Université de Montréal

On Learning and Generalization in Unstructured Task Spaces

par

Bhairav Mehta

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en Informatique

August 21, 2020

Université de Montréal

Faculté des arts et des sciences

Ce mémoire intitulé

On Learning and Generalization in Unstructured Task Spaces

présenté par

Bhairav Mehta

a été évalué par un jury composé des personnes suivantes :

Pierre-Luc Bacon
(président-rapporteur)

Liam Paull
(directeur de recherche)

Christopher J. Pal
(codirecteur)

Hugo Larochelle
(membre du jury)

Résumé

L'apprentissage robotique est incroyablement prometteur pour l'intelligence artificielle incarnée, avec un apprentissage par renforcement apparemment parfait pour les robots du futur: apprendre de l'expérience, s'adapter à la volée et généraliser à des scénarios invisibles.

Cependant, notre réalité actuelle nécessite de grandes quantités de données pour former la plus simple des politiques d'apprentissage par renforcement robotique, ce qui a suscité un regain d'intérêt de la formation entièrement dans des simulateurs de physique efficaces. Le but étant l'intelligence incorporée, les politiques formées à la simulation sont transférées sur du matériel réel pour évaluation; cependant, comme aucune simulation n'est un modèle parfait du monde réel, les politiques transférées se heurtent à l'écart de transfert *sim2real*: les erreurs se sont produites lors du déplacement des politiques des simulateurs vers le monde réel en raison d'effets non modélisés dans des modèles physiques inexacts et approximatifs.

La randomisation de domaine - l'idée de randomiser tous les paramètres physiques dans un simulateur, forçant une politique à être robuste aux changements de distribution - s'est avérée utile pour transférer des politiques d'apprentissage par renforcement sur de vrais robots. En pratique, cependant, la méthode implique un processus difficile, d'essais et d'erreurs, montrant une grande variance à la fois en termes de convergence et de performances. Nous introduisons Active Domain Randomization, un algorithme qui implique l'apprentissage du curriculum dans des espaces de tâches *non structurés* (espaces de tâches où une notion de *difficulté* - tâches intuitivement faciles ou difficiles - n'est pas facilement disponible). La randomisation de domaine active montre de bonnes performances sur le pourrait utiliser zero shot sur de vrais robots. La thèse introduit également d'autres variantes de l'algorithme, dont une qui permet d'incorporer un a priori de sécurité et une qui s'applique au domaine de l'apprentissage par méta-renforcement. Nous analysons également l'apprentissage du curriculum dans une perspective d'optimisation et tentons de justifier les avantages de l'algorithme en étudiant les interférences de gradient.

Mots Clés: domain randomization, robotics, transfer learning, curriculum learning

Abstract

Robotic learning holds incredible promise for embodied artificial intelligence, with reinforcement learning seemingly a strong candidate to be the *software* of robots of the future: learning from experience, adapting on the fly, and generalizing to unseen scenarios.

However, our current reality requires vast amounts of data to train the simplest of robotic reinforcement learning policies, leading to a surge of interest of training entirely in efficient physics simulators. As the goal is embodied intelligence, policies trained in simulation are transferred onto real hardware for evaluation; yet, as no simulation is a perfect model of the real world, transferred policies run into the *sim2real* transfer gap: the errors accrued when shifting policies from simulators to the real world due to unmodeled effects in inaccurate, approximate physics models.

Domain randomization - the idea of randomizing all physical parameters in a simulator, forcing a policy to be robust to distributional shifts - has proven useful in transferring reinforcement learning policies onto real robots. In practice, however, the method involves a difficult, trial-and-error process, showing high variance in both convergence and performance. We introduce Active Domain Randomization, an algorithm that involves curriculum learning in *unstructured* task spaces (task spaces where a notion of *difficulty* - intuitively easy or hard tasks - is not readily available). Active Domain Randomization shows strong performance on zero-shot transfer on real robots. The thesis also introduces other variants of the algorithm, including one that allows for the incorporation of a safety prior and one that is applicable to the field of Meta-Reinforcement Learning. We also analyze curriculum learning from an optimization perspective and attempt to justify the benefit of the algorithm by studying gradient interference.

Keywords: domain randomization, robotics, transfer learning, curriculum learning

Contents

List of tables

List of figures

Liste des sigles et des abréviations

RL	*Reinforcement Learning*
DR	*Domain Randomization*
VDR	*Vanilla Domain Randomization*
DRL	*Deep Reinforcement Learning*
ADR	*Active Domain Randomization*
MDP	*Markov Decision Process*
SVGD	*Stein Variational Gradient Descent*
SVPG	*Stein Variational Policy Gradient*
MES	*main engine strength*
DoF	*degrees of freedom*
RBF	*Radial Basis Function*

A2C	*Advantage Actor-Critic*
UDR	*Uniform Domain Randomization*
BO	*Bayesian Optimization*
RARL	*Robust Adversarial Reinforcement Learning*
MaxEnt RL	*Maximum Entropy Reinforcement Learning*
GAIL	*Generative Adversarial Reinforcement Learning*
GP	*Gaussian Process*
CLRL	*Curriculum Learning for Reinforcement Learning*

Remerciements

Most of this work is the product of countless conversations I have been lucky to have had with dozens of my closest collaborators, friends, and mentors. Thank you to:

- **The Mila Slack** And all of the brilliant people that come along with it!
- **IVADO, Jane Street, Depth First Learning, UdeM, Mila, and Duckietown** for providing me with financial support to pursue research and teaching without stress.
- **IFT-6760A and Guillaume Rabusseau** who showed me that hard things can be taught incredibly well.
- **Jacob Miller** for answering every question I have ever had about teaching, physics, and grad school.
- **Kris Sankaran** for our discussions on how to use my work to better our community and society.
- **My REAL labmates** Sai, Gunshi, Nithin, Vincent, Dhaivat, Krishna, Breandan, Mark, and Dishank, for stimulating discussions, (not-so) "awkward" group meetings, and board game sessions during the Montreal winters.
- **My Duckietown family** What an incredible experience to continue to be a part of!
- **My mentors** Florian and Maxime, who helped me get things moving on countless projects, and by showing me the ropes on everything from simulators to Duckietown.
- **My closest collaborators** Sharath - thank you for your incredible hard work; Julian - thank you for being the most persevering and optimistic researcher I have come across (and, for the book recommendations and countless enjoyable conversations); Manfred - thank you for introducing me to graphs and eight-dollar meals at Polytechnique... and, for asking the toughest questions about my worst ideas.
- **My advisers** Liam Paull, Christopher Pal, and Andrea Censi, who, during my time at Mila, helped me grow into a (sort-of) productive researcher, but more importantly, a curious mind.

Chapter 1

Introduction

Machine learning often concerns itself with learning information from *data*. In regression settings, we often want to predict things like housing prices, or, as in classification, whether this particular image is a dog or cat. However, implicit in all main branches of machine learning - supervised, unsupervised, and reinforcement - is the notion of a data *distribution*. Despite the fact that we may never see it, there *is* a data distribution - some distribution over real estate pricing, animal images, etc. from which the data we observe is sampled from.

The machine learning paradigm comprises of two, coarsely-defined stages: training and testing. During training, algorithms aim to extract patterns in data: mapping, for example, from images to labels or from states to actions. The data in this stage comes with some supervisory signal, enabling an algorithm to *learn*. Learning consists of updating parameters via incorporation of the signal into an algorithm's decision-making process.

During testing, the algorithm, using its parameters acquired from the training stage, generates open-loop predictions given data. Often, at this stage, the supervisory signal is known only to the human experimenter, enabling fair comparison across algorithms and approaches.

This thesis generally concerns itself with problems under the jurisdiction of *transfer learning*. The previous paragraphs, which describe the theoretical setup of most machine learning problems, require the existence of a data distribution $p(x)$. $p(x)$ can be the distribution of handwritten digits, natural images, or states seen by a policy in a reinforcement learning environment. The true $p(x)$ is often high-dimensional, vaguely-defined, and most importantly, unattainable. However, its existence is guaranteed in most machine learning problem settings and is often held constant across training and testing. Transfer learning comes into play when it doesn't.

What does it mean to *change* a distribution? Mathematically, it is simple: given some random variable X (representing our data), we train an algorithm on data distribution $p(x)$

which has some support. Then, we test our output on a different distribution $q(x)$, one with a different support than p.

While transfer learning can be simple shifts in data distribution (i.e the label distribution of cats vs. dogs being $60\% - 40\%$ at testing, when equal at training), many real-world problems can see dramatic changes in distribution. Transfer learning is precisely the issue that makes things like deep learning-based autonomous driving [**31**] and in-the-wild chatbots [**88**] so difficult to deploy.

Throughout much of this thesis, the transfer learning problem studied is *sim2real transfer*: a robotics-centric problem that occurs when training policies fully in simulation while testing them on real robotic hardware. This problem is introduced in its entirety in Chapter 2.

The rest of this thesis is organized as follows:

- **Section 1.1** covers mathematical preliminaries common to the work covered in this thesis.
- **Section 1.2** covers related work in the space of reinforcement learning, robot learning, multi-task learning, and curriculum learning.
- **Section 1.3** covers a phenomenon of deep learning and generalization in supervised learning, one which is exploited by the rest of the work covered in the thesis.
- **Chapter 2** covers *Active Domain Randomization* [**46**], a novel algorithm which aims to address the *sim2real* transfer problem.
- **Chapter 3** covers optimization characteristics of curriculum learning for neural networks.
- **Chapter 4** covers an application of Active Domain Randomization within the meta-reinforcement learning domain.

1.1. Preliminaries

In this section, we lay out the necessary theoretical and experimental tools to understand the remainder of the thesis. For brevity, we provide external references, and summarize only the required information below.

1.1.1. Reinforcement Learning

We consider a Reinforcement Learning (RL) framework [**76**] where some task T is defined by a Markov Decision Process (MDP) consisting of a state space S, action space A, state transition function $P : S \times A \mapsto S$, reward function $r : S \times A \mapsto \mathbf{R}$, and discount factor $\gamma \in (0,1)$. The goal for an agent trying to solve T is to learn a policy π with parameters θ that maximizes the expected total discounted reward. We define a rollout $\tau = (s_0, a_0 ..., s_T, a_T)$ to be the sequence of states s_t and actions $a_t \sim \pi(a_t|s_t)$ executed by a policy π in the environment.

To maximize the reward, we maximize a utility function, $J(\pi_\theta)$, with policy gradient methods [**77**]:

$$J(\pi_\theta) = \mathbf{E}_{s,a\sim\pi_\theta}\Big[\sum_{t=0}^{\infty}\gamma^t r(s_t, a_t)\Big] \tag{1.1.1}$$

In our work, we use various variants of policy gradient estimators (REINFORCE [**89**], REINFORCE with baseline, Actor-Critic [**37, 40**]), but in general, we can use various methods to compute the policy gradient (direction of maximum ascent): $\nabla_\theta J(\pi_\theta)$.

1.1.2. Maximum Entropy Reinforcement Learning

Oftentimes, actions in reinforcement learning policies are defined as distributions over the action distribution: $p(a|s)$. Such a decision allows us to cleanly incorporate randomness into the learning process, which is crucial for agents to explore the state spaces of their environments.

Traditionally, these action distributions are characterized by simple distributions: categorical, for discrete action spaces, or by a Gaussian in the continuous action case. Using distributions allows for the characterization of entropy, which can be roughly thought of as the *randomness* of the distribution.

$$H(X) = \mathbf{E}_X[I(X)] = -\sum_x p(x)\log p(x) \tag{1.1.2}$$

where $I(X)$ is the self-information of the random variable X.

Maximum Entropy RL (MaxEnt RL) deals with maximizing the *long-term entropy* jointly with the expected return. MaxEnt RL maximizes entropy *inside* of the expectation operator, rather than the single-step entropy (which is what is known as an entropy bonus). This can be written as:

$$\pi^* = \arg\max_\pi \ \mathbf{E}_\pi\Big[\sum_{t=0}^{\infty}\gamma^t(r_t + \alpha H)\Big] \tag{1.1.3}$$

with entropy H being controlled by temperature parameter α. The entropy H is shorthand for the entropy of the policy distribution at each timestep.

While MaxEnt RL is commonly thought of as maximizing entropy, the entropy term itself is derived from a KL-Divergence term:

$$\pi^* = \arg\max_\pi \ \mathbf{E}_\pi\Big[\sum_{t=0}^{\infty}\gamma^t(r_t + \alpha D_{KL}[\pi||\bar{\pi}])\Big] \tag{1.1.4}$$

where the KL-Divergence of π is calculated with respect to a reference policy, $\bar{\pi}$. Following [**71**], when we take $\bar{\pi}$ to be a constantly-uniform policy, we recover the MaxEnt RL formulation (up to a constant, which correctly gets dropped when taking the maximization).

Taking a uniform prior, we encourage our learned policy to maximize reward *while* acting as randomly as possible. But, if given a prior policy to follow, MaxEnt RL provides a natural way to incorporate it into the decision making and learning process.

1.1.3. Domain Randomization

Domain randomization (DR) is a technique to increase the generalization capability of policies trained in simulation. Domain Randomization (DR) requires a prescribed set of N_{rand} simulation parameters to randomize, as well as corresponding ranges to sample them from. A set of parameters is sampled from a *randomization space* $\Xi \subset \mathbf{R}^{N_{rand}}$, where each randomization parameter $\xi^{(i)}$ is bounded on a closed interval $\{\left[\xi_{low}^{(i)}, \xi_{high}^{(i)}\right]\}_{i=1}^{N_{rand}}$.

When a configuration $\xi \in \Xi$ is passed to a non-differentiable simulator S[1], it generates an environment E. At the start of each episode, the parameters are uniformly sampled from the ranges, and the environment generated from those values is used to train the agent policy π.

DR may perturb any to all elements of the task T's underlying MDP[2], with the exception of keeping R and γ constant. DR therefore generates a set of MDPs that are superficially similar, but can vary greatly in difficulty depending on the character of the randomization. Upon transfer to the target domain, the hope is that the agent policy has learned to generalize across MDPs, and sees the final domain as just another variation of parameters.

The most common instantiation of DR, UDR is summarized in Algorithm 1.

Algorithm 1 Uniform-Sampling Domain Randomization

Input: Ξ: Randomization space, S: Simulator
Initialize π_θ: agent policy, $\mathcal{T}_{rand} = \varnothing$
for each episode t **do**
    for $i = 1$ **to** N_{rand} **do** ▷ Uniformly sample parameters
        $\xi_t^{(i)} \sim U\left[\xi_{low}^{(i)}, \xi_{high}^{(i)}\right]$
    $E_t \leftarrow S(\xi_t)$ ▷ Generate randomized environments
    rollout $\tau_t \sim \pi_\theta(\cdot; E_t)$ ▷ Rollout policy in randomized environments
    $\mathcal{T}_{rand} \leftarrow \mathcal{T}_{rand} \cup \tau_t$
    for each policy update step **do** ▷ Agent policy update
        with experience buffer $\mathcal{T}_{rand}$ **update**:
            $\theta \leftarrow \theta + \nu \nabla_\theta J(\pi_\theta)$ ▷ Policy gradient update

UDR generates randomized environment instances E_t by uniformly sampling Ξ. The agent policy π is then trained on rollouts τ_t produced in randomized environments E_t.

[1]We use the assumption of non-differentability because: (A) most physics simulators are non-differentiability (B) differentiable simulators have more efficient methods of propogating reward signals that REINFORCE-like estimates.

1.1.4. Curriculum Learning

Curriculum learning focuses on a meta-optimization task, where a black-box learner is required to learn a goal task ξ_g or a distribution of tasks, $p(\xi)$. By abstracting away the inner-loop learner, curriculum learning focuses on the *sequence of tasks* to show the agent, rather than focusing on learning how to solve any particular task. Oftentimes, curriculum learning focuses on showing the learner easier versions of the goal task - for example, in a navigation setting, starting states closer to the goal may be shown to the agent first [**21, 3**], progressively getting harder until the goal task ξ_g is achieved. Curriculum learning works well with structured task spaces, or when there is a clear notion of difficulty between tasks. Learning a curriculum in an *unstructured* task space, where a notion of *easier tasks* cannot be discerned, is still an open research question.

1.1.5. Transfer Learning

When training machine learning models, we often make an assumption that the underlying data-generating distribution $p(x)$ is constant for both training and testing [**61**], even if the training and testing datasets are disjoint. However, when the data-generating distribution is not held constant between training and testing, the problem setting becomes an instantiation of transfer learning.

When training machine learning models solely on distribution $p(x)$ and testing on a different distribution $q(x)$, *zero-shot transfer* deals with problem settings where no further optimization occurs at test time using distribution $q(x)$. The model trained in the data-regime defined by $p(x)$, say a simulator, is used without tuning in the new distribution $q(x)$, which for example, can be deployed on a real-world robot.

While zero-shot transfer is often motivated through the lens of *expense* (i.e it is cheaper to train a robotic agent in a simulation than on the physical robot), many real-world applications of machine learning allow for fine-tuning in the test domain. When only a few data samples can be collected and used for optimization, the problem lies in the *few-shot transfer* regime: for example, a robotic agent, trained in simulation, may get the opportunity to execute a small number of real-world rollouts, ideally used to quickly fine-tune to the test distribution. However, in the context of RL, fine-tuning at test time requires additional policy gradient steps. Often, evaluating the reward for RL algorithms at test time can be problematic, leading to a surge of interest in zero-shot transfer learning.

1.1.6. Bayesian Optimization

In the Bayesian Optimization (BO) framework [**6**], we are concerned with the global optimization of some stationary function $f : \Phi \to \mathbf{R}$:

$$\phi^* = \arg\max_{\phi \in \Phi} f(\phi) \tag{1.1.5}$$

If given knowledge about f (i.e f is convex, piece-wise constant, etc.) we can use specialized techniques, but in general, the structure of the underlying function is not known. Moreover, f is often considered to be *expensive* to evaluate, which rules out the use of exhaustive search or gradient-based methods from the choice of optimization tools.

Since f is often expensive to evaluate, we need to carefully pick *where* to evaluate our function. Often given a limited *budget*, or number of evaluations, we transform the original optimization problem into one that optimizes where to next evaluate the function f. The function which handles this decision is called the *acquisition function*. The acquisition function is a much cheaper function to optimize, and is often chosen or designed using prior knowledge about the types of solutions that an experimenter expects. The maximization of the acquisition function $a(x)$ is sequential, meaning that past choices of iterates and their function evaluations inform future evaluations. The acquisition maximization can be written as:

$$\phi_{t+1} = \arg\max_{\phi} a(\phi | D_t) \tag{1.1.6}$$

where dataset D_t is the set of past iterates paired with their evaluations up until time t, $D_t = \{\phi_{t'}, f(\phi_{t'}))\}_{t'=1}^{t}$.

1.1.7. Meta-Learning

Most deep learning models are built to solve only one task and often lack the ability to generalize and quickly adapt to solve a new set of tasks. Meta-learning involves learning a *learning algorithm* which can adapt quickly rather than learning from scratch. Several methods have been proposed, treating the learning algorithm as a recurrent model capable of remembering past experience [**70, 51, 48**], as a non-parametric model [**36, 84, 73**], or as an optimization problem [**64, 19**]. In this paper, we focus on a popular version of a gradient-based meta-learning algorithm called Model Agnostic Meta-Learning (MAML; [**19**]).

1.1.8. Gradient-based Meta-Learning

The main idea in MAML is to find a good parameter initialization such that the model can adapt to a new task, τ, quickly. Formally, given a distribution of tasks $p(\tau)$ and a loss function $\mathcal{L}_\tau$ corresponding to each task, the aim is to find parameters θ such that the model f_θ can adapt to new tasks with one or few gradient steps. For example, in the case of a single gradient step, the parameters θ'_τ adapted to the task τ are

$$\theta'_\tau = \theta - \alpha \nabla_\theta \mathcal{L}_\tau(\mathcal{D}_{\text{train}}, f_\theta), \tag{1.1.7}$$

with step size α, where the loss is evaluated on a (typically small) dataset $\mathcal{D}_{\text{train}}$ of training examples from task τ. In order to find a good initial value of the parameters θ, the objective function being optimized in MAML is written as

$$\min_{\theta} \sum_{\tau_i} \mathcal{L}_{\tau_i}(\mathcal{D}_{\text{test}}, f_{\theta'_{\tau_i}}), \tag{1.1.8}$$

where it evaluates the performance in generalization on some test examples $\mathcal{D}_{\text{test}}$ for task τ. The meta objective function is optimized by gradient descent where the parameters are updated according to

$$\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{\tau_i} \mathcal{L}_{\tau_i}(\mathcal{D}_{\text{test}}, f_{\theta'_{\tau_i}}), \tag{1.1.9}$$

where β is the outer step size.

1.1.9. Meta-Reinforcement Learning

In addition to few-shot supervised learning problems, where the number of training examples is small, meta-learning has also been successfully applied to reinforcement learning problems. In meta-reinforcement learning, the goal is to find a policy that can quickly adapt to new environments, generally from only a few trajectories. [**63**] treat this problem by conditioning the policy on a latent representation of the task, and [**16, 85**] represent the reinforcement learning algorithm as a recurrent network, inspired by the "black-box" meta-learning methods mentioned above. Some meta-learning algorithms can even be adapted to reinforcement learning with minimal changes [**48**]. In particular, MAML has also shown some success in robotics applications [**20**]. In the context of reinforcement learning, $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$ are datasets of trajectories sampled by the policies before and after adaptation (i.e rollouts in $\mathcal{D}_{\text{train}}$ are sampled before the gradient step in Equation 1.1.7, whereas those in $\mathcal{D}_{\text{test}}$ are sampled after). The loss function used for the adaptation is REINFORCE [**90**], and the outer, meta objective in Equation 1.1.9 is optimized using TRPO [**72**].

1.1.10. Stein Variational Policy Gradient

For interested readers, we provide a brief overview of Stein's Method, Stein Variational Gradient Descent, and Stein Variational Policy Gradient. A more thorough overview can be found in [**44**].

Stein's Method [**75**] is an approach for obtaining bounds on distances between distributions, but has generally stayed in the realm of theoretical statistics. Recently, [**23**] and [**41**] applied Stein's Method to machine learning, showing that it can be used efficiently for goodness-of-fit tests. As a follow-up, [**42**] derived *Stein Variational Gradient Descent (SVGD)*, a gradient-based variational inference algorithm that iteratively transforms a set of particles into a target distribution, using an underlying connection between Stein's Method and KL-Divergence minimization.

However, in a reinforcement learning context, the target distribution is not known, and therefore needs to be sampled from via interactions with an environment. [**43**] extended SVGD to Stein Variational Policy Gradient (SVPG), which learns an ensemble of policies μ_ϕ in a maximum-entropy RL framework [**94**].

$$\max_{\mu} \ \mathbf{E}_\mu[J(\mu)] + \alpha\mathcal{H}(\mu) \tag{1.1.10}$$

SVPG uses SVGD to iteratively update an ensemble of N policies or *particles* $\mu_\phi = \{\mu_{\phi_i}\}_{i=1}^N$ using:

$$\mu_{\phi_i} \leftarrow \mu_{\phi_i} + \frac{\epsilon}{N}\sum_{j=1}^{N}[\nabla_{\mu_{\phi_j}} J(\mu_{\phi_j})k(\mu_{\phi_j}, \mu_{\phi_i}) + \alpha\nabla_{\mu_{\phi_j}} k(\mu_{\phi_j}, \mu_{\phi_i})] \tag{1.1.11}$$

with step size ϵ and positive definite kernel k. This update rule balances exploitation (first term moves particles towards high-reward regions) and exploration (second term repulses similar policies). As the original authors use an improper prior on the distribution of particles, we simplify the notation by dropping the term from Equation 1.1.11.

1.2. Related Work

In this section, we aggregate related work that helps understand the landscape of task distribution learning, adaptive simulators, and curriculum learning within various contexts.

1.2.1. Dynamic and Adversarial Simulators

Simulators have played a crucial role in transferring learned policies onto real robots, and many different strategies have been proposed. Randomizing simulation parameters for better generalization or transfer performance is a well-established idea in evolutionary robotics [**92, 5**], but recently has emerged as an effective way to perform zero-shot transfer of deep reinforcement learning policies in difficult tasks [**2, 79, 57, 69**].

Learnable simulations are also an effective way to adapt a simulation to a particular target environment. [**10**] and [**68**] use RL for effective transfer by learning parameters of a simulation that accurately describes the target domain, but require the target domain for reward calculation, which can lead to overfitting. In contrast, Active Domain Randomization (ADR), presented in Ch.2, requires no target domain, but rather only a reference domain (the default simulation parameters) and a general range for each parameter. ADR encourages diversity, and as a result, gives the agent a wider variety of experience. In addition, unlike [**10**], our method does not require carefully-tuned (co-)variances or task-specific cost functions. Concurrently, [**32**] also showed the advantages of learning adversarial simulations and the disadvantages of purely uniform randomization distributions in object detection tasks.

To improve policy robustness, Robust Adversarial Reinforcement Learning (RARL) [**58**] jointly trains both an agent and an adversary who applies environment forces that disrupt the agent's task progress. ADR removes the zero-sum game dynamics, which have been known to decrease training stability [**47**]. More importantly, our method's final outputs - the SVPG-based sampling strategy and discriminator - are reusable and can be used to train new agents as shown in Appendix A.3, whereas a trained RARL adversary, would overpower any new agent and impede learning progress.

1.2.2. Active Learning and Informative Samples

Active learning methods in supervised learning try to construct a *representative*, sometimes time-variant, dataset from a large pool of unlabelled data by proposing elements to be labeled. The chosen samples are labelled by an oracle and sent back to the model for use. Similarly, ADR searches for what environments may be most useful to the agent at any given time. Active learners, like BO methods discussed in Section 1.1.6, often require an acquisition function (derived from a notion of model uncertainty) to chose the next sample. Since ADR handles this decision through the explore-exploit framework of RL and the α in SVPG, ADR sidesteps the well-known scalability issues of both active learning and BO [**82**].

Recently, [**81**] showed that certain examples in popular computer vision datasets are harder to learn and that some examples are forgotten much quicker than others. We explore the same phenomenon in the space of MDPs defined by our randomization ranges and try to find the "examples" that cause our agent the most trouble. Unlike in active learning or [**81**], we have no oracle or supervisory loss signal in RL, and instead, attempt to learn a proxy signal for ADR via a discriminator.

1.2.3. Generalization in Reinforcement Learning

Generalization in RL has long been one of the holy grails of the field, and recent work like [**55**], [**11**], and [**18**] highlight the tendency of deep RL policies to overfit to details of the training environment. Our experiments exhibit the same phenomena, but our method improves upon the state of the art by explicitly searching for and varying the environment aspects that our agent policy may have overfit to. We find that our agents, when trained more frequently on these problematic samples, show better generalization over all environments tested.

1.2.4. Interference and Transfer in Multi-Task Learning

Multi-task learning deals with training a single agent to do multiple things, such as a robotic arm trained to pick up objects and open doors. Often in multi-task learning, we aim to train a *single* policy that can accomplish everything, rather than training multiple,

separate policies for each task. When sharing parameters, or more commonly, using the same network across tasks, gradients from various sources can *interfere* or *transfer*, hindering or enabling learning across the curriculum. Recently, [**65**] have shown that when distinct tasks with minimal gradient interference are trained on, the resulting agent transfers better in a multi-task learning scenario.

When two separate tasks have gradients pointing in opposite directions, they can cancel out, or combine (average) into a third direction that is not optimal for either task. When taking the average gradient step, it has been empirically shown by [**9**] that when gradients do not interfere, agents improve their multi-task performance. Ideas have been proposed that these gradients *align* better [**56**], but in our setting, we focus on the sampled gradients from the tasks without any transformation.

We can quantify *transfer* and *interference* of gradients using the cosine similarity of gradients from two tasks, T_a and T_b, as follows:

$$\rho_{ab}(\theta) = \frac{\langle \nabla J_{T_a}(\theta), \nabla J_{T_b}(\theta) \rangle}{||\nabla J_{T_a}(\theta)||||\nabla J_{T_b}(\theta)||} \tag{1.2.1}$$

where cosine similarity $\rho(\theta)$ is bounded between $+1$ (full transfer) and -1 (full interference).

1.2.5. Optimization Analysis in Reinforcement Learning

In policy optimization, we traditionally consider a RL framework [**76**] where some task T is defined by a MDP consisting of a state space S, action space A, state transition function $P : S \times A \mapsto S$, reward function $R : S \times A \mapsto \mathbb{R}$, and discount factor $\gamma \in (0,1)$. The goal for an agent trying to solve T is to learn a policy π with parameters θ that maximizes the expected total discounted reward.

After training the policy in this maximization setting, ideally, any solution found by policy optimization would be some kind of local maximum. However, under a non-stationary MDP, standard policy optimization algorithms are no longer guaranteed to converge to any type of optima. In addition, when working with function approximators in high dimensions, optimization analysis of solutions is very difficult.

Recently, [**1**] showed that with linear policies, we can analyze policy optimization solutions using perturbative methods. Briefly, they approximate the solution neighborhood as a local quadratic and estimate the curvature of the neighborhood by perturbing the solution with a minor amount of noise. This allows for estimation of the policy quality from the perturbations' effects. While the method produces a high variance estimate (we characterize the perturbation using accumulated reward as a metric), with enough noise samples, meaningful conclusions regarding solutions found with policy optimization can still be drawn.

1.2.6. Task Distributions in Meta-Reinforcement Learning

When discussing meta-reinforcement learning, to the best of our knowledge, the task distribution $p(\tau)$ has never been studied or ablated upon. As most benchmark environments and tasks in meta-RL stem from two papers ([**19, 66**], with the task distributions being prescribed with the environments), the discussion in meta-RL papers has almost always centered around the computation of the updates [**66**], practical improvements and approximations made to improve efficiency or learning exploration policies with meta-learning [**74, 26, 25**]. In this section, we briefly discuss prior work in curriculum learning that bears the most similarity to the analyses we conduct here.

Starting with the seminal curriculum learning paper [**4**], many different proposals to learn an optimal *ordering* over tasks have been studied. Curriculum learning has been tackled with Bayesian Optimization [**83**], multi-armed bandits [**24**], and evolutionary strategies [**86**] in supervised learning and reinforcement learning settings, but here, we focus on the latter. However, in most work, the task space is almost always discrete, with a teacher agent looking to choose the best next task over a set of N pre-made tasks. The notion of *best* has also been explored in-depth, with metrics being based on a variety of things from ground-truth accuracy or reward to adversarial gains between a teacher and student agent [**59**].

However, up until recently, the notion of continuously-parameterized curriculum learning has been studied less often. Often, continuous-task curriculum learning exploits a notion of *difficulty* in the task itself. In order to get agents to hop over large gaps, it's been empirically easier to get them to jump over smaller ones first [**28**]; likewise, in navigation domains, it is easier to show easier goals and *grow* a goal space [**60**], or even work backward towards the start state in a reverse curriculum manner [**21**].

While deep reinforcement learning, particularly in robotics, has seen a large amount of curriculum learning papers in recent times [**46, 53**], curriculum learning has not been extensively researched in meta-RL. This may be partly due to the naissance of the field; only recently was a large-scale, multi-task benchmark for meta-RL released [**91**]. As we hope to show in this thesis, the notions of tasks, task distributions, and curricula in meta-learning are fruitful avenues of study and can make (or break) many of the meta-learning algorithms in use today.

1.3. Preliminary Results Regarding Neural Networks and Generalization

Before continuing onto the sim2real problem, we use this section to highlight a curious phenomenon: the generalization of a neural network and its non-intuitive dependence on the data distribution.

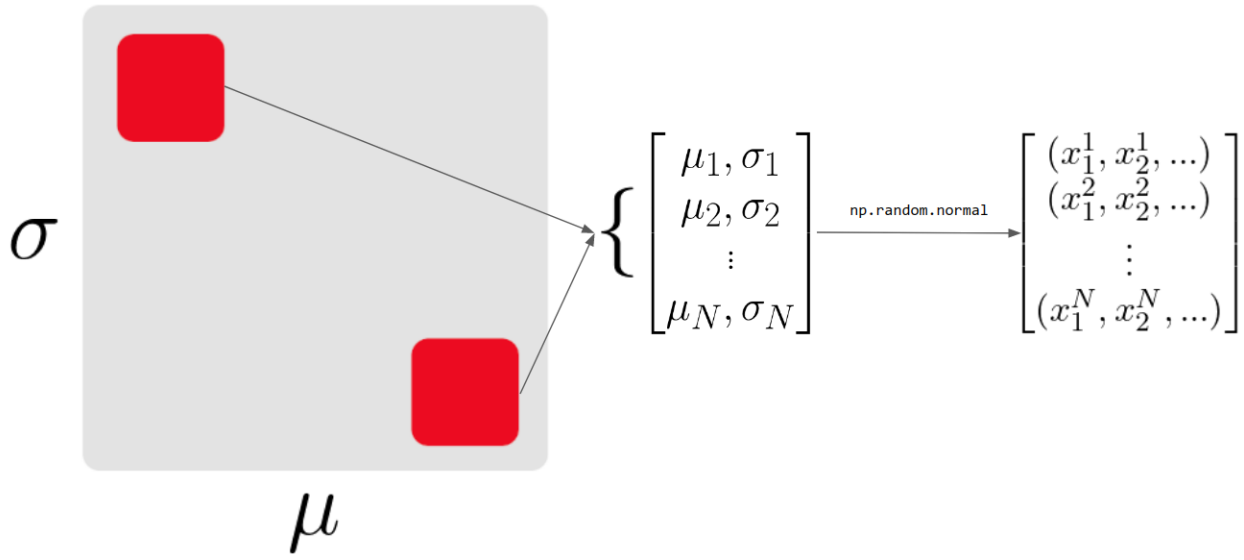


Fig. 1.1. A diagram of data generation setup. We define regions of the training set by zoning off rectangles of (μ, σ) pairs, and then use standard library tools to generate samples from that distribution

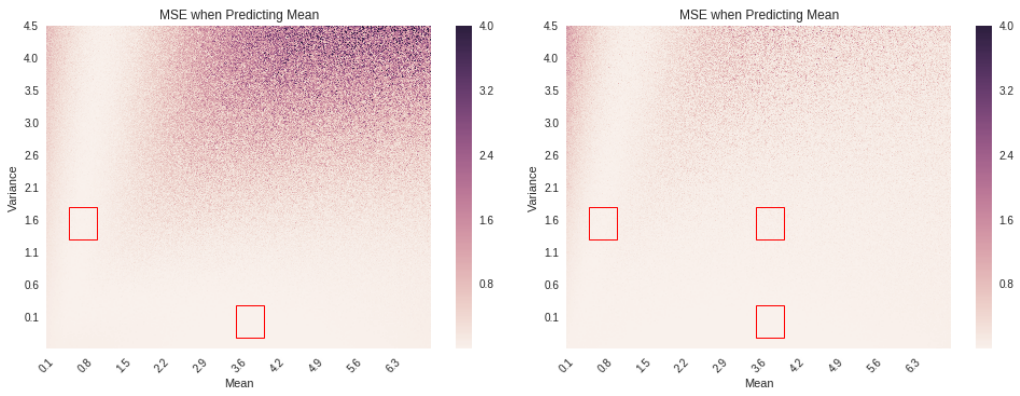
1.3.1. Informative Samples

With the exception of Chapter 5, much of this thesis examines work in which neural networks are used to learn policies or distributions. Neural networks, from a theoretical perspective, are poorly understood. One of the foremost theoretical challenges in the field today is the generalization properties of neural networks.

Neural networks have been known to have strangely-strong out-of-distribution performance, and while hypotheses circulate regarding optimum geometry [**15, 30**], information content [**78**], and the characteristics of stochastic gradient descent [**93**], the problem is, as of writing, still very much open. Here, we make no attempt to address the problem of generalization in neural networks, but rather point out a curious phenomenon which we exploit throughout the rest of this thesis.

We train a small neural network to infer the mean of a normal distribution, given samples from that distribution. Illustrated in Figure 1.1, we generate a training set as follows: we block off regions of mean and standard deviation pairs (the red squares) and generate data samples from that distribution. This vector of samples is sent into the neural network and is used to predict the mean of the normal distribution it is sampled from.

We then finely discretize a grid, and at each point (a (μ, σ) pair) generate data samples to feed through the network to predict means. We then colorize the error, shown in the plots of Figure 1.2, where a darker color is a higher prediction error.



(a) (b)

Fig. 1.2. Two starkly different stories of generalization, differing only in the data distribution shown to the neural network. Figure (b) adds a small training area (upper-right red box) that, when included, leads to almost perfect generalization across the domain. Darker is higher error.

In Figure 1.2(a), we see an unsurprising pattern. The network does fairly well with interpolation: predicting values of μ correctly around or between things it saw during training. Naturally, it does poorly in the top-right region, when tested on (μ, σ) pairs well outside of the training distribution.

However, when we add a region to the training distribution (the upper-right red box in Figure 1.2(b)), we see that the network learns to *generalize*, even for points far outside of the training distribution[3]. Despite the additional training region being placed in an already high-performing region, the addition here allows the network to generalize across the test distribution.

This simple experiment allows us to conclude that certain regions of the data distribution may be more informative, helping guide the neural network to an optimum that allows for strong generalization. Yet, even in this example, such regions of data distribution are often unknown and can be counter-intuitive when found. In traditional machine learning scenarios, the problem of finding such regions can become ever more difficult.

Curriculum learning aims to introduce these data regions iteratively (here, we mixed all the data from regions into one large dataset), introducing complex learning dynamics that depend on a moving data distribution. In addition, human experimenters often use a *easy-to-hard* heuristic to guide the scheduling of region introduction, but many problems may not have this intuitive difficulty readily available. The next chapter introduces solutions to the curriculum learning problem in *unstructured* data distributions, using this phenomenon - the phenomenon of representative examples - as an underlying motivation.

[3]We hold constant the architecture, random seeds, training epochs, and the total number of samples seen by the network. We show averaged results across three runs.

Chapter 2

Active Domain Randomization

Recent trends in Deep Reinforcement Learning (DRL) exhibit a growing interest in zero-shot domain transfer, i.e. when a policy is learned in a source domain and is then tested *without finetuning* in a previously unseen target domain. Zero-shot transfer is particularly useful when the task in the target domain is inaccessible, complex, or expensive, such as gathering rollouts from a real-world robot. An ideal agent would learn to *generalize* across domains; it would accomplish the task without exploiting irrelevant features or deficiencies in the source domain (i.e., approximate physics in simulators), which may vary dramatically after transfer.

One promising approach for zero-shot transfer has been Domain Randomization (DR) [**79**]. DR uniformly randomizes environment parameters (e.g. friction, motor torque) in heuristically defined ranges after every training episode. By randomizing everything that might vary in the target environment, the hope is that the agent will view the target domain as just another variation. However, recent works suggest that the sample complexity grows exponentially with the number of randomization parameters, even when dealing only with transfer between simulations (i.e. in [**2**] Figure 8). In addition, when using DR *unsuccessfully*, policy transfer fails, but with no clear way to understand the underlying cause. After a failed transfer, randomization ranges are tweaked heuristically via human trial-and-error. Repeating this process iteratively leads to arbitrary ranges that do (or do not) lead to policy convergence without any insight into how those settings may affect the learned behavior.

In this chapter, we demonstrate that the strategy of *uniformly* sampling environment parameters from predefined ranges is suboptimal and propose an alternative sampling method, **Active Domain Randomization (ADR)**. ADR, shown conceptually in Figure 2.1, formulates DR as a search for randomized environments that maximize utility for the agent policy. Concretely, we aim to find environments that *currently* cause difficulties for the agent policy, dedicating more training time to these troublesome parameter settings. We cast this active search as a RL problem where the ADR sampling policy of the environment is parameterized with SVPG [**43**]. ADR focuses on problematic regions of the randomization space

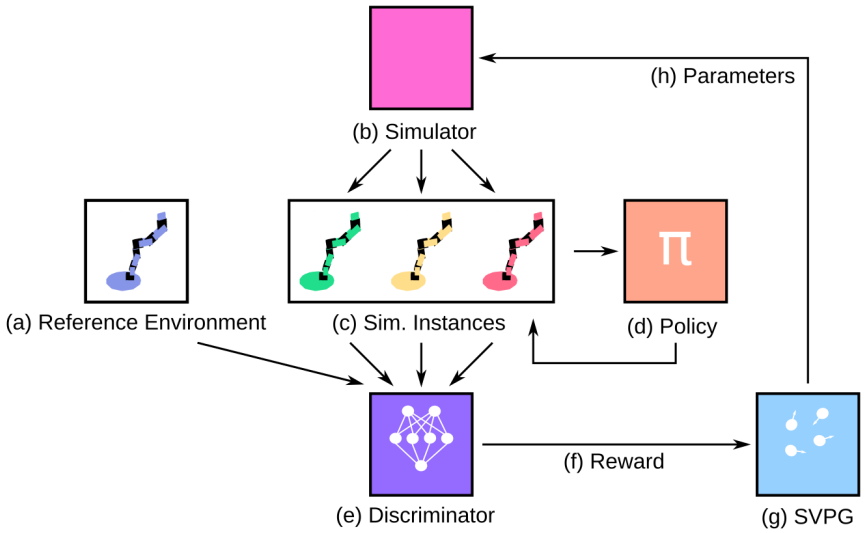


Fig. 2.1. ADR proposes **randomized environments** (c) or simulation instances from a **simulator** (b) and rolls out an **agent policy** (d) in those instances. The **discriminator** (e) learns a **reward** (f) as a proxy for environment difficulty by distinguishing between rollouts in the **reference environment** (a) and randomized instances, which is used to train **SVPG particles** (g). The particles propose a diverse set of environments, trying to find the environment **parameters** (h) that are currently causing the agent the most difficulty.

by learning a discriminative reward computed from discrepancies in policy rollouts generated in randomized and reference environments.

We first showcase ADR in a simple environment where the benefits of training on more challenging variations are apparent and interpretable (Section 2.4.3). In this case, we demonstrate that ADR learns to preferentially select parameters from these more challenging parameter regions while still adapting to the policy's current deficiencies. We then apply ADR to more complex environments and real robot settings (Section 2.4.5) and show that even with high-dimensional search spaces and unmodeled dynamics, policies trained with ADR exhibit superior generalization and lower overall variance than their UDR[1] counterparts.

Finally, we show the safety-critical capabilities of our formulation; by ingesting *arbitrarily off-policy* data, we can fit the parameters of the target environment while using a natural, learned prior on the simulation parameters as a regularizer. This allows the few-shot learning variant of ADR, **ADR+**, to maximize performance on the target task, robot, or environment, all while staying robust across the entire generalization range. ADR+ trains policies that are robust to non-representative datasets, while enabling the use of robotic trajectories collected safely both offline and off-policy.

[1]We refer to the original version of Domain Randomization as Uniform-(Sampling) Domain Randomization (UDR), since the entire parameter space is sampled uniformly randomly.

2.1. Problem Formulation

In this section, we focus on learning a curriculum of environments to show the agent in the domain randomization setting. Unlike UDR seen in Algorithm 1, we aim to show that a *learned* curriculum would be beneficial in this setting, as it has proven to be useful in many other machine learning applications [**67, 28**]. Given a predefined randomization space, we aim to learn a curriculum that maximizes generalization on *all* target tasks - whether that be unseen variations of the same, simulated task, or even a completely different task altogether (i.e transfer onto a real robot).

After [**45**], subsequent works [**54, 86**] have explored the idea of progressively growing curricula in such spaces, using metrics such as return and complex scheduling schemes to heuristically tune when to *grow* the space of possible environments. In contrast, we focus on a fixed, but unstructured, space of environments, looking for beneficial curricula in those spaces.

2.1.1. Domain Randomization, Curriculum Learning, and Bayesian Optimization

As shown in Section 1.1.3, domain randomization generates new environments by randomly perturbing the underlying MDP. This generates a space of related tasks, all of which the agent is expected to be able to solve at the end of training. Traditionally, we define a randomization space, uniformly sample environments from this space, train agents, and evaluate generalization.

In contrast, a curriculum learning setup makes an assumption of *optimality* - that there exists an optimal sequencing of tasks, that, when used to train an agent, generates the best possible agent. Our problem focuses on learning a curriculum of randomized environments in order to train a black-box RL policy π_θ to *generalize* across a wide distribution of environments.

On the surface, the setup is similar to the BO problem formulation, with one small caveat: only the final policy π_θ is evaluated for generalization. Thus, the maximization quantity is now the sequence of iterates, or environments, rather than just a single value. As the goal of curriculum learning is to find the sequence of tasks that generate the best agent after training, in a BO setting, each sequence of training environments would comprise a single datapoint.

If we take function f from the BO setting to be the deterministic *evaluation* of a parameterized policy π_θ, and the series of task iterates $\{\xi_i\}$ to be the environments which the policy π_θ uses to train, we obtain the objective for what we refer to as Curriculum Learning for Reinforcement Learning (CLRL): find the tasks that when used for training, maximize the return of the *final* policy.

2.1.2. Challenges of Bayesian Optimization for Curriculum Learning

Such a setup fits nicely within the BO application area of experiment design [**49**]. However, due to the fact that curriculum learning is not permutation invariant with respect to training tasks, the BO framework would require the entire *sequence* of tasks to be optimized over. To get a single signal for the optimization step, such a setup requires the *complete* training of an RL agent per proposed curriculum. Since we can only evaluate final generalization to perform the BO update, every unique sequence of tasks leads to a different function evaluation. In other words, the experiment space scales *combinatorially* as a function of the BO budget (which, in the DR setting, can be thought of the number of episodes).

As BO in CLRL would require an inordinate amount of training cycles, we aim to relax the usual definitions of the objective function f and acquisition function a. Rather than selecting the entire curriculum for the agent at once, a more reasonable setup would be to pick each environment configuration ξ_i one at a time. Such a setup now allows for iterative adjustments to the curriculum, using feedback from each timestep to pick the next best environment.

With the iterative approach, we are left to find a suitable definition for functions f and a. A reasonable choice would be to take f as a function evaluation, except now a function evaluation of the *current* policy after training on the current environment ξ_i.

While practical, the switch comes at a large cost. By switching to a sequential optimization in the CLRL setting, we lose any analog of an acquisition function. In addition, due to the definition of f and the embedded optimization step of the agent π_θ, the following properties no longer hold:

(1) **Stationarity** - Each particular policy gradient update changes the underlying function that is being maximized; the task that would have provided the most benefit to an agent at time t is not the same as the maximum-benefit task after a policy gradient update at time $t+1$. In BO, the function is thought to be stationary, and while research has been proposed to combat noise and small evaluation errors of f [**39**], in general, the framework breaks down quickly without this assumption.

(2) **Stochasticity** - Up to experimental errors, evaluation of f in BO is generally considered to be deterministic. In the CLRL setting, evaluating f involves stochastically sampling a policy and estimating returns. Stochasticity in a BO setting slows learning, as more function evaluations are needed in order to reduce the uncertainty at any given point.

(3) **Irreversibility** - A bad function evaluation in the BO setting, while wasteful, does not affect future evaluations and can be safely disregarded. In the CLRL setting, a poor policy gradient step can introduce optimization difficulties that affect future

steps; therefore, each step in a CLRL setting is "worth more", and greater care must be taken.

Even though we no longer have one, an acquisition function would still be helpful in the CLRL setting. However, the function needs to be inherently robust to the issues above, while remaining a strong indicator for which environment to sample next. In the following sections, we show why the reinforcement learning paradigm is a more suitable approach to deal with the non-stationarity inherent in the CLRL problem, and introduce our own method, *Active Domain Randomization*.

2.2. Active Domain Randomization

The notion of optimal task sequencing in curriculum learning, discussed in Section 2.1.1, implies that at each step, there is a *next-best* environment for the agent to train on. However, outside of contrived or specific problem settings, finding these optimal environments at each step is difficult because **(1)** An intuitive task ordering of MDP instances or parameter ranges is rarely known beforehand and **(2)** Domain Randomization (DR) is used mostly when the space of randomized parameters is high-dimensional or noninterpretable.

Drawing analogies with BO literature, one can consider the randomization space as a search space. Traditionally, in BO, the search for where to evaluate an objective is informed by acquisition functions, which trade off exploitation of the objective with exploration in the uncertain regions of the space [**7**]. However, as we saw in 2.1.2, unlike the stationary objectives seen in BO, training the agent policy renders our optimization non-stationary: the environment with the highest utility at time t is likely not the same as the maximum utility environment at time $t + 1$.

This requires us to redefine the notion of an acquisition function while simultaneously dealing with BO's deficiencies with higher-dimensional inputs [**87**].

Algorithm 2 Active Domain Randomization

Input: Ξ: Randomization space, S: Simulator, ξ_{ref}: reference parameters
Initialize π_θ: agent policy, μ_ϕ: SVPG particles, D_ψ: discriminator, $E_{ref} \leftarrow S(\xi_{ref})$: reference environment
while not *max_timesteps* **do**
    for each particle μ_{ϕ_i} **do**
        rollout $\xi_i \sim \mu_{\phi_i}(\cdot)$
    for each ξ_i **do**
        $E_i \leftarrow S(\xi_i)$ ▷ Generate randomized environment
        rollout $\tau_i \sim \pi_\theta(\cdot; E_i)$ ▷ Rollout in randomized environment
        $\mathcal{T}_{rand} \leftarrow \mathcal{T}_{rand} \cup \tau_i$
        rollout $\tau_{ref} \sim \pi_\theta(\cdot; E_{ref})$
        $\mathcal{T}_{ref} \leftarrow \mathcal{T}_{ref} \cup \tau_{ref}$
    for each $\tau_i \in \mathcal{T}_{rand}$ **do** ▷ Calculate discriminative reward for env.
        Calculate $r_i = \log D_\psi(y|\tau_i \sim \pi(\cdot; E_i))$
    for each particle μ_{ϕ_i} with discriminative reward r_i **do**
        Update acquisition function analog μ_ϕ using learned rewards r_i ▷ Eq. 1.1.11
    with $\mathcal{T}_{rand}$ **update**:
      $\theta \leftarrow \theta + \nu \nabla_\theta J(\pi_\theta)$ ▷ Agent policy gradient updates
    Update D_ψ with τ_i and τ_{ref} using SGD.

To this end, we propose ADR, summarized in Algorithm 4 and Figure 2.1. ADR provides a framework for manipulating a more general analog of an *acquisition function*, selecting the most informative MDPs for the agent within the randomization space.

By formulating the search as an RL problem, ADR learns a policy μ_ϕ where states are proposed randomization configurations $\xi \in \Xi$ and actions are continuous changes to those parameters. While RL convergence guarantees do not hold in non-stationary MDPs, empirically, deep RL algorithms are suitable candidates to deal with non-stationary environments. In addition, framing the optimization as an RL problem allows us to more easily deal with the stochasticity of function evaluations, folding the stochasticity from the environment into the policy gradient update. RL allows us to remove the computational explosion described in Section 2.1.2, allowing us to select (and incorporate feedback from) a single training environment at each timestep, rather than optimizing for the entire curriculum all at once.

Our method takes as input an untrained agent policy π, a simulator S (which is a map from task parameters ξ to MDP environments E), a randomization space Ξ (see Section 1.1.3), and a reference environment, E_{ref}. The reference environment is the default environment, often shipping with the original task definition.

We parameterize the policy μ_ϕ with SVPG, a particle-based policy gradient method that builds on Stein's method. SVPG, as covered in Section 1.1.10, optimizes for diversity while also maximizing reward. Using SVPG for μ_ϕ allows us to both find high-value environments

for agent training, while also improving diversity due to kernel repulsion terms in Equation 1.1.11.

We proceed to train the agent policy π on the randomized instances E_i, just as in UDR. To generate each randomized environment, we rollout the SVPG particles $\{\mu_{\phi_i}\}$ to generate the set of task definitions $\{\xi_i\}$. Our simulator S uses the task definitions to generate environment instances E_i. We then roll out π on each randomized instance E_i and store each trajectory τ_i.

For every randomized trajectory generated, we use the *same* policy to collect and store a reference trajectory τ_{ref} by rolling out π in the default environment E_{ref} (lines 10-12, Figure 2.1a, c).

Using the agent policy trajectories, we train the ensemble μ_ϕ with a discriminator-based reward, similar to the reward seen in [**17**]:

$$r_D = \log D_\psi(y|\tau_i \sim \pi(\cdot; E_i)) \tag{2.2.1}$$

where y is a boolean variable denoting the discriminator's prediction of which type of environment (a randomized environment E_i or reference environment E_{ref}) the trajectory τ_i was generated from. Using each stored randomized trajectory τ_i by passing them through the discriminator D_ψ, which predicts the type of environment (reference or randomized) each trajectory was generated from.

For each particle of SVPG, the rewards are calculated with the discriminator and used in the joint update, derived from combining Equations 1.1.11 and 2.2.1:

$$\Delta\mu_{\phi_i} = \frac{\epsilon}{n}\sum_{j=1}^{n}\left[\nabla_{\phi_j}\sum_{t=0}^{H}\frac{\gamma^t}{\alpha}(\log D_\psi(y|\tau_t \sim \pi(\cdot; E_t)) + \alpha H)k(\phi_i, \phi_j) + \nabla_{\phi_j}k(\phi_i, \phi_j)\right] \tag{2.2.2}$$

Intuitively, we reward the policy μ_ϕ for finding regions of the randomization space that produce environment instances where the *same* agent policy π acts differently than in the reference environment. Since, in SVPG, each particle proposes its own environment settings ξ_i (lines 4-6, Figure 2.1h), all of which are passed to the agent for training, the agent policy benefits from the same environment variety seen in UDR. However, unlike UDR, μ_ϕ can use the learned reward to focus on problematic MDP instances while still being efficiently parallelizable. μ_ϕ serves as the learned analog of the acquisition function, using the discriminative reward as a proxy for choosing what types of environments are most beneficial to show the agent policy next.

The agent policy π sees and trains *only* on the randomized environments (as it would in traditional DR), using the standard environment reward for updates. As the agent improves on the proposed, problematic environments, it becomes more difficult to differentiate whether any given state transition was generated from the reference or randomized environment.

Thus, ADR can find what parts of the randomization space the agent is currently performing poorly on, and can *actively* update its sampling strategy throughout the training process. ADR fulfills the gap of the acquisition function we were missing in Section 2.1.2, providing a more flexible and efficient approach to learn curricula in unstructured task spaces.

By parameterizing the higher-level policy with the particle-based SVPG, we no longer maximize solely the utility function - given by the acquisition function analog in Equation 1.1.6 - but rather also prioritize *diversity* within the optimization itself. Each particle of ADR can be seen as an independent acquisition function, interacting with the other particles via the kernel term seen in Equation 1.1.11. This improves the diversity of samples, and as we shall see in Section 2.4.3, performance in domain randomization and zero-shot transfer settings.

2.3. Adapting a Simulator with Real World Trajectories

ADR enjoys practical benefits, especially as they relate to robot learning. ADR learns curricula fully in simulation, which in turn train policies that generalize robustly to the real world without finetuning on real-world robots.

However, in many realistic robotics settings, the benefits of zero-shot transfer in robotics are often overstated: in many cases, experimenters have *limited* access to the robot and can often run hand-designed controllers to collect trajectories. Yet, while training policies directly on the real robot is almost always out of question, even finetuning policies on hardware can prove to be dangerous and expensive. To most experimenters, running policies is less desirable than running *controllers* - hardware-specific algorithms that can safely generate complex and varied trajectories. In this section, we show how ADR can be extended to adapt the simulation using this previously-collected, safe, off-policy data from the real robot, allowing for better, safer transfer upon completion of training.

2.3.1. Using Target Domain Data in Reinforcement Learning

In section 2.4.3, we show that ADR can robustly transfer to completely unseen environments in a reliable manner. However, in the context of robot learning, we are often transferring our policy to a *single* robot - here, generalization across a wide range of environment settings is less important than reliably solving the task-specific to the robot we care about. If we knew the parameters that "defined" the robot in simulation, an intuitive choice would be to plug those back into our simulation, train on that *specific* environment, and then transfer a policy back to our robot.

Many approaches exist for finding those parameters, lying under the umbrella of system identification: since experiment designers know what parameters can vary in simulation, we can perform system identification safely using real robot data, and train in our simulation

according to the found parameter settings. In addition, supervised learning approaches, such as GAIL [**29**] or SimOpt [**10**], can be used to regress environment parameters until the simulated trajectories are indistinguishable from the real-world data that was collected.

In both traditional and learned settings, such approaches makes one strong assumption:

The data is sufficiently informative, as to distinguish between various modes of the parameter estimate distribution.

This, similar to domain randomization, is an assumption that makes environment parameter identification a trial-and-error process: if the identified parameters generate policies that do not work in the real world, the only solution is to collect more diverse data and rerun the process again. In addition, the benefits of training on a wide range of simulation parameters are made clear throughout both this thesis and the original domain randomization papers. As regression or system identification provide mainly a point estimates of system parameters, using those to solely train policies may reintroduce the negative effects of simulation overfitting, which are discussed in Section 2.4.3.

2.3.2. Priors in Maximum Entropy Simulators

Instead of regressing simulation parameters to single values, we would like to fit a distribution in parameter space, especially as inferring system parameters from trajectories may be an already ill-posed problem. Additionally, regularizing any estimates would likely improve generalization, especially if we have a "ground truth" environment distribution that, when used for training, generates policies that generalize well. Active Domain Randomization can be thought of as finding a *worst-case* distribution: environments that induce robustness due to their difficulty for the agent policy. However, when we negate the discriminator reward fed into SVPG (which is traditionally rewarded for finding environments where the agent policy π acts differently than in the reference environment) and swap out a reference environment (Figure 2.1(a)) with real-robot trajectories, our approach now prioritizes finding environments where π generates trajectories that look like the ones from the real-world dataset. This is similar to GAIL and traditional system identification approaches, but since running the standard ADR generates that worst-case distribution, we can use that to *regularize* the environment parameters we regress from robot trajectories. Intuitively, we ask ADR to generate environments that match the trajectories provided from the target environment while not straying too far from the distribution is proven to generate good, robust policies.

To incorporate this ADR-prior, we begin with the maximum-entropy RL objective from Section 1.1.2:

$$\mu^* = \arg\max_{\mu} \ \mathbf{E}_{\mu}\Big[\sum_{t=0}^{\infty} \gamma^t (r_t + \alpha D_{KL}[\mu||\bar{\mu}])\Big] \tag{2.3.1}$$

As this choice of prior $\bar{\mu}$ is arbitrary, we can more generally use a prior policy, μ_0, in this formulation as well.

$$\mu^* = \arg\max_{\mu} \ \mathbf{E}_{\mu}\Big[\overbrace{\sum_{t=0}^{\infty} \gamma^t (r_t + \alpha D_{KL}[\mu||\mu_0])}^{J_{ME}}\Big] \tag{2.3.2}$$

This formulation also works with the particle-based SVPG, where each particle update is now individually regularized against the prior policy. We can then write the policy update for particle μ_{ϕ_i} as follows:

$$\mu_{\phi_i} \leftarrow \mu_{\phi_i} + \frac{\epsilon}{n} \sum_{j=1}^{n} \Big[\nabla_{\phi_j}\Big((\frac{1}{\alpha} J_{ME}(\phi_j) k(\phi_i, \phi_j)\Big) + \nabla_{\phi_j} k(\phi_i, \phi_j)\Big] \tag{2.3.3}$$

where ϵ is the step size and $J_{ME}(\phi_j)$ is the particle-based maximum-entropy variant of the traditional utility function, as labelled in Equation 2.3.2.

Since our goal is to incorporate the robustness of the learned-ADR policy while fitting a new policy to match real-world data, we can use the particles from a previous ADR run as μ_0, incorporating a *prior* on simulation parameters that we have found to induce beneficial qualities such as robustness. However, as both μ and μ_0 are actually ensembles of policies (particles), calculating the D_{KL} term would require fitting high-dimensional kernel density estimators, which is problematic when the number of data points (in our case, parameter vectors of each particle) is considerably smaller than the dimensionality (the size of each particle's parameter vector). Instead, we approximate Equation 2.3.3 with a constraint on the *action* spaces, and set up a one-to-one mapping between new and prior particles, as done in [**33**]:

$$\mu^* \simeq \arg\max_{\mu} \mathbf{E}_{\mu}[\sum \gamma^t (r_t + \alpha D_{KL}[\mu^i||\mu_0^i])] \tag{2.3.4}$$

where each μ^i is a particle policy. To derive the joint update for each particle, we simply replace the entropy term $\mathcal{H}$ in Equation 2.2.2 with the one-to-one KL constraint shown in Equation 2.3.4.

We fully describe this algorithm, ADR+, in Algorithm 3.

Algorithm 3 ADR+

Input: Ξ: Randomization space, S: Simulator, τ_{expert}: Expert trajectories from target environment
Initialize π_θ: agent policy, μ_ϕ: SVPG particles, D_ψ: discriminator, $E_{ref} \leftarrow S(\xi_{ref})$: reference environment
Generate prior particles $\{\mu_0\} = \texttt{ADR}(\Xi, E_{ref})$
while not *max_timesteps* **do**
    for each particle μ_ϕ **do**
        rollout $\xi_i \sim \mu_\phi(\cdot)$
    for each ξ_i **do** ▷ Generate rollouts in randomized environments
        $E_i \leftarrow S(\xi_i)$,
        rollout $\tau_i \sim \pi_\theta(\cdot; E_i)$
        $\mathcal{T}_{rand} \leftarrow \mathcal{T}_{rand} \cup \tau_i$
    for each $\tau_i \in \mathcal{T}_{rand}$ **do** ▷ Compute rewards for each proposed environment
        Calculate r_i for ξ_i / E_i (**negating** Eq. (2.2.1))
    with $\mathcal{T}_{rand}$ **update**: ▷ Agent policy gradient update
      $\theta \leftarrow \theta + \nu \nabla_\theta J(\pi_\theta)$
    Update particles with frozen priors $\{\mu_0\}$ using Eq. (2.3.4)
    Update D_ψ with τ_i and τ_{expert} using SGD.

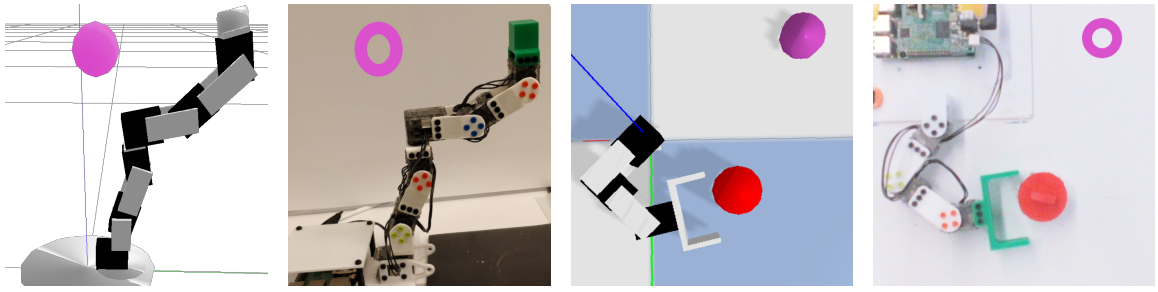
2.4. Results

2.4.1. Experiment Details

To test ADR, we experiment on OpenAI Gym environments [**8**] across various tasks, both simulated and real: **(a)** `LunarLander-v2`, a 2 degrees of freedom (DoF) environment where the agent has to softly land a spacecraft, implemented in Box2D (detailed in Section 2.2), **(b)** `Pusher-3DOF-v0`, a 3 DoF arm from [**27**] that has to push a puck to a target, implemented in Mujoco [**80**], and **(c)** `ErgoReacher-v0`, a 4 DoF arm from [**22**] which has to touch a goal with its end effector, implemented in the Bullet Physics Engine [**12**]. For sim2real experiments, we recreate this environment setup on a real Poppy Ergo Jr. robot [**38**] shown in Figure2.2 (a) and (b), and also create **(d)** `ErgoPusher-v0` an environment similar to `Pusher-3DOF-v0` with a real robot analog seen in Figure 2.2 (c) and (d). We provide a detailed account of the randomized parameters in each environment in Table A.1 in Appendix A.4.

For few shot learning experiments (experiments where we incorporate an ADR-prior) , we use multiple variants of `LunarLander-v2` to generate trajectories, and continue to evaluate *both* generalization and target-task performance.

All simulated experiments are run with five seeds each with five random resets, **totaling 25 independent trials per evaluation point**. All experimental results are plotted mean-averaged with one standard deviation shown. Detailed experiment information can be found in Appendix A.6.



(a) (b) (c) (d)

Fig. 2.2. Along with simulated environments, we display ADR on zero-shot transfer tasks onto real robots.

2.4.2. Zero-Shot Learning: Toy Experiments

To investigate whether ADR's learned sampling strategy provides a tangible benefit for agent generalization, we start by comparing it against traditional DR (labeled as UDR) on `LunarLander-v2` and vary only the main engine strength (MES). In Figure 2.3, we see that ADR approaches expert-levels of generalization whereas UDR fails to generalize on lower main engine strength (MES) ranges.

We compare the learning progress for the different methods on the *hard* environment instances ($\xi_{MES} \sim U[8, 11]$) in Figure 4(a). ADR significantly outperforms both the baseline (trained only on MES of 13) and the UDR agent (trained seeing environments with $\xi_{MES} \sim U[8, 20]$) in terms of performance.

Figures 4(b) and 4(c) showcase the adaptability of ADR by showing generalization and sampling distributions at various stages of training. ADR samples approximately uniformly for the first 650K steps, but then finds a deficiency in the policy on higher ranges of the MES. As those areas become more frequently sampled between 650K-800K steps, the agent learns to solve all of the higher-MES environments, as shown by the generalization curve for 800K steps. As a result, the discriminator is no longer able to differentiate reference and randomized trajectories from the higher MES regions, and starts to reward environment instances generated in the lower end of the MES range, which improves generalization towards the completion of training.

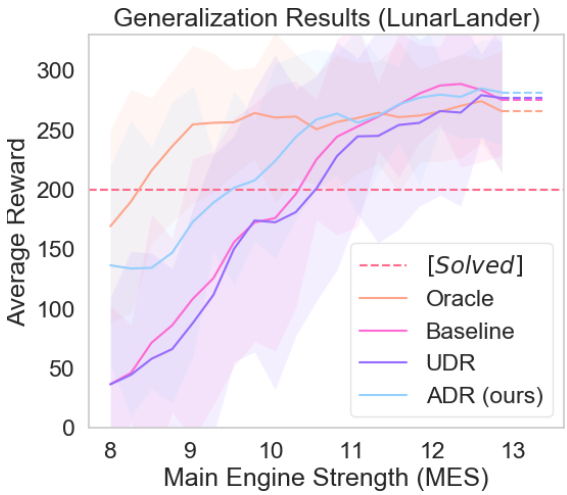
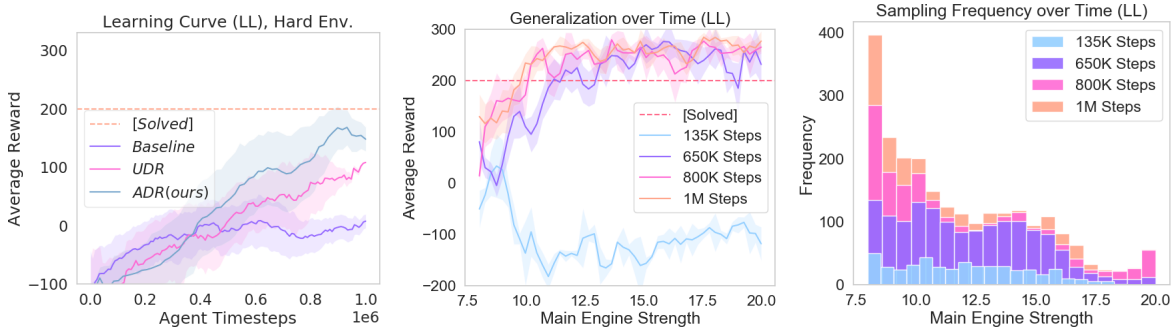


Fig. 2.3. Agent generalization, expressed as performance across different engine strength settings in **LunarLander**. We compare the following approaches: Baseline (default environment dynamics); Uniform Domain Randomization (UDR); Active Domain Randomization (ADR, our approach); and Oracle (a handpicked randomization range of MES [8, 11]). ADR achieves for near-expert levels of generalization, while both Baseline and UDR fail to solve lower MES tasks.

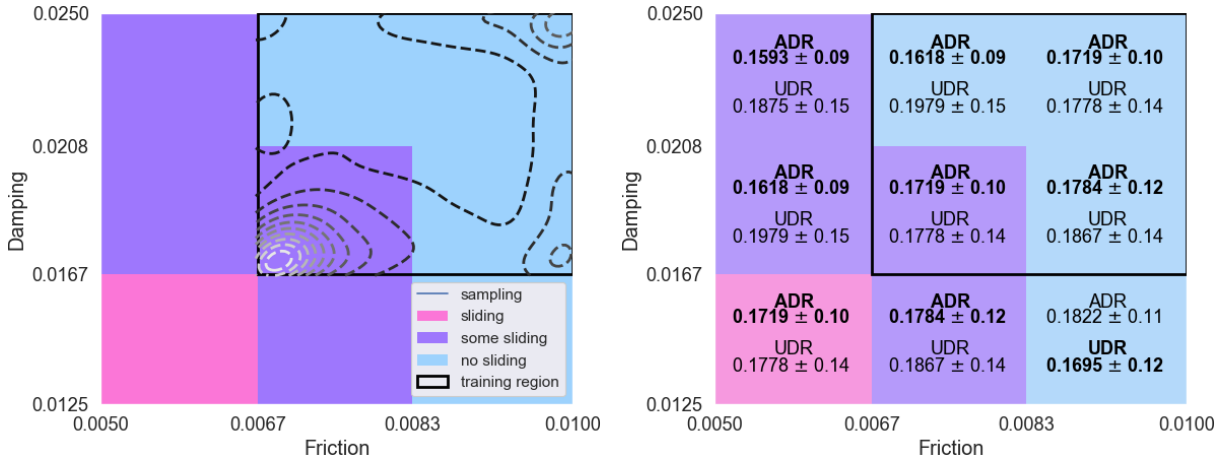


(a) (b) (c)

Fig. 2.4. Learning curves over time in **LunarLander**. Higher is better. **(a)** Performance on particularly difficult settings - our approach outperforms both the policy trained on a single simulator instance ("baseline") and the UDR approach. **(b)** Agent generalization in **LunarLander** over time during training when using ADR. **(c)** Adaptive sampling visualized. ADR, seen in **(b)** and **(c)**, adapts to where the agent is struggling the most, improving generalization performance by end of training.

2.4.3. Zero-Shot Learning: Randomization in High Dimensions

If the intuitions that drive ADR are correct, we should see increased benefit of a learned sampling strategy with larger N_{rand} due to the increasing sparsity of *informative* environments when sampling uniformly. We first explore ADR's performance on `Pusher3Dof-v0`, an environment where $N_{rand} = 2$. Both randomization dimensions (puck damping, puck friction



(a) (b)

Fig. 2.5. In **Pusher-3Dof**, the environment dynamics are characterized by friction and damping of the sliding puck, where sliding correlates with the difficulty of the task (as highlighted by cyan, purple, and pink - from easy to hard). **(a)** During training, the algorithm only had access to a limited, easier range of dynamics (black outlined box in the upper right). **(b)** Performance measured by distance to target, lower is better.

loss) affect whether or not the puck retains momentum and continues to slide after making contact with the agent's end effector. Lowering the values of these parameters *simultaneously* creates an intuitively-harder environment, where the puck continues to slide after being hit. In the reference environment, the puck retains no momentum and must be continuously pushed in order to move. We qualitatively visualize the effect of these parameters on puck sliding in Figure 5(a).

From Figure 5(b), we see ADR's improved robustness to *extrapolation* - or when the target domain lies *outside the training region*. We train two agents, one using ADR and one using UDR, and show them only the training regions encapsulated by the dark, outlined box in the top-right of Figure 5(a). Qualitatively, only 25% of the environments have dynamics which cause the puck to slide, which are the hardest environments to solve in the training region. We see that from the sampling histogram overlaid on Figure 5(a) that ADR prioritizes the single, harder purple region more than the light blue regions, allowing for better generalization to the unseen test domains, as shown in Figure 5(b). ADR outperforms UDR in all but one test region and produces policies with less variance than their UDR counterparts.

2.4.4. Zero-Shot Learning: Randomization in *Uninterpretable* Dimensions

We further show the significance of ADR over UDR on `ErgoReacher-v0`, where $N_{rand} = 8$. It is now impossible to infer intuitively which environments are *hard* due to the complex

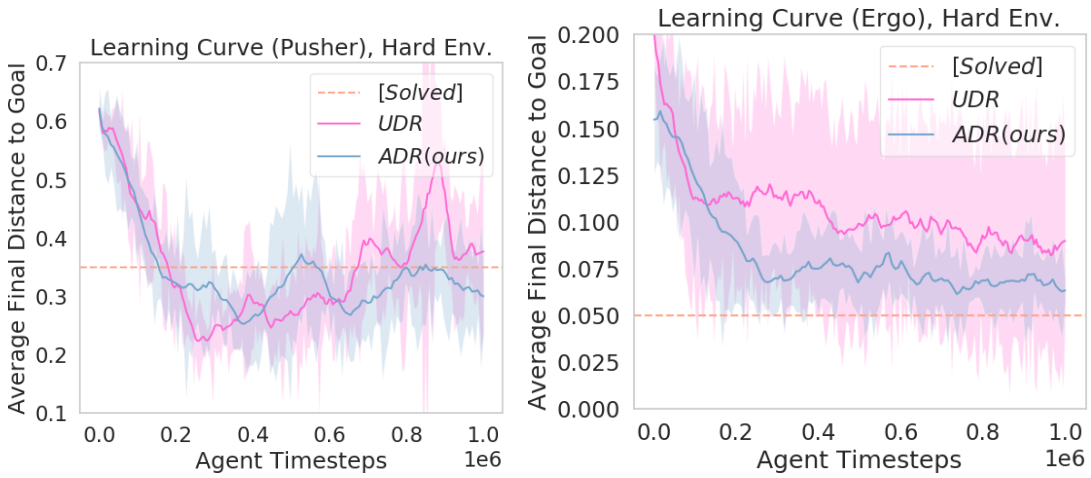


Fig. 2.6. Learning curves over time in **(a) Pusher3Dof-v0** and **(b) ErgoReacher** on held-out, difficult environment settings. Our approach outperforms both the policy trained with the UDR approach both in terms of performance and variance.

interactions between the eight randomization parameters (gains and maximum torques for each joint). For demonstration purposes, we test extrapolation by creating a held-out target environment with extremely low values for torque and gain, which causes certain states in the environment to lead to *catastrophic* failure - gravity pulls the robot end-effector down, and the robot is not strong enough to pull itself back up. We show an example of an agent getting trapped in a catastrophic failure state in Figure A.4, Appendix A.4.1.

To generalize effectively, the sampling policy should prioritize environments with lower torque and gain values in order for the agent to operate in such states precisely. However, since the hard evaluation environment is not seen during training, ADR must learn to prioritize the hardest environments that it can see, while still learning behaviors that can operate well across the entire training region.

From Figure 6(a) (learning curves for `Pusher3Dof-v0` on the unseen, hard environment - the pink square in Figure 2.5) and 6(b) (learning curves for `ErgoReacher-v0` on unseen, hard environment), we observe the detrimental effects of uniform sampling. In `Pusher3Dof-v0`, we see that UDR *unlearns* the good behaviors it acquired in the beginning of training. When training neural networks in both supervised and reinforcement learning settings, this phenomenon has been dubbed as *catastrophic forgetting* [**35**]. ADR seems to exhibit this slightly (leading to "hills" in the curve), but due to the adaptive nature the algorithm, it is able to adjust quickly and retain better performance across all environments.

UDR's high variance on `ErgoReacher-v0` highlights another issue: by continuously training on a random mix of hard and easy MDP instances, both beneficial and detrimental agent behaviors can be learned and unlearned throughout training. This mixing can lead to

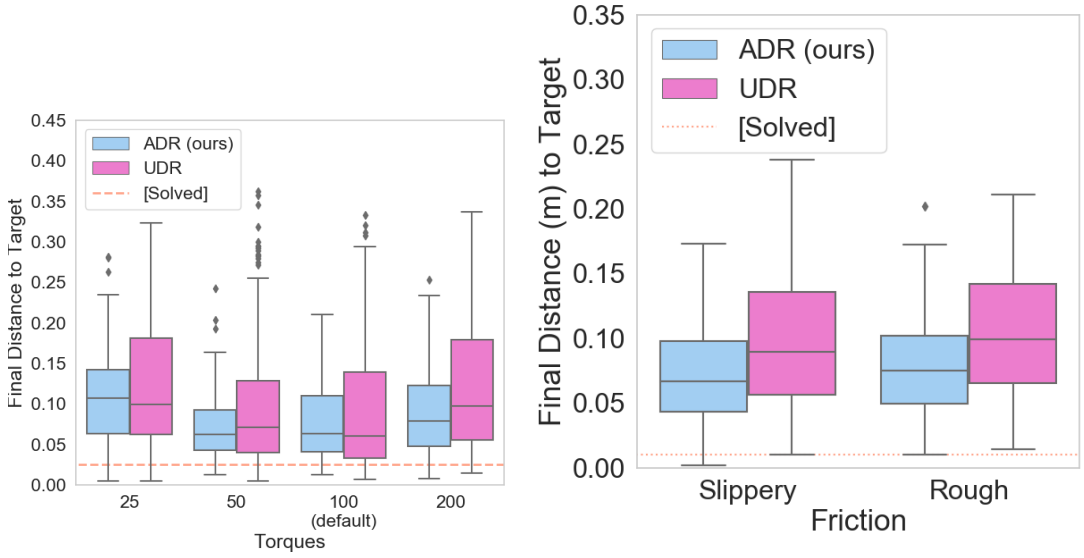


Fig. 2.7. Zero-shot transfer onto real robots **(a) ErgoReacher** and **(b) ErgoPusher**. In both environments, we assess generalization by manually changing torque strength and puck friction respectively.

high-variance, inconsistent, and unpredictable behavior upon transfer. By focusing on those harder environments and allowing the definition of *hard* to adapt over time, ADR shows more consistent performance and better overall generalization than UDR in all environments tested.

2.4.5. Sim2Real Zero-Shot Transfer Experiments

In *sim2real* (simulation to reality) transfer, many policies fail due to unmodeled dynamics within the simulators, as policies may have overfit to or exploited simulation-specific details of their training environments. While the deficiencies and high variance of UDR are clear even in simulated environments, one of the most impressive results of domain randomization was zero-shot transfer out of simulation onto robots. However, we find that the same issues of unpredictable performance apply to UDR-trained policies in the real world as well.

We take each method's (ADR and UDR) five independent simulation-trained policies on both `ErgoReacher-v0` and `ErgoPusher-v0` and transfer them without fine tuning onto the real robot. We rollout only the final policy on the robot, and show performance in Figure 2.7. To evaluate generalization, we alter the environment manually: on `ErgoReacher-v0`, we change the values of the torques (higher torque means the arm moves at higher speed and accelerates faster); on `ErgoPusher-v0`, we change the friction of the sliding puck (slippery or rough). For each environment, we evaluate each of the policies with 25 random goals (125 independent evaluations per method per environment setting).

Even in zero-shot transfer tasks onto real robots, ADR policies obtain overall better or similar performance than UDR policies trained in the same conditions. More importantly, ADR policies are more consistent and display lower spread across all environments, which is crucial when safely evaluating reinforcement learning policies on real-world robots.

2.4.6. Few-Shot Learning: Safe Finetuning with ADR

In this section, we study the problem of *few-shot* robotic learning in the context of safety-critical applications. By safety critical, we mean that optimizing the learned policy *cannot* be done by rolling out the policy on the robot; this allows for off-policy, safely-collected trajectories, but rules many algorithms that require running the *learned* policy on the robot.

In the few-shot experimental setup, we provide a dataset of N expert-collected trajectories. These trajectories can be arbitrarily off-policy, meaning that they need not be generated by any learned policy, but rather via heuristic, traditional control, or human operator. The learning algorithm ideally would use these trajectories to fine-tune, but in a limited manner: for example, the algorithm cannot take on-policy policy gradient steps on the collected data, due to the mismatch between the behavior and learned policies.

2.4.7. Evaluation and Results

In the few-shot, robotic learning case, we have two main criteria that we would like any learning algorithm to achieve:

- **Performance**: Given some expert trajectories sampled from a target environment, we would like our algorithm to optimize for performance on that particular target environment.
- **Robustness**: However, given either non-representative trajectories or too small a quantity of trajectories, we would like to avoid *overfitting* to the dataset. An ideal algorithm will be robust to lack of data diversity and estimation errors, especially as learning environmental parameters from trajectories can be an underspecified (and sometimes, degenerate) optimization problem.[2] Unfortunately, datasets do not often come labelled as underspecified or degenerate, which leads to a need for regularization.

A safe, successful few-shot robotic learning algorithm is one that succeeds on both fronts; suboptimality in the performance objective will degrade the quality of the transferred solution, while approaches that fail to account for robustness will fail in the event of a non-representive dataset.

	GAIL	ADR+
MES=10	218.1 ± 34.5	193.1 ± 23.6
MES=16	234.2 ± 22.3	225.5 ± 38.9

Tableau 2.1. Both algorithms, when given trajectories representative of the target environment, perform strongly.

2.4.8. Performance on Target Environment

When dealing with a *fully-representative* dataset (i.e $N_{traj} = 34$, trajectories pulled from the same dataset agents are being tested on), we see that both methods, GAIL [**29**] and ADR+ do relatively well in the LunarLander environment. This aligns well with the characteristics of both methods: GAIL, at convergence, trains a policy to match the state-action distribution of the dataset, whereas ADR+ drives particles towards that region of randomization space (in this case, the single-dimensional axis of engine strength), allowing the agent to see and train on the target environment more often.

2.4.9. Robustness to Non-representative Trajectories

	GAIL	ADR+
MES=10	-10.5 ± 30.1	145.9 ± 40.1
MES=16	216.1 ± 9.8	201.8 ± 25.6

Tableau 2.2. When given noisy or non-representative trajectories, GAIL fails to recover on the harder environment, whereas ADR+ is able to stay robust via the prior term.

However, when we show the GAIL agent *non-representative* trajectories (i.e a full-sized dataset, but from the *other* environment), we see that it struggles on the much harder task of MES=10. Meanwhile, the regularizing effect of ADR+ allows it to still show the agent a diversity of environments (due to the standard ADR prior), allowing it to avoid overfitting to a single target environment. This quality is incredibly useful in real robotic applications, where robots, which may vary in calibration or manufacturing quality, often need a *single* policy to operate across the whole fleet.

2.4.10. Robustness to Quantity of Expert Trajectories

In addition, when we show both agents *underspecified* datasets (i.e trajectories pulled from target environment, but just fewer of them than empirically needed for convergence), we see that GAIL struggles on even the easier, MES=16 environment and **D**oes **N**ot **C**onverge (DNC) on the harder environments. While ADR+'s performance does not nearly match the full dataset case, we see that it outperforms GAIL by orders of magnitude. The incorporation

[2]As an example, consider estimating a ball's mass from trajectories of the ball only bouncing.

	GAIL (Easy)	**ADR+ (Easy)**	**GAIL (Hard)**	**ADR+ (Hard)**
N_{traj} **= 1**	-80.5 ± 100.1	10.4 ± 72.1	DNC	DNC
N_{traj} **= 5**	-11.1 ± 40.9	89.1 ± 32.5	DNC	1.8 ± 10.1
N_{traj} **= 10**	145.3 ± 27.8	225.4 ± 17.0	40.6 ± 27.1	98.1 ± 37.1

Tableau 2.3. ADR seems slightly more robust in data-limited scenarios, whereas GAIL fails to converge in limited data settings on the harder environment.

of a regularizing prior alongside the diversity of environments leads to strong performance *and* robustness, both qualities of any robotic policy transfer algorithm.

Chapter 3

Optimization Qualities of Multi-task Learning

As we saw in Chapter 2, not all generated MDPs are equally useful for learning, leading to Active Domain Randomization (ADR) - which poses the search for informative randomization parameters (environments, or equivalently, MDPs) as a reinforcement learning problem. ADR scores the usefulness of various randomized environment instances by comparing trajectory rollouts from a randomized simulation instance to rollouts from a reference simulation instance. Intuitively, the regions of the randomization space that generate distinguishable rollouts are of more interest and should be focused on during the training period, as they correspond to more difficult environments. The work showed empirical performance improvements over DR.

This chapter focuses on trying to understand *why* ADR works in practice. More generally, we focus on the effect of task distributions and curricula on policy optimization. We are interested in why certain distributions of tasks (or, if that distribution evolves over time, *curricula* of tasks) induce varying generalization performances. We will study this by characterizing the optimization trajectory and the optima found by each strategy using empirical perturbation methods.

3.1. Motivating Example

Before we can make wide claims about the effect of domain randomization, and more generally, curricula, on policy optimization, we need to ensure that the choice of curricula can truly impact optimization in a significant way. DR, as discussed, randomizes various parameters of the simulator, generating a multitude of MDPs for an agent to train on. We focus on a toy environment: `LunarLander-v2`, where the task is to ground a lander in a designated zone, and reward is based on the quality of the landing (fuel used, impact velocity, etc). `LunarLander-v2` has one main axis of randomization that we vary: the main engine strength (MES). Different values of MES change the MDP, and can be seen as different tasks. The MES, shown on the X-axis of Figure 3.1, controls the strength of the landing

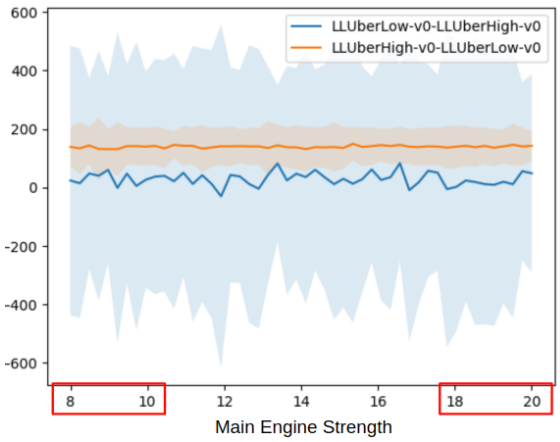


Fig. 3.1. Effects of curriculum order on agent performance. A wrong curriculum, shown in blue, can induce high variance, and overall poor performance.

engine for the agent, crucial for the task performance. When we move this value away from its default value of 13, we can generate superficially-similar MDPs that to an agent, are easier or more difficult to solve.

To study the effects of curricula on optimization, we artificially generate two "buckets" of tasks: *UberLow*, where the MES is uniformly sampled at every episode from the values of $[8, 10]$; and *UberHigh*, where the MES is uniformly sampled at every episode from the values of $[18, 20]$. The buckets of tasks are boxed in red in Figure 3.1.

We train two agents, both for one million time-steps. The only difference between the two agents is the ordering of task buckets they see: for the first 500,000 steps (cumulative through episodes), one agent sees *UberLow* MDPs, whereas the other sees *UberHigh* MDPs. At the 500,000 step mark, the tasks are switched.

To simplify the analysis, we use linear networks [**62**] trained with REINFORCE [**89**], which have been shown to perform well in simple continuous-control tasks. For every evaluation point, we train 10 seeds and generate 50 rollouts from each seed, totaling 500 independent evaluations for each point. We plot the mean average of all 500 runs, and shade in one standard deviation. To evaluate policy generalization and quality, we sweep across the entire MES randomization range of [8, 20] and roll out the final policy in the generated environments.

Figure 3.1 shows the results obtained from both experiments. The blue curve shows the results of generalization when the agent was shown *UberLow*, then *UberHigh*, whereas the orange curve shows the switched curriculum. The curriculum seems to have a strong effect on the stability of optimization, as well as average performance. These results hint at the notion of an underlying *optimal curriculum*, or an optimal sequence of tasks that an agent should train on. However, as this is a very strong assumption, in the following section,

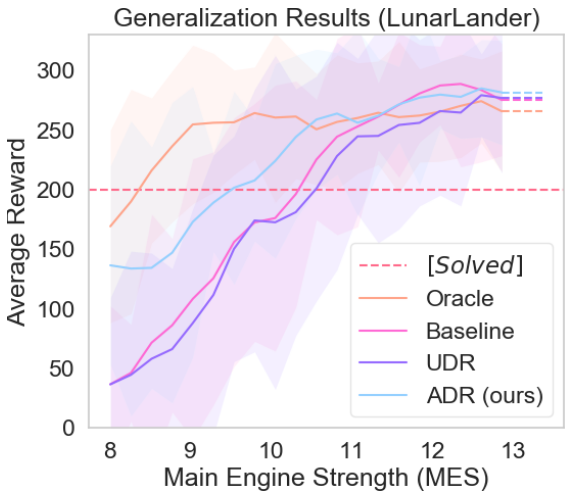


Fig. 3.2. Generalization for various agents who saw different MES randomization ranges. Crossing the *Solved* line "earlier" on x-axis means more environments solved (better generalization).

we explore the optimization of the two agents and generate hypotheses to explain the vast difference in stability between the two.

3.1.1. Gradient Interference in a Two-Task Setting

As noted in Section 1.2.4, continual learning researchers have long been aware of the issues of conflicting gradients across tasks and how they can prematurely halt learning, even when a network's capacity has not saturated [**34**]. In this section, we take our toy example of "two" tasks (which are actually buckets of related tasks) and explore the compatibility of gradients through time.

To quantify *compatibility*, we use Equation 1.2.1 and take the two agents described above at various times through training, and generate heatmaps that show positive cosine similarity between gradients sampled from two tasks (shown on each axis of the heatmap). Briefly, we take an agent policy, sweep it across the full $[8, 20]$ range, and calculate the sampled gradients from that task (i.e a particular value for MES). We average these gradients over 500 independent runs, and use Equation 1.2.1 to generate a heatmap by performing an all-pairs comparison between gradients, shown in Figures 3.3 and 3.4.

In the heatmaps shown in Figure 3.3 and Figure 3.4, we plot the *positive* cosine similarity as dark blue ($\rho_{ab} = +1$). This scheme allows the more important metric, interference, to be grouped together into the yellow regions of the plots ($\rho_{ab} \leq 0$). When a point on the plot is colored yellow, we can denote the two tasks (on x and y-axis) as *incompatible*, meaning that the two tasks' gradients interfere with one another. Keeping in mind that these are all high variance estimates, we focus on *patches* of incompatible tasks, which are more likely to actually be incompatible than randomly dispersed points of yellow.

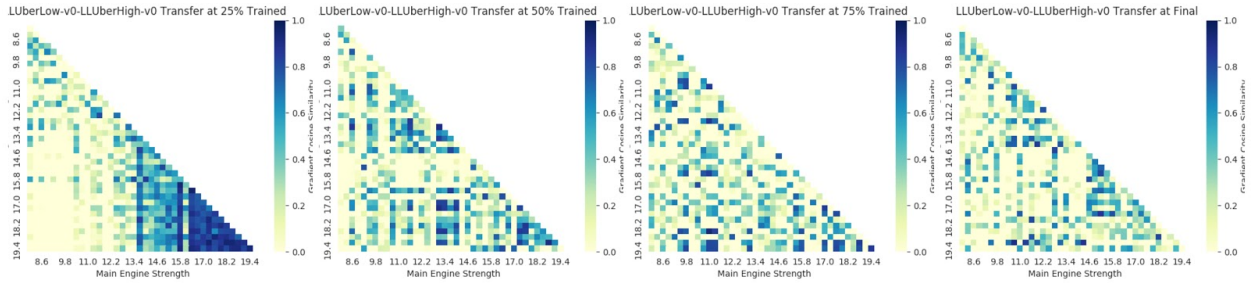


Fig. 3.3. Each axis represents a discretized main engine strength, from which gradients are sampled and averaged across 25 episodes. The heatmap shows the cosine similarity between the corresponding task on the X and Y axes. The different panels show gradient similarity (higher cosine similarity shown as darker blues) after 25%**,** 50%**,** 75%**, and** 100% **training respectively.** A bad curriculum, *UberLow* then *UberHigh*, as shown in the blue curve in Figure 3.1, seems to generate patches of incompatible tasks, earlier in training, as shown by growing amounts of yellow.

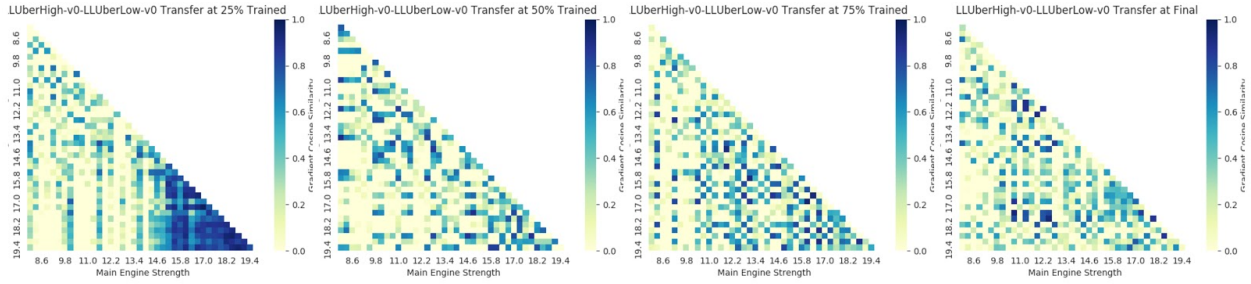


Fig. 3.4. A good curriculum, *UberHigh* then *UberLow*, as shown in the orange curve in Figure 3.1, seems to maintain better transfer through training, as shown by less patches of yellow in later stages of optimization (see Panel 3).

In general, we find that the blue curve, the agent that sees *UberLow* then *UberHigh*, shows large, consistent patches of incompatible tasks much earlier in training than its counterpart. In addition, we find that these patches are consistent throughout training, which is not the case of the alternate-curriculum agent (Figures 3.3, 3.4).

3.2. Increasing Complexity

While interesting, the experiment detailed in the previous section is unrealistic. One advantage that domain randomization has over continual learning is that explicit tasks (or in our case, "buckets" of tasks) need not be defined; continual leaning research has often been criticized for engineering implicit task curricula based on difficulty (See OpenReview discussion of [**65**]). On the contrary, domain randomization's random sampling breaks these types of sequences, treating each task on an equal level throughout training.

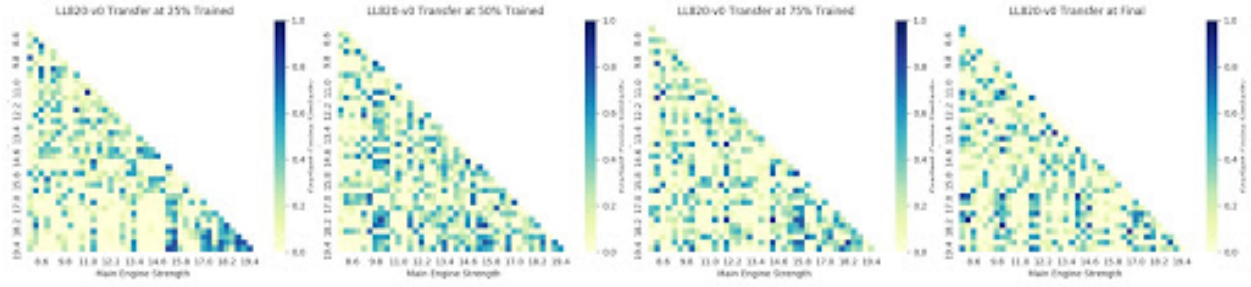


Fig. 3.5. Policies trained with traditional domain randomization show large patches of incompatible tasks that show up early in training (interfering gradients shown as yellow), potentially leading to high variance in convergence, as noted in Chapter 2.

h

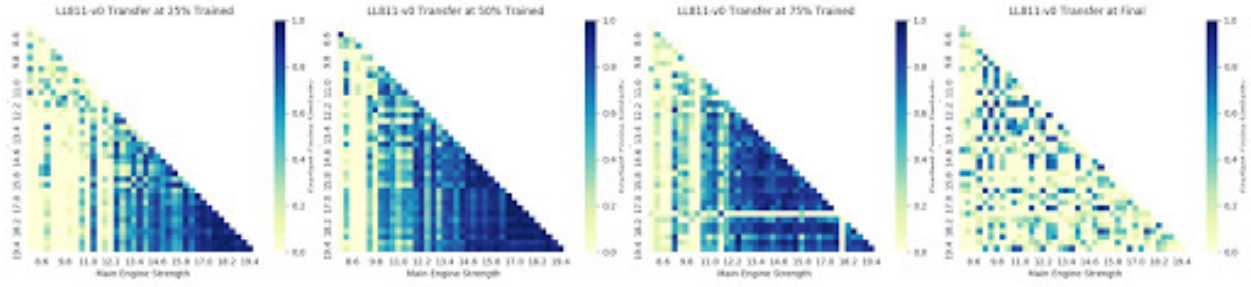


Fig. 3.6. Policies trained with focused domain randomization seem to exhibit high transfer between tasks, which seems to enable better overall generalization by the end of training.

However, as we have seen above, even in domain randomization, the notion of a curriculum is helpful for optimization. In this section, we study a more realistic version of domain randomization: one where "buckets" of tasks are not designated. Rather, we specify only the general ranges from which to vary each environment parameter.

3.2.1. Effect of Curriculum with Domain Randomization

In `LunarLander-v2`, we continue to vary the main engine strength (MES). Again, we evaluate the generalization performance of each of our curriculum choices and draw conclusions of our optimization analyses conditioned on what we know works empirically.

We train multiple agents, with the only difference being the randomization ranges for MES during training. Figure 3.2 shows the final generalization performance of each agent by sweeping across the entire randomization range of [8, 20] and rolling out the policy in the generated environments. We see that focusing on certain MDPs (labeled as *Oracle*, MES $\sim U(8, 11)$) improves generalization over traditional DR (labeled as *UDR*, MES $\sim U(8, 20)$), even when the evaluation environment is outside of the training distribution.

In Figure 3.5 and 3.6, we compare the gradient incompatibilities through training when an agent sees a random sample from the full range (MES $\sim [8, 20]$) or a sample from the subset of the range (MES $\sim [8, 11]$) (Figure 3.5, Figure 3.6 respectively). Interestingly, we see the same types of patterns as the experiment described in Section 3.1.1. The bad curriculum (Figure 3.5), which in this case is *traditional domain randomization* (MES $\sim [8, 20]$), shows large patches of incompatible tasks early in training. The agent exposed to *focused domain randomization* (MES $\sim [8, 11]$, Figure 3.6) shows high task compatibility throughout training, until a final solution is found [1].

As shown in Figure 3.2, we find that high task compatibility throughout training correlates with higher empirical generalization performance, although this statement needs to be rigorously tested before further claims are made.

3.2.2. Analyzing Active Domain Randomization

Active Domain Randomization (ADR) was motivated by the fact that the ranges shown in *focused domain randomization* will not generally be known, so an adaptive algorithm was proposed that focused on finding environment instances (a curriculum) that highlighted the agent policy's current weaknesses. Posed as a higher level reinforcement learning problem, ADR showed empirical improvements over standard domain randomization in zero-shot learning settings, including sim2real robotic transfer. In Figure 3.2, we see that ADR approaches generalization levels of the Oracle.

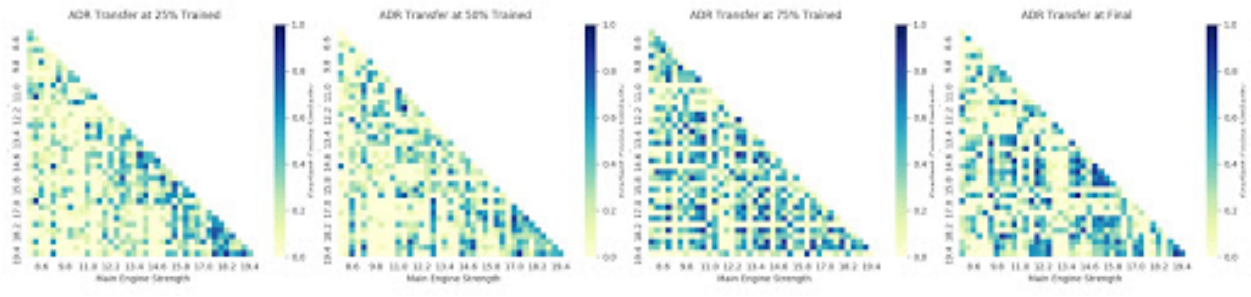


Fig. 3.7. Active Domain Randomization shows its ability to adapt and bring the policy back to areas with less inter-task interference, as shown between Panel 2 and Panel 3.

We can see a different visualization of ADR's adaptiveness in Figure 3.7; by learning a curriculum, we can recover and move out from regions of policy space that show large amounts of interference, as seen by the gradual decrease in task incompatibility over time (specifically between Panel 2 and Panel 3, or at 50% and 75% of training completion respectively).

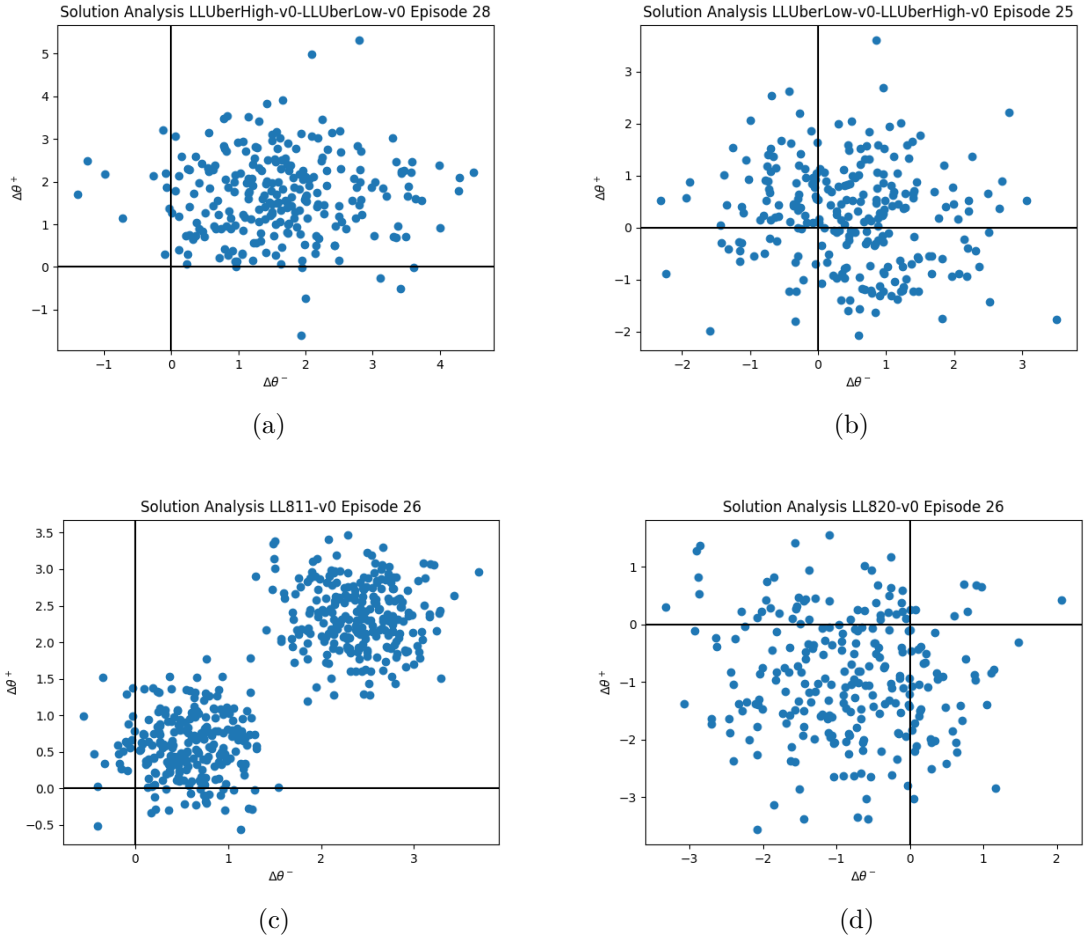
[1] An investigation on why even the good agent shows large patches of incompatible tasks at the end of training is needed. A simple explanation might be that the final solution has nestled into a local optima.

3.3. Anti-Patterns in Optimization

However, this initial investigation of learning dynamics has not resolved all problems in continual and multi-task learning. In fact, in this section, we show that these learning schemes produce counter-intuitive *final* solutions, despite the fact that their learning dynamics point to gradient interference being the main culprit.

Using techniques from [**1**], we use perturbative methods to characterize the final solutions found by each agent. While subject to many conditions and constraints, in general, we expect (a) better performing policies to be at a *local maxima* (as discounted return maximization is a maximization problem) and (b) better performing policies to have "smoother" optimization paths.

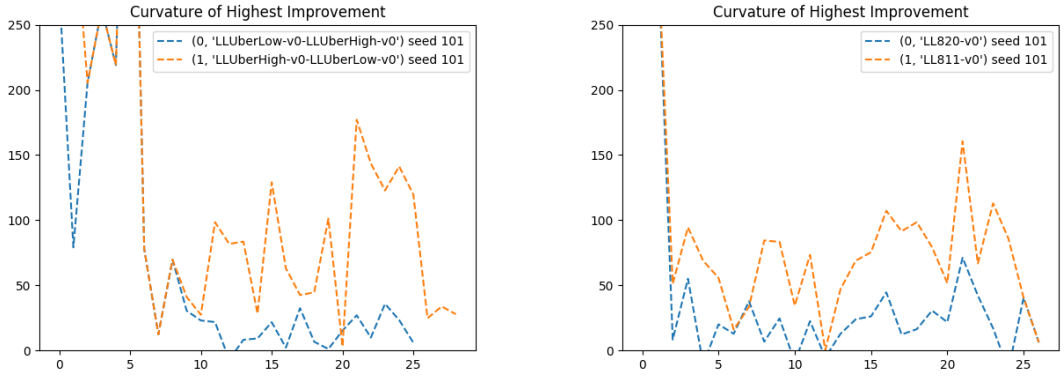
However, when we analyze our solutions, we get counter-intuitive results. Worse performing policies seem to be at local maximas or saddle points, but more importantly, our better performers seem to be at local minimas. In addition, when we plot the curvature of the direction of highest improvement (whose volatility seems to be a good metric for optimization smoothness), we find that the better performing policies seem to be more volatile. While only a single, representative seed is shown here, the results across many seeds seem to be consistent.



(a) (b)

(c) (d)

Fig. 3.8. In **(a)**, final policies trained with *UberHigh, UberLow* (the orange curve in Figure 3.1), seem to be local minima. In **(b)** Final policies trained with *UberLow, UberHigh* (the blue curve in Figure 3.1), seem to be saddles. In **(c)** final Policies trained with MES $\sim U(8, 11)$ (Oracle in Figure 3.2) seem to be local minima, while in **(d)**, final Policies trained with MES $\sim U(8, 20)$ (UDR in Figure 3.2) seem to be saddles



(a) (b)

Fig. 3.9. Curvature analysis through training for various curricula. In **(a)**, the worse performing agent (blue curve, *UberLowUberHigh*), seems to have a smoother optimization path than its more better-performing, flipped-curriculum counterpart. In **(b)** Policies trained with full domain randomization (blue curve), also empirically shown to be less stable in practice, show smoother optimization paths.

.

Chapter 4

Curriculum in Meta-Learning

Meta-learning concerns building models or agents that can learn how to adapt quickly to new tasks from datasets which are orders of magnitudes smaller than their standard supervised learning counterparts. Put differently, meta-learning concerns learning *how to learn*, rather than simply maximizing performance on a single task or dataset. Gradient-based meta-learning has seen a surge of interest, with the foremost algorithm being Model-Agnostic Meta-Learning (MAML) [**19**]. Gradient-based meta-learners are fully trainable via gradient descent and have shown strong performance on various supervised and reinforcement learning tasks [**19, 63**].

The focus of this chapter is on an understudied hyperparameter within the gradient-based meta-learning framework: the distribution of tasks. In MAML, this distribution is assumed given and is used to sample tasks for meta-training of the MAML agent. In supervised learning, this quantity is relatively well-defined, as we often have a large dataset for a task such as image classification [1]. As a result, the distribution of tasks, $p(\tau)$, is built from random minibatches sampled from this distribution.

However, in the meta-reinforcement learning setting, this task distribution is poorly defined and is often handcrafted by a human experimenter with the target task in mind. While the task samples themselves are pulled randomly from a range or distribution (i.e a locomotor asked to achieve a target velocity), the distribution *itself* needs to be specified. In practice, the distribution $p(\tau)$ turns out to be an extremely sensitive hyperparameter in Meta-RL: too "wide" of a distribution (i.e the variety of tasks is too large) leads to underfitting, with agents unable to specialize to the given target task even with larger numbers of gradient steps; too "narrow", and we see poor generalization and adaption to even slightly out-of-distribution environments.

Even worse, randomly sampling (as is often the case) from $p(\tau)$ can allow for sampling of tasks that can cause interference and optimization difficulties, especially when tasks are

[1] Even in regression, a function such as a sinusoid is often provided by the experimenter as the task distribution.

qualitatively different (due to difficulty or task definitions being changed too much by the physical parameters that are varied).

This phenomena, called *meta-overfitting* (or meta-underfitting, in the former, "wide" case), is not new to recent deep reinforcement learning problem settings. Domain randomization [**79**], a popular *sim2real* transfer method, faces many of the same issues when learning robotic policies purely in simulation. Here, we will show that meta-reinforcement learning has analogous issues regarding generalization, which we can attribute to the random sampling of tasks. We will then describe the repurposing of *Active Domain Randomization* (Chapter 2), which aims to learn a curriculum of tasks in unstructured task spaces. In this chapter, we address the problem of meta-overfitting by explicitly optimizing the task distribution represented by $p(\tau)$. The incorporation of a learned curriculum leads to stronger generalization performance and more robust optimization. Our results highlight the need for continued work in the analysis of the effect of task distributions on meta-RL performance and underscores the potential for curriculum learning techniques.

4.1. Motivation

We begin with a simple question:

Does the meta-training task distribution $p(\tau)$ in meta-RL really matter?

To explore the answer, we run a standard meta-reinforcement learning benchmark, *2D-Navigation-Dense*. In this environment, a point-mass must navigate to a goal, with rewards given at each timestep proportional to the Euclidean distance between the goal and the current position.

We take the hyperparameters and experiment setup from the original MAML work and simply change *the task distribution* from which the 2D goal is *uniformly* sampled. We then show generalization results of the final, meta-learned initial policy *after a single gradient step*. We then track the generalization of the one-step adaptation performance across a wide range of target goals.

In *2D-Navigation-Dense*, the training distribution prescribes goals where each coordinate is traditionally sampled between $[-0.5, 0.5]$ (the second plot in Figure 4.1) with the agent always beginning at $[0, 0]$. We then evaluate each goal in the grid between $[-2, 2]$ at 0.5 intervals, allowing us to test both in- and out-of-distribution generalization.

We see from Figure 4.1 an interesting phenomenon, particularly as the training environment shifts away from the one which samples goal coordinates $g_t \sim [-0.5, 0.5]$. While the standard environment from [**19**] generalizes reasonably well, shifting the training distribution even slightly ruins generalization of the adapted policy. What's more, when shown the

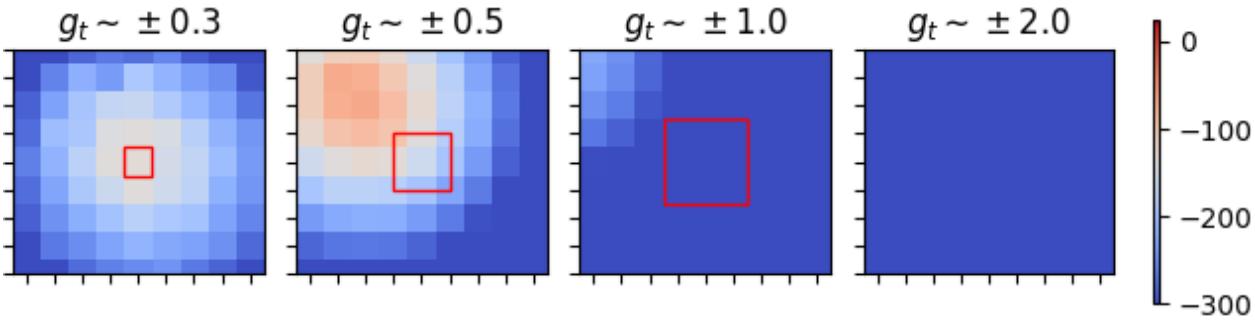


Fig. 4.1. Various agents' final adaption to a range of target tasks. The agents vary only in training task distributions, shown as red overlaid boxes. **Redder is higher reward.**

entire test distribution, MAML fails to generalize to it. We see that on even the simplest environments, the meta-training task distribution seems to have a profound effect, motivating the need for dedicating more attention towards selecting the task distribution $p(\tau)$.

Upon further inspection, we find that shifting the meta-training distribution destabilizes MAML, leading to poor performance when averaged. The first environment, where $g_t \sim [-0.3, 0.3]$ has three out of five random seeds that converge, with the latter two, $g_t \sim [-1.0, 1.0]$ and $g_t \sim [-2.0, 2.0]$, have two and one seeds that converge respectively. The original task distribution sees convergence in all five random seeds tested, hinting at a difference in stability due to the goals, and therefore task distribution, that each agent sees. This hints at a hidden importance of the task distribution $p(\tau)$, a hypothesis we explore in greater detail in the next section.

4.2. Method

As we saw in the previous section, uniformly sampling tasks from a set task distributions highly affects generalization performance of the resulting meta-learning agent. Consequently, we optimize for a curriculum over the task distribution $p(\tau)$:

$$\tau_i \sim p(\tau) \min_{\theta} \sum_{\tau_i} \mathcal{L}_{\tau_i}(f_{\theta_i'}) \tag{4.2.1}$$

where θ_i' are the updated parameters after a single meta-gradient update.

While curriculum learning has had success in scenarios where task-spaces are structured, learning curricula in unstructured task-spaces, where an intuitive scale of *difficulty* might be lacking, is an understudied topic. However, learning such curricula has seen a surge of interest in the problem of *simulation transfer* in robotics, where policies trained in simulation are transferred zero-shot (no fine-tuning) for use on real hardware. Using a method called domain randomization [**79**], several recent methods [**53, 50**] propose how to learn a curriculum of *randomizations* - which randomized environments would be most useful to show the learning agent in order to make progress on the held-out target task: the real robot.

In the meta-RL setting, the learned curriculum would be over the space of tasks. For example, in *2D-Navigation-Dense*, this would be where goals are sampled, or in *HalfCheetahVelocity*, another popular meta-RL benchmark, the goal velocity the locomotor must achieve.

As learning the curriculum is often treated as a reinforcement learning problem, it requires a reward in order to calculate policy gradients. While many of the methods from the domain randomization literature use proxies such as completion rates or average reward, the optimization scheme depends on the reward function of the *task*. In meta-learning, optimization and reward maximization on a *single* task is not the goal, and such an approach may lead to counter-intuitive results.

A more natural fit in the meta-learning scenario would be to somehow use the *qualitative difference* between the pre- and post-adaptation trajectories. Like a good teacher with a struggling student, the curriculum could shift towards where the meta-learner needs help. For example, tasks in which *negative adaptation* [**13**] occurs, or where the return from a pre-adapted agent is higher the post-adapted agent, would be prime tasks to focus on for training.

To this end, we modify *Active Domain Randomization* (ADR) to calculate such a score between the two types of trajectories. Rather than using a reference environment as in ADR, we ask a discriminator to differentiate between the pre- and post-adaptation trajectories. If a particular task generates trajectories that can be distinguished by the discriminator after adaptation, we focus more heavily on these tasks by providing the high-level optimizer, parameterized by Stein Variational Policy Gradient, a higher reward.

Concretely, we provide the particles the reward:

$$r_i = \log(f_\psi(y|D_i)) \tag{4.2.2}$$

where discriminator f_ψ produces a boolean prediction of whether the trajectory D_i is a pre-adaptation ($y = 0$) or post-adaptation ($y = 1$) trajectory. We present the algorithm, which we term Meta-ADR, in Algorithm 4. We also illustrate the algorithm in Figure 4.2.

Meta-ADR learns a curriculum in this unstructured task space without relying on the notion of task performance or reward functions. Note that Meta-ADR runs the original MAML algorithm as a subroutine, but in fact Meta-ADR can run any meta-learning subroutine (i.e Reptile [**52**], PEARL [**63**], or First-Order MAML). In this work, we abstract away the meta-learning subroutine, focusing instead on the effect of task distributions on the learner's generalization capabilities. An advantage of Meta-ADR over the ADR original formulation is that unlike ADR, Meta-ADR requires no additional rollouts, using the rollouts already required by gradient-based meta-reinforcement learners to optimize the curriculum.

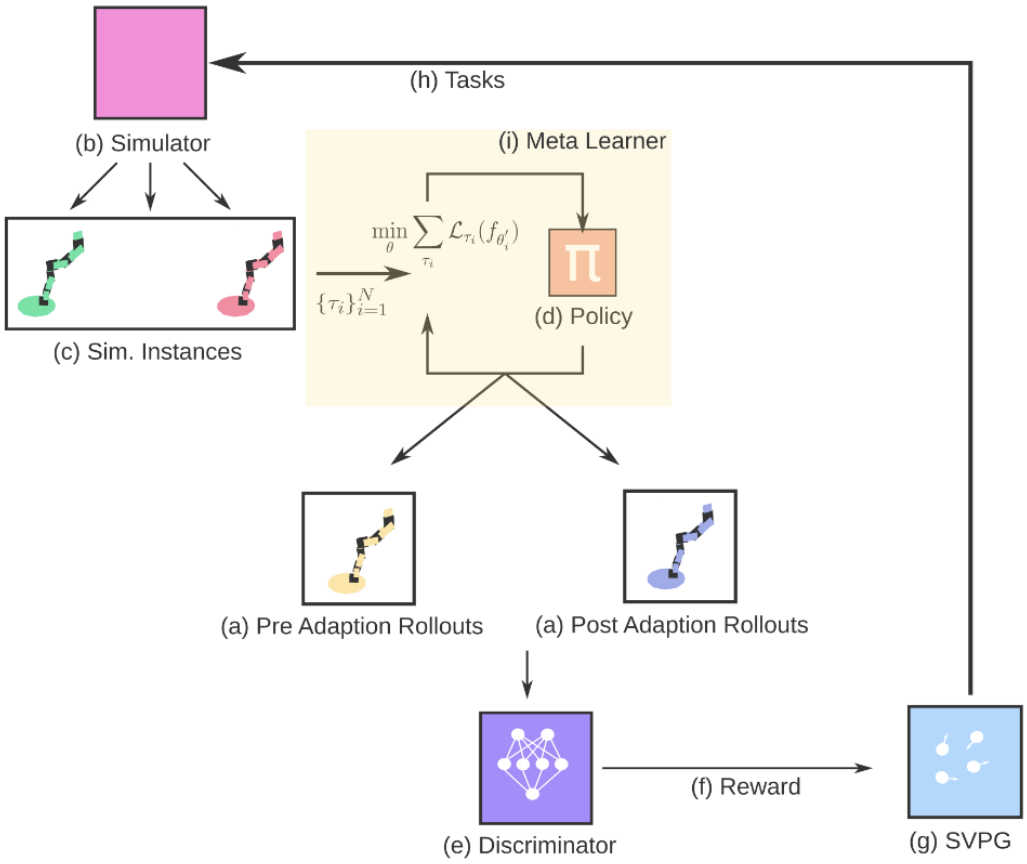


Fig. 4.2. Meta-ADR proposes tasks to a meta-RL agent, helping learn a curriculum of tasks rather than uniformly sampling them from a set distribution. A discriminator learns a reward as a proxy for task-difficulty, using pre- and post-adaptation rollouts as input. The reward is used to train SVPG particles, which find the tasks causing the meta-learner the most difficulty after adaption. The particles propose a diverse set of tasks, trying to find the tasks that are currently causing the agent the most difficulty.

4.3. Results

In this section, we show the results of uniform sampling of the standard MAML agent when changing task distribution $p(\tau)$, while also benchmarking against a MAML agent trained with a learned task distribution using Meta-ADR. All hyperparameters for each task are taken from [**19, 66**], with the exception that we take the *final* policy at the end of 200 meta-training epochs instead of the best-performing policy over 500 meta-training epochs. We use the code from [**14**] to run all of our experiments. Unless otherwise noted, all experiments are run and averaged across five random seeds. All results are shown after a single gradient step during meta-test time. For each task, we artificially create a generalization

Algorithm 4 Meta-ADR

Input Task distribution $p(\tau)$
Initialize π_θ: agent policy, μ_ϕ: SVPG particles, f_ψ: discriminator
while not *max_epochs* **do**
    for each particle μ_ϕ **do**
        sample tasks $\tau_i \sim \mu_\phi(\cdot)$, bounded by support of $p(\tau)$
    for each τ_i **do**
        D_{pre}, $D_{post} = \text{MAML}_{RL}(\pi_\theta, \tau_i)$
        Calculate r_i for τ_i using D_{post} (Eq. (4.2.2))
    // Gradient Updates
    Update particles using SVPG update rule and r_i
    Update f_ψ with D_{pre} and D_{post} using SGD.

range; potentially disjoint from the training distribution of target goals, velocities, headings, etc., and we evaluate each agent both in- and out-of-distribution.

Importantly, since our focus is on generalization, we **evaluate the final policy**, rather than the standard, *best-performing* policy. As MAML produces a final *initial* policy, when evaluating for generalization for meta-learning, we adapt that initial policy to each target task, and report the adaption results. In addition, in certain sections, we discuss *negative* adaption, which is simply the performance difference between the final, adapted policy and the final, initial policy. When this quantity is negative, as noted in [**13**], we say that the policy has *negatively* adapted to the task.

We present results from standard Meta-RL benchmarks in Sections 4.4 and 4.5, and in general find that Meta-ADR stabilizes the adaption procedure. However, this finding is not universal, as we note in Section 4.6.

In Subsections 4.6, we highlight a need for better benchmarking and failure cases (overfitting and biased, non-uniform generalization) that both Meta-ADR and uniform-sampling methods seem to suffer from.

4.4. Navigation

In this section we evaluate meta-ADR on two navigation tasks: 2D-Navigation-Dense and Ant-Navigation.

4.4.1. 2D-Navigation-Dense

We train the *same* meta-learning agent from Section 4.1 on *2D-Navigation-Dense*, except this time we use the tasks[2] proposed by Meta-ADR, using a learned curriculum to propose

[2]In navigation environments, tasks are parameterized by the `x, y` location of the goal.

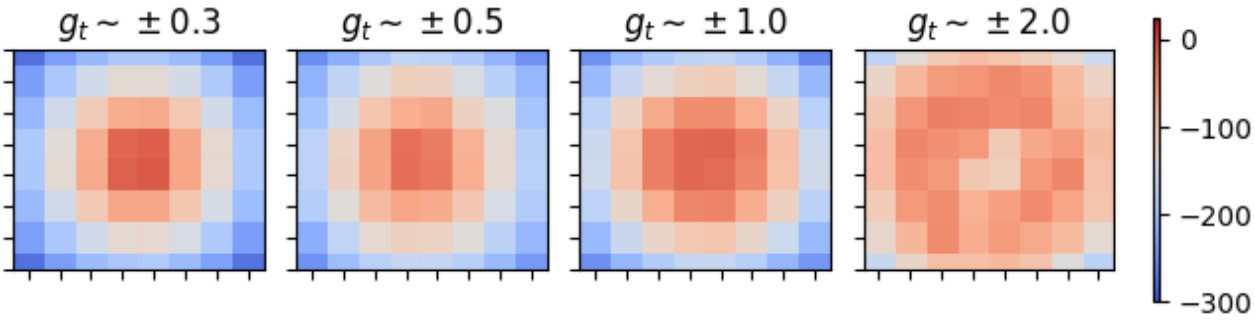


Fig. 4.3. When a curriculum of tasks is learned with Meta-ADR, we see the stability of MAML improve. **Redder is higher reward.**

the next best task for the agent. We evaluate generalization across the same scaled up square spanning the ranges of $[-2, 2]$ in both dimensions.

From Figure 4.3, we see that, with a learned curriculum, the agent generalizes much better, especially *within* the training distribution. A MAML agent trained with a Meta-ADR curriculum also generalizes out-of-distribution with much stronger performance. These results hint at the strong dependence of MAML performance and the task distribution $p(\tau)$, especially when compared to those in Figure 4.1. Learning such a task curriculum with a method such as Meta-ADR helps alleviate some instability.

4.4.2. Ant-Navigation

Interestingly, on a more complex variant of the same task, *Ant-Navigation*, the benefits of such a learned curriculum are minimized. In this task, an eight-legged locomoter is tasked with achieving goal positions sampled from a predetermined range; the standard environment samples the goal positions from a box centered at $(0, 0)$, with each coordinate sampled from $g \sim [-3, 3]$. We systematically evaluate each agent on a grid with both axes ranging between $[-7, 7]$, with a 0.5 step interval.

In Figure 4.4, we qualitatively see the same generalization across all training task distributions when comparing a randomly sampled task curriculum and a learned one. We hypothesize that this stability comes mainly from the control components of the reward, leading to a smoother, stabler performance across all training distributions. In addition, generalization is unaffected by the choice of distribution, pointing to differences between this task and the simpler version.

Compared to the *2D-Navigation-Dense*, *Ant-Navigation* also receives a dense reward related to its distance to target, specifically "control cost, a contact cost, a survival reward, and a penalty equal to its L1 distance to the target position." In comparison, the *2D-Navigation-Dense* task, while a simpler control problem, receives reward information only related to the Euclidean distance to the goal. Counter-intuitively, this simplicity results in

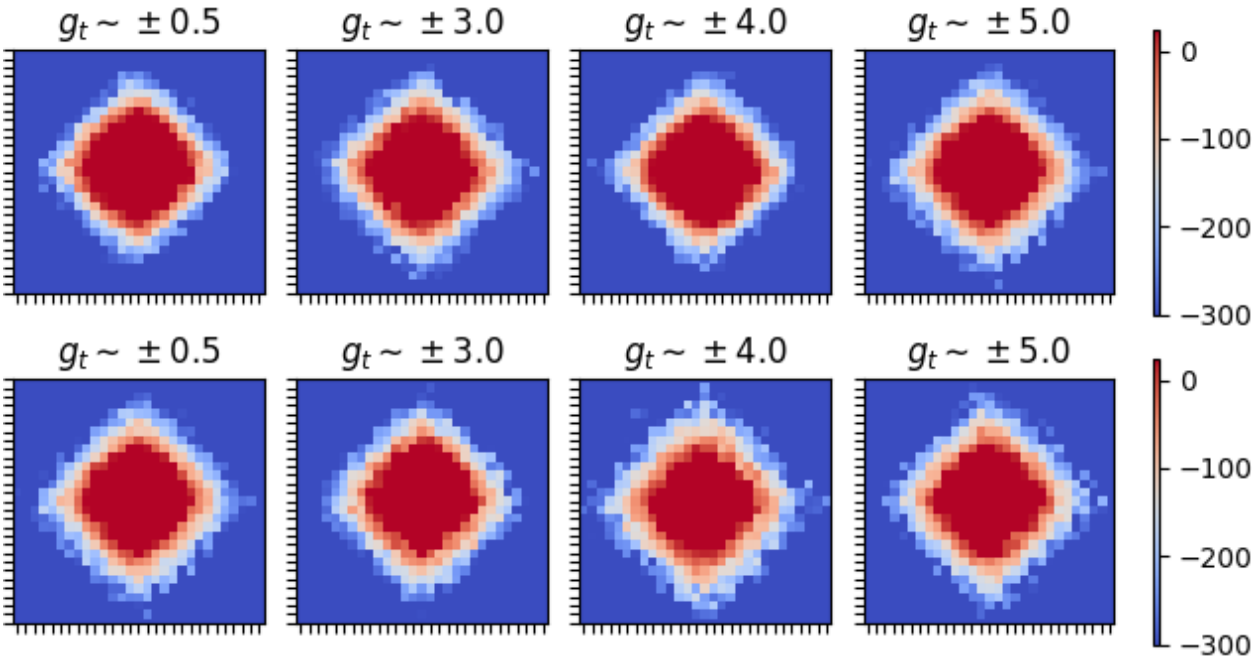


Fig. 4.4. In the Ant-Navigation task, both uniformly sampled goals (top) and a learned curriculum of goals with Meta-ADR (bottom) are stable in performance. We attribute this to the extra components in the reward function. Redder is higher reward.

less stable performance when uniformly sampling tasks, an ablation which we hope to study in future work.

4.5. Locomotion

We now consider *locomotion*, another standard meta-RL benchmark, where we are tasked with training an agent to quickly move in a particular target velocity (Section 4.5.1) or in a particular direction (Section 4.5.2). In this section, we focus on two high-dimensional continuous control problems. In the *AntVelocity*, an eight-legged locomoter must run at a specific speed, with the task space (both for learned and random curricula) being the target velocity. In *Humanoid-Direc-2D*, a benchmark introduced by [**66**], an agent must learn to run in a target direction, θ in a 2D plane.

Both tasks are extremely high-dimensional in both observation and action space. The ant has a (111×1) sized observation space, with each step requiring an action vector of length eight. The Humanoid, which takes in a (376×1) element state, requires an action vector of length 17.

4.5.1. Target Velocity Tasks

When dealing with the target velocity task, we train an ant locomoter to attain target speeds sampled from $v_t \sim [0, 3]$ (Figure 4.5 left, the standard variant of *AntVelocity*) and speeds sampled from $v_t \sim [0, 5]$ (Figure 4.5 right).

Fig. 4.5. Ant-Velocity sees less of a benefit from curriculum, but performance is greatly affected by a correctly-calibrated task distribution (left). In a miscalibrated one (right), we see that performance from a learned curriculum is slightly more stable.

While we see that learned curricula make insignificant amounts of performance *improvement* over random sampling when shown the same task distribution, we see large differences in performance between task distributions, motivating our hypothesis that $p(\tau)$ is a crucial hyperparameter for successful meta-RL. In additon, we notice that the highest scores are attained on the velocities closer to the easiest variant of the task: a $v_t = 0$, which requires the locomoter to stand completely still.

4.5.2. Humanoid Directional

In the standard variant of *Humanoid-Direc-2D*, a locomoter is tasked with running in a particular direction, sampled from $[0, 2\pi]$. This task makes no distinction regarding target velocity, but rather calculates the reward based on the agent's heading and other control costs.

In this task, we shift the distribution from $[0, 2\pi]$ to subsets of this range, subsequently training and evaluating MAML agents across the entire range of tasks between $[0, 2\pi]$, as seen in the first two panels of Figure 4.6. Again, we compare agents trained with the standard

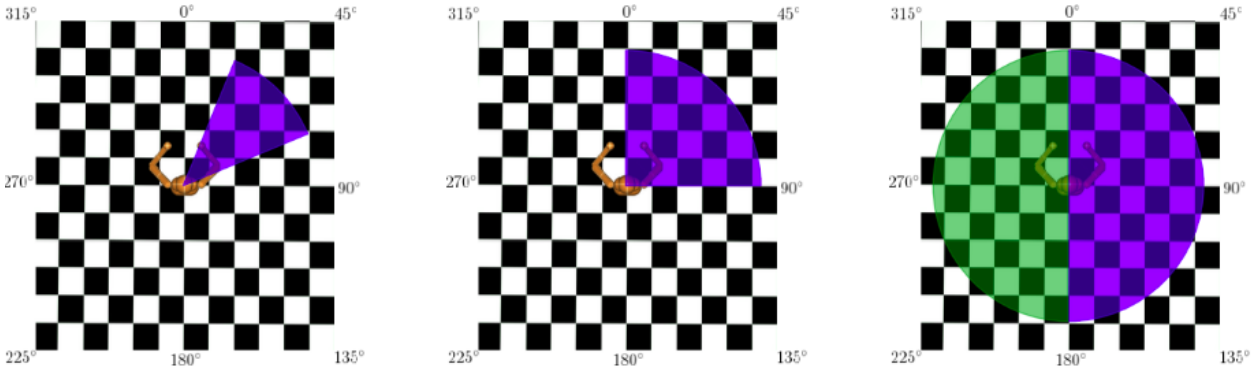


Fig. 4.6. In the high-dimensional Humanoid Directional task, we evaluate many different training distributions to understand the effect of $p(\tau)$ on generalization in difficult continuous control environments. In particular, we focus on *symmetric* variants of tasks - task distributions that mirror each other, such as $0-\pi$ and $\pi-2\pi$ in the right panel. Intuitively, when averaged over many trials, such mirrored distrbutions should produce similar trends of in and out-of-distribution generalization.

Tableau 4.1. We compare agents trained with random curricula on different but symmetric task distributions $p(\tau)$. Changing the distribution leads to counter-intuitive drops in performance on tasks both in- and out-of-distribution.

p(τ)	$\theta = 0$	$\theta = 30$	$\theta = 60$
0 - 360	*62.39±7.01*	*64.7±5.38*	*99.93±77.5*
0 - 60	71.51±16.74	82.95±32.06	95.96±37.49
0 - 180	171.77±117.8	221.4±91.66	87.78±45.41
300 - 360	65.64±10.42	95.21±40.08	105.4±50.75
180 - 360	134.52±70.07	79.69±26.01	59.52±2.73

uniformly-random sampled task distribution against those trained with a learned curriculum using Meta-ADR.

When studying the generalization capabilities on this difficult continuous control task, we are particularly interested in *symmetric* versions of the task; for example, tasks that sample the right and left semi-circles of the task space. We repeat this experiment with many variants of this symmetric task description, and report representative results due to space in Figure 4.7.

When testing various training distributions, we find that, in general, learned curricula stabilize the algorithm. We see more consistent performance increases, with smaller losses in performance in the directions that *UniformSampling-MAML* outperforms the learned curriculum. However, as noted in Tables 4.1 and 4.2, we see that again, the task distribution $p(\tau)$ is an extremely sensitive hyperparameter, causing large shifts in performance when uniformly sampling from those ranges. Worse, this hyperparameter seems to cause *counter-intuitive* gains and drops in performance, both on in *and* out-of-distribution tasks.

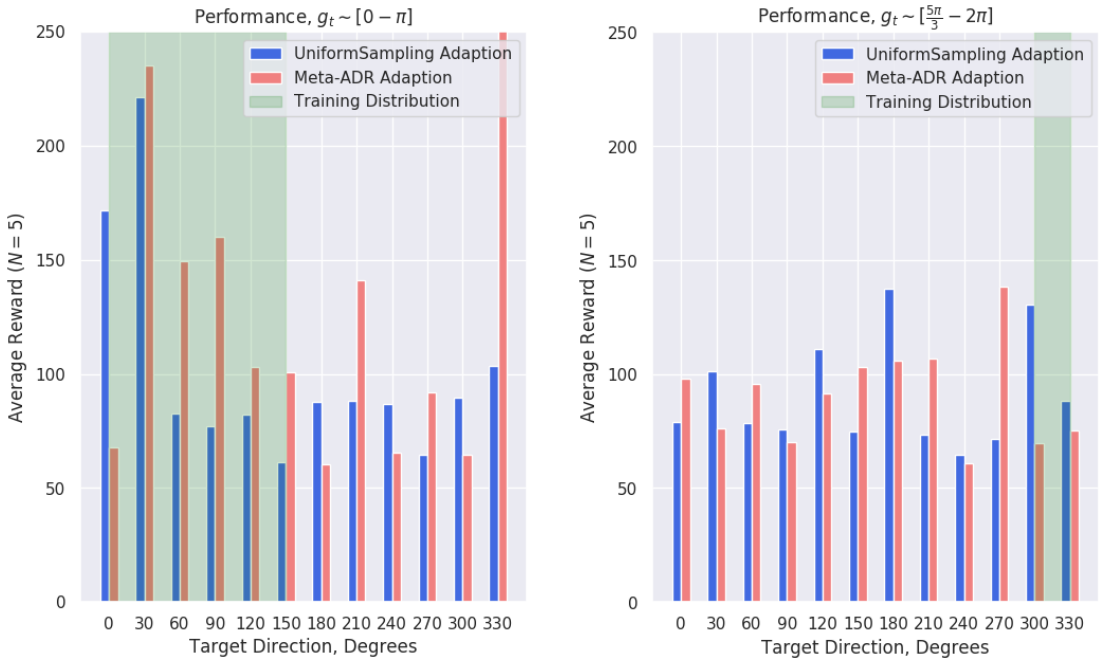


Fig. 4.7. In complex, high-dimensional environments, training task distributions can wildly vary performance. Even in the Humanoid Directional task, Meta-ADR allows MAML to generalize across the range, although it too is affected in terms of total return when compared to the same algorithm trained with "good" task distributions.

Tableau 4.2. Evaluating tasks that are *qualitatively* similar, for example running at a heading offset from the starting heading by 30 degrees to the *left or right*, leads to different performances from the same algorithm.

p(τ)	$\theta = 180$	$\theta = 330$	$\theta = 300$
0 - 360	*154.1±121.6*	*81.26±16.39*	*75.47±18.92*
0 - 60	120.2±62.74	84.34±35.9	126.2±101.3
0 - 180	87.78±45.41	103.5±87.24	89.49±31.0
300 - 360	116.03±52.44	81.25±43.82	100.45±48.41
180 - 360	99.1±88.85	80.52±21.79	80.67±21.82

While learned curricula seem to help in such a task, a more important consideration from many of these experiments is the variance in performance *between* tasks. As *generalization* across evaluation tasks is a difficult metric to characterize due to the inherent issues when *comparing* methods, it is tempting to take the best performing tasks, or average across the whole range. However, as we show in the remaining sections, closer inspection on each of the above experiments sheds light on major issues with the evaluation approaches standard in the meta-RL community today.

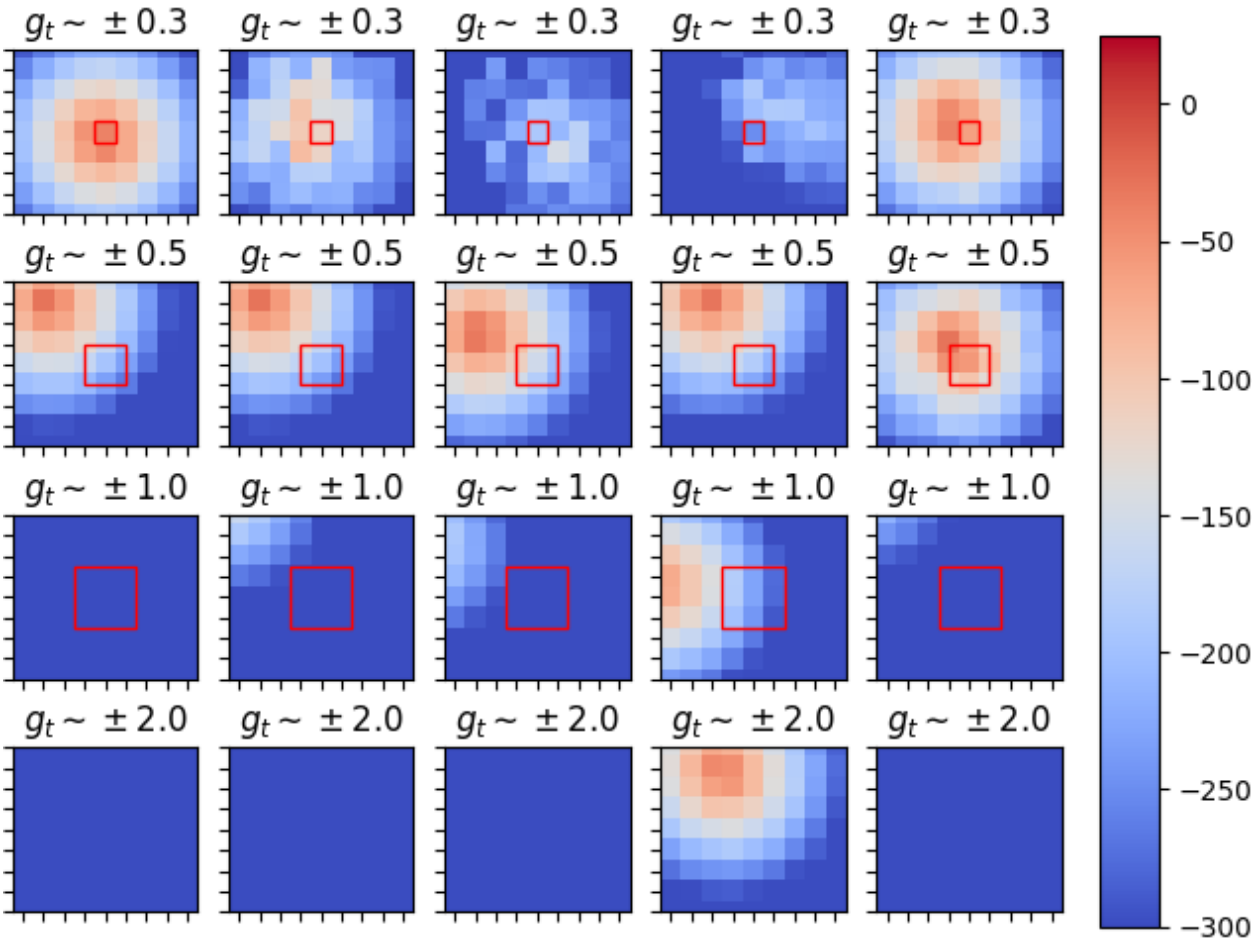


Fig. 4.8. Uniform sampling causes MAML to show bias towards certain tasks, with the effect being compounded with instability when using "bad" task distributions, here shown as $\pm 0.3, \pm 1.0, \pm 2.0$ in the 2D-Navigation-Dense environment.

4.6. Failure Cases of MAML

In this section, we discuss intermediate and auxillary results from each of our previous experiments, highlighting uninterpretable algorithm bias, meta-overfitting, and performance benchmarking in meta-RL.

4.6.1. Non-Uniform Generalization

To readers surprised by the poor generalization capabilities of MAML on such a simple task seen in Figure 4.1, we offer Figure 4.8, an unfiltered look at each seed used to calculate each image in Figure 4.1.

What we immediately notice is the high variance in all but the standard variant of the task, an agent trained on goals with coordinates sampled from $g_t \sim [-0.5, 0.5]$. We even see a reoccurring bias towards certain tasks (visualized as the *top-left* of the grid). Interestingly,

when changing the uniform sampling to a learned curriculum, we no longer see such high-variance in convergence across tasks. While our results seem in opposition to many works in the meta-reinforcement learning area, we restate that in our setting, we can only evaluate the *final* policy, as the notion of *best*-performing loses much of its meaning when evaluating for generalization.

4.6.2. Meta-Overfitting

Many works in the meta-reinforcement learning setting focus on final adaption *performance*, but few works focus on the *loss of performance* after the adaption step. Coined by [**13**] as *negative adaption*, the definition is simple: the loss in performance *after* a gradient step at meta-test time. Negative adaptation occurs when a pre-adaption policy has overfit to a particular task during meta-training. During meta-test time, an additional gradient steps degrade performance, leading to negative adaptation.

We extensively evaluate negative adaption in the *Humanoid-Direc-2D* benchmark described in Section 4.5.2, providing correlation results between performance and the *difference* between the post- and pre-adaption performance.

When we systematically evaluate negative adaption across all tested *Humanoid-Direc-2D* training distributions, we notice an interesting correlation between performance and the amount of negative-adaptation. Both methods produce near-linear relationships between the two quantities, but when evaluating generalization, we need to focus on the left-hand side of the x-axis, where policies already are performing poorly, and what qualitative effects extra gradient steps have.

We notice a characteristic sign of *meta-overfitting*, where strongly performing policies continue to perform well, but poorly performing ones stagnate, or more often, degrade in performance. When tested, Meta-ADR does not help in this regard, despite having slightly stronger final performance in tasks.

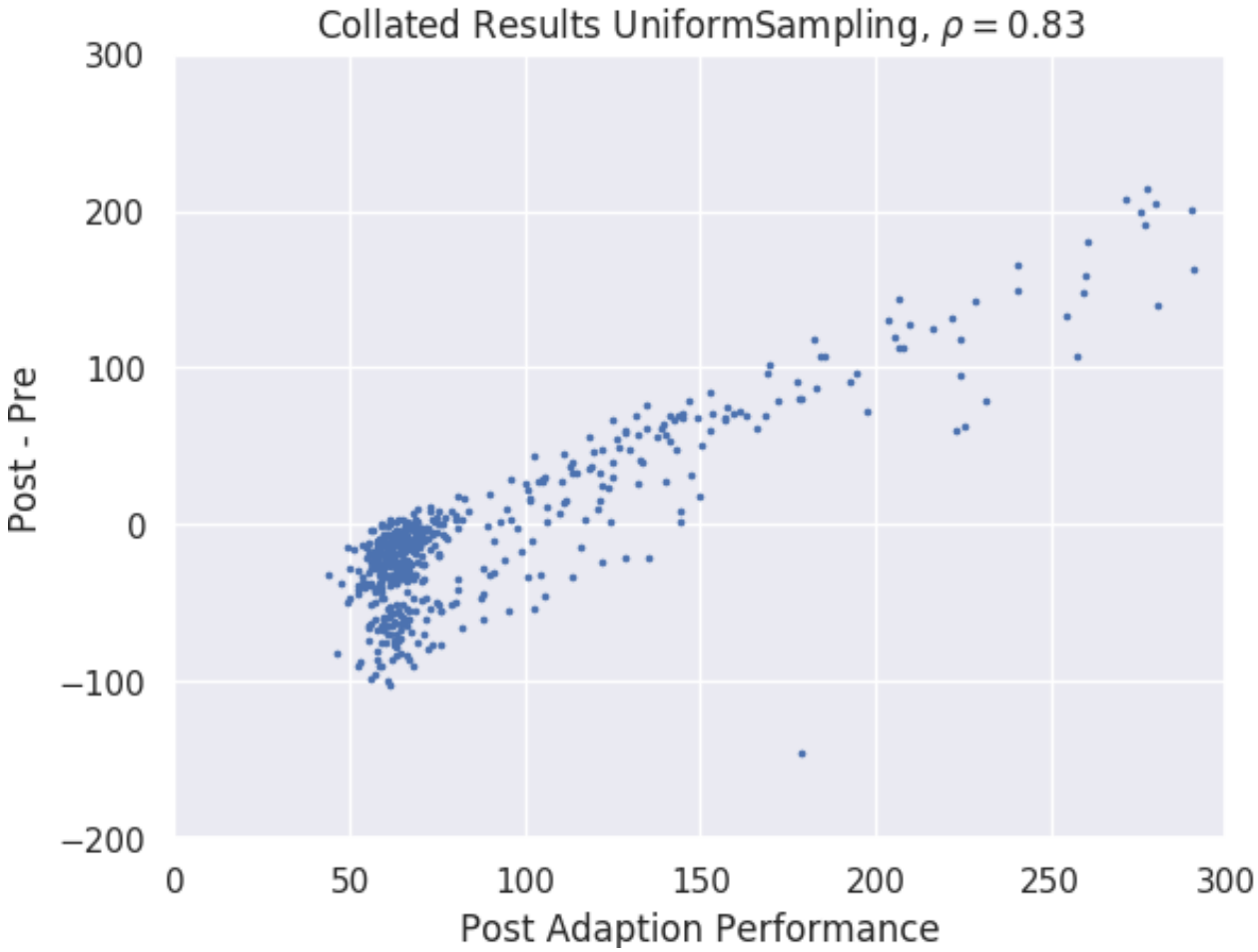


Fig. 4.9. When we correlate the final performance (x-axis) with the quality of *adaption* (denoted as `post-pre` on the y-axis), we see a troubling trend. MAML seems to overfit to certain tasks, with many tasks that were already neglected during training showing worse post-adaptation returns.

Chapter 5

Conclusion

Data distributions are central to machine learning; despite being treated as a *given* in most work, they have complex interplays with models, optimizers, learning dynamics, and learning algorithms. Reinforcement learning - with its non-stationarity of data distribution - complicates analysis even further. Meta-reinforcement learning generates *i.i.d* samples of these *non-i.i.d* learning scenarios, mixing learning updates of supervised and reinforcement learning.

In this thesis, we explored the idea of generating curricula in unstructured task spaces and proposed Active Domain Randomization as a way to solve some of the core issues. We showed strong results in zero-shot, few-shot, and meta-learning settings, and provided some initial analysis on the effects of curricula on optimization.

Multi-task and transfer learning are core to deploying machine learning in the wild. A concern is that the community does even have a unified language with which to discuss multi-task learning: When will these systems fail? When will they work? Can we know in advance?

Multi-task learning comes in many variants and deserves to be treated as such. In the future, work needs to highlight these flavors of multi-task learning, bringing the field to firmer theoretical ground. From here, we can fruitfully discuss benefits and shortcomings, and from which angles we can systematically attack the problems of multi-task learning.

Références bibliographiques

[1] Zafarali Ahmed, Nicolas Le Roux, Mohammad Norouzi et Dale Schuurmans : Understanding the impact of entropy on policy optimization. *CoRR*, abs/1811.11214, 2018.

[2] Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray *et al.* : Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177*, 2018.

[3] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel et Wojciech Zaremba : Hindsight experience replay, 2017.

[4] Yoshua Bengio, Jérôme Louradour, Ronan Collobert et Jason Weston : Curriculum learning. *In Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 41–48, New York, NY, USA, 2009. ACM.

[5] J Bongard et Hod Lipson : Once more unto the breach: Co-evolving a robot and its simulator. *In Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems (ALIFE9)*, 2004.

[6] Eric Brochu, Vlad M. Cora et Nando de Freitas : A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *CoRR*, abs/1012.2599, 2010.

[7] Eric Brochu, Vlad M. Cora et Nando de Freitas : A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *CoRR*, abs/1012.2599, 2010.

[8] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang et Wojciech Zaremba : Openai gym, 2016.

[9] Arslan Chaudhry, Puneet K. Dokania, Thalaiyasingam Ajanthan et Philip H. S. Torr : Riemannian walk for incremental learning: Understanding forgetting and intransigence. *In The European Conference on Computer Vision (ECCV)*, September 2018.

[10] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff et Dieter Fox : Closing the sim-to-real loop: Adapting simulation randomization with real world experience. *arXiv preprint arXiv:1810.05687*, 2018.

[11] Karl Cobbe, Oleg Klimov, Christopher Hesse, Taehoon Kim et John Schulman : Quantifying generalization in reinforcement learning. *CoRR*, abs/1812.02341, 2018.

[12] Erwin Coumans : Bullet physics simulation. *In ACM SIGGRAPH 2015 Courses*, SIGGRAPH '15, New York, NY, USA, 2015. ACM.

[13] Tristan Deleu et Yoshua Bengio : The effects of negative adaptation in Model-Agnostic Meta-Learning. *CoRR*, abs/1812.02159, 2018.

[14] Tristan Deleu et Hosseini Seyedarian Guiroy, Simon : Reinforcement Learning with Model-Agnostic Meta-Learning in PyTorch, 2018.

[15] Laurent Dinh, Razvan Pascanu, Samy Bengio et Yoshua Bengio : Sharp minima can generalize for deep nets. *CoRR*, abs/1703.04933, 2017.

[16] Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever et Pieter Abbeel : RL2: Fast Reinforcement Learning via Slow Reinforcement Learning. 2016.

[17] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz et Sergey Levine : Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.

[18] Jesse Farebrother, Marlos C. Machado et Michael Bowling : Generalization and regularization in DQN, 2018.

[19] Chelsea Finn, Pieter Abbeel et Sergey Levine : Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. *International Conference on Machine Learning*, 2017.

[20] Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel et Sergey Levine : One-Shot Visual Imitation Learning via Meta-Learning. *Conference on Robot Learning*, 2017.

[21] Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang et Pieter Abbeel : Reverse curriculum generation for reinforcement learning, 2017.

[22] Florian Golemo, Adrien Ali Taiga, Aaron Courville et Pierre-Yves Oudeyer : Sim-to-real transfer with neural-augmented robot simulation. *In Conference on Robot Learning*, 2018.

[23] Jackson Gorham et Lester Mackey : Measuring sample quality with stein's method, 2015.

[24] Alex Graves, Marc G. Bellemare, Jacob Menick, Remi Munos et Koray Kavukcuoglu : Automated curriculum learning for neural networks, 2017.

[25] Abhishek Gupta, Russell Mendonca, YuXuan Liu, Pieter Abbeel et Sergey Levine : Meta-reinforcement learning of structured exploration strategies, 2018.

[26] Swaminathan Gurumurthy, Sumit Kumar et Katia Sycara : Mame : Model-agnostic meta-exploration, 2019.

[27] Tuomas Haarnoja, Vitchyr Pong, Aurick Zhou, Murtaza Dalal, Pieter Abbeel et Sergey Levine : Composable deep reinforcement learning for robotic manipulation. *arXiv preprint arXiv:1803.06773*, 2018.

[28] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, S. M. Ali Eslami, Martin Riedmiller et David Silver : Emergence of locomotion behaviours in rich environments, 2017.

[29] Jonathan Ho et Stefano Ermon : Generative adversarial imitation learning. *CoRR*, abs/1606.03476, 2016.

[30] Sepp Hochreiter et Jürgen Schmidhuber : Flat minima. *Neural Computation*, 9(1):1–42, 1997.

[31] Andrej Karpathy : Multi-task learning in the wild.

[32] Rawal Khirodkar, Donghyun Yoo et Kris M Kitani : Vadra: Visual adversarial domain randomization and augmentation. *arXiv preprint arXiv:1812.00491*, 2018.

[33] Taesup Kim, Jaesik Yoon, Ousmane Dia, Sungwoong Kim, Yoshua Bengio et Sungjin Ahn : Bayesian model-agnostic meta-learning. *CoRR*, abs/1806.03836, 2018.

[34] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran et Raia Hadsell : Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

[35] James Kirkpatrick, Razvan Pascanu, Neil C. Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran et Raia Hadsell : Overcoming catastrophic forgetting in neural networks. *CoRR*, abs/1612.00796, 2016.

[36] Gregory Koch, Richard Zemel et Ruslan Salakhutdinov : Siamese Neural Networks for One-shot Image Recognition. *International Conference on Machine Learning*, 2015.

[37] Vijay Konda et John Tsitsiklis : Actor-critic algorithms. *In SIAM Journal on Control and Optimization*, pages 1008–1014. MIT Press, 2000.

[38] Matthieu Lapeyre : *Poppy: open-source, 3D printed and fully-modular robotic platform for science, art and education*. Theses, Université de Bordeaux, novembre 2014.

[39] Benjamin Letham, Brian Karrer, Guilherme Ottoni et Eytan Bakshy : Constrained bayesian optimization with noisy experiments. *Bayesian Analysis*, 14(2):495–519, Jun 2019.

[40] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver et Daan Wierstra : Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[41] Qiang Liu, Jason D. Lee et Michael I. Jordan : A kernelized stein discrepancy for goodness-of-fit tests and model evaluation, 2016.

[42] Qiang Liu et Dilin Wang : Stein variational gradient descent: A general purpose bayesian inference algorithm. *In Advances in neural information processing systems*, pages 2378–2386, 2016.

[43] Yang Liu, Prajit Ramachandran, Qiang Liu et Jian Peng : Stein variational policy gradient, 2017.

[44] Bhairav Mehta : Depth first learning: Stein variational gradient descent.

[45] Bhairav Mehta, Manfred Diaz, Florian Golemo, Christopher J. Pal et Liam Paull : Active domain randomization. *Conference on Robot Learning*, Edition 2, 2010.

[46] Bhairav Mehta, Manfred Diaz, Florian Golemo, Christopher J. Pal et Liam Paull : Active domain randomization. *CoRR*, abs/1904.04762, 2019.

[47] Lars Mescheder, Andreas Geiger et Sebastian Nowozin : Which training methods for GANs do actually converge? *In International Conference on Machine Learning*, 2018.

[48] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen et Pieter Abbeel : A Simple Neural Attentive Meta-Learner. *International Conference on Learning Representations*, 2018.

[49] J. B. Mockus et L. J. Mockus : Bayesian approach to global optimization and application to multiobjective and constrained problems. *J. Optim. Theory Appl.*, 70(1):157–172, juillet 1991.

[50] Melissa Mozifian, Juan Camilo Gamboa Higuera, David Meger et Gregory Dudek : Learning domain randomization distributions for transfer of locomotion policies. *CoRR*, abs/1906.00410, 2019.

[51] Tsendsuren Munkhdalai et Hong Yu : Meta Networks. *International Conference on Machine Learning*, 2017.

[52] Alex Nichol, Joshua Achiam et John Schulman : On first-order meta-learning algorithms, 2018.

[53] 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, Jadwiga Tworek, Peter Welinder, Lilian Weng, Qi-Ming Yuan, Wojciech Zaremba et Lefei Zhang : Solving rubik's cube with a robot hand. *ArXiv*, abs/1910.07113, 2019.

[54] 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 et Lei Zhang : Solving rubik's cube with a robot hand, 2019.

[55] Charles Packer, Katelyn Gao, Jernej Kos, Philipp Krähenbühl, Vladlen Koltun et Dawn Song : Assessing generalization in deep reinforcement learning, 2018.

[56] Eunbyung Park et Junier B. Oliva : Meta-curvature. *CoRR*, abs/1902.03356, 2019.

[57] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba et Pieter Abbeel : Sim-to-real transfer of robotic control with dynamics randomization. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018.

[58] Lerrel Pinto, James Davidson, Rahul Sukthankar et Abhinav Gupta : Robust adversarial reinforcement learning, 2017.

[59] Lerrel Pinto, James Davidson, Rahul Sukthankar et Abhinav Gupta : Robust adversarial reinforcement learning, 2017.

[60] Vitchyr H. Pong, Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl et Sergey Levine : Skew-fit: State-covering self-supervised reinforcement learning, 2019.

[61] L. Y. Pratt : Discriminability-based transfer between neural networks. *In Proceedings of the 5th International Conference on Neural Information Processing Systems*, NIPS'92, page 204–211, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.

[62] Aravind Rajeswaran, Kendall Lowrey, Emanuel Todorov et Sham Kakade : Towards generalization and simplicity in continuous control. *CoRR*, abs/1703.02660, 2017.

[63] Kate Rakelly, Aurick Zhou, Deirdre Quillen, Chelsea Finn et Sergey Levine : Efficient Off-Policy Meta-Reinforcement Learning via Probabilistic Context Variables. *International Conference on Machine Learning*, 2019.

[64] Sachin Ravi et Hugo Larochelle : Optimization as a model for few-shot learning. *International Conference on Learning Representations*, 2017.

[65] Ajemian Liu Rish Tu Riemer, Cases et Tesauro : Learning to learn without forgetting by maximizing transfer and minimizing interference. *arXiv preprint arXiv:1810.11910*, 2018.

[66] Jonas Rothfuss, Dennis Lee, Ignasi Clavera, Tamim Asfour et Pieter Abbeel : Promp: Proximal meta-policy search, 2018.

[67] Sebastian Ruder : *Neural Transfer Learning for Natural Language Processing.* Thèse de doctorat, National University of Ireland, Galway, 2019.

[68] Nataniel Ruiz, Samuel Schulter et Manmohan Chandraker : Learning to simulate, 2018.

[69] Fereshteh Sadeghi et Sergey Levine : (cad)$ˆ2$rl: Real single-image flight without a single real image. *CoRR*, abs/1611.04201, 2016.

[70] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra et Timothy Lillicrap : Meta-Learning with Memory-Augmented Neural Networks. *International Conference on Machine Learning*, 2016.

[71] John Schulman, Pieter Abbeel et Xi Chen : Equivalence between policy gradients and soft q-learning. *CoRR*, abs/1704.06440, 2017.

[72] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan et Pieter Abbeel : Trust Region Policy Optimization, 2015.

[73] Jake Snell, Kevin Swersky et Richard S. Zemel : Prototypical Networks for Few-shot Learning. *Conference on Neural Information Processing Systems*, 2017.

[74] Bradly C. Stadie, Ge Yang, Rein Houthooft, Xi Chen, Yan Duan, Yuhuai Wu, Pieter Abbeel et Ilya Sutskever : Some considerations on learning to explore via meta-reinforcement learning. 2018.

[75] Charles Stein : A bound for the error in the normal approximation to the distribution of a sum of dependent random variables. *In Proceedings of the Sixth Berkeley Symposium on Mathematical Statistics*

and Probability, Volume 2: Probability Theory, pages 583–602, Berkeley, Calif., 1972. University of California Press.

[76] Richard S Sutton et Andrew G Barto : *Reinforcement Learning: An introduction*. MIT Press, 2018.

[77] Richard S. Sutton, David McAllester, Satinder Singh et Yishay Mansour : Policy gradient methods for reinforcement learning with function approximation. *In Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, page 1057–1063, Cambridge, MA, USA, 1999. MIT Press.

[78] Naftali Tishby et Noga Zaslavsky : Deep learning and the information bottleneck principle. *CoRR*, abs/1503.02406, 2015.

[79] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba et Pieter Abbeel : Domain randomization for transferring deep neural networks from simulation to the real world. *In Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE, 2017.

[80] Emanuel Todorov, Tom Erez et Yuval Tassa : Mujoco: A physics engine for model-based control. *In IROS*. IEEE, 2012.

[81] Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio et Geoffrey J Gordon : An empirical study of example forgetting during deep neural network learning. *arXiv preprint arXiv:1812.05159*, 2018.

[82] Simon Tong : *Active Learning: Theory and Applications*. Thèse de doctorat, Stanford University USA, 2001. AAI3028187.

[83] Yulia Tsvetkov, Manaal Faruqui, Wang Ling, Brian MacWhinney et Chris Dyer : Learning the curriculum with Bayesian optimization for task-specific word representation learning. *In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 130–139, Berlin, Germany, août 2016. Association for Computational Linguistics.

[84] Oriol Vinyals, Charles Blundell, Timothy P. Lillicrap, Koray Kavukcuoglu et Daan Wierstra : Matching Networks for One Shot Learning. *Conference on Neural Information Processing Systems*, 2016.

[85] Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran et Matt Botvinick : Learning to reinforcement learn. 2016.

[86] Rui Wang, Joel Lehman, Jeff Clune et Kenneth O. Stanley : Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions, 2019.

[87] Ziyu Wang, Masrour Zoghi, Frank Hutter, David Matheson et Nando De Freitas : Bayesian optimization in high dimensions via random embeddings. *In Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, 2013.

[88] Wikipedia : Tay (bot) wikipedia page.

[89] Ronald J. Williams : Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, May 1992.

[90] Ronald J. Williams : Simple statistical gradient-following algorithms for connectionist reinforcement learning. *In Machine Learning*, 1992.

[91] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn et Sergey Levine : Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning, 2019.

[92] Juan Cristóbal Zagal, Javier Ruiz-del Solar et Paul Vallejos : Back to reality: Crossing the reality gap in evolutionary robotics. *IFAC Proceedings Volumes*, 37(8), 2004.

[93] Chiyuan Zhang, Qianli Liao, Alexander Rakhlin, Brando Miranda, Noah Golowich et Tomaso Poggio : Memo no . 067 june 27 , 2017 theory of deep learning iii : Generalization properties of sgd. 2017.

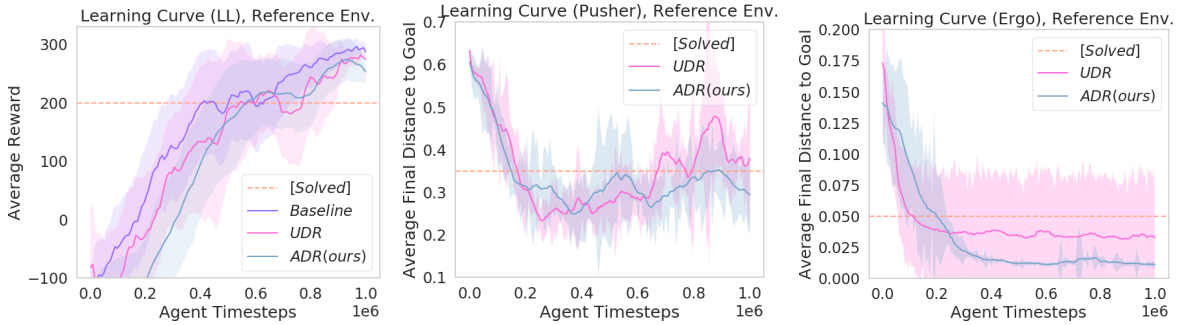
[94] Brian D Ziebart : *Modeling Purposeful Adaptive Behavior with the Principle of Maximum Causal Entropy*. Thèse de doctorat, CMU, 2010.

Annexe A

Appendices

A.1. Learning Curves for Reference Environments

For space concerns, we show only the *hard* generalization curves for all environments in the main document. For completeness, we include learning curves on the reference environment here.



(a) (b) (c)

Fig. A.1. Learning curves over time reference environments. **(a)** LunarLander **(b)** Pusher-3Dof **(c)** ErgoReacher.

A.2. Interpretability Benefits of ADR

One of the secondary benefits of ADR is its insight into incompatibilities between the task and randomization ranges. We demonstrate the simple effects of this phenomenon in a one-dimensional `LunarLander-v2`, where we only randomize the main engine strength. Our initial experiments varied this parameter between 6 and 20, which lead to ADR learning degenerate agent policies by learning to propose the lopsided blue distribution in Figure A.2. Upon inspection of the simulation, we see that when the parameter has a value of less than approximately 8, the task becomes almost impossible to solve due to the other environment factors (in this case the lander always hits the ground too fast, which it is penalized for).

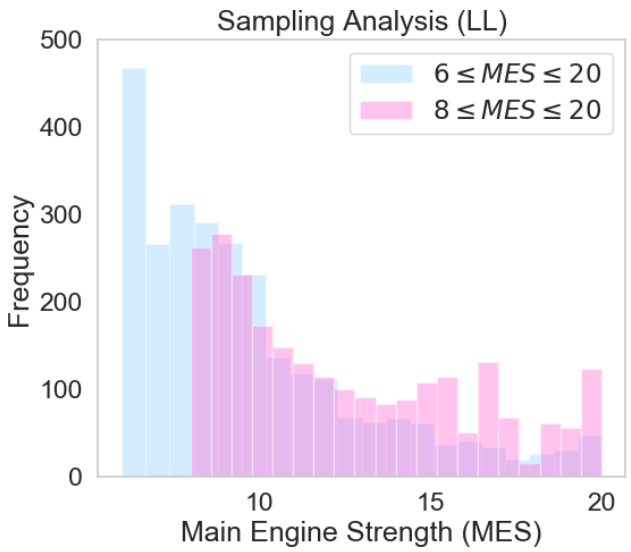


Fig. A.2. Sampling frequency across engine strengths when varying the randomization ranges. The updated, red distribution shows a much milder unevenness in the distribution, while still learning to focus on the harder instances.

After adjusting the parameter ranges to more sensible values, we see a better sampled distribution in pink, which still gives more preference to the hard environments in the lower engine strength range. Most importantly, ADR allows for analysis that is both *focused* - we know exactly what part of the simulation is causing trouble - and *pre-transfer*, i.e. done before a more expensive experiment such as real robot transfer has taken place. With UDR, the agents would be equally trained on these degenerate environments, leading to policies with potentially undefined behavior (or, as seen in Section 2.4.4, unlearn good behaviors) in these truly out-of-distribution simulations.

A.3. Bootstrapping Training of New Agents

Unlike DR, ADR's learned sampling strategy and discriminator can be reused to train new agents from scratch. To test the transferability of the sampling strategy, we first train an instance of ADR on `LunarLander-v2`, and then extract the SVPG particles and discriminator. We then replace the agent policy with an random network initialization, and once again train according the the details in Section 2.4.1. From Figure 3(a), it can be seen that the bootstrapped agent generalization is even better than the one learned with ADR from scratch. However, its training speed on the default environment ($\xi_{MES} = 13$) is relatively lower.

A.4. Environment Details

Please see Table A.1.

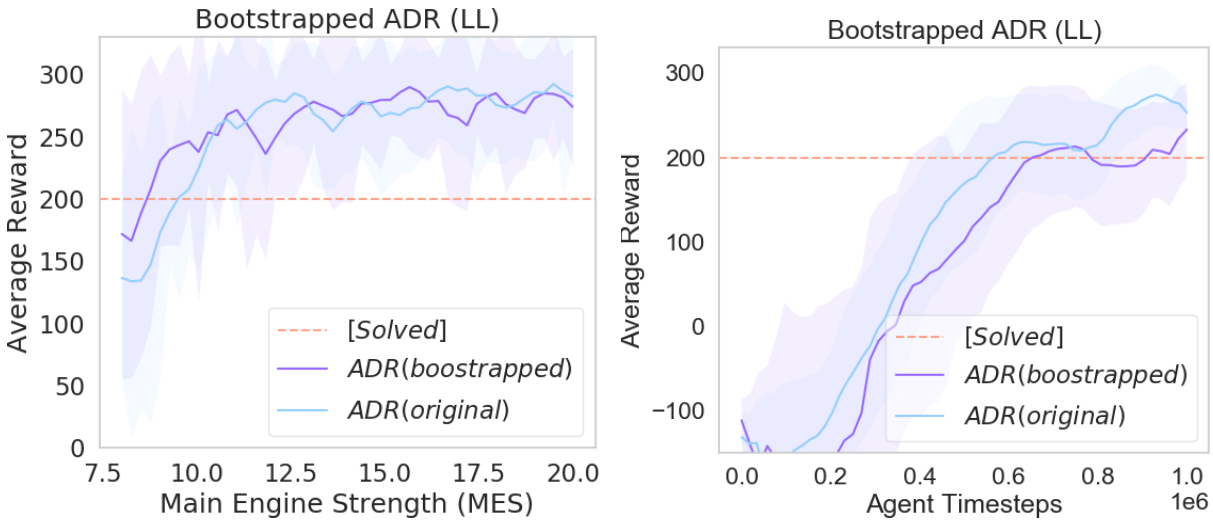


Fig. A.3. Generalization and default environment learning progression on `LunarLander-v2` when using ADR to bootstrap a new policy. Higher is better.

A.4.1. Catastrophic Failure States in ErgoReacher

In Figure A.4, we show an example progression to a *catastrophic failure state* in the held-out, simulated target environment of `ErgoReacher-v0`, with extremely low torque and gain values.

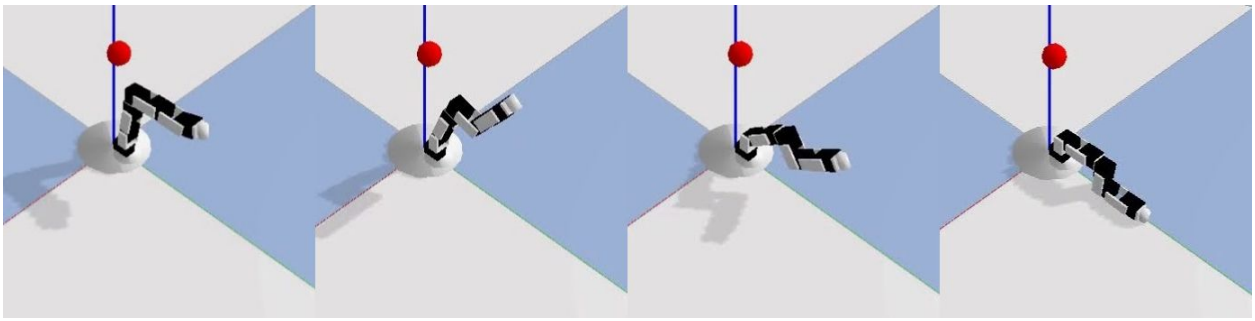


Fig. A.4. An example progression (left to right) of an agent moving to a catastrophic failure state (Panel 4) in the hard `ErgoReacher-v0` environment.

A.5. Untruncated Plots for Lunar Lander

All policies on Lunar Lander described in our paper receive a *Solved* score when the engine strengths are above 12, which is why truncated plots are shown in the main document. For clarity, we show the full, untruncated plot in Figure A.5.

Environment	N_{rand}	Types of Randomizations	Train Ranges	Test Ranges
`LunarLander-v2`	1	Main Engine Strength	$[8, 20]$	$[8, 11]$
`Pusher-3DOF-v0`	2	Puck Fric., Puck Joint Damping	$[0.67, 1.0]\times$ def.	$[0.5, 0.67]\times$ def.
`ErgoPusher-v0`	2	Puck Fric., Puck Joint Damping	$[0.67, 1.0]\times$ def.	$[0.5, 0.67]\times$ def.
`ErgoReacher-v0`	8	Joint Damping	$[0.3, 2.0]\times$ def.	$0.2\times$ def.
		Joint Max Torque	$[1.0, 4.0]\times$ def.	def.

Tableau A.1. We summarize the environments used, as well as characteristics about the randomizations performed in each environment.

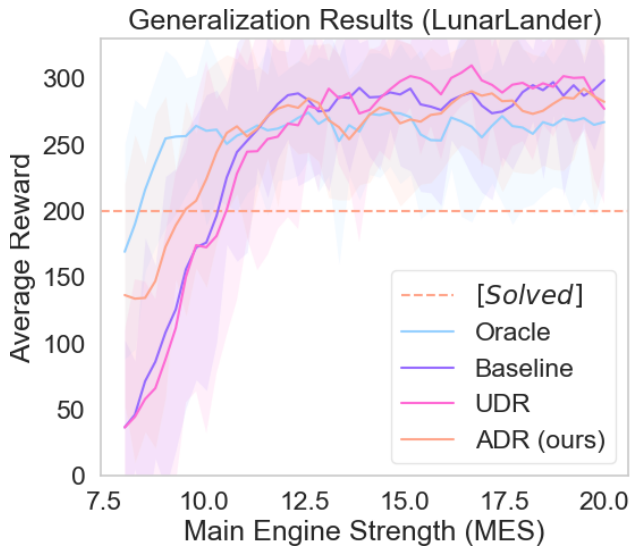


Fig. A.5. Generalization on `LunarLander-v2` for an expert interval selection, ADR, and UDR. Higher is better.

A.6. Network Architectures and Experimental Hyperparameters

All experiments can be reproduced using our Github repository[1].

All of our experiments use the same network architectures and experiment hyperparameters, except for the number of particles N. For any experiment with `LunarLander-v2`, we use $N = 10$. For both other environments, we use $N = 15$. All other hyperparameters and network architectures remain constant, which we detail below. All networks use the Adam optimizer kingma2014adam.

We run Algorithm 4 until 1 million *agent timesteps* are reached - i.e. the agent policy takes 1M steps in the randomized environments. We also cap each episode off a particular number of timesteps according to the documentation associated with brockman2016gym. In

[1]https://github.com/montrealrobotics/active-domainrand

particular, `LunarLander-v2` has an episode time limit of 1000 environment timesteps, whereas both `Pusher-3DOF-v0` and `ErgoReacher-v0` use an episode time limit of 100 timesteps.

For our agent policy, we use an implementation of DDPG (particularly, `OurDDPG.py`) from the Github repository associated with Fujimoto2018AddressingFA. The actor and critic both have two hidden layers of 400 and 300 neurons respectively, and use `ReLU` activations. Our discriminator-based rewarder is a two-layer neural network, both layers having 128 neurons. The hidden layers use `tanh` activation, and the network outputs a `sigmoid` for prediction.

The agent particles in SVPG are parameterized by a two-layer actor-critic architecture, both layers in both networks having 100 neurons. We use Advantage Actor-Critic (A2C) to calculate unbiased and low variance gradient estimates. All of the hidden layers use `tanh` activation and are orthogonally initialized, with a learning rate of 0.0003 and discount factor $\gamma = 0.99$. They operate on a $\mathbf{R}^{N_{rand}}$ continuous space, with each axis bounded between $[0, 1]$. We allow for set the max step length to be 0.05, and every 50 timesteps, we reset each particle and randomly initialize its state using a N_{rand}-dimensional uniform distribution. We use a temperature $\alpha = 10$ with an RBF-Kernel as was done in svpg. In our work we use an Radial Basis Function (RBF) kernel with median baseline as described in svpg and an A2C policy gradient estimator mnih2016asynchronous, although both the kernel and estimator could be substituted with alternative methods gangwani2018diverse. To ensure diversity of environments throughout training, we always roll out the SVPG particles using a non-deterministic sample.

For DDPG, we use a learning rate $\nu = 0.001$, target update coefficient of 0.005, discount factor $\gamma = 0.99$, and batch size of 1000. We let the policy run for 1000 steps before any updates, and clip the max action of the actor between $[-1, 1]$ as prescribed by each environment.

Our discriminator-based reward generator is a network with two, 128-neuron layers with a learning rate of .0002 and a binary cross entropy loss (i.e. is this a *randomized* or *reference* trajectory). To calculate the reward for a trajectory for any environment, we split each trajectory into its (s_t, a_t, s_{t+1}) constituents, pass each tuple through the discriminator, and average the outputs, which is then set as the reward for the trajectory. Our batch size is set to be 128, and most importantly, as done in eysenbach2018diversity, we calculate the reward for examples before using those same examples to train the discriminator.