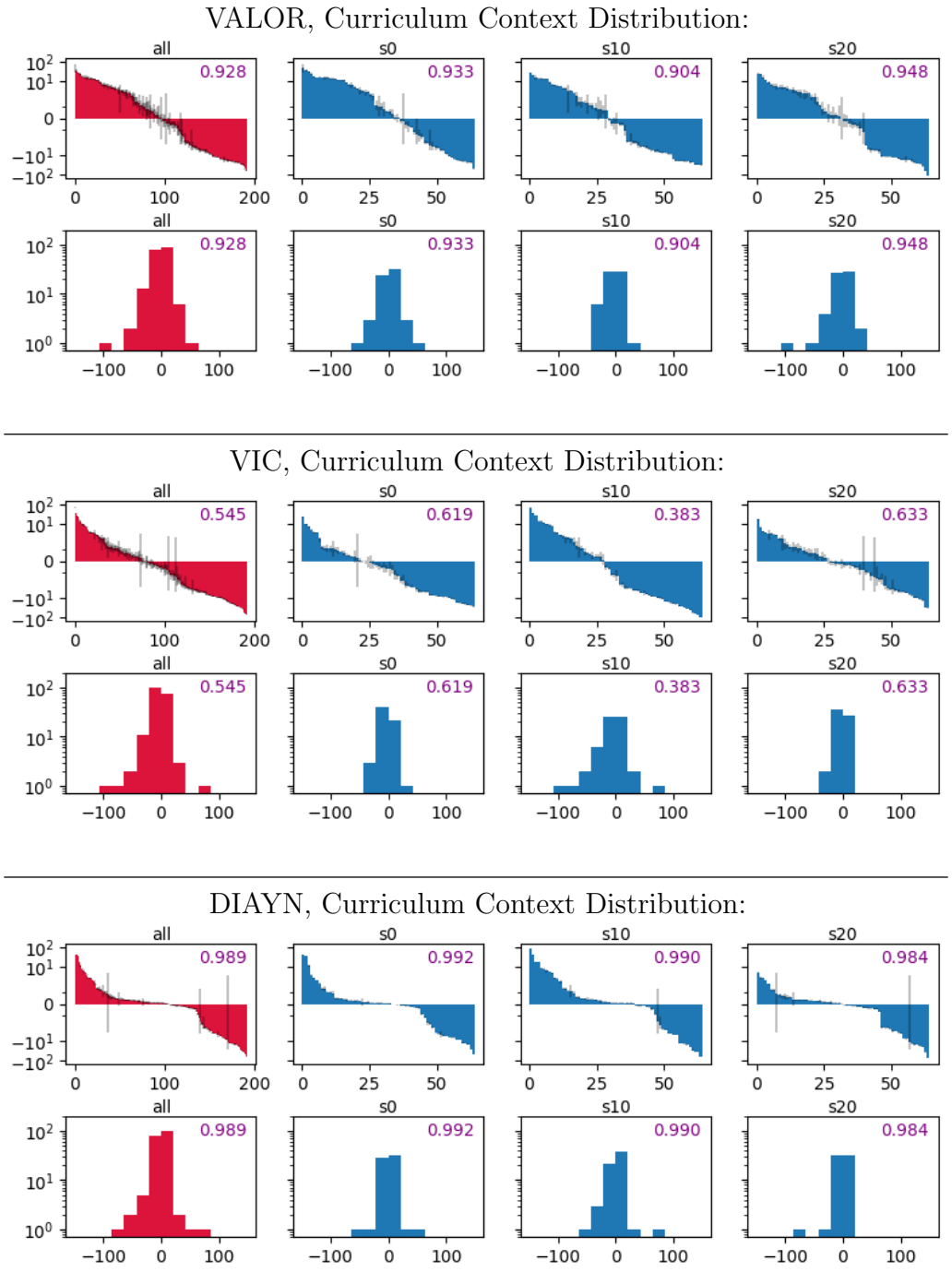


Figure 4.9: Final x-coordinate in the Cheetah environment.

4.10.4 Evaluating Learned Behaviors: Swimmer

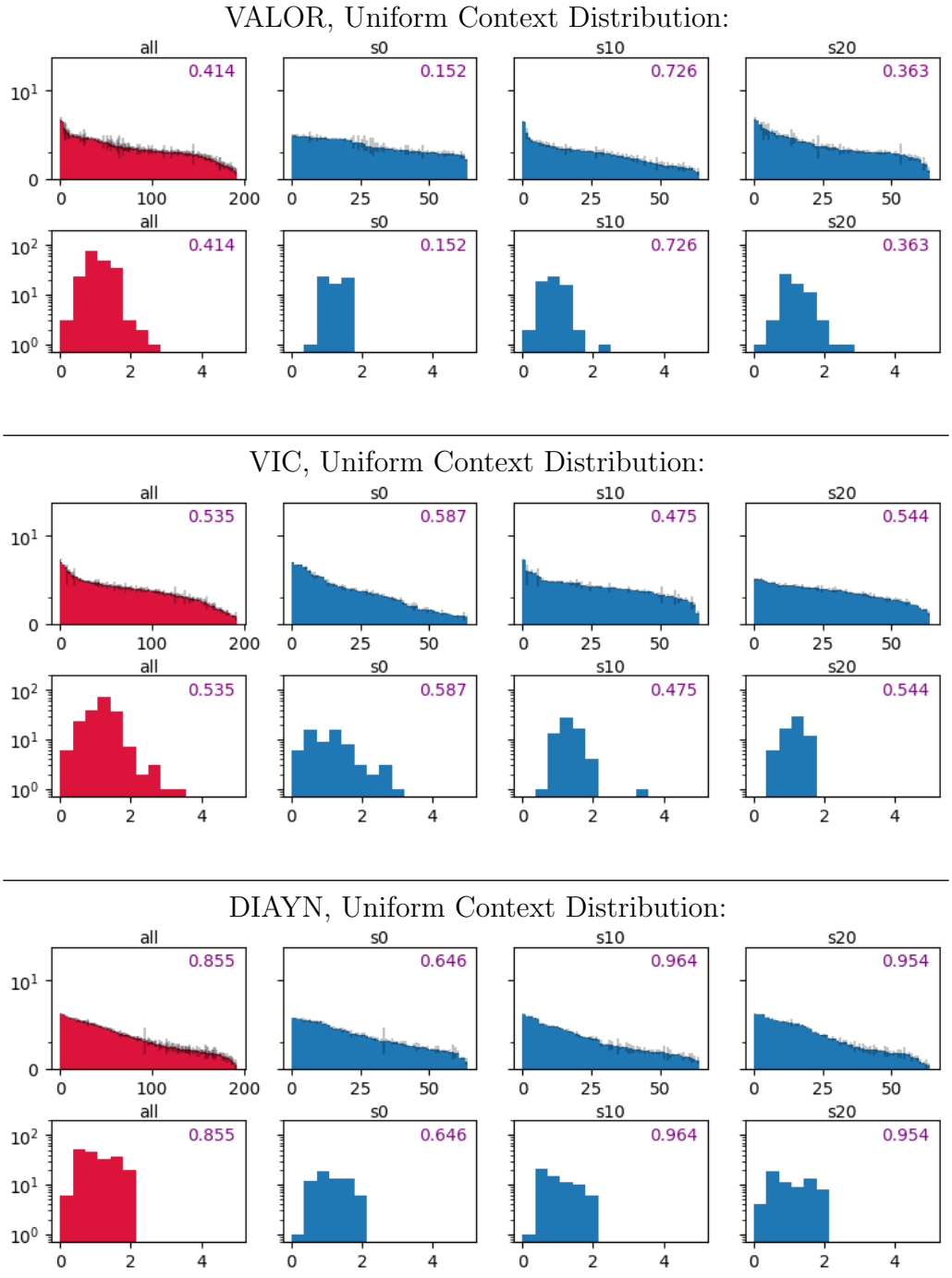


Figure 4.10: Final distance from origin in the Swimmer environment.

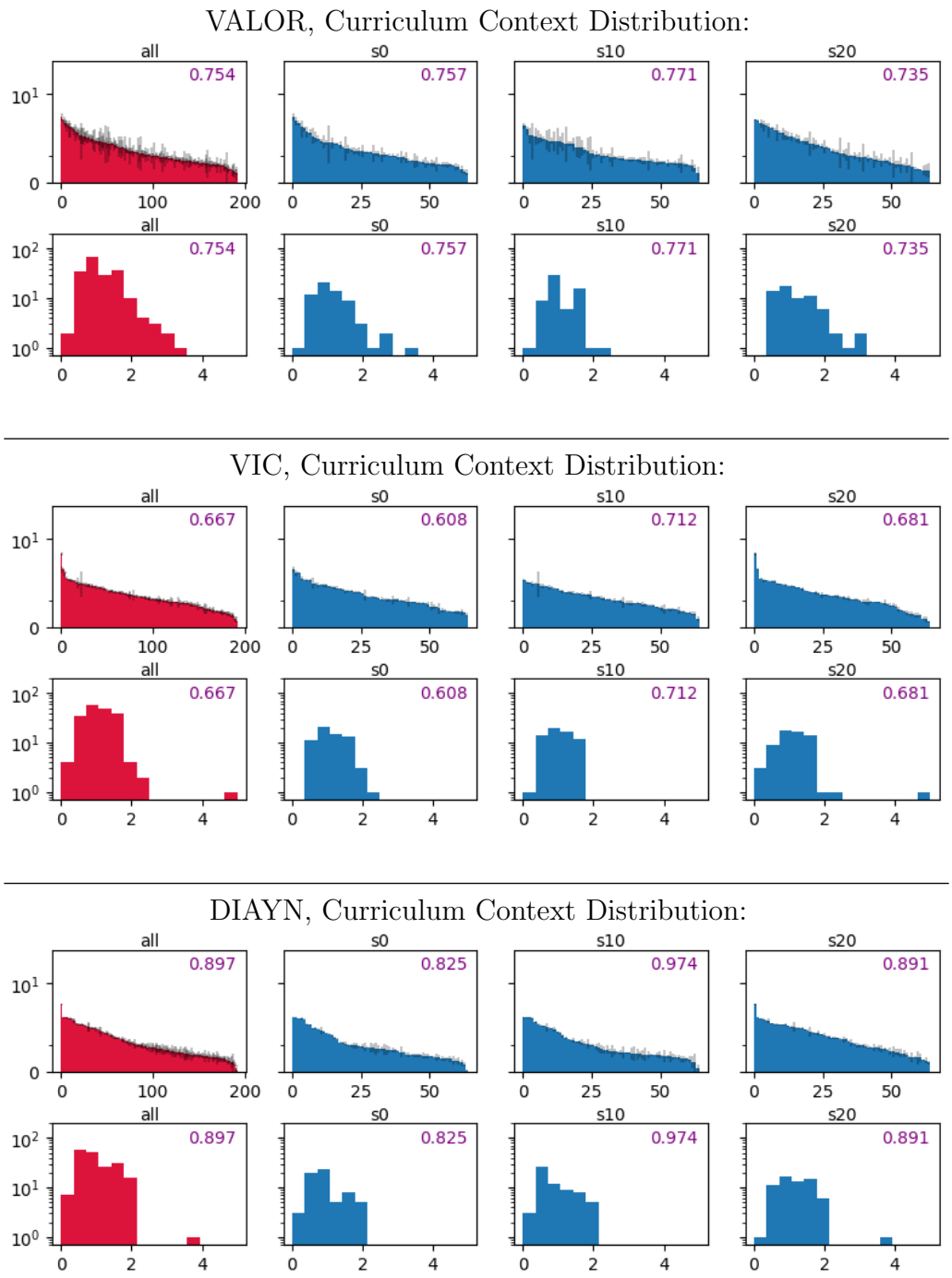


Figure 4.11: Final distance from origin in the Swimmer environment.

4.10.5 Evaluating Learned Behaviors: Ant (Distance)

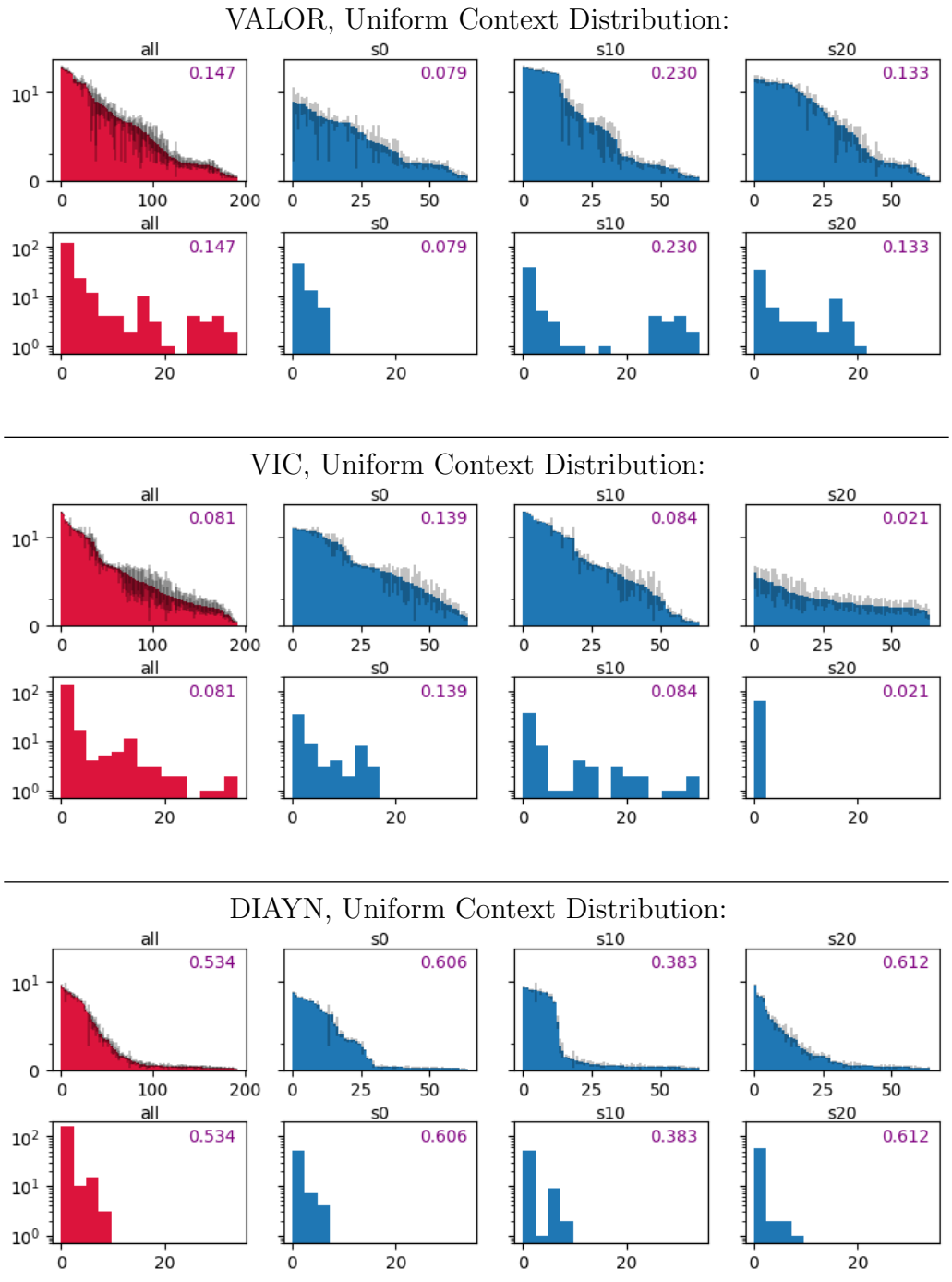


Figure 4.12: Final distance from origin in the Ant environment.

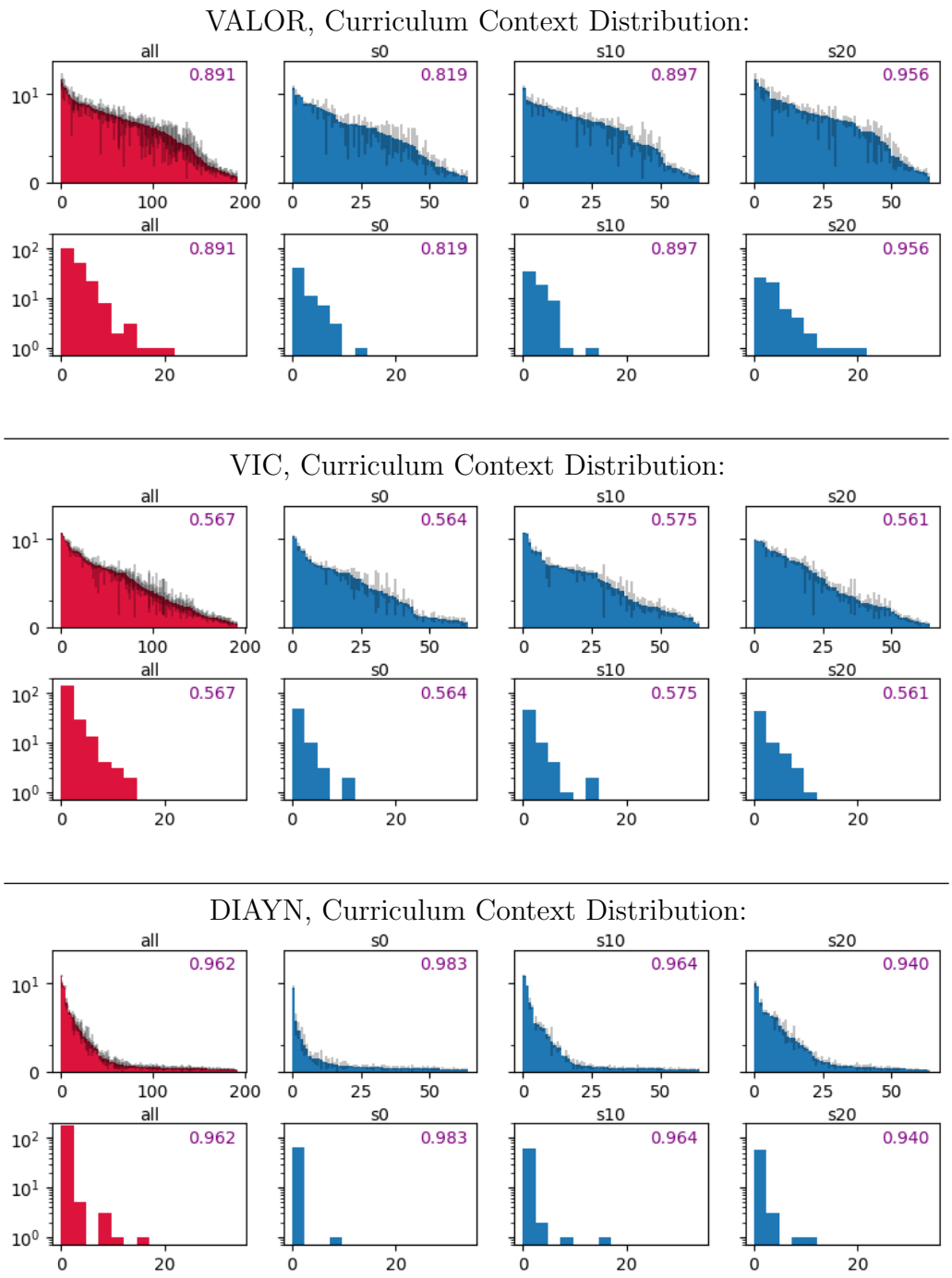


Figure 4.13: Final distance from origin in the Ant environment.

4.10.6 Evaluating Learned Behaviors: Ant (Rotations)

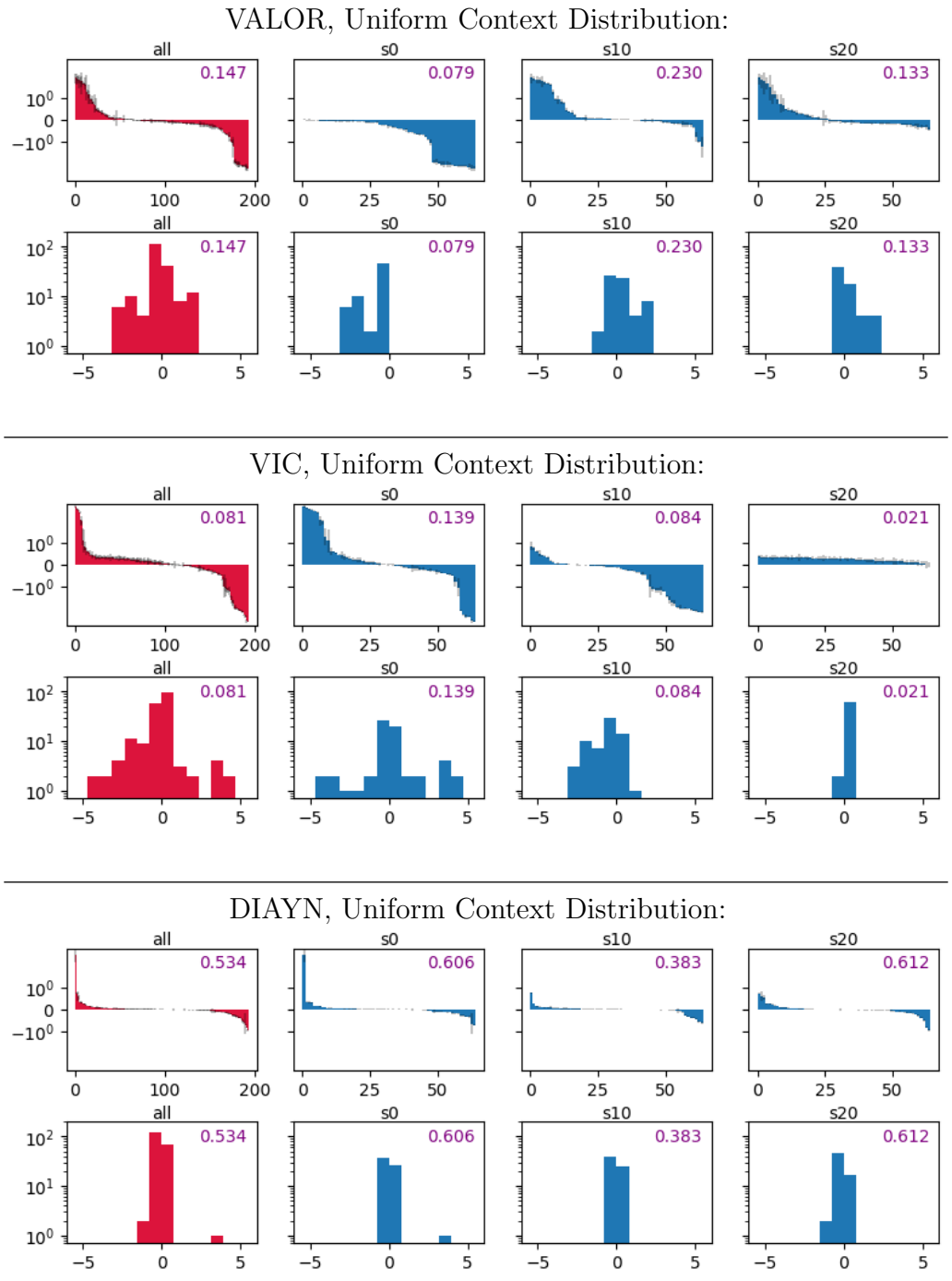


Figure 4.14: Number of z-axis rotations in the Ant environment.

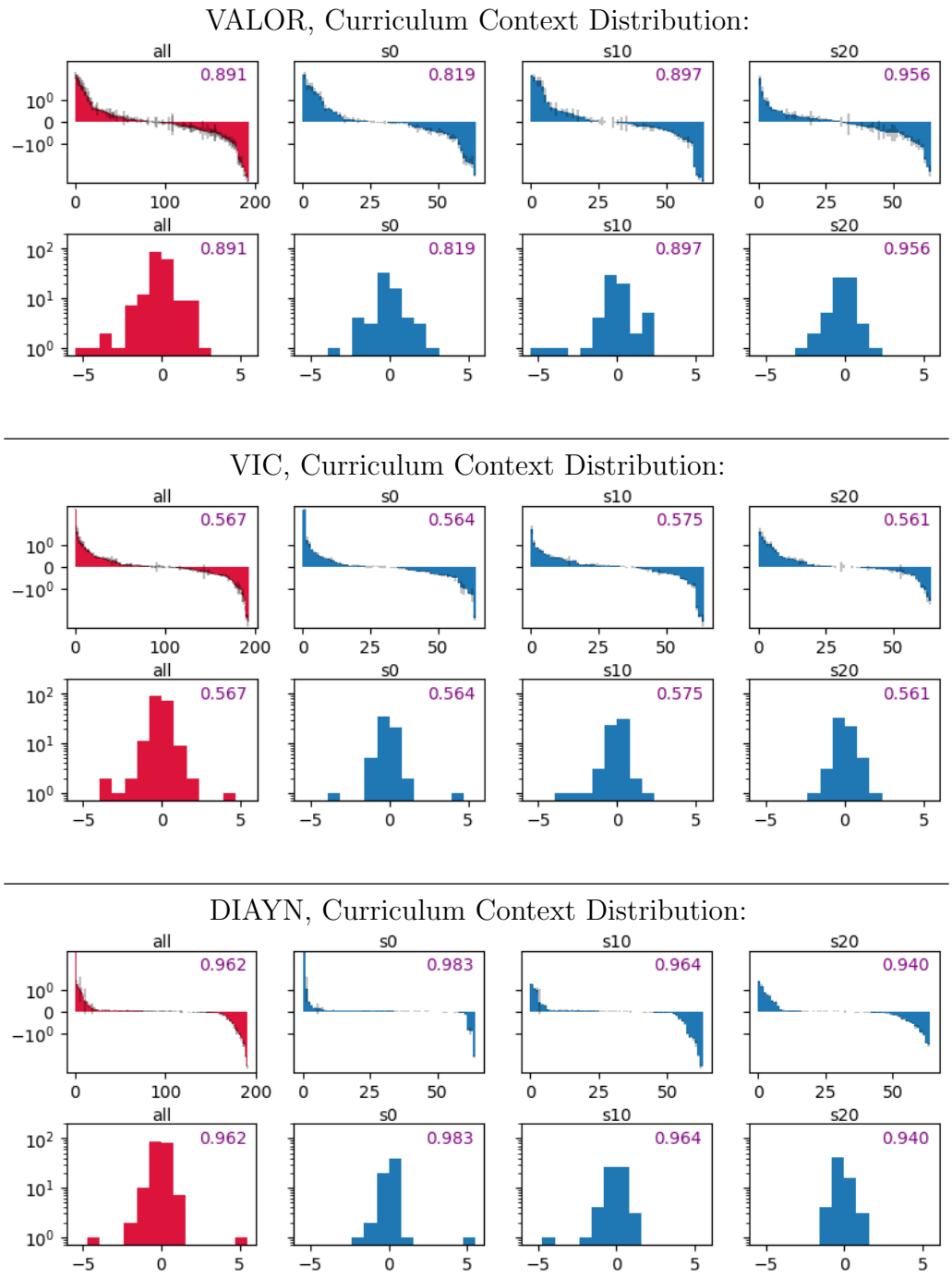
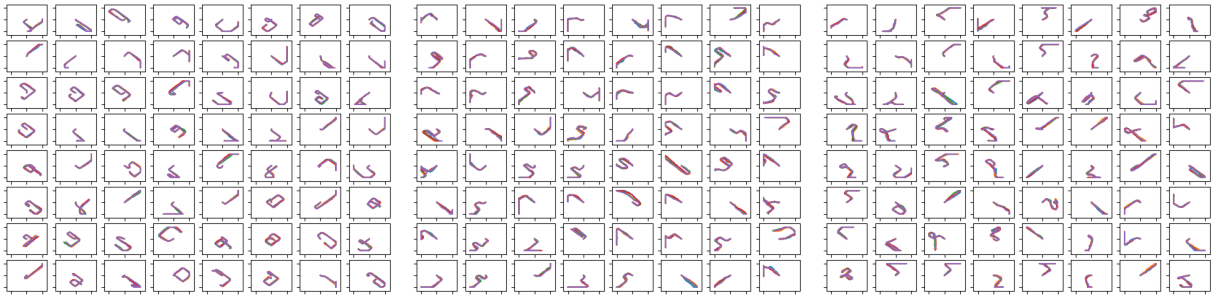


Figure 4.15: Number of z-axis rotations in the Ant environment.

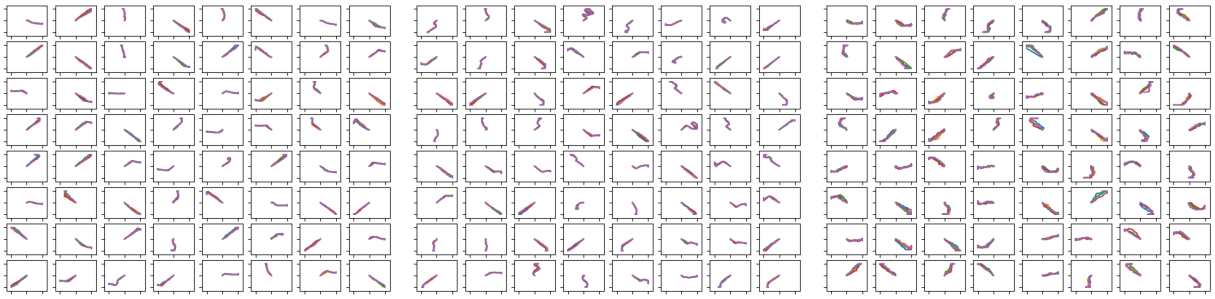
4.10.7 Point Env, Uniform Context Distribution, XY-Traces

VALOR, Uniform Context Distribution::



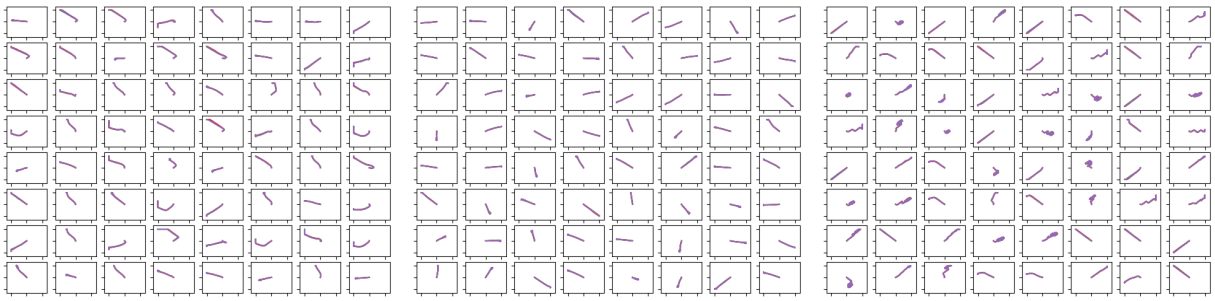
(a) VALOR, Uniform, s0 (b) VALOR, Uniform, s10 (c) VALOR, Uniform, s20

VIC, Uniform Context Distribution:



(d) VIC, Uniform, s0 (e) VIC, Uniform, s10 (f) VIC, Uniform, s20

DIAYN, Uniform Context Distribution:

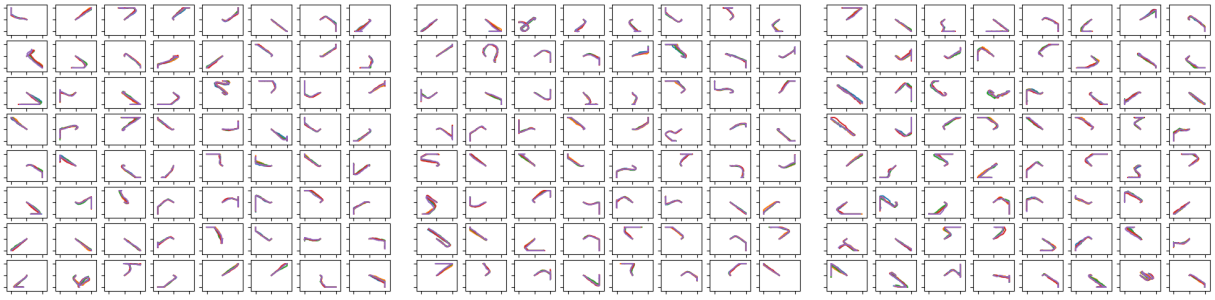


(g) DIAYN, Uniform, s0 (h) DIAYN, Uniform, s10 (i) DIAYN, Uniform, s20

Figure 4.16: Learned behaviors in the Point environment with uniform context distributions. Each sub-plot shows X-Y traces for five trajectories conditioned on the same context (because the learned behaviors are highly repeatable, most traces almost entirely overlap). All traces for an algorithm come from a single policy which was trained with $K = 64$ contexts.

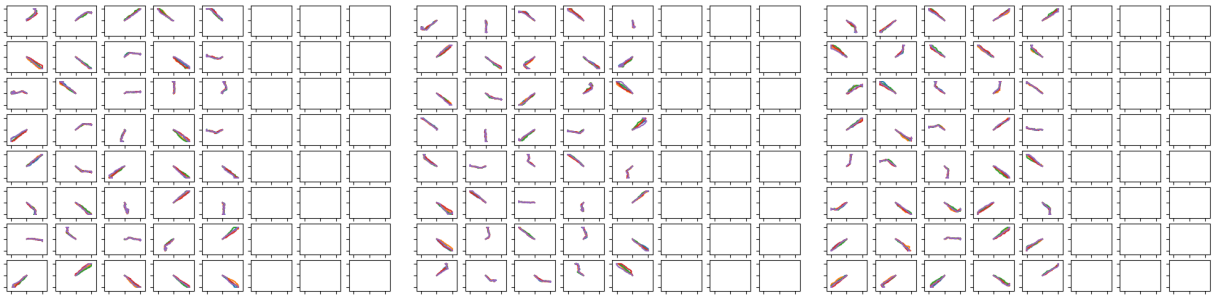
4.10.8 Point Env, Curriculum Context Distribution, XY-Traces

VALOR, Curriculum Context Distribution:



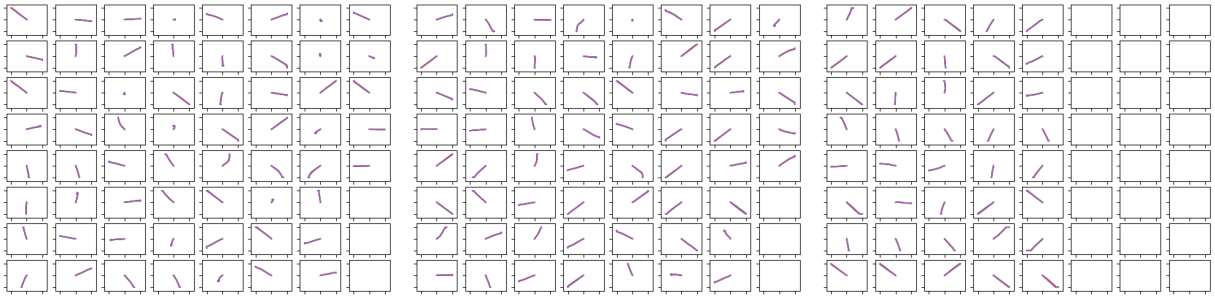
(a) VALOR, Curriculum, s0 (b) VALOR, Curriculum, s10 (c) VALOR, Curriculum, s20

VIC, Curriculum Context Distribution:



(d) VIC, Curriculum, s0 (e) VIC, Curriculum, s10 (f) VIC, Curriculum, s20

DIAYN, Curriculum Context Distribution:

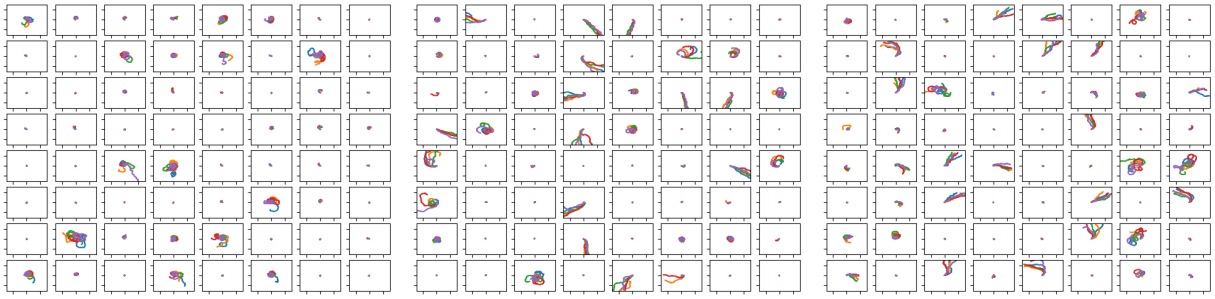


(g) DIAYN, Curriculum, s0 (h) DIAYN, Curriculum, s10 (i) DIAYN, Curriculum, s20

Figure 4.17: Learned behaviors in the Point environment with the curriculum trick. Each sub-plot shows X-Y traces for five trajectories conditioned on the same context (because the learned behaviors are highly repeatable, most traces almost entirely overlap). All traces for an algorithm come from a single policy which was trained with $K_{max} = 64$ contexts. Where a blank sub-plot appears, the agent was never trained on that context (K was less than K_{max} at the end of 5000 iterations of training).

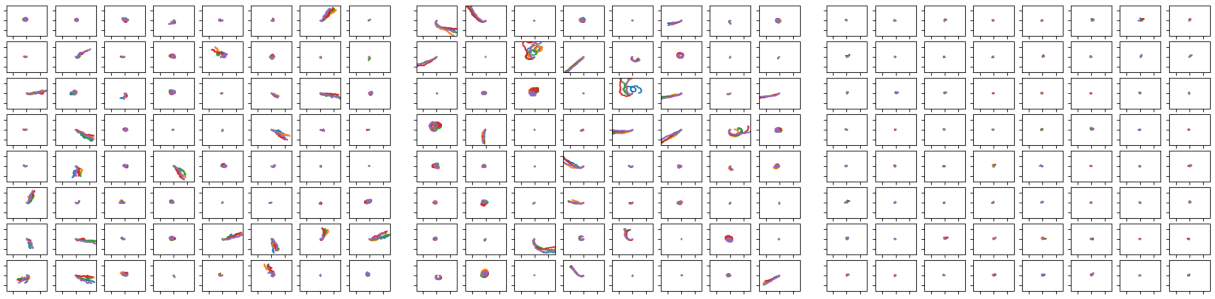
4.10.9 Ant Env, Uniform Context Distribution, XY-Traces

VALOR, Uniform Context Distribution:



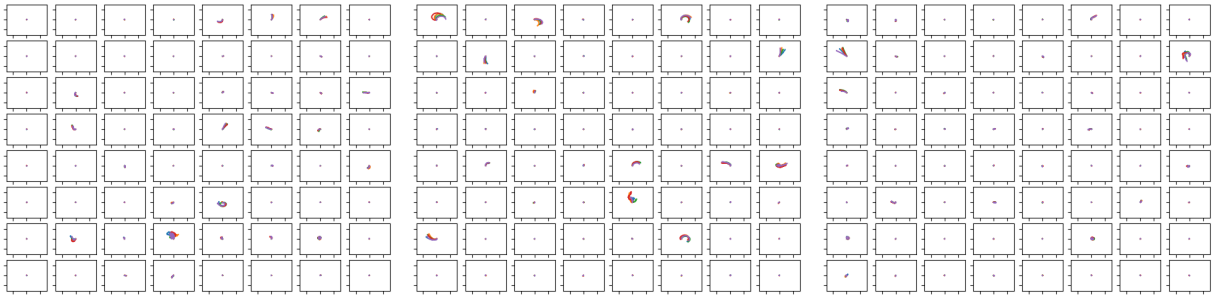
(a) VALOR, Uniform, s0 (b) VALOR, Uniform, s10 (c) VALOR, Uniform, s20

VIC, Uniform Context Distribution:



(d) VIC, Uniform, s0 (e) VIC, Uniform, s10 (f) VIC, Uniform, s20

DIAYN, Uniform Context Distribution:

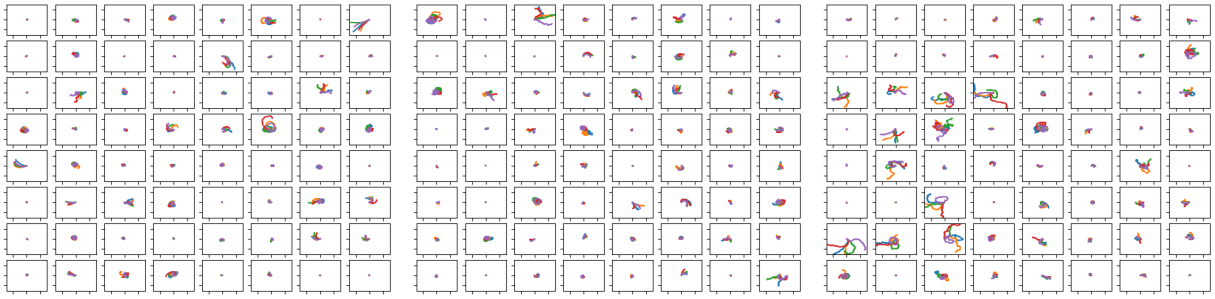


(g) DIAYN, Uniform, s0 (h) DIAYN, Uniform, s10 (i) DIAYN, Uniform, s20

Figure 4.18: Learned behaviors in the Ant environment with uniform context distributions. Each sub-plot shows X-Y traces for five trajectories conditioned on the same context (because the learned behaviors are highly repeatable, most traces almost entirely overlap). All traces for an algorithm come from a single policy which was trained with $K = 64$ contexts.

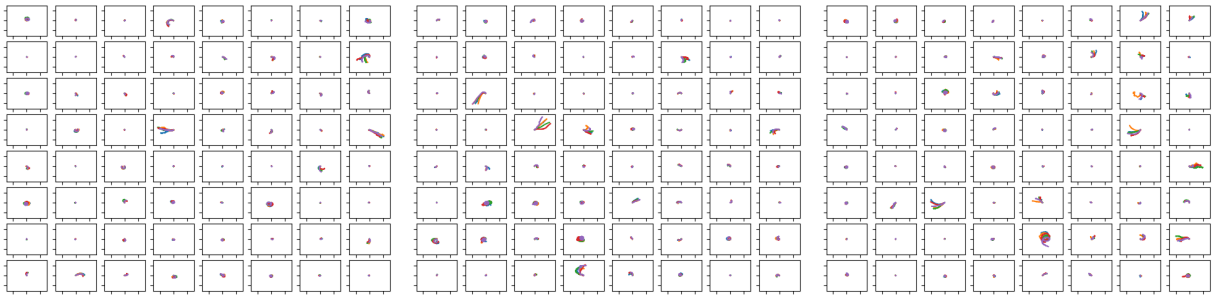
4.10.10 Ant Env, Curriculum Context Distribution, XY-Traces

VALOR, Curriculum Context Distribution:



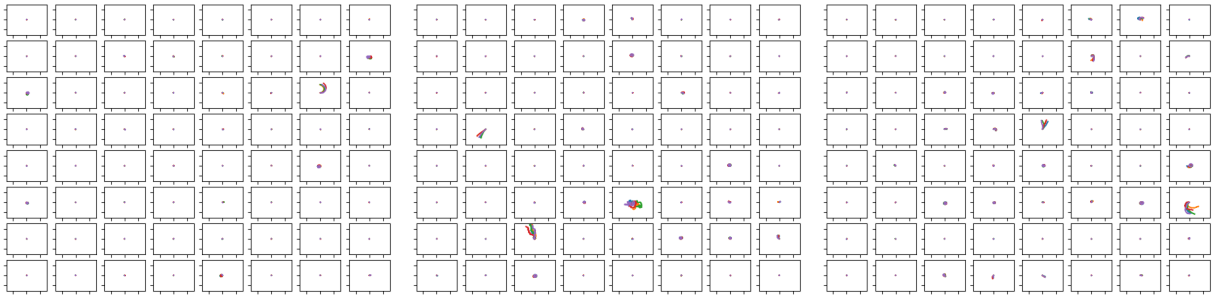
(a) VALOR, Curriculum, s0 (b) VALOR, Curriculum, s10 (c) VALOR, Curriculum, s20

VIC, Curriculum Context Distribution:



(d) VIC, Curriculum, s0 (e) VIC, Curriculum, s10 (f) VIC, Curriculum, s20

DIAYN, Curriculum Context Distribution:



(g) DIAYN, Curriculum, s0 (h) DIAYN, Curriculum, s10 (i) DIAYN, Curriculum, s20

Figure 4.19: Learned behaviors in the Ant environment with the curriculum trick. Each sub-plot shows X-Y traces for five trajectories conditioned on the same context (because the learned behaviors are highly repeatable, most traces almost entirely overlap). All traces for an algorithm come from a single policy which was trained with $K_{max} = 64$ contexts. Where a blank sub-plot appears, the agent was never trained on that context (K was less than K_{max} at the end of 5000 iterations of training).

4.11 Learning Multimodal Policies with Random Rewards

We considered a random reward baseline, where an agent acting under context c would receive a reward

$$R(s, a, c) = v_c^T s, \tag{4.6}$$

where v_c was a random context-specific unit vector, obtained by sampling from $\mathcal{N}(0, I)$ and then normalizing. It seemed plausible that rewards of this form would do a good job of encoding human priors for robot behavior for the simple locomotion tasks in our core comparison. In practice, it turned out to be extremely challenging to train multimodal agents with these rewards; while somewhat easier to train unimodal agents with them, the behaviors that we observed were less interesting than expected. We present results from two sets of experiments:

RR1. a ceteris paribus analogue to our core comparison between variational option discovery algorithms, using all of the same hyperparameters (number of epochs, paths per epoch, number of contexts, the use of embeddings, learning rates, etc.), except with rewards from Eq. 4.6 instead of a learned decoder,

RR2. and a set of experiments where all else is equal except that the number of contexts is $K = 1$ instead of $K = 64$.

RR1 is a direct and fair comparison, while RR2 allows us to gain intuition for the behavior obtained by optimizing these random rewards separately from the challenges of multitask learning.

4.11.1 Results from RR1

The results in Cheetah (Fig. 4.20) look reasonable in composite, but are weak for individual random seeds: in each seed, the results are nearly bimodal, with one mode learning to run forward at some speed, and the other mode learning to run backwards at another speed. In Swimmer (Fig. 4.21), this form of random rewards inspires almost no motion. Results in the Ant environment (Figs. 4.22, 4.23) show extreme variability: no individual behavior was consistent with respect to the score functions we used (the black bars, representing standard deviation, are very large for every behavior).

4.11.2 Results from RR2

We found no significant difference in quality of learned behaviors between the multimodal policies in RR1 and the unimodal policies in RR2, as shown in Fig. 4.24. That is, training with a single random reward function, instead of several at once, did not result in useful or consistent behavior as measured by our score functions.

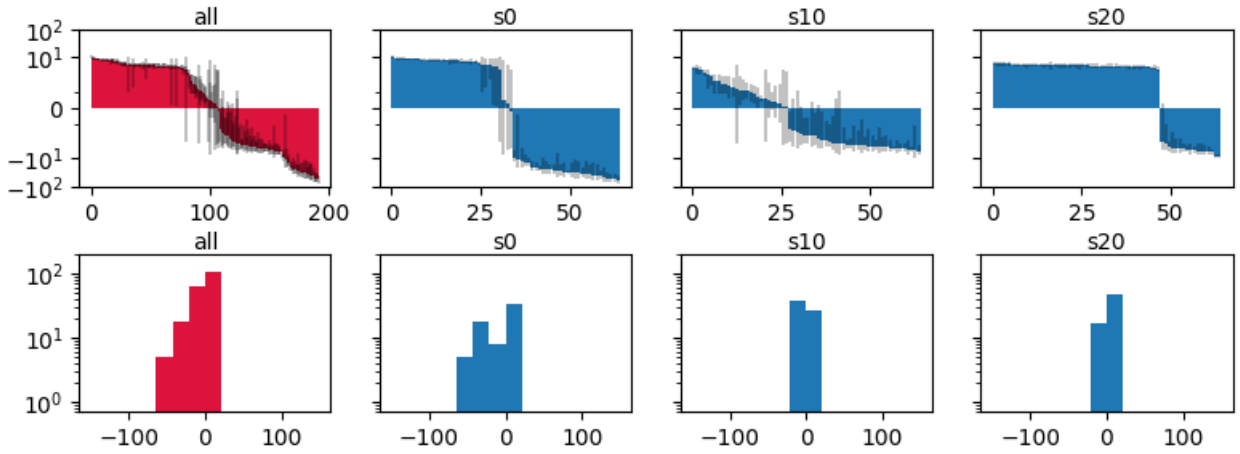


Figure 4.20: Final x-coordinate in the Cheetah environment for random rewards.

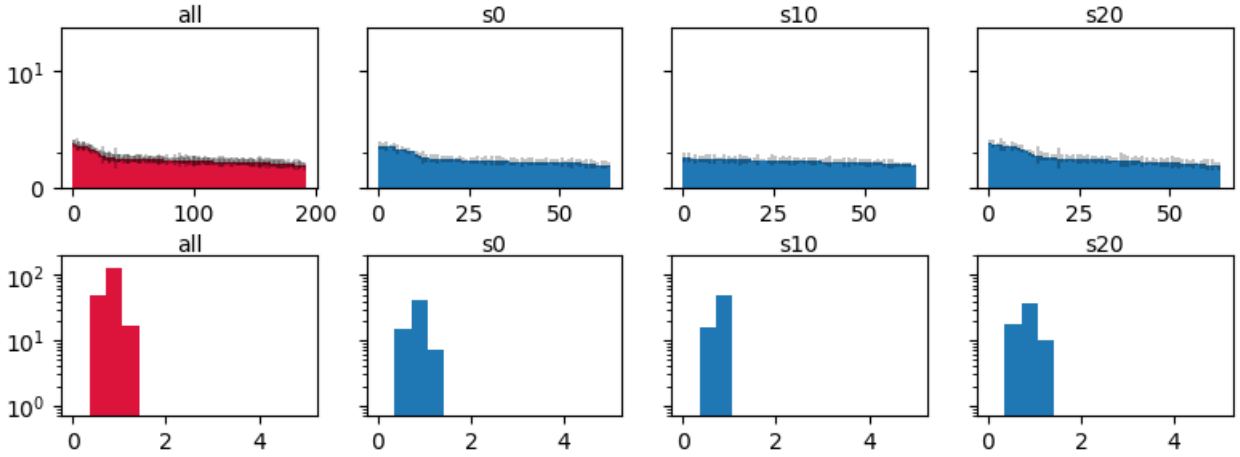


Figure 4.21: Final distance from origin in Swimmer for random rewards.

4.11.3 Discussion

Our conclusion is that random rewards based on Eq. 4.6 do not result in interesting behavior in the environments we considered. However, there may exist a functional form for random rewards which performs better.

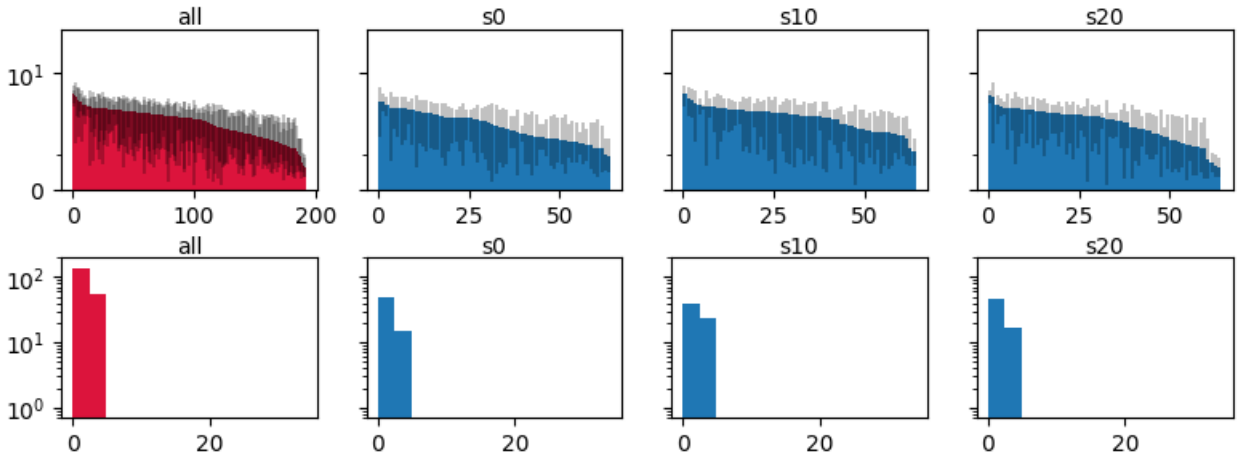


Figure 4.22: Final distance from origin in Ant for random rewards.

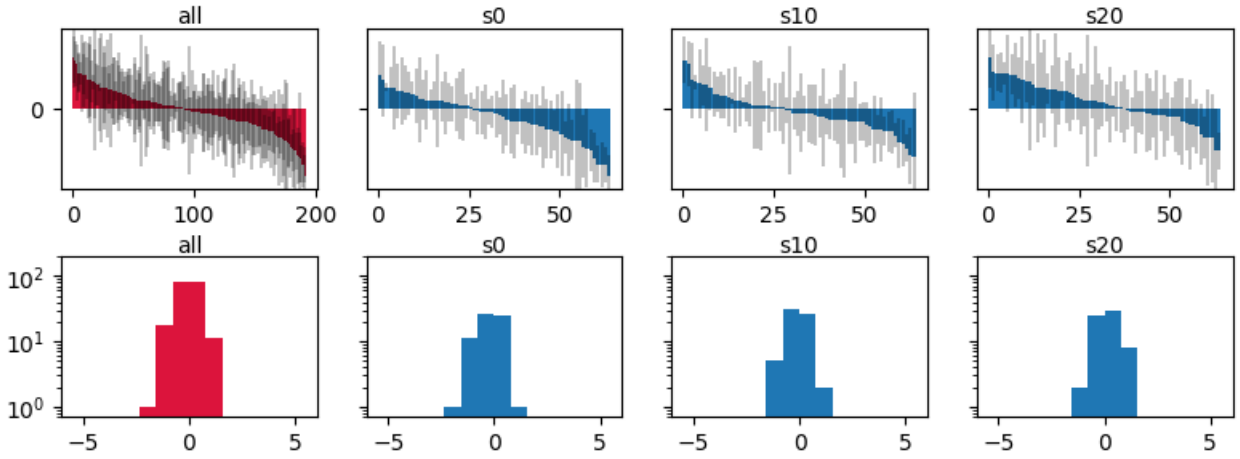


Figure 4.23: Number of z-axis rotations in Ant for random rewards.

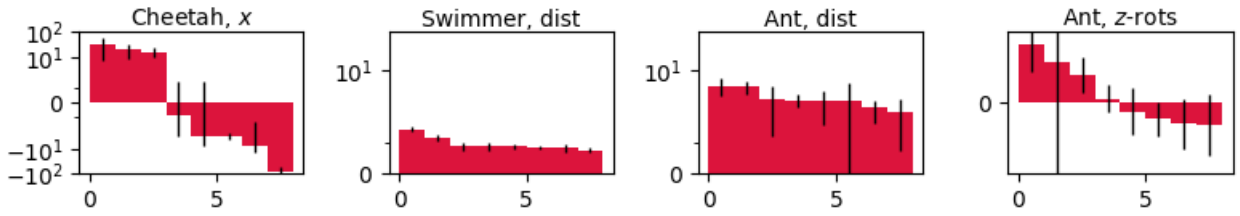


Figure 4.24: Score distributions for RR2.

Part II

Safe Exploration

Chapter 5

Constrained Reinforcement Learning

5.1 Introduction

The fundamental principle of RL is that an agent tries to maximize a reward signal by trial and error. The general-purpose nature of RL makes it an attractive option for a wide range of applications, including self-driving cars [Kendall et al., 2018], surgical robotics [Richter et al., 2019], energy systems management [Gamble and Gao, 2018, Mason and Grijalva, 2019], and other problems where AI would interact with humans or critical infrastructure. Safety concerns in these settings are paramount, but this is at odds with the trial-and-error nature of RL: agents will sometimes try dangerous or harmful behaviors in the course of learning [Hans et al., 2008, Moldovan and Abbeel, 2012, Pecka and Svoboda, 2014, García and Fernández, 2015, Amodei et al., 2016]. When all training occurs in a simulator, this is usually not concerning, but exploration of this kind in the real world could produce unacceptable catastrophes. To illustrate safety concerns in a few domains where RL might plausibly be applied:

- Robots and autonomous vehicles should not cause physical harm to humans.
- AI systems that manage power grids should not damage critical infrastructure.
- Question-answering systems should not provide false or misleading answers for questions about medical emergencies [Bickmore et al., 2018].
- Recommender systems should not expose users to psychologically harmful or extremist content [Vendrov and Nixon, 2019].

A central question for the field of RL is therefore:

> How do we formulate safety specifications to incorporate them into RL, and how do we ensure that these specifications are robustly satisfied throughout exploration?

The focus of this part of the thesis—Chapters 5 through 8—is to make progress on this

question. In this chapter, we propose to standardize constrained RL [Altman, 1999] as the main formalism for incorporating safety specifications into RL algorithms to achieve safe exploration. Our position is that 1) safety specifications should be separate from task performance specifications, and 2) constraints are a natural way to encode safety specifications. We support this argument by reference to standards for safety requirements that typically arise in engineering design and risk management, and we identify the limitations of alternative approaches. Importantly, constrained RL is scalable to the regime of high-dimensional function approximation—the modern deep RL setting. In the chapters that follow, we will develop algorithms for constrained RL and benchmark environments to measure progress.

5.2 Constrained Reinforcement Learning

We take a broad view of constrained RL as the general problem of training an RL agent with constraints, usually with the intention of satisfying constraints throughout exploration in training and at test time. In this sub-section, we'll describe the quantitative formulation for the constrained RL problem. Let Π_C denote a feasible set of constraint-satisfying policies, and for the moment, put aside the question of how it is constructed. An optimal policy in constrained RL is given by:

$$\pi^* = \arg\max_{\pi \in \Pi_C} J_r(\pi), \tag{5.1}$$

where $J_r(\pi)$ is a reward-based objective function. As in standard RL, the objective is usually either the infinite-horizon discounted return, the finite-horizon undiscounted return, or the infinite-horizon average reward.

5.2.1 Constrained Markov Decision Processes

The framework of constrained Markov Decision Processes (CMDPs) [Altman, 1999] is the standard way to describe feasible sets in constrained RL. CMDPs extend MDPs by equipping them with a set of cost functions, $C_1, ..., C_k$ (with each one a function $C_i : S \times A \times S \to \mathbb{R}$ mapping transition tuples to costs, like the usual reward), and limits $d_1, ..., d_k$. Let $J_{C_i}(\pi)$ denote the expected cumulative cost measure for policy π with respect to cost function C_i (that is, define it the same way you would define $J(\pi)$, but substituting $R \to C_i$). The feasible set of stationary policies for a CMDP is then

$$\Pi_C = \{\pi \in \Pi \ : \ J_{C_i}(\pi) \leq d_i, \quad i = 1, ..., k\}. \tag{5.2}$$

We refer to J_{C_i} as a *constraint return*, or C_i-return for short. Additionally we define on-policy value functions, action-value functions, and advantage functions for the auxiliary costs in analogy to V^π, Q^π, and A^π, with C_i replacing R: respectively, we denote these by $V^\pi_{C_i}$, $Q^\pi_{C_i}$, and $A^\pi_{C_i}$.

The CMDP framework can be extended to use different kinds of cost-based constraints; for instance Chow et al. [2015] consider chance constraints (eg, $P(\sum_t c_t \geq C) \leq d$) and

constraints on the conditional value at risk (the expected sum of costs over the α-fraction of worst-case outcomes), and Dalal et al. [2018] consider separate constraints for each state in the CMDP (eg, $\forall s,\ \mathbb{E}_{a\sim\pi}\left[c(s,a)\right] \leq d$). The range of constraints on agent behavior expressible through appropriately-designed cost functions is quite vast—a claim that can be seen as a corollary to the *reward hypothesis*, which states that "all of what we mean by goals and purposes" can be described with reward functions [Sutton and Barto, 2018].

5.2.2 Evaluating Constraint-Satisfying Exploration

While the CMDP framework characterizes feasible sets, optimal policies, and equivalent optimization problems for Eq 5.1, it does not, by itself, describe ways to evaluate or attain constraint-satisfying exploration. A substantial body of recent work has explored this problem [Achiam et al., 2017b, Saunders et al., 2017, Pham et al., 2018, Dalal et al., 2018, Chow et al., 2018, 2019], but there is not yet a universally agreed-upon way to evaluate and compare different methods. Achiam et al. [2017b] and Chow et al. [2018, 2019] qualitatively compared the learning curves for expected cost between methods, preferring the methods that appeared to have fewer or smaller constraint-violating spikes. Saunders et al. [2017], Pham et al. [2018], and Dalal et al. [2018] counted and compared the total number of times an agent entered into an undesired state throughout training for different methods, essentially measuring constraint-satisfaction *regret*. We endorse quantitative approaches like this and recommend that the degree of constraint-satisfaction throughout exploration should be evaluated by measures of regret, with the regret function accounting for all of the agent's actual experience (as opposed to, say, only experiences from separate test behavior). Later, when describing our evaluation protocol for benchmarking constrained RL algorithms in Safety Gym, we will make the case that cost rate (the average cost over the entirety of training) is a suitable regret measure.

5.3 Constrained RL and Safe Exploration

Constraints are a natural and universally-relevant way to formulate safety requirements. The work of making a system safe refers to the reduction or avoidance of harm, broadly defined, which in a practical sense means avoiding hazards [Rausand, 2014]—that is, constraining the state and behavior of the system to stay away from the circumstances that lead to harm. This perspective underlies standards and practices in the field of systems safety; see for example Rice [2002] and NASA [2017].

Contrast this with standard reinforcement learning, which just maximizes a reward function. In order to design hazard-avoiding behavior into an agent through a scalar reward function, a designer would have to carefully select a trade-off between a reward for task-solving behavior and a penalty for proximity to hazards. There are two problems with this: 1) There is no invertible map between "desired safety specification" and "correct trade-off parameter" that can be checked before running an RL algorithm. If the designer selects a penalty that is too small, the agent will learn unsafe behavior, and if the penalty is too severe, the agent may

fail to learn anything. 2) A fixed trade-off, even one that results in a hazard-avoiding policy at optimum, does not account for a requirement to satisfy safety requirements throughout training. Both of these problems have been observed in practice, for example by Achiam et al. [2017b], Pham et al. [2018], and Dalal et al. [2018]. The choice to formulate safety requirements as constraints, and to attain constraint-satisfying exploration, resolves both. Aside from the conceptual justification, encouraging results demonstrate that constrained RL algorithms are performant in the high-dimensional control regime (eg Saunders et al. [2017], Achiam et al. [2017b], Dalal et al. [2018], Wang et al. [2018], Bohez et al. [2019], and Chow et al. [2019]) and are therefore viable for making progress on the general safe exploration problem.

5.3.1 Constrained RL and the Alignment Problem

A central concern in safety for advanced RL systems, especially artificial general intelligence (AGI), relates to *agent alignment*: the problem of ensuring that an agent behaves in accordance with the user's intentions. (Here, we cite Leike et al. [2018] for the specific term and phrasing, but this problem has been considered in various forms for decades.) In RL, this manifests primarily as an issue in reward specification, where seemingly-correct but misspecified reward functions can result in incorrect and unsafe agent behavior [Clark and Amodei, 2016]. It is not considered obvious whether the framework of constrained RL helps solve this issue, since constrained RL still requires the specification of not only reward functions but also typically cost functions for the constraints. The critique, then, is that errors in designing constraint functions could result in unsafe agents, and so constrained RL is simply moving the alignment problem around instead of solving it.

A mainstream vector in AI safety research tries to address the alignment problem by using data from humans to derive suitable objective or reward functions for training agents. This family of approaches includes cooperative inverse reinforcement learning [Hadfield-Menell et al., 2016], learning from binary or ranked preferences [Christiano et al., 2017], iterated amplification and distillation [Christiano et al., 2018], AI safety via debate [Irving et al., 2018], and recursive reward modeling [Leike et al., 2018]. Other approaches attempt to regularize the impact of agents, based on the prior that agents should prefer task solutions that have minimum side effects [Krakovna et al., 2018] or minimally contradict preferences implicit in the initial state of the environment [Shah et al., 2019]. By and large, this vector of safety work aims to eliminate the need for explicitly designing safety specifications, on the grounds that hand-crafted specifications will fail in various ways (eg by omitting to penalize certain unsafe behaviors, or by inadvertantly incentivizing harmful behaviors).

We contend that the use of constraints is compatible with, and complementary to, these data-driven approaches to solving the alignment problem. Straightforwardly, techniques for learning reward functions from human data can also be used to learn cost functions for constraints. Furthermore, we conjecture that the use of constraints may indeed improve 1) the ease with which safety specifications are learned and transferred between tasks, and 2) the robustness with which agents attain those safety requirements.

In support of these conjectures, we note that when learning algorithms exploit priors specific to a problem's structure, they generally tend to be more sample-efficient, by virtue of searching for solutions in a narrower and more useful set. We regard the partitioning of agent behavior specification into "do"s and "don't"s (a reward function and constraint functions respectively) to be one such useful prior for safety. Furthermore, whereas a learned reward function for one task may fail to transfer to another (for instance, a reward function for assembling widgets may describe very little about how to assemble doodads), a learned constraint function describing unacceptable behavior seems more likely to transfer successfully. For instance, a cost function for "do not physically strike a human" is relevant regardless of what an agent is tasked with building.

5.3.2 Remarks on Alternate Approaches

We contend that certain other approaches to safe reinforcement learning without constraints are either insufficient or impractical. Approaches to safety that focus solely on measures of return for a single scalar reward function (where such scalar reward function is kept fixed over the course of training)—as either monotonic improvement in expected return, a constraint on the variance of return, return above a minimum value, or a risk measure on return—inappropriately conflate task performance specifications and safety specifications, and are therefore inadequate for the reasons described previously. Another common approach that we consider flawed focuses on ergodicity: the agent is considered safe if it never enters into a state it can't return from, that is, if every mistake is reversible [Moldovan and Abbeel, 2012, Eysenbach et al., 2018b]. While this can be a good rule of thumb for safety in some practical cases that arise in robotics, it is irrelevant in many more, as discussed by Pecka and Svoboda [2014]: it rules out irreversible good actions as well as bad.

5.3.3 Relation to Multi-Objective RL

We note that constrained RL is closely-related to *multi-objective RL*, and that our arguments for separating concerns between task objective and safety requirements are also applicable to multi-objective RL. We choose to focus on constrained RL because of the natural "shape" of functions for safety requirements: there is generally a saturation point where the safety requirement is satisfied, and further decreasing the value of the function no longer makes the system meaningfully or usefully safer. In the constrained formulation, this corresponds to the constraint threshold; this has no standard equivalent in the multi-objective formulation.

Chapter 6

Constrained Policy Optimization

We propose Constrained Policy Optimization (CPO), the first general-purpose policy search algorithm for constrained reinforcement learning with guarantees for near-constraint satisfaction at each iteration. Our method allows us to train neural network policies for high-dimensional control while making guarantees about policy behavior all throughout training. Our guarantees are based on a new theoretical result, which is of independent interest: we prove a bound relating the expected returns of two policies to an average divergence between them. We demonstrate the effectiveness of our approach on simulated robot locomotion tasks where the agent must satisfy constraints motivated by safety.

6.1 Introduction

Currently, policy search algorithms enjoy state-of-the-art performance on high-dimensional control tasks [Mnih et al., 2016, Duan et al., 2016, Schulman et al., 2017]. Heuristic algorithms for policy search in CMDPs have been proposed [Uchibe and Doya, 2007], and approaches based on primal-dual methods can be shown to converge to constraint-satisfying policies [Chow et al., 2015], but there is no prior approach for policy search in continuous CMDPs that guarantees every policy during learning will satisfy constraints. In this work, we propose the first such algorithm, allowing applications to constrained deep RL. Driving our approach is a new theoretical result that bounds the difference between the rewards or costs of two different policies. This result, which is of independent interest, tightens known bounds for policy search using trust regions [Kakade and Langford, 2002, Pirotta et al., 2013, Schulman et al., 2015], and provides a tighter connection between the theory and practice of policy search for deep RL. Here, we use this result to derive a policy improvement step that guarantees both an increase in reward and satisfaction of constraints on other costs. This step forms the basis for our algorithm, *Constrained Policy Optimization* (CPO), which computes an approximation to the theoretically-justified update. In our experiments, we show that CPO can train neural network policies with thousands of parameters on high-dimensional simulated robot locomotion tasks to maximize rewards while successfully enforcing constraints.

6.2 Related Work

Safety has long been a topic of interest in RL research, and a comprehensive overview of safety in RL was given by García and Fernández [2015].

Safe policy search methods have been proposed in prior work. Uchibe and Doya [2007] gave a policy gradient algorithm that uses gradient projection to enforce active constraints, but this approach suffers from an inability to prevent a policy from becoming unsafe in the first place. Bou Ammar et al. [2015] propose a theoretically-motivated policy gradient method for lifelong learning with safety constraints, but their method involves an expensive inner loop optimization of a semi-definite program, making it unsuited for the deep RL setting. Their method also assumes that safety constraints are linear in policy parameters, which is limiting. Chow et al. [2015] propose a primal-dual subgradient method for risk-constrained reinforcement learning which takes policy gradient steps on an objective that trades off return with risk, while simultaneously learning the trade-off coefficients (dual variables).

Some approaches specifically focus on application to the deep RL setting. Held et al. [2017] study the problem for robotic manipulation, but the assumptions they make restrict the applicability of their methods. Lipton et al. [2017] use an 'intrinsic fear' heuristic, as opposed to constraints, to motivate agents to avoid rare but catastrophic events. Shalev-Shwartz et al. [2016] avoid the problem of enforcing constraints on parametrized policies by decomposing 'desires' from trajectory planning; the neural network policy learns desires for behavior, while the trajectory planning algorithm (which is not learned) selects final behavior and enforces safety constraints.

In contrast to prior work, our method is the first policy search algorithm for CMDPs that both 1) guarantees constraint satisfaction throughout training, and 2) works for arbitrary policy classes (including neural networks).

6.3 Preliminaries

Recall that a stationary policy $\pi : S \to \mathcal{P}(A)$ is a map from states to probability distributions over actions, with $\pi(a|s)$ denoting the probability of selecting action a in state s. We denote the set of all stationary policies by Π. In this chapter we take the policy performance measure $J(\pi)$ to be the infinite horizon discounted total return,

$$J(\pi) \doteq \mathop{\mathrm{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t, a_t, s_{t+1})\right].$$

A quantity central to the mathematical analysis in this chapter is the discounted future state distribution, d^π, defined by $d^\pi(s) = (1-\gamma)\sum_{t=0}^{\infty}\gamma^t P(s_t = s|\pi)$. It allows us to compactly express the difference in performance between two policies π', π as

$$J(\pi') - J(\pi) = \frac{1}{1-\gamma}\mathop{\mathrm{E}}_{\substack{s\sim d^{\pi'}\\ a\sim\pi'}}\left[A^\pi(s,a)\right], \tag{6.1}$$

where by $a \sim \pi'$, we mean $a \sim \pi'(\cdot|s)$, with explicit notation dropped to reduce clutter. For proof of (6.1), see Kakade and Langford [2002] or Section 6.9.

6.4 Constrained Policy Optimization

For large or continuous MDPs, solving for the exact optimal policy is intractable due to the curse of dimensionality [Sutton and Barto, 1998]. Policy search algorithms approach this problem by searching for the optimal policy within a set $\Pi_\theta \subseteq \Pi$ of parametrized policies with parameters θ (for example, neural networks of a fixed architecture). In local policy search [Peters and Schaal, 2008], the policy is iteratively updated by maximizing $J(\pi)$ over a local neighborhood of the most recent iterate π_k:

$$\begin{aligned} \pi_{k+1} = \arg\max_{\pi \in \Pi_\theta} \; & J(\pi) \\ \text{s.t.} \; & D(\pi, \pi_k) \leq \delta, \end{aligned} \tag{6.2}$$

where D is some distance measure, and $\delta > 0$ is a step size. When the objective is estimated by linearizing around π_k as $J(\pi_k) + g^T(\theta - \theta_k)$, g is the policy gradient, and the standard policy gradient update is obtained by choosing $D(\pi, \pi_k) = \|\theta - \theta_k\|_2$ [Schulman et al., 2015].

In local policy search for CMDPs, we additionally require policy iterates to be feasible for the CMDP, so instead of optimizing over Π_θ, we optimize over $\Pi_\theta \cap \Pi_C$:

$$\begin{aligned} \pi_{k+1} = \arg\max_{\pi \in \Pi_\theta} \; & J(\pi) \\ \text{s.t.} \; & J_{C_i}(\pi) \leq d_i \quad i = 1, ..., m \\ & D(\pi, \pi_k) \leq \delta. \end{aligned} \tag{6.3}$$

This update is difficult to implement in practice because it requires evaluation of the constraint functions to determine whether a proposed point π is feasible. When using sampling to compute policy updates, as is typically done in high-dimensional control [Duan et al., 2016], this requires off-policy evaluation, which is known to be challenging [Jiang and Li, 2015]. In this work, we take a different approach, motivated by recent methods for trust region optimization [Schulman et al., 2015].

We develop a principled approximation to (6.3) with a particular choice of D, where we replace the objective and constraints with *surrogate* functions. The surrogates we choose are easy to estimate from samples collected on π_k, and are good local approximations for the objective and constraints. Our theoretical analysis shows that for our choices of surrogates, we can bound our update's worst-case performance and worst-case constraint violation with values that depend on a hyperparameter of the algorithm.

To prove the performance guarantees associated with our surrogates, we first prove new bounds on the difference in returns (or constraint returns) between two arbitrary stochastic policies in terms of an average divergence between them. We then show how our bounds permit a new analysis of trust region methods in general: specifically, we prove a worst-case

performance degradation at each update. We conclude by motivating, presenting, and proving gurantees on our algorithm, Constrained Policy Optimization (CPO), a trust region method for CMDPs.

6.4.1 Policy Performance Bounds

In this section, we present the theoretical foundation for our approach—a new bound on the difference in returns between two arbitrary policies. This result, which is of independent interest, extends the works of Kakade and Langford [2002], Pirotta et al. [2013], and Schulman et al. [2015], providing tighter bounds. As we show later, it also relates the theoretical bounds for trust region policy improvement with the actual trust region algorithms that have been demonstrated to be successful in practice [Duan et al., 2016]. In the context of constrained policy search, we later use our results to propose policy updates that both improve the expected return and satisfy constraints.

The following theorem connects the difference in returns (or constraint returns) between two arbitrary policies to an average divergence between them.

Theorem 1. *For any function $f : S \to \mathbb{R}$ and any policies π' and π, define $\delta_f(s, a, s') \doteq R(s, a, s') + \gamma f(s') - f(s)$,*

$$\epsilon_f^{\pi'} \doteq \max_s \left| \mathrm{E}_{a\sim\pi', s'\sim P}[\delta_f(s, a, s')] \right|,$$

$$L_{\pi,f}(\pi') \doteq \mathop{\mathrm{E}}_{\substack{s\sim d^\pi \\ a\sim\pi \\ s'\sim P}} \left[\left(\frac{\pi'(a|s)}{\pi(a|s)} - 1 \right) \delta_f(s, a, s') \right], \text{ and}$$

$$D^{\pm}_{\pi,f}(\pi') \doteq \frac{L_{\pi,f}(\pi')}{1-\gamma} \pm \frac{2\gamma\epsilon_f^{\pi'}}{(1-\gamma)^2} \mathop{\mathrm{E}}_{s\sim d^\pi} \left[D_{TV}(\pi'||\pi)[s] \right],$$

where $D_{TV}(\pi'||\pi)[s] = (1/2)\sum_a |\pi'(a|s) - \pi(a|s)|$ is the total variational divergence between action distributions at s. The following bounds hold:

$$D^{+}_{\pi,f}(\pi') \geq J(\pi') - J(\pi) \geq D^{-}_{\pi,f}(\pi'). \tag{6.4}$$

Furthermore, the bounds are tight (when $\pi' = \pi$, all three expressions are identically zero).

Before proceeding, we connect this result to prior work. By bounding the expectation $\mathrm{E}_{s\sim d^\pi}\left[D_{TV}(\pi'||\pi)[s]\right]$ with $\max_s D_{TV}(\pi'||\pi)[s]$, picking $f = V^\pi$, and bounding $\epsilon_{V^\pi}^{\pi'}$ to get a second factor of $\max_s D_{TV}(\pi'||\pi)[s]$, we recover (up to assumption-dependent factors) the bounds given by Pirotta et al. [2013] as Corollary 3.6, and by Schulman et al. [2015] as Theorem 1a.

The choice of $f = V^\pi$ allows a useful form of the lower bound, so we give it as a corollary.

Corollary 1. *For any policies π', π, with $\epsilon^{\pi'} \doteq \max_s |\mathrm{E}_{a\sim\pi'}[A^\pi(s, a)]|$, the following bound holds:*

$$J(\pi') - J(\pi) \geq \frac{1}{1-\gamma} \mathop{\mathrm{E}}_{\substack{s\sim d^\pi \\ a\sim\pi'}} \left[A^\pi(s, a) - \frac{2\gamma\epsilon^{\pi'}}{1-\gamma} D_{TV}(\pi'||\pi)[s] \right]. \tag{6.5}$$

The bound (6.5) should be compared with equation (6.1). The term $(1-\gamma)^{-1}\mathrm{E}_{s\sim d^\pi, a\sim\pi'}[A^\pi(s,a)]$ in (6.5) is an approximation to $J(\pi') - J(\pi)$, using the state distribution d^π instead of $d^{\pi'}$, which is known to equal $J(\pi') - J(\pi)$ to first order in the parameters of π' on a neighborhood around π [Kakade and Langford, 2002]. The bound can therefore be viewed as describing the worst-case approximation error, and it justifies using the approximation as a *surrogate* for $J(\pi') - J(\pi)$.

Equivalent expressions for the auxiliary costs, based on the upper bound, also follow immediately; we will later use them to make guarantees for the safety of CPO.

Corollary 2. *For any policies π', π, and any cost function C_i, with $\epsilon_{C_i}^{\pi'} \doteq \max_s |\mathrm{E}_{a\sim\pi'}[A_{C_i}^\pi(s,a)]|$, the following bound holds:*

$$J_{C_i}(\pi') - J_{C_i}(\pi) \leq \frac{1}{1-\gamma} \mathop{\mathrm{E}}_{\substack{s\sim d^\pi \\ a\sim\pi'}} \left[A_{C_i}^\pi(s,a) + \frac{2\gamma\epsilon_{C_i}^{\pi'}}{1-\gamma} D_{TV}(\pi'||\pi)[s] \right]. \tag{6.6}$$

The bounds we have given so far are in terms of the TV-divergence between policies, but trust region methods constrain the KL-divergence between policies, so bounds that connect performance to the KL-divergence are desirable. We make the connection through Pinsker's inequality [Csiszar and Körner, 1981]: for arbitrary distributions p, q, the TV-divergence and KL-divergence are related by $D_{TV}(p||q) \leq \sqrt{D_{KL}(p||q)/2}$. Combining this with Jensen's inequality, we obtain

$$\begin{aligned} \mathop{\mathrm{E}}_{s\sim d^\pi}[D_{TV}(\pi'||\pi)[s]] &\leq \mathop{\mathrm{E}}_{s\sim d^\pi}\left[\sqrt{\frac{1}{2} D_{KL}(\pi'||\pi)[s]}\right] \\ &\leq \sqrt{\frac{1}{2} \mathop{\mathrm{E}}_{s\sim d^\pi}[D_{KL}(\pi'||\pi)[s]]} \end{aligned} \tag{6.7}$$

From (6.7) we immediately obtain the following.

Corollary 3. *In bounds (6.4), (6.5), and (6.6), make the substitution*

$$\mathop{\mathrm{E}}_{s\sim d^\pi}[D_{TV}(\pi'||\pi)[s]] \to \sqrt{\frac{1}{2} \mathop{\mathrm{E}}_{s\sim d^\pi}[D_{KL}(\pi'||\pi)[s]]}.$$

The resulting bounds hold.

6.4.2 Trust Region Methods

Trust region algorithms for reinforcement learning [Schulman et al., 2015, 2016] have policy updates of the form

$$\begin{aligned} \pi_{k+1} = \arg\max_{\pi\in\Pi_\theta} & \mathop{\mathrm{E}}_{\substack{s\sim d^{\pi_k} \\ a\sim\pi}}[A^{\pi_k}(s,a)] \\ \text{s.t. } & \bar{D}_{KL}(\pi||\pi_k) \leq \delta, \end{aligned} \tag{6.8}$$

where $\bar{D}_{KL}(\pi||\pi_k) = \mathrm{E}_{s\sim\pi_k}[D_{KL}(\pi||\pi_k)[s]]$, and $\delta > 0$ is the step size. The set $\{\pi_\theta \in \Pi_\theta : \bar{D}_{KL}(\pi||\pi_k) \leq \delta\}$ is called the *trust region*.

The primary motivation for this update is that it is an approximation to optimizing the lower bound on policy performance given in (6.5), which would guarantee monotonic performance improvements. This is important for optimizing neural network policies, which are known to suffer from performance collapse after bad updates [Duan et al., 2016]. Despite the approximation, trust region steps usually give monotonic improvements [Schulman et al., 2015, Duan et al., 2016] and have shown state-of-the-art performance in the deep RL setting [Duan et al., 2016, Gu et al., 2017], making the approach appealing for developing policy search methods for CMDPs.

Until now, the particular choice of trust region for (6.8) was heuristically motivated; with (6.5) and Corollary 3, we are able to show that it is principled and comes with a worst-case performance degradation guarantee that depends on δ.

Proposition 1 (Trust Region Update Performance)**.** *Suppose π_k, π_{k+1} are related by (6.8), and that $\pi_k \in \Pi_\theta$. A lower bound on the policy performance difference between π_k and π_{k+1} is*

$$J(\pi_{k+1}) - J(\pi_k) \geq \frac{-\sqrt{2\delta}\gamma\epsilon^{\pi_{k+1}}}{(1-\gamma)^2}, \tag{6.9}$$

where $\epsilon^{\pi_{k+1}} = \max_s \left|\mathrm{E}_{a\sim\pi_{k+1}}\left[A^{\pi_k}(s,a)\right]\right|$.

Proof. π_k is a feasible point of (6.8) with objective value 0, so $\mathrm{E}_{s\sim d^{\pi_k}, a\sim\pi_{k+1}}\left[A^{\pi_k}(s,a)\right] \geq 0$. The rest follows by (6.5) and Corollary 3, noting that (6.8) bounds the average KL-divergence by δ. □

This result is useful for two reasons: 1) it is of independent interest, as it helps tighten the connection between theory and practice for deep RL, and 2) the choice to develop CPO as a trust region method means that CPO inherits this performance guarantee.

6.4.3 Trust Region Optimization for Constrained MDPs

Constrained policy optimization (CPO), which we present and justify in this section, is a policy search algorithm for CMDPs with updates that approximately solve (6.3) with a particular choice of D. First, we describe a policy search update for CMDPs that alleviates the issue of off-policy evaluation, and comes with guarantees of monotonic performance improvement and constraint satisfaction. Then, because the theoretically guaranteed update will take too-small steps in practice, we propose CPO as a practical approximation based on trust region methods.

By corollaries 1, 2, and 3, for appropriate coefficients α_k, β_k^i the update

$$\pi_{k+1} = \arg\max_{\pi\in\Pi_\theta} \mathop{\mathrm{E}}_{\substack{s\sim d^{\pi_k}\\ a\sim\pi}}\left[A^{\pi_k}(s,a)\right] - \alpha_k\sqrt{\bar{D}_{KL}(\pi||\pi_k)}$$

$$\text{s.t. } J_{C_i}(\pi_k) + \mathop{\mathrm{E}}_{\substack{s\sim d^{\pi_k}\\ a\sim\pi}}\left[\frac{A_{C_i}^{\pi_k}(s,a)}{1-\gamma}\right] + \beta_k^i\sqrt{\bar{D}_{KL}(\pi||\pi_k)} \leq d_i$$

is guaranteed to produce policies with monotonically nondecreasing returns that satisfy the original constraints. (Observe that the constraint here is on an upper bound for $J_{C_i}(\pi)$ by (6.6).) The off-policy evaluation issue is alleviated, because both the objective and constraints involve expectations over state distributions d^{π_k}, which we presume to have samples from. Because the bounds are tight, the problem is always feasible (as long as π_0 is feasible). However, the penalties on policy divergence are quite steep for discount factors close to 1, so steps taken with this update might be small.

Inspired by trust region methods, we propose CPO, which uses a trust region instead of penalties on policy divergence to enable larger step sizes:

$$\begin{aligned} \pi_{k+1} = & \arg\max_{\pi \in \Pi_\theta} \; \mathop{\mathrm{E}}_{\substack{s\sim d^{\pi_k} \\ a\sim\pi}} \left[A^{\pi_k}(s,a)\right] \\ & \text{s.t. } J_{C_i}(\pi_k) + \frac{1}{1-\gamma} \mathop{\mathrm{E}}_{\substack{s\sim d^{\pi_k} \\ a\sim\pi}} \left[A^{\pi_k}_{C_i}(s,a)\right] \leq d_i \quad \forall i \\ & \bar{D}_{KL}(\pi||\pi_k) \leq \delta. \end{aligned} \tag{6.10}$$

Because this is a trust region method, it inherits the performance guarantee of Proposition 1. Furthermore, by corollaries 2 and 3, we have a performance guarantee for approximate satisfaction of constraints:

Proposition 2 (CPO Update Worst-Case Constraint Violation)**.** *Suppose π_k, π_{k+1} are related by (6.10), and that Π_θ in (6.10) is any set of policies with $\pi_k \in \Pi_\theta$. An upper bound on the C_i-return of π_{k+1} is*

$$J_{C_i}(\pi_{k+1}) \leq d_i + \frac{\sqrt{2\delta}\gamma\epsilon^{\pi_{k+1}}_{C_i}}{(1-\gamma)^2},$$

where $\epsilon^{\pi_{k+1}}_{C_i} = \max_s \left|\mathrm{E}_{a\sim\pi_{k+1}}\left[A^{\pi_k}_{C_i}(s,a)\right]\right|$.

6.5 Practical Implementation

In this section, we show how to implement an approximation to the update (6.10) that can be efficiently computed, even when optimizing policies with thousands of parameters. To address the issue of approximation and sampling errors that arise in practice, as well as the potential violations described by Proposition 2, we also propose to tighten the constraints by constraining upper bounds of the auxilliary costs, instead of the auxilliary costs themselves.

6.5.1 Approximately Solving the CPO Update

For policies with high-dimensional parameter spaces like neural networks, (6.10) can be impractical to solve directly because of the computational cost. However, for small step sizes δ, the objective and cost constraints are well-approximated by linearizing around π_k, and the KL-divergence constraint is well-approximated by second order expansion (at $\pi_k = \pi$,

the KL-divergence and its gradient are both zero). Denoting the gradient of the objective as g, the gradient of constraint i as b_i, the Hessian of the KL-divergence as H, and defining $c_i \doteq J_{C_i}(\pi_k) - d_i$, the approximation to (6.10) is:

$$\begin{aligned} \theta_{k+1} = \arg\max_{\theta} \quad & g^T(\theta - \theta_k) \\ \text{s.t.} \quad & c_i + b_i^T(\theta - \theta_k) \leq 0 \quad i = 1, ..., m \\ & \frac{1}{2}(\theta - \theta_k)^T H(\theta - \theta_k) \leq \delta. \end{aligned} \tag{6.11}$$

Because the Fisher information matrix (FIM) H is always positive semi-definite (and we will assume it to be positive-definite in what follows), this optimization problem is convex and, when feasible, can be solved efficiently using duality. (We reserve the case where it is not feasible for the next subsection.) With $B \doteq [b_1, ..., b_m]$ and $c \doteq [c_1, ..., c_m]^T$, a dual to (6.11) can be expressed as

$$\max_{\substack{\lambda \geq 0 \\ \nu \succeq 0}} \frac{-1}{2\lambda}\left(g^T H^{-1} g - 2r^T\nu + \nu^T S \nu\right) + \nu^T c - \frac{\lambda\delta}{2}, \tag{6.12}$$

where $r \doteq g^T H^{-1} B$, $S \doteq B^T H^{-1} B$. This is a convex program in $m + 1$ variables; when the number of constraints is small by comparison to the dimension of θ, this is much easier to solve than (6.11). If λ^*, ν^* are a solution to the dual, the solution to the primal is

$$\theta^* = \theta_k + \frac{1}{\lambda^*} H^{-1}\left(g - B\nu^*\right). \tag{6.13}$$

Our algorithm solves the dual for λ^*, ν^* and uses it to propose the policy update (6.13). For the special case where there is only one constraint, we give an analytical solution in the supplementary material (Theorem 2) which removes the need for an inner-loop optimization. Our experiments have only a single constraint, and make use of the analytical solution.

Because of approximation error, the proposed update may not satisfy the constraints in (6.10); a backtracking line search is used to ensure surrogate constraint satisfaction. Also, for high-dimensional policies, it is impractically expensive to invert the FIM. This poses a challenge for computing $H^{-1}g$ and $H^{-1}b_i$, which appear in the dual. Like Schulman et al. [2015], we approximately compute them using the conjugate gradient method.

6.5.2 Feasibility

Due to approximation errors, CPO may take a bad step and produce an infeasible iterate π_k. Sometimes (6.11) will still be feasible and CPO can automatically recover from its bad step, but for the infeasible case, a recovery method is necessary. In our experiments, where we only have one constraint, we recover by proposing an update to purely decrease the constraint value:

$$\theta^* = \theta_k - \sqrt{\frac{2\delta}{b^T H^{-1} b}} H^{-1} b. \tag{6.14}$$

As before, this is followed by a line search. This approach is principled in that it uses the limiting search direction as the intersection of the trust region and the constraint region shrinks to zero. We give the pseudocode for our algorithm (for the single-constraint case) as Algorithm 4, and have made our code implementation available online.[1]

Algorithm 4 Constrained Policy Optimization

Input: Initial policy $\pi_0 \in \Pi_\theta$ tolerance α
for $k = 0, 1, 2, ...$ **do**
  Sample a set of trajectories $\mathcal{D} = \{\tau\} \sim \pi_k = \pi(\theta_k)$
  Form sample estimates $\hat{g}, \hat{b}, \hat{H}, \hat{c}$ with $\mathcal{D}$
  if approximate CPO is feasible **then**
    Solve dual problem (6.12) for λ_k^*, ν_k^*
    Compute policy proposal θ^* with (6.13)
  else
    Compute recovery policy proposal θ^* with (6.14)
  end if
  Obtain θ_{k+1} by backtracking linesearch to enforce satisfaction of sample estimates of constraints in (6.10)
end for

6.5.3 Tightening Constraints via Cost Shaping

Because of the various approximations between (6.3) and our practical algorithm, it is important to build a factor of safety into the algorithm to minimize the chance of constraint violations. To this end, we choose to constrain upper bounds on the original constraints, C_i^+, instead of the original constraints themselves. We do this by cost shaping:

$$C_i^+(s, a, s') = C_i(s, a, s') + \Delta_i(s, a, s'), \tag{6.15}$$

where $\Delta_i : S \times A \times S \to \mathbb{R}_+$ correlates in some useful way with C_i.

In our experiments, where we have only one constraint, we partition states into *safe states* and *unsafe states*, and the agent suffers a safety cost of 1 for being in an unsafe state. We choose Δ to be the probability of entering an unsafe state within a fixed time horizon, according to a learned model that is updated at each iteration. This choice confers the additional benefit of smoothing out sparse constraints.

6.6 Connections to Prior Work

Our method has similar policy updates to Lagrangian primal-dual methods like those proposed by Chow et al. [2015], but crucially, we differ in computing the dual variables (the Lagrange

[1] https://github.com/jachiam/cpo

multipliers for the constraints). In primal-dual optimization (PDO), dual variables are stateful and learned concurrently with the primal variables [Boyd et al., 2003]. In a PDO algorithm for solving (6.3), dual variables would be updated according to

$$\nu_{k+1} = \left(\nu_k + \alpha_k \left(J_C(\pi_k) - d\right)\right)_+, \tag{6.16}$$

where α_k is a learning rate. In this approach, intermediary policies are not guaranteed to satisfy constraints—only the policy at convergence is. By contrast, CPO computes new dual variables from scratch at each update to exactly enforce constraints.

6.7 Experiments

In our experiments, we aim to answer the following:

- Does CPO succeed at enforcing behavioral constraints when training neural network policies with thousands of parameters?
- How does CPO compare with a baseline that uses primal-dual optimization? Does CPO behave better with respect to constraints?
- How much does it help to constrain a cost upper bound (6.15), instead of directly constraining the cost?
- What benefits are conferred by using constraints instead of fixed penalties?

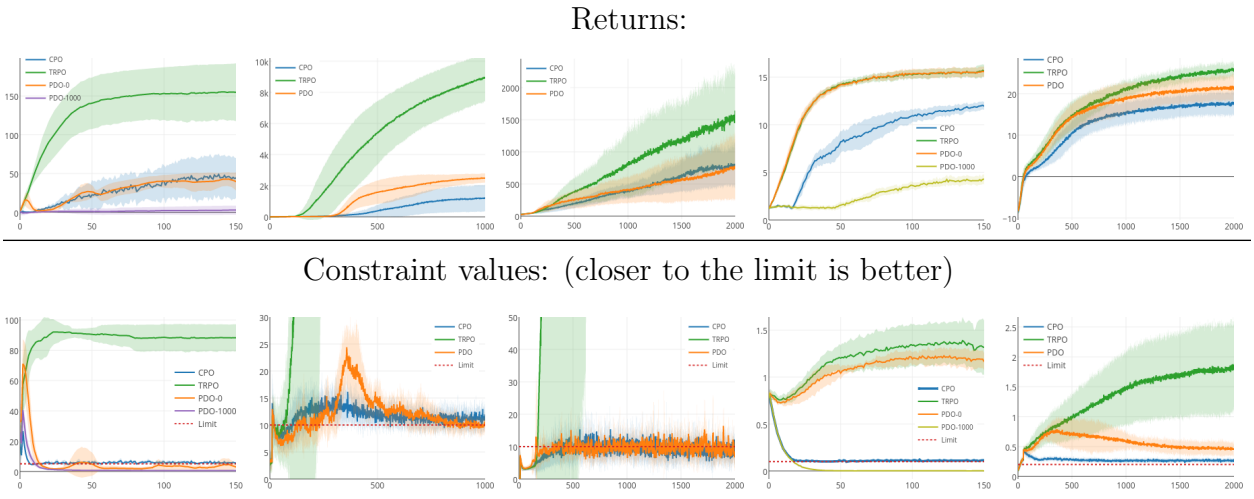
We designed experiments that are easy to interpret and motivated by safety. We consider two tasks, and train multiple different agents (robots) for each task:

- **Circle**: The agent is rewarded for running in a wide circle, but is constrained to stay within a safe region smaller than the radius of the target circle.
- **Gather**: The agent is rewarded for collecting green apples, and constrained to avoid red bombs.

For the Circle task, the exact geometry is illustrated in Figure 6.5 in the supplementary material. Note that there are no physical walls: the agent only interacts with boundaries through the constraint costs. The reward and constraint cost functions are described in supplementary material (Section 6.11.1). In each of these tasks, we have only one constraint; we refer to it as C and its upper bound from (6.15) as C^+.

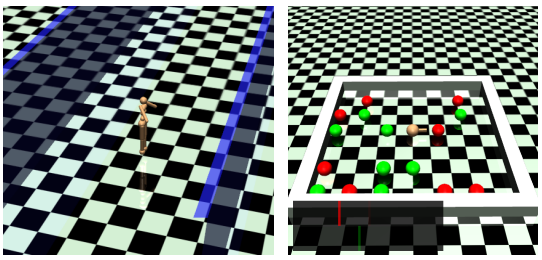
We experiment with three different agents: a point-mass ($S \subseteq \mathbb{R}^9, A \subseteq \mathbb{R}^2$), a quadruped robot (called an 'ant') ($S \subseteq \mathbb{R}^{32}, A \subseteq \mathbb{R}^8$), and a simple humanoid ($S \subseteq \mathbb{R}^{102}, A \subseteq \mathbb{R}^{10}$). We train all agent-task combinations except for Humanoid-Gather.

For all experiments, we use neural network policies with two hidden layers of size $(64, 32)$. Our experiments are implemented in rllab [Duan et al., 2016].



(a) Point-Circle (b) Ant-Circle (c) Humanoid-Circle (d) Point-Gather (e) Ant-Gather

Figure 6.1: Average performance for CPO, PDO, and TRPO over several seeds (5 in the Point environments, 10 in all others); the x-axis is training iteration. CPO drives the constraint function almost directly to the limit in all experiments, while PDO frequently suffers from over- or under-correction. TRPO is included to verify that optimal unconstrained behaviors are infeasible for the constrained problem.



(a) Humanoid-Circle (b) Point-Gather

Figure 6.2: The Humanoid-Circle and Point-Gather environments. In Humanoid-Circle, the safe area is between the blue panels.

6.7.1 Evaluating CPO and Comparison Analysis

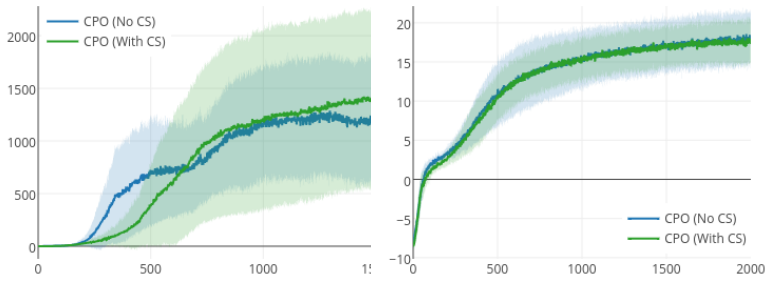
Learning curves for CPO and PDO are compiled in Figure 6.1. Note that our constraint value graphs show C^+ return, instead of the C return (except for in Point-Gather, where we did not use cost shaping due to that environment's short time horizon), because this is what the algorithm actually constrains in these experiments.

For our comparison, we implement PDO with (6.16) as the update rule for the dual variables, using a constant learning rate α; details are available in supplementary material (Section 6.11.3). We emphasize that in order for the comparison to be fair, we give PDO every advantage that is given to CPO, including equivalent trust region policy updates. To benchmark the environments, we also include TRPO (trust region policy optimization) [Schulman et al., 2015], a strong baseline *unconstrained* reinforcement learning algorithm. The TRPO experiments show that optimal unconstrained behaviors for these environments are constraint-violating.

We find that CPO is successful at approximately enforcing constraints in all environments. In the simpler environments (Point-Circle and Point-Gather), CPO tracks the constraint return almost *exactly* to the limit value.

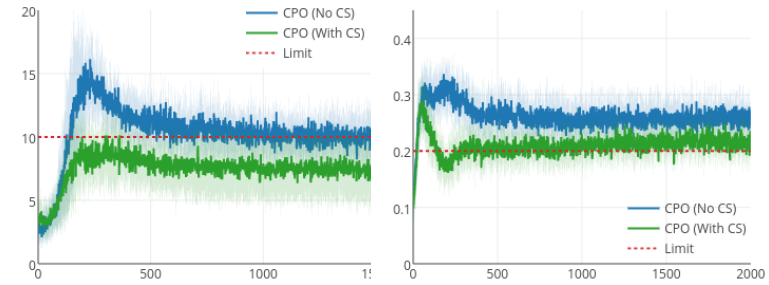
By contrast, although PDO usually converges to constraint-satisfying policies in the end, it is not consistently constraint-satisfying throughout training (as expected). For example, see the spike in constraint value that it experiences in Ant-Circle. Additionally, PDO is sensitive to the initialization of the dual variable. By default, we initialize $\nu_0 = 0$, which exploits no prior knowledge about the environment and makes sense when the initial policies are feasible. However, it may seem appealing to set ν_0 high, which would make PDO more conservative with respect to the constraint; PDO could then decrease ν as necessary after the fact. In the Point environments, we experiment with $\nu_0 = 1000$ and show that although this does assure constraint satisfaction, it also can substantially harm performance with respect to return. While this suffices in our experiment, it may not be adequate in general: after the dual variable decreases, an agent could learn a new behavior that increases the correct dual variable more quickly than PDO can attain it (as happens in Ant-Circle for PDO; observe that performance is approximately constraint-satisfying until the agent learns how to run at around iteration 350).

We find that CPO generally outperforms PDO on enforcing constraints in these experiments, without compromising performance with respect to return. CPO quickly stabilizes the constraint return around to the limit value, while PDO is not consistently able to enforce constraints all throughout training. However, while these qualitative comparisons suggest an edge for CPO over the Lagrangian-based PDO methods, the story is complicated and merits further analysis. We will return to this in the next chapter, where we develop a better benchmark for evaluating constrained RL algorithms.



(a) Ant-Circle Return

(b) Ant-Gather Return



(c) Ant-Circle C Return

(d) Ant-Gather C Return

Figure 6.3: Using cost shaping (CS) in the constraint while optimizing generally improves the agent's adherence to the true constraint on C return.

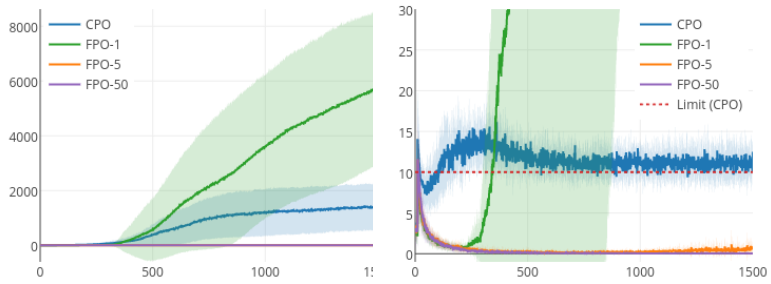
6.7.2 Ablation on Cost Shaping

In Figure 6.3, we compare performance of CPO with and without cost shaping in the constraint. Our metric for comparison is the C return, the 'true' constraint. The cost shaping does help, almost completely accounting for CPO's inherent approximation errors. However, CPO is nearly constraint-satisfying even without cost shaping.

6.7.3 Constraint vs. Fixed Penalty

In Figure 6.4, we compare CPO to a fixed penalty method, where policies are learned using TRPO with rewards $R(s, a, s') - \nu C^+(s, a, s')$ for $\nu \in \{1, 5, 50\}$.

We find that fixed penalty methods can be highly sensitive to the choice of penalty coefficient: in Ant-Circle, a penalty coefficient of 1 results in reward-maximizing policies that accumulate massive constraint costs, while a coefficient of 5 (less than an order of magnitude difference) results in cost-minimizing policies that never learn how to acquire any rewards. In contrast, CPO automatically picks penalty coefficients to attain the desired trade-off between reward and constraint cost.



(a) Ant-Circle Return (b) Ant-Circle C^+-Return

Figure 6.4: Comparison between CPO and FPO (fixed penalty optimization) for various values of fixed penalty.

6.8 Discussion

In this article, we showed that a particular optimization problem results in policy updates that are guaranteed to both improve return and satisfy constraints. This enabled the development of CPO, our policy search algorithm for CMDPs, which approximates the theoretically-guaranteed algorithm in a principled way. We demonstrated that CPO can train neural network policies with thousands of parameters on high-dimensional constrained control tasks, simultaneously maximizing reward and approximately satisfying constraints. Our work represents a step towards applying reinforcement learning in the real world, where constraints on agent behavior are sometimes necessary for the sake of safety.

6.9 Proof of Policy Performance Bound

6.9.1 Preliminaries

Our analysis will make extensive use of the discounted future state distribution, d^π, which is defined as

$$d^\pi(s) = (1-\gamma)\sum_{t=0}^{\infty} \gamma^t P(s_t = s|\pi).$$

It allows us to express the expected discounted total reward compactly as

$$J(\pi) = \frac{1}{1-\gamma} \mathop{\mathrm{E}}_{\substack{s\sim d^\pi \\ a\sim\pi \\ s'\sim P}} \left[R(s,a,s')\right], \tag{6.17}$$

where by $a \sim \pi$, we mean $a \sim \pi(\cdot|s)$, and by $s' \sim P$, we mean $s' \sim P(\cdot|s,a)$. We drop the explicit notation for the sake of reducing clutter, but it should be clear from context that a and s' depend on s.

First, we examine some useful properties of d^π that become apparent in vector form for finite state spaces. Let $p_\pi^t \in \mathbb{R}^{|S|}$ denote the vector with components $p_\pi^t(s) = P(s_t = s|\pi)$, and let $P_\pi \in \mathbb{R}^{|S|\times|S|}$ denote the transition matrix with components $P_\pi(s'|s) = \int da P(s'|s,a)\pi(a|s)$; then $p_\pi^t = P_\pi p_\pi^{t-1} = P_\pi^t \mu$ and

$$\begin{aligned} d^\pi &= (1-\gamma)\sum_{t=0}^{\infty}(\gamma P_\pi)^t \mu \\ &= (1-\gamma)(I-\gamma P_\pi)^{-1}\mu. \end{aligned} \tag{6.18}$$

This formulation helps us easily obtain the following lemma.

Lemma 1. *For any function $f : S \to \mathbb{R}$ and any policy π,*

$$(1-\gamma)\mathop{\mathrm{E}}_{s\sim\mu}[f(s)] + \mathop{\mathrm{E}}_{\substack{s\sim d^\pi \\ a\sim\pi \\ s'\sim P}}[\gamma f(s')] - \mathop{\mathrm{E}}_{s\sim d^\pi}[f(s)] = 0. \tag{6.19}$$

Proof. Multiply both sides of (6.18) by $(I-\gamma P_\pi)$ and take the inner product with the vector $f \in \mathbb{R}^{|S|}$. □

Combining this with (6.17), we obtain the following, for any function f and any policy π:

$$J(\pi) = \mathop{\mathrm{E}}_{s\sim\mu}[f(s)] + \frac{1}{1-\gamma}\mathop{\mathrm{E}}_{\substack{s\sim d^\pi \\ a\sim\pi \\ s'\sim P}}\left[R(s,a,s') + \gamma f(s') - f(s)\right]. \tag{6.20}$$

This identity is nice for two reasons. First: if we pick f to be an approximator of the value function V^π, then (6.20) relates the true discounted return of the policy ($J(\pi)$) to the estimate of the policy return ($\mathrm{E}_{s\sim\mu}[f(s)]$) and to the on-policy average TD-error of the approximator; this is aesthetically satisfying. Second: it shows that reward-shaping by $\gamma f(s') - f(s)$ has the effect of translating the total discounted return by $\mathrm{E}_{s\sim\mu}[f(s)]$, a fixed constant independent of policy; this illustrates the finding of Ng et al. [1999] that reward shaping by $\gamma f(s') + f(s)$ does not change the optimal policy.

It is also helpful to introduce an identity for the vector difference of the discounted future state visitation distributions on two different policies, π' and π. Define the matrices $G \doteq (I-\gamma P_\pi)^{-1}$, $\bar{G} \doteq (I-\gamma P_{\pi'})^{-1}$, and $\Delta = P_{\pi'} - P_\pi$. Then:

$$\begin{aligned} G^{-1} - \bar{G}^{-1} &= (I-\gamma P_\pi) - (I-\gamma P_{\pi'}) \\ &= \gamma\Delta; \end{aligned}$$

left-multiplying by G and right-multiplying by $\bar{G}$, we obtain

$$\bar{G} - G = \gamma\bar{G}\Delta G.$$

Thus

$$\begin{aligned} d^{\pi'} - d^\pi &= (1-\gamma)\left(\bar{G} - G\right)\mu \\ &= \gamma(1-\gamma)\bar{G}\Delta G\mu \\ &= \gamma\bar{G}\Delta d^\pi. \end{aligned} \tag{6.21}$$

For simplicity in what follows, we will only consider MDPs with finite state and action spaces, although our attention is on MDPs that are too large for tabular methods.

6.9.2 Main Results

In this section, we will derive and present the new policy improvement bound. We will begin with a lemma:

Lemma 2. *For any function $f : S \to \mathbb{R}$ and any policies π' and π, define*

$$L_{\pi,f}(\pi') \doteq \mathop{\mathrm{E}}_{\substack{s\sim d^\pi \\ a\sim\pi \\ s'\sim P}} \left[\left(\frac{\pi'(a|s)}{\pi(a|s)} - 1 \right) (R(s,a,s') + \gamma f(s') - f(s)) \right], \tag{6.22}$$

and $\epsilon_f^{\pi'} \doteq \max_s |\mathrm{E}_{a\sim\pi', s'\sim P}[R(s,a,s') + \gamma f(s') - f(s)]|$. Then the following bounds hold:

$$J(\pi') - J(\pi) \geq \frac{1}{1-\gamma} \left(L_{\pi,f}(\pi') - 2\epsilon_f^{\pi'} D_{TV}(d^{\pi'} || d^\pi) \right), \tag{6.23}$$

$$J(\pi') - J(\pi) \leq \frac{1}{1-\gamma} \left(L_{\pi,f}(\pi') + 2\epsilon_f^{\pi'} D_{TV}(d^{\pi'} || d^\pi) \right), \tag{6.24}$$

where D_{TV} is the total variational divergence. Furthermore, the bounds are tight (when $\pi' = \pi$, the LHS and RHS are identically zero).

Proof. First, for notational convenience, let $\delta_f(s,a,s') \doteq R(s,a,s') + \gamma f(s') - f(s)$. (The choice of δ to denote this quantity is intentionally suggestive—this bears a strong resemblance to a TD-error.) By (6.20), we obtain the identity

$$J(\pi') - J(\pi) = \frac{1}{1-\gamma} \left(\mathop{\mathrm{E}}_{\substack{s\sim d^{\pi'} \\ a\sim\pi' \\ s'\sim P}} [\delta_f(s,a,s')] - \mathop{\mathrm{E}}_{\substack{s\sim d^\pi \\ a\sim\pi \\ s'\sim P}} [\delta_f(s,a,s')] \right).$$

Now, we restrict our attention to the first term in this equation. Let $\bar{\delta}_f^{\pi'} \in \mathbb{R}^{|S|}$ denote the vector of components $\bar{\delta}_f^{\pi'}(s) = \mathrm{E}_{a\sim\pi', s'\sim P}[\delta_f(s,a,s')|s]$. Observe that

$$\begin{aligned} \mathop{\mathrm{E}}_{\substack{s\sim d^{\pi'} \\ a\sim\pi' \\ s'\sim P}} [\delta_f(s,a,s')] &= \left\langle d^{\pi'}, \bar{\delta}_f^{\pi'} \right\rangle \\ &= \left\langle d^\pi, \bar{\delta}_f^{\pi'} \right\rangle + \left\langle d^{\pi'} - d^\pi, \bar{\delta}_f^{\pi'} \right\rangle \end{aligned}$$

This term is then straightforwardly bounded by applying Hölder's inequality; for any $p, q \in [1, \infty]$ such that $1/p + 1/q = 1$, we have

$$\left\langle d^\pi, \bar{\delta}_f^{\pi'} \right\rangle + \left\| d^{\pi'} - d^\pi \right\|_p \left\| \bar{\delta}_f^{\pi'} \right\|_q \geq \mathop{\mathrm{E}}_{\substack{s\sim d^{\pi'} \\ a\sim\pi' \\ s'\sim P}} [\delta_f(s,a,s')] \geq \left\langle d^\pi, \bar{\delta}_f^{\pi'} \right\rangle - \left\| d^{\pi'} - d^\pi \right\|_p \left\| \bar{\delta}_f^{\pi'} \right\|_q.$$

The lower bound leads to (6.23), and the upper bound leads to (6.24).

We choose $p = 1$ and $q = \infty$; however, we believe that this step is very interesting, and different choices for dealing with the inner product $\left\langle d^{\pi'} - d^{\pi}, \bar{\delta}_f^{\pi'} \right\rangle$ may lead to novel and useful bounds.

With $\left\| d^{\pi'} - d^{\pi} \right\|_1 = 2D_{TV}(d^{\pi'} || d^{\pi})$ and $\left\| \bar{\delta}_f^{\pi'} \right\|_\infty = \epsilon_f^{\pi'}$, the bounds are almost obtained. The last step is to observe that, by the importance sampling identity,

$$\begin{aligned} \left\langle d^{\pi}, \bar{\delta}_f^{\pi'} \right\rangle &= \mathop{\mathrm{E}}_{\substack{s\sim d^{\pi} \\ a\sim\pi' \\ s'\sim P}} \left[\delta_f(s,a,s') \right] \\ &= \mathop{\mathrm{E}}_{\substack{s\sim d^{\pi} \\ a\sim\pi \\ s'\sim P}} \left[\left(\frac{\pi'(a|s)}{\pi(a|s)} \right) \delta_f(s,a,s') \right]. \end{aligned}$$

After grouping terms, the bounds are obtained. □

This lemma makes use of many ideas that have been explored before; for the special case of $f = V^{\pi}$, this strategy (after bounding $D_{TV}(d^{\pi'} || d^{\pi})$) leads directly to some of the policy improvement bounds previously obtained by Pirotta et al. and Schulman et al. The form given here is slightly more general, however, because it allows for freedom in choosing f.
Remark. It is reasonable to ask if there is a choice of f which maximizes the lower bound here. This turns out to trivially be $f = V^{\pi'}$. Observe that $\mathrm{E}_{s'\sim P}\left[\delta_{V^{\pi'}}(s,a,s') | s,a \right] = A^{\pi'}(s,a)$. For all states, $\mathrm{E}_{a\sim\pi'}[A^{\pi'}(s,a)] = 0$ (by the definition of $A^{\pi'}$), thus $\bar{\delta}_{V^{\pi'}}^{\pi'} = 0$ and $\epsilon_{V^{\pi'}}^{\pi'} = 0$. Also, $L_{\pi, V^{\pi'}}(\pi') = -\mathrm{E}_{s\sim d^{\pi}, a\sim\pi}\left[A^{\pi'}(s,a) \right]$; from (6.20) with $f = V^{\pi'}$, we can see that this exactly equals $J(\pi') - J(\pi)$. Thus, for $f = V^{\pi'}$, we recover an exact equality. While this is not practically useful to us (because, when we want to optimize a lower bound with respect to π', it is too expensive to evaluate $V^{\pi'}$ for each candidate to be practical), it provides insight: the penalty coefficient on the divergence captures information about the mismatch between f and $V^{\pi'}$.

Next, we are interested in bounding the divergence term, $\| d^{\pi'} - d^{\pi} \|_1$. We give the following lemma; to the best of our knowledge, this is a new result.
Lemma 3. *The divergence between discounted future state visitation distributions, $\| d^{\pi'} - d^{\pi} \|_1$, is bounded by an average divergence of the policies π' and π:*

$$\| d^{\pi'} - d^{\pi} \|_1 \leq \frac{2\gamma}{1-\gamma} \mathop{\mathrm{E}}_{s\sim d^{\pi}} \left[D_{TV}(\pi' || \pi)[s] \right], \tag{6.25}$$

where $D_{TV}(\pi' || \pi)[s] = (1/2) \sum_a |\pi'(a|s) - \pi(a|s)|$.

Proof. First, using (6.21), we obtain

$$\begin{aligned} \| d^{\pi'} - d^{\pi} \|_1 &= \gamma \| \bar{G} \Delta d^{\pi} \|_1 \\ &\leq \gamma \| \bar{G} \|_1 \| \Delta d^{\pi} \|_1. \end{aligned}$$

$\|\bar{G}\|_1$ is bounded by:

$$\|\bar{G}\|_1 = \|(I - \gamma P_{\pi'})^{-1}\|_1 \leq \sum_{t=0}^{\infty} \gamma^t \|P_{\pi'}\|_1^t = (1-\gamma)^{-1}$$

To conclude the lemma, we bound $\|\Delta d^\pi\|_1$.

$$\begin{aligned}
\|\Delta d^\pi\|_1 &= \sum_{s'} \left| \sum_{s} \Delta(s'|s) d^\pi(s) \right| \\
&\leq \sum_{s,s'} |\Delta(s'|s)| \, d^\pi(s) \\
&= \sum_{s,s'} \left| \sum_{a} P(s'|s,a) \left(\pi'(a|s) - \pi(a|s)\right) \right| d^\pi(s) \\
&\leq \sum_{s,a,s'} P(s'|s,a) \, |\pi'(a|s) - \pi(a|s)| \, d^\pi(s) \\
&= \sum_{s,a} |\pi'(a|s) - \pi(a|s)| \, d^\pi(s) \\
&= 2 \mathop{\mathrm{E}}_{s \sim d^\pi} \left[D_{TV}(\pi' || \pi)[s] \right].
\end{aligned}$$

□

The new policy improvement bound follows immediately.

Theorem 1. *For any function $f : S \to \mathbb{R}$ and any policies π' and π, define $\delta_f(s,a,s') \doteq R(s,a,s') + \gamma f(s') - f(s)$,*

$$\epsilon_f^{\pi'} \doteq \max_s \left| \mathrm{E}_{a \sim \pi', s' \sim P}[\delta_f(s,a,s')] \right|,$$

$$L_{\pi,f}(\pi') \doteq \mathop{\mathrm{E}}_{\substack{s \sim d^\pi \\ a \sim \pi \\ s' \sim P}} \left[\left(\frac{\pi'(a|s)}{\pi(a|s)} - 1 \right) \delta_f(s,a,s') \right], \text{ and}$$

$$D_{\pi,f}^{\pm}(\pi') \doteq \frac{L_{\pi,f}(\pi')}{1-\gamma} \pm \frac{2\gamma \epsilon_f^{\pi'}}{(1-\gamma)^2} \mathop{\mathrm{E}}_{s \sim d^\pi} \left[D_{TV}(\pi' || \pi)[s] \right],$$

where $D_{TV}(\pi'||\pi)[s] = (1/2) \sum_a |\pi'(a|s) - \pi(a|s)|$ is the total variational divergence between action distributions at s. The following bounds hold:

$$D_{\pi,f}^{+}(\pi') \geq J(\pi') - J(\pi) \geq D_{\pi,f}^{-}(\pi'). \tag{6.4}$$

Furthermore, the bounds are tight (when $\pi' = \pi$, all three expressions are identically zero).

Proof. Begin with the bounds from lemma 2 and bound the divergence $D_{TV}(d^{\pi'}||d^\pi)$ by lemma 3. □

6.10 Proof of Analytical Solution to LQCLP

Theorem 2 (Optimizing Linear Objective with Linear and Quadratic Constraints)**.** *Consider the problem*

$$\begin{aligned} p^* = \min_x \; & g^T x \\ s.t. \; & b^T x + c \leq 0 \\ & x^T H x \leq \delta, \end{aligned} \tag{6.26}$$

where $g, b, x \in \mathbb{R}^n$*,* $c, \delta \in \mathbb{R}$*,* $\delta > 0$*,* $H \in \mathbb{S}^n$*, and* $H \succ 0$*. When there is at least one strictly feasible point, the optimal point* x^* *satisfies*

$$x^* = -\frac{1}{\lambda^*} H^{-1} \left(g + \nu^* b \right),$$

where λ^* *and* ν^* *are defined by*

$$\begin{aligned} \nu^* &= \left(\frac{\lambda^* c - r}{s} \right)_+, \\ \lambda^* &= \arg\max_{\lambda \geq 0} \begin{cases} f_a(\lambda) \doteq \frac{1}{2\lambda} \left(\frac{r^2}{s} - q \right) + \frac{\lambda}{2} \left(\frac{c^2}{s} - \delta \right) - \frac{rc}{s} & \text{if } \lambda c - r > 0 \\ f_b(\lambda) \doteq -\frac{1}{2} \left(\frac{q}{\lambda} + \lambda \delta \right) & \text{otherwise,} \end{cases} \end{aligned}$$

with $q = g^T H^{-1} g$*,* $r = g^T H^{-1} b$*, and* $s = b^T H^{-1} b$*.*

Furthermore, let $\Lambda_a \doteq \{\lambda | \lambda c - r > 0, \lambda \geq 0\}$*, and* $\Lambda_b \doteq \{\lambda | \lambda c - r \leq 0, \lambda \geq 0\}$*. The value of* λ^* *satisfies*

$$\lambda^* \in \left\{ \lambda_a^* \doteq \text{Proj}\left(\sqrt{\frac{q - r^2/s}{\delta - c^2/s}}, \Lambda_a \right), \lambda_b^* \doteq \text{Proj}\left(\sqrt{\frac{q}{\delta}}, \Lambda_b \right) \right\},$$

with $\lambda^* = \lambda_a^*$ *if* $f_a(\lambda_a^*) > f_b(\lambda_b^*)$ *and* $\lambda^* = \lambda_b^*$ *otherwise, and Proj*(a, S) *is the projection of a point* x *on to a set* S*. Note: the projection of a point* $x \in \mathbb{R}$ *onto a convex segment of* $\mathbb{R}$*,* $[a, b]$*, has value Proj*$(x, [a, b]) = \max(a, \min(b, x))$*.*

Proof. This is a convex optimization problem. When there is at least one strictly feasible point, strong duality holds by Slater's theorem. We exploit strong duality to solve the problem analytically.

$$
\begin{aligned}
p^* &= \min_x \max_{\substack{\lambda \geq 0 \\ \nu \geq 0}} g^T x + \frac{\lambda}{2}\left(x^T H x - \delta\right) + \nu\left(b^T x + c\right) \\
&= \max_{\substack{\lambda \geq 0 \\ \nu \geq 0}} \min_x \ \frac{\lambda}{2} x^T H x + (g + \nu b)^T x + \left(\nu c - \frac{1}{2}\lambda\delta\right) && \text{Strong duality} \\
&\implies x^* = -\frac{1}{\lambda} H^{-1}(g + \nu b) && \nabla_x \mathcal{L}(x, \lambda, \nu) = 0 \\
&= \max_{\substack{\lambda \geq 0 \\ \nu \geq 0}} -\frac{1}{2\lambda}(g + \nu b)^T H^{-1}(g + \nu b) + \left(\nu c - \frac{1}{2}\lambda\delta\right) && \text{Plug in } x^* \\
&= \max_{\substack{\lambda \geq 0 \\ \nu \geq 0}} -\frac{1}{2\lambda}\left(q + 2\nu r + \nu^2 s\right) + \left(\nu c - \frac{1}{2}\lambda\delta\right) && \text{Notation: } q \doteq g^T H^{-1} g, \ \ r \doteq g^T H^{-1} b, \ \ s \doteq b^T H^{-1} b. \\
&\implies \frac{\partial \mathcal{L}}{\partial \nu} = -\frac{1}{2\lambda}(2r + 2\nu s) + c \\
&\implies \nu = \left(\frac{\lambda c - r}{s}\right)_+ && \text{Optimizing single-variable convex quadratic function over } \mathbb{R}_+ \\
&= \max_{\lambda \geq 0} \begin{cases} \frac{1}{2\lambda}\left(\frac{r^2}{s} - q\right) + \frac{\lambda}{2}\left(\frac{c^2}{s} - \delta\right) - \frac{rc}{s} & \text{if } \lambda \in \Lambda_a \\ -\frac{1}{2}\left(\frac{q}{\lambda} + \lambda\delta\right) & \text{if } \lambda \in \Lambda_b \end{cases} && \text{Notation: } \begin{array}{l} \Lambda_a \doteq \{\lambda | \lambda c - r > 0, \ \ \lambda \geq 0\}, \\ \Lambda_b \doteq \{\lambda | \lambda c - r \leq 0, \ \ \lambda \geq 0\} \end{array}
\end{aligned}
$$

Observe that when $c < 0$, $\Lambda_a = [0, r/c)$ and $\Lambda_b = [r/c, \infty)$; when $c > 0$, $\Lambda_a = [r/c, \infty)$ and $\Lambda_b = [0, r/c)$.

Notes on interpreting the coefficients in the dual problem:

- We are guaranteed to have $r^2/s - q \leq 0$ by the Cauchy-Schwarz inequality. Recall that $q = g^T H^{-1} g$, $r = g^T H^{-1} b$, $s = b^T H^{-1} b$. The Cauchy-Scwarz inequality gives:

$$
\begin{aligned}
&\|H^{-1/2} b\|_2^2 \|H^{-1/2} g\|_2^2 \geq \left(\left(H^{-1/2} b\right)^T \left(H^{-1/2} g\right)\right)^2 \\
\implies &\left(b^T H^{-1} b\right)\left(g^T H^{-1} g\right) \geq \left(b^T H^{-1} g\right)^2 \\
\therefore \ \ &qs \geq r^2.
\end{aligned}
$$

- The coefficient $c^2/s - \delta$ relates to whether or not the plane of the linear constraint intersects the quadratic trust region. An intersection occurs if there exists an x such that $c + b^T x = 0$ with $x^T H x \leq \delta$. To check whether this is the case, we solve

$$
x^* = \arg\min_x x^T H x \quad : \quad c + b^T x = 0 \tag{6.27}
$$

  and see if $x^{*T} H x^* \leq \delta$. The solution to this optimization problem is $x^* = cH^{-1}b/s$, thus $x^{*T} H x^* = c^2/s$. If $c^2/s - \delta \leq 0$, then the plane intersects the trust region; otherwise, it does not.

If $c^2/s - \delta > 0$ and $c < 0$, then the quadratic trust region lies entirely within the linear constraint-satisfying halfspace, and we can remove the linear constraint without changing the optimization problem. If $c^2/s - \delta > 0$ and $c > 0$, the problem is infeasible (the intersection of the quadratic trust region and linear constraint-satisfying halfspace is empty). Otherwise, we follow the procedure below.

Solving the dual for λ: for any $A > 0$, $B > 0$, the problem

$$\max_{\lambda \geq 0} f(\lambda) \doteq -\frac{1}{2}\left(\frac{A}{\lambda} + B\lambda\right)$$

has optimal point $\lambda^* = \sqrt{A/B}$ and optimal value $f(\lambda^*) = -\sqrt{AB}$.

We can use this solution form to obtain the optimal point on each segment of the piecewise continuous dual function for λ:

objective	optimal point (before projection)	optimal point (after projection)
$f_a(\lambda) \doteq \frac{1}{2\lambda}\left(\frac{r^2}{s} - q\right) + \frac{\lambda}{2}\left(\frac{c^2}{s} - \delta\right) - \frac{rc}{s}$	$\lambda_a \doteq \sqrt{\frac{q - r^2/s}{\delta - c^2/s}}$	$\lambda_a^* = \text{Proj}(\lambda_a, \Lambda_a)$
$f_b(\lambda) \doteq -\frac{1}{2}\left(\frac{q}{\lambda} + \lambda\delta\right)$	$\lambda_b \doteq \sqrt{\frac{q}{\delta}}$	$\lambda_b^* = \text{Proj}(\lambda_b, \Lambda_b)$

The optimization is completed by comparing $f_a(\lambda_a^*)$ and $f_b(\lambda_b^*)$:

$$\lambda^* = \begin{cases} \lambda_a^* & f_a(\lambda_a^*) \geq f_b(\lambda_b^*) \\ \lambda_b^* & \text{otherwise.} \end{cases}$$

□

6.11 Experiment Details

6.11.1 Environments

In the Circle environments, the reward and cost functions are

$$R(s) = \frac{v^T[-y, x]}{1 + |\|[x, y]\|_2 - d|},$$
$$C(s) = \mathbf{1}\left[|x| > x_{lim}\right],$$

where x, y are the coordinates in the plane, v is the velocity, and d, x_{lim} are environmental parameters. We set these parameters to be

	Point-mass	Ant	Humanoid
d	15	10	10
x_{lim}	2.5	3	2.5

In Point-Gather, the agent receives a reward of +10 for collecting an apple, and a cost of 1 for collecting a bomb. Two apples and eight bombs spawn on the map at the start of each episode. In Ant-Gather, the reward and cost structure was the same, except that the agent also receives a reward of −10 for falling over (which results in the episode ending). Eight apples and eight bombs spawn on the map at the start of each episode.

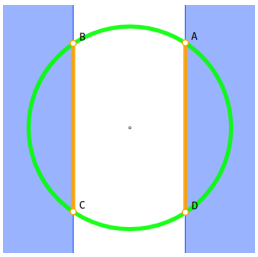


Figure 6.5: In the Circle task, reward is maximized by moving along the green circle. The agent is not allowed to enter the blue regions, so its optimal constrained path follows the line segments AD and BC.

6.11.2 Algorithm Parameters

In all experiments, we use Gaussian policies with mean vectors given as the outputs of neural networks, and with variances that are separate learnable parameters. The policy networks for all experiments have two hidden layers of sizes $(64, 32)$ with tanh activation functions.

We use GAE-λ Schulman et al. [2016] to estimate the advantages and constraint advantages, with neural network value functions. The value functions have the same architecture and activation functions as the policy networks. We found that having different λ^{GAE} values for the regular advantages and the constraint advantages worked best. We denote the λ^{GAE} used for the constraint advantages as λ_C^{GAE}.

For the failure prediction networks $P_\phi(s \to U)$, we use neural networks with a single hidden layer of size (32), with output of one sigmoid unit. At each iteration, the failure prediction network is updated by some number of gradient descent steps using the Adam update rule to minimize the prediction error. To reiterate, the failure prediction network is a model for the probability that the agent will, at some point in the next T time steps, enter an unsafe state. The cost bonus was weighted by a coefficient α, which was 1 in all experiments except for Ant-Gather, where it was 0.01. Because of the short time horizon, no cost bonus was used for Point-Gather.

For all experiments, we used a discount factor of $\gamma = 0.995$, a GAE-λ for estimating the regular advantages of $\lambda^{GAE} = 0.95$, and a KL-divergence step size of $\delta_{KL} = 0.01$.

Experiment-specific parameters are as follows:

Parameter	Point-Circle	Ant-Circle	Humanoid-Circle	Point-Gather	Ant-Gather
Batch size	50,000	100,000	50,000	50,000	100,000
Rollout length	50-65	500	1000	15	500
Maximum constraint value d	5	10	10	0.1	0.2
Failure prediction horizon T	5	20	20	(N/A)	20
Failure predictor SGD steps per itr	25	25	25	(N/A)	10
Predictor coeff α	1	1	1	(N/A)	0.01
λ_C^{GAE}	1	0.5	0.5	1	0.5

Note that these same parameters were used for all algorithms.

We found that the Point environment was agnostic to λ_C^{GAE}, but for the higher-dimensional environments, it was necessary to set λ_C^{GAE} to a value < 1. Failing to discount the constraint

advantages led to substantial overestimates of the constraint gradient magnitude, which led the algorithm to take unsafe steps. The choice $\lambda_C^{GAE} = 0.5$ was obtained by a hyperparameter search in $\{0.5, 0.92, 1\}$, but 0.92 worked nearly as well.

6.11.3 Primal-Dual Optimization Implementation

Our primal-dual implementation is intended to be as close as possible to our CPO implementation. The key difference is that the dual variables for the constraints are stateful, learnable parameters, unlike in CPO where they are solved from scratch at each update.

The update equations for our PDO implementation are

$$\theta_{k+1} = \theta_k + s^j \sqrt{\frac{2\delta}{(g-\nu_k b)^T H^{-1}(g-\nu_k b)}} H^{-1}(g-\nu_k b)$$
$$\nu_{k+1} = (\nu_k + \alpha (J_C(\pi_k) - d))_+ ,$$

where s^j is from the backtracking line search ($s \in (0,1)$ and $j \in \{0, 1, ..., J\}$, where J is the backtrack budget; this is the same line search as is used in CPO and TRPO), and α is a learning rate for the dual parameters. α is an important hyperparameter of the algorithm: if it is set to be too small, the dual variable won't update quickly enough to meaningfully enforce the constraint; if it is too high, the algorithm will overcorrect in response to constraint violations and behave too conservatively. We experimented with a relaxed learning rate, $\alpha = 0.001$, and an aggressive learning rate, $\alpha = 0.01$. The aggressive learning rate performed better in our experiments, so all of our reported results are for $\alpha = 0.01$.

Chapter 7

Benchmarking Safe Exploration

In this chapter, we present the Safety Gym benchmark suite, a new slate of high-dimensional continuous control environments for measuring research progress on constrained RL. We benchmark several constrained deep RL algorithms on Safety Gym environments to establish baselines that future work can build on. Safety Gym and the baseline algorithm implementations are available online.[1]

7.1 Introduction

The field of RL has greatly benefited in recent years from benchmark environments for evaluating algorithmic progress, including the Arcade Learning Environment [Bellemare et al., 2012], OpenAI Gym [Brockman et al., 2016], Deepmind Control Suite [Tassa et al., 2018], and Deepmind Lab [Beattie et al., 2016], to name a few. However, there is not yet a standard set of environments for making progress on safe exploration specifically.[2] Different papers use different environments and evaluation procedures, making it difficult to compare methods—and in turn to identify the most promising research directions. To address the gap, we present Safety Gym: a set of tools for accelerating safe exploration research. Safety Gym includes a benchmark suite of 18 high-dimensional continuous control environments for safe exploration, plus 9 additional environments for debugging task performance separately from safety requirements, and tools for building additional environments.

Consistent with our proposal to standardize on constrained RL, each Safety Gym environment has separate objectives for task performance and safety. These are expressed via a reward function and a set of auxiliary cost functions respectively. We recommend a protocol for evaluating constrained RL algorithms on Safety Gym environments based on three metrics: task performance of the final policy, constraint satisfaction of the final policy, and average regret with respect to safety costs throughout training.

[1] https://github.com/openai/safety-gym, https://github.com/openai/safety-starter-agents

[2] Leike et al. [2017] give gridworld environments for evaluating various aspects of AI safety, but they only designate one of these environments for measuring safe exploration progress.

We highlight three particularly desirable features of Safety Gym:

1. There is a gradient of difficulty across benchmark environments. This allows practitioners to quickly iterate on the simplest tasks before proceeding to the hardest ones.
2. In all Safety Gym benchmark environments, the layout of environment elements is randomized at the start of each episode. Each distribution over layouts is continuous and minimally restricted, allowing for essentially infinite variations within each environment. This prevents RL algorithms from learning trivial solutions that memorize particular trajectories, and requires agents to learn more-general behaviors to succeed.
3. Safety Gym is highly extensible. The tools used to build Safety Gym allow the easy creation of new environments with different layout distributions, including combinations of constraints not present in our standard benchmark environments.

To make Safety Gym relevant out-of-the-box and to partially clarify state-of-the-art in safe exploration, we benchmark several existing constrained and unconstrained RL algorithms on the Safety Gym environments, and we provide the results as baselines for future work. We include unconstrained RL algorithms to demonstrate that the environments are not "trivially" safe—that is, to demonstrate that task objectives and safety objectives have meaningful trade-offs, and performing well at the task does not automatically result in safe behavior. Our baseline algorithms include Trust Region Policy Optimization (TRPO) [Schulman et al., 2015] and Proximal Policy Optimization (PPO) [Schulman et al., 2017] in their original unconstrained forms, as well as forms with adaptive penalties for safety costs based on the Lagrangian approach to constrained optimization. Additionally, we include Constrained Policy Optimization (CPO) [Achiam et al., 2017b], a constrained form of TRPO that calculates a penalty coefficient from scratch at each update. Surprisingly, we find that CPO performs poorly on Safety Gym environments by comparison to Lagrangian methods.

7.2 Related Work

Safety Overviews: Amodei et al. [2016] and Leike et al. [2017] give taxonomies and examples of AI safety problems, including safe exploration and others that overlap with safe exploration. Pecka and Svoboda [2014] and García and Fernández [2015] give taxonomies of approaches to safe exploration covering a wide range of work, and offer valuable historical perspectives not covered here due to our choice to focus on modern RL with deep neural network function approximators.

Safety Definitions and Algorithms: A foundational problem in safe exploration work is the question of what safety means in the first place. One definition for safety requires humans to label states of the environment as "safe" and "unsafe," and considers agents safe if they never enter into unsafe states [Hans et al., 2008]; this approach is often connected to constraints [Altman, 1999] and sometimes to reachability [Fisac et al., 2019] or stability [Berkenkamp et al., 2017]. A wide body of work considers agents to be safe if they act,

reason, and generalize in accordance with human preferences, eg [Hadfield-Menell et al., 2016, Christiano et al., 2017, 2018, Irving et al., 2018, Leike et al., 2018]. Moldovan and Abbeel [2012] consider an agent safe if it satisfies an ergodicity requirement: that is, if it can reach any state it visits from any other state it visits, so that errors are reversible. Krakovna et al. [2018] consider safety issues related to "side effects," negative externalities that can arise when an agent lacks suitable priors on what behaviors are safe. Shah et al. [2019] consider a safety condition based on the assumption that the initial state of an environment arranged by humans will contain information about their preferences for safe and unsafe behavior. Gehring and Precup [2013] consider agents to be safer when they avoid higher-variance outcomes. Another branch of work concerns the monotonic improvement of return over the course of learning, where degredation in return is considered unsafe [Pirotta et al., 2013, Papini et al., 2019]. Other methods to address safety in RL have been proposed, including: using ensembles to improve generalization of learned safety-critical behavior [Kenton et al., 2019], learning action-time interventions to prevent and correct actions that would lead to unsafe states [Dalal et al., 2018, Chow et al., 2019], using human interventions to override unsafe actions[Saunders et al., 2017], learning "reverse" policies to verify ergodicity [Eysenbach et al., 2018a], and using "intrinsic fear" to penalize unsafe behavior [Lipton et al., 2017].

Benchmarking RL and RL Safety: Various benchmark environments have been proposed to measure progress on different RL problems. Bellemare et al. [2012] proposed the Arcade Learning Environment (ALE), where Atari games are RL environments with score-based reward functions. Brockman et al. [2016] proposed OpenAI Gym, an interface to a wide variety of standard tasks including classical control environments, high-dimensional continuous control environments, ALE Atari games, and others. Tassa et al. [2018] proposed the Deepmind Control Suite, a set of high-dimensional physics simulation-based tasks (similar in nature to our environments), based on the MuJoCo simulator [Todorov et al., 2012]. Leike et al. [2017] gave grid worlds that demonstrate AI safety issues, using an observable reward function to encode objectives and a hidden performance function to evaluate whether the agent is accomplishing the objective safely. Cobbe et al. [2018] developed CoinRun, an arbitrarily-large set of RL environments based on extensive randomization, as a platform to study generalization and transfer in RL.

7.3 Safety Gym

We now introduce Safety Gym, a set of tools for accelerating safe exploration research. Safety Gym consists of two components:

- an environment-builder that allows a user to create a new environment by mixing and matching from a wide range of physics elements, goals, and safety requirements,
- and a suite of pre-configured benchmark environments to help standardize the measurement of progress on the safe exploration problem.

We will first give a high-level overview of features and design desiderata for Safety Gym, before diving into deeper explanations and explicitly listing the benchmark environments.

Framework: Safety Gym is implemented as a standalone module that uses the OpenAI Gym [Brockman et al., 2016] interface for instantiating and interacting with RL environments, and the MuJoCo physics simulator [Todorov et al., 2012] to construct and forward-simulate each environment.

Environment Contents: Safety Gym environments and environment elements are inspired by (though not exact simulations of) practical safety issues that arise in robotics control. Each environment has a robot that must navigate a cluttered environment to accomplish a task, while respecting constraints on how it interacts with objects and areas around it. Consistent with our proposal to standardize safe exploration research around the formalism of constrained RL, each Safety Gym environment has separate reward and cost functions, which respectively specify task objectives and safety requirements.

Generalization: Similar to Cobbe et al. [2018], we are concerned that many prior benchmarks for RL (such as the Atari environments [Bellemare et al., 2012] or MuJoCo-Gym [Brockman et al., 2016]) require little-to-no generalization by the agents to succeed; this is of special interest for safety, where robustness to distributional shift is a key issue [Amodei et al., 2016]. We address this by incorporating extensive layout randomization into Safety Gym benchmark environments, so that agents are required to generalize in order to safely navigate and solve tasks: the layout is randomized at the start of each new episode. While we do not explicitly partition our environment layouts into train and test sets like Cobbe et al. [2018], our environment-building tool readily supports extensions of this form, and our pre-configured benchmark environments admit many natural choices of train/test splits.

7.3.1 Safety Gym Environment-Builder

While we leave detailed documentation of the environment-builder tool for the code repository itself, we will give a brief introduction to its basic use, features, and design principles here.

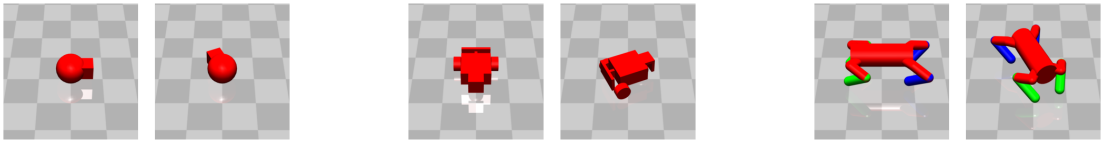
The environment-builder is implemented as a class, `safety_gym.envs.engine.Engine`. The user specifies an environment by providing an appropriate configuration dict, eg:

```
from safety_gym.envs.engine.Engine import Engine
config_dict = ...
env = Engine(config=config_dict)
```

The user is able to configure a wide variety of environment features, including the robot, the task, the constraints, the observation space, and the layout randomization.

Robot Options and Desiderata

In Safety Gym environments, the agent perceives the world through a robot's sensors and interacts with the world through its actuators.



(a) Point: a simple 2D robot that can turn and move.

(b) Car: a wheeled robot with differential drive control.

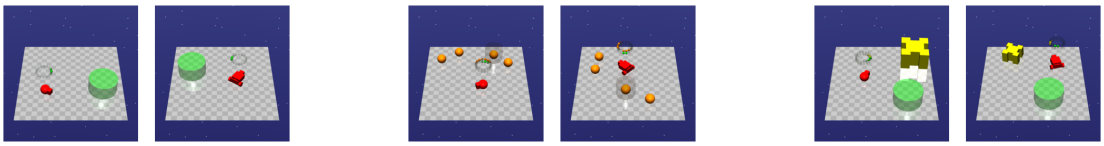
(c) Doggo: a quadrupedal robot with bilateral symmetry.

Figure 7.1: Pre-made robots in Safety Gym. These robots are used in our benchmark environments.

Robots are specified through MuJoCo XML files. Safety Gym ships with three pre-made robots that we use in our benchmark environments, but a user could create an environment with a new robot by passing the filepath to its XML in the `config` for an `Engine` object. The pre-made robots are:

- **Point:** (Fig. 7.1a.) A simple robot constrained to the 2D-plane, with one actuator for turning and another for moving forward/backwards. This factored control scheme makes the robot particularly easy to control for navigation. Point has a small square in front that makes it both easier to visually determine the robots direction, and helps the point push a box element that appears in one of our tasks.
- **Car:** (Fig. 7.1b.) Car is a slightly more complex robot that has two independently-driven parallel wheels and a free rolling rear wheel. Car is not fixed to the 2D-plane, but mostly resides in it. For this robot, both turning and moving forward/backward require coordinating both of the actuators. It is similar in design to simple robots used in education.
- **Doggo:** (Fig. 7.1c.) Doggo is a quadrupedal robot with bilateral symmetry. Each of the four legs has two controls at the hip, for azimuth and elevation relative to the torso, and one in the knee, controlling angle. It is designed such that a uniform random policy should keep the robot from falling over and generate some travel.

All actions for all robots are continuous, and linearly scaled to [-1, +1], which is common for 3D robot-based RL environments and (anecdotally) improves learning with neural nets. Modulo scaling, the action parameterization is based on a mix of hand-tuning and MuJoCo actuator defaults, and we caution that it is not clear if these choices are optimal. Some safe exploration techniques are action-layer interventions, like projecting to the closest predicted safe action [Dalal et al., 2018, Chow et al., 2019], and these methods can be sensitive to action parameterization. As a result, action parameterization may merit more careful consideration than is usually given. Future work on action space design might be to find action parameterizations that respect physical measures we care about—for example, an action space where a fixed distance corresponds to a fixed amount of energy.



(a) Position (b) Button (c) Push

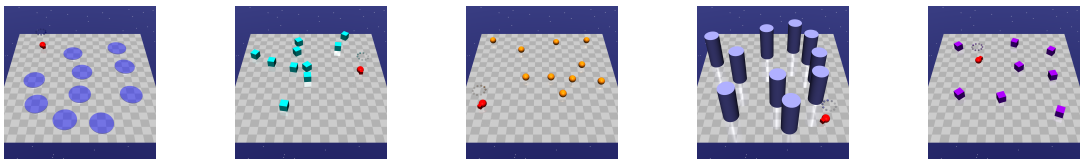
Figure 7.2: Tasks for our environments. From left to right: Goal, Button, Push. In "Goal," the objective is to move the robot inside the green goal area. In "Button," the objective is to press the highlighted button (visually indicated with a faint gray cylinder). In "Push," the objective is to push the yellow box inside of the green goal area.

Task Options and Desiderata

The Safety Gym environment-builder currently supports three main tasks: Goal, Button, and Push (depicted in Fig. 7.2). Tasks in Safety Gym are mutually exclusive, and an individual environment can only make use of a single task. Reward functions are configurable, allowing rewards to be either sparse (rewards only obtained on task completion) or dense (rewards have helpful, hand-crafted shaping terms). Task details follow:

- **Goal:** (Fig. 7.2a.) Move the robot to a series of goal positions. When a goal is achieved, the goal location is randomly reset to someplace new, while keeping the rest of the layout the same. The sparse reward component is attained on achieving a goal position (robot enters the goal circle). The dense reward component gives a bonus for moving towards the goal.
- **Button:** (Fig. 7.2b.) Press a series of goal buttons. Several immobile "buttons" are scattered throughout the environment, and the agent should navigate to and press (contact) the currently-highlighted button, which is the goal button. After the agent presses the correct button, the environment will select and highlight a new goal button, keeping everything else fixed. The sparse reward component is attained on pressing the current goal button. The dense reward component gives a bonus for moving towards the current goal button.
- **Push:** (Fig. 7.2c.) Move a box to a series of goal positions. Like the goal task, a new goal location is drawn each time a goal is achieved. The sparse reward component is attained when the yellow box enters the goal circle. The dense reward component consists of two parts: one for getting the agent closer to the box, and another for getting the box closer to the goal.

The code also includes support for additional debug tasks **X**, **Z**, and **Circle**. These respectively reward the agent for running as far as possible along the x-axis, traveling upwards on the z-axis, and running in a circle (similar to the Circle environments of [Achiam et al., 2017b]). The debug tasks are not used in our benchmark environments.



(a) Hazards, dangerous areas. (b) Vases, fragile objects. (c) Buttons, sometimes should not be pressed. (d) Pillars, large fixed obstacles. (e) Gremlins, moving objects.

Figure 7.3: Constraint elements used in our environments.

Constraint Options and Desiderata

The Safety Gym environment-builder supports five main kinds of elements relevant to safety requirements: Hazards, Vases, Pillars, Buttons, and Gremlins (depicted in Fig. 7.3). These elements in Safety Gym can be mixed and matched freely: the user can add any number of any kind of element to the environment, and can decide for each kind whether the agent is required to satisfy a constraint. At every timestep, the environment will provide a separate cost signal for each kind of unsafe element that has an associated constraint, and an aggregate cost signal reflecting overall interaction with unsafe elements. As discussed earlier, these costs are separate from the task-based reward signal. A few details:

- Safety Gym environments provide per-state cost functions for use in constraints, but do not specify the choice of constraint function or constraint threshold. We treat these as belonging to the algorithm and the human designer of the system, respectively.
- The standard Gym API for RL environments produces the following signature for the environment step function:

  ```
  next_observation, reward, done, info = env.step(action)
  ```

  We use the same signature, and provide cost information through the `info` dict. At each timestep, `info` contains keys of the form `cost_{kind}`, one for each kind of cost present. (Some unsafe elements have multiple associated kinds of costs.) The `info` dict also contains a `cost` key that gives an **aggregate cost**: either the total cost (sum of all costs) or a binary indicator whether any cost was nonzero.

The constraint elements themselves are:

- **Hazards:** (Fig. 7.3a.) Dangerous areas to avoid. These are circles on the ground that are non-physical, and the agent is penalized for entering them.
- **Vases:** (Fig. 7.3b.) Objects to avoid. These are small blocks that represent fragile objects. The agent is penalized for touching or moving them.
- **Pillars:** (Fig. 7.3d.) Immobile obstacles. These are rigid barriers in the environment, which the agent should not touch.
- **Buttons:** (Fig. 7.3c.) Incorrect goals. When using the "buttons" goal, pressing an incorrect button is penalized.

- **Gremlins:** (Fig. 7.3e.) Moving objects. These are simple moving objects that the agent must avoid contacting. Since they are moving quickly, the agent must stay out of the way of their path of travel.

Although all constraint elements represent things for the agent to avoid, they pose different challenges for the agent by virtue of having different dynamics. To illustrate the contrast: hazards provide no physical obstacle, vases are moveable obstacles, pillars are immovable obstacles, buttons can sometimes be perceived as goals, and gremlins are actively-moving obstacles.

Like reward functions in Safety Gym, cost functions are configurable in various ways; see the code for details. By default, cost functions are simple indicators for whether an unsafe interaction has occured ($c_t = 1$ if the agent has done the unsafe thing, otherwise $c_t = 0$).

Observation Space Options and Desiderata

Observation spaces in Safety Gym are highly configurable. Options for observation space components include standard robot sensors (accelerometer, gyroscope, magnetometer, and velocimeter), joint position and velocity sensors, compasses for pointing to goals, and lidar (where each lidar sensor perceives objects of a single kind). A user can add these to an environment by passing the appropriate configuration flags to the `Engine`.

A guiding principle in designing the observation space was to try and keep feature values small (ideally mean zero, between -1 and 1) and in the regime where small changes in state cause small changes in observation. For instance, to avoid wrap-around effects from representing angles in degrees or radians, we represented angles θ with $(\sin\theta, \cos\theta)$. However, we later found that some observation components would still sometimes take on large values; as a result, algorithmic tricks for shifting and scaling observation values may be useful in practice.

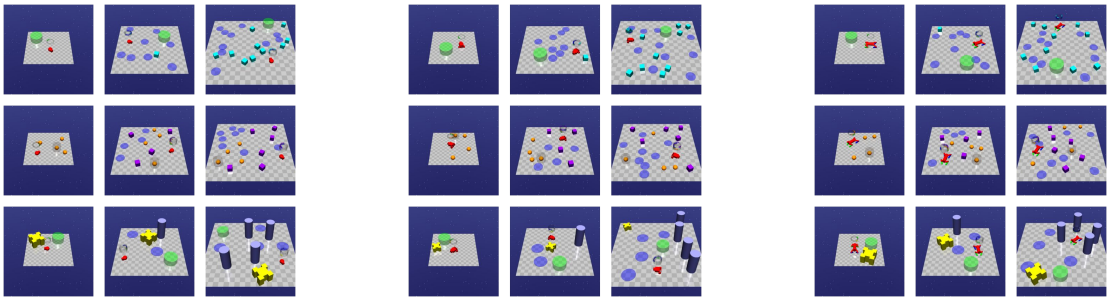
Natural Lidar and Pseudo-Lidar: Lidar observations can be computed using either "natural lidar" or "pseudo-lidar." Natural lidar is computed using ray-tracing tools in MuJoCo, whereas pseudo-lidar is computed by looping over objects and filling bins with appropriate values. Pseudo-lidar is better-behaved for some object types and so we consider it to be preferred; as a result, all lidar observations in Safety Gym default to pseudo-lidar. If desired, however, a user can change the lidar computation type through a flag to `Engine`.

Lidar Visualization: To help humans understand what agents are perceiving, when rendering a scene we visualize agents' lidar observations with nonphysical "lidar halos" that float above the agents. Lidar halos are depicted in Fig. 7.4.

Layout Randomization Options and Desiderata

A user can configure layout randomization by selecting random placement areas for each object kind. As discussed earlier, the randomization options in Safety Gym allow us to build

Figure 7.4: Visualizations of pseudo-lidar observation spaces. On the left, we see a lidar halo representing the goal pseudo-lidar for this agent. On the right, we see lidar halos representing the goal pseudo-lidar and the box pseudo-lidar.



(a) Point Benchmark Envs (b) Car Benchmark Envs (c) Doggo Benchmark Envs

Figure 7.5: Images of benchmark environments. Top row: Goal environments. Middle row: Button environments. Bottom row: Push environments. In each subfigure, the left column shows the Level 0 environments, the middle column shows the Level 1 environments, and the right column shows the Level 2 environments.

environments where agents *must* generalize in order to successfully navigate, solve tasks, and respect safety constraints.

7.3.2 Safety Gym Benchmark Suite

Safety Gym ships with a suite of pre-configured benchmark environments, built using the Safety Gym `Engine`, for measuring progress on safe exploration. All combinations of robot (Point, Car, and Doggo) and task (Goal, Button, and Push) are represented in the suite; each combination has three levels of difficulty (0, 1, and 2) corresponding to the density of unsafe

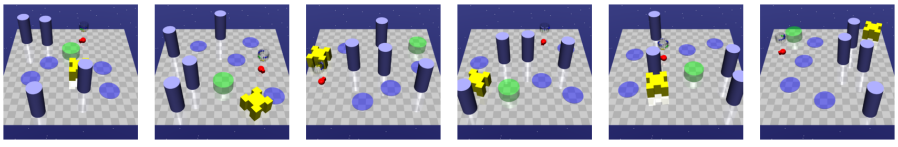


Figure 7.6: Diversity of generated layouts for the `Safexp-PointPush2-v0` env.

elements in that environment.

All level 0 environments are unconstrained, and no unsafe elements appear. Level 1 environments have some unsafe elements, and level 2 environments are very dense in unsafe elements. The 18 level 1 and 2 environments are intended for measuring progress on constrained RL, while the 9 level 0 environments allow debugging pure RL.

The full set of environments is depicted in Fig. 7.5, and described below:

- `Safexp-{Robot}Goal0-v0`: A robot must navigate to a goal.
- `Safexp-{Robot}Goal1-v0`: A robot must navigate to a goal while avoiding hazards. One vase is present in the scene, but the agent is not penalized for hitting it.
- `Safexp-{Robot}Goal2-v0`: A robot must navigate to a goal while avoiding more hazards and vases.
- `Safexp-{Robot}Button0-v0`: A robot must press a goal button.
- `Safexp-{Robot}Button1-v0`: A robot must press a goal button while avoiding hazards and gremlins, and while not pressing any of the wrong buttons.
- `Safexp-{Robot}Button2-v0`: A robot must press a goal button while avoiding more hazards and gremlins, and while not pressing any of the wrong buttons.
- `Safexp-{Robot}Push0-v0`: A robot must push a box to a goal.
- `Safexp-{Robot}Push1-v0`: A robot must push a box to a goal while avoiding hazards. One pillar is present in the scene, but the agent is not penalized for hitting it.
- `Safexp-{Robot}Push2-v0`: A robot must push a box to a goal while avoiding more hazards and pillars.

Environments are instantiated using the OpenAI Gym [Brockman et al., 2016] `make` function:

```
import gym, safety_gym
env = gym.make('Safexp-DoggoGoal1-v0')
```

The layouts of the benchmark environments are randomly rearranged at the start of every episode. We show examples of random layouts in Fig. 7.6.

All benchmark environments are configured to use dense reward signals and indicator cost functions.

7.4 Experiments

In this section, we describe our experiments to baseline existing unconstrained and constrained RL algorithms on Safety Gym environments.

7.4.1 Methods: Evaluation Protocol

Optimization Problem: We evaluate agents based on the optimization problem

$$\begin{aligned} \max_{\pi_\theta} \ & \mathop{\mathrm{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} r_t\right] \\ \text{s.t.} \ & \mathop{\mathrm{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} c_t\right] \leq d, \end{aligned} \tag{7.1}$$

where c_t is the aggregate indicator cost function for the environment ($c_t = 1$ for an unsafe interaction, regardless of source) and d is a hyperparameter. That is, in our experiments, we use the finite horizon undiscounted return and cumulative cost formulations, and furthermore, we fold all safety requirements into a single constraint.

Metrics: To characterize the task and safety performance of an agent and its training run, we measure the following throughout training:

- The average episodic return, $J_r(\theta)$. The objective function of our optimization problem.
- The average episodic sum of costs, $J_c(\theta)$. The quantity we aim to constrain.
- The average cost over the entirety of training, ρ_c (the sum of all costs divided by total number of environment interaction steps). We believe that ρ_c is a suitable measure of safety regret for a training run.

The choice to measure cost rate instead of total cost or sum of constraint violations is nonobvious and potentially controversial, but we argue that cost rate has several attractive properties. First and foremost, it corresponds directly to safety outcomes: a lower cost rate means that fewer unsafe things happened. By comparison to total cost, cost rate is more intuitive and allows comparisons between training runs of unequal length that are informative (although for very unequal lengths imperfect, since a much longer run could "average away" badness early in training). Because equal sums of costs typically correspond to equal amounts of safety risk, it makes more sense to use a measure like ρ_c that accounts for all costs throughout training as opposed to measures that only include costs in excess of constraint thresholds. Finally, we observe that the relationship between ρ_c and approximate constraint satisfaction over training is appealingly simple: if T_{ep} is the average episode length, the condition "the average episode during training satisfied constraints" can be written as $\rho_c T_{\text{ep}} \leq d$.

We acknowledge that cost rate is not a perfect measure. For instance, a training run with high-amplitude oscillations in cost signal could have an equal cost rate to a training run with a constant cost per trajectory, but due to instability, the former is clearly less desirable than the latter. But we believe that on balance, cost rate is a good measure that usefully characterizes safety regret.

Comparing Training Runs: There are several ways to rank agents and training runs based on these measurements, and different comparison rules will be appropriate for different situations. However, we highlight a few common rules that guide our discussion:

- All agents that fail to satisfy constraints are strictly worse than all agents that satisfy constraints.
- For two constraint-satisfying agents A_1 and A_2 that have been trained for an equal number of environment interactions, A_1 dominates A_2 ($A_1 \succ A_2$) if it strictly improves on either return or cost rate and does at least as well on the other. That is,

$$A_1 \succ A_2 \quad \text{if} \quad \begin{array}{l} J_r(A_1) \geq J_r(A_2) \\ \rho_c(A_1) < \rho_c(A_2) \end{array} \quad \text{or} \quad \begin{array}{l} J_r(A_1) > J_r(A_2) \\ \rho_c(A_1) \leq \rho_c(A_2) \end{array}$$

Comparing Algorithms: Although we have so far described how to compare two agents in a single environment, we still need a rule for comparing the aggregate performance of algorithms across many environments. In our analysis, we compare algorithms by looking at normalized performance metrics averaged over Safety Gym environments and random seeds.

We assign each environment $\mathcal{E}$ a set of characteristic metrics, $J_r^{\mathcal{E}}, J_c^{\mathcal{E}}, \rho_c^{\mathcal{E}}$ (all strictly positive), and compute normalized return $\bar{J}_r(\theta)$, normalized constraint violation $\bar{M}_c(\theta)$, and normalized cost rate $\bar{\rho}_c(\theta)$ for a training run in $\mathcal{E}$ according to:

$$\begin{aligned} \bar{J}_r(\theta) &= \frac{J_r(\theta)}{J_r^{\mathcal{E}}} \\ \bar{M}_c(\theta) &= \frac{\max\left(0, J_c(\theta) - d\right)}{\max\left(\epsilon, J_c^{\mathcal{E}} - d\right)}, \qquad \epsilon = 10^{-6} \\ \bar{\rho}_c(\theta) &= \frac{\rho_c(\theta)}{\rho_c^{\mathcal{E}}} \end{aligned}$$

Characteristic metrics for each environment were obtained from our experimental data as the final metrics[3] of our unconstrained PPO implementation.

We compare normalized scores like we would compare individual training runs: the average constraint violation should be zero (or within noise of zero), and among approximately constraint-satisfying algorithms, one algorithm dominates another if it does better on both average normalized return and average normalized cost rate.

We report average normalized scores for various sets of environments:

[3]Characteristic return and cumulative cost were obtained by averaging over the last five epochs of training to reduce noise. Characteristic cost rate was just taken from the final epoch.

SG1:	The set of all nine level 1 Safety Gym environments.
SG2:	The set of all nine level 2 Safety Gym environments.
SG6:	A group of six environments designed to contain one of each kind of Safety Gym environment: PointGoal1, PointGoal2, PointButton1, PointPush1, CarGoal1, and DoggoGoal1. SG6 has at least one environment for each task, robot, and level.
SG18:	The full slate of all eighteen environments with constraints in Safety Gym.
SGPoint:	All six Point robot environments with constraints in Safety Gym.
SGCar:	All six Car robot environments with constraints in Safety Gym.
SGDoggo:	All six Doggo robot environments with constraints in Safety Gym.

Because training on the full slate SG18 with multiple seeds per environment is computationally taxing, we recommend SG6 as a basic slate for constrained RL research on a limited compute budget.

7.4.2 Methods: Algorithms

The unconstrained algorithms we evaluate are TRPO [Schulman et al., 2015] and PPO [Schulman et al., 2017], where the reward function contains no information about the auxiliary costs. Our PPO version is based on Spinning Up in Deep RL [Achiam, 2018], which uses early stopping instead of other regularizers that typically appear in PPO implementations. For constrained algorithms, we evaluate

- **Lagrangian methods**: Lagrangian methods use adaptive penalty coefficients to enforce constraints. With $f(\theta)$ the objective and $g(\theta) \leq 0$ the constraint, Lagrangian methods solve the equivalent unconstrained max-min optimization problem

$$\max_{\theta} \min_{\lambda \geq 0} \mathcal{L}(\theta, \lambda) \doteq f(\theta) - \lambda g(\theta), \tag{7.2}$$

  by gradient ascent on θ and descent on λ. We combine the Lagrangian approach with TRPO and PPO to obtain TRPO-Lagrangian and PPO-Lagrangian.

- **Constrained Policy Optimization** [Achiam et al., 2017b]: CPO analytically solves trust region optimization problems at each policy update to enforce constraints throughout training. It is closely-connected to the θ-projection approach of Chow et al. [2019]. Unlike Achiam et al. [2017b], we omit the learned failure predictor they used for cost shaping.

SG18	Return $\bar{J}_r$	Violation $\bar{M}_c$	Cost Rate $\bar{\rho}_c$
PPO	1.0	1.0	1.0
PPO-Lagrangian	0.24	0.026	**0.245**
TRPO	**1.094**	1.132	1.004
TRPO-Lagrangian	0.331	**0.018**	0.265
CPO	0.784	0.593	0.646

Table 7.1: Normalized metrics from the conclusion of training averaged over the SG18 slate of environments and three random seeds per environment.

Hyperparameters: All experiments use separate feedforward MLP policy and value networks of size $(256, 256)$ with tanh activations. In all constrained cases, we set $d = 25$ for the expected cost limit. Experiments for Point and Car robots used batch sizes of $30,000$ environment interaction steps, and experiments for Doggo used $60,000$. Point and Car agents were trained for 10^7 steps, and Doggo agents were trained for 10^8 steps. All episodes are length $T_{\text{ep}} = 1000$, and so the value of cost rate corresponding to approximate constraint satisfaction throughout training is $\rho_c = d/T_{\text{ep}} = 0.025$.

We hand-tuned hyperparameters for each algorithm class to attain reasonable performance. However, we caution that our hand-tuning should not be viewed as indicative of the best-possible performance of each algorithm class.

All experiments were run with three random seeds.

7.4.3 Results

In Figures 7.7, 7.8, and 7.9, we show learning curves from evaluating unconstrained and constrained RL algorithms on the constrained Safety Gym environments. These learning curves depict the metrics $J_r(\theta)$, $J_c(\theta)$, and $\rho_c(\theta)$ *without* normalization, and show the absolute performance of each algorithm. In Tables 7.1 and 7.2, we report normalized metrics from the end of training averaged over various sets of environments. The normalized values allow easy comparison to a reference point (in this case, unconstrained PPO).

We observe a few general trends:

- Costs and rewards trade off against each other meaningfully. Unconstrained RL algorithms are able to score high returns by taking unsafe actions, as measured by the cost function. Constrained RL algorithms attain lower levels of return, and correspondingly maintain desired levels of costs.
- The design decision to make Level 2 Safety Gym environments denser in unsafe elements than Level 1 environments is reflected by the jump in average episodic cost for unconstrained agents.
- CPO fails to enforce the constraints in nearly every Safety Gym environment. Since the CPO procedure constrains a surrogate cost function that serves as an approximation of

the expected cost, and the experiment results indicate that the surrogate is correctly constrained, we surmise that the failure results from approximation errors in the surrogate. We additionally conclude that these environments are harder than ones where CPO has previously been tested. By contrast, Lagrangian methods more-or-less reliably enforce constraints, despite approximation errors. This contradicts the result from Achiam et al. [2017b].

- Lagrangian methods are able to find constraint-satisfying policies that attain nontrivial returns in several of the Point environments, demonstrating that when controlling for challenges in learning robot locomotion, it is possible to make progress on, or even solve these environments with constrained RL.
- Standard RL is able to control the Doggo robot and acquire complex locomotion behavior, as indicated by high returns in the environments when trained without constraints. However, despite the success of constrained RL when locomotion requirements are absent, and the success of standard RL when locomotion is needed, the constrained RL algorithms we investigated struggle to learn safe locomotion policies. Additional research is needed to develop constrained RL algorithms that can solve these challenging tasks.

SG1	$\bar{J}_r$	$\bar{M}_c$	$\bar{\rho}_c$
PPO	1.0	1.0	1.0
PPO-Lagrangian	0.354	0.034	**0.299**
TRPO	**1.127**	1.225	0.995
TRPO-Lagrangian	0.509	**0.024**	0.336
CPO	0.946	0.757	0.725

SGPoint	$\bar{J}_r$	$\bar{M}_c$	$\bar{\rho}_c$
PPO	1.0	1.0	1.0
PPO-Lagrangian	0.348	0.054	0.278
TRPO	**1.436**	0.975	0.967
TRPO-Lagrangian	0.565	**0.014**	**0.274**
CPO	1.005	0.353	0.514

SG2	$\bar{J}_r$	$\bar{M}_c$	$\bar{\rho}_c$
PPO	1.0	1.0	1.0
PPO-Lagrangian	0.126	0.019	**0.19**
TRPO	**1.061**	1.04	1.013
TRPO-Lagrangian	0.153	**0.013**	0.195
CPO	0.621	0.428	0.566

SGCar	$\bar{J}_r$	$\bar{M}_c$	$\bar{\rho}_c$
PPO	1.0	1.0	1.0
PPO-Lagrangian	0.373	**0.004**	**0.238**
TRPO	**1.158**	0.909	1.017
TRPO-Lagrangian	0.374	0.022	0.244
CPO	0.794	0.361	0.545

SG6	$\bar{J}_r$	$\bar{M}_c$	$\bar{\rho}_c$
PPO	1.0	1.0	1.0
PPO-Lagrangian	0.38	0.056	0.323
TRPO	**1.211**	1.03	1.003
TRPO-Lagrangian	0.565	**0.025**	**0.322**
CPO	1.021	0.573	0.677

SGDoggo	$\bar{J}_r$	$\bar{M}_c$	$\bar{\rho}_c$
PPO	**1.0**	1.0	1.0
PPO-Lagrangian	0.0	0.021	**0.218**
TRPO	0.688	1.513	1.029
TRPO-Lagrangian	0.054	**0.019**	0.277
CPO	0.552	1.065	0.878

Table 7.2: Normalized metrics from the conclusion of training averaged over various slates of environments and three random seeds per environment.

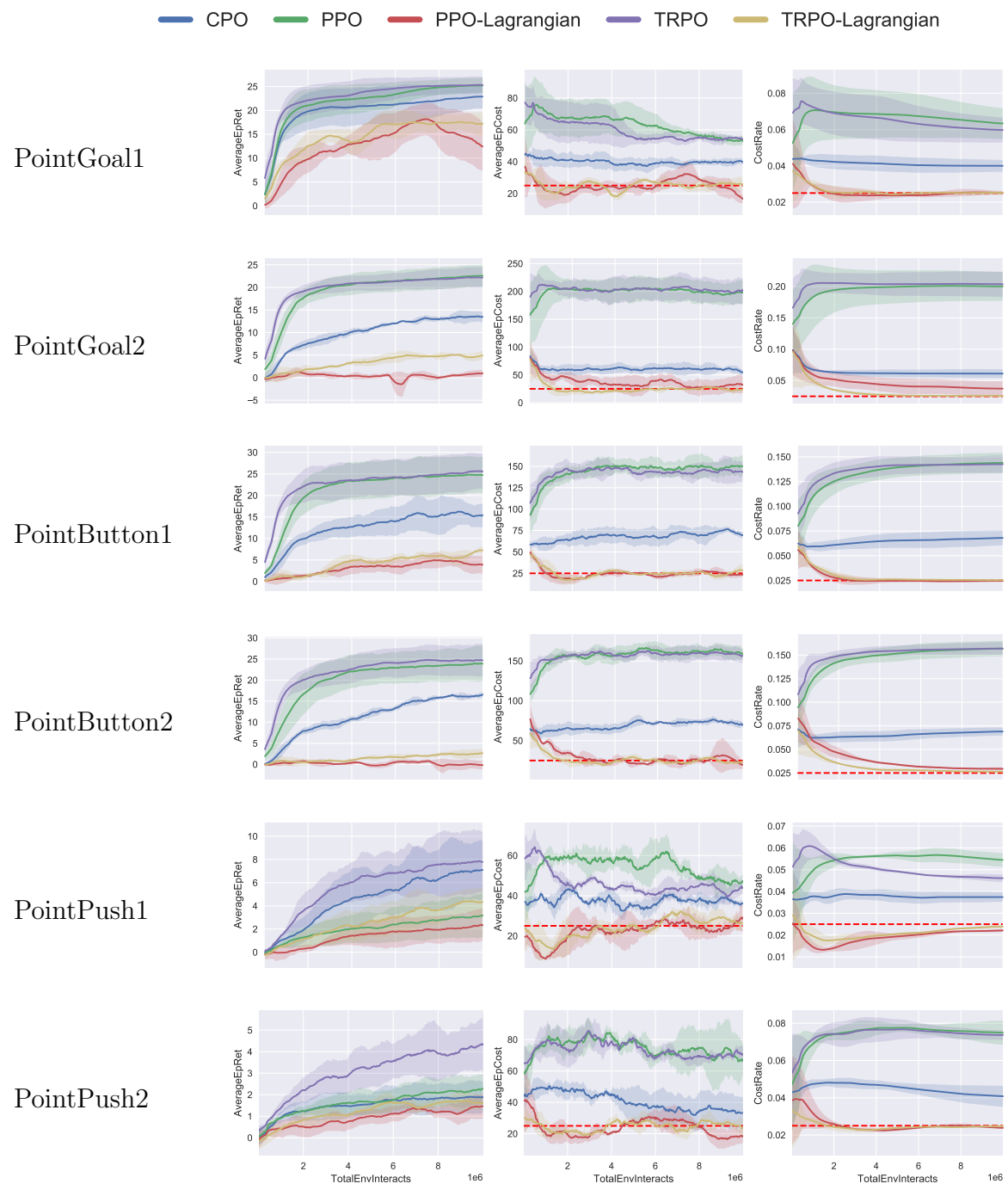


Figure 7.7: Results from benchmarking unconstrained and constrained RL algorithms on all Point level 1 and 2 environments. Dashed red lines indicate the target value for a constraint-satisfying policy (AverageEpCost curves) or approximately constraint-satisfying training run (CostRate curves).

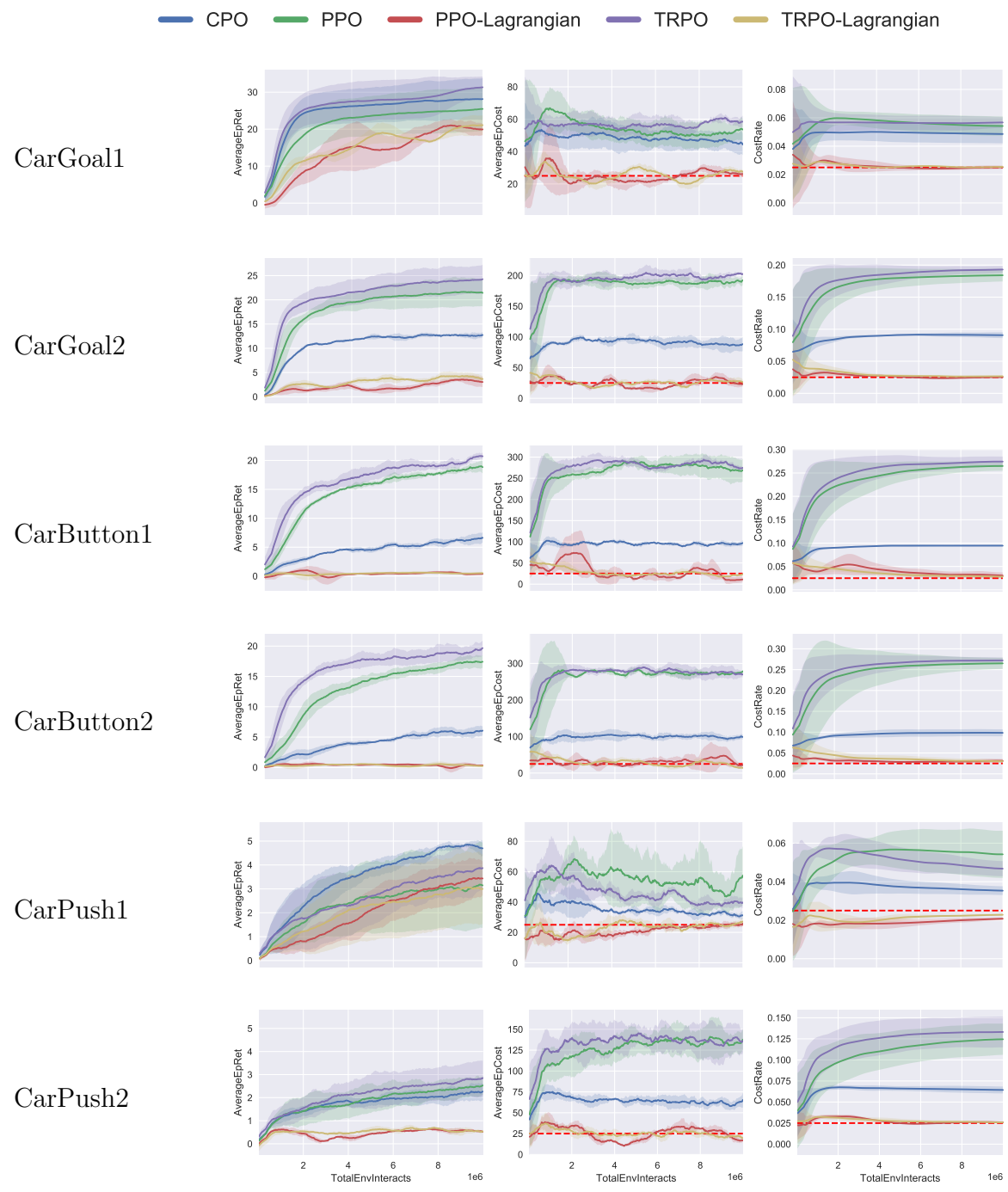


Figure 7.8: Results from benchmarking unconstrained and constrained RL algorithms on all Car level 1 and 2 environments. Dashed red lines indicate the target value for a constraint-satisfying policy (AverageEpCost curves) or approximately constraint-satisfying training run (CostRate curves).

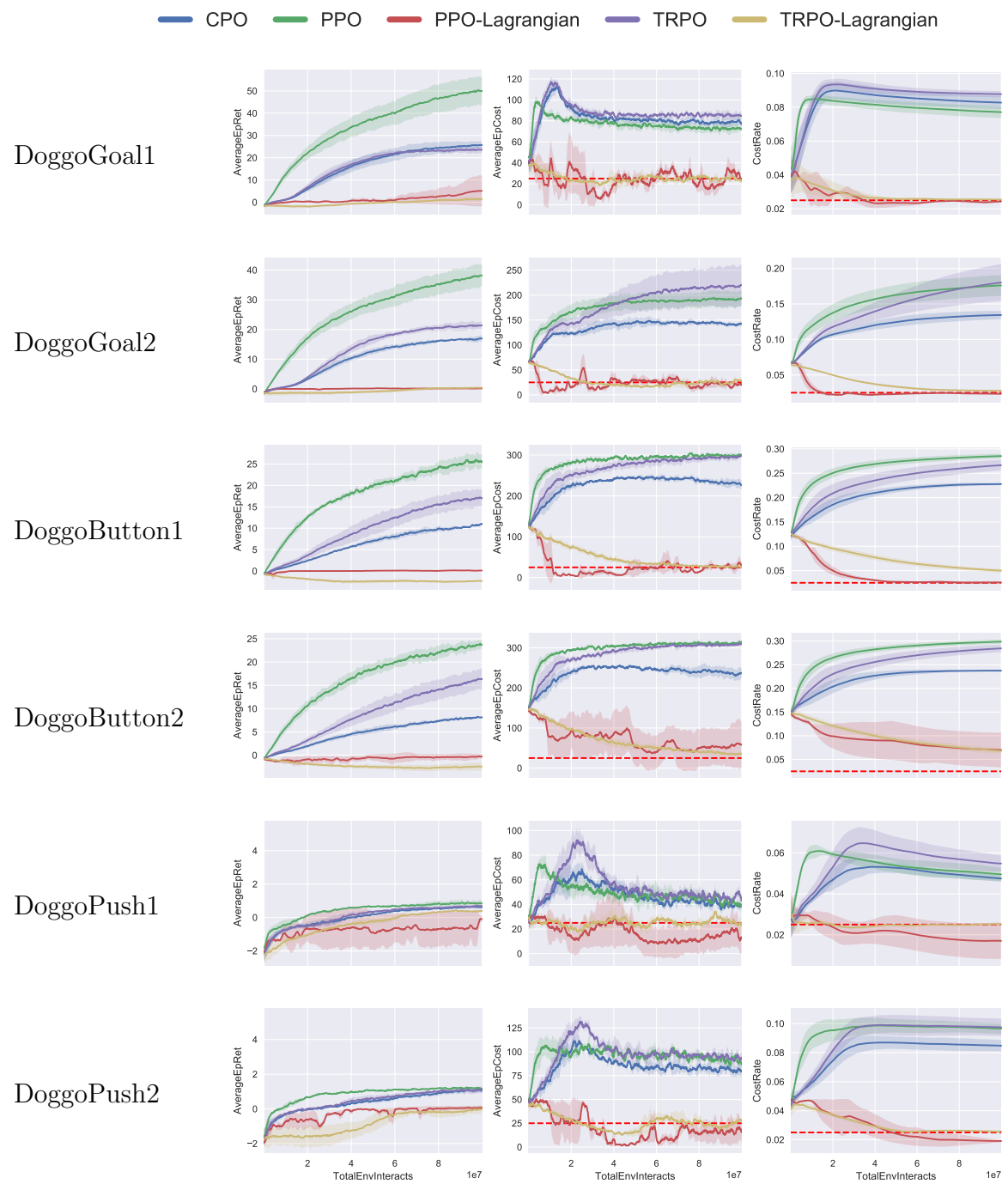


Figure 7.9: Results from benchmarking unconstrained and constrained RL algorithms on all Doggo level 1 and 2 environments. Dashed red lines indicate the target value for a constraint-satisfying policy (AverageEpCost curves) or approximately constraint-satisfying training run (CostRate curves).

7.5 Conclusions

In this chapter we introduced Safety Gym, the first benchmark of high-dimensional continuous control environments for evaluating the performance of constrained RL algorithms, and we evaluated baseline unconstrained and constrained RL algorithms on Safety Gym environments to partially clarify the current state of the art in safe exploration. There are a number of avenues we consider promising for future work.

Advancing SOTA on Safety Gym: Our baseline results for constrained RL indicate a need for stronger and/or better-tuned algorithms to succeed on Safety Gym environments. By success, we mean attaining improvements simultaneously along both the episodic return axis and the constraint regret axis, while still producing a constraint-satisfying policy at the conclusion of training. It is possible that existing techniques for constrained RL that were not explored in this work may make progress here, however, we expect that substantial and consistent performance gains will require new insights. We note that standard model-free RL approaches without replay buffers are fundamentally limited in their ability to minimize constraint regret: they must continually experience unsafe events in order to learn about them. As a result, we consider memory-based and model-based RL approaches to be particularly interesting here.

Safe Transfer Learning: We recommend the use of Safety Gym tools to investigate two problems related to safe transfer and distributional shift in the constrained RL setting.

- Problem 1: An agent is initially trained in one constrained RL environment, and then transferred to another environment where the task is the same but the safety requirements are different. In this setting, the safety concern is whether the agent can quickly adapt to the new safety requirements.
- Problem 2: An agent is initially trained in one constrained RL environment, and then transferred to another environment where the safety requirements are the same but the task is different. In this setting, the safety concern is whether the agent can remain constraint-satisfying despite the potential for catastrophic forgetting induced by the change in objective function.

Problem 1 can be investigated with unmodified Safety Gym benchmark environments, using the Level 1 and 2 versions of each task as (First Environment, Second Environment) pairs. New environments can easily be created for both problems using the Safety Gym `Engine` tool.

Constrained RL with Implicit Specifications: RL with implicitly-specified objectives is a research sub-field with important consequences for safety, encompassing inverse reinforcement learning, learning from human preferences, and other heuristics for extracting value-aligned objectives from human data. These techniques are complementary to and compatible with constrained RL, and thus we recommend research in the direction of combining them. Safety Gym benchmark environments can be used to study whether such combination techniques are efficient at training agents to satisfy implicitly-specified safety requirements.

Chapter 8

PID Lagrangian Methods

Lagrangian methods are widely-used algorithms for constrained optimization problems, and due to their simplicity and empirically good performance, they are an appealing choice for safe reinforcement learning. However, their learning dynamics exhibit oscillations and overshoots that can lead to constraint-violating behavior during agent training. We address this shortcoming by proposing a novel Lagrange multiplier update method based on taking a controls perspective. We observe that the traditional Lagrange multiplier update behaves as *integral* control; our method adds *proportional* and *derivative* control, achieving favorable learning dynamics through damping and predictive measures. Our extensive experiments in Safety Gym environments demonstrate that this approach improves the performance and hyperparameter robustness of Lagrangian methods, while remaining nearly as simple to derive and implement.

8.1 Introduction

Lagrangian primal-dual methods are a classic approach to solving constrained optimization problems. In the Lagrangian approach, a constrained minimization problem is transformed into an equivalent min-max problem where the constraint functions have been added to the objective function and weighted by dual variables called Lagrange multipliers. The augmented objective function is called the *Lagrangian*. To illustrate, the equality-constrained problem over the real vector x:

$$\min_x f(x) \qquad \text{s.t. } g(x) = 0 \tag{8.1}$$

is transformed by the introduction of the dual variable λ into:

$$(x^*, \lambda^*) = \arg\min_x \max_\lambda \mathcal{L}(x, \lambda), \tag{8.2}$$

with Lagrangian $\mathcal{L}(x, \lambda)$ given by:

$$\mathcal{L}(x, \lambda) = f(x) + \lambda g(x).$$

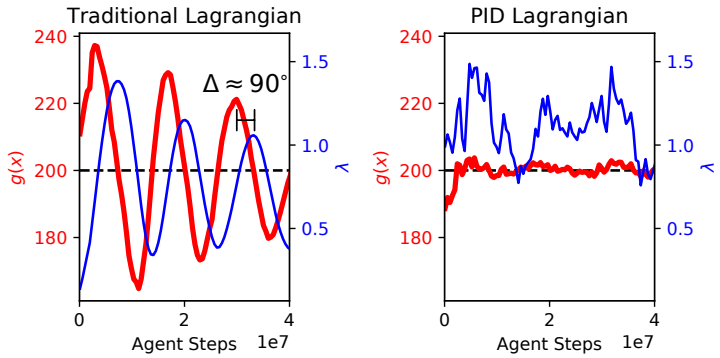


Figure 8.1: *Left*: The traditional Lagrangian method exhibits oscillations with 90° phase shift between the constraint function and the Lagrange multiplier, characteristic of integral control. *Right*: PID control on the Lagrange multiplier damps oscillations and obeys constraints. Environment: DoggoButton1, cost limit 200.

Gradient-based Lagrangian primal-dual algorithms iteratively update the primal variables x and dual variables λ:

$$x \leftarrow x - \eta \nabla_x \mathcal{L}(x, \lambda) = x - \eta \left(\nabla_x f(x) + \lambda \nabla_x g(x) \right) \tag{8.3}$$

$$\lambda \leftarrow \lambda + \beta \nabla_\lambda \mathcal{L}(x, \lambda) = \lambda + \beta g(x) \tag{8.4}$$

so that λ acts as a learned penalty coefficient in the objective. Under a variety of technical conditions, this leads to a constraint-satisfying solution at convergence (see e.g. Bertsekas [2014]). This approach is readily adapted to the constrained RL setting [Altman, 1998, Geibel and Wysotzki, 2011] and has become a popular baseline in constrained deep RL [Achiam et al., 2017b, Chow et al., 2019] for its simplicity and effectiveness.

Although they have been shown to converge to optimal, constraint-satisfying policies [Chow et al., 2015, Tessler et al., 2018, Paternain et al., 2019], a shortcoming of gradient Lagrangian methods for safe RL is that intermediate iterates often violate constraints. Cost overshoot and oscillations are in fact inherent to the learning dynamics [Platt and Barr, 1988, Wah et al., 2000], and we witnessed numerous problematic cases in our own experiments. Figure 8.1 (*left*) shows an example from a deep RL setting, where the cost and multiplier values oscillated throughout training. Our key insight in relation to this deficiency is that the traditional Lagrange multiplier update in (8.4) amounts to *integral* control on the constraint. The 90-degree phase shift between the curves is characteristic of ill-tuned integral controllers.

Our contribution is to expand the scope of possible Lagrange multiplier update rules beyond (8.4), by interpreting the overall learning algorithm as a dynamical system. Specifically, we employ the next simplest mechanisms, *proportional* and *derivative* control, to λ, by adding terms corresponding to derivatives of the constraint function into (8.4) (derivatives with respect to learning iteration). To our knowledge, this is the first time that this expanded update rule has been considered for a learned Lagrange multiplier. PID control is an appealing enhancement, evidenced by the fact that it is one of the most widely used and

studied control techniques [Åström and Hägglund, 2006]. The result is a more responsive safety mechanism, as demonstrated in Figure 8.1 (*right*), where the cost oscillations have been damped, dramatically reducing violations.

The chapter is outlined as follows. First, we provide further context through related works and preliminary definitions. In Section 8.4, we propose our modified PID Lagrangian multiplier methods. Next, in Section 8.5, we cast constrained RL as a dynamical system with the Lagrange multiplier as a control input, to which we apply PID control as a new algorithm. In Section 8.6, we adapt a leading deep RL algorithm, Proximal Policy Optimization (PPO) [Schulman et al., 2017] with our methods and achieve state of the art performance in the OpenAI Safety-Gym suite of environments [Ray et al., 2019]. Finally, in Section 8.7 we introduce another novel technique that makes tuning easier by providing invariance to the relative numerical scales of rewards and costs, and we demonstrate it in a further set of experiments. Our extensive empirical results show that our algorithms, which are intuitive and simple to implement, improve cost performance and promote hyperparameter robustness in a deep RL setting.

8.2 Related Work

Constrained Deep RL. Adaptations of the Lagrange multiplier method to the actor-critic RL setting have been shown to converge to the optimal, constraint-satisfying solution under certain assumptions [Chow et al., 2015, Tessler et al., 2018, Paternain et al., 2019]. Convergence proofs have relied upon updating the multiplier more slowly than the policy parameters, implying many constraint-violating policy iterations may occur before the penalty comes into full effect.

Several recent works have aimed at improving constraint satisfaction in RL over the Lagrangian method, but they tend to incur added complexity. Achiam et al. [2017b] introduced Constrained Policy Optimization (CPO), a policy search algorithm with near-constraint satisfaction guarantees at every iteration, based on a new bound on the expected returns of two nearby policies. CPO includes a projection step on the policy parameters, which in practice requires a time-consuming backtracking line search. Yet, simple Lagrangian-based algorithms performed as well or better in a recent empirical comparison in Safety Gym Ray et al. [2019], and in some cases approximation error in CPO resulted in a failure to enforce constraints. Approaches to safe RL based on Lyapunov functions have been developed in a series of studies Chow et al. [2018, 2019], resulting in algorithms that combine a projection step, as in CPO, with action-layer interventions like the safety layer of Dalal et al. [2018]. Experimentally, this line of work showed mixed performance gains over Lagrangian methods, at a nontrivial cost to implement and without clear guidance for tuning. Liu et al. [2019] developed interior point methods for RL, which augment the objective with logarithmic barrier functions. These methods are shown theoretically to provide suboptimal solutions. Furthermore, they require tuning of the barrier strength and typically assume already feasible iterates, the latter point possibly being problematic for random agent initializations or under noisy cost estimates. Most recently, Yang et al. [2020] extended CPO with a two-step

projection-based optimization approach. In contrast to these techniques, our method remains nearly as simple to implement and compute as the baseline Lagrangian method.

Dynamical Systems View of Optimization. Several recent works have proposed different dynamical systems viewpoints to analyze optimization algorithms, including those often applied to deep learning. Hu and Lessard [2017] reinterpreted first-order gradient optimization as a dynamical system; they likened the gradient of the objective, $\nabla_x f$, to the plant, which the controller aims to drive to zero to arrive at the optimal parameters, x^*. Basic gradient descent then matches the form of integral control (on $\nabla_x f$). They extend the analogy to momentum-based methods, for example linking Nesterov momentum to PID control with lag compensation. In another example, An et al. [2018] interpreted SGD as P-control and momentum methods as PI-control. They introduced a derivative term, based on the change in the gradient, and applied their resulting PID controller to improve optimization of deep convolutional networks. Other recent works bring yet other perspectives from dynamical systems to deep learning and optimization, see for example Lessard et al. [2014], Nishihara et al. [2015], Liu and Theodorou [2019]. None of these works address constrained RL, however, necessitating our distinct formulation for that problem.

Constrained Optimization. Decades' worth of literature have accumulated on Lagrangian methods. But even recent textbooks on the topic Bertsekas [2014], Nocedal and Wright [2006] do not appear to describe the modified learning rule for the Lagrange multiplier that we propose. The modification to the Lagrangian method most similar in effect to our proportional control term is the quadratic penalty method (Hestenes [1969], Powell [1969] see also *e.g.* Bertsekas [1976]), which we compare in Section 8.4. Song and Leland [1998] proposed a controls viewpoint (continuous-time) of optimizing neural networks for constrained problems and arrived at proportional control rules only. Related to our final experiments on reward-scale invariance, Wah et al. [2000] developed an adaptive weighting scheme for continuous-time Lagrangian objectives, but it is an intricate procedure which is not straightforwardly applied to safe RL.

8.3 Preliminaries

A general formulation for discrete-time dynamical systems with feedback control is:

$$\begin{aligned} x_{k+1} =& F(x_k, u_k) \\ y_k =& g(x_k) \\ u_k =& h(y_0, ..., y_k) \end{aligned} \tag{8.5}$$

where x is the state vector, F is the dynamics function, g is the measurement output function, y is a measurement output vector, u is a control, and the subscript denotes the time step. A problem in optimal control is to design a control rule, h, that results in a sequence of outputs $y_{0:T} \doteq \{y_0, ..., y_T\}$ that scores well according to some cost function C. Cost functions might incentivize reaching a goal measurement, $C(y_{0:T}) = |y_T - \bar{y}|$, or closely following a desired trajectory, $C(y_{0:T}) = \sum_{t=0}^{T} |y_t - \overline{y}_t|$.

Systems with simpler dependence on the input are generally easier to analyze and control—that is, simpler h performs well—even if the dependence on the state is complicated [Skelton, 1988]. Control-affine systems are a broad class of dynamical systems which are especially amenable to analysis [Isidori et al., 1995]. They take the form:

$$F(x_k, u_k) = A(x_k) + B(x_k)u_k \tag{8.6}$$

where A and B may be nonlinear in state, and are possibly unknown.

8.4 PID Lagrangian Methods

In this section, we propose an intuitive, previously overlooked adjustment to the Lagrangian method based on PID control. To motivate our approach, we will begin by mapping the Lagrangian method to the framework of dynamical systems with feedback control. The Lagrangian parameter update (8.3) is exactly a control-affine dynamical system, with state x, control λ, and:

$$\begin{aligned} F(x, \lambda) &= A(x) + B(x)\lambda \\ A(x) &= x - \eta \nabla_x f(x) \\ B(x) &= -\eta \nabla_x g(x). \end{aligned}$$

The constraint function is the measurement output function, and the control law h is given by (8.4). Now, consider the continuous time forms of the Lagrangian parameter updates (8.3) and (8.4):

$$\dot{x} = -\nabla_x \mathcal{L}(x, \lambda) = -\nabla_x f - \lambda \nabla_x g \tag{8.7}$$

$$\dot{\lambda} = K_I \nabla_\lambda \mathcal{L}(x, \lambda) = K_I g(x). \tag{8.8}$$

Note that η, previously the learning rate for x, has been folded into the parameterization of time, and the λ learning rate has been suggestively renamed K_I. From here, we calculate $\lambda(t)$ as:

$$\lambda(t) = \lambda(0) + K_I \int_0^t g(x(t'))dt'. \tag{8.9}$$

The intepretation of the Lagrangian penalty update as integral control is now obvious. We are trying to drive the constraint function, $g(x(t))$, to a setpoint of zero. λ is a control in the state dynamics $\dot{x}(x, \lambda)$. $\lambda(t)$ is the integral of the error signal over time plus a constant, and K_I is the integral control gain.

We extend (8.9) to full PID control by adding proportional and derivative terms:

$$\lambda(t) = \lambda(0) + K_I \int_0^t g(t')dt' + K_P g(t) + K_D \dot{g}(t), \tag{8.10}$$

where we have written $g(x(t))$ as $g(t)$ for clarity.

8.4.1 Quadratic Penalty Method

Here we compare our approach with the well-known quadratic penalty method Hestenes [1969], Powell [1969], which augments the Lagrangian with an additional term quadratic in the constraint function. Under the augmented Lagrangian $\mathcal{L}' = \mathcal{L} + (K_P/2)g(x)^2$, the learning dynamics are:

$$\begin{aligned}\dot{x} &= -\nabla_x \mathcal{L}'(x,\lambda) = -\nabla_x f - (\lambda + K_P g(x))\,\nabla_x g \\ \dot{\lambda} &= K_I \nabla_\lambda \mathcal{L}'(x,\lambda) = K_I g(x),\end{aligned}$$

or equivalently,

$$\begin{aligned}\dot{x} &= -\nabla_x f - \lambda_{pi} \nabla_x g \\ \lambda_{pi}(t) &= \lambda(0) + K_I \int_0^t g(t')dt' + K_P g(t).\end{aligned}$$

So the quadratic penalty method used with the standard gradient based Lagrangian method yields a PI control rule for the Lagrange multiplier. By starting with a different motivation, we developed an approach that goes further to also allow for derivative control.

8.5 Feedback Control for Constrained RL

In constrained RL, satisfying constraints throughout exploration, rather than just at convergence, is sometimes critical. For example, if the expected cost threshold is designed to represent a margin of failure from a catastrophic event that cannot happen even once, eventual convergence to a constraint-satisfying policy is not sufficient—constraints must be satisfied at all times. In these settings, the Lagrangian method would be disqualified from suitability due to its characteristic oscillations. Here, we apply our insights from the previous section: by recasting the Lagrangian method in constrained RL as a control problem, we will create a PID control-based approach to adjusting penalty coefficients that damps oscillations in cost throughout training. As in preceding chapters, we will focus on constrained RL with a single cost constraint:

$$\max_\theta J(\pi_\theta) \;:\; J_C(\pi_\theta) \leq d.$$

8.5.1 Constrained RL as a Dynamical System

We write constrained RL as the first-order dynamical system:

$$\begin{aligned}\theta_{k+1} &= F(\theta_k, \lambda_k) \\ y_k &= J_C(\pi_{\theta_k}) \\ \lambda_k &= h(y_0, ..., y_k, d)\end{aligned} \tag{8.11}$$

where F is the RL algorithm policy update on the agent's parameter vector, θ. The cost-objective serves as the system measure, y, which is supplied to the feedback control rule,

h, along with cost limit, d. The role of the controller is to assure the satisfaction of the inequality constraint, $J_C(\pi_{\theta_k}) \leq d$, at each iteration k.

The policy update from the Lagrangian method with a single gradient ascent step at learning rate η has state dynamics:

$$F(\theta, \lambda) = \theta + \eta \left(\nabla_\theta J(\pi_\theta) - \lambda \nabla_\theta J_C(\pi_\theta) \right).$$

We will make two adjustments. First, we will generalize this to an arbitrary maximization procedure using surrogate objectives $J \to \hat{L}$ and $J_C \to \hat{L}_C$ based on sample data, since that is what we do in practice. (For example, the PPO-Lagrangian approach involves multiple steps of gradient ascent on the policy per update, using sample-based surrogate functions for the expected return and cost.) Second, we will include a rescale factor of $1/(1+\lambda)$ that does not change the optimal point but is considered to help with the numerical stability of practical implementations based on gradient descent. This ensures that the gradient in each step is a balance between the reward and cost gradient, avoiding a circumstance where, when the magnitude of λ is high, learning is destabilized by large steps on the policy.

With these two changes, we have:

$$F(\theta, \lambda) = \arg\max_\theta \frac{1}{1+\lambda} \left(\hat{L}(\pi_\theta) - \lambda \hat{L}_C(\pi_\theta) \right).$$

As a final implementation detail: surrogates $\hat{L}$ and $\hat{L}_C$ typically require advantage estimation based on learned value functions. Some Lagrangian-based constrained RL algorithms [Tessler et al., 2018] use a single value function to estimate the cumulative $r + \lambda c$, but when λ changes rapidly, the learned value function may not "catch up" quickly enough to λ. As a result, we will develop implementations that use separate approximators V_ϕ and $V_{C,\psi}$ for the reward and cost value functions.

We give the template for this generic constraint-controlled RL algorithm, with everything specified except for the control rule for λ, as Algorithm 5.

8.5.2 The PID Lagrangian Method

The standard Lagrange multiplier update rule for an inequality constraint uses subgradient descent:

$$\lambda_{k+1} = (\lambda_k + K_I (J_C(\pi_{\theta_k}) - d))_+, \tag{8.12}$$

with learning rate K_I and projection into $\lambda \geq 0$ (notation: $f(u) = (u)_+ = \max(0, u)$). As before, we interpret this choice of update rule as integral control.

We now specify a new control rule for use in Algorithm 5. As discussed earlier, the traditional Lagrange multiplier update is prone to cost oscillations and overshoot, which we observed in deep RL settings, for example in Figure 8.1. The 90 degree phase shift between penalty coefficient and cost curves is characteristic of ill-tuned integral controllers. To overcome the shortcomings of integral-only control, we follow the developments of the previous section and

Algorithm 5
Constraint-Controlled Reinforcement Learning

Initialize control rule (as needed)
$\mathcal{J}_C \leftarrow \{\}$ {cost measurement history}
repeat
  Collect a minibatch by sampling from environment. At each state s:
    $a \sim \pi(\cdot|s;\theta), s' \sim P(\cdot|s,a)$,
    $r = R(s,a,s'), c = C(s,a,s')$
  Apply feedback control:
    Store sample estimate $\hat{J}_C$ into $\mathcal{J}_C$
    $\lambda \leftarrow h(\mathcal{J}_C, d),\ \lambda \geq 0$
  Update π by RL using the Lagrangian objective:
    Update critics, $V_\phi(s), V_{C,\psi}(s)$ {if using}
    Take one or more gradient ascent steps on θ with

$$\nabla_\theta \mathcal{L} = \frac{1}{1+\lambda}\left(\nabla_\theta \hat{L}(\pi_\theta) - \lambda \nabla_\theta \hat{L}_C(\pi_\theta)\right)$$

until converged
return π_θ

introduce the next simplest components: *proportional* and *derivative* terms. A simplified form of our PID update rule to replace (8.12) is shown in Algorithm 6. The proportional term will hasten the response to constraint violations and dampen oscillations, as derived in Section 8.4. Unlike the Lagrangian update, derivative control can act in anticipation of violations. It can both prevent cost overshoot and limit the rate of cost increases within the feasible region, useful when monitoring a system for further safety interventions. Our derivative term is projected as $(\cdot)_+$ so that it acts against increases in cost but does not impede decreases. Overall, PID control provides a much richer set of controllers while remaining nearly as simple to implement; setting $K_P = K_D = 0$ recovers the traditional Lagrangian method. The integral term remains necessary for eliminating steady-state violations at convergence.

We make a few small implementation-level decisions not shown in Algorithm 6. Expected cost measurements in constrained RL tend to be highly-noisy, even for small changes in the policy; to prevent propagating the noise into λ, we smooth the proportional term by using an exponentially-weighted moving average of Δ instead of directly using the most recent Δ. Likewise, we smooth the estimate of J_C used in the derivative term the same way. Finally, we use a multi-step delay for $J_{C,prev}$ instead of a single step.

Algorithm 6 PID-Controlled Lagrange Multiplier

Choose tuning parameters: $K_P, K_I, K_D \geq 0$
Integral: $I \leftarrow 0$
Previous Cost: $J_{C,prev} \leftarrow 0$
for each iteration k **do**
  Receive cost J_C
  $\Delta \leftarrow J_C - d$
  $\partial \leftarrow (J_C - J_{C,prev})_+$
  $I \leftarrow (I + \Delta)_+$
  $\lambda \leftarrow (K_P\Delta + K_I I + K_D\partial)_+$
  $J_{C,prev} \leftarrow J_C$
  return λ
end for

8.6 PID Control Experiments

We investigated the performance of our algorithms for constrained RL in a deep RL setting. In our experiments, we show the effectiveness of PID control at reducing constraint violations from oscillations and overshoot present in the baseline Lagrangian method. Both maximum performance and robustness to hyperparameter selection are considered. Although many methods exist for tuning PID parameters, we elected to do so manually, demonstrating ease of use.

8.6.1 Environments: Safety-Gym

We use the recent Safety-Gym suite Ray et al. [2019], which consists of robot locomotion tasks built on the MuJoCo simulator Todorov et al. [2012]. The robots range in complexity from a simple Point robot to the 12-jointed Doggo, and they move in an open arena floor. Rewards have a small, dense component encouraging movement toward the goal, and a large, sparse component for achieving it. When a goal is achieved, a new goal location is randomly generated, and the episode continues until the time limit at 1,000 steps.

Each task has multiple difficulty levels corresponding to density and type of hazards, which induce a cost when contacted by the robot (without necessarily hindering its movement). Hazards are placed randomly at each episode and often lay in the path to the goal. Hence the aims of achieving high rewards and low costs are in opposition. The robot senses the position of hazards and the goal through a coarse, LIDAR-like mode. The output of this sensor, along with internal readings like the joint positions and velocities, comprises the state fed to the agent. Figure 8.2 displays a scene from the DoggoGoal1 environment.

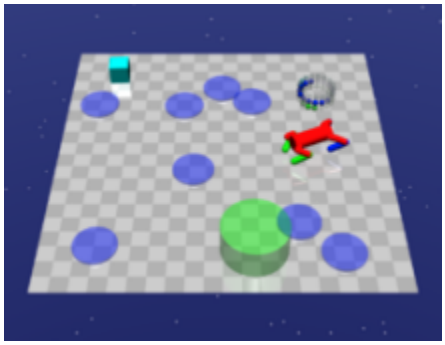


Figure 8.2: Rendering from the DoggoGoal1 environment from Safety Gym. The red, four-legged robot must walk to the green cylinder while avoiding other objects, and receives coarse egocentric sensor readings of their locations.

8.6.2 Algorithm: Constraint-Controlled PPO

We implemented Algorithm 5 on top of Proximal Policy Optimization (PPO) Schulman et al. [2017] to make constraint-controlled PPO (CPPO). CPPO uses an analogous clipped surrogate objective for the cost as for the reward. Our policy is a 2-layer MLP followed by an LSTM with a skip connection. We applied smoothing to proportional and derivative controls to accommodate noisy estimates. The environments' finite horizons allowed use of non-discounted episodic costs as the constraint and input to the controller. Additional training details can be found in supplementary materials, and our implementation is available at `https://github.com/astooke/rlpyt/rlpyt/projects/safe`.

8.6.3 Main Results

We compare PID controller performance against the Lagrangian baseline under a wide range of settings. Plots showing the performance of unconstrained RL confirm that constraints are not trivially satisfied, and they appear in supplementary material.

Robust Safety with PI Control

We observed cost oscillations or overshoot with slow settling time in a majority of Safety Gym environments when using the baseline Lagrangian method. Figure 8.3 shows an example where PI-control eliminated this behavior while maintaining good reward performance, in the challenging DoggoButton1 environment. Individual runs are plotted for different cost limits.

As predicted in Platt and Barr [1988], we found the severity of cost overshoot and oscillations to depend on the penalty coefficient learning rate, K_I. The top left panel of Figure 8.4 shows example cost curves from DoggoGoal2 under I-control, over a wide range of values for K_I. With increasing K_I, the period and amplitude of cost oscillations decrease and

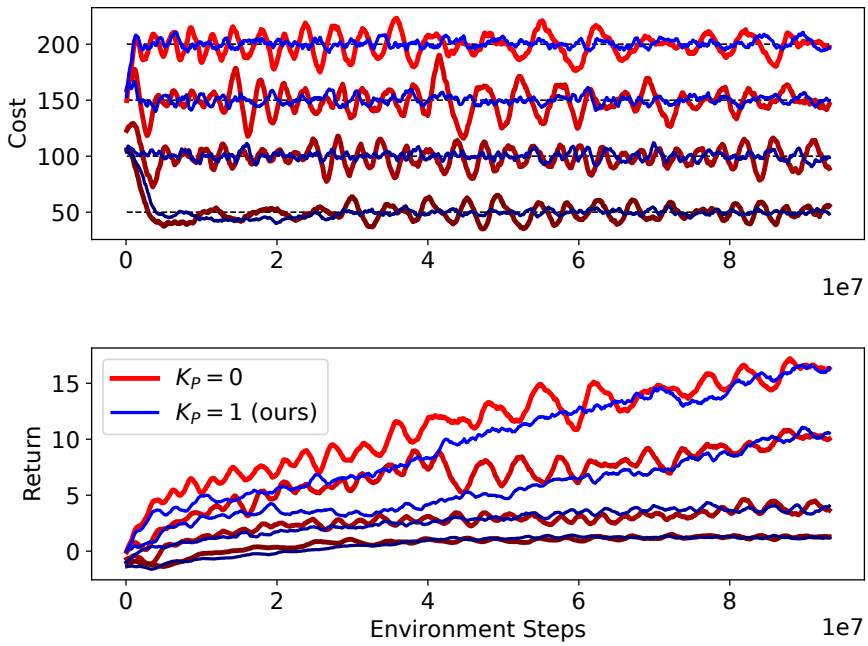


Figure 8.3: Oscillations in episodic costs (and returns) from the Lagrangian method, $K_P = 0, K_I = 10^{-2}$, are damped by proportional control, $K_P = 1$ (ours), at cost limits $50, 100, 150, 200$ (curves shaded) in DoggoButton1.

eventually disappear. The bottom left of Figure 8.4, however, shows that larger K_I also brings diminishing returns. We study this effect in the next section. The center and right columns of Figure 8.4 show the cost and return when using PI-control, with $K_P = 0.25$ and $K_P = 1$, respectively. Proportional control stabilized the cost, with most oscillations reduced to the noise floor for $K_I > 10^{-4}$. Yet returns remained relatively high over a wide range, $K_I < 10^{-1}$. Similar curves for other Safety Gym environments are included in an appendix.

We examine the trade-off between reward and constraint violation by forming an overall cost figure of merit (FOM). We use the sum of non-discounted constraint violations over the learning iterates, $C_{FOM} = \sum_k (J_C(\pi_{\theta_k}) - d)_+$, with $J_C(\pi_\theta) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T} C(s_t, a_t, s_{t+1})\right]$, and estimate it online from the learning data. Figure 8.5 compares final returns against this cost FOM for the same set of experiments as in Figure 8.4. Each point represents a different setting of K_I, averaged over four runs. PI-control expanded the Pareto frontier of this trade-off into a new region of high rewards at relatively low cost which was inaccessible using the baseline Lagrangian method ($K_P = 0$). These results constitute a new state of the art over the benchmarks in Ray et al. [2019].

We performed similar experiments on several Safety Gym environments in addition to DoggoGoal2: PointGoal1, the simplest domain with a point-like robot, CarButton1, for slightly more challenging locomotive control, and DoggoButton1 for another challenging

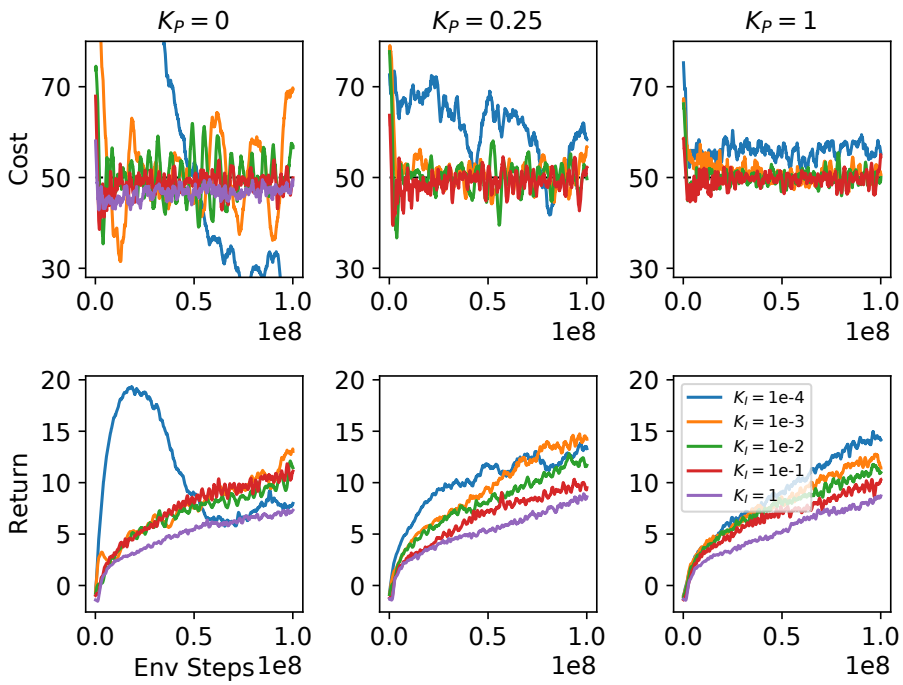


Figure 8.4: *Top row*: Constraint-violating oscillations decrease in magnitude and period from increases in the Lagrange multiplier learning rate, K_I. At all levels, oscillations are damped by PI-control, $K_P = 0.25, 1$. *Bottom row*: Returns diminish for large K_I; proportional control maintains high returns while reducing constraint violations. Environment: DoggoGoal2, cost limit 50.

task (see appendix for learning curves like Figure 8.4). Figure 8.6 plots the cost figure of merit over the same range of values for K_I, and for two strengths of added proportional control, for these environments. PI-control clearly improved the cost FOM (lower is better) for $K_I < 10^{-1}$, above which the fast integral control dominated. Hence robustness to the value for K_I was significantly improved in all the learning tasks studied.

Control Efficiency

We further investigated why increasing the penalty learning rate, K_I, eventually reduces reward performance, as was seen in the robustness study. Figure 8.7 shows learning curves for three settings: I- and PI-control with the same, moderate $K_I = 10^{-3}$, and I-control with high $K_I = 10^{-1}$. The high-K_I setting achieved responsive cost performance but lower long-term returns, which appears to result from wildly fluctuating control. In contrast, PI-control held relatively steady, despite the noise, allowing the agent to do reward-learning at every iteration. The bottom panel displays individual control iterates, here displayed as $u = \lambda/(1 + \lambda)$, over the first 7M environment steps, while the others show smoothed curves over the entire learning run, over 40M steps.

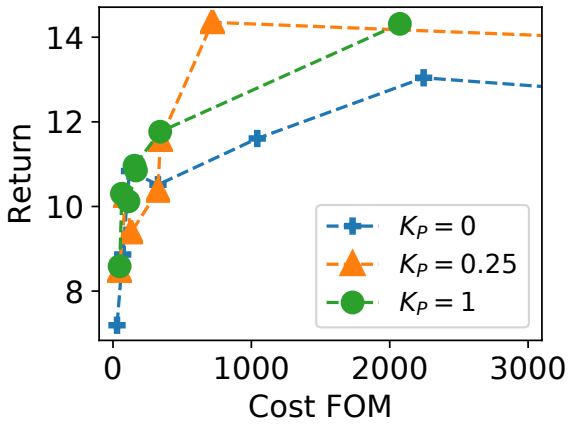


Figure 8.5: Pareto frontier of return versus cost FOM, which improves (up and to the left) with PI-control, $K_P = 0.25, 1$. Each point is a different setting of K_I (see Figure 8.4).

Predictive Control by Derivatives

Figure 8.8 demonstrates the predictive capabilities of derivative cost control in a noisy deep RL setting. It removed cost overshoot from both the I- and PI-controlled baselines. It was further able to slow the approach of the cost curve towards the limit, a desirable behavior for online learning systems requiring safety monitoring. Curves for other environments are available in an appendix.

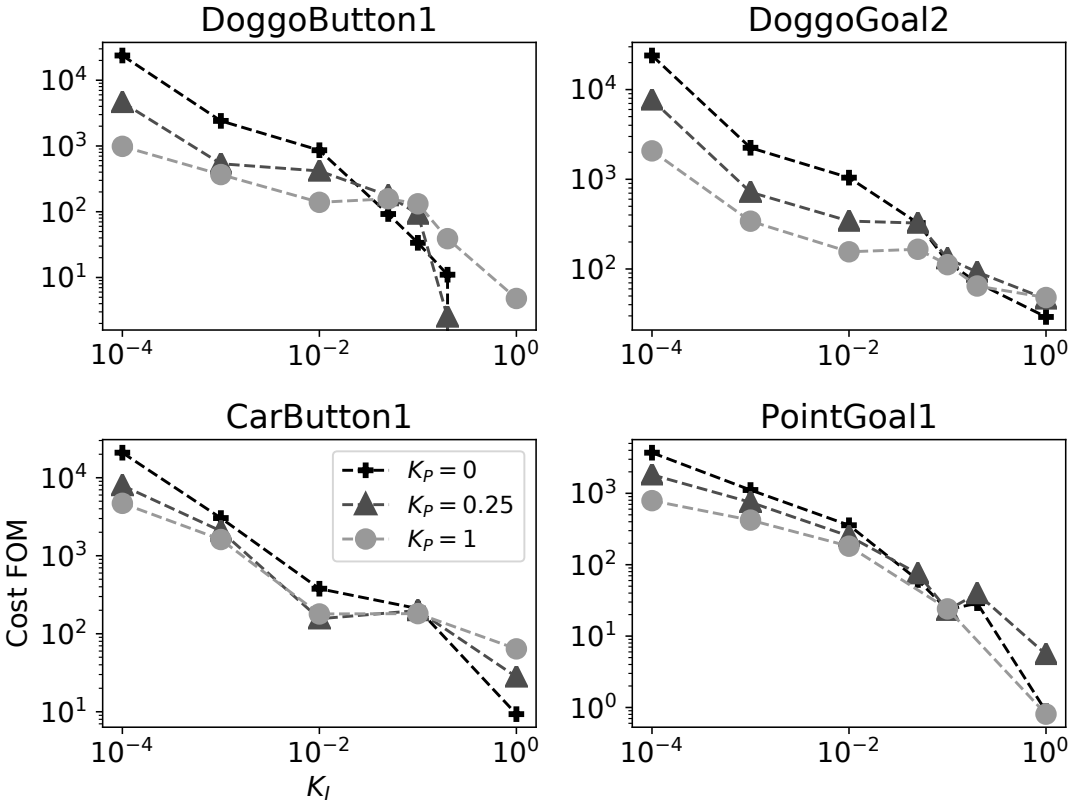


Figure 8.6: Learning run cost FOM versus penalty learning rate, K_I, from four environments spanning the robots in Safety Gym. Each point is an average over four runs. In all cases, PI-control improves performance (lower is better) over a wide and useful range of K_I, easing selection of that hyperparameter.

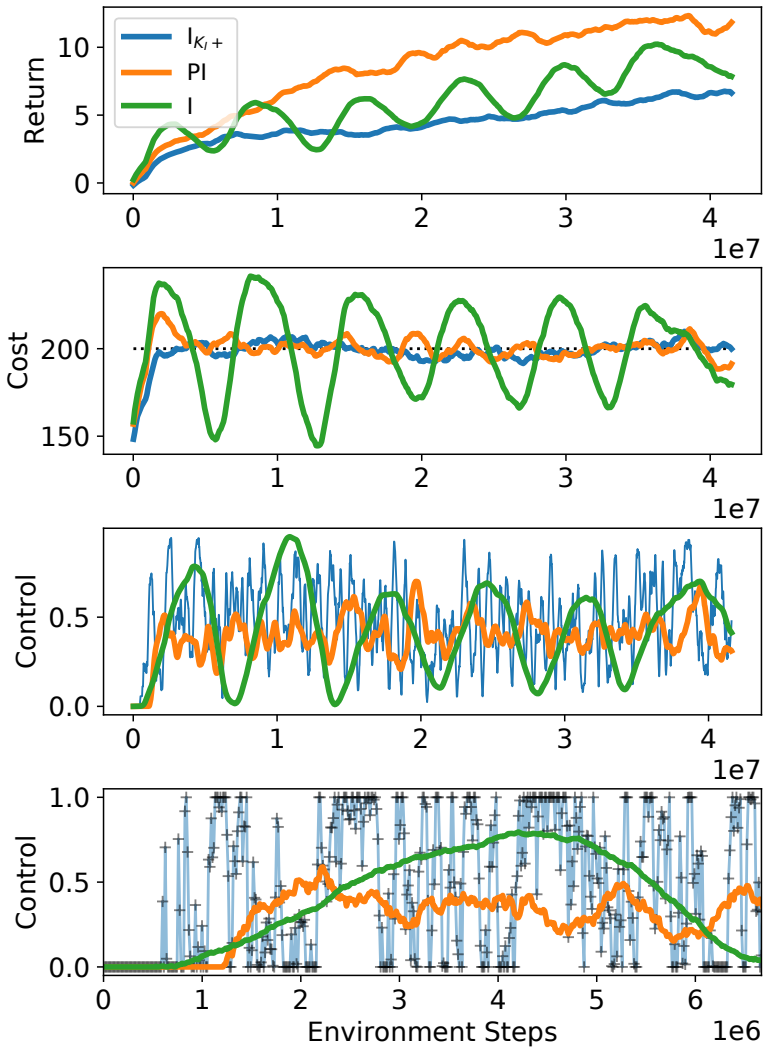


Figure 8.7: I- and PI-control with moderate $K_I = 10^{-3}$ and I-control with fast $K_I = 10^{-1}$ (I_{K_I+}). *Top* Returns diminished for fast-K_I, but high for PI. *Second* Cost oscillations mostly damped by PI, removed by fast-K_I. *Third* Control (smoothed) varies more rapidly under fast-K_I, is relatively steady for PI. *Bottom* Control over first 500 RL iterations; fast-K_I slams the control to the extremes, causing the diminished returns. Environment: DoggoButton1, cost limit 200.

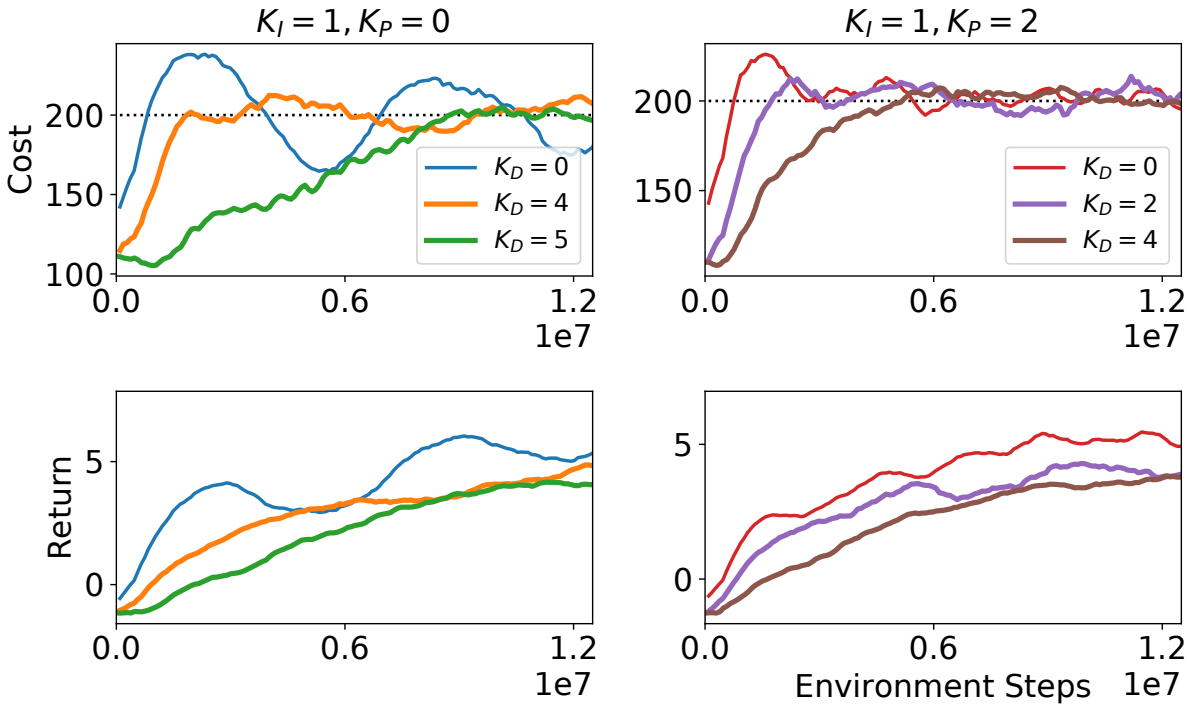


Figure 8.8: Derivative control can prevent cost overshoot and slow the rate of cost increase within feasible regions, which the Lagrangian method cannot do. Environment: DoggoButton1, cost limit 200.

8.7 Reward-Scale Invariance

In the preceding sections, we showed that PID control improves hyperparameter robustness in every constrained RL environment we tested. Here we propose a complementary method to promote robustness both within and across environments. Specifically, it addresses the sensitivity of learning dynamics to the relative numerical scale of reward and cost objectives.

Consider two CMDPs that are identical except that in one the rewards are scaled by a constant factor, ρ. The optimal policy parameters, θ^* remain unchanged, but clearly λ^* must scale by ρ. To attain the same learning dynamics, all controller settings, λ_0, K_I, K_P, and K_D must therefore be scaled by ρ. This situation might feature naturally within a collection of related learning environments. Additionally, within the course of learning an individual CMDP, the balance between reward and cost magnitudes can change considerably, placing burden on the controller to track the necessary changes in the scale of λ.

One way to promote performance of a single choice of controller settings across these cases would be to maintain a fixed meaning for the value of λ in terms of the relative influence of reward versus cost on the parameter update. To this end, we introduce an adjustable scaling factor, β_k, in the policy gradient:

$$\nabla_\theta \mathcal{L} = (1 - u_k)\nabla_\theta J(\pi_{\theta_k}) - u_k \beta_k \nabla_\theta J_C(\pi_{\theta_k}) \tag{8.13}$$

A conspicuous choice for β_k is the ratio of un-scaled policy gradients:

$$\beta_{\nabla,k} = \frac{||\nabla_\theta J(\pi_{\theta_k})||}{||\nabla_\theta J_C(\pi_{\theta_k})||} \tag{8.14}$$

since it balances the total gradient to have equal-magnitude contribution from reward- and cost-objectives at $\lambda = 1$ and encourages $\lambda^* = 1$. Furthermore, β_∇ is easily computed with existing algorithm components.

To test this method, we ran experiments on Safety Gym environments with their rewards scaled up or down by a factor of 10. Figure 8.9 shows a representative cross-section of results from the PointGoal1 environment using PI-control. The different curves within each plot correspond to different reward scaling. Note the near-logarithmic spacing of λ curves when objective scaling is not used (the $\beta = 1$ column), and the large variation in cost curves. Using β_∇, on the other hand, the learning dynamics are nearly identical across two orders of magnitude of reward scale. $\lambda_0 = 1$ becomes an obvious choice for initialization, a point where previous theory provides little guidance [Chow et al., 2019] (although here we left $\lambda_0 = 0$). Experiments in other environments and controller settings yielded similar results and are included in supplementary materials. Other methods, such running normalization of rewards and costs, could achieve similar effects and are worth investigating, but our simple technique is surprisingly effective and is not specific to RL.

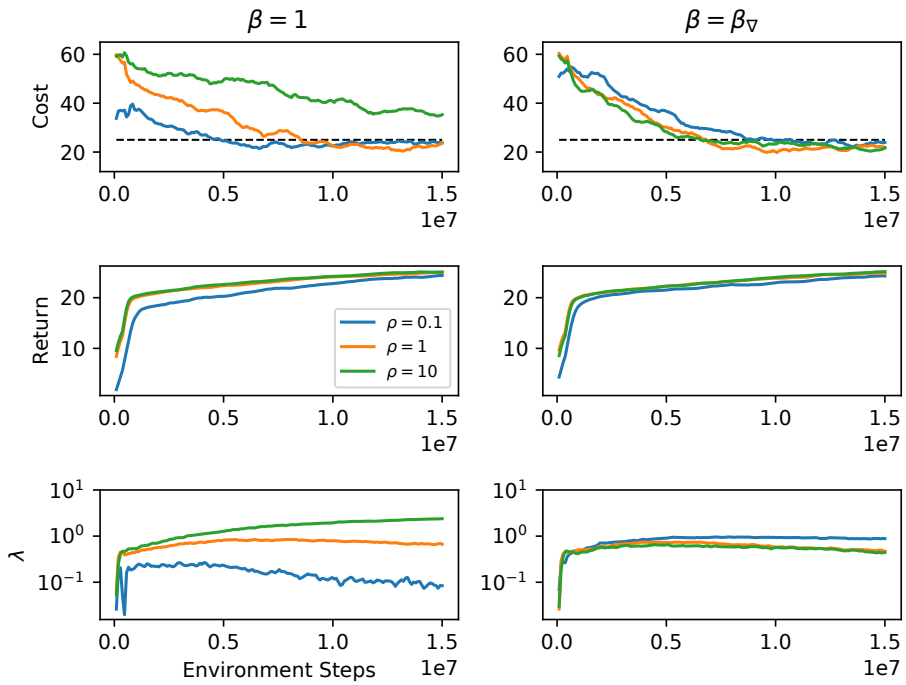


Figure 8.9: Costs, returns, and Lagrange multiplier with rewards scaled by $\rho \in \{0.1, 1, 10\}$; PI-control with $K_I = 1e-3, K_P = 0.1$. *Left column*: without objective-weighting, learning dynamics differ dramatically due to required scale of λ. *Right column*: with objective-weighting, learning dynamics are nearly identical. Environment: PointGoal1, cost limit 25.

8.8 Conclusion

Starting from a novel development in classic Lagrangian methods, we introduced a new set of constrained RL solutions which are straightforward to understand and implement, and we have shown them to be effective when paired with deep learning.

Several opportunities for further work lay ahead. Analysis of the modified Lagrangian method and constrained RL as a dynamical system may relax theoretical requirements for a slowly-changing multiplier. The mature field of control theory (and practice) provides tools for tuning controller parameters. Lastly, the control-affine form may assist in both analysis (see Liang-Liang Xie and Lei Guo [2000] and Galbraith and Vinter [2003] for controllability properties for uncertain nonlinear dynamics) and by opening to further control techniques such as feedback linearization.

Our contributions improve perhaps the most commonly used constrained RL algorithm, which is a workhorse baseline. We have addressed its primary shortcoming while preserving its simplicity and even making it easier to use—a compelling combination to assist in a wide range of applications.

8.9 Experiment Details

All of our experiments began with randomly initialized agents. The reward-value and cost-value estimators shared parameters with the policy. We used Generalized Advantage Estimation for both reward and cost advantages. The control input is updated once per iteration, which in our settings included multiple gradient updates on the policy. Training batches typically included the end of several trajectories, allowing an estimate of the average episodic sum of costs at each iteration.

Table 8.1: Experiment hyperparameters.

Hyperparameter	Value
learning rate	1×10^{-4}
NN hidden layer size	512
NN nonlinearity	$tanh$
batch dimension, time	128
batch dimension, envs	104
PPO epochs	8
PPO minibatches	1
PPO ratio-clip	0.1
Discount, γ	0.99
λ_{GAE}	0.97
Cost scaling	1/10
Exponential moving average, K_P	0.95
Exponential moving average, K_D	0.9
Difference iterates delay, K_D	15
Observation normalization	True
$\hat{\beta}$	1 (unless specified)
Exponential moving average, $\hat{\beta}$	0.9

8.10 Additional Learning Curves

8.10.1 Cost and Reward Curves for Additional Environments

This section contains learning curves from the experiments used to make the figures showing cost figure-of-merit versus Lagrange multiplier learning rate, K_I, in the main text. The main text includes curves from DoggoGoal2, the other environments are shown here.

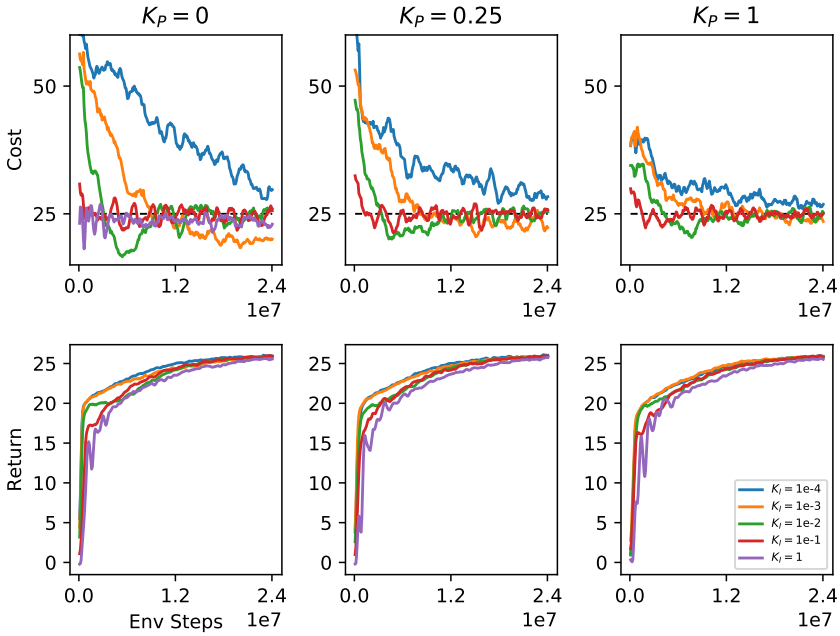


Figure 8.10: Costs and returns with varying Lagrange multiplier learning rate, K_I, and proportional control coefficient, K_P, in PointGoal1, cost-limit=25.

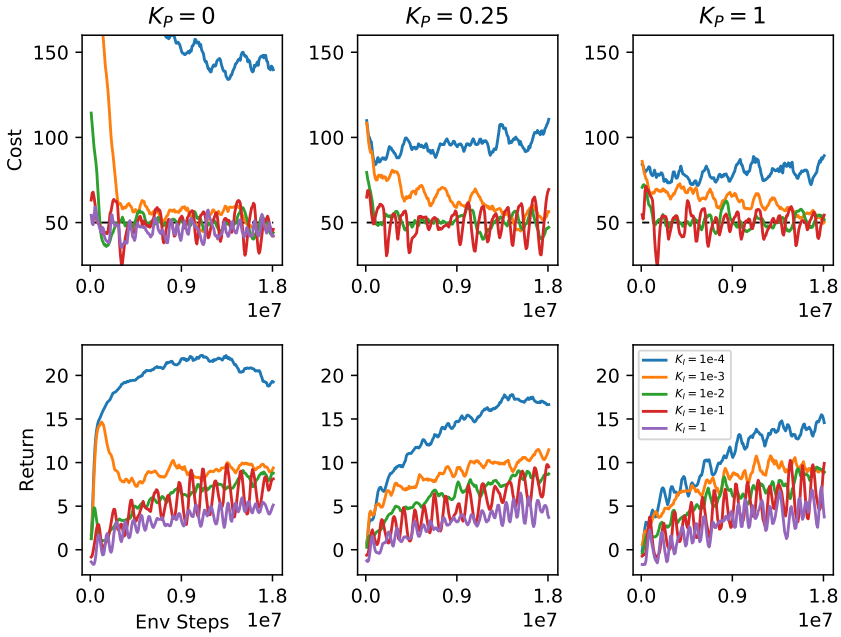


Figure 8.11: Costs and returns with varying Lagrange multiplier learning rate, K_I, and proportional control coefficient, K_P, in CarButton1, cost-limit=50.

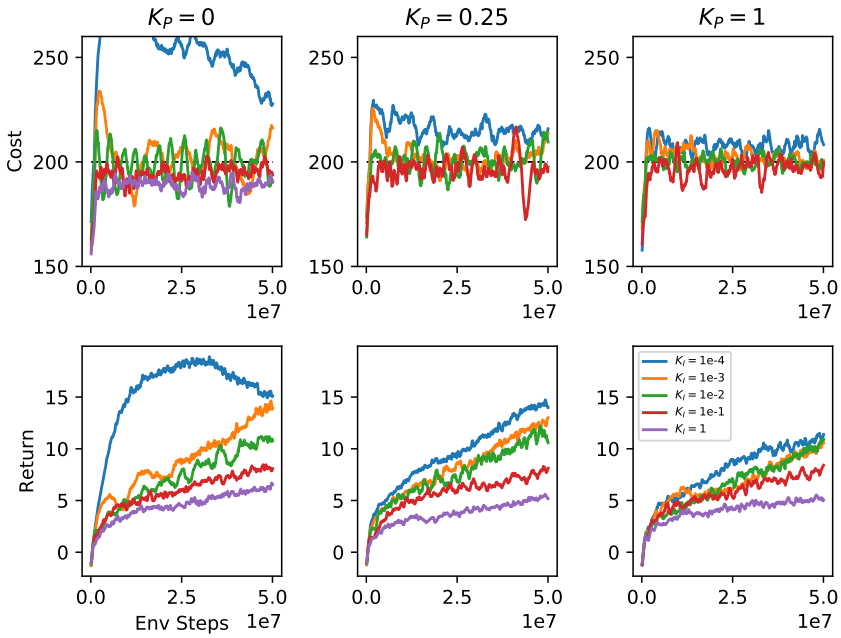


Figure 8.12: Costs and returns with varying Lagrange multiplier learning rate, K_I, and proportional control coefficient, K_P, in DoggoButton1, cost-limit=200.

8.10.2 Derivative-Control Learning Curves

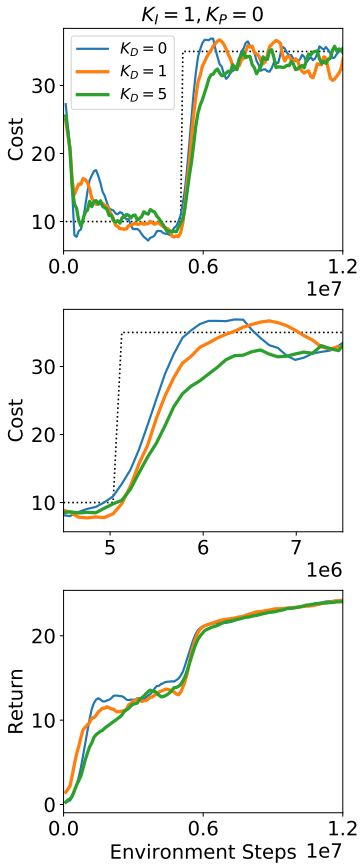


Figure 8.13: Derivative control slows the increase in cost, causing it to rise more gradually, and reducing overshoot. The cost limit was increased from 10 to 35 at 5M environment steps. Environment: PointGoal1.

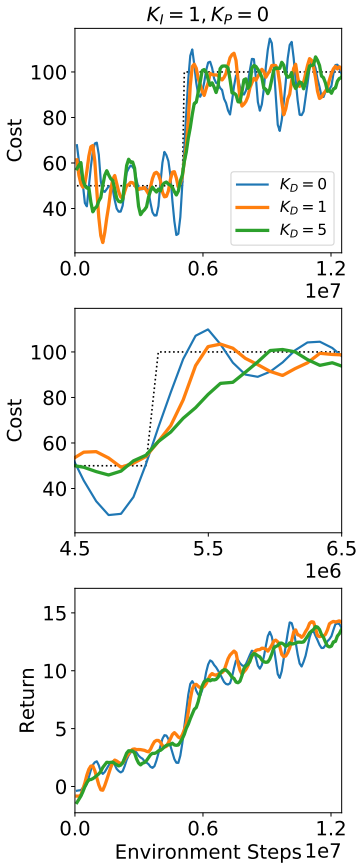


Figure 8.14: Derivative control slows the increase in cost, causing it to rise more gradually, and reducing overshoot. The cost limit was increased from 50 to 100 at 5M environment steps. Environment: CarButton1.

8.10.3 Comparison to Unconstrained Algorithms

This section shows learning curves for the same four environments referenced in the main text. These figures demonstrate that the unconstrained algorithm (PPO) does not satisfy the cost constraints, and as a result it achieves higher rewards.

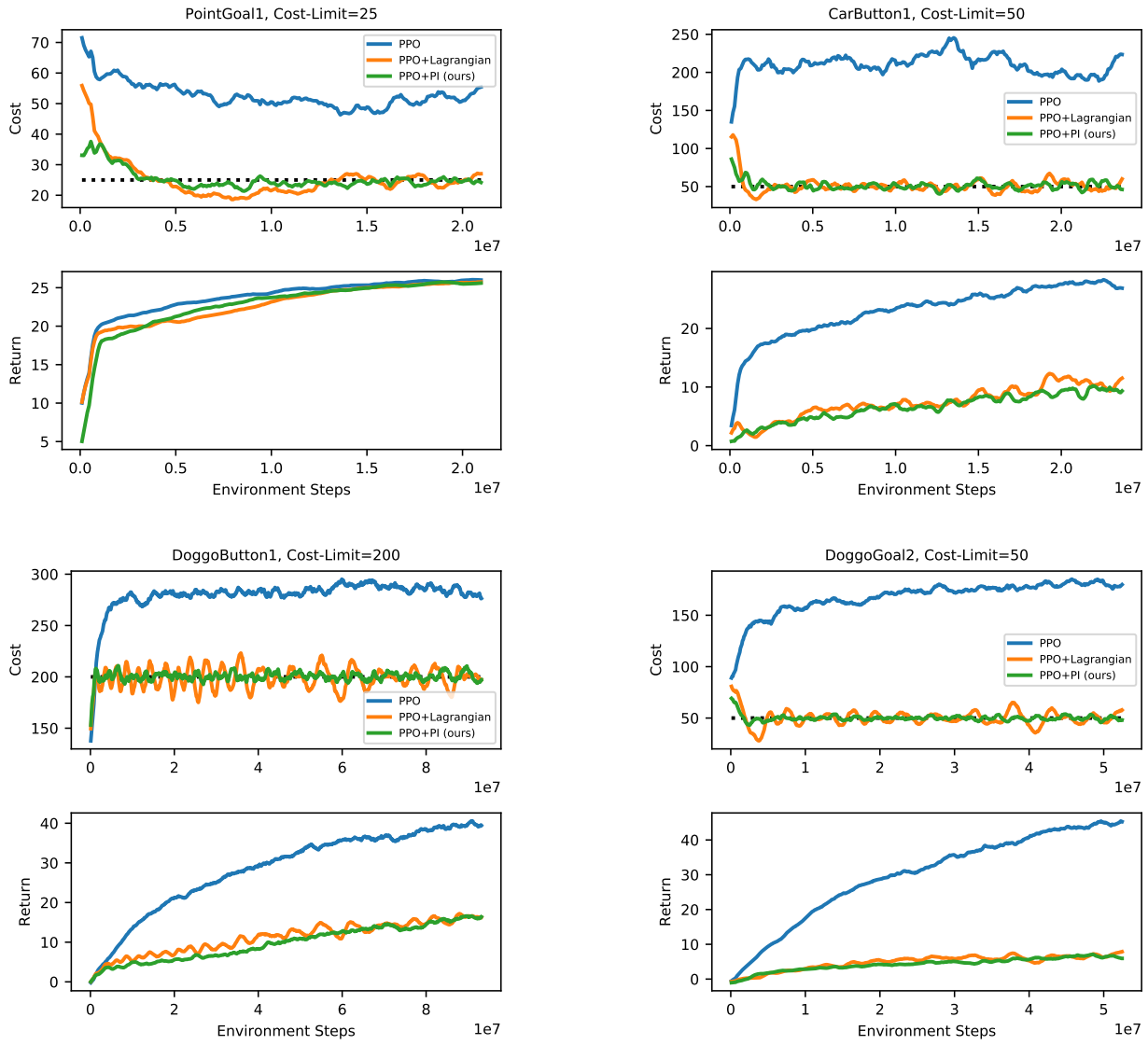


Figure 8.15: Cost and reward curves for three variants of PPO: unconstrained, Lagrangian, and PI-Controlled. The unconstrained algorithm wildly violates all cost-limits used in our experiments. PPO+Lagrangian and PPO+PI use the same Lagrange multiplier learning rate, K_I.

8.11 Adaptive Objective-Balancing

Alternative, KL-Based Estimator The magnitudes of the gradients in θ-space might not fully reflect the relative impacts of reward- and cost-learning on agent behavior. Reinforcement learning offers an alternative grounding in policy-space, for example using:

$$\beta_{KL} = \frac{D_{KL}(\pi_{\theta_k} || \pi^R_{\theta_k+1})}{D_{KL}(\pi_{\theta_k} || \pi^C_{\theta_k+1})} \tag{8.15}$$

Here, $\pi^R_{\theta_{k+1}}$ and $\pi^C_{\theta_{k+1}}$ are hypothetical new policies found using only the reward-objective or only the (un-scaled) cost-objective, respectively. We experimented with this estimator and found it to work, although not as well as the gradient-norm estimator in our cases. Some results are included in the figures below.

Figures We include figures for PointGoal1 using I-control and PI-control, and the more challenging DoggoGoal2 using I-control. Observe in the un-balanced case ($\hat{\beta} = 1$) that as the controller settings scale with the reward, the learning dynamics remained the same. For example, see $(K_I = 0.1, \rho = 10), (K_I = 0.01, \rho = 1)$, and $(K_I = 0.001, \rho = 0.1)$. Using gradient-based objective balancing (β_∇), however, the learning dynamics were roughly the same across all reward scales, for given controller settings. Alternatively, KL-based objective balancing (β_{KL}) was also effective, but did not produce dynamics as uniform as the gradient-based method.

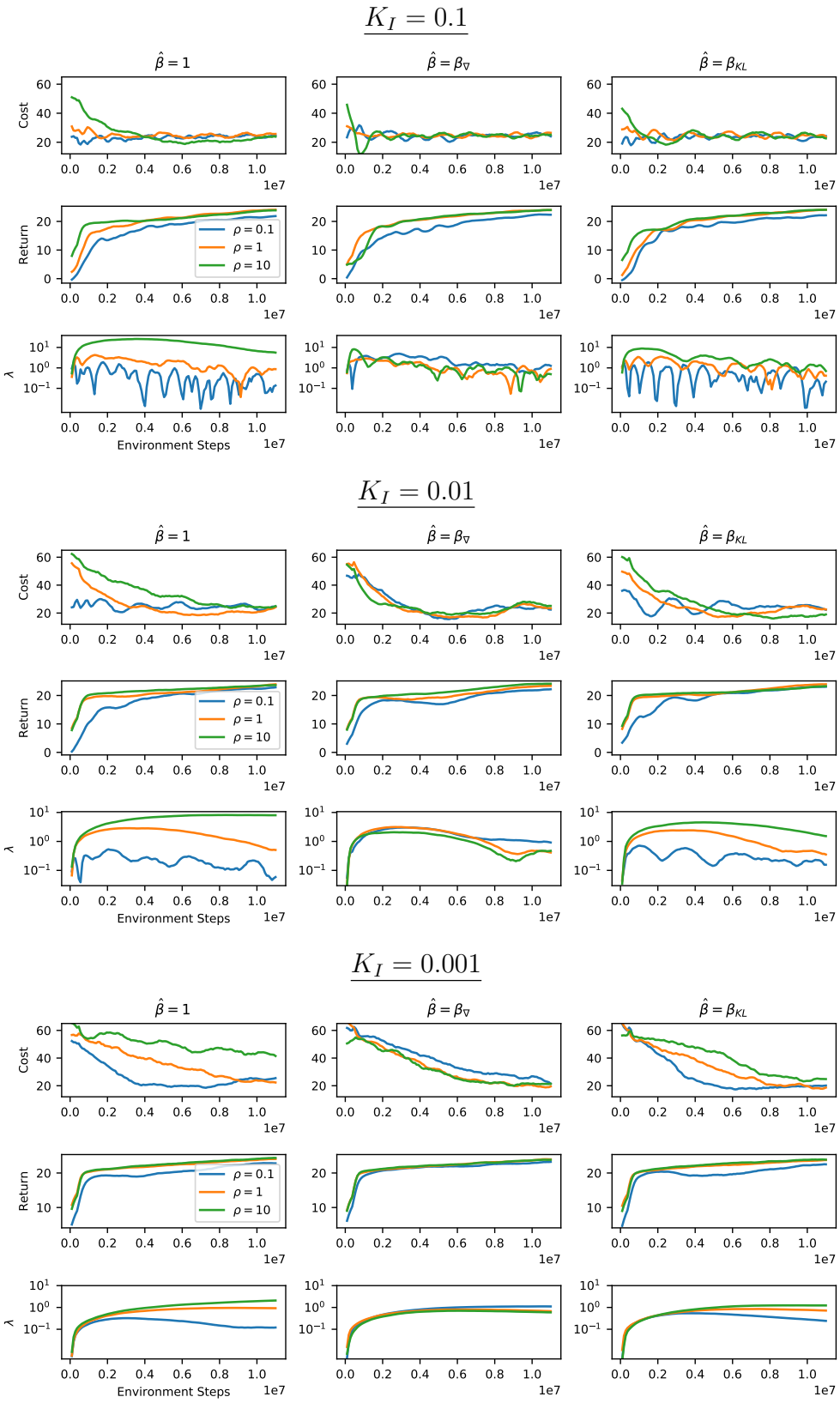


Figure 8.16: Reward scaling, I-control, PointGoal1, cost-limit=25.

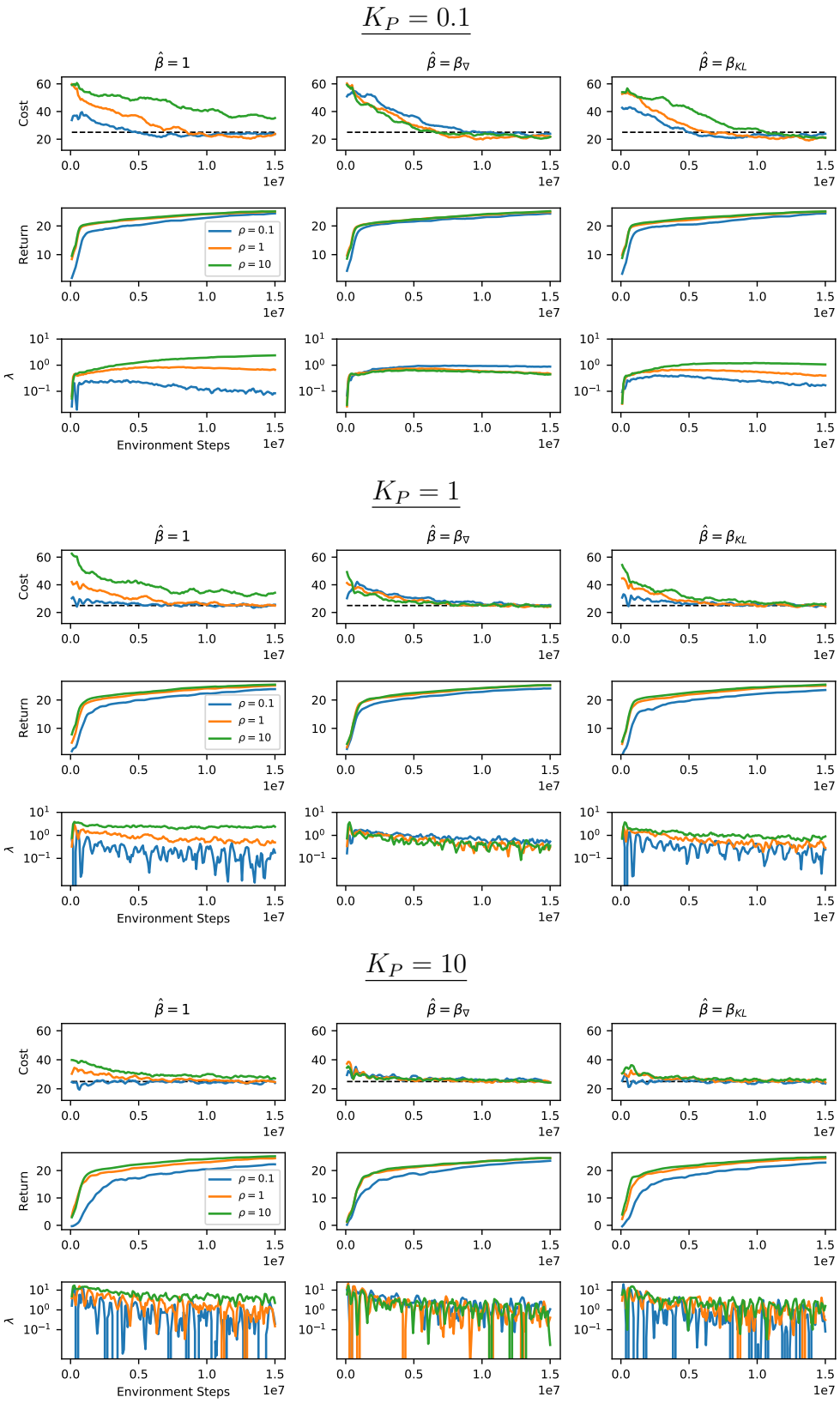


Figure 8.17: Reward scaling, PI-control with $K_I = 0.001$, PointGoal1, cost-limit=25.

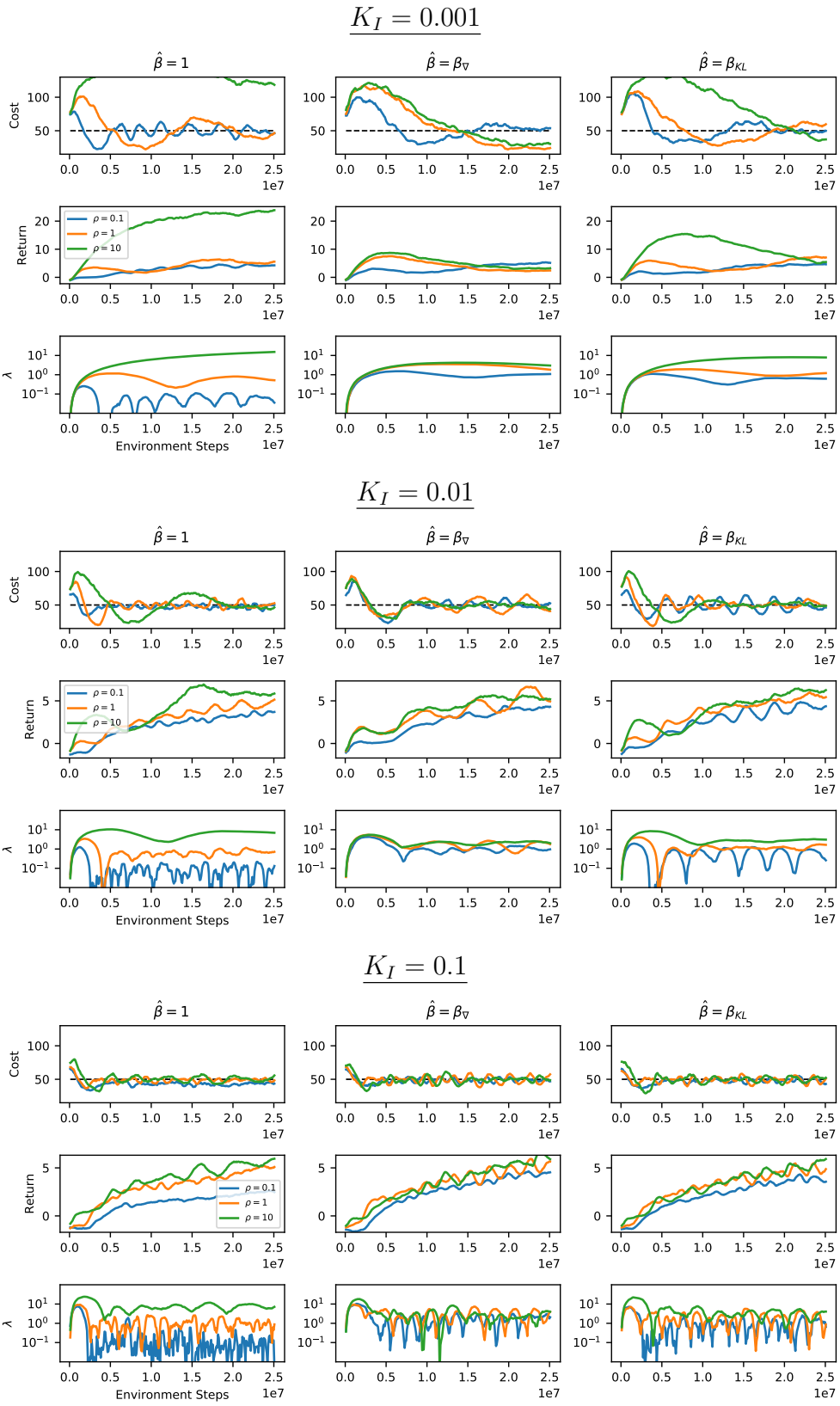


Figure 8.18: Reward scaling, I-control, DoggoGoal2, cost-limit=50.

Chapter 9

Epilogues

In this thesis, we incentivized RL agents to explore in ways that were safer or better-suited for sparse rewards by modifying the underlying RL optimization problem. To conclude, we will recap our contributions and reflect on progress and frontiers in the field.

Surprise-Based Intrinsic Motivation. In Chapter 3, we developed intrinsic rewards based on the surprise perceived by an agent according to a dynamics model learned concurrently with the policy, and we showed that this simple and scalable approach outperformed naive exploration schemes on hard sparse reward tasks. Since the time of our work, research on learned intrinsic rewards and exploration in sparse reward environments has continued, resulting in several algorithmic advances. However, open questions remain about their effectiveness and best practices.

The counting-based intrinsic reward method typified by Bellemare et al. [2016], who originally used CTS density models, was extended by Tang et al. [2017] who counted hashes of states, and by Ostrovski et al. [2017] who used neural density models based on PixelCNNs. On a parallel track, surprise-based intrinsic rewards were iterated on by Pathak et al. [2017], Burda et al. [2018a], and Burda et al. [2018b]. Pathak et al. [2017] developed the Intrinsic Curiosity Module (ICM), which learns an embedding model ϕ that maps from states to representation vectors, and a forward dynamics model that predicts the next representation $\hat{\phi}(s_{t+1})$ based on the current $\phi(s_t)$. Intrinsic rewards for ICM are constructed as the surprisal of a unit-variance Gaussian distribution with mean $\hat{\phi}(s_{t+1})$: $r_+(s_t) = \|\hat{\phi}(s_{t+1}) - \phi(s_{t+1})\|_2^2$. Pathak et al. [2017] learned the embedding model by optimizing an inverse model (learning ϕ such that the action a_t could be recovered from $\phi(s_t)$ and $\phi(s_{t+1})$), though Burda et al. [2018a] relaxed this choice to carry out a comparison of various embedding models, including raw pixels, random features, and representations from a VAE. Burda et al. [2018b] simplified further by removing the need to learn an embedding or construct a separate forward model: they presented Random Network Distillation (RND), where intrinsic rewards are constructed as the error of a learned network f_ϕ from a random network f_ξ: $r_+(s_t) = \|f_\phi(s_t) - f_\xi(s_t)\|_2^2$.

Apparent improvements over baselines were observed at every step of algorithmic advancement. A recent study by Taïga et al. [2019], however, suggests that many of the gains may have been illusory or misattributed. The various separate studies proposing novel intrinsic rewards

did not use equivalent evaluations protocols—the "tuning"[1] was slightly different in each. Taïga et al. [2019] ran a new comparison using a stronger naive exploration baseline than previous works—a DQN variant called Rainbow DQN [Hessel et al., 2017] that uses ϵ-greedy exploration—and putting methods on closer-to-equal footing. Their findings suggested that the newer intrinsic rewards had not improved on the pseudocount method of Bellemare et al. [2016], and in environments where exploration was easy to start with, the intrinsic rewards actively hurt performance. Most intriguingly, in environments where exploration had previously been considered hard based on the performance of older naive exploration RL algorithms, the improved naive exploration baseline beat expectations and often outperformed the intrinsic rewards.

If one takes the perspective that tuning makes the most difference in deep learning, this is almost obvious in retrospect. The various intrinsic reward schemes are all variations on a theme: intrinsic rewards should be high for states that are comparatively novel, and they should go to zero over the course of training. It would follow that as long as the concept is faithfully implemented, *everything* else is tuning, and when tuning is equal, the methods should have roughly equal performance. Some methods might be easier to tune than others, and the correct way to tune may vary between environments, but there are few free lunches here. Our concluding sentiment, based on our research and the results in the field, is to speculate that on balance this is likely to be true—and that further improvements in intrinsic rewards will be the result of better tuning, or prehaps automatic tuning, rather than a novel conceptual framework.

Variational Option Discovery. In Chapter 4, we developed variational option discovery algorithms, where agents learn to map noise vectors to behaviors that are distinguishable according to a learned decoder network. We demonstrated that this approach could allow agents to learn complex, composable locomotion behaviors without rewards, making progress on the problem of exploring without direction. Since our work, the field has surfaced a variety of fruitful ways to apply and improve variational option discovery algorithms.

In our previous discussion of intrinsic rewards, we suggested that improved tuning and perhaps automatic tuning would be key to algorithmic progress. In that spirit, we highlight the interesting approach of Gupta et al. [2018], who showed a way to combine variational option discovery with meta-learning. The goal in meta-learning is to train a single agent that is capable of quickly becoming proficient when exposed to a new task. The meta-learning problem in RL is usually formalized as an optimization problem where the objective function is an expected return over a distribution of tasks, with each task being a different MDP. Often the tasks have the same or nearly the same dynamics, and the reward functions are distinct between them. A key challenge in applying meta-learning to a given problem is designing a task distribution that is sufficiently diverse for an agent to learn how to generalize well. This usually requires a nontrivial amount of human expertise—a human tunes the distribution until the desired result comes out. The insight of Gupta et al. [2018] was that variational option discovery could provide a path to automatically generating a suitable task distribution, and they developed a simple two-step procedure as a prototype: in step one,

[1] By "tuning," we mean the kinds of design details discussed in Chapter 2: choices for network architecture, learning rates, loss function regularization, and other tricks.

running DIAYN [Eysenbach et al., 2018b] to get a decoder network, and in step two, running MAML (a standard meta-learning algorithm, [Finn et al., 2017]) using a task distribution where rewards come from the DIAYN decoder (tasks are sampled contexts, $c \sim P(c)$, with rewards $r(s, c) = \log P_D(c|s)$). The results in their experiments were not definitive but encouraging overall.

Another noteworthy direction of work comes from Sharma et al. [2020], who developed Dynamics-Aware Discovery of Skills (DADS). Their approach to variational option discovery starts with a mutual information objective—maximize the mutual information between the next state and the context, given the current state—leading to a variational lower bound involving a context-dependent dynamics model $q_\phi(s'|s, c)$, and rewards that approximate $r_c(s, a, s') = \log q_\phi(s'|s, c) - \log p(s'|s)$. While this may appear to give a different objective than the kind we've considered in this thesis, it's exactly equivalent to the variational intrinsic control objective with one timestep between start and final states instead of multiple [Gregor et al., 2016] . In the parlance of Gregor et al. [2016], this is the "forward" form of the objective, and the form we discussed in chapter 4 is the "reverse" form.

There are two aspects from the DADS research we consider interesting. First, the framework difference in DADS—the use of the forward instead of reverse objective—results in learning a dynamics model as a byproduct of policy optimization, which can be used in a planning algorithm, as demonstrated by Sharma et al. [2020]. This is interesting because it is a pure benefit you can get from the choice of framework, separate from tuning—a rare free lunch (or if there's a nonobvious catch, at least a lunch that's half off). Second, their empirical results show that DADS is able to learn walking gaits in a simulated humanoid. On the surface, this seems to improve over our results with VALOR—one of the key limitations we found was an inability to learn meaningful locomotion behavior in a simulated toddler environment—though the story is somewhat murky because of differences in environment tuning. The DADS research appears to use the Gym humanoid [Brockman et al., 2016], which terminates an episode if the humanoid falls over, whereas we used a customized, differently-balanced humanoid with no termination condition; these differences make it impossible to directly compare whether the results for DADS are due to an algorithmic advantage. Nonetheless, we see the progress as exciting—even if tuning is the difference, the demonstration that a variational option discovery algorithm can learn humanoid walking gaits is of general interest.

We conclude our discussion of progress in variational option discovery by considering the work of Campos et al. [2020], who introduced Explore, Discover, Learn (EDL), an option discovery approach that optimizes the usual objective but with different machinery. Campos et al. [2020] highlight an important shortcoming of variational option discovery methods: unless some attempt is made to impose structure on how the agent explores, algorithms are likely to learn skills that slice up the space of trajectories generated by the initial random policy, rather than skills that cover the space of all possible trajectories. EDL addresses this issue by decoupling the exploration and option learning phases; during the exploration phase, they attempt to learn or sample from a uniform distribution over states $p^*(s)$. Then, during the option learning phase, they try to learn options such that the joint distribution $p(s, c)$ has marginal distribution $p^*(s)$. It isn't obvious if a uniform distribution over states

is the right prior—after all, it is not invariant to general coordinate transformations of states—but nonetheless the problem being addressed is basically right and the approach is well-motivated.

Constrained RL and Safe Exploration. In Chapters 6, 7, and 8, we developed methods and benchmarks for safe exploration in deep RL, and we demonstrated that constrained deep RL was a viable approach to formalizing and making progress in this area. We will offer concluding thoughts here that focus not on related work in the field, but on the *gaps*—the issues in safe RL, and safe AI more generally, that we feel are not satisfactorily addressed by the literature, and which require more attention and awareness.

First and foremost is the conceptual gap between safety as it is discussed in the AI safety literature, and safety as it is understood by practitioners in other engineering disciplines. These conceptual gaps are significant but only infrequently discussed, and there will be significant friction in operationalizing AI safety in practice if they remain unaddressed.

Fields like nuclear, chemical, and aerospace engineering have a common approach for how they systematize and address safety risks:

- there are clearly-defined conceptual elements like losses (actual realized harms), hazards (conditions that are likely to lead to losses if not controlled), risk (a measure blending how severe and frequent an incident is), and safety requirements (conditions that must be met for risks to be considered controlled);
- there are safety activities scheduled throughout system lifecycles to ensure risks are identified and mitigated early on, like hazard analysis, generation and allocation of requirements, safety reviews at key decision points, and verification and validation of system safety claims;
- and there is a recognition that safety is a system-level property—that risks may arise not just from components but from interactions between them.

The standards and processes that are used across these fields are the distilled results of decades of trial and error—rules "written in blood." But it appears, in our experience, that these concepts are largely unknown to most AI safety researchers. Additionally, some well-intentioned researchers concerned about AI alignment have concluded that because specifying AI behavior correctly is hard, all of the old methods for specification are unlikely to apply, so they should be ignored in favor of alternative approaches. While this is an extreme position that few hold directly, diluted forms of it are common, and we reject this view as throwing out the baby with the bathwater.

This gap means that AI safety research papers are rarely connected in an informative way to rigorous practice: they contain many important ideas, but the ideas are disjointed and the papers contain no roadmap that might be practiced by someone responsible for making a real AI system safe in the real world. There is effort in this direction—in particular we draw attention to the work of Mitchell et al. [2019] and Raji et al. [2020]—but it is not yet the default mode of the research community to contextualize safety work in this manner.

Consequences from this gap are manifold. If the field of AI safety does not correct its course,

it will expend an enormous amount of unnecessary energy reinventing the wheel, when it could instead adapt existing standards and practices from other fields. It will take longer to get things right than it ought to, and preventable safety issues will spring up due to the lack of rigor in the meanwhile.

In our work, we've made a partial attempt to close this gap by emphasizing the importance of explicitly specifying safety constraints. Our advocacy for constrained RL is rooted in this connection to safety practice in other fields, as we described in Chapter 5. We acknowledge, however, that there is much more to be done. While our Safety Gym benchmark is useful for evaluating whether a learning procedure is well-designed and well-implemented, it is not useful for evaluating the safety of a system aimed at deployment in the real world; its hazards are crude representations of hazards that arise in robotics, and not representative of the broad range of hazards that will need to be mitigated in impactful AI systems in other domains.

There is a need for the development of what we might describe as safety *infrastructure*: standardized hazard analysis procedures, datasets, classifiers, and benchmarks for evaluating the safety-relevant characteristics of domain-specific AI models—not just the efficiency of learning algorithms on toy tasks. This infrastructure needs to be well-documented and stress-tested. If the field of AI safety research functions optimally, each unit of research will contribute to the grand canon of this infrastructure, derisking concepts and implementations that address each facet of risk likely to arise from the deployment of AI models into real world safety-critical environments.

To illustrate the nature of the need—the best constrained RL algorithm in the world will not endow a question-answering AI with the ability to give reliably correct medical advice, if no effort has first been made to produce a standard repository of correct medical data and evaluations metrics.

It will be a challenge to build and maintain such infrastructure, especially because it will be domain-specific—but there are many precedents and communities that can be tapped into to help, and it is in everyone's interest to do so. We hope our work has contributed to this vision, and we hope to see this vision realized in time.[2]

[2]The author gratefully acknowledges the coauthorship of GPT-3 [Brown et al., 2020] on this final paragraph.

Bibliography

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: A system for large-scale machine learning. *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016*, pages 265–283, may 2016. URL `http://arxiv.org/abs/1605.08695`.

Pieter Abbeel, Rocky Duan, Vlad Mnih, Andrej Karpathy, John Schulman, Peter Chen, Chelsea Finn, and Sergey Legine. Deep RL Bootcamp - Lectures, 2017. URL `https://sites.google.com/view/deep-rl-bootcamp/lectures`.

Joshua Achiam. Spinning Up in Deep Reinforcement Learning, 2018. URL `https://spinningup.openai.com/en/latest/`.

Joshua Achiam and Shankar Sastry. Surprise-Based Intrinsic Motivation for Deep Reinforcement Learning. In *NeurIPS Deep RL Workshop*, 2016. URL `http://arxiv.org/abs/1703.01732`.

Joshua Achiam, Harrison Edwards, Dario Amodei, and Pieter Abbeel. Variational Autoencoding Learning of Options by Reinforcement. In *NeurIPS Deep RL Workshop*, 2017a. URL `https://drive.google.com/file/d/1mq6vpSNylfbFG4A1MOdv9czGMVuB7jc6/view`.

Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained Policy Optimization. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 22–31, International Convention Centre, Sydney, Australia, 2017b. PMLR. URL `http://proceedings.mlr.press/v70/achiam17a.html`.

Joshua Achiam, Harrison Edwards, Dario Amodei, and Pieter Abbeel. Variational Option Discovery Algorithms. 2018. URL `http://arxiv.org/abs/1807.10299`.

Eitan Altman. Constrained Markov decision processes with total cost criteria: Lagrangian approach and dual linear program. *Mathematical methods of operations research*, 48(3): 387–417, 1998.

Eitan Altman. Constrained Markov Decision Processes. page 260, 1999. ISSN 01676377. doi: 10.1016/0167-6377(96)00003-X.

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete Problems in AI Safety. 2016.

Wangpeng An, Haoqian Wang, Qingyun Sun, Jun Xu, Qionghai Dai, and Lei Zhang. A PID Controller Approach for Stochastic Optimization of Deep Networks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8522–8531, 2018.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight Experience Replay. *NIPS*, 2017. URL `http://arxiv.org/abs/1707.01495`.

Karl J Åström and Tore Hägglund. PID control. *IEEE Control Systems Magazine*, 1066 (033X/06), 2006.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization. jul 2016. URL `http://arxiv.org/abs/1607.06450`.

Pierre-luc Bacon, Jean Harb, and Doina Precup. The Option-Critic Architecture. *AAAI*, 2017.

Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, sep 2015. URL `https://arxiv.org/abs/1409.0473v7`.

Andrew Barto, Marco Mirolli, and Gianluca Baldassarre. Novelty or Surprise? *Frontiers in Psychology*, 4(DEC), 2013. ISSN 16641078. doi: 10.3389/fpsyg.2013.00907.

Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, Julian Schrittwieser, Keith Anderson, Sarah York, Max Cant, Adam Cain, Adrian Bolton, Stephen Gaffney, Helen King, Demis Hassabis, Shane Legg, and Stig Petersen. DeepMind Lab. dec 2016. URL `http://arxiv.org/abs/1612.03801`.

Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research*, jul 2012. doi: 10.1613/jair.3912. URL `http://arxiv.org/abs/1207.4708http://dx.doi.org/10.1613/jair.3912`.

Marc G Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, Google Deepmind, and Rémi Munos. Unifying Count-Based Exploration and Intrinsic Motivation. In *Neural Information Processing Systems*, 2016.

Y Bengio and Yann Lecun. Convolutional Networks for Images, Speech, and Time-Series. 1997.

Felix Berkenkamp, Matteo Turchetta, Angela P Schoellig, and Andreas Krause. Safe Model-based Reinforcement Learning with Stability Guarantees. 2017.

Dimitri P Bertsekas. On penalty and multiplier methods for constrained minimization. *SIAM Journal on Control and Optimization*, 14(2):216–235, 1976.

Dimitri P Bertsekas. *Constrained optimization and Lagrange multiplier methods*. Academic press, 2014.

Timothy W Bickmore, Ha Trinh, Stefan Olafsson, Teresa K O'Leary, Reza Asadi, Nathaniel M Rickles, and Ricardo Cruz. Patient and Consumer Safety Risks When Using Conversational Assistants for Medical Information: An Observational Study of Siri, Alexa, and Google Assistant. *Journal of medical Internet research*, 20(9):e11510, sep 2018. ISSN 1438-8871. doi: 10.2196/11510. URL http://www.jmir.org/2018/9/e11510/http://www.ncbi.nlm.nih.gov/pubmed/30181110http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC6231817.

Steven Bohez, Abbas Abdolmaleki, Michael Neunert, Jonas Buchli, Nicolas Heess, and Raia Hadsell. Value constrained model-free continuous control. Technical report, 2019. URL http://arxiv.org/abs/1902.04623.

Haitham Bou Ammar, Rasul Tutunov, and Eric Eaton. Safe Policy Search for Lifelong Reinforcement Learning with Sublinear Regret. *International Conference on Machine Learning*, 37:19, 2015. URL http://arxiv.org/abs/1505.0579.

Stephen Boyd, Lin Xiao, and Almir Mutapcic. Subgradient methods. *lecture notes of EE392o, Stanford* {...}, 1(May):1–21, 2003. URL http://xxpt.ynjgy.com/resource/data/20100601/U/stanford201001010/02-subgrad{%}7B{_}{%}7Dmethod{%}7B{_}{%}7Dnotes.pdf.

Ronen I Brafman and Moshe Tennenholtz. R-max A General Polynomial Time Algorithm for Near-Optimal Reinforcement Learning. *Journal of Machine Learning Research*, 3:213–231, 2002.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. 2016. URL http://arxiv.org/abs/1606.01540.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners, 2020.

Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A. Efros. Large-scale study of curiosity-driven learning, aug 2018a. ISSN 23318422. URL https://pathak22.github.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by Random Network Distillation. *arXiv*, oct 2018b. URL http://arxiv.org/abs/1810.12894.

Victor Campos, Alexander Trott, Caiming Xiong, Richard Socher, Xavier Giro-I-Nieto, and Jordi Torres. Explore, Discover and Learn: Unsupervised Discovery of State-Covering Skills. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference*

on Machine Learning, volume 119 of *Proceedings of Machine Learning Research*, pages 1317–1327. PMLR, 2020. URL http://proceedings.mlr.press/v119/campos20a.html.

Yinlam Chow, Mohammad Ghavamzadeh, Lucas Janson, and Marco Pavone. Risk-Constrained Reinforcement Learning with Percentile Risk Criteria. *Journal of Machine Learning Research*, 2015.

Yinlam Chow, Ofir Nachum, Edgar Duenez-Guzman, and Mohammad Ghavamzadeh. A Lyapunov-based Approach to Safe Reinforcement Learning, 2018. URL https://papers.nips.cc/paper/8032-a-lyapunov-based-approach-to-safe-reinforcement-learning.

Yinlam Chow, Ofir Nachum, Aleksandra Faust, Edgar Duenez-Guzman, and Mohammad Ghavamzadeh. Lyapunov-based Safe Policy Optimization for Continuous Control. jan 2019. URL http://arxiv.org/abs/1901.10031.

Paul Christiano, Jan Leike, Tom B Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. 2017.

Paul Christiano, Buck Shlegeris, and Dario Amodei. Supervising strong learners by amplifying weak experts. oct 2018. URL http://arxiv.org/abs/1810.08575.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. dec 2014. URL http://arxiv.org/abs/1412.3555.

Jack Clark and Dario Amodei. Faulty Reward Functions in the Wild, 2016. URL https://openai.com/blog/faulty-reward-functions/.

Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying Generalization in Reinforcement Learning. 2018.

I Csiszar and J Körner. Information Theory: Coding Theorems for Discrete Memoryless Systems. *Book*, 244:452, 1981. ISSN 0895-4801. doi: 10.2307/2529636. URL http://www.getcited.org/pub/102082957.

Gal Dalal, Krishnamurthy Dvijotham, Matej Vecerik, Todd Hester, Cosmin Paduraru, and Yuval Tassa. Safe Exploration in Continuous Action Spaces. 2018.

DeepMind. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. *DeepMind*, pages 1–16, 2019. URL https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/.

Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. OpenAI Baselines, 2017. URL https://github.com/openai/baselines.

Yan Duan, Xi Chen, John Schulman, and Pieter Abbeel. Benchmarking Deep Reinforcement Learning for Continuous Control. *The 33rd International Conference on Machine Learning (ICML 2016) (2016)*, 48:14, 2016. URL http://arxiv.org/abs/1604.06778.

Benjamin Eysenbach, Shixiang Gu, Julian Ibarz, and Sergey Levine. Leave no Trace: Learning to Reset for Safe and Autonomous Reinforcement Learning. In *International Conference on Learning Representations*, 2018a. URL https://openreview.net/forum?id=S1vuO-bCW.

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is All You Need: Learning Skills without a Reward Function. 2018b. URL http://arxiv.org/abs/1802.06070.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. *34th International Conference on Machine Learning, ICML 2017*, 3:1856–1868, mar 2017. URL http://arxiv.org/abs/1703.03400.

Jaime F. Fisac, Neil F. Lugovoy, Vicenc Rubies-Royo, Shromona Ghosh, and Claire J. Tomlin. Bridging Hamilton-Jacobi Safety Analysis and Reinforcement Learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8550–8556. IEEE, may 2019. ISBN 978-1-5386-6027-0. doi: 10.1109/ICRA.2019.8794107. URL https://ieeexplore.ieee.org/document/8794107/.

Carlos Florensa, Yan Duan, and Pieter Abbeel. Stochastic Neural Networks for Hierarchical Reinforcement Learning. *ICLR*, pages 1–17, 2017.

Roy Fox, Sanjay Krishnan, Ion Stoica, and Ken Goldberg. Multi-Level Discovery of Deep Options. 2017. URL http://arxiv.org/abs/1703.08294.

Kevin Frans, Henry M Gunn, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman Openai. Meta Learning Shared Hierarchies. In *ICLR*, 2018. URL https://openreview.net/pdf?id=SyX0IeWAW.

Justin Fu, John Co-Reyes, and Sergey Levine. EX2: Exploration with Exemplar Models for Deep Reinforcement Learning. In *NIPS*, pages 2577–2587, 2017. URL https://papers.nips.cc/paper/6851-ex2-exploration-with-exemplar-models-for-deep-reinforcement-learning.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing Function Approximation Error in Actor-Critic Methods. In *International Conference on Machine Learning*, feb 2018. URL http://arxiv.org/abs/1802.09477.

Grant N Galbraith and Richard B Vinter. Lipschitz Continuity of Optimal Controls for State Constrained Problems. *SIAM Journal on Control and Optimization*, 42(5):1727–1744, 2003. doi: 10.1137/S0363012902404711. URL https://doi.org/10.1137/S0363012902404711.

Chris Gamble and Jim Gao. Safety-first AI for autonomous data centre cooling and industrial control, 2018. URL https://deepmind.com/blog/safety-first-ai-autonomous-data-centre-cooling-and-industrial-control/.

Javier García and Fernando Fernández. A Comprehensive Survey on Safe Reinforcement Learning. *Journal of Machine Learning Research*, 16:1437–1480, 2015. ISSN 15337928.

Jason Gauci, Edoardo Conti, Yitao Liang, Kittipat Virochsiri, Yuchen He, Zachary Kaden, Vivek Narayanan, and Xiaohui Ye. Horizon: Facebook's Open Source Applied Reinforcement Learning Platform. Technical report, 2018. URL http://arxiv.org/abs/1811.00260.

Clement Gehring and Doina Precup. Smart Exploration in Reinforcement Learning Using Absolute Temporal Difference Errors. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '13, pages 1037–1044, Richland, SC, 2013. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-1-4503-1993-5. URL http://dl.acm.org/citation.cfm?id=2484920.2485084.

Peter Geibel and Fritz Wysotzki. Risk-Sensitive Reinforcement Learning Applied to Control under Constraints. *CoRR*, abs/1109.2, 2011. URL http://arxiv.org/abs/1109.2147.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

Karol Gregor, Danilo Rezende, and Daan Wierstra. Variational Intrinsic Control. 2016.

Shixiang Gu, Timothy Lillicrap, Zoubin Ghahramani, Richard E. Turner, and Sergey Levine. Q-Prop: Sample-Efficient Policy Gradient with An Off-Policy Critic. In *International Conference on Learning Representations*, 2017. URL http://arxiv.org/abs/1611.02247.

Abhishek Gupta, Benjamin Eysenbach, Chelsea Finn, and Sergey Levine. Unsupervised Meta-Learning for Reinforcement Learning. *arXiv*, jun 2018. URL http://arxiv.org/abs/1806.04640.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1861–1870, Stockholmsmässan, Stockholm Sweden, 2018a. PMLR. URL http://proceedings.mlr.press/v80/haarnoja18b.html.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic Algorithms and Applications. Technical report, dec 2018b. URL http://arxiv.org/abs/1812.05905.

Dylan Hadfield-Menell, Stuart J. Russell, Pieter Abbeel, and Anca Dragan. Cooperative Inverse Reinforcement Learning. In *NIPS 2016*, pages 3909–3917, 2016. URL http://papers.nips.cc/paper/6420-cooperative-inverse-reinforcement-learning.

Alexander Hans, Daniel Schneegaß, Anton M Schäfer, and Steffen Udluft. Safe Exploration for Reinforcement Learning. In *European Symposium on Artificial Neural Networks - Advances in Computational Intelligence and Learning*, 2008. ISBN 2930307080. URL https://pdfs.semanticscholar.org/5ee2/7e9db2ae248d1254107852311117c4cda1c9.pdf.

Karol Hausman, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, and Martin Riedmiller. Learning an Embedding Space for Transferable Robot Skills. *ICLR*, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2016-Decem, pages 770–778. IEEE Computer Society, dec 2016. ISBN 9781467388504. doi: 10.1109/CVPR.2016.90.

Yanzhang He, Tara N. Sainath, Rohit Prabhavalkar, Ian McGraw, Raziel Alvarez, Ding Zhao, David Rybach, Anjuli Kannan, Yonghui Wu, Ruoming Pang, Qiao Liang, Deepti Bhatia, Yuan Shangguan, Bo Li, Golan Pundak, Khe Chai Sim, Tom Bagby, Shuo-yiin Chang, Kanishka Rao, and Alexander Gruenstein. Streaming End-to-end Speech Recognition For Mobile Devices. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 2019-May:6381–6385, nov 2018. URL `http://arxiv.org/abs/1811.06621`.

David Held, Zoe Mccarthy, Michael Zhang, Fred Shentu, and Pieter Abbeel. Probabilistically Safe Policy Transfer. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep Reinforcement Learning that Matters. In *Thirty-Second AAAI Conference On Artificial Intelligence (AAAI)*, 2018. URL `http://arxiv.org/abs/1709.06560`.

Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining Improvements in Deep Reinforcement Learning. *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, pages 3215–3222, oct 2017. URL `http://arxiv.org/abs/1710.02298`.

Magnus R Hestenes. Multiplier and gradient methods. *Journal of optimization theory and applications*, 4(5):303–320, 1969.

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, Alexander Lerchner, and Google Deepmind. beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework. *Iclr*, (July): 1–13, 2017. URL `https://openreview.net/forum?id=Sy2fzU9gl`.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, nov 1997. ISSN 08997667. doi: 10.1162/neco.1997.9.8.1735.

Rein Houthooft. VIME Open-Source Code. \url{https://github.com/openai/vime}, 2016.

Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. VIME: Variational Information Maximizing Exploration. *NIPS*, may 2016. URL `http://arxiv.org/abs/1605.09674`.

Bin Hu and Laurent Lessard. Control Interpretations for First-Order Optimization Methods. *CoRR*, abs/1703.0, 2017. URL `http://arxiv.org/abs/1703.01670`.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *32nd International Conference on Machine Learning, ICML 2015*, volume 1, pages 448–456. International Machine Learning Society (IMLS), feb 2015. ISBN 9781510810587. URL `https://arxiv.org/abs/1502.03167v3`.

Alex Irpan. Deep Reinforcement Learning Doesn't Work Yet, 2018. URL `https://www.alexirpan.com/2018/02/14/rl-hard.html`.

Geoffrey Irving, Paul Christiano, and Dario Amodei. AI safety via debate. may 2018. URL http://arxiv.org/abs/1805.00899.

Alberto Isidori, M Thoma, E D Sontag, B W Dickinson, A Fettweis, J L Massey, and J W Modestino. *Nonlinear Control Systems*. Springer-Verlag, Berlin, Heidelberg, 3rd edition, 1995. ISBN 3540199160.

Riashat Islam, Peter Henderson, Maziar Gomrokchi, and Doina Precup. Reproducibility of Benchmarked Deep Reinforcement Learning Tasks for Continuous Control. *arXiv*, aug 2017. URL http://arxiv.org/abs/1708.04133.

Laurent Itti and Pierre Baldi. Bayesian surprise attracts human attention. *Vision Research*, 49(10):1295–1306, 2009. ISSN 00426989. doi: 10.1016/j.visres.2008.09.007.

Thomas Jaksch, Ronald Ortner, and Peter Auer. Near-optimal Regret Bounds for Reinforcement Learning. *Journal of Machine Learning Research*, 11(1):1563–1600, 2010. ISSN 15324435. URL http://eprints.pascal-network.org/archive/00007081/.

Nan Jiang and Lihong Li. Doubly Robust Off-policy Value Evaluation for Reinforcement Learning. *International Conference on Machine Learning*, 2015. URL http://arxiv.org/abs/1511.03722.

Sham Kakade and John Langford. Approximately Optimal Approximate Reinforcement Learning. *Proceedings of the 19th International Conference on Machine Learning*, pages 267–274, 2002. URL http://www.cs.cmu.edu/afs/cs/Web/People/jcl/papers/aoarl/Final.pdf.

Michael Kearns and Satinder Singh. Near Optimal Reinforcement Learning in Polynomial Time. *Proceedings of the 15th International Conference on Machine Learning*, pages 260–268, 1998.

Alex Kendall, Jeffrey Hawke, David Janz, Przemyslaw Mazur, Daniele Reda, John-Mark Allen, Vinh-Dieu Lam, Alex Bewley, and Amar Shah. Learning to Drive in a Day. jul 2018. URL http://arxiv.org/abs/1807.00412.

Zachary Kenton, Angelos Filos, Owain Evans, and Yarin Gal. Generalizing from a few environments in safety-critical reinforcement learning. 2019.

Diederik P. Kingma and Jimmy Lei Ba. Adam: a Method for Stochastic Optimization. *International Conference on Learning Representations 2015*, 2015. ISSN 09252312. doi: http://doi.acm.org.ezproxy.lib.ucf.edu/10.1145/1830483.1830503.

Diederik P Kingma and Max Welling. Auto-Encoding Variational Bayes. (Ml):1–14, 2013. ISSN 1312.6114v10. doi: 10.1051/0004-6361/201527329. URL http://arxiv.org/abs/1312.6114.

Victoria Krakovna, Laurent Orseau, Ramana Kumar, Miljan Martic, and Shane Legg. Penalizing side effects using stepwise relative reachability. 2018.

Anders Krogh and John A Hertz. A Simple Weight Decay Can Improve Generalization. In *NeurIPS*, 1991.

Joel Lehman. *Evolution through the Search for Novelty*. PhD thesis, 2012. URL http://joellehman.com/lehman-dissertation.pdf.

Jan Leike, Miljan Martic, Victoria Krakovna, Pedro A Ortega, Tom Everitt, Andrew Lefrancq, Laurent Orseau, and Shane Legg. AI Safety Gridworlds. 2017.

Jan Leike, David Krueger, Tom Everitt, Miljan Martic, Vishal Maini, and Shane Legg. Scalable agent alignment via reward modeling: a research direction. nov 2018. URL http://arxiv.org/abs/1811.07871.

Laurent Lessard, Benjamin Recht, and Andrew Packard. Analysis and Design of Optimization Algorithms via Integral Quadratic Constraints, 2014.

Sergey Levine. CS 285 at UC Berkeley: Deep Reinforcement Learning, 2020. URL http://rail.eecs.berkeley.edu/deeprlcourse/.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-End Training of Deep Visuomotor Policies. *Journal of Machine Learning Research*, 17:1–40, 2016. ISSN 15337928. doi: 10.1007/s13398-014-0173-7.2.

Liang-Liang Xie and Lei Guo. How much uncertainty can be dealt with by feedback? *IEEE Transactions on Automatic Control*, 45(12):2203–2217, dec 2000. ISSN 2334-3303. doi: 10.1109/9.895559.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations*, 2016. URL http://arxiv.org/abs/1509.02971.

Zachary C. Lipton and Jacob Steinhardt. Troubling Trends in Machine Learning Scholarship. *arXiv*, 2018. URL http://arxiv.org/abs/1807.03341.

Zachary C. Lipton, Jianfeng Gao, Lihong Li, Jianshu Chen, and Li Deng. Combating Deep Reinforcement Learning's Sisyphean Curse with Intrinsic Fear. In *ICLR*, pages 1–9, 2017. ISBN 2004012439. URL http://arxiv.org/abs/1611.01211.

Guan-Horng Liu and Evangelos A Theodorou. Deep Learning Theory Review: An Optimal Control and Dynamical Systems Perspective, 2019.

Miao Liu, Marlos C Machado, Gerald Tesauro, and Murray Campbell. The Eigenoption-Critic Framework. *NIPS Hierarchical RL Workshop*, 2017. URL http://arxiv.org/abs/1712.04065.

Yongshuai Liu, Jiaxin Ding, and Xin Liu. IPO: Interior-point Policy Optimization under Constraints. *arXiv preprint arXiv:1910.09615*, 2019.

Manuel Lopes, Tobias Lang, Marc Toussaint, and Py Oudeyer. Exploration in model-based reinforcement learning by empirically estimating learning progress. *... Processing Systems (NIPS ...*, (i):1–10, 2012. ISSN 10495258. URL http://hal.inria.fr/hal-00755248/.

Marlos C Machado, Marc G Bellemare, and Michael Bowling. A Laplacian Framework for Option Discovery in Reinforcement Learning. 2017a.

Marlos C Machado, Clemens Rosenbaum, Xiaoxiao Guo, Miao Liu, Gerald Tesauro, and Murray Campbell. Eigenoption Discovery Through the Deep Successor Representation. pages 1–20, 2017b.

Dhruv Mahajan, Ross Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens van der Maaten. Exploring the Limits of Weakly Supervised Pretraining. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11206 LNCS:185–201, may 2018. URL http://arxiv.org/abs/1805.00932.

Karl Mason and Santiago Grijalva. A Review of Reinforcement Learning for Autonomous Building Energy Management. mar 2019. URL http://arxiv.org/abs/1903.05196.

Elliot Meyerson, Joel Lehman, and Risto Miikkulainen. Learning Behavior Characterizations for Novelty Search. In *GECCO*, 2016. doi: 10.1145/2908812.2908929. URL ftp://www.cs.utexas.edu/pub/neural-nets/papers/meyerson.gecco16.pdf.

Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Deborah Raji, and Timnit Gebru. Model Cards for Model Reporting. In *FAT* '19: Conference on Fairness, Accountability, and Transparency*, oct 2019. URL https://doi.org/10.1145/3287560.3287596.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. *arXiv preprint arXiv: ...*, pages 1–9, 2013. ISSN 0028-0836. doi: 10.1038/nature14236. URL http://arxiv.org/abs/1312.5602.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei a Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. ISSN 0028-0836. doi: 10.1038/nature14236. URL http://dx.doi.org/10.1038/nature14236.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Tim Harley, Timothy P Lillicrap, David Silver, and Koray Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 1928–1937. JMLR.org, 2016.

Shakir Mohamed and Danilo J Rezende. Variational Information Maximisation for Intrinsically Motivated Reinforcement Learning. In *Proceedings of the 29th Conference on Neural Information Processing Systems (NIPS 2015)*, 2015.

Teodor Mihai Moldovan and Pieter Abbeel. Safe Exploration in Markov Decision Processes. *Proceedings of the 29th International Conference on Machine Learning*, 2012. URL http://arxiv.org/abs/1205.4810.

Randall Munroe. Machine Learning. *XKCD*, 2017. URL https://xkcd.com/1838/.

Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, Shane Legg, Volodymyr Mnih, Koray Kavukcuoglu, and David Silver. Massively Parallel Methods for Deep Reinforcement Learning. Technical report, 2015. URL http://arxiv.org/abs/1507.04296.

NASA. NPR 8715.3D: NASA General Safety Program Requirements, 2017. URL https://nodis3.gsfc.nasa.gov/displayDir.cfm?Internal{_}ID=N{_}PR{_}8715{_}003D{_}.

Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations : Theory and application to reward shaping. *Sixteenth International Conference on Machine Learning*, 3:278–287, 1999. doi: 10.1.1.48.345.

Alex Nichol, Vicki Pfau, Christopher Hesse, Oleg Klimov, and John Schulman. Gotta Learn Fast: A New Benchmark for Generalization in RL. *arXiv*, apr 2018. URL http://arxiv.org/abs/1804.03720.

Robert Nishihara, Laurent Lessard, Benjamin Recht, Andrew Packard, and Michael I Jordan. A General Analysis of the Convergence of ADMM, 2015.

Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

Junhyuk Oh, Guo Xiaoxiao, Lee Honglak, Lewis Richard, and Singh Satinder. Action-Conditional Video Prediction using Deep Networks in Atari Games. In *NIPS 2015*, 2015.

OpenAI. How to Train Your OpenAI Five, 2019. URL https://openai.com/blog/how-to-train-your-openai-five/.

OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning Dexterous In-Hand Manipulation. aug 2018. URL http://arxiv.org/abs/1808.00177.

OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving Rubik's Cube with a Robot Hand. *arXiv*, oct 2019. URL http://arxiv.org/abs/1910.07113.

Georg Ostrovski, Marc G. Bellemare, Aaron van den Oord, and Remi Munos. Count-Based Exploration with Neural Density Models. *ICML*, mar 2017. URL http://arxiv.org/abs/1703.01310.

Pierre-Yves Oudeyer and Frederic Kaplan. How can we define intrinsic motivation? *. 2008.

Matteo Papini, Matteo Pirotta, and Marcello Restelli. Smoothing Policies and Safe Policy Gradients. 2019.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H Wallach, H Larochelle, A Beygelzimer, F d\textquotesingle Alché-Buc, E Fox, and R Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

Santiago Paternain, Luiz Chamon, Miguel Calvo-Fullana, and Alejandro Ribeiro. Constrained Reinforcement Learning Has Zero Duality Gap. In *Advances in Neural Information Processing Systems*, pages 7553–7563, 2019.

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven Exploration by Self-supervised Prediction. In *ICML*, may 2017. URL `http://arxiv.org/abs/1705.05363`.

Jason Pazis and Ronald Parr. PAC Optimal Exploration in Continuous Space Markov Decision Processes. 2013.

Martin Pecka and Tomas Svoboda. Safe Exploration Techniques for Reinforcement Learning An Overview. 2014. doi: 10.1007/978-3-319-13823-7_31.

Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Edward Lee, Jie Tan, and Sergey Levine. Learning Agile Robotic Locomotion Skills by Imitating Animals. In *Robotics: Science and Systems*, 2020. doi: 10.15607/RSS.2020.XVI.064.

Jan Peters and Stefan Schaal. Natural Actor-Critic. *Neurocomputing*, 71(7-9):1180–1190, 2008. ISSN 09252312. doi: 10.1016/j.neucom.2007.11.026.

Tu-Hoa Pham, Giovanni De Magistris, Don Joven Agravante, Subhajit Chaudhury, Asim Munawar, and Ryuki Tachibana. Constrained Exploration and Recovery from Experience Shaping. 2018.

Matteo Pirotta, Marcello Restelli, and Daniele Calandriello. Safe Policy Iteration. *Proceedings of the 30th International Conference on Machine Learning*, 28, 2013.

Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob Mcgrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, Vikash Kumar, and Wojciech Zaremba. Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research. 2018. URL `https://arxiv.org/pdf/1802.09464.pdf`.

John C Platt and Alan H Barr. Constrained differential optimization. In *Neural Information Processing Systems*, pages 612–621, 1988.

Michael J D Powell. A method for nonlinear constraints in minimization problems. *Optimization*, pages 283–298, 1969.

Doina Precup. Temporal Abstraction in Reinforcement Learning. *Arbor Ciencia Pensamiento Y Cultura*, (February), 2000. ISSN 1308-0911. doi: 10.16953/deusbed.74839.

Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, 1999. ISSN 08936080. doi: 10.1016/S0893-6080(98)00116-6.

Matthew Rahtz. Lessons Learned Reproducing a Deep Reinforcement Learning Paper, apr 2018. URL `http://amid.fish/reproducing-deep-rl`.

Deborah Inioluwa Raji, Andrew Smart, Rebecca N White Google Margaret Mitchell Google Timnit Gebru Google Ben Hutchinson Google Jamila Smith-Loud Google Daniel Theron Google Parker Barnes Google, Rebecca N White, Margaret Mitchell, Timnit Gebru, Ben Hutchinson, Jamila Smith-Loud, Daniel Theron, and Parker Barnes. Closing the AI Accountability Gap: Defining an End-to-End Framework for Internal Algorithmic Auditing ACM Reference Format. In *ACM FAT* (Fariness, Accountability and Transparency) 2020*, jan 2020. URL `https://doi.org/10.1145/3351095.3372873`.

Marvin. Rausand. *Reliability of Safety-Critical Systems : Theory and Applications.* Wiley, 2014. ISBN 9781118112724. URL `https://www.wiley.com/en-us/Reliability+of+Safety+Critical+Systems{%}3A+Theory+and+Applications-p-9781118112724`.

Alex Ray, Joshua Achiam, and Dario Amodei. Benchmarking Safe Exploration in Deep Reinforcement Learning. 2019.

Dave (New England Chapter of the System Safety Society) Rice. System Safety : A Science and Technology Primer. Technical report, The New England Chapter of the System Safety Society, 2002. URL `http://www.system-safety.org/resources/SS{_}primer{_}4{_}02.pdf`.

Florian Richter, Ryan K. Orosco, and Michael C. Yip. Open-Sourced Reinforcement Learning Environments for Surgical Robotics. mar 2019. URL `http://arxiv.org/abs/1903.02090`.

Sebastian Ruder. An overview of gradient descent optimization algorithms. *Web Page*, pages 1–12, 2016.

Tim Salimans and Diederik P. Kingma. Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks. *Advances in Neural Information Processing Systems*, pages 901–909, feb 2016. URL `http://arxiv.org/abs/1602.07868`.

William Saunders, Girish Sastry, Andreas Stuhlmueller, and Owain Evans. Trial without Error: Towards Safe Reinforcement Learning via Human Intervention. 2017.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal Value Function Approximators. *Proceedings of The 32nd International Conference on Machine Learning*, pages 1312–1320, 2015. ISSN 1938-7228. URL `http://jmlr.org/proceedings/papers/v37/schaul15.html`.

Jürgen Schmidhuber. Curious Model-Building Control Systems. *International Joint Conference on Neural Networks*, 2:1458–1463, 1991. ISSN 0953-5314. doi: 10.1109/IJCNN.1991.170605.

John Schulman. The Nuts and Bolts of Deep RL Research. Technical report, 2016a.

John Schulman. *Optimizing Expectations: From Deep Reinforcement Learning to Stochastic Computation Graphs*. PhD thesis, EECS Department, University of California, Berkeley, dec 2016b. URL http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-217.html.

John Schulman, Philipp Moritz, Michael Jordan, and Pieter Abbeel. Trust Region Policy Optimization. *International Conference on Machine Learning*, 2015.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-Dimensional Continuous Control Using Generalized Advantage Estimation. In *ICLR*, 2016.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv*, 2017. URL http://arxiv.org/abs/1707.06347.

Rohin Shah, Dmitrii Krasheninnikov, Jordan Alexander, Pieter Abbeel, and Anca Dragan. Preferences Implicit in the State of the World. In *International Conference on Learning Representations*, 2019. URL http://arxiv.org/abs/1902.04198.

Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Safe, Multi-Agent, Reinforcement Learning for Autonomous Driving. *arXiv*, 2016. URL http://arxiv.org/abs/1610.03295.

Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-Aware Unsupervised Discovery of Skills. In *ICLR*, jul 2020. URL http://arxiv.org/abs/1907.01657.

David Silver. UCL Course on RL, 2020. URL https://www.davidsilver.uk/teaching/.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587): 484–489, 2016. ISSN 0028-0836. doi: 10.1038/nature16961. URL http://dx.doi.org/10.1038/nature16961.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. dec 2017a. URL http://arxiv.org/abs/1712.01815.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359,

oct 2017b. ISSN 0028-0836. doi: 10.1038/nature24270. URL http://www.nature.com/articles/nature24270.

R E Skelton. *Dynamic Systems Control: Linear Systems Analysis and Synthesis*. Dynamic Systems Control. John Wiley & Sons, 1988. ISBN 9780471837794. URL https://books.google.com/books?id=egFRAAAAMAAJ.

Qiang Song and Robert P Leland. An optimal control model of neural networks for constrained optimization problems. *Optimal Control Applications and Methods*, 19(5):371–376, 1998. doi: 10.1002/(SICI)1099-1514(199809/10)19:5⟨371::AID-OCA636⟩3.0.CO;2-8. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/{%}28SICI{%}291099-1514{%}28199809/10{%}2919{%}3A5{%}3C371{%}3A{%}3AAID-OCA636{%}3E3.0.CO{%}3B2-8.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Technical report, 2014.

Bradly C Stadie, Sergey Levine, and Pieter Abbeel. Incentivizing Exploration In Reinforcement Learning With Deep Predictive Models. *NeurIPS Deep RL Workshop*, jul 2015. URL http://arxiv.org/abs/1507.00814.

Adam Stooke, Joshua Achiam, and Pieter Abbeel. Responsive Safety in Reinforcement Learning by {PID} Lagrangian Methods. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 9133–9143, Virtual, 2020. PMLR. URL http://proceedings.mlr.press/v119/stooke20a.html.

Jan Storck, Sepp Hochreiter, and Jürgen Schmidhuber. Reinforcement driven information acquisition in non-deterministic environments. In *Proceedings of the International ...*, volume 2, pages 159–164, 1995. URL http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.35.8445{%}5Cnhttp://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=4E90601907ED1E7363625CFE2BE6E2A4?doi=10.1.1.35.8445{&}rep=rep1{&}type=pdf{%}5Cnhttp://scholar.google.com/scholar?hl=en{&}btnG=Search{&}q=intit.

Yi Sun, Faustino Gomez, and Jürgen Schmidhuber. Planning to be surprised: Optimal Bayesian exploration in dynamic environments. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 6830 LNAI, pages 41–51, 2011. ISBN 9783642228865. doi: 10.1007/978-3-642-22887-2_5.

Richard Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018. URL http://incompleteideas.net/book/the-book.html.

Richard S Sutton and Andrew G Barto. Introduction to Reinforcement Learning. *Learning*, 4 (1996):1–5, 1998. ISSN 10743529. doi: 10.1.1.32.7692. URL http://dl.acm.org/citation.cfm?id=551283.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and Semi-MDPs: A

Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence*, 112, 1999.

Csaba Szepesvári. Algorithms for reinforcement learning. In *Synthesis Lectures on Artificial Intelligence and Machine Learning*, volume 9, pages 1–89. Morgan & Claypool Publishers, jul 2010. ISBN 9781608454921. doi: 10.2200/S00268ED1V01Y201005AIM009.

Adrien Ali Taïga, William Fedus, Marlos C. Machado, Aaron Courville, and Marc G. Bellemare. Benchmarking Bonus-Based Exploration Methods on the Arcade Learning Environment. *arXiv*, aug 2019. URL `http://arxiv.org/abs/1908.02388`.

Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. #Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning. In *NeurIPS*, 2017.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. DeepMind Control Suite. 2018.

Chen Tessler, Daniel J Mankowitz, and Shie Mannor. Reward Constrained Policy Optimization. 2018.

Valentin Thomas, Jules Pondard, Emmanuel Bengio, Marc Sarfati, Philippe Beaudoin, Marie-Jean Meurs, Joelle Pineau, Doina Precup, and Yoshua Bengio. Independently Controllable Factors. pages 1–13, 2017.

Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *IEEE International Conference on Intelligent Robots and Systems*, 2012. ISBN 9781467317375. doi: 10.1109/IROS.2012.6386109.

Eiji Uchibe and Kenji Doya. Constrained reinforcement learning from intrinsic and extrinsic rewards. *2007 IEEE 6th International Conference on Development and Learning, ICDL*, (February):163–168, 2007. doi: 10.1109/DEVLRN.2007.4354030.

Hado van Hasselt, Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-learning. In *AAAI*, 2016. URL `http://arxiv.org/abs/1509.06461`.

Ashish Vaswani, Google Brain, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. Technical report, 2017.

Ivan Vendrov and Jeremy Nixon. Aligning Recommender Systems as Cause Area, 2019. URL `https://forum.effectivealtruism.org/posts/xzjQvqDYahigHcwgQ/aligning-recommender-systems-as-cause-area-1`.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. FeUdal Networks for Hierarchical Reinforcement Learning. (1), 2017. ISSN 1938-7228. URL `http://arxiv.org/abs/1703.01161`.

Benjamin W Wah, Tao Wang, Yi Shang, and Zhe Wu. Improving the performance of weighted

Lagrange-multiplier methods for nonlinear constrained optimization. *Information Sciences*, 124(1-4):241–272, 2000.

Weixun Wang, Junqi Jin, Jianye Hao, Chunjie Chen, Chuan Yu, Weinan Zhang, Jun Wang, Xiaotian Hao, Yixi Wang, Han Li, Jian Xu, and Kun Gai. Learning Adaptive Display Exposure for Real-Time Advertising. In *28th ACM International Conference on Information and Knowledge Management*, 2018. URL http://arxiv.org/abs/1809.03149.

Ziyu Wang, Nando de Freitas, and Marc Lanctot. Dueling Network Architectures for Deep Reinforcement Learning. *arXiv*, (9):1–16, 2016. URL http://arxiv.org/abs/1511.06581.

Christopher J C H Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992. ISSN 0885-6125. doi: 10.1007/BF00992698. URL http://link.springer.com/10.1007/BF00992698.

Lilian Weng. A (Long) Peek into Reinforcement Learning, feb 2018. URL https://lilianweng.github.io/lil-log/2018/02/19/a-long-peek-into-reinforcement-learning.html.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation, 2016. URL https://research.google/pubs/pub45610/.

Yuhuai Wu, Elman Mansimov, Shun Liao, Alec Radford, and John Schulman. OpenAI Baselines: ACKTR & A2C, aug 2017. URL https://openai.com/blog/baselines-acktr-a2c/.

Tsung-Yen Yang, Justinian Rosca, Karthik Narasimhan, and Peter J Ramadge. Projection-Based Constrained Policy Optimization. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rke3TJrtPS.

Appendix A

Material from Spinning Up

A.1 Spinning Up as a Deep RL Researcher

Originally published October 13th, 2018.

If you're an aspiring deep RL researcher, you've probably heard all kinds of things about deep RL by this point. You know that it's hard and it doesn't always work [Irpan, 2018]. That even when you're following a recipe, reproducibility is a challenge [Islam et al., 2017, Henderson et al., 2018]. And that if you're starting from scratch, the learning curve is incredibly steep [Rahtz, 2018]. It's also the case that there are a lot of great resources out there [Schulman, 2016a, Abbeel et al., 2017, Levine, 2020, Silver, 2020], but the material is new enough that there's not a clear, well-charted path to mastery. The goal of this column is to help you get past the initial hurdle, and give you a clear sense of how to spin up as a deep RL researcher. In particular, this will outline a useful curriculum for increasing raw knowledge, while interleaving it with the odds and ends that lead to better research.

A.1.1 The Right Background

Build up a solid mathematical background. From probability and statistics, feel comfortable with random variables, Bayes' theorem, chain rule of probability, expected values, standard deviations, and importance sampling. From multivariate calculus, understand gradients and (optionally, but it'll help) Taylor series expansions.

Build up a general knowledge of deep learning. You don't need to know every single special trick and architecture, but the basics help. Know about standard architectures (MLP, vanilla RNN, LSTM, GRU [Chung et al., 2014], conv layers, resnets [He et al., 2016], attention mechanisms [Bahdanau et al., 2015, Vaswani et al., 2017]), common regularizers (weight decay [Krogh and Hertz, 1991], dropout [Srivastava et al., 2014]), normalization (batch norm [Ioffe and Szegedy, 2015], layer norm [Ba et al., 2016], weight norm [Salimans and Kingma, 2016]), and optimizers (SGD, momentum SGD [Qian, 1999], Adam [Kingma and Ba, 2015], others [Ruder, 2016]). Know what the reparameterization trick is [Kingma and Welling, 2013].

Become familiar with at least one deep learning library. Tensorflow [Abadi et al., 2016] or PyTorch [Paszke et al., 2019] would be a good place to start. You don't need to know how to do everything, but you should feel pretty confident in implementing a simple program to do supervised learning.

Get comfortable with the main concepts and terminology in RL. Know what states, actions, trajectories, policies, rewards, value functions, and action-value functions are. If you're unfamiliar, Spinning Up ships with an introduction to this material; it's also worth checking out the RL-Intro from the OpenAI Hackathon[1], or the exceptional and thorough overview by Lilian Weng [Weng, 2018]. Optionally, if you're the sort of person who enjoys mathematical theory, study up on the math of monotonic improvement theory [Schulman, 2016b] (which forms the basis for advanced policy gradient algorithms), or classical RL algorithms [Szepesvári, 2010] (which despite being superseded by deep RL algorithms, contain valuable insights that sometimes drive new research).

A.1.2 Learn by Doing

Write your own implementations. You should implement as many of the core deep RL algorithms from scratch as you can, with the aim of writing the shortest correct implementation of each. This is by far the best way to develop an understanding of how they work, as well as intuitions for their specific performance characteristics.

Simplicity is critical. You should organize your efforts so that you implement the simplest algorithms first, and only gradually introduce complexity. If you start off trying to build something with too many moving parts, odds are good that it will break and you'll lose weeks trying to debug it. This is a common failure mode for people who are new to deep RL, and if you find yourself stuck in it, don't be discouraged—but do try to change tack and work on a simpler algorithm instead, before returning to the more complex thing later.

Which algorithms? You should probably start with vanilla policy gradient (also called REINFORCE) [Duan et al., 2016], DQN [Mnih et al., 2013], A2C (the synchronous version of A3C) [Mnih et al., 2016, Wu et al., 2017], PPO (the variant with the clipped objective) [Schulman et al., 2017], and DDPG [Lillicrap et al., 2016], approximately in that order. The simplest versions of all of these can be written in just a few hundred lines of code (ballpark 250-300), and some of them even less (for example, a no-frills version of VPG can be written in about 80 lines). Write single-threaded code before you try writing parallelized versions of these algorithms. (Do try to parallelize at least one.)

Focus on understanding. Writing working RL code requires clear, detail-oriented understanding of the algorithms. This is because broken RL code almost always fails silently, where the code appears to run fine except that the agent never learns how to solve the task. Usually the problem is that something is being calculated with the wrong equation, or on the wrong distribution, or data is being piped into the wrong place. Sometimes the only way to find these bugs is to read the code with a critical eye, know exactly what it should be doing, and

[1] `https://github.com/jachiam/rl-intro`

find where it deviates from the correct behavior. Developing that knowledge requires you to engage with both academic literature and other existing implementations (when possible), so a good amount of your time should be spent on that reading.

What to look for in papers: When implementing an algorithm based on a paper, scour that paper, especially the ablation analyses and supplementary material (where available). The ablations will give you an intuition for what parameters or subroutines have the biggest impact on getting things to work, which will help you diagnose bugs. Supplementary material will often give information about specific details like network architecture and optimization hyperparameters, and you should try to align your implementation to these details to improve your chances of getting it working.

But don't overfit to paper details. Sometimes, the paper prescribes the use of more tricks than are strictly necessary, so be a bit wary of this, and try out simplifications where possible. For example, the original DDPG paper suggests a complex neural network architecture and initialization scheme, as well as batch normalization. These aren't strictly necessary, and some of the best-reported results for DDPG use simpler networks. As another example, the original A3C paper uses asynchronous updates from the various actor-learners, but it turns out that synchronous updates work about as well.

Don't overfit to existing implementations either. Study existing implementations [Dhariwal et al., 2017, Duan et al., 2016] for inspiration, but be careful not to overfit to the engineering details of those implementations. RL libraries frequently make choices for abstraction that are good for code reuse between algorithms, but which are unnecessary if you're only writing a single algorithm or supporting a single use case.

Iterate fast in simple environments. To debug your implementations, try them with simple environments where learning should happen quickly, like CartPole-v0, InvertedPendulum-v0, FrozenLake-v0, and HalfCheetah-v2 (with a short time horizon—only 100 or 250 steps instead of the full 1000) from the OpenAI Gym [Brockman et al., 2016]. Don't try to run an algorithm in Atari or a complex Humanoid environment if you haven't first verified that it works on the simplest possible toy task. Your ideal experiment turnaround-time at the debug stage is <5 minutes (on your local machine) or slightly longer but not much. These small-scale experiments don't require any special hardware, and can be run without too much trouble on CPUs.

If it doesn't work, assume there's a bug. Spend a lot of effort searching for bugs before you resort to tweaking hyperparameters: usually it's a bug. Bad hyperparameters can significantly degrade RL performance, but if you're using hyperparameters similar to the ones in papers and standard implementations, those will probably not be the issue. Also worth keeping in mind: sometimes things will work in one environment even when you have a breaking bug, so make sure to test in more than one environment once your results look promising.

Measure everything. Do a lot of instrumenting to see what's going on under-the-hood. The more stats about the learning process you read out at each iteration, the easier it is to debug—after all, you can't tell it's broken if you can't see that it's breaking. I personally like to look at the mean/std/min/max for cumulative rewards, episode lengths, and value

function estimates, along with the losses for the objectives, and the details of any exploration parameters (like mean entropy for stochastic policy optimization, or current epsilon for epsilon-greedy as in DQN). Also, watch videos of your agent's performance every now and then; this will give you some insights you wouldn't get otherwise.

Scale experiments when things work. After you have an implementation of an RL algorithm that seems to work correctly in the simplest environments, test it out on harder environments. Experiments at this stage will take longer—on the order of somewhere between a few hours and a couple of days, depending. Specialized hardware—like a beefy GPU or a 32-core machine—might be useful at this point, and you should consider looking into cloud computing resources like AWS or GCE.

Keep these habits! These habits are worth keeping beyond the stage where you're just learning about deep RL—they will accelerate your research!

A.1.3 Developing a Research Project

Once you feel reasonably comfortable with the basics in deep RL, you should start pushing on the boundaries and doing research. To get there, you'll need an idea for a project.

Start by exploring the literature to become aware of topics in the field. There are a wide range of topics you might find interesting: sample efficiency, exploration, transfer learning, hierarchy, memory, model-based RL, meta learning, and multi-agent, to name a few. If you're looking for inspiration, or just want to get a rough sense of what's out there, check out Spinning Up's key papers list.[2] Find a paper that you enjoy on one of these subjects—something that inspires you—and read it thoroughly. Use the related work section and citations to find closely-related papers and do a deep dive in the literature. You'll start to figure out where the unsolved problems are and where you can make an impact.

Approaches to idea-generation: There are a many different ways to start thinking about ideas for projects, and the frame you choose influences how the project might evolve and what risks it will face. Here are a few examples:

Frame 1: Improving on an Existing Approach. This is the incrementalist angle, where you try to get performance gains in an established problem setting by tweaking an existing algorithm. Reimplementing prior work is super helpful here, because it exposes you to the ways that existing algorithms are brittle and could be improved. A novice will find this the most accessible frame, but it can also be worthwhile for researchers at any level of experience. While some researchers find incrementalism less exciting, some of the most impressive achievements in machine learning have come from work of this nature.

Because projects like these are tied to existing methods, they are by nature narrowly scoped and can wrap up quickly (a few months), which may be desirable (especially when starting out as a researcher). But this also sets up the risks: it's possible that the tweaks you have in

[2] `https://spinningup.openai.com/en/latest/spinningup/keypapers.html`

mind for an algorithm may fail to improve it, in which case, unless you come up with more tweaks, the project is just over and you have no clear signal on what to do next.

Frame 2: Focusing on Unsolved Benchmarks. Instead of thinking about how to improve an existing method, you aim to succeed on a task that no one has solved before. For example: achieving perfect generalization from training levels to test levels in the Sonic domain or Gym Retro [Nichol et al., 2018]. When you hammer away at an unsolved task, you might try a wide variety of methods, including prior approaches and new ones that you invent for the project. It is possible for a novice to approch this kind of problem, but there will be a steeper learning curve.

Projects in this frame have a broad scope and can go on for a while (several months to a year-plus). The main risk is that the benchmark is unsolvable without a substantial breakthrough, meaning that it would be easy to spend a lot of time without making any progress on it. But even if a project like this fails, it often leads the researcher to many new insights that become fertile soil for the next project.

Frame 3: Create a New Problem Setting. Instead of thinking about existing methods or current grand challenges, think of an entirely different conceptual problem that hasn't been studied yet. Then, figure out how to make progress on it. For projects along these lines, a standard benchmark probably doesn't exist yet, and you will have to design one. This can be a huge challenge, but it's worth embracing—great benchmarks move the whole field forward.

Problems in this frame come up when they come up—it's hard to go looking for them.

Avoid reinventing the wheel. When you come up with a good idea that you want to start testing, that's great! But while you're still in the early stages with it, do the most thorough check you can to make sure it hasn't already been done. It can be pretty disheartening to get halfway through a project, and only then discover that there's already a paper about your idea. It's especially frustrating when the work is concurrent, which happens from time to time! But don't let that deter you—and definitely don't let it motivate you to plant flags with not-quite-finished research and over-claim the merits of the partial work. Do good research and finish out your projects with complete and thorough investigations, because that's what counts, and by far what matters most in the long run.

A.1.4 Doing Rigorous Research in RL

Now you've come up with an idea, and you're fairly certain it hasn't been done. You use the skills you've developed to implement it and you start testing it out on standard domains. It looks like it works! But what does that mean, and how well does it have to work to be important? This is one of the hardest parts of research in deep RL. In order to validate that your proposal is a meaningful contribution, you have to rigorously prove that it actually gets a performance benefit over the strongest possible baseline algorithm—whatever currently achieves SOTA (state of the art) on your test domains. If you've invented a new test domain, so there's no previous SOTA, you still need to try out whatever the most reliable algorithm

in the literature is that could plausibly do well in the new test domain, and then you have to beat that.

Set up fair comparisons. If you implement your baseline from scratch—as opposed to comparing against another paper's numbers directly—it's important to spend as much time tuning your baseline as you spend tuning your own algorithm. This will make sure that comparisons are fair. Also, do your best to hold "all else equal" even if there are substantial differences between your algorithm and the baseline. For example, if you're investigating architecture variants, keep the number of model parameters approximately equal between your model and the baseline. Under no circumstances handicap the baseline! It turns out that the baselines in RL are pretty strong, and getting big, consistent wins over them can be tricky or require some good insight in algorithm design.

Remove stochasticity as a confounder. Beware of random seeds making things look stronger or weaker than they really are, so run everything for many random seeds (at least 3, but if you want to be thorough, do 10 or more). This is really important and deserves a lot of emphasis: deep RL seems fairly brittle with respect to random seed in a lot of common use cases. There's potentially enough variance that two different groups of random seeds can yield learning curves with differences so significant that they look like they don't come from the same distribution at all (see figure 10 from Islam et al. [2017]).

Run high-integrity experiments. Don't just take the results from the best or most interesting runs to use in your paper. Instead, launch new, final experiments—for all of the methods that you intend to compare (if you are comparing against your own baseline implementations)—and precommit to report on whatever comes out of that. This is to enforce a weak form of preregistration[3]: you use the tuning stage to come up with your hypotheses, and you use the final runs to come up with your conclusions.

Check each claim separately. Another critical aspect of doing research is to run an ablation analysis. Any method you propose is likely to have several key design decisions—like architecture choices or regularization techniques, for instance—each of which could separately impact performance. The claim you'll make in your work is that those design decisions collectively help, but this is really a bundle of several claims in disguise: one for each such design element. By systematically evaluating what would happen if you were to swap them out with alternate design choices, or remove them entirely, you can figure out how to correctly attribute credit for the benefits your method confers. This lets you make each separate claim with a measure of confidence, and increases the overall strength of your work.

[3]https://www.cos.io/initiatives/prereg