Towards Truly Open-Ended Reinforcement Learning

Jack Parker-Holder

St Peter's College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Michaelmas 2022

For Chelsea, who continues to teach me more than any research paper could.

Acknowledgements

People say it takes a village to raise a PhD, for me it is closer to an entire city of amazing people. As a consequence, this section is rather long.

First and foremost, I will be eternally grateful to my advisor Stephen Roberts for giving me the opportunity to pursue my dream of doing a PhD at the University of Oxford. But that was just the start. Every step of the way (through some "unprecedented" times) Steve has enabled me to do my best. He has actively encouraged diverse research topics and allowed me to work with a wide range of collaborators while crucially maintaining a core vision for my research. I could not have had a supervisor more perfect for what I wanted from a PhD.

I want to thank Nick Hawes and Kenneth Stanley for graciously agreeing to examine my thesis. They not only drastically improved the thesis with their suggestions but also taught me a huge amount during the viva.

It is safe to say I would not even be in this field if it wasn't for Krzysztof Choromanski. Meeting Krzysztof was one of those moments that completely changed my life. He showed me that research was accessible with enough passion and hard work and gave me my first opportunity in the field. He also introduced me to Yunhao Tang and Aldo Pacchiano, who helped inspire me to apply for a PhD.

In Spring 2020 Aldo introduced me to an old friend of his, Jakob Foerster. This not only resulted in one of the most entertaining ventures of my PhD, but also introduced me to a fantastic mentor and friend. I want to thank Jakob for patiently teaching me all about value functions, explaining the DiCE operator several times and more recently helping make my job talks coherent!

I was incredibly fortunate to spend time with the FAIR London team, who took my research to a new level. Particular thanks to Tim Rocktäschel for giving the opportunity and for being such a great mentor and inspiration throughout my PhD. My time at FAIR was very much centered around my work with Minqi Jiang, who taught me a huge amount and remains one of my closest collaborators. I also had the pleasure of working with Edward Grefenstette, Michael Dennis, Roberta Raileanu, Yingchen Xu and Mikayel Samvelyan. I also want to thank Tim and Ed for making me feel welcome in the UCL DARK group, a truly amazing lab filled with great people.

I have also had the pleasure of working with fantastic people closer to home, in the MLRG in Oxford. In particular Phil Ball, who I met on my first day

in Oxford, which in hindsight was an incredible turn of fate. While most of our discussion has been about football, we have also managed to publish several papers together. Special thanks to Cong Lu who joined the group and helped take our research in new directions. I also had great fun working Sam Kessler, Xingchen Wan and Tim Rudner.

Thank you to Vu Nguyen for being my Bayes Opt jungle guide, without whom the PB2 papers would not have been possible. These efforts led us into the broader AutoRL community, where I particularly want to thank Xingyou Song and Aleksandra Faust for interesting discussions and advice. Thanks to Ted Moskovitz and Michael Arbel for a super fun collaboration stemming from our passion for BGRL (alongside Aldo).

I want to thank Richard Everett for helping me learn more about DeepMind, where I am incredibly excited to be starting the next chapter of my career. I am looking forward to working with a whole new set of amazing people and I am grateful once again to Tim for what is a dream opportunity. I hope this thesis can be a springboard for exciting future work.

Finally, I want to thank my family for supporting me on this adventure. To my mum, thank you for always putting me first and teaching me the value of hard work. To my sister Emma, I appreciate you more than you will ever know. Now to the Parker-Holders, who have gone from two to four since the start of my PhD. The first joiner, Doris has been an amazing companion during these years working from home, while Edie has been an absolute joy. Neither of them can read (for entirely different reasons) but thank you to both. Lastly, thank you to my partner, Chelsea, to whom this thesis is dedicated.

Abstract

Deep reinforcement learning (RL) has achieved some remarkable successes in the past decade, from super human performance in games to real world problems such as robotics and even Nuclear Fusion. Indeed, given its generality, many prominent researchers in the field believe RL alone may be sufficient for producing Artificial General Intelligence (AGI). It is easy to see why, RL is in theory an open-ended process where an agent never stops learning from its own experiences, given a suitably complex environment. In this thesis we posit that the key factor limiting RL agents is the requirement for static, human designed configurations. On the agent side, we typically tune a single set of hyperparameters for a specific agent with a specific architecture, ignoring the fact these may need to adapt over time. At the same time, even if we did have a powerful RL agent, we lack a sufficiently complex environment that could facilitate the learning of general behaviors.

We hypothesize that the only way to get past this problem is to embrace *open-endedness*, by designing systems with infinite capacity to produce new, interesting things. In the first part we introduce new methods for adapting several important agent hyperparameters on the fly, using a population of agents. We show this makes it possible for agents to adapt several of their own hyperparameters over time. Next we introduce a new approach for automatically designing environments, evolving a curriculum that constantly proposes new challenges at the frontier of the student agent's capabilities. Combining these two advances could produce an open-ended learning system where agents and environments co-adapt over time, producing increasingly complex problems and an agent that can solve them.

However, even this would not be *truly* open-ended, as it would eventually plateau once the agent can solve every task from a human-specified distribution. In the second part of the thesis we propose directions to make this system unbounded. First, we introduce two new approaches for encouraging the discovery of diverse solutions, which we show can help avoid deceptive local optima and discover a broader set of behaviors. Finally, we posit that one path towards a truly open-ended system is to remove the need for human-designed simulated environments altogether and instead train agents inside learned world models. We discuss several contributions in this area including active data acquisition to improve the world model, as well as an approach to produce synthetic experiences inside the world model that increase the robustness of our agents. We conclude with a proposal for a future system combining these insights, which we believe could be truly open-ended.

Declaration

I declare that no part of this thesis has been, or is being, submitted for any qualification other than the degree of Doctor of Philosophy at the University of Oxford. This thesis is entirely my own work and, except where otherwise stated, describes my own research.

Jack Parker-Holder, St Peter's College

Contents

Notation	Meaning
$\mathcal{S}$	State space
$\mathcal{A}$	Action space
P	Transition function
R	Reward function
$\mathcal{O}$	Observation space
ρ_o	Initial state distribution
γ	Discount factor
t	Inner loop environment timestep
H	Maximum inner loop environment timestep
n	Outer loop trial index
N	Maximum outer loop trial index
V	State-value function
Q	Action-value function
A	Advantage function
J	Expected total reward function
L	Loss function
f	Objective function
θ	Neural network parameters
ψ	World model parameters
B	Batch size
λ	Bootstrapping hyperparameter
ζ	All unspecified components
x	Set of agent hyperparameters, subset of ζ
η	Subset of agent hyperparameters that are differentiable
ξ	Set of environment hyperparameters, subset of ζ

Table 1: Table of notation used.

1 Introduction

"Give a man a fish, and you feed him for a day. Teach a man to fish, and you feed him for a lifetime."

—Proverb

1.1 Is Reward Enough for Artificial General Intelligence?

Reinforcement learning (RL, Sutton & Barto [290]) is a paradigm where agents learn entirely from their own interactions in an environment to maximize expected cumulative reward. This differs from other settings in machine learning in that agents can learn to plan and reason, in order to take actions with long term consequences. In theory there is no limit to an agent's ability to learn from its own experience, thus RL offers the potential for open-ended learning.

Over the past decade, the combination of RL with deep neural networks [87, 144, 252], has led to a series of remarkable achievements. The first major triumph for so-called *deep* RL[1] agents came when researchers showed it was possible to learn to play Atari games at a human level, solely from pixels [190]. Other successes came in quick succession, as the *AlphaGo* agent famously defeated one of the world's

[1]From now on when we refer to "RL" we mean "deep RL".

best Go players [271] while RL also proved capable of competing with humans in competitive esports games [22, 312]. It has not only been games where RL has had a significant impact, in robotics, RL has been used to transfer from simulation to the real world [7, 118, 209, 226, 298] and for training from large datasets [126, 166]. In recent times, RL has even led to impact outside of its traditional domains, for example navigating in the real world [19] or controlling plasma [53].

With such rapid progress in the field, it is possible to extrapolate further and consider what may be possible using RL in the future. RL has already shown the ability to find totally new solutions to problems studied by humans for centuries, for example the famous "move 37" by AlphaGo in the second match against Lee Sedol. Indeed, given its generality, some of the most prominent researchers in the field believe RL alone may be enough to produce Artificial General Intelligence (AGI) [274]. In this thesis we will take this claim at face value, and focus on the missing ingredients required to produce general agents with RL. Further, we seek to establish research directions that when solved could bridge this gap.

Before we begin investigating future research directions we need to (briefly) clarify what we mean when we refer to AGI. First a caveat, there are many possible definitions of intelligence [90, 157], and it is not the goal of this thesis to propose a new one. More as a guide than a specific objective, in this thesis we take inspiration from the work pioneered by Hutter [110] and Legg & Hutter [158], who originally referred to the *universal intelligence* of an agent π. Concretely, Legg & Hutter [158] propose the following definition:

$$\Upsilon(\pi) \coloneqq \sum_{\mu \in E} 2^{-K(\mu)} V_\mu^\pi \tag{1.1}$$

Or informally: *"Intelligence measures an agent's ability to achieve goals in a wide range of environments"*. Here a "wide range of environments" is represented by E, the space of all well-defined reward summable environments inversely weighted by K, the Kolmogarov complexity [139] of a binary string used to describe an environment, from an agent's perspective. The success of the agent π is represented by the value function V_μ^π, which we will define more rigorously in Section 2.1.

The benefit of viewing AGI this way is that it is no longer a binary label, but instead takes a continuous range of values. Thus, we can make progress towards AGI, however distant it may be, by looking to increase the value of Equation 1.1. So what might this look like for current RL methods? It is clear that while we have achieved great successes, many of these have been a more narrow form of AI, for example solving individual games. Further, many agents fail to generalize to even minor variations in their environments [45, 136, 214], often overfitting to specific components of the training environment [276, 339] or other agents acting in the environment [106]. Thus, to increase the value of Equation 1.1, we must focus on increasing the diversity of tasks that agents can solve, make agents robust to variations in the environment and also to other agents.

The next question is how can we achieve this with RL? If we look more closely at Silver *et al.* [274], the authors conjecture that *"powerful reinforcement learning agents, when placed in complex environments, will in practice give rise to sophisticated expressions of intelligence. If this conjecture is correct, it offers a complete pathway towards the implementation of artificial general intelligence."* Thus we can clearly see the issue here—we do not yet have "powerful reinforcement learning agents" or "complex environments", or definitions of either.

In this thesis we posit that the key limiting factor in achieving more general agents with RL is the reliance on specific hand-coded agent and environment configurations, which lack the ability to increase their complexity over time. If we train our agents on simple problems, then we can quickly achieve mastery but ultimately achieve a low value in Equation 1.1. However, placing an agent in a highly complex environment makes learning challenging [100]. Instead, we believe a possible path towards generally capable agents is a system that begins with simple configurations but makes it possible for them to automatically become more complex over time. To achieve this, we propose methods to automatically discover increasingly challenging problems for our agents to solve, while also introducing a way for our agents to adapt to solve them. This is the problem of open-endedness.

1.2 The Grand Challenge of Open-Endedness

The problem we seek to address in this thesis is making our RL training systems more *open-ended*, which we hypothesize is the key missing ingredient to produce more generally capable agents. Indeed, open-endedness could arguably be seen as the most important grand challenge remaining in AI [281]. But first, what constitutes open-endedness and how can we measure progress towards this goal?

Defining open-endedness remains a challenge and an open research question in itself. There have been many efforts to do so, for example Soros & Stanley [277] set out four conditions for evolutionary systems to be open-ended. For brevity, in this thesis we simply refer to an open-ended learning system as one that *continues to learn new, interesting things up to the limit of modern computation*. Note that while this may appear vague, many modern systems clearly fail to meet this criteria. Indeed, this can be seen if we consider the dominant paradigm in current RL research, an agent learning to play a single game. Once the agent has achieved mastery, there is nothing more to learn. Thus, under most intuitive definitions of "interesting" there would be a limit to the capability of the agent with hundreds or thousands of GPU hours (i.e. the limit of modern computation).

In general, the approach we take to design open-ended RL systems is to allow the system to learn more [42], a recipe which has a proven track record of success. A canonical example of this is the AlphaGo agent, that famously beat one of the world's best Go players after training with RL alongside human data [271]. Researchers found that the agent actually got stronger when the human data was removed, as the new *AlphaZero* agent was able to learn its own, novel strategies with self-play [272, 273]. AlphaZero was subsequently beaten by *MuZero*, which not only learned its own strategies but also the rules of the game itself [264].

However, even MuZero will reach a plateau once it has mastered every possible strategy discoverable in Go. Thus, in this thesis we aim to build systems that don't just solve individual problems, but also discover their own problems altogether (see: Chapter 4). Further, we introduce methods that allow agents to adapt their configurations on the fly, making it possible for them to discover new, effective

strategies to solving the stream of problems encountered (see: Chapter 3). This forms Part I, which we generally refer to *Automated Reinforcement Learning*, since multiple components of the RL pipeline are learned by the algorithm during training. In recent times, methods similar to those proposed in Part I have been combined to form a powerful, more open-ended learning system that produces agents with general capability across a broad distribution of environments [296]. This offers promise that these methods may scale, but alone it may still produce a system that eventually stops learning new things once the agent can solve every world in the procedural task distribution.

Indeed, much like AGI, open-endedness is not a binary label we can place on a system. In a recent paper, Lehman *et al.* [159] point out that these state-of-the-art systems are only a "weak" form of open-endedness—they are capable of producing new, interesting things for a brief period of time but will still eventually plateau. By contrast, a "strong" open-ended system should be capable of producing interesting things even a million years later. There are many hypotheses for how to produce such a system, with decades of research in Artificial Life focusing on designing simulated worlds that can produce it [89, 213, 295], while Lehman *et al.* [159] propose that "conditional invention" may be required, i.e. the ability for agents to place their innovations back into the environment.

In Part II we present an alternative route to the strong form of open-endedness, in what we refer to as *truly open-ended* RL. Concretely, we propose to learn *world models* directly from data (see: Chapter 6). The methods we present operate from RL agent trajectories, but demonstrate a path to learning from human-generated data—bootstrapping an existing open-ended system that has been operating for millennia, human cultural evolution. To drive innovation in this setting, we also present two distinct methods for generating diverse solutions (see: Chapter 5), which we believe will be critical to produce unbounded innovation in an open-ended system.

1.3 Overall Structure

The rest of this thesis is structured as follows:

Preliminaries

We begin by reviewing the necessary background material on deep reinforcement learning and methods for automating underspecified components, referred to as Automated Reinforcement Learning or *AutoRL*. We include many details from the following survey:

- **Jack Parker-Holder***, Raghu Rajan*, Xingyou Song*, André Biedenkapp, Yingjie Miao, Theresa Eimer, Baohe Zhang, Vu Nguyen, Roberto Calandra, Aleksandra Faust, Frank Hutter, Marius Lindauer *Automated Reinforcement Learning (AutoRL): A Survey and Open Problems* JAIR, 2022

Contributions: Jack co-led this effort, contributing to the original concept and writing sections of the paper.

1.3.1 Part I: Automated Reinforcement Learning

In this part we make the following contributions:

- We introduce *Population Based Bandits* (PB2), an algorithm for learning and adapting agent configurations on the fly during a single training run.
- We propose a new method for evolving adaptive curricula, capable of producing robust generalist agents within a given task distribution.

We hypothesize that sizable improvements towards making RL systems more open ended is via automatically discovering and adapting large components of the system's hyperparameters. We break this problem down into two pillars: *agents tune themselves* and *environments continuously propose new challenges*. In Chapter 3 we first focus on developing agents that can learn and adapt a large component of their configuration on the fly in a single training run. This, we believe, can lead to *powerful* RL agents that never stop learning. Next, in Chapter 4 we discuss

approaches for evolving an adaptive distribution of training environments. As we show, this makes it possible to automatically discover a curriculum that leads to the discovery of *complex* environments, while training an agent that can solve them. We end Part I by discussing the potential for combining these two components into a coevolutionary [26] system where environments get increasingly challenging and agents adapt to solve them. See below for a more specific list of the contributions.

Agents that Tune Themselves We introduce approaches for automatically discovering several components of an RL agent configuration in a single training run. We focus on two publications:

- **Jack Parker-Holder**, Vu Nguyen, Stephen Roberts *Provably Efficient Online Hyperparameter Optimization with Population-Based Bandits* NeurIPS, 2020
- **Jack Parker-Holder**, Vu Nguyen, Shaan Desai, Stephen Roberts *Tuning Mixed Input Hyperparameters on the Fly for Efficient Population Based AutoRL* NeurIPS, 2021

Contributions: Jack conceived the ideas, ran experiments and wrote the papers. All theoretical results were provided by Vu.

Environments that Continuously Propose New Challenges We discuss how we can automatically design environments to learn a curriculum for a deep RL agent. The primary focus is the following publication:

- **Jack Parker-Holder***, Minqi Jiang*, Michael Dennis, Mikayel Samvelyan, Jakob Foerster, Edward Grefenstette, Tim Rocktäschel *Evolving Curricula with Regret-Based Environment Design* ICML, 2022

Which is related to and builds upon a previous publication:

- Minqi Jiang*, Michael Dennis*, **Jack Parker-Holder**, Jakob Foerster, Edward Grefenstette, Tim Rocktäschel *Replay-Guided Adversarial Environment Design* NeurIPS, 2021

Contributions: Jack contributed to the ideas, experiments and paper writing.

1.3.2 Part II: Achieving Open-Endedness

This part includes the following contributions:

1. We introduce two new algorithms for discovering diverse solutions, via behavioral (DvD) and genotypic (Ridge Rider) diversity.

2. We propose novel information theoretic approaches for collecting data to produce effective world models, inspired by active learning.

Combining methods from Part I should produce a powerful coevolutionary system, where the environment constantly proposes difficult challenges for an agent that can adapt to solve them. However, as we point out, this alone may not be enough for open-endedness—it will likely converge to a single solution to a single distribution of environments. Further, this solution may not even be optimal, since in many cases direct optimization does not find the most powerful behaviors [268, 325]. The second part of the thesis is more speculative in nature, again broken into two chapters, each of which proposes a possible direction to make our RL training systems unbounded. First in Chapter 5 we introduce methods to encourage diversity in the learning dynamics, by inducing the discovery of a greater breadth of creative and maybe even unexpected behaviors. The final hurdle we seek to overcome in Chapter 6 is the reliance on human-designed simulated environments, where instead we propose to train agents inside world models, learned directly from data. This essentially provides sufficient foundations for an "AI generating algorithm" [42], which can fully learn its own environments, while learning RL algorithms and configurations to solve them.

The Need for Diversity Emphasizes the importance of diversity in open-ended reinforcement learning systems by introducing two distinct approaches for producing diverse solutions, using behavioral and parameter-based or *genotypic* diversity:

- **Jack Parker-Holder***, Aldo Pacchiano*, Krzysztof Choromanski, Stephen Roberts *Effective Diversity in Population-Based Reinforcement Learning* NeurIPS (Spotlight), 2020

1. Introduction

- **Jack Parker-Holder***, Luke Metz, Cinjon Resnick, Hengyuan Hu, Adam Lerer, Alistair HP Letcher, Alex Peysakhovich, Aldo Pacchiano, Jakob Foerster* *Ridge Rider: Finding diverse solutions by following eigenvectors of the Hessian* NeurIPS, 2020

Contributions: Jack contributed to the ideas, experiments and paper writing. Aldo led the effort on the theoretical side. Jakob conceived the "ridge rider" concept.

Learning World Models We discuss the possibility of learning simulators, by training *world models*. The contributions in this area revolve around two components, optimally selecting data to improve the model:

- Philip Ball*, **Jack Parker-Holder***, Aldo Pacchiano, Krzysztof Choromanski, Stephen Roberts *Ready Policy One: World Building Through Active Learning* ICML, 2020

- Yingchen Xu*, **Jack Parker-Holder***, Aldo Pacchiano*, Philip Ball*, Oleh Rybkin, Stephen Roberts, Tim Rocktäschel, Edward Grefenstette *Learning General World Models in a Handful of Reward-Free Deployments* NeurIPS, 2022

And augmenting the model to generate synthetic experiences for improved agent generalization:

- Philip Ball*, Cong Lu*, **Jack Parker-Holder**, Stephen Roberts *Augmented World Models Facilitate Zero-Shot Dynamics Generalization From a Single Offline Environment* ICML, 2021

Contributions: Jack contributed to the concepts and paper writing in all three efforts. Jack also co-led experiments in the cases with equal first authorship.

Conclusion We conclude the thesis by proposing to combine many of the presented contributions to produce a *truly* open-ended reinforcement learning system.

2 Preliminaries

Contents

This chapter is a self-contained primer on the main topics covered in the thesis. We begin by defining the RL problem, before introducing the predominant paradigms in the field.[1] Next we move to the AutoRL problem, which seeks to automate key components of the RL training pipeline. This latter discussion comes from a recent survey on the topic [219].

[1] Much of this introduction comes from existing resources (e.g. OpenAI [208]).

2.1 Reinforcement Learning

Reinforcement learning (RL, Sutton & Barto [290]) considers the problem of an agent interacting in an environment, learning from its own experience to maximize cumulative extrinsic reward.

2.1.1 Markov Decision Processes

At each timestep t in an *episode*, an agent receives a state $s_t \in \mathcal{S}$ and takes an action $a_t \in \mathcal{A}$. Decisions are typically taken by a stochastic *policy* π, thus the action is sampled $a_t \sim \pi(s_t)$. The environment is comprised of transition and reward functions, denoted by $P : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ and $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ respectively. P takes both the state and action and returns a distribution over next states, $s_{t+1} \sim P(s_t, a_t)$, while R takes a state, action, next state triple and returns a scalar reward $r_t = R(s_t, a_t, s_{t+1})$. Finally, initial states are sampled from a start-state distribution $s_0 \sim \rho_0(.)$.

The combination of a state and action space, reward and transition function and start-state distribution defines a Markov Decision Process (MDP, Puterman [236]) which we denote $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \rho_0)$, typically referred to as an *environment*. A policy π, acting in the environment, produces a trajectory $\tau = \{s_1, a_1, \ldots, s_H, a_H\}$ for an episode with horizon H. Given that the per-timestep reward r_t is a function of each (s_t, a_t, s_{t+1}), we can then define the cumulative reward of a trajectory as the sum of the per transition rewards, i.e.

$$R(\tau) = \sum_{t=0}^{H} r_t \tag{2.1}$$

We can also consider infinite-horizon episodes. In this case we consider discounted returns, making rewards earned in the short term worth more than those in the future. We do this using a *discount factor* $\gamma \in (0, 1)$, producing the following cumulative return:

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t \tag{2.2}$$

Since actions in the trajectory are sampled from a policy, we can then define the reinforcement learning problem as finding a policy $\pi^\star$ that maximizes expected returns in the environment, i.e.

$$\pi^\star = \arg\max_\pi \mathbb{E}_{\tau\sim\pi}[R(\tau)] \tag{2.3}$$

2.1.2 Partial Observability

An MDP observes the Markov property, which means that the state $s_t \in \mathcal{S}$ contains all necessary information to select the optimal action $a_t \in \mathcal{A}$ in the environment. However, this is not always achievable in practice, either because information is not available to the agent or because processing every piece of relevant information may be intractable.

To address this setting, we further define a partially-observable MDP (POMDP) with two additional components: $\mathcal{O}$ represents the set of observations and $\Omega : \mathcal{S} \times \mathcal{A} \times \mathcal{O} \to \mathbb{R}^+$ describes the probability density function of an observation given a state and action.

In practice most environments we deal with will be POMDPs, where it may be beneficial to accumulate knowledge from previous observations to inform a belief of the current state. This increases the challenge for learning an optimal policy $\pi^\star$, since we typically need to consider policies that can infer the true state s_t from the observation o_t, as well as then choosing the correct action to take.

2.1.3 Value Functions

Given a policy π we can define the return from a state s as follows:

$$V^\pi(s) = \mathbb{E}_{\tau\sim\pi}[R(\tau)|s_0 = s] \tag{2.4}$$

The value function is a central component of many RL algorithms. In some cases, we also wish to quantify the return from a state s, given an action a:

$$Q^\pi(s,a) = \mathbb{E}_{\tau\sim\pi}[R(\tau)|s_0 = s, a_0 = a] \tag{2.5}$$

Thus we have ways to define both the value of a state and the value of an action given a state. These two quantities are linked via the *Bellman equation*:

$$Q^{\pi}(s,a) = \mathbb{E}_{s' \sim P(s,a)}[R(s,a,s') + \gamma V(s') | s_0 = s, a_0 = a] \tag{2.6}$$

In other words, the value of an action is equal to the single step return plus the expected future value of the next state. Now we have a notion of the value of a state and action, we may return to the optimal policy $\pi^{\star}$, defining the optimal value function as:

$$V^{\star} = \max_{\pi} \mathbb{E}_{\tau \sim \pi}[R(\tau) | s_0 = s] \tag{2.7}$$

Finally, we define an *advantage* function as the improvement over the expected return from taking an action a:

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s) \tag{2.8}$$

Equipped with these definitions we will move to two classes of *model free* RL algorithms, either directly optimizing a policy (policy gradient algorithms) or seeking to learn a value function (off policy methods).

2.1.4 Policy Gradient Algorithms

Policy gradient algorithms seek to directly learn a policy by maximizing the probability of taking actions that led to good outcomes. Given a policy parameterized by θ, our objective is to maximize the expected total reward of a trajectory, namely:

$$J(\pi_{\theta}) = \mathbb{E}_{\tau \sim \pi_{\theta}} R(\tau) \tag{2.9}$$

Since, in deep RL, we typically use a neural network policy, we thus optimize θ by taking the following gradient steps:

$$\theta_{t+1} = \theta_t + \alpha \nabla_{\theta} J(\pi_{\theta_t}) \tag{2.10}$$

in which $\alpha \in [0, 1]$ is a learning rate hyperparameter. Using the fact that the MDP may be factorized as a product of transitions and expected actions given states, it is possible to derive a gradient for Equation 2.9 as follows [228, 291, 324]:

$$\nabla_\theta J(\pi_{\theta_t}) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T} \nabla_\theta \log\pi(a_t|s_t) R(\tau)\right] \tag{2.11}$$

However, note that this assigns credit rather clumsily. Each action is given the reward of the entire trajectory it took place in. In fact, it is not important to consider the rewards before an action is taken, so many use the more useful *reward-to-go* formulation:

$$\nabla_\theta J(\pi_{\theta_t}) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T} \nabla_\theta \log\pi(a_t|s_t) \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1})\right] \tag{2.12}$$

Going further, it has been shown [291] we can subtract a *baseline* from the reward-to-go, further reducing variance. i.e.

$$\nabla_\theta J(\pi_{\theta_t}) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T} \nabla_\theta \log\pi(a_t|s_t) \left(\sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}) - b(s_t)\right)\right] \tag{2.13}$$

which is typically the value function of the state: $V^\pi(s_t)$. In this thesis we will consider estimating the value function with a neural network, for example by minimizing mean squared error between the model prediction and the true reward-to-go.

Thus, the objective in Equation 2.13, seeks to increase the probability of taking actions which lead to improved performance when compared to the predicted returns. Interestingly, we can also use the Q-function, and have $b(s) = Q^\pi(s, a)$. These methods are generally known as *actor-critic* algorithms because the critic comes in the form of the value function which in a sense scores the performance (by subtracting its expected reward from that state) [290]. Indeed, we can use the learned Q-values to produce an *advantage* A, defined as $A(s, a) = Q(s, a) - V(s)$, in other words, the increase in value for taking an action in a given state.

Even with reduced variance from a baseline, policy gradient methods remain challenging to work with, since a small update can lead to a change that catastrophically harms learning. A significant breakthrough came with TRPO [265], a method that showed monotonic improvement guarantees by optimizing a policy with a

constraint to remain close to the previous (or default) policy in the behavior space. This method was simplified with a first order approximation in PPO [267]. Given its prominence in the literature, and relative simplicity, many works in this thesis build upon PPO. In particular, we make use of the clipped PPO objective, as follows:

$$\mathcal{L}_{\text{PPO}}(\theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_\mu(a|s)}A^{\pi_\mu}, g(\theta,\mu)A^{\pi_\mu}\right), \tag{2.14}$$

$$\text{where } g(\theta,\mu) = \text{clip}\left(\frac{\pi_\theta(a|s)}{\pi_\mu(a|s)}, 1-\epsilon, 1+\epsilon\right) \tag{2.15}$$

for a previous or behavior policy π_μ, an advantage function A and a clipping hyperparameter ϵ. Essentially, PPO seeks to maximize the advantage such that the policy does not deviate too far from the behavior policy (measured as a likelihood ratio). The clip parameter specifies the size of a permissable deviation. Despite this relative simplicity, PPO has achieved tremendous success, for example by scaling to play esports games at a competitive level [22].

2.1.5 Off Policy RL

Since policy gradient algorithms learn from data collected from the current policy, they are referred to as being *on policy*. We can also learn from *off policy* data, for example with the Q-learning algorithm [318]. In its most basic form, Q-learning maintains an *estimate* $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ of the optimal value function, and, given a sequence of transition tuples (s, a, r, s'), updates $Q(s, a)$ towards the target $r + \gamma \max_{a' \in \mathcal{A}} Q(s', a')$, where $\gamma \in (0, 1)$ is the discount factor.

One of the first breakthroughs in deep RL came in combining Q-learning with deep neural networks, to produce the Deep Q-Learning algorithm, known as DQN [190]. The cornerstone of DQN is a deep neural network learned to approximate the Q-values by optimizing the following objective:

$$L(\theta) = \mathbb{E}_{s,a}[(y - Q(s, a; \theta))^2] \tag{2.16}$$

Where the target $y = \mathbb{E}_{s' \sim P}[r_t + \gamma \max_{a'} Q(s', a')]$, θ corresponds to the neural network weights and γ is the discount factor. DQN was the first RL agent to

achieve strong performance across the full suite of Atari games [20]. Since then, there has been an explosion of interest in the field, with a swathe of improvements to the original DQN algorithm proposed [103, 260, 307].

Given that off policy methods like DQN are typically an order of magnitude more sample efficient than the policy gradient algorithms discussed previously, there has been great interest in producing *off policy* actor critic algorithms. The canonical method is DDPG [172], which, like DQN, tries to learn an optimal Q function $Q^*(s, a)$, such that it can follow it to take actions, via $a = \max_a Q^*(s, a)$. However, while DQN uses this greedy policy to act in the environment, DDPG instead seeks to *learn a separate policy network* $\mu(s)$, and shows that $\max_a Q^*(s, a) \approx Q(s, \mu(s))$, i.e. the action taken by the policy can be the optimal action. Learning this policy is simple, a neural network is trained to output actions that maximize the expected Q values given a state. This framework has been used to produce state-of-the art off policy actor critic algorithms [80, 95, 141, 331].

2.1.6 Model-Based RL

Thus far, the methods described are designed to learn online from experience generated in an environment, which is typically an MDP, i.e. $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \rho_0)$. Instead, *model-based* methods seek to learn a model of the environment, by approximating both the transition and reward functions, denoted by P and R respectively. This model, referred to henceforth as a *world model*, is typically a deep neural network. In this thesis we will use ψ to represent the model parameters, hence we denote the approximate dynamics and reward functions as P_ψ and R_ψ. Thus, the world model induces a new "imaginary" MDP, $\mathcal{M}_\psi = (\mathcal{S}, \mathcal{A}, P_\psi, R_\psi, \rho_0)$.

We focus on Dyna-style model-based RL [289], whereby we train a policy (π_θ) using "imagined" transitions, solely inside $\mathcal{M}_\psi$. We can use any model free algorithm to optimize the policy, and further, we can train the policy on a single hardware accelerator with parallelization since the simulator is now a neural network world model [148]. When training from low dimensional states (such as lidar), the

common paradigm is to use an ensemble of probabilistic dynamics models, originally introduced in [204] and shown to be effective for learning world models [40, 119].

World models have also shown to be effective from high dimensional observations such as pixels, typically making use of variational encoders [135]. The first work to show this was Ha & Schmidhuber [94], with several methods building on this architecture in a series of works introducing PlaNet [98], Dreamer [97] and DreamerV2 [99]. These works rely on a model with two components: an *Encoder* which maps sensory inputs to latent codes and a *Recurrent State Space Model* or RSSM, which predicts future latent codes given actions and the previous latent.

2.1.7 Evaluation

It is non trivial to evaluate the performance of deep RL agents and compare them across tasks and benchmarks [102]. Indeed, when combining scores from multiple environments it is common to use a mean or median score, both of which contain issues. The mean can be overly influenced by a handful of large (or small) scores, while the median is robust to these outliers but will ignore the majority of the data, for instance an agent could get a score of zero in 49% of environments but the median would not capture this. In this thesis we will largely adopt recent practices recommended by Agarwal *et al.* [2] in the `rliable` library. In particular we will make use of the Inter-Quartile Mean (IQM), which corresponds to the mean performance after removing the outliers. This is essentially a compromise between the mean and median, it ignores outliers but then averages over the middle quartiles rather than taking a single point estimate at the 50th percentile.

2.2 Automated Reinforcement Learning

While deep RL has achieved many successes, one of its biggest drawbacks is the large number of underspecified parameters, requiring significant RL expertise to use it for each new problem. In this section we introduce the emerging field of *Automated Reinforcement Learning* (AutoRL, Parker-Holder *et al.* [219]), the topic of the first major contributions in the thesis (see: Part I).

To formalize the AutoRL problem, recall the RL objective of Equation 2.9, which is to maximize the expected return of a trajectory sampled by a policy, i.e. $\mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)]$. We now define an additional set parameters ζ relating to unspecified components of the RL training objective, such as unknown agent hyperparameters (including architecture and algorithm) and environment configuration. This produces the following optimization objective:

$$\max_{\theta} J(\theta;\zeta) \text{ where } J(\theta;\zeta) = \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)] \tag{2.17}$$

Note that this is a maximization over θ, i.e. given a fixed training configuration ζ, the best agent parameters are $\theta^*_\zeta \in \arg\max_\theta J(\theta;\zeta)$. To optimize ζ, we have a separate *evaluation objective* $F(\zeta)$ that refers to the performance of θ^*_ζ in a given distribution of MDPs [321]. In the setting where ζ only represents agent configurations (such as architecture or learning rate), then the evaluation will simply be $F(\zeta) = J(\theta^*_\zeta;\zeta)$. Thus, AutoRL can be defined as a maximization over ζ, as follows:

$$\max_{\zeta} F(\zeta) \tag{2.18}$$

This is a bilevel optimization problem, where an outer loop seeks to find the optimal setting for ζ, but each evaluation of $F(\zeta)$ requires training an agent with that given configuration using an inner loop, to maximize $J(\theta;\zeta)$.

2.2.1 What Needs to be Automated?

Recall that the key objective in this thesis is to make RL systems more open-ended. To do this we propose methods to automatically discover and crucially also adapt both agent and environment configurations, to produce a system which can increase complexity over time. In order to prepare for the later chapters we will briefly discuss both of these areas here before expanding significantly on them in Part I.

When seeking to discover powerful RL algorithms, the first question is: what algorithm do we use? As we have seen, this can be model based or model free, and if the latter it could use an on policy or off policy approach. Even if we have a fixed algorithm such as PPO, we need to decide the policy architecture. In practice this

is typically a small feed forward neural network for MDPs with a low dimensional state space, a convolutional neural network for pixel-based environments [64], or a recurrent network for POMDPs.

Further, there are a multitude of other choices, such as he type of exploration strategy [12, 13, 75, 231, 261], data augmentation type [154, 240] or even the choice of optimizer [34]. One level below these choices, the success of many RL algorithms hinges on finding optimal hyperparameters. Returning to on policy actor critic algorithms such as PPO, recent work has shown that hyperparameters can play a large role [6]. Concretely, PPO requires hyperparameters such as the learning rate α, the clip parameter ϵ, and decay γ, along with a hyperparameter in the advantage function, λ [266]. As if this wasn't enough, a recent study showed we also need to consider *code level* hyperparameters [63] after recent work found nine implementation details (such as reward scaling) had a significant impact on performance. Thus, it is clear that although PPO can be effective, it requires careful tuning if it is to be used across multiple problems.

2.2.2 Bayesian Optimization

There are many different approaches for maximizing $F(\zeta)$. In this work, we primarily focus on *blackbox optimization* based approaches, which do not consider the structure of the inner loop and thus cannot exploit it. The most common approaches to this problem in the RL literature are grid search and, less frequently, random search [21]. These methods sample values of ζ from the full distribution and train agents end-to-end using the sampled configurations. The blackbox function $F(\zeta)$ simply returns the final performance.

By contrast, Bayesian Optimization (BO, Brochu *et al.* [27] and Shahriari *et al.* [270]) is a sequential model-based blackbox optimization method [108]. Since it is sequential, after each trial (or batch of trials) the model is updated and a new set of points are selected based on all the knowledge acquired thus far. As a result, BO has been shown to be dramatically more sample efficient than competing approaches. This is crucial when evaluating the blackbox function is expensive, which is often

the case in RL. A canonical example of this is with the AlphaGo agent, where BO was shown to significantly increase the win rate [37].

BO is a global optimizer, thus it can propose to evaluate any point ζ in the domain, which is typically considered to be a compact, convex subset $\mathcal{D} \in \mathbb{R}^d$ where d is the number of hyperparameters. At each step t of the optimization process, BO will propose a new ζ_t and receive as feedback a noisy label $y_t = F_t(\zeta_t) + \epsilon_t$ where $\epsilon \sim \mathcal{N}(0, \sigma^2)$, for a Gaussian with mean zero and variance σ^2. Thus, the samples are unbiased and in expectation, $y = F(\zeta)$. In our setting, the function F corresponds to training an RL agent with the parameters ζ, which will inevitably contain noise for example through the random seed.

The core component of BO is a *surrogate model* of the blackbox function, which is typically a Gaussian Process (GP, Rasmussen & Williams [245]). A GP is a nonparametric model, fully specified by a mean function $\mu_t : \mathcal{D} \to \mathbb{R}$ and a kernel (covariance function) $k : \mathcal{D} \times \mathcal{D} \to \mathbb{R}$. The choice of kernel plays a crucial role in the GP, with a variety of options proposed [59]. In this work we focus on one of the most well-known, the squared exponential kernel, given as:

$$k^{SE}(\zeta, \zeta') = \exp\left(\frac{-\|\zeta - \zeta'\|^2}{2l}\right) \tag{2.19}$$

for a lengthscale hyperparameter l. After we have observed T data points $\{(\zeta_t, F(\zeta_t))\}_{t=1}^T$, the GP posterior belief at new point $\zeta'_t \in \mathcal{D}$, $F_t(\zeta'_t)$ follows a Gaussian distribution with mean $\mu_t(\zeta')$ and variance $\sigma_t^2(\zeta')$ as:

$$\mu_t(\zeta') \coloneqq \mathbf{k}_t(\zeta')^T(\mathbf{K}_t + \sigma^2\mathbf{I})^{-1}\mathbf{y}_t \tag{2.20}$$

$$\sigma_t^2(\zeta') \coloneqq k(\zeta', \zeta') - \mathbf{k}_t(\zeta')^T(\mathbf{K}_t + \sigma^2\mathbf{I})^{-1}\mathbf{k}_t(\zeta'), \tag{2.21}$$

where $\mathbf{K}_t \coloneqq \{k(\zeta_i, \zeta_j)\}_{i,j=1}^t$ and $\mathbf{k}_t \coloneqq \{k(\zeta_i, \zeta'_t)\}_{i=1}^t$. Thus, the GP provides us with an approximate mean prediction for a new point ζ', as well as a corresponding uncertainty estimate. To select a new point trading off these two quantities, BO uses an *acquisition function*. In this work, we use the Upper Confidence Bound

(UCB), first proposed in this context by Srinivas *et al.* [278]. UCB combines the posterior mean and variance as follows:

$$\text{UCB}(\zeta') = \mu_t(\zeta') + \sqrt{\beta_t}\sigma_t(\zeta'), \tag{2.22}$$

where $\beta \in \mathbb{R}$ is a scalar weighting the exploration-exploitation trade off in the search space. Equipped with this acquisition function we have a principled way to sample new points, balancing exploration (high uncertainty) and exploitation (high predicted mean). When the model subsequently updates with $F(\zeta')$, the posterior uncertainty will reduce at ζ'.

2.2.3 Population-Based Training

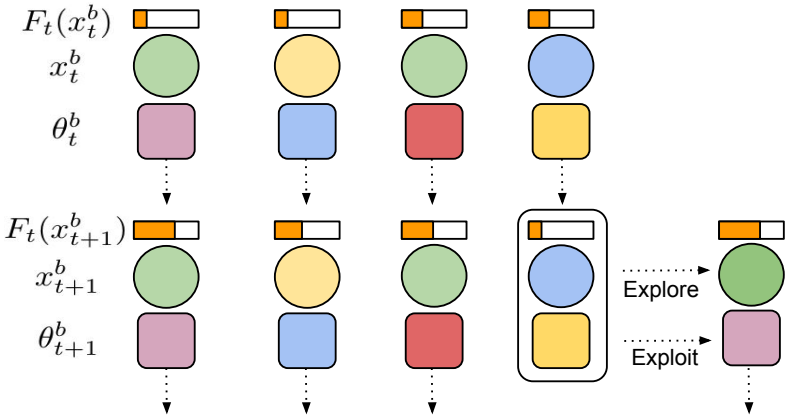


Figure 2.1: An overview of the PBT framework. We have a population of B agents, each with weights θ_t^b and hyperparameters x_t^b, where $b \in B$ is the agent index and t refers to the number of training increments. The performance of the agent is represented by F_t. At each t, the weakest agents are terminated, and replaced with copies of the strongest agents, in what is known as the exploit step. The hyperparameters of these new copied agents are then updated in an explore procedure.

Recall that with BO we typically select a ζ and fully evaluate it before re-selecting a new point. By contrast, some methods adapt ζ on the fly *during* training (the inner loop $J(\theta;\zeta)$). In particular, the Population-Based Training (PBT, Jaderberg *et al.* [116]) algorithm, which trains a population of B agents in parallel.

If we define $x \subset \zeta$ to be the hyperparameters of an RL agent, then each agent $b \in B$ has both hyperparameters x_t^b and weights θ_t^b, where t now refers to the number of training steps elapsed in a *single training run*. At every t_{ready} interval (i.e. if t is a multiple of t_{ready}), the agents are ranked based on their performance and each of the lowest ranked agents is replaced with an agent uniformly sampled from the top of agents. The fraction defining the top/bottom groups is a hyperparameter, in this thesis we keep it fixed at 25% for simplicity. This is referred to as the "exploit" step. We not have duplicate agents in the population, so to ensure adequate coverage of the hyperparameter space, PBT subsequently conducts an "explore" step, by randomly perturbing the hyperparameters of the newly copied agents. This is typically done by multiplying or dividing continuous values by a fixed value, or randomly assigning categorical variables to a new category. Once this process is complete, the agents continue training.

This leads to the learning of hyperparameter *schedules*—a single agent can have different configurations at different time-steps during a single training process. This is important in the context of training deep RL agents as dynamic hyperparameter schedules have been shown to be effective [64, 116, 220]. On the other hand, most of the existing hyperparameter optimization approaches aim to find a fixed set of hyperparameters over the course of optimization.

2.2.4 Unsupervised Environment Design

The traditional MDP framework as introduced in Section 2.1.1 assumes a single reward and transition function, which are fixed throughout training. In many real-world problems, agents may experience variations not seen during training. Thus it is crucial agents are capable of robust transfer to these novel, unseen variations.

To address this issue, we use the recently introduced *Underspecified* POMDP, or UPOMDP [54], given by UPOMDP $= (\mathcal{O}, \Omega, \mathcal{S}, \mathcal{A}, \Xi, P, R, \rho_0)$. We may consider the UPOMDP identical to a POMDP with the addition of $\Xi \subset \zeta$ representing the free parameters of the environment, similar to the *context* in a Contextual or Hidden Parameter MDP [133, 191]. These parameters can be distinct at every time

step of an episode and incorporated into the transition function $P : \mathcal{S} \times \mathcal{A} \times \Xi \to \mathcal{S}$, for example if the dynamics vary in different regions of the state space. Following Jiang *et al.* [120] we define a *level* $\mathcal{M}_\xi$ as an environment resulting from a fixed $\xi \in \Xi$. We define the value of a policy π in $\mathcal{M}_\xi$ to be $V^\xi(\pi) = [\sum_{i=0}^{T} r_t \gamma^t]$ where r_t are the rewards achieved by π in $\mathcal{M}_\xi$. UPOMDPs benefit from their generality, since Ξ can represent possible transition dynamics (for example in sim2real [7, 209, 226]), changes in observations, different reward functions, or world topology in procedurally generated environments.

Unsupervised Environment Design (UED, Dennis *et al.* [54]) seeks to produce a series of levels that form a curriculum for a *student* agent, such that the student agent is capable of systematic generalization across all possible levels in the space defined by Ξ. UED typically views levels as produced by a generator (or *teacher*) maximizing some utility function, $U_t(\pi, \xi)$. For example, uniform sampling (or *domain randomization* [298]) corresponds to a teacher with a constant utility function, for any constant C:

$$U_t^U(\pi, \xi) = C. \tag{2.23}$$

Recent UED methods use teachers that maximize *regret*, defined as the difference between the expected return of the current policy and the optimal policy. The teacher's utility is then defined as

$$U_t^R(\pi, \xi) = \operatorname*{argmax}_{\pi^* \in \Pi} \{\textsc{Regret}^\xi(\pi, \pi^*)\} \tag{2.24}$$

$$= \operatorname*{argmax}_{\pi^* \in \Pi} \{V^\xi(\pi^*) - V^\xi(\pi)\}. \tag{2.25}$$

Regret-based objectives are desirable, as they have been shown to promote the simplest possible levels that the student cannot currently solve [54]. More formally, if $S_t = \Pi$ (the set of policies) is the strategy set of the student and $S_t = \Xi$ (the set of environment parameters) is the strategy set of the teacher, then if the learning process reaches a Nash equilibrium, the resulting student policy π provably converges to a minimax regret policy, defined as:

$$\pi = \operatorname*{argmin}_{\pi \in \Pi} \{ \max_{\xi, \pi^* \in \Xi, \Pi} \{\textsc{Regret}^\xi(\pi, \pi^*)\}\}. \tag{2.26}$$

However, without access to π^*, UED algorithms must approximate the regret. One approach introduced by Dennis *et al.* [54] in the PAIRED algorithm is to estimate regret as the difference in return attained by the main student (or protagonist) agent and a second (antagonist) agent. By maximizing this difference, the teacher maximizes an approximation of the student's regret. It is important to note that even with an accurate regret estimate multi-agent learning systems may not always converge in practice [186], so we may never reach the Nash equilibrium.

2.2.5 Summary

In this section we introduced the automated reinforcement learning (AutoRL) problem, which seeks to learn unspecified components of an RL system, parameterized by ζ. As we have discussed, ζ can correspond to agent or environment components, while it could also represent both. We described contrasting approaches for learning ζ, either using Bayesian Optimization or Population Based Training. In the first part of this thesis we will delve further into the frontier of AutoRL, first discussing new hybrid approaches for tuning agent configurations on the fly, before finishing with a new approach for unsupervised environment design.

Part I

Automated Reinforcement Learning

3

Agents that Tune Themselves

Contents

3.1 Introduction

In this chapter we focus on approaches to adapt reinforcement learning (RL) agent hyperparameters, for example the learning rate, exploration parameters or neural network architecture, which we denote by $x \subset \zeta$. It has been repeatedly shown

that the success of RL agents relies on careful tuning of these parameters, often making the difference between success and failure [6, 34, 63, 102, 114, 201]. This problem is only becoming more prevalent—as new and more complex algorithms are introduced, the number of components to tune continues to grow [6]. As a result, it is often challenging to reproduce RL results given the significant number of degrees of freedom in modern algorithms [6].

Crucially for this thesis, this dependence on fixed, tuned hyperparameters means RL agents often lack the ability to continuously increase their representation capacity in complex, open-ended environments. This is in clear violation of the fourth necessary condition proposed for open-endedness (see: Chapter 1, [277]). Indeed, the capability of current methods may be severely *understated*, as better hyperparameter configurations alone could boost performance [37, 338]. For example, Zhang *et al.* [338] showed that when effectively tuning on a model-based RL algorithm it is possible to actually break the MuJoCo simulator [299]. Further, Lu *et al.* [175] showed that it is possible to significantly improve the performance of an offline RL algorithm by using Bayesian Optimization rather than grid search.

In this chapter we focus on the recent impressive results for *Population Based Training* (PBT, [116, 167]), which has been a key component of several prominent works in RL [64, 115, 173, 263, 338], including one of the most large scale examples of open-ended learning [296]. PBT works by training agents in parallel, with an evolutionary outer loop optimizing hyperparameters, periodically replacing weaker agents with perturbations of stronger ones (see: Sec 2.2.3).

However, since PBT relies on random search to explore the hyperparameter space, it requires a large population size to tune a small number of parameters, limiting its potential to learn an entire agent configuration. Indeed, the original PBT results required 20 agents to successfully tune three hyperparameters, one of which had to have a narrow range to improve performance [116]. In this chapter we propose a new class of model-based PBT algorithms. Rather than rely on random search in the hyperparameter space, the methods we propose make use of Bayesian Optimization (BO, see: Sec 2.2.2). During training, the BO algorithm uses all data

from previous agents to select a configuration most likely to lead to *the biggest improvement* in performance for the agent for that stage of training.

We call this new class of methods *Population Based Bandits* or PB2, since it relies on a special form of BO called GP-bandit optimization [278]. We show that PB2 is capable of effectively tuning several continuous hyperparameters with a small population size, while also tuning mixed input hyperparameters for optimizing agents with data augmentation in challenging procedurally generated environments.

3.2 Related Work

One of the primary challenges in AutoRL is the inherent nonstationarity of the RL problem, as agents explore different behaviors and corresponding regions of the state space over time [112]. Thus, adaptive methods have become increasingly popular, but they are certainly not new. Indeed, adaptive methods for selecting hyperparameters in RL have been prominent since the 1990s. Sutton & Singh [288] proposed three alternative methods for adaptive weighting schemes in TD algorithms. Kearns & Singh [128] derived upper bounds on the error of temporal-difference algorithms, and used these bounds to derive schedules for the bootstrapping hyperparameter λ. Downey & Sanner [57] used Bayesian Model Averaging to select the λ bootstrapping hyperparameter for TD methods. More recently, White & White [320] proposed λ-greedy to adapt λ as a function of the state and achieve an approximately optimal bias-variance trade-off. Paul *et al.* [220] proposed HOOF which uses random search with off-policy data to periodically select new hyperparameters for policy gradient algorithms.

Several algorithms make use of bandits to adapt hyperparameters. In a distributed setting, Schaul *et al.* [259] proposed to adapt several behavioral hyperparameters, such as the degree of stochasticity of an agent, using a notion of learning progress as feedback. This idea inspired Agent57 [12] which adaptively selects from several exploration policies to become the first algorithm to achieve human level performance on all 57 games in the Arcade Learning Environment [20]. Moskovitz *et al.* [194] showed that adapting the degree of optimism in off

policy actor critic methods could lead to significant improvement in continuous control tasks. In particular, Moskovitz *et al.* [194] observed that their algorithm was able to use different levels of optimism not only across different environments but also at different stages of training in a single environment. Finally, Riquelme *et al.* [247] considered adaptively switching between Temporal Difference (TD) learning and Monte Carlo (MC) policy evaluation using learned confidence intervals that detect biases of TD estimates.

Another approach that is recently growing in popularity is *meta-gradients* [70], a class of methods that considers tuning differentiable hyperparameters using second order gradients. Meta-gradients have been shown to be highly effective for tuning hyperparameters for RL agents [71, 327, 335], while they have also been used to discover auxiliary tasks [310], discover options [311] and learn RL objectives online, demonstrating strong asymptotic performance [328].

For the rest of this chapter we will focus on *Population Based Training* (PBT, Jaderberg *et al.* [116]), an Evolutionary Algorithm (EA, [8]). PBT is most similar to Lamarckian EAs [322] in which parameters are inherited whilst hyperparameters are evolved. Meanwhile, other methods learn both hyperparameters and weights [33, 107]. These works motivate the recently introduced PBT algorithm [116], whereby network parameters are learned via gradient descent, while hyperparameters are evolved. Its key strengths lie in the ability to learn high performing hyperparameter schedules in a single training run, leading to strong performance in a variety of settings [64, 167, 173, 263]. PBT works by focusing on high performing configurations, however, this greedy property means it is unlikely to explore areas of the search space which take longer to perform well, even if they are asymptotically superior. In addition, the core components of the algorithm rely on handcrafted meta-hyperparameters, such as the degree of mutation. These design choices mean the algorithm lacks theoretical guarantees. We address these issues in our work.

In the BO community there has recently been increasing focus on distributed implementations [4, 86] which seek to select a batch of configurations for concurrent evaluation. Despite this increased efficiency, these methods still require multiple

training runs. Another recent method, Freeze-Thaw Bayesian Optimization [292] considers a 'bag' of current solutions, with their future loss assumed to follow an exponential decay. The next model to optimize is chosen via entropy maximization. The method, however, makes assumptions about the shape of the loss curve, does not adapt hyperparameters during optimization (as in PBT) and is sequential rather than parallelized. Our approach takes appealing properties of both these methods. First and foremost we select hyperparameters using Bayesian principles i.e. we seek to select points that are maximally informative given current beliefs before updating the model to form a new posterior. However, we also wish to adapt agent configurations in an online fashion. We note that alternative uses of BO for AutoRL have considered a multi-fidelity setting [68, 253]. This considers a fixed configuration but for shortened time periods, rather than an adaptive one over a longer time period.

Finally, we note that we are not the first to consider combining Bayesian and Evolutionary approaches. In Pelikan *et al.* [223], the authors demonstrate that using an early version of BO improves a simple Genetic Algorithm. These ideas were extended for a number of years [222, 224], but have not recently been extended to modern, deep learning-based settings. We hope this chapter can provide a platform for a renewed focus in this direction.

3.3 Efficient Population Based Training

Throughout this chapter we will be focusing on a specific class of algorithms, introduced in Parker-Holder *et al.* [216, 217]. These methods combine ideas from both PBT and BO to produce an efficient population-based AutoRL algorithm that can tune multiple components of the agent configuration on the fly. Recall that we defined $x \subset \zeta$ to be the set of agent hyperparameters. In this section we will use x_t to refer to the agent hyperparameters after t training steps.

The key theoretical formalism in this chapter is that of *regret*. To formalize this problem, let $F_t(x_t)$ be an objective function under a given set of hyperparameters at timestep t. In this context we consider $F_t(x_t)$ to be the cumulative reward for a deep

RL agent.[1] When training for a total of T steps, our goal is to maximize the final performance $F_T(x_T)$. We formulate this problem as optimizing the time-varying black-box reward function f_t, over $\mathcal{D}$. Every t_{ready} steps, we observe and record noisy observations, $y_t = f_t(x_t) + \epsilon_t$, where $\epsilon_t \sim \mathcal{N}(0, \sigma^2\mathbf{I})$ for some fixed σ^2. The function f_t represents the change in F_t after training for t_{ready} steps, i.e. $F_t - F_{t-t_{\text{ready}}}$. We define the best choice at each timestep as $x_t^* = \text{argmax}_{x_t \in \mathcal{D}} f_t(x_t)$, and so the **regret** of each decision as $\text{REGRET}^x(x_t, x_t^*) = f_t(x_t^*) - f_t(x_t)$. We denote the cumulative regret as simply REGRET_T^x, which refers to $\text{REGRET}_T^x = \sum_{t=1}^T \text{REGRET}^x(x_t, x_t^*)$.

Lemma 1. *Maximizing the final performance F_T of a model with respect to a given hyperparameter schedule $\{x_t\}_{t=1}^T$ is equivalent to maximizing the time-varying black-box function $f_t(x_t)$ and minimizing the corresponding cumulative regret REGRET_T^x,*

$$\max F_T(x_T) = \max \sum_{t=1}^{T} f_t(x_t) = \min \sum_{t=1}^{T} \text{REGRET}^x(x_t, x_t^*). \tag{3.1}$$

In subsequent sections, we present a time-varying bandit approach which is used to minimize the cumulative regret. Lemma 1 shows this is equivalent to maximizing the final performance/reward of a neural network model (see: Section A.2 in the appendix for the proof).

3.3.1 The PB2 Algorithm

We now introduce Population Based Bandits (PB2) for optimizing the hyperparameter schedule $\left(x_t^b\right), \forall t = 1, ..., T$ using parallel agents $b = 1, ..., B$. After each agent b completes t_{ready} training steps, we store the data (y_t^b, t, x_t^b) in a dataset D_t which will be used to make an informed decision for the next set of hyperparameters.

We below present the mechanism to select the next hyperparameters for parallel agents. Motivated by the equivalence of the maximized reward and minimized cumulative regret in Lemma 1, we propose the parallel time-varying bandit optimization.

[1] The framework is general enough to consider other applications, but we focus on RL in this thesis.

3.3.2 Parallel Gaussian Process Bandits for a Time-Varying Function

We first describe the time-varying Gaussian process as the surrogate models, then we extend it to the parallel setting for our PB2 algorithm.

Time-varying Gaussian process as the surrogate model. Following previous works in the GP-bandit literature [278], we model f_t using a Gaussian Process (GP, Rasmussen & Williams [245]). Recall from our previous discussion (see Section: 2.2.2) that a GP is specified by a mean and kernel (or covariance) function. When we introduced GPs we assumed that we were training models end-to-end,

To represent the non-stationary nature of a neural network training process, we cast the problem of optimizing neural network hyperparameters as time-varying bandit optimization. We follow [25] to formulate this problem by modeling the reward function under the time-varying setting as follows:

$$f_1(x) = g_1(x), \quad f_{t+1}(x) = \sqrt{1-\omega} f_t(x) + \sqrt{\omega} g_{t+1}(x) \quad \forall t \geq 2, \tag{3.2}$$

where $g_1, g_2, ...$ are independent random functions with $g \sim GP(0, k)$ and $\omega \in [0, 1]$ models how the function varies with time, such that if $\omega = 0$ we return to GP-UCB and if $\omega = 1$ then each evaluation is completely independent. This model introduces a new hyperparameter (ω), however, we note it can be optimized by maximizing the marginal likelihood. This leads to the extensions of Equation 2.20 and 2.21 using the new covariance matrix $\tilde{\mathbf{K}}_t = \mathbf{K}_t \circ \mathbf{K}_t^{\text{time}}$ where $\mathbf{K}_t^{\text{time}} = [(1-\omega)^{|i-j|/2}]_{i,j=1}^{T}$ and $\tilde{\mathbf{k}}_t(x) = \mathbf{k}_t \circ \mathbf{k}_t^{time}$ with $\mathbf{k}_t^{time} = [(1-\omega)^{(T+1-i)/2}]_{i=1}^{T}$. Here $\circ$ refers to the Hadamard product.

Selecting hyperparameters for parallel agents. In the PBT setting, we consider an entire population of parallel agents. This changes the problem from a sequential to a *batch* blackbox optimization. This poses an additional challenge to select multiple points simultaneously x_t^b without full knowledge of all $\{(x_t^j, y_t^j)\}_{j=1}^{b-1}$. A key observation in Desautels *et al.* [55] is that since a GP's variance (Eqn. 2.21) does not depend on y_t, the acquisition function can account for incomplete trials by

updating the uncertainty at the pending evaluation points. Concretely, we define x_t^b to be the b-th point selected in a batch, after t timesteps. This point may draw on information from $t + (b-1)$ previously selected points. In the single agent, sequential case, we set $B = 1$. Thus, at the iteration t, we find a next batch of B samples $\left[x_t^1, x_t^2, ...x_t^B\right]$ by sequentially maximizing the following acquisition function:

$$x_t^b = \underset{x \in \mathcal{D}}{\operatorname{argmax}}\, \mu_{t,1}(x) + \sqrt{\beta_t}\sigma_{t,b}(x), \forall b = 1, ...B \tag{3.3}$$

for $\beta_t > 0$. In Eqn. (3.3) we have the mean from the previous batch ($\mu_{t,1}(x)$) which is fixed, but can update the uncertainty using our knowledge of the agents currently training ($\sigma_{t,b}(x)$). This significantly reduces redundancy, as the model is able to explore distinct regions of the space.

3.3.3 PB2 Algorithm and Convergence Guarantee

Algorithm 1 Population Based Bandits (PB2)

Input: Rank threshold parameter λ, population size B.
Initialize: Network weights $\{\theta_0^b\}_{b=1}^B$, hyperparameters $\{x_0^b\}_{b=1}^B$, dataset $D_0 = \emptyset$
(in parallel) for $t = 1, \ldots, T-1$ **do**

1. **Update Models:** $\theta_t^b \leftarrow \text{step}(\theta_{t-1}^b | x_{t-1}^b)$
2. **Evaluate Models:** $y_t^b = F_t(x_t^b) - F_{t-1}(x_{t-1}^b) + \epsilon_t$ for all b
3. **Record Data:** $D_t = D_{t-1} \cup \{(y_t^b, t, x_t^b)\}_{b=1}^B$
4. If $t \mod t_{\text{ready}} = 0$:

   - **Copy weights:** Rank agents, if θ^b is in the bottom $\lambda\%$ then copy weights θ^j from the top $\lambda\%$.
   - **Select hyperparameters:** Fit a GP model to D_t and select hyperparameters $x_t^b, \forall b \leq B$ by maximizing Eq. (3.3).

Return the best trained model θ

To estimate the GP predictive distribution in Equations (2.20, 2.21), we use the product form $\tilde{k} = k^{\text{SE}} \circ k^{\text{time}}$ [25, 142], which uses the squared exponential kernel from Equation 2.19 while also considering the time varying nature of training a neural network. Then, we use Equation 3.3 to select a batch of points by utilizing the reduction in uncertainty for models currently training. As far as we know, this

is the first use of a time-varying kernel in the PBT-style setting. Unlike PBT, this allows us to efficiently make use of data from previous trials when selecting new configurations, rather than reverting to a uniform prior, or perturbing existing configurations. The full algorithm is shown in Algorithm 1.

Next we present our main theoretical result, showing that PB2 is able to achieve sublinear regret, the first such result for a PBT-style algorithm.

Theorem 2. *Let the domain $\mathcal{D} \subset [0, r]^d$ be compact and convex where d is the dimension and suppose that the kernel is such that $f_t \sim GP(0, k)$ is almost surely continuously differentiable and satisfies Lipschitz assumptions $\forall L_t \geq 0, t \leq \mathcal{T}, p(\sup \left|\frac{\partial f_t()}{\partial^{(d)}}\right| \geq L_t) \leq ae^{-(L_t/b)^2}$ for some a, b. Pick $\delta \in (0, 1)$, set $\beta_T = 2 \log \frac{\pi^2 T^2}{2\delta} + 2d \log rdbT^2 \sqrt{\log \frac{da\pi^2 T^2}{2\delta}}$ and define $C_1 = 32/\log(1 + \sigma_f^2)$, the PB2 algorithm satisfies the following regret bound after T time steps over B parallel agents with probability at least $1 - \delta$:*

$$\textsc{Regret}^x_{TB} = \sum_{t=1}^{T} f_t(^*_t) - \max_{b=1,\ldots,B} f_{t(t,b)} \leq \sqrt{C_1 T \beta_T \left(\frac{T}{\tilde{N}B} + 1\right)\left(\gamma_{\tilde{N}B} + \left[\tilde{N}B\right]^3 \omega\right)} + 2$$

the bound holds for any block length $\tilde{N} \in \{1, \ldots, T\}$ and $B \ll T$.

The proof of this result is in the Appendix (see: Section A.2.2). This result shows the regret for PB2 decreases as we increase the population size B. This should be expected, since adding computational resources should benefit final performance (which is equivalent to minimizing regret). We demonstrate this property in Figure 3.1. Note that when using a single agent $B = 1$, our bound becomes the time-varying GP-UCB setting in [25].

In our setting, if the time-varying function is highly correlated, i.e., the information between $f_1(.)$ and $f_T(.)$ does not change significantly, we have $\omega \to 0$ and $\tilde{N} \to T$. Then, the regret bound grows sublinearly with the number of iterations T, i.e., $\lim_{T\to\infty} \frac{\textsc{Regret}^x_{TB}}{T} = 0$. On the other hand (in the worst case), if the time-varying function is not correlated at all, such as $\tilde{N} \to 1$ and $\omega \to 1$, then PB2 achieves linear regret [25].

Remark. This theoretical result is novel and significant in two folds. First, by showing the equivalent between maximizing reward and minimizing the bandit regret, this regret bound quantifies the gap between the optimal reward and the achieved reward (using parameters selected by PB2). The regret bound extends the result established by [25, 278], to a more general case with parallelization. To the best of our knowledge, this is the first kind of convergence guarantee in a PBT-style algorithm.

Our approach is advantageous against all existing batch BO approaches [55, 86] in that PB2 considers optimization in a *single training run* of parallel agents while the existing works need to evaluate using *multiple training runs* of parallel agents which are more expensive. In addition, PB2 can learn a *schedule* of hyperparameters while the existing batch BO can only learn a static configuration.

3.3.4 Empirical Results

We consider optimizing a policy for continuous control problems from the OpenAI Gym [28]. In particular, we seek to optimize four continuous hyperparameters for Proximal Policy Optimization (PPO, Schulman *et al.* [267]), for the `BipedalWalker` and `LunarLanderContinuous` tasks (see: Table A.2 in the Appendix for the full ranges). Our primary baseline is PBT, where we use an identical configuration to PB2 aside from the selection of x_t^b (the explore step). We also compare against a random search (RS) baseline [21]. Random search is a challenging baseline because it does not make assumptions about the underlying problem, and typically achieves close to optimal performance asymptotically [109]. Finally, we also compare our results against a recent state-of-the-art distributed algorithm (ASHA, Li *et al.* [168]). ASHA is one of the most recent distributed methods, shown to outperform PBT for supervised learning. We believe we are the first to test it for RL. We evaluate population sizes of $B \in \{4, 8\}$, which means the algorithm can be run locally on most modern computers. This is significantly less than the $B > 20$ used in the original PBT paper [116].

All experiments were conducted using the tune library [170, 171].[2] Our GP implementation is extended from GPy [88], where we use the squared exponential kernel in combination with the time kernel as described in Section 3.3.2. We optimize all GP-kernel hyperparameters, as well as the block length $\tilde{N}$, by maximizing the marginal likelihood [245].

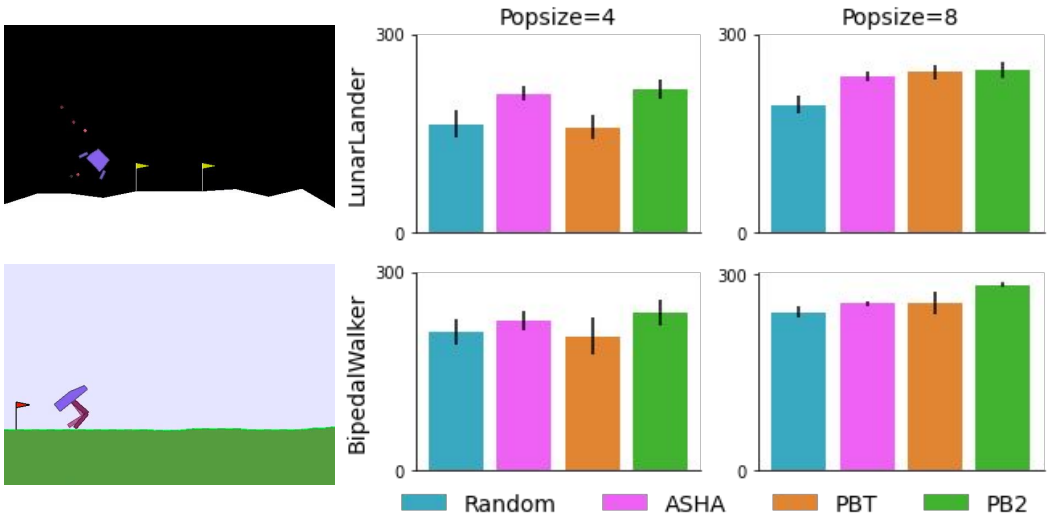


Figure 3.1: Final performance of the best PPO agent when training with different hyperparameter optimization algorithms. Agents are trained for 1M steps, plots show the mean and the standard error of the mean for ten independent runs.

As we see in Figure 3.1, PB2 performs consistently well in both environments with both population sizes. By contrast, with a population size of four we see that PBT is worse than random search, despite the ability to produce a dynamic schedule. This example demonstrates the possibilities of PB2 for tuning more hyperparameters with fewer resources.

3.4 PB2 with Mixed Inputs

In the previous sections we showed it was possible to increase the number of *continuous* hyperparameters we can tune on the fly with a population of agents. However, our goal is much grander—to tune the entire agent configuration.

[2]See code here: https://github.com/jparkerholder/PB2 and in an open source demo as a part of ray tune https://docs.ray.io/en/latest/tune/examples/pb2_example.html

In this section we discuss approaches to extend these ideas further, dealing with *mixed-input* hyperparameter spaces, consisting of both continuous and categorical variables. This could represent multiple possible components of an RL agent, for example the choice of optimizer or even the type of exploration strategy [12]. Crucially, as we will see, the methods we propose can model the *dependence* between the hyperparameters.

3.4.1 Problem Setting

Recall in the previous section we proposed a method for optimizing continuous hyperparameters x, a subset of the full unspecified component of the RL training pipeline, ζ. In this section we increase the search space, defining $c \in \mathbb{N}$ to be a vector of *categorical* agent hyperparameters. We then seek to search over a hybrid space $z = [x, c]$ containing both continuous and categorical variables. This allows us to increase the proportion of the agent configuration we are able to tune.

3.4.2 Extending PB2 to Categorical Variables

We propose a new class of hierarchical population based approaches for handling continuous and categorical variables. To the best of our knowledge, this is the first provably efficient approach for optimizing the categorical variables in the PBT family. First we introduce a new time-varying version of the parallel EXP3 algorithm, which we call TV.EXP3.M, used in all of our methods to select categories. We then present three alternative approaches to subsequently select the continuous variables in a hierarchical fashion, varying in their degree of dependence on the categorical choices.

3.4.3 Time-varying Parallel EXP3

We introduce TV.EXP3.M, a new algorithm for parallel multi-armed bandits (MAB) in a time-varying setting with adversarial feedback. TV.EXP3.M is an extension of the multiple play EXP3 algorithm (or EXP3.M, Uchiya *et al.* [306]) for time-varying rewards. At each round, TV.EXP3.M takes multiple actions from the set of all possible choices [306], making it possible to use this approach with a

population of agents. See Algorithm 2 for a high level summary and Algorithm 13 in Section A.3.1 for further details.

Algorithm 2 TV.EXP3.M (brief)

Input: $\gamma = \sqrt{\frac{C\ln(C/B)}{(e-1)BT}}$, $\alpha = \frac{1}{T}$, C #categorical choice, T #max iteration, B #multiple play
Initialize: $w_c = 1, \forall c = 1...C$ and denote $\eta = (\frac{1}{B} - \frac{\gamma}{C})\frac{1}{1-\gamma}$
for $t = 1$ *to* T **do**
1. Normalize w to prevent from over exploitation
2. Compute the prob for each arm $p_t^c, \forall c$
3. Select a batch of B choices: $S_t = [c_t^1, c_t^2, ..., c_t^B] = \text{DepRound}\left(B, \left[p_t^1 p_t^2 ....p_t^C\right]\right)$
4. Observe the reward after evaluating S_t
5. Update the weights using η for each arm $w_t^c, \forall c$

end

We treat each categorical choice as an arm in the bandit setting. Given a collection of C categories, we select a batch of B points

$$S_t = [c_t^1, c_t^2, ..., c_t^B] = \text{TV.EXP3.M}(D_{t-1}). \tag{3.4}$$

To efficiently select a set of B distinct arms from $[C]$, we use the technique of dependent rounding (DepRound) while satisfying the condition that each arm c is selected with probability p_c exactly [81]. TV.EXP3.M is provably efficient, with a sublinear regret bound.

Theorem 3. *Set* $\alpha = \frac{1}{T}$ *and* $\gamma = \min\left\{1, \sqrt{\frac{C\ln(C/B)}{(e-1)BT}}\right\}$, *we assume the reward distributions changes at arbitrary instances, but the total number of change points is no more than* $V \ll \sqrt{T}$ *times. The expected regret of TV.EXP3.M satisfies the following sublinear bound*

$$\mathbb{E}\left[Regret_{TB}^x\right] \leq [1 + e + V]\sqrt{(e-1)\frac{CT}{B}\ln\frac{CT}{B}}.$$

3.4.4 New Exploration Strategies

Now we are ready to present two novel hierarchical exploration strategies. Both methods use TV.EXP3.M to select categories before subsequently choosing continuous variables.

Algorithm 3 PB2-Mult: explore

1. Select $\{c_t^b\}_{b=1}^B$ using TV.EXP3.M
2. Filter dataset.
3. Select x_t^b by optimizing Eqn. (3.3).

Output: $\{c_t^b\}_{b=1}^B$, $\{x_t^b\}_{b=1}^B$

PB2-Mult (Algorithm 3): We consider a special version of dependency in which the presence of the continuous variables entirely depends on if the corresponding category is switched on. This not only allows us to fully incorporate dependence between hyperparameters, but also allows us to have different hyperparameters depending on the category. For example, if our categorical variable is the choice of neural network optimizer, then the current optimal learning rate may vary significantly if using Adam [134] or SGD. Furthermore, Adam requires additional hyperparameters, β_1 and β_2. We handle this dependency by constructing *multiple* time-varying GP surrogate models, each corresponding to a choice of category. We call this method **PB2-Mult**. PB2-Mult is the best choice if the continuous variables are highly dependent on the category, or if there are differing numbers of continuous variables for each category.

Algorithm 4 PB2-Mix: explore

1. Select $\{c_t^b\}_{b=1}^B$ using TV.EXP3.M
2. Select x_t^b by optimizing Eqn. (3.3), using the kernel from Eqn. (3.6)

Output: $\{c_t^b\}_{b=1}^B$, $\{x_t^b\}_{b=1}^B$

PB2-Mix (Algorithm 4): We also propose a mechanism to incorporate dependence between continuous and categorical variables *directly into the GP kernel*, in what we call **PB2-Mix**. To do this, we first introduce a kernel between C-dimensional categorical inputs, $c \in \mathbb{N}^C$, as follows:

$$\mathbf{k}_c = \frac{\sigma_2}{C} \sum \mathbb{I}(c, c') \tag{3.5}$$

where σ^2 is the kernel variance, and $\mathbb{I}(c, c')$ returns 1 if $c = c'$ and is zero otherwise. This kernel was originally used in Ru *et al.* [249], who note the similarity with the squared exponential kernel. However, in our setting we must also consider the

time varying nature of the problem, since we are optimizing hyperparameters for a changing agent. We thus define our time-varying categorical kernal as follows: $\mathbf{k}_{ct} = \mathbf{k}_c \circ \mathbf{k}_t^{time}$, where $\mathbf{k}_t^{time}$ is defined in Section 3.3.2 and $\circ$ refers to the Hadamard product. Now we have a way to quantify similarity between categorical choices, we wish to jointly model the continuous and categorical variables. In a similar fashion to Ru *et al.* [249], we wish to consider a mixture of both sum and product kernels, weighted by $\lambda \in [0, 1]$, as follows:

$$\mathbf{k}_z(z, z') = (1 - \lambda)\left(\tilde{\mathbf{k}}_t(x, x') + \mathbf{k}_{ct}(c, c')\right) + \lambda\tilde{\mathbf{k}}_t(x, x')\mathbf{k}_{ct}(c, c') \tag{3.6}$$

where $\tilde{\mathbf{k}}_t(x, x')$ is defined in Section 3.3.2. Note that λ can be optimized alongside other kernel hyperparameters, using gradients defined in Appendix A.3.3. Indeed, this choice of λ decides whether to prioritize modelling dependence (high λ) vs. reducing λ if there are many zero entries in one of the kernels (making the product zero).

To make use of Equation 3.6, we first select a batch of B categorical choices $c_t^b, \forall b \leq B$ using TV.EXP3.M. Next we utilize the new kernel to learn the time-varying GP model and select the next batch of continuous points $x_t^b, \forall b \leq B$ following Equation 3.3, as in PB2-Rand. Each agent b will then train with $[x_t^b, c_t^b]$. Thus, PB2-Mix in principle achieves the best of both worlds: it is able to learn the relative importance of the continuous and categorical kernels, while including all of the data when making decisions.

3.4.5 Main Theoretical Results

We extend the sublinear regret bound from PB2-Rand to PB2-Mult, providing performance guarantees. Under the categorical-continuous setting, we show the equivalence that minimizing the cumulative regret R_T is equivalent to maximizing the reward, presented in Lem. 1 in the appendix. Our main theoretical result is to derive the regret bound for PB2-Mult.

Theorem 4. *The PB2-Mult algorithm has cumulative regret bounded with high probability*

$$\mathbb{E}\left[\textsc{Regret}^{x}_{TB}\right] \lesssim \mathcal{O}\left(V\sqrt{\frac{CT}{B}\ln T} + \sqrt{\frac{T^2_{c^*_t}}{\tilde{N}B}(\gamma_{\tilde{N}B} + \left[\tilde{N}B\right]^3\omega)}\right)$$

for $\tilde{N} \to T_{c^*_t}$, $\omega \to 0$, *and* $V \ll \sqrt{T}$.

From Theorem 3, we have $T_{c^*_t} \to \infty$ when $T \to \infty$. Asymptotically PB2-Mult achieves sublinear convergence rate, i.e., $\lim_{T\to\infty} \frac{\mathbb{E}\left[\textsc{Regret}^{x}_{TB}\right]}{T} = 0$. We refer to Sec. A.3.2 in the Appendix for proofs.

3.4.6 Empirical Results

There has recently been increased interest in testing RL algorithms in procedurally generated environments [44, 202]. These typically consist of a set of parameters which are used to generate *levels*, consisting of changes to the observation or layout of the environment, requiring agents to generalize [124, 248]. In this paper we focus on the Procgen benchmark [44], a set of 16 environments with visual input and a fixed number of procedurally generated levels. Given the recent success of data augmentation in Procgen, we adopt the author's implementation of *data regularized actor critic* or DrAC [240], using a ResNet architecture [101] as originally introduced in IMPALA [64]. As well as the categorical data augmentation, we also tune the regularization coefficient α_r introduced in DrAC, as well as more classically well-studied PPO hyperparameters such as the clip parameter ϵ, the learning rate and entropy exploration coefficient. The bounds for each, as well as the fixed hyperparameters, are given in the Appendix (Table A.3). We learn these hyperparameters, as well as the categorical data augmentation *on the fly* in a single training run, starting with random values.

We train a population of $B = 4$ agents, each for 25M steps, with $t_{\text{ready}} = 500\text{k}$ steps, thus the explore step is called approximately fifty times. All of the PBT-style methods use the same truncation selection approach, whereby the bottom 25% of agents are replaced with copies of the top 25%. Since we have a population size of $B = 4$ this equates to replacing the worst agent with the best. We use seven Procgen games: BigFish, CaveFlyer, CoinRun, FruitBot, Jumper, Leaper and StarPilot. At the end, each agent is evaluated on 100 train and test levels

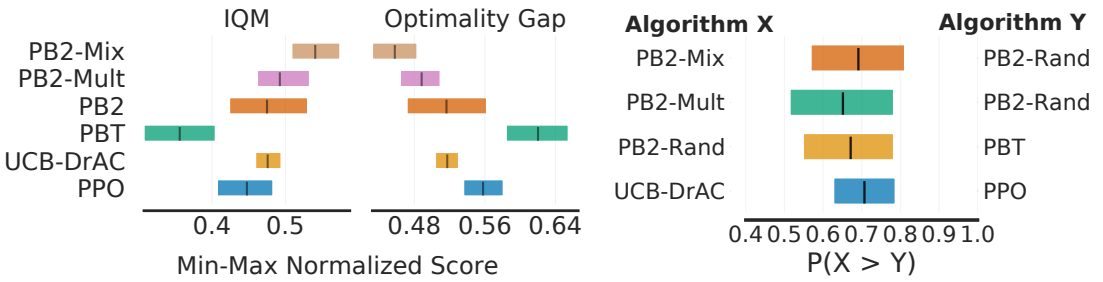
Table 3.1: Test performance for seven Procgen games. † indicates training was conducted for a single agent for 25M timesteps. Hyperparameters were initialized at optimized values, meaning the effective number of timesteps is much higher. Final performance is the average of 100 trials and results were taken from [240]. ‡ indicates training was conducted by a population of four agents for 25M timesteps each, with several hyperparameters initialized at random. Each agent is evaluated for 100 trials on train and test levels and we present the mean test performance of the agent with the best training performance. The "Normalized" returns are with respect to the PB2-Rand baseline, $\star$ indicates $p < 0.05$ in a t-test over PB2-Rand. Bold = within 1 std of the best mean.

Environment	PPO†	UCB-DrAC†	PBT‡	PB2-Rand‡	PB2-Mult‡	PB2-Mix‡
BigFish	4.0 ± 1.2	$\mathbf{9.7 \pm 1.0}$	6.6 ± 2.4	8.2 ± 0.1	$\mathbf{9.6 \pm 1.7}$	$\mathbf{10.6 \pm 1.9}$
CaveFlyer	5.1 ± 0.9	$\mathbf{5.3 \pm 0.9}$	4.4 ± 1.0	$\mathbf{5.8 \pm 1.8}$	$\mathbf{5.6 \pm 0.8}$	$\mathbf{6.0 \pm 0.7}$
CoinRun	$\mathbf{8.5 \pm 0.5}$	$\mathbf{8.5 \pm 0.5}$	6.8 ± 1.0	$\mathbf{8.1 \pm 0.1}$	$\mathbf{8.5 \pm 0.4}$	$\mathbf{8.1 \pm 0.2}$
FruitBot	26.7 ± 0.8	28.3 ± 0.9	25.6 ± 0.2	$\mathbf{29.5 \pm 1.9}$	$\mathbf{29.6 \pm 0.8}$	$\mathbf{29.0 \pm 0.6}$
Jumper	5.8 ± 0.5	$\mathbf{6.4 \pm 0.6}$	$\mathbf{6.2 \pm 0.3}$	5.4 ± 0.5	$\mathbf{5.9 \pm 0.1}$	$\mathbf{6.2 \pm 0.3}$
Leaper	$\mathbf{4.9 \pm 0.7}$	$\mathbf{5.0 \pm 0.3}$	3.6 ± 0.6	$\mathbf{5.0 \pm 2.4}$	$\mathbf{5.0 \pm 1.4}$	$\mathbf{6.6 \pm 1.9}$
StarPilot	24.7 ± 3.4	30.2 ± 2.8	26.9 ± 7.9	33.6 ± 3.1	$\mathbf{35.3 \pm 1.5}$	$\mathbf{36.3 \pm 2.4}$
Normalized (%)			85.6 ± 22.4	100 ± 23.8	105.3 ± 14.6	$112.1 \pm 20.4^{\star}$

and we present the test performance from the agent with the best final training performance in each population. Our primary baseline is PB2-Rand, which we note was already shown to improve upon both Hyperband [168, 169] and vanilla BO [27] when optimizing continuous agent hyperparameters (see: Section 3.3.4). As such, we choose to use our computational resources to run five trials for all seven games rather than by confirming this result. Instead, we do include the original PBT algorithm as a baseline, as well as single agent results from Raileanu *et al.* [240]. All of the PBT results can be replicated using our open source codebase, which was designed to make use of our single GPU available and thus is not parallelized.[3]

The final results are shown in Table 3.1, where each agent is evaluated for 100 trials, a common protocol in Procgen [121, 240]. For population-based methods, we select the agent with the best *training performance* and show its test performance. Given the challenge of comparing results, we present an additional metric which is a score normalized by the PB2-Rand performance. For each seed in each game,

[3] https://github.com/jparkerholder/procgen_autorl



(a) Aggregate Performance Statistics

(b) Probability of Improvement

Figure 3.2: (a) Aggregate performance statistics across all seven Procgen environments. (b) The probability of improvement on a new task from the same distribution. Shaded areas represent 95% confidence intervals.

the score is divided by the mean PB2-Rand score for that given game. We then show the mean of these normalized scores, comprising of 35 results. In addition, we note the challenge in comparing aggregate metrics in RL, and as such, we also make use of the recently introduced `rliable` library, designed to provide robust metrics that take into account the uncertainty in experimental results [2]. In Figure 3.2 we show aggregate performance statistics across all tasks, where we clearly see that PB2-Mix and PB2-Mult outperform PB2-Rand. Our other results confirm what was already known—with limited data ($B = 4$) the original PBT performs poorly (worse than PPO without data augmentation), while PB2-Rand provides large improvements over this.

Despite using a small population size, we are able to train policies that perform competitively with state-of-the-art methods which used a much larger grid search. We see that PB2-Rand significantly outperforms PBT, and furthermore our new approaches modeling categorical variables improve upon PB2-Rand. Notably, we see the strongest performance from PB2-Mix, which is able to exploit the dependence between all variables in a joint kernel. The average gains for PB2-Mix over PB2-Rand are statistically significant, with $p = 0.044$ in Welch's t-test [319], used since it is more reliable when the two samples have unequal variances.

Do we need to model dependence? To explore the dependence of the parameters in the Procgen setting, we tested relationship between each continuous

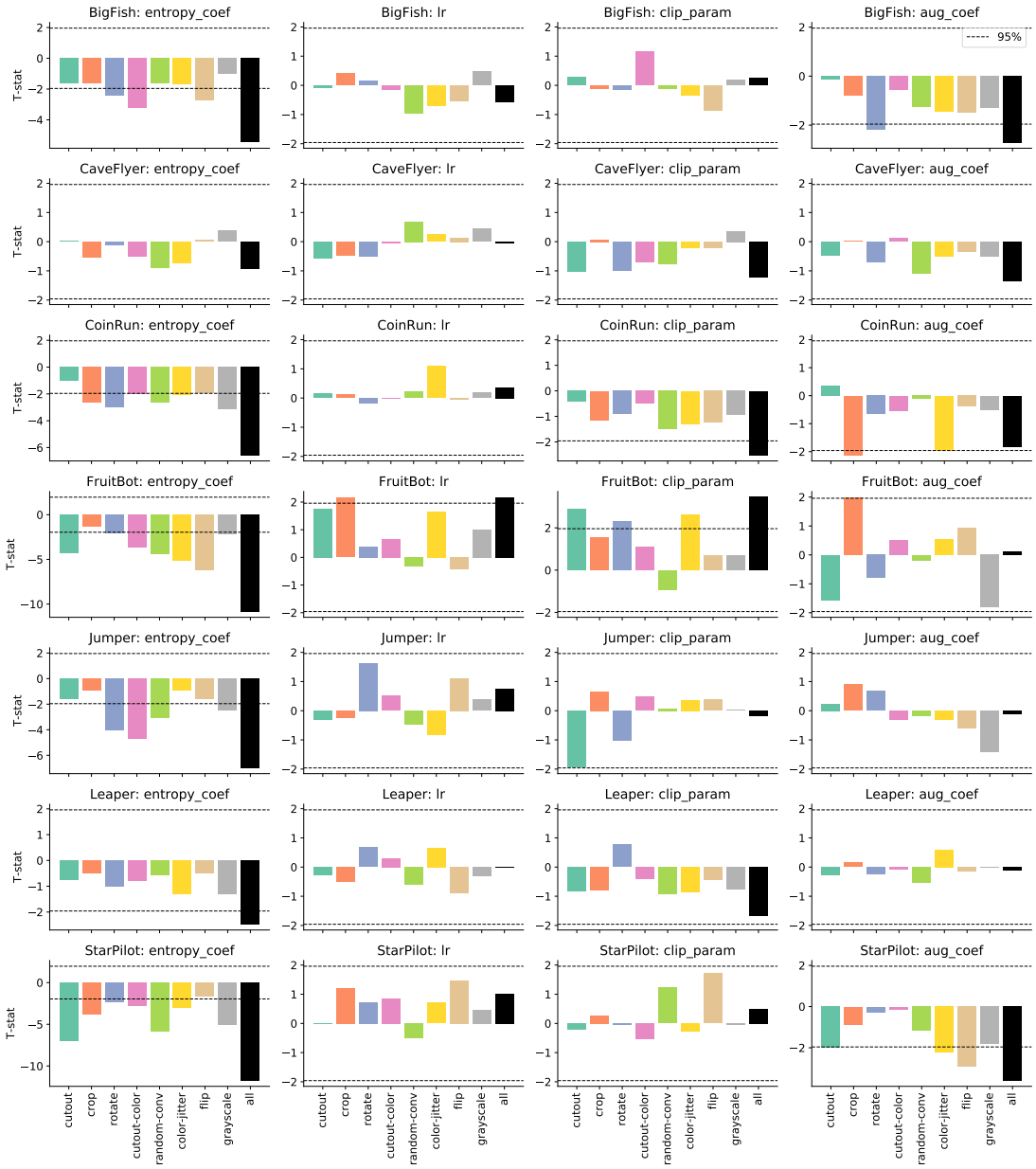


Figure 3.3: T-values for univariate regressions predicting reward change given a continuous hyperparameter, conditioned on category. We see that the relationship between continuous variables and training performance often exhibits a dependence on the data augmentation type. These plots use data from all runs of all algorithms. Plots show critical t-values for $p < 0.05$.

hyperparameter and the subsequent change in reward ($f_t(.)$). For each *individual* hyperparameter, we *conditioned the data on the category* currently being used, and fit a linear regression model. In Figure 3.3 we show the t-values for all environment-

hyperparameter pairs. As we see, the relationship between continuous variables and learning performance varies depending on the category selected, confirming a dependence in Procgen [275]. In particular, we see for BigFish and Jumper the clip parameter relationship with training performance is heavily dependent on the data augmentation type used. Interestingly we also include an example from FruitBot, where the dependence seems to be weaker as the clip param is positively related with improving policies for all but one category. This is likely the reason for relatively stronger performance for PB2-Rand in FruitBot.

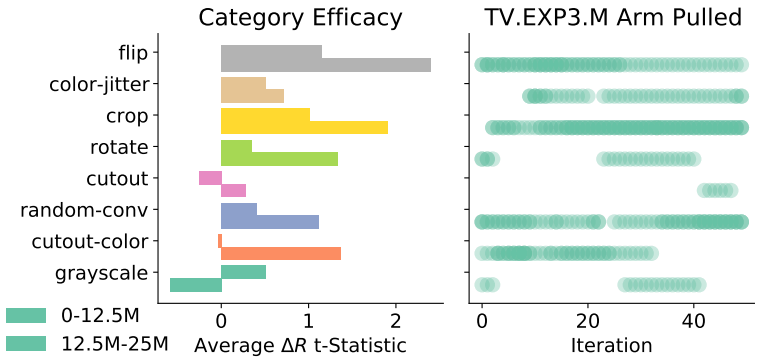
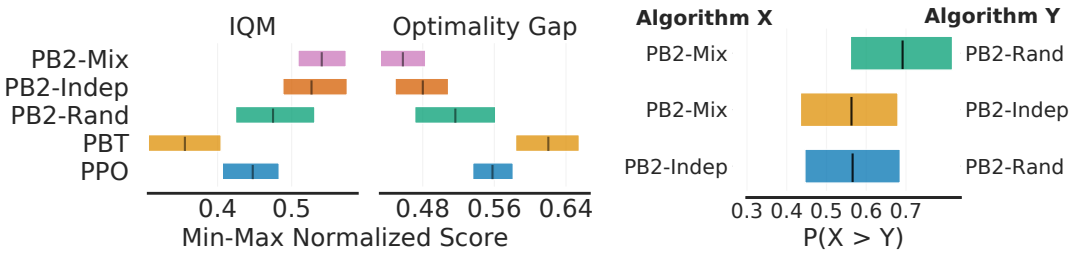


Figure 3.4: Left: The t-statistic for the mean reward change for each category in the BigFish environment, across all experiments, in the first and second half of training. This is calculated by taking the mean reward and normalizing by population size and standard deviation. Right: All arms selected by TV.EXP3.M.

We also investigate the effectiveness of TV.EXP3.M in selecting the data augmentation type. In Figure 3.4 we show the t-statistic for the mean change in reward for each category in the BigFish environment, as well as all the categories selected by the new PB2 variants using TV.EXP3.M. We see that flip is the most selected augmentation, especially in the first half of training when it has a particularly large t-value. Interestingly, we also see that cutout-color is frequently selected at the *beginning*, but not used at all at the end when it is less effective, thus demonstrating TV.EXP3.M is able to adapt effectively.

Now we have seen there is clear evidence of dependence between the hyperparameters, we seek to answer this question on the downstream task by training RL agents. We introduce an ablation of our method called "PB2-Indep" which uses



(a) Aggregate Statistics (b) Probability of Improvement

Figure 3.5: Ablation Study:(a) and b) show aggregate performance statistics vs. an ablation "PB2-Indep" which independently (yet efficiently) selects continuous and categorical variables.

TV.EXP3.M to select the data augmentation type but then ignores this information when using the GP to select continuous hyperparameters. We once again evaluate this method with five runs on all seven Procgen games, and show the aggregate results in Figure 3.5. Indeed, we see that PB2-Indep does improve upon PB2-Rand, since selecting the best data augmentation should provide performance gains, but we do not see the full benefit of PB2-Mix which is able to select continuous variables conditioned on this information. We see that the probability of improvement for PB2-Mix over PB2-Indep and PB2-Indep over PB2-Rand are both equal (just over 56%), which indicates the gain for PB2-Mix comes equally from efficiently selecting the data augmentation type and then subsequently conditioning on the output of that selection when selecting continuous variables. When combined, we get over 69% probability of improvement vs. the baseline.

3.5 Discussion and Future Extensions

The key benefit of PBT algorithms is the ability to make use of large population sizes with the same wall clock time of a single agent. Thus far, we have only considered the resource constrained setting which is important but not representative of a setting that would be useful for labs with access to more resources. Thus, the question remains: how do these new methods scale? To test this we train with a population size of 12 for PB2-Rand, PB2-Mult and PB2-Mix on the BigFish task. We repeat the experiment for three seeds and report the mean test return from the

agent with the best training performance in 3.5c). As we see, PB2-Mix remains the strongest method, with final performance of 16 ± 1.15. This is comparable to the state of the art performance [121, 239]. Since our method is orthogonal to these, it is likely combining them could provide further gains.

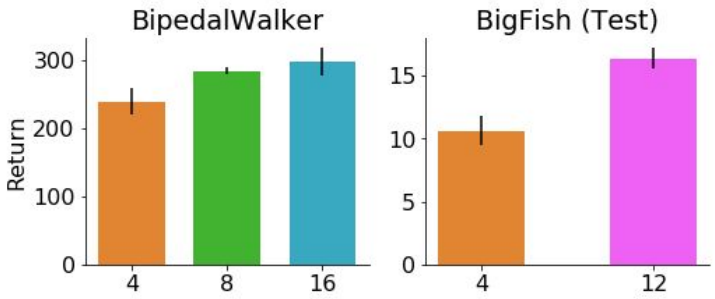


Figure 3.6: Scaling up: shows the test curves with $B = 12$ agents, plots show the mean of three runs with error bars showing the standard error of the mean.

We are particularly excited by many future directions from here. The most natural is to adapt a greater proportion of the agent configuration: in that light, a very recent work [314] proposed to extend PB2 using "generational learning" [296] to adapt architectures on the fly. This leads to sizable gains on the brax environment [77], with agents learning complex architectures that would be challenging to train from scratch. But to achieve a truly open-ended RL system we may need to go even further. Can we learn and adapt entire algorithms altogether, extending recent work [43, 137, 207]? Outside of increasing the search space, improvements can be made on the algorithm itself, for example exploring whether it is possible to share experience across agents [76]. We can also consider whether we can benefit from encouraging these agents to be diverse with respect to one another [218, 235]. Finally, we are also curious to see whether recent innovations in other areas of AutoML can be combined in a PBT-style framework [246]. We think the combination of these ideas could lead to dramatic gains in performance for RL agents, ultimately making it possible to go beyond the capability of human-designed configurations. If we can achieve this, the only thing missing may be a sufficiently complex environment.

4

Environments that Continuously Propose New Challenges

"Everyone has a plan until they get punched in the mouth."

— Mike Tyson

Contents

4.1 Introduction

In our discussion so far, we assumed that our agent was given a fixed environment, or distribution of environments, and was able to tune itself to achieve strong performance. But what happens next? Unless our environment already captures all of the intricacies of the real world, it is likely that our agent is limited in its

generality and the system will plateau, no longer producing new, interesting problems. Furthermore, if we instead place an agent inside an already complex environment it may be too challenging to learn for even the most sophisticated RL algorithms [100].

This chapter focuses on the use of adaptive curricula for training more generally-capable agents. By adapting the training distribution over the parameters of an environment, adaptive curricula have been shown to produce more robust policies in fewer training steps [121, 233]. For example, these parameters may correspond to friction coefficients in a robotics simulator or maze layouts for a navigation task. Each concrete setting of parameters results in an environment instance called a *level*. Indeed, adaptive curricula have played a key role in many prominent RL agents, either over opponents [312], game levels [296], or parameters of a simulator [7, 209].

Unsupervised Environment Design (UED, Dennis *et al.* [54]) formalizes the problem of finding adaptive curricula, whereby a teacher agent designs levels using feedback from a student, which seeks to solve them. When using *regret* as feedback, Dennis *et al.* [54] showed that if the system reaches equilibrium, then the student must follow a minimax regret strategy, i.e. the student would be capable of solving all solvable environments. Empirically, this approach produces student policies exhibiting impressive zero-shot transfer to challenging human designed environments [54, 92, 120]. However, training such an adversarial teacher remains a challenge, and so far, the strongest empirical results come from curating randomly sampled levels for high-regret, rather than learning to directly design such levels [120]. Such a curational approach is unable to take advantage of any previously discovered level structures, and its performance can be expected to degrade as the size of the design space grows.

An important benefit of adaptive curricula is the possibility of open-ended learning [277], given the curriculum can be steered toward constantly designing novel tasks for the agent to solve. While generating truly open-ended learning remains a grand challenge [281], recent works in the evolutionary community have taken the first steps in this direction, through methods such as Minimal Criteria Coevolution (MCC, Brant & Stanley [26]) and POET [315, 316]. These approaches

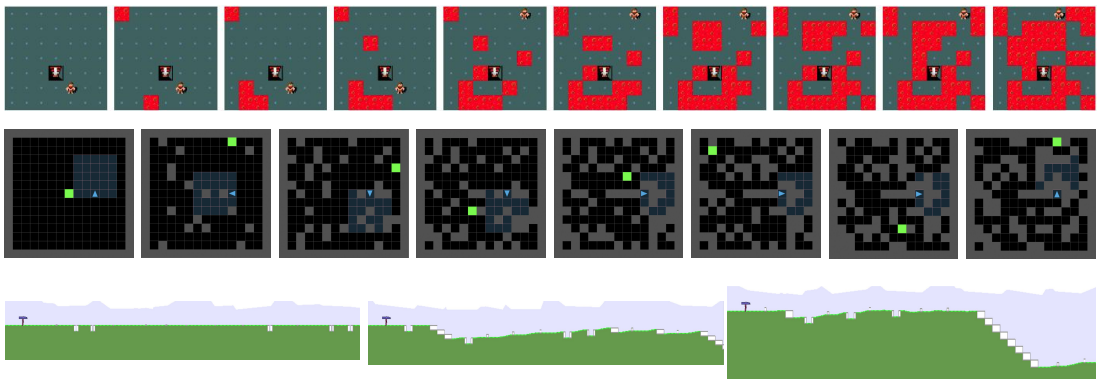


Figure 4.1: The evolution of a level in three different environments: MiniHack lava grids, MiniGrid mazes and BipedalWalker terrains. In each case, the far left shows a base level, acting as a parent for subsequent edited levels to the right. Each level along the evolutionary path has a high regret for the student agent at that point in time. Thus the level difficulty co-evolves with the agent's capabilities. In each environment, we see that despite starting with simple levels, the pursuit of high regret leads to increasingly complex challenges. This complexity emerges entirely without relying on any environment-specific exploration heuristics. Note that since the agent can move diagonally in the lava environment, the final level is solvable.

show that evolving levels can effectively produce agents capable of solving a diverse range of challenging tasks. In contrast to prior UED works, these evolutionary methods directly take advantage of the most useful structures found so far in a constant process of mutation and selection. However, the key drawback of these methods is the reliance on domain specific heuristics, while also requiring vast computational resources, making it challenging for the community to make further progress in this direction.

In this work, we seek to harness the power and potential open-endedness of evolution in a principled regret-based curriculum. We introduce a new algorithm, called *Adversarially Compounding Complexity by Editing Levels*, or ACCEL. ACCEL evolves a curriculum by making small *edits* (e.g. mutations) to previously high-regret levels, thus constantly producing new levels at the frontier of the student agent's capabilities (see: Figure 4.4). Levels generated by ACCEL begin simple but quickly

become more complex. This benefits both the beginning of training [23, 262], where the student begins learning much faster, while rapidly co-evolving the policy with the environment to solve increasingly complex levels (see Figure 4.1).

We believe ACCEL provides the best of both worlds: an evolutionary approach that can generate increasingly complex environments, combined with a regret-based curator that reduces the need for domain-specific heuristics and provides theoretical robustness guarantees in equilibrium. ACCEL leads to strong empirical gains in both sparse navigation tasks and a 2D bipedal locomotion task over challenging terrain. In both domains, ACCEL demonstrates the ability to rapidly increases level complexity while producing highly capable agents. ACCEL produces and solves highly challenging levels with a fraction of the compute of previous approaches, reaching comparable performance to POET while training on less than 0.05% of the total number of samples on a single GPU.

4.2 Related Work

The inability of deep RL agents to reliably generalize has drawn considerable attention [1, 44, 84, 111, 214, 339], with policies often failing to adapt to changes in the observation [276], dynamics [17], or reward [337]. In this work, we seek to provide agents with a set of training levels to produce a policy that is robust to these variations.

In particular, we focus on the *Unsupervised Environment Design* (UED, Dennis *et al.* [54]) paradigm, which aims to design environment directly, such that they induce experiences that result in learning the more robust policies. Domain Randomization (DR, Jakobi [117] and Sadeghi & Levine [254]) can be viewed as the most basic form of UED. DR has been particularly successful in areas such as robotics [7, 118, 209, 298], with extensions proposing to actively update the DR distribution [187, 244]. This chapter directly extends *Prioritized Level Replay* (PLR, Jiang *et al.* [121]), a method for curating DR levels such that those with high learning potential can be revisited by the student agent for training. PLR is related to TSCL [183], self-paced learning [62, 138], and ALP-GMM [233], which seek to

maintain and update distributions over environment parameterizations based on the recent performance of the agent. Recently, a method similar to PLR was shown to be capable of producing generally capable agents in a simulated game world with a smooth space of levels [296].

Dennis *et al.* [54] first formalized UED and introduced the PAIRED algorithm, a minimax regret [258] UED algorithm whereby an environment adversary learns to present levels that maximize regret, approximated as the difference in performance between the main student agent and a second agent. PAIRED produces agents with improved zero-shot transfer to unseen environments and has been extended to train agents that can robustly navigate websites [92]. Adversarial objectives have also been considered in robotics [230] and in directly searching for situations in which the agent sees the weakest performance [66]. POET [315, 316] considers co-evolving a *population* of minimax environments and agents that solve them. We take inspiration from the evolutionary nature of POET but train a *single agent*, which is beneficial as it takes significantly fewer resources, while also eliminating the agent selection problem.

UED is inherently related to the field of *Automatic Curriculum Learning* (ACL, Baranes & Oudeyer [18], Florensa *et al.* [74], and Portelas *et al.* [234]), which seeks to provide an adaptive curriculum of increasingly challenging tasks or goals given a (typically) fixed environment [5]. This setting differs from a general UPOMDP, where a task or goal specifier is not provided. Furthermore, UED approaches seek to *fully specify* environments, rather than just goals within a fixed environment. In Asymmetric Self-Play [285] one agent proposes goals for another, leading to effective curricula for challenging robotic manipulation tasks [210]. AMIGo [32] and APT-Gen [69] provide adaptive curricula for hard-exploration, goal-conditioned problems. Many ACL methods emphasize learning to reach goals or states with high uncertainty [232, 237, 343], either using generative [73] or world models [188].

Automatic environment design has also been considered in the symbolic AI community as a means to shape an agent's decisions [340, 341]. Automated environment design [129, 130] seeks to redesign specific levels to improve the

agent's performance within them. In contrast, UED adapts curricula that improves performance across levels.

Our work also relates to the field of *procedural content generation* (PCG, Justesen *et al.* [124] and Risi & Togelius [248]), which seeks to algorithmically generate diverse levels. Popular PCG environments used in RL include the Procgen Benchmark [44], MiniGrid [38], Obstacle Tower [122], GVGAI [227], and the NetHack Learning Environment [149]. This work uses the recently proposed MiniHack environment [256], which provides a flexible framework for creating diverse environments. Within the PCG community, automatically generating game levels has been of interest for more than a decade [301], with recent methods making use of machine learning [24, 286]. PCGRL [60, 131] frames level design as an RL problem, whereby the design policy makes incremental changes to the level to maximize some arbitrary objective subject to game-specific constraints. Unlike ACCEL, it makes use of hand-designed dense rewards and focuses on the design of levels for human players, rather than as an avenue to training highly-robust agents.

4.3 Methods for Unsupervised Environment Design

The work in this chapter builds primarily on three related works, which we discuss below. The first is *Paired Open-Ended Trailblazer* (POET) [315], which coevolves a population of agent-environment pairs using a minimal criterion [26]. POET uses a "stepping stone" procedure to transfer agents across environments, making it possible to learn highly complex behaviors that would not likely be discovered from scratch. One of the main drawbacks for POET is its complexity—the method requires a population of 20 agents training in parallel, with a goal switching process that involves evaluating agents on several different environments. In total the procedure uses several hundred billion timesteps. Secondly, POET requires handcrafted heuristics, since it uses an adversarial objective it pre-filters levels to be within a given reward range. If this range was removed, the adversarial objective would simply propose unsolvable levels rendering the process ineffective for learning strong policies.

In contrast to POET, Dennis *et al.* [54] proposed an approach that avoids using heuristics to select environments. Instead, they introduce an adversary that designs environments to maximize *regret*, defined as the difference in return between a new antagonist agent and the student policy (referred to as the protagonist). Their approach is called *Protagonist Antagonist Induced Regret Environment Design* or PAIRED. Since the antagonist must get a positive reward to incur positive regret, this means the PAIRED adversary proposes solvable levels. Further, since regret is highest for simpler levels, the curriculum naturally discovers the simplest set of levels the student policy cannot currently solve. Indeed, Dennis *et al.* [54] also showed theoretically that if the process reaches a Nash Equilibrium, the student policy is following a minimax regret strategy. However, PAIRED is not without flaws. In particular, PAIRED often falls into local optima, and is susceptible to cycling. It is easy to see why, the adversary has to learn from sparse rewards (regret at the end of an episode), with challenging credit assignment. For example, it is not clear which block placed by the adversary impacted the regret.

The third approach is the most simple. Rather than seek to design new levels from scratch, the *Prioritized Level Replay* (PLR, Jiang *et al.* [121]) algorithm samples random levels from the domain randomization distribution and then replays those with high learning potential. This is proxied by "value loss", taken as the sum of the per time step TD errors in an episode. Despite its simplicity, PLR was shown to provide large performance gains on the Procgen benchmark [44] by generating an automatic curriculum over level seed.

While PLR offers promising performance, it does not have the same theoretical properties of PAIRED, and it has not yet been used for large scale unsupervised environment design. In this section, we propose an extension to PLR to make both of these things possible.

4.4 Robust Prioritized Level Replay

In Jiang *et al.* [120] we introduce a new perspective on PAIRED and PLR. Rather then seeing them as distinct methods, we instead view them as instantitations of

a novel class of algorithms, which we refer to as *Dual Curriculum Design* (DCD). DCD algorithms contain both a generator (producing new levels) and a curator (replaying them). Thus, PAIRED can be seen as an regret-based generator without a curator, and PLR is a random generator with a value-loss based curator.

Before introducing new algorithms, we provide a brief overview of the theoretical foundation underpinning this new class of algorithms. We formalize DCD as a three player game between a student and two teachers (generator and curator), which we call a *dual curriculum game*. Recall from Section 2.2.4 that in UED the teacher follows a utility function U_t. With two teachers, the first teacher plays this game with probability p and the second with $(1-p)$, thus, the combined "teacher" now maximizes the joint reward of $pU_t^1 + (1-p)U_t^2$. This can be seen as a joint strategy for a single teacher. In Jiang *et al.* [120] we show that when we reach a NE of the dual curriculum game, the teachers arrive at approximate best responses for both the base game with the joint objective and with their own objectives, meaning they are also in an approximate NE of the base game with either teacher.

When looking specifically at PLR, we have $U_t^1 = C$ since the first teacher is following a domain randomization strategy, and U_t^2 is seeking to maximize value loss. The first change we make is to switch the second teacher (curator) objective to approximate *regret*, U_t^R. One way to do this is to change from using the value loss (which captures both upside and downside surprises) to the *Positive Value Loss*, defined as follows:

$$U_t^R = \frac{1}{T}\sum_{t=0}^{T} \max\left(\sum_{k=t}^{T}(\gamma\lambda)^{k-t}\delta_k, 0\right) \tag{4.1}$$

where λ and γ are the Generalized Advantage Estimation (GAE, Schulman *et al.* [266]) and MDP discount factors respectively, and δ_t, the TD-error at timestep t.

With this change, by Corollary 1 of Jiang *et al.* [120] we see that PLR can achieve theoretical guarantees at NE, in a similar vein to PAIRED. However, the guarantees are weaker. Concretely, we have the new PLR instantiation of a dual curriculum game then the first teacher maximizes regret $U_t^1 = U_t^R$ and the second teacher plays randomly $U_t^2 = U_t^U$. Recall that $\xi \in \mathbb{R}^d$ corresponds to a

d-dimensional vector encoding the environment parameters for a UPOMDP (see Section 2.2.4). Let $V^{\xi}(\xi)$ be bounded in $[B^-, B^+]$ for all ξ, π. Then the student policy π is $2(B^+ - B^-)(1-p)$ close to having optimal worst-case regret (see Jiang *et al.* [120] for a more detailed discussion).

This result immediately leads to a new insight—if we set $p = 0$ then the student policy will follow a minimax regret strategy at equilibrium. Thus, counter intuitively, the student learns more effectively by training on less data, i.e. by applying a stop gradient when collecting experience on samples from the random generator and only updating the policy on curated, high regret levels. This new algorithm, which we call *Robust PLR* or PLR$^{\perp}$ is shown in Algorithm 5.

Algorithm 5 Robust PLR (PLR$^{\perp}$)

Randomly initialize policy $\pi(\phi)$ and an empty level buffer, $\mathbf{\Lambda}$ of size K.
while *not converged* **do**
  Sample replay-decision Bernoulli, $d \sim P_D(d)$
  if $d = 0$ **then**
    Sample level ξ from level generator
    Collect π's trajectory τ on ξ, with a stop-gradient $\phi_{\perp}$ *i.e. Do not update* π
  else
    Use PLR to sample a replay level from the level store, $\xi \sim \mathbf{\Lambda}$
    Collect policy trajectory τ on ξ and update π with rewards $\boldsymbol{R}(\tau)$
  end
  Compute PLR score, $S = \textbf{score}(\tau, \pi)$
  Update $\mathbf{\Lambda}$ with ξ using score S
end

4.4.1 Empirical Results

We tested Robust PLR in both a partially observable navigation environment and a continuous control CarRacing environment from OpenAI Gym [28]. For all experiments we train the student via Proximal Policy Optimization (PPO, Schulman *et al.* [267]), see the background (Section 2.1.4) for more details. Here we focus on the CarRacing domain, with navigation results presented in the next section.

This environment entails continuous control with dense rewards, a 3-dimensional action space, and partial, pixel observations, with the goal of driving a full lap

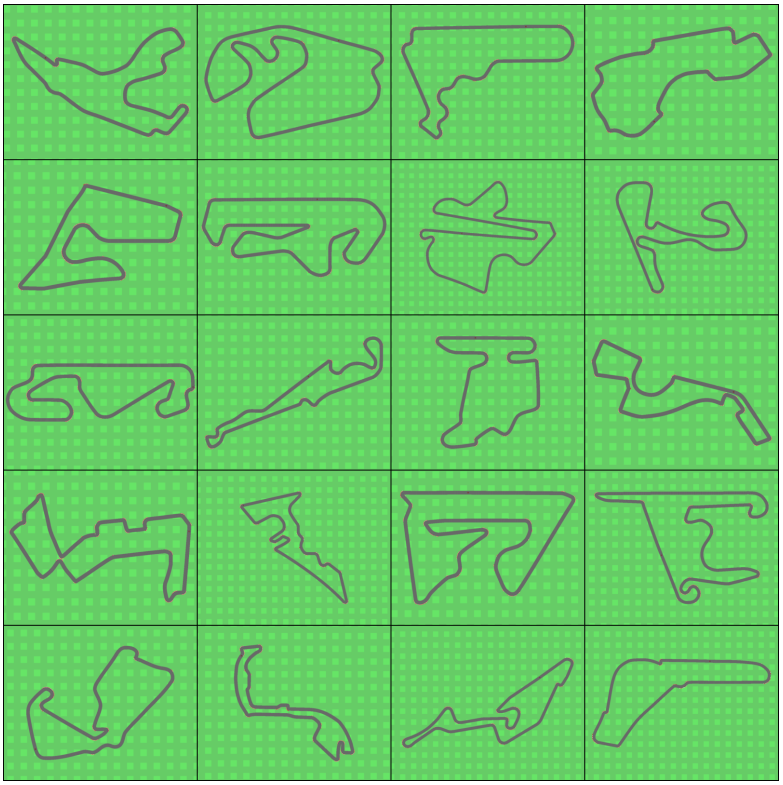


Figure 4.2: CarRacing F1 Benchmark.

around a track. To enable unsupervised environment design of any closed-loop track, we reparameterize CarRacing to generate tracks as Bézier curves [192] with arbitrary control points. The teacher generates levels by choosing a sequence of up to 12 control points, which uniquely defines a Bézier track within specific, predefined curvature constraints. After 5M steps of training, we test the zero-shot transfer performance of policies trained by each method on 20 levels replicating official human-designed Formula One (F1) tracks, shown in Figure 4.2. Note that these tracks are significantly out of distribution, as they cannot be defined with just 12 control points.

In Figure 4.3 we show the progression of zero-shot transfer performance for the original CarRacing environment, as well as three F1 tracks of varying difficulty, while also including the final performance on the full F1 benchmark. For the final performance, we also evaluated the state-of-the-art CarRacing agent from [294] on our new F1 benchmark. As we see in these results, Robust PLR (blue) provides a significant performance boost over PLR (pink), outperforming the state-of-the-

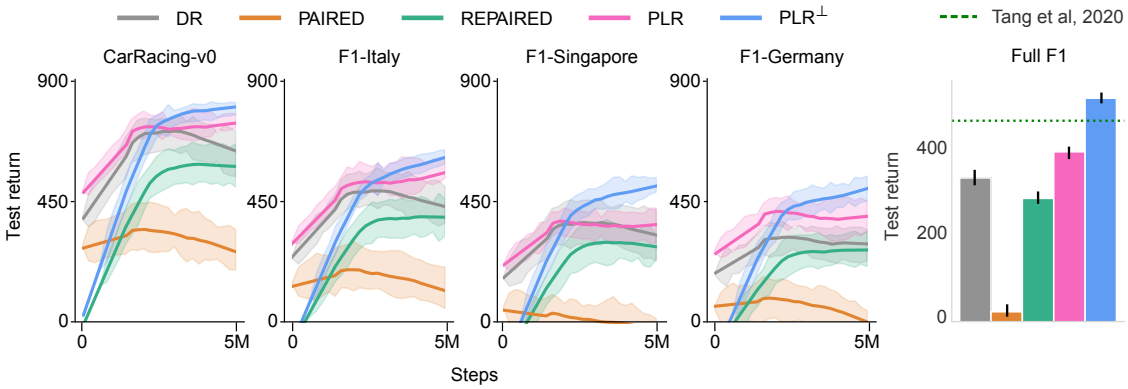


Figure 4.3: Zero-shot transfer performance. Plots show mean and standard error over 10 runs.

art Attention Agent. Interestingly, PAIRED struggles in this setting due to the challenges of designing environments using RL.

To conclude, this section, we introduced Robust PLR, a simple regret-based UED algorithm that empirically produces policies with strong generalization capabilities. However, it is clear to see this method remains limited—it can only curate randomly sampled levels. Indeed, Robust PLR's inability to build off of previously discovered levels may make it challenging to sample complex structures with the frequency required to train agents capable of solving them. Further, random search will suffer from the curse-of-dimensionality in higher-dimensional design spaces, where randomly encountering levels at the frontier of the agent's current capabilities can be highly unlikely.

4.5 Adversarially Compounding Complexity by Editing Levels

As we have seen so far, a simple curation strategy can be highly effective for UED. However, it lacks creativity and the ability to build complexity from existing knowledge. In this section we draw inspiration from POET, and seek to harness the power of evolution in a regret-based curriculum. We introduce a new method, which we call *Adversarially Compounding Complexity by Editing Levels* or ACCEL [215].

4.5.1 ACCEL Algorithm

In this section we introduce a new algorithm for UED, combining an evolutionary environment generator with a principled regret-based curator. Unlike Robust PLR which relies on random sampling to produce new batches of training levels, we instead propose to make *edits* (e.g. mutations) to previously curated ones. Evolutionary methods have been effective in a variety of challenging optimization problems [235, 279], yet typically rely on handcrafted, domain-specific rules. For example, POET manually filters BipedalWalker levels to have a return in the range $[50, 300]$. The key insight in this work is that regret serves as a domain-agnostic fitness function for evolution, making it possible to consistently produce batches of levels at the frontier of agent capabilities across domains. Indeed, by iteratively editing and curating the resulting levels, the levels in the level replay buffer quickly increase in complexity. As such, we call our method *Adversarially Compounding Complexity by Editing Levels*, or ACCEL.

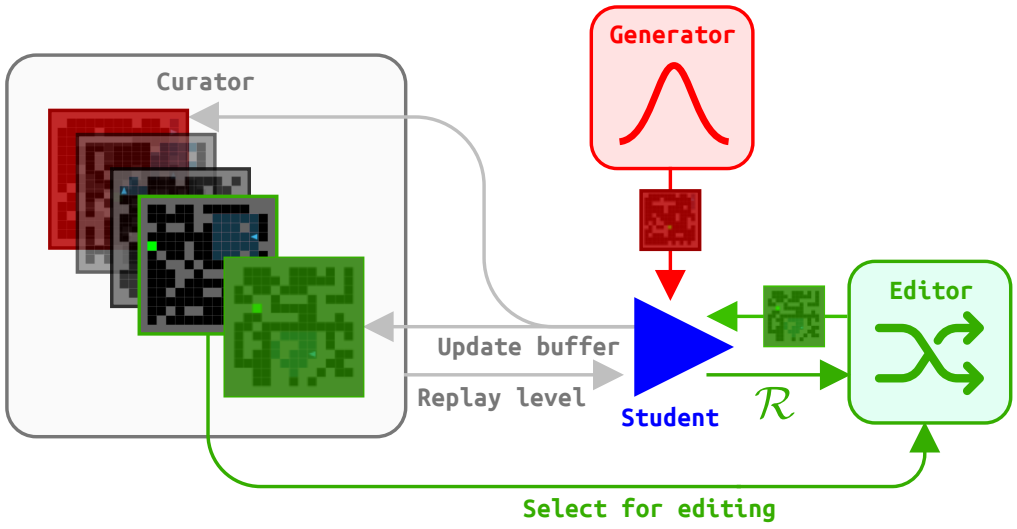


Figure 4.4: An overview of ACCEL. Levels are randomly sampled from a generator and evaluated, with high-regret levels added to the level replay buffer. The curator selects levels to replay, and the student only trains on replay levels. After training, the replayed levels are edited and evaluated again to compute regret for level replay.

ACCEL does not rely on a specific editing mechanism, which could be any mutation process used in other open-ended evolutionary approaches [277]. In this

paper, editing involves making a handful of changes (e.g. adding or removing obstacles in a maze), which can operate directly on environment elements within the level or on a more indirect encoding such as the latent-space representation of the level under a generative model of the environment. In general, editing may rely on more advanced mechanisms, such as search-based methods, but in this work we predominantly make use of simple, random mutations. ACCEL makes the key assumption that regret varies smoothly with the environment parameters Ξ, such that small mutations in the levels lead to equally small changes in regret. If this is the case, then small edits to a single high-regret level should lead to the discovery of entire batches of high-regret levels—which could be an otherwise challenging task in high-dimensional design spaces.

Following Robust PLR, as described in the previous section, we do not immediately train on edited levels. Instead, we first evaluate them and only add them to the level replay buffer if they have high regret, estimated by positive value loss (Equation 4.1). We consider two different criteria for selecting which replayed levels to edit: Under the "easy" criterion, we edit those levels which the agent can currently solve with low regret, approximated as the agent's return minus its regret. Under the "batch" criterion, we simply edit the entire batch of replayed levels. The full procedure is shown in Algorithm 6.

ACCEL can be seen as a UED algorithm taking a step toward open-ended evolution [281], where the fitness is approximate regret, as levels only stay in the population (that is, the level replay buffer) if they meet the high-regret criterion for curation. However, ACCEL avoids some important weaknesses of evolutionary algorithms such as POET: First, ACCEL maintains a population of levels, but not a population of agents. Thus, ACCEL requires only a single desktop GPU for training. In contrast, evolutionary approaches typically require a CPU cluster. Moreoever, forgoing an agent population allows ACCEL to avoid the agent selection problem. Instead, ACCEL directly trains a single generally-capable agent. Finally, since ACCEL uses a minimax *regret* objective (rather than minimax adversarial as

Algorithm 6 Adversarially Compounding Complexity by Editing Levels (ACCEL)

Input: Level buffer size K, initial fill ratio ρ, level generator
Initialize: Initialize policy $\pi(\phi)$, level buffer $\mathbf{\Lambda}$
Sample $K * \rho$ initial levels to populate $\mathbf{\Lambda}$
while *not converged* **do**
  Sample replay decision $d \sim P_D(d)$
  if $d = 0$ **then**
    Sample level ξ from level generator
    Collect π's trajectory τ on ξ, with stop-gradient $\phi_\perp$
    Compute regret score S for ξ (Equation 4.1)
    Update $\mathbf{\Lambda}$ with ξ if score S meets threshold
  else
    Sample a replay level, $\xi \sim \mathbf{\Lambda}$
    Collect policy trajectory τ on ξ
    Update π with rewards $\boldsymbol{R}(\tau)$
    Edit ξ to produce ξ'
    Collect π's trajectory τ on ξ', with stop-gradient $\phi_\perp$
    Compute regret score S (S') for ξ (ξ')
    Update $\mathbf{\Lambda}$ with ξ (ξ') if score S (S') meets threshold
    (Optionally) Update Editor using score S
  end
end

in POET), it naturally promotes solvable levels at the frontier of agent's capabilities, without relying on domain-specific knowledge (such as reward ranges).

Note that just like Robust PLR, ACCEL can also be seen as an instatiation of Dual Curriculum Design. Once again, the first teacher is playing a regret-based strategy, but now the second teacher is combining a random and evolutionary approach. However, since the student policy only trains on levels from the curator (the regret-based teacher), ACCEL inherits the robustness guarantees in equilbrium from Robust PLR:

Remark 1. *If ACCEL reaches a Nash equilibrium, then the student follows a minimax regret strategy.*

In stark contrast, other evolutionary approaches primarily justify their applicability solely via empirical results on specific domains. As we will show next, a key strength of ACCEL is its generality, as it can produce highly capable agents in a diverse range of environments, without domain knowledge.

4.5.2 Experiments

In our experiments we seek to compare agents trained with ACCEL with several of the best-performing UED baselines. In all cases, we train a student agent via PPO. To evaluate the quality of the resulting curricula, we show all performance with respect to the number of gradient updates for the student policy, as opposed to total number of environment interactions, which is, in any case, often comparable for PLR and ACCEL (see Table B.6). For a full list of hyperparameters for each experiment please see Table B.7 in Section B.3.3. Our primary baseline is Robust PLR [120], which combines the random generator with a regret-based curation mechanism. Note that we will simply use "PLR" to refer to the robust form of the algorithm from here onwards. The other baselines are domain randomization (DR), PAIRED [54], and a minimax adversarial teacher. The minimax baseline corresponds to the objective used in POET without the hand-coded constraints. We leave the comparison to population-based methods to future work due to the computational expense required. We report results in a consistent manner across environments—in each case, we show the emergent complexity during training and report test performance with the aggregate inter-quartile mean (IQM) and optimality gap using the recently introduced `rliable` library [2].

We begin with a partially-observable navigation environment, where we test our agents on challenging out-of-distribution environments. Next, we compare each method on the continuous-control environment from Wang *et al.* [315], featuring a highly challenging distribution of training levels that requires the agent to master multiple behaviors to achieve strong performance.

Partially Observable Navigation We begin with a maze navigation environment based on MiniGrid [38], originally introduced in Dennis *et al.* [54]. Despite being a conceptually simple environment, training robust agents in this domain requires a large-scale experiment: Our agents train for 20k updates (≈350M steps, see Table B.6), learning an LSTM-based policy with a 147-dimensional partially-observable observation. Our DR baseline samples between 0 and 60 blocks to

place, providing a sufficient range for PLR to form a curriculum. For ACCEL we begin with empty rooms and randomly edit the block locations (by adding or removing blocks), as well as the goal location. After replay, we edit levels selected via the "easy" criterion—essentially moving levels back to the learning frontier once their learning potential has been reduced. In Figure 4.5, we report training performance and complexity metrics. We see that ACCEL rapidly compounds complexity, leading to training levels with significantly higher block count and longer solution paths than other methods.

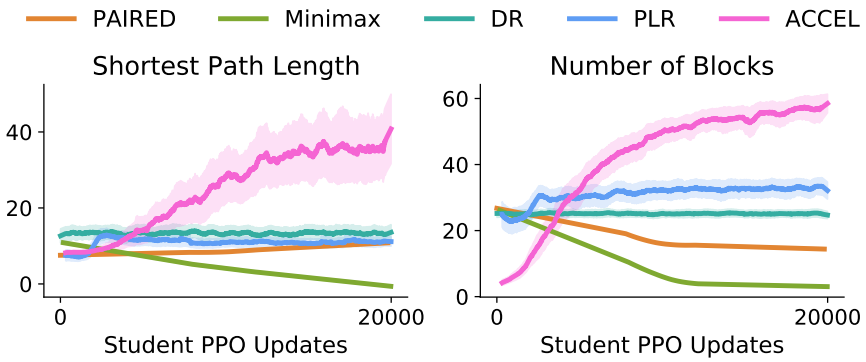


Figure 4.5: Emergent complexity metrics for maze navigation levels during training. Plots show the mean and standard error across five training seeds.

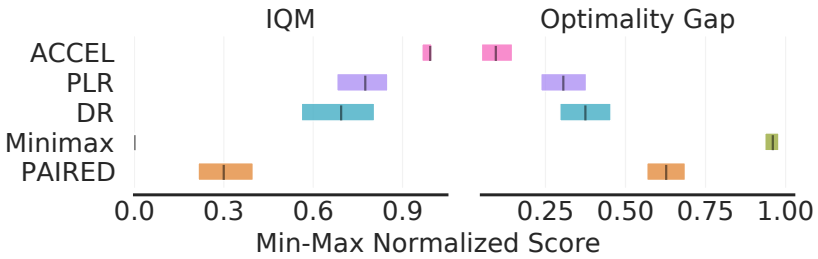


Figure 4.6: Maze navigation aggregate test performance. Plots show the inter-quartile mean (IQM) and optimality gap, with 95% confidence intervals shaded.

We evaluate the zero-shot transfer performance of each method on a series of held-out test environments, as done in prior works. For DR, PLR, and ACCEL, evaluation occurs after 20k student PPO updates, focusing the comparison on the effect of the curriculum. The minimax and PAIRED results are those reported in Jiang *et al.* [120] at 250M training steps (≈30k updates). As we see, ACCEL performs at least

as well as the next best method in almost all test environments, with particularly strong performance in Labyrinth and Maze. As reported in Figure 4.6, ACCEL achieves drastically stronger performance than all other methods in aggregate across all test environments: its IQM approaches a perfect solved rate compared to below 80% for the next best method, PLR, with an 80.2% probability of improvement over PLR. In Appendix B.1 we show the individual test levels in Figure B.1, with full test results in Table B.1. Example levels generated by each method are shown in Figure 4.7, where we see ACCEL produces mazes that qualitatively appear to exhibit more structure than the baselines. Indeed, it seems highly challenging to sample solvable levels with this high block count and complexity at random, but it occurs naturally with ACCEL since complexity compounds over time, building on previously solvable levels.

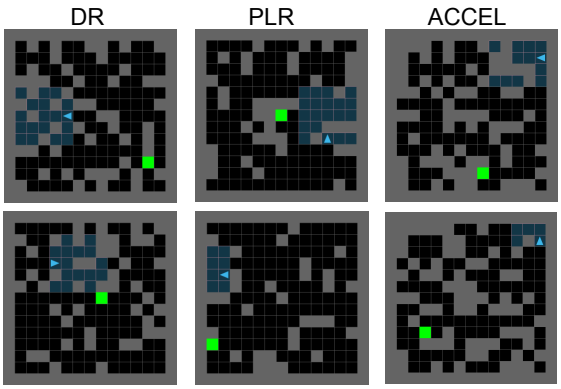


Figure 4.7: Example levels generated by DR, PLR and ACCEL.

Next, we consider an even more challenging setting—we use a larger version of "PerfectMaze", a procedurally-generated maze environment, shown in Figure 4.8. Each randomly-generated maze has 51×51 tiles, an order of magnitude larger than training levels, with a maximum episode length of over 5k steps. We evaluate each agents for each training seed for 100 episodes, using the same checkpoints used to produce Figure 4.6. The evaluation results in Figure 4.8 show ACCEL significantly outperforms all baselines, achieving a success rate of 53% compared to the next best method, PLR, which attains a success rate of 25%, while all

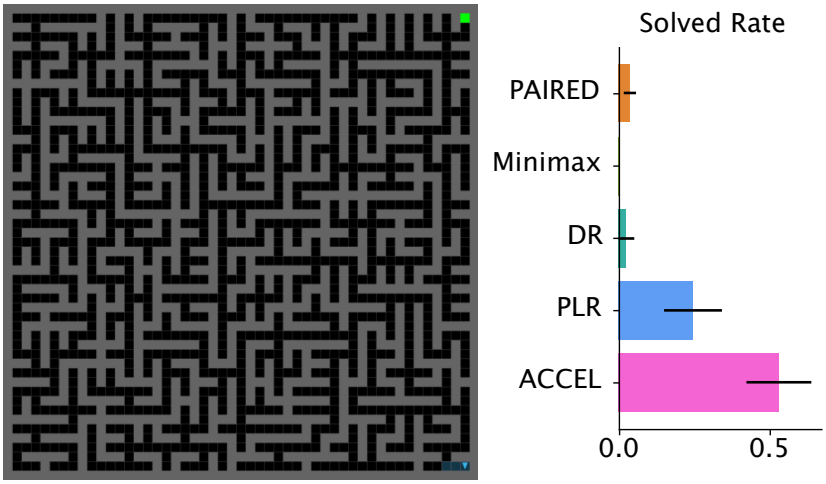


Figure 4.8: Zero-shot performance on a large procedurally-generated maze environment. The bars show mean and standard error over 5 training seeds, each evaluated over 100 episodes. ACCEL achieves over double the success rate of the next best method, despite beginning with empty rooms.

other methods fail. Notably, ACCEL learns to approximately follow the "left-hand" rule for solving single-component mazes.

We seek to understand the key drivers of ACCEL's outperformance: Incremental changes to a level can lead to a diverse batch of new ones [284], which may move those that are currently too hard or too easy towards the frontier of the agent's capabilities. This diversity may prevent overfitting. For example, in Figure 4.9, we see three edits of the same level produced by ACCEL. Each has a similar initial observation, yet requires the agent to explore in different directions to reach the goal, thereby pressuring the agent to actively explore the environment. Making edits such as these *outside* of the direct trajectory of the agent can also be seen as a form of data augmentation that changes the observation but not the optimal policy. Such data augmentations have been shown to improve sample efficiency and robustness in RL [141, 154, 240].

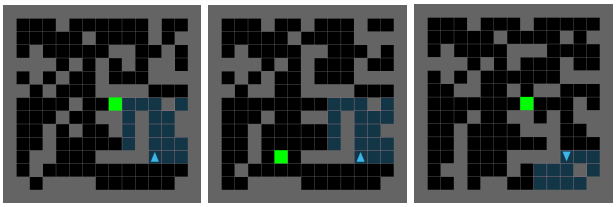


Figure 4.9: Despite sharing a common ancestor, each of these levels require different behaviors to solve. Left: the agent can go up or left and reach the goal. Middle: the goal is on the left, while on the right the left path is blocked.

Walking in Challenging Terrain Finally, we evaluate our approach on a continuous-control environment with dense rewards, using the BipedalWalker environment from Wang *et al.* [315]. As in Wang *et al.* [315] we use a modified version of the `BipedalWalkerHardcore` environment from OpenAI Gym [28]. We include all eight parameters in the design space, rather than the subset used in Wang *et al.* [315], making our setting more complex. This environment is detailed at length in Appendix B.3.1. We run all baselines from previous experiments, alongside an additional baseline, ALP-GMM [233], which was originally tested on BipedalWalker. We train agents for 30k student updates, equivalent to between 1B to 2B total environment steps, depending on the method (see Table B.6). During training we evaluate agents on both the simple `BipedalWalker` and more challenging `BipedalWalker-Hardcore` environments, in addition to four environments testing the agent's effectiveness against specific, isolated challenges otherwise present to varying degrees in training levels: {`Stump`, `PitGap`, `Stairs`, `Roughness`}, shown in Figure 4.10.

After 30k PPO updates, we conduct a more rigorous evaluation, using 100 test episodes in each test environment, and report the aggregate results in Figure 4.11 (normalized assuming a return range of {0,300}). ACCEL significantly outperforms all baselines, achieving close to 75% of optimal performance, almost three times the next best baseline (PLR). All other baselines struggle, likely due to the challenging environment design space containing a high proportion of levels that are not conducive to learning. Faced with such challenging levels, agents may learn to

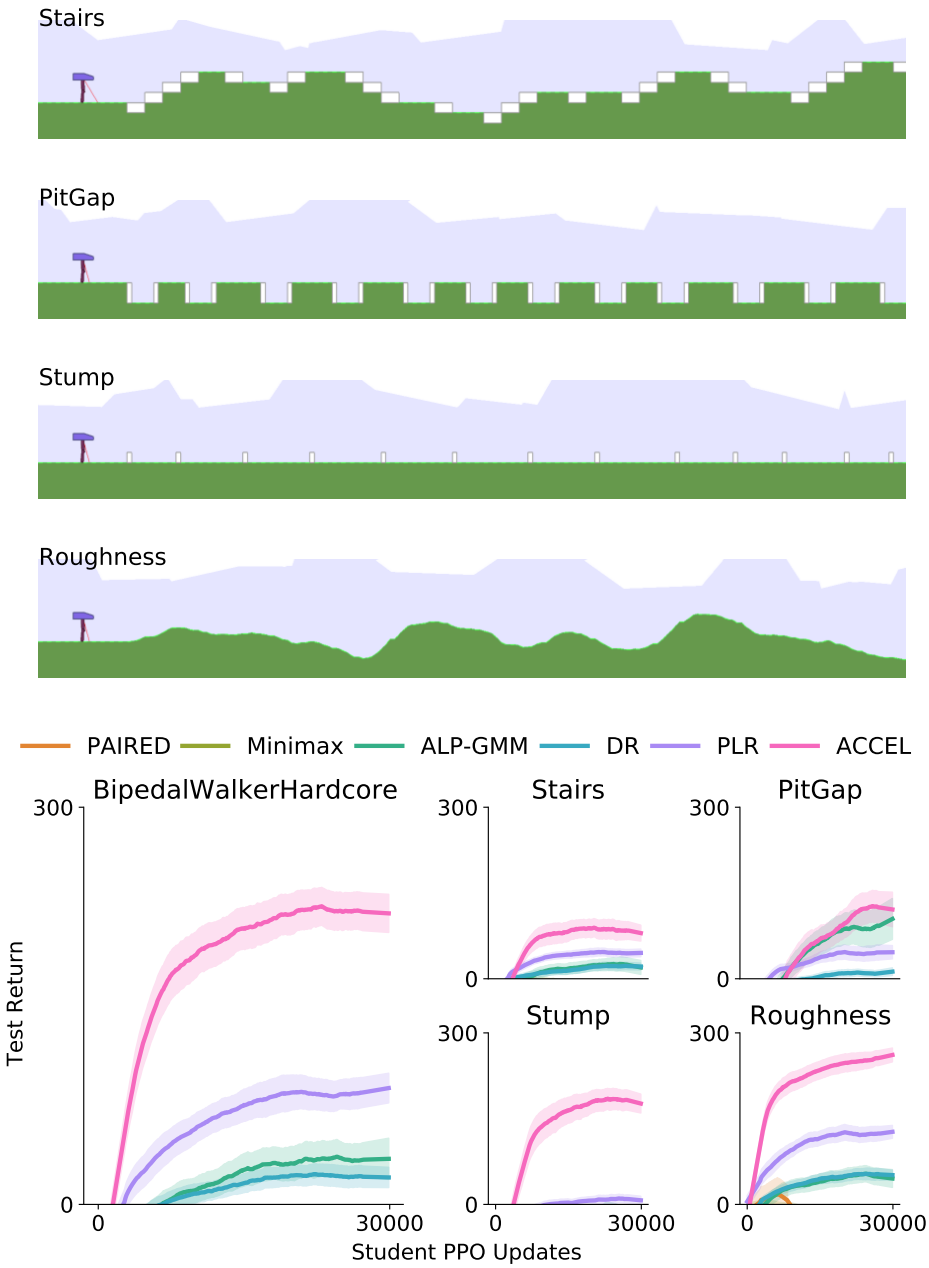


Figure 4.10: **Top**: Examples of the four individual challenges tested in BipedalWalker, **Bottom**: Performance on BipedalWalker test environments during training. Results show mean and standard error. Returns below zero are omitted.

resort to the locally optimal behavior of preventing itself from falling (avoiding a -100 penalty), rather than attempt forward locomotion. Finally, we see ALP-GMM was unable to perform well when the design space was increased from 2D (as in Portelas *et al.* [233]) to 8D.

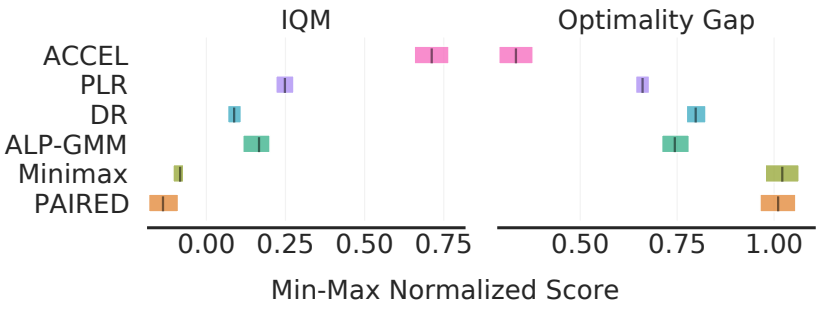


Figure 4.11: BipedalWalker aggregate test performance. Plots show the interquartile mean (IQM) and optimality gap, with 95% confidence intervals shaded.

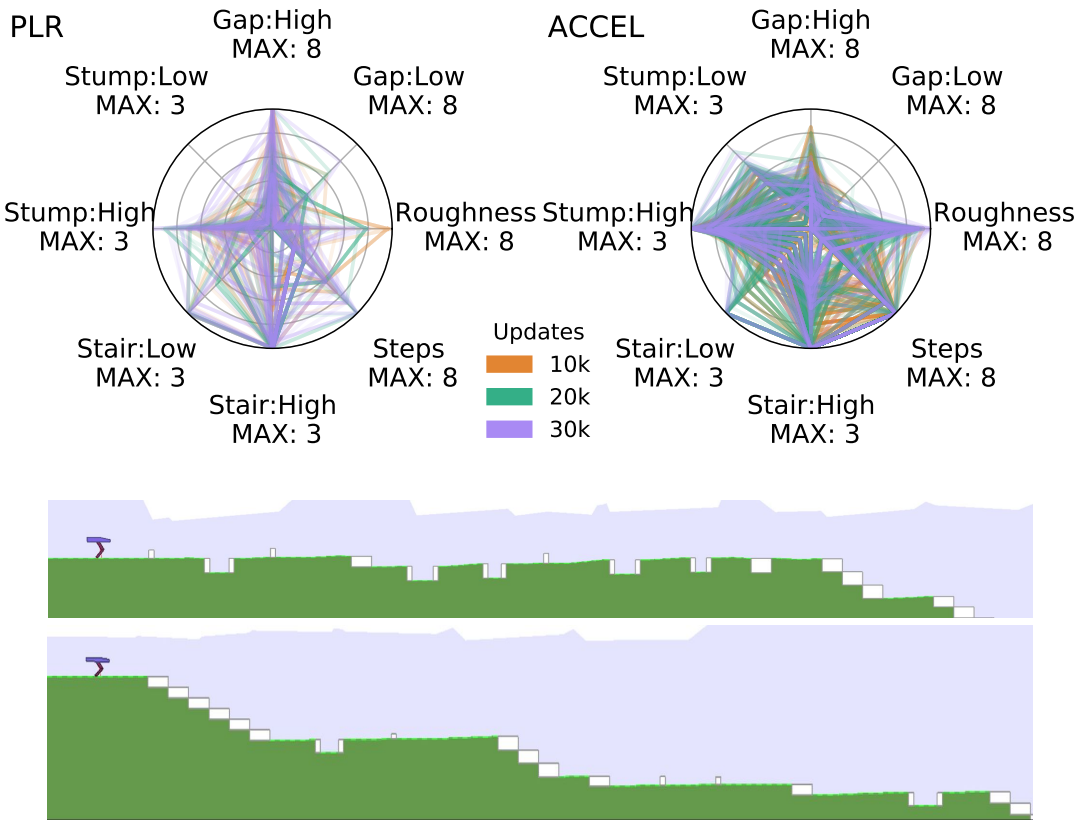


Figure 4.12: Emergent Complexity in BipedalWalker. Top: Rose plot of complexity metrics of BipedalWalker levels discovered by PLR and ACCEL. Each line represents a solved level from the associated checkpoint. All levels are among the top-100 highest regret levels for the given checkpoint. Bottom: Two levels created and solved by ACCEL.

Next we seek to understand the properties of the evolving distribution high regret levels. We include all *solved levels* from the top 100 regret levels after 10k, 20k and 30k student updates. For each level we show all eight parameters in Figure 4.12 (top), with the PLR agent as a comparison. As we see, ACCEL solves a vast quantity of difficult levels, of comparable difficulty with other methods such as POET, but using a fraction of the compute. For comparison, ACCEL used 2.07B timesteps to get to 30k student updates, which is less than 0.5% of what is used in Wang *et al.* [315]. In the next section we seek a more direct comparison with results from POET.

POET Comparison For a more direct comparison with POET[1] we ran an experiment using the smaller 5D BipedalWalker environment encoding used in Wang *et al.* [315]. We train each method for ten independent runs for 50k student PPO updates. For evaluating the difficulty of generated levels, we use the thresholds provided in Wang *et al.* [315], summarized in Table 4.1. A level meeting none of these thresholds is considered "Easy", while meeting one, two or three is considered "Challenging", "Very Challenging" or "Extremely Challenging" respectively.

Table 4.1: Environment encoding thresholds for 5D BipedalWalker.

Stump Height (High)	Pit Gap (High)	Ground Roughness
≥ 2.4	≥ 6	≥ 4.5

In Figure 4.13, we show the composition of the ACCEL level replay buffer during training. As we see, ACCEL produces an increasing number of "Extremely Challenging" levels as training progresses. This is a significant achievement given that Wang *et al.* [315] showed it was not possible to create levels in this category with their evolutionary curriculum, without including a complex stepping stone procedure. We thus see regret-based UED offers a computationally cheaper alternative to producing such levels. Moreover, POET explicitly encourages the environment

[1]It is important to note that the purpose of these two methods is different—POET seeks to discover a population of diverse specialists that can solve individual, extremely challenging levels, while the goal of ACCEL is to train a single generalist. We can attempt to compare these methods in terms of the complexity of levels generated, but it does not present the full picture.

parameters to reach high values through an explicit novelty bonus, whereas the complexity discovered by ACCEL is completely emergent, arising purely through the pursuit of high-regret levels.

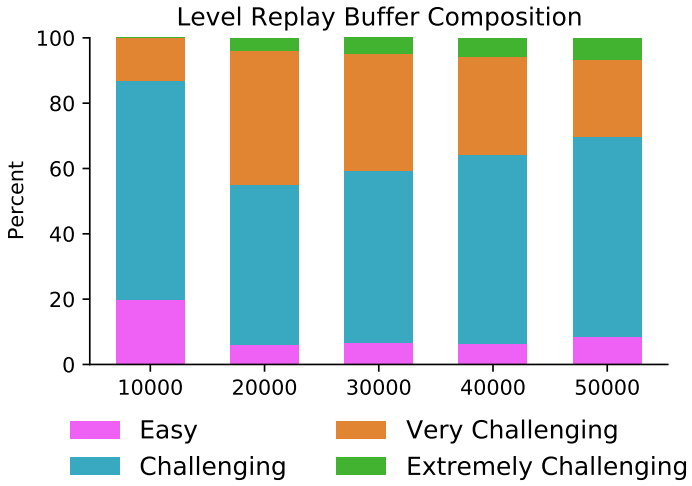


Figure 4.13: Percentage of the ACCEL level replay buffer falling into each difficulty category. Note the complexity is purely emergent in the pursuit of high-regret levels.

We further evaluate agents trained in the 5D BipedalWalker environment on the settings outlined in Figure 4.10, and report the results in Table 4.2. Note that the `Stairs` environment is now out-of-distribution, as the agent never experiences stairs during training. As we saw in the higher dimensional setting, the ACCEL agent is able to perform well across all settings, outperforming both baselines.

Table 4.2: Test solved rate after 50k updates: all ten runs of each method are evaluated on held out environments for 100 episodes. Extremely Challenging evaluations use 1000 episodes due to the diversity of the levels produced. Mean and sem shown. Bold indicates performance within a standard error of the best mean.

	PLR	ALP-GMM	ACCEL
`Stump`	0.04 ± 0.02	0.07 ± 0.02	$\mathbf{0.44 \pm 0.08}$
`PitGap`	0.2 ± 0.09	$\mathbf{0.58 \pm 0.08}$	$\mathbf{0.61 \pm 0.08}$
`Roughness`	0.23 ± 0.04	0.13 ± 0.03	$\mathbf{0.73 \pm 0.03}$
`Stairs`	0.02 ± 0.0	0.01 ± 0.0	$\mathbf{0.12 \pm 0.02}$
`Hardcore`	0.21 ± 0.04	0.2 ± 0.04	$\mathbf{0.65 \pm 0.02}$
`Extreme`	0.01 ± 0.01	0.02 ± 0.01	$\mathbf{0.12 \pm 0.02}$

Taking our evaluation one step further, we also test all methods on a held out distribution of "Extremely Challenging" levels. In this case, each time the environment is reset we resample the parameters, such that they meet all three criteria in Table 4.1. This inevitably leads to a highly diverse set of test levels. To mitigate the risk of stochasticity influencing the outcome, we evaluate each method for 1000 episodes. We compare the solved rate for each method in Table 4.2. Finally, we seek to evaluate our agents on specific levels produced by POET. We used the rose plots in [315] to create six "Extremely Challenging" environments, each solved by one of the three POET runs. Unsurprisingly our agents found these levels challenging, with low success rates. We note that this is not a perfect comparison—POET fixes the environment seed, thereby solving one level from each parameterization, while we repeatedly sample from the same distribution. However, despite this, 9 out of 10 of our independent runs solved at least one of the six environments at least once out of 100 trials per run. See the Appendix (Table B.3) for more detail on this experiment.

To summarize, we believe ACCEL is capable of producing levels of comparable complexity to POET, without a novelty bonus or handcoded heuristics, and using a fraction of the computational cost. Moreover, we train a single agent that is robust across environment challenges, while POET selects multiple agents, each tailored to individual challenges. Therefore, we believe our method produces agents that are more robust, and thus, more generally capable. That being said, there may be cases where this is not desirable—and in some settings it may be preferred to have a population of diverse specialists. This contrast presents exciting research questions which are beyond the scope of this thesis, where instead we focus on a single agent with general capability.

Do we need to start simple? To provide a better understanding of ACCEL, we conduct a simple ablation study to test the importance of the editing mechanism (E) and the inductive bias of starting simple (S). In Figure 4.14 we show the performance of three approaches: 1) sample and replay DR levels (PLR), 2) sample, replay and edit DR levels (PLR+E) and 3) sample simple levels, replay and edit

(this is ACCEL but for clarity we refer to it here as PLR+E+S). As we see, editing levels leads to improved performance, while starting simple is more important in the BipedalWalker environment.

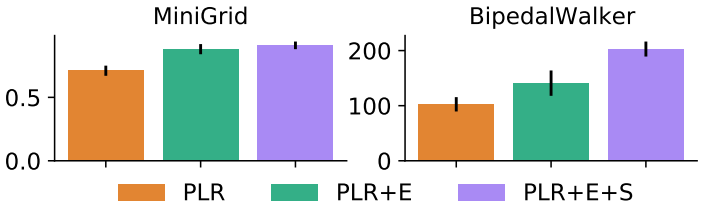


Figure 4.14: Aggregate returns for editing ablations in MiniGrid and BipedalWalker. E=editing, S=start simple. Plots show the mean and standard error.

4.6 Discussion

In our experiments we have demonstrated that first Robust PLR and then ACCEL can form highly effective curricula in challenging and diverse environments. In particular in the BipedalWalker setting we were able to match the complexity of POET using a single agent, with a small fraction of the computational cost. We thus believe we have shown evidence that ACCEL would be an effective method for training agents in more open-ended UED design spaces.

Importantly, the goal of the methods in this chapter differs from the work of Wang *et al.* [315]. The primary motivation of Robust PLR and ACCEL is to produce a single robust *generalist* agent that can solve a wide range of challenges. Regret-based curricula seek to prioritize the simplest levels that the agent cannot currently solve. In contrast, POET co-evolves agent-environment pairs in order to find *specialist* policies for solving a single highly complex level. POET's specialized agents may likely learn to solve challenging environments outside the capabilities of even ACCEL's generalist agents, but at the cost of potentially overfitting to their paired levels. Thus, unlike ACCEL, the policies produced by POET should not be expected to be robust across the full distribution of levels. The relative merits of POET and ACCEL thus highlight an important trade-off between specialization and generalization.

4. Environments that Continuously Propose New Challenges

We note that ACCEL does come with limitations. First, while the simplicity of the method is appealing, ACCEL includes an inductive bias with the ability to begin with a simple base case (e.g. an empty room). This may not always be possible in practice, as in some settings the simplest example (in terms of the entities placed in the environment) may actually be a more difficult environment to solve, for example, in a game where an agent must hide from an opponent it will actually be more challenging with fewer obstacles [14]. Finally, it is possible that larger design spaces will require additional mechanisms, such as actively encouraging diversity in the level space. In the next chapter we discuss this further, with possible future directions to yield more open-ended learning.

Part II

Achieving Open-Endedness

5
The Need for Diversity

"Insanity is doing the same thing over and over again and expecting different results."

— Attributed to a Narcotics Anonymous pamphlet

Contents

5.1 Introduction

In the previous chapters we described methods to automate both an agent and environment configuration, operating on the fly in a single training run. Combining approaches for adapting both agent and environment configurations may lead to

high performance on challenging tasks, but it may not be truly open-ended. Indeed, the coevolutionary process between the agent and environment remains inherently greedy, with environments seeking out levels that are *currently* high regret, and agent configurations being selected for *immediate* improvement. Thus, it may be susceptible to converging to one type of solution, failing to discover a diversity of levels and as a result lacking the diverse behaviors required to solve them.

The discovery of diverse solutions has long been known to play a crucial role in the evolutionary community [185]. A canonical example is improved robustness, for example work showing that an agent can adapt to damage if it possess a repertoire of gaits [50]. Indeed, if we want our systems to be open-ended, it may also be crucial to maintain "stepping stones", that may not initially be selected if purely filtering by an objective function [315, 316, 325]. Indeed, components that are crucial for future success may not be discovered if we are optimizing exclusively for reward [280]. This was demonstrated in the canonical *picbreeder* experiment [268], where humans provided the selection mechanism for images to evolve. Several interesting images emerged, each requiring intermediate steps that looked nothing like the final result. The means for producing diverse solutions varies depending on the community—in RL, it is common to use state-based representations for policies [67], in *multi-agent* RL there have been a series of works using diversity in the responses [15, 82], while in the Quality Diversity community it is common to make use of a domain specific behavior descriptor [50, 203].

In this chapter we introduce two distinct approaches for producing diverse solutions, each with its own unique trade-offs. First, in Section 5.3 we discuss *behavioral* diversity [196], which focuses on producing agents that take different actions in the same regions of the state space. This has the benefit of being data efficient since we do not need to collect additional samples from the environment, while it is also task agnostic. This approach builds directly on recent work and emphasizes the importance of jointly optimizing over an entire population as opposed to considering the pairwise distance of agents.

Next in Section 5.4 we revisit an area that has been less popular in recent times–parameter-based or "genotypic" diversity. This is typically challenging due to the challenge of searching through a high dimensional weight space. We solve this problem by introducing structure, by following different directions (or ridges) in the parameter space. This approach does not require a (human) specified behavioral objective and thus offers promise as being a general way to find diverse solutions.

Finally, we note that while the methods we propose offer a promising route to discover diverse solutions or agents, they may not be sufficient for discovering diverse environments. It will likely require additional innovations to ensure the entire coevolutionary process maintains diversity [26].

5.2 Related Work

In this chapter we focus on training diverse populations of agents, which has been an active area of research in a wide range of communities.

In evolutionary computation, forms of parameter-based (or genotypic) diversity, have been considered for several decades, for example by modifying the fitness of each individual based on proximity to others [85, 180, 185, 257, 300]. More recently, behavioral diversity has been the dominant approach [195, 196, 303, 304]. A seminal idea in this space is *novelty search*, which seeks to *only* optimize for diversity, ignoring task reward [160, 162]. This has been shown to be effective in several deceptive environments which cannot be solved by optimizing for reward (or fitness) alone.

More recently, the field of *Quality Diversity* (QD, Pugh *et al.* [235]), has risen to prominence, with a suite of methods for discovering a set of high performing diverse solutions. A key motivation for QD has been to learn a set of policies that are robust to changes in the agent or environment, for example an agent can adapt to damage if it possess a repertoire of gaits [50, 203, 297]. QD methods have also been shown to improve exploration, for example Conti *et al.* [47] combined novelty search and reward in a large-scale RL setting with deceptive rewards. Indeed, novelty search itself is one of the primary QD algorithms, especially when combined with local competition [161].

In recent times diversity has become a more prominent theme in the RL community, with similar motivation albeit differing terminology. In particular, diversity has also played a key role in aiding exploration [104, 182, 229, 344] and robustness [147, 333, 334], with additional work proposing to use diversity for unsupervised skill discovery [104].

Finally, in multi-agent RL, diversity has been crucial to explore the space of possible co-players, both in competitive [10, 15, 82, 152, 312] and cooperative games [177, 283]. As RL is being used for larger and more complex settings, the need for diversity is ever increasing, and will likely play a more prominent role in more large scale applications [312, 329].

5.3 Behavioral Diversity via Determinants

This section focuses on behavioral diversity, which seeks to encourage agents to take different actions given the same observation. Our key contribution is to introduce a new joint population update, measuring the diversity of the entire population with a formulation inspired by Determinantal Point Processes [145].

5.3.1 Task Agnostic Behavioral Embeddings

Many popular policy gradient algorithms [265, 267] consider a setting whereby a new policy $\pi_{\theta_{t+1}}$ is optimized with a constraint on the size of the update. This requirement can be considered as a constraint on the behavior of the new policy [212]. Despite the remarkable success of these algorithms, there has been little consideration for action-based behavior embeddings for novelty search methods.

Inspired by this approach, we choose to represent policies as follows:

Definition 5. *Let θ^i be a vector of neural network parameters encoding a policy π_{θ^i} and let $\mathcal{S}$ be a finite set of states. The **Behavioral Embedding** of θ^i is defined as: $\phi(\theta^i) = \{\pi_{\theta^i}(.|s)\}_{s \in \mathcal{S}}$.*

This approach allows us to represent the behavior of policies in vectorized form, as $\phi : \theta \to \mathbb{R}^l$, where $l = |a| \times N$, $|a|$ is the dimensionality of each action $a \in \mathcal{A}$ and

N is the total number of states. When N is the number of states in a finite MDP, the policies are determimistic and the embedding is as in Definition 5, we have:

$$\phi(\theta^i) = \phi(\theta^j) \iff \pi_{\theta^i} = \pi_{\theta^j}, \tag{5.1}$$

where θ^i, θ^j are vectorized parameters. In other words, the policies are the same since they always take the same action in every state of the finite MDP. Note that this does not imply that $\theta^i = \theta^j$.

5.3.2 Joint Population Update

Equipped with this notion of a behavioral embedding, we can now consider a means to compare two policies. Consider a smooth kernel k, such that $k(\mathbf{x}_1, \mathbf{x}_2) \leq 1$, for $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^d$. A popular choice of kernel is the squared exponential (SE), defined as follows:

$$k_{\text{SE}}(\mathbf{x}_1, \mathbf{x}_2) = \exp\left(-\frac{||\mathbf{x}_1 - \mathbf{x}_2||^2}{2l^2}\right), \tag{5.2}$$

for some length-scale $l > 0$. Now moving back to policy embeddings, we extend our previous analysis to the kernel (or similarity) between two embeddings, as follows:

$$k(\phi(\theta^i), \phi(\theta^j)) = 1 \iff \pi_{\theta^i} = \pi_{\theta^j} \tag{5.3}$$

We consider two policies to be distinct if $k(\phi(\theta^i), \phi(\theta^j)) = 0$. With a flexible way of measuring inter-policy similarity at hand, we can define the *population-wide* diversity as follows:

Definition 6. ***(Population Diversity)*** *Consider a finite set of M policies, parameterized by $\Theta = \{\theta^1, ..., \theta^M\}$, with $\theta^i \in \mathbb{R}^d$. We denote*

$$\text{Div}(\Theta) \stackrel{\text{def}}{=} \det(k(\phi(\theta_t^i), \phi(\theta_t^j))_{i,j=1}^M) = \det(\mathbf{K}) \tag{5.4}$$

where $k : \mathbb{R}^l \times \mathbb{R}^l \to \mathbb{R}$ is a given kernel function. Matrix $\mathbf{K}$ is positive semidefinite since all principal minors of $\det(\mathbf{K})$ are nonnegative.

This formulation is heavily inspired by Determinantal Point Processes (DPPs, [145]), a mechanism which produces diverse subsets by sampling proportionally to the determinant of the kernel matrix of points within the subset. From a geometric perspective, the determinant of the kernel matrix represents the volume of a parallelepiped spanned by feature maps corresponding to the kernel choice. We seek to maximize this volume, effectively "filling" the feature (or behavior) space.

We take a multi-objective approach [51, 52] to optimizing each agent in the population, combining *individual* agent rewards and *population-wide* diversity. Note that there are multiple ways this can be optimized, but we follow Conti *et al.* [47] and opt for the simplest formulation combining both components to form a single objective as follows:

$$J(\Theta_t) = \underbrace{\sum_{i=1}^{M} \mathbb{E}_{\tau \sim \pi_{\theta^i}}[R(\tau)]}_{\text{individual rewards}} + \underbrace{\lambda_t \text{Div}(\Theta_t)}_{\text{population diversity}} \tag{5.5}$$

where $\lambda_t \in (0, 1)$ is the trade-off between reward and diversity. This fundamentally differs from the original novelty search formulation, since we directly optimize for Θ_t rather than separately considering $\{\theta^i\}_{i=1}^M$. We call our method *Diversity via Determinants*, or DvD.

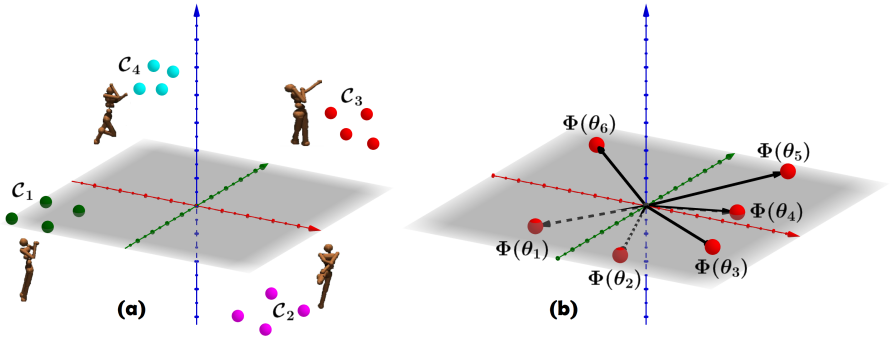


Figure 5.1: Determinant vs. pairwise distance. (a): populations of agents split into four clusters with agents within cluster discovering similar policies. (b): embedded policies $\phi(\theta_1), ..., \phi(\theta_6)$ lie in a grey hyperplane. In (a) resources within a cluster are wasted since agents discover very similar policies. In (b) all six embeddings can be described as linear combinations of embeddings of fewer canonical policies. In both settings the mean pairwise distance will be high but diversity as measured by determinants is low.

Using the determinant to quantify diversity prevents the undesirable clustering phenomenon, where a population evolves to a collection of conjugate classes. To illustrate this point, consider a simple scenario, where all M agents are partitioned into clusters. By increasing the distance between the clusters one can easily make the novelty measured as an average distance between agents' embedded policies as large as desired, but that is not true for the corresponding determinants which will be zero if the similarity matrix is low rank. Furthermore, even if all pairwise distances are large, the determinants can be still close to zero if spaces where agents' high-dimensional policy embeddings live can be accurately approximated by much lower dimensional ones. Standard methods are too crude to measure novelty in such a way (see: Fig. 5.1).

5.3.3 An Illustrative Example: Tabular MDP

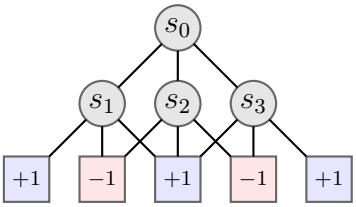


Figure 5.2: A simple MDP.

Consider the simple MDP in Fig. 5.2. There are four states $\mathcal{S} = \{s_0, s_1, s_2, s_3\}$, each has three actions, $\mathcal{A} = \{\text{-}1, 0, 1\}$ corresponding to left, down and right respectively. In addition, there are five terminal states, three of which achieve the maximum reward (+1). Let $\phi^* = \{\phi(\theta^i)\}_{i=1}^5 = \{[\text{-}1, \text{-}1], [\text{-}1, 1], [0, 0], [1, \text{-}1], [1, 1]\}$, be the set of optimal policies. If we have a population of five agents, each achieving a positive reward, the determinant of the 5×5 kernel matrix is only > 0 if the population of agents is exactly ϕ^*. This may seem trivial, but note the same is not true for the pairwise distance. If we let $d(\Theta) = \frac{1}{n} \sum_{i=1}^{n} \sum_{j>i} ||\phi(\theta^i) - \phi(\theta^j)||_2$, then ϕ^* does not maximize d. One such example which achieves a higher value would be $\phi' = \{[\text{-}1, \text{-}1], [\text{-}1, 1], [1, \text{-}1], [1, 1], [1, 1]\}$. See the following link for a colab demonstration of this example: https://bit.ly/2XAlirX.

5.3.4 Practical Algorithms

In most practical settings the state space is intractably or infinitely large. Therefore, we must sample the states $\{s_i\}_{i=1}^n$, where $n < N$, and compute the embedding as an expectation as follows:

$$\widehat{\phi}(\theta_i) = \mathbb{E}_{s\sim\mathcal{S}}[\{\pi_{\theta^i}(.|s)\}] \tag{5.6}$$

In our experiments we choose to randomly sample the states s, which corresponds to frequency weights. Other possibilities include selecting diverse ensembles of states via DPP-driven sampling or using probabilistic models to select less frequently visited states. We explore each of these options in the experiments where we show the representative power of this action-based embedding is not overly sensitive to these design choices. However, it remains a potential degree of freedom for future improvement in our method, potentially allowing a far smaller number of states to be used.

An Evolutionary Algorithm: DvD-ES

We begin with an algorithm based on Evolution Strategies (ES, Salimans *et al.* [255] and Wierstra *et al.* [323]). ES methods treat the cumulative reward of a single episode (Equation 2.1) as a blackbox function, parameterized by the policy weights. Concretely, consider $J(\pi_\theta)$ to be a blackbox function $F : \mathbb{R}^d \to \mathbb{R}$ taking as input parameters $\theta \in \mathbb{R}^d$ of a policy π_θ and outputting $R(\tau)$. Rather than take gradients of F, we can instead consider the Gaussian smoothing [200] F_σ of F, and then obtain stochastic gradients as follows:

$$\nabla F_\sigma(\theta) = \frac{1}{\sigma}\mathbb{E}_{\mathbf{g}\sim\mathcal{N}(0,\mathbf{I}_d)}[F(\theta+\sigma\mathbf{g})\mathbf{g}]. \tag{5.7}$$

where $\sigma > 0$ is a hyperparameter quantifying the smoothing level. This equation leads to several Monte Carlo gradient estimators that can be used for practical ES algorithms (see: Choromanski *et al.* [39] for several examples).

Extending the single policy update of ES to a population, we now simultaneously perturb an entire set of M policies $\Theta_t = \{\theta_t^i\}_{i=1}^M$, with rewards computed locally

and diversity computed globally. These two objectives are combined to produce a blackbox function with respect to Θ_t. At every iteration we sample Mk Gaussian perturbation vectors $\{\mathbf{g}_i^m\}_{i=1,...,k}^{m=1,...,M}$. We use two partitionings of this Mk-element subset that illustrate our dual objective, high local rewards and large global diversity. The first partitioning assigns to the m^{th} cpu worker a set $\{\mathbf{g}_1^m, ..., \mathbf{g}_k^m\}$. These are the perturbations used by the worker to compute its local rewards. The second partitioning splits $\{\mathbf{g}_i^m\}_{i=1,...,k}^{m=1,...,M}$ into subsets: $\mathcal{D}_i = \{\mathbf{g}_i^1, ..., \mathbf{g}_i^M\}$. Instead of measuring the contribution of an individual $\mathbf{g}_i^m$ to the diversity, we measure the contribution of the entire $\mathcal{D}_i$. This motivates the following definition of diversity:

$$\text{Div}_t(i) = \text{Div}_t(\theta_t^1 + \mathbf{g}_i^1, ..., \theta_t^M + \mathbf{g}_i^M). \tag{5.8}$$

Thus, the DvD-ES gradient update is the following:

$$\theta_{t+1}^m = \theta_t^m + \frac{\eta}{k\sigma} \sum_{i=1}^{k} [(1 - \lambda_t) R_i^m + \lambda_t \text{Div}_t(i)] \mathbf{g}_i^m. \tag{5.9}$$

where $\sigma > 0$ is the smoothing parameter [200, 255], k is the number of Monte Carlo samples used to compute the gradient, η is the learning rate, and the embeddings are computed by sampling states from the most recent iteration.

An Off-Policy RL Algorithm: DvD-TD3

It is also possible to compute analytic gradients for the diversity term in Equation 5.5. This means we can update policies with respect to the joint diversity using automatic differentiation.

Lemma 7. *The gradient of* $\log(\det(\mathbf{K}))$ *with respect to* $\mathbf{\Theta} = \theta^1, \cdots, \theta^M$ *equals:* $\nabla_\theta \log(\det(\mathbf{K})) = (\nabla_\theta \Phi(\theta))(\nabla_\Phi \mathbf{K})\mathbf{K}^{-1}$, *where* $\phi(\theta) = \phi(\theta^1) \cdots \phi(\theta^M)$.

The proof of this lemma is in the Appendix, Section C.1. Inspired by [123], we introduce DvD-TD3 [80], using multiple policies to collect data for a shared replay buffer. This is done by dividing the total data collection by the number of policies. When optimizing the policies, we use an augmented loss function, as in Equation 5.5, and make use of the samples in the existing batch for the embedding.

5.3.5 Discussion

In this section we introduced DvD, a novel approach for computing the behavioral diversity of a population of policies. DvD represents behaviors as the actions taken for a set of states (off policy) and computes a population-wide similarity (or kernel) matrix. The population diversity is then the determinant of this matrix. We show in a toy problem that DvD can reduce redundancy in a population and introduce two practical algorithms for using these ideas at larger scale, with experiments included in [218].

While this approach benefits from tractability, it may often lead to redundancy, particularly in large action spaces. Future work may consider learning dimensions of the action space to prioritize for diversity, for example by finding impactful actions [241]. Alternatively, we may seek to be diverse in a more meaningful space, for example we could attempt to learn the behavioral embeddings [49], or seek to make use of language as a summary of agent trajectories [197, 293].

5.4 Genotypic Diversity by Riding Ridges

In the previous section we introduced a new approach for measuring and subsequently rewarding diverse behaviors. The form of this objective is commonly used in RL, whereby reward is augmented by an additional loss to produce a multi-objective loss function. However, while many works have successfully relied on this formation, one must consider *Goodhart's law*: "When a measure becomes a target, it ceases to be a good measure." [282]. For example, consider the famous case of a deep network that achieved high accuracy in detecting pneumonia from X-ray scans. When it was applied in new, unseen hospitals it was ineffective, as it had learned to make use of hospital-specific tags in the training images to minimize the loss [336]. This is a specific case of a "shortcut solution" [83], which often occurs in deep learning since we make use of high capacity models and simply seek for them to minimize a single number, regardless of the mechanism to do so. Thus, it

may be possible that simply changing the objective may not give us the desired outcome—qualitatively diverse, high quality solutions.

In this section we take a step towards addressing such issues using parameter-based (or genotypic) diversity, making it applicable across modalities. One might imagine a plausible approach to finding different minima of the loss landscape would be to initialize gradient descent near a saddle point of the loss in multiple replicates, and hope that it descends the loss surface in different directions of negative curvature. Unfortunately, from any position near a saddle point, gradient descent will always curve towards the direction of most negative curvature (see Appendix C.2.1), and there may be many symmetric directions of high curvature.

Instead, we first seek to find a saddle point and then subsequently *force* different replicates to follow distinct, orthogonal directions of negative curvature by iteratively following each of the eigenvectors of the Hessian until we can no longer reduce the loss, at which point we repeat the branching process. Repeating this process hopefully leads the replicates to minima corresponding to distinct convex subspaces of the parameter space, essentially converting an optimization problem into search [153]. We refer to our procedure as *Ridge Rider* (RR). RR is less "greedy" than standard SGD methods, with Empirical Risk Minimization (ERM, [309]), which will always take the most locally loss reducing step. This greediness in SGD stems from it following the path with highest expected local reduction in the loss. Consequently, some minima, which might actually represent the solutions we seek, are never found.

5.4.1 The Ridge Rider Method

Throughout this section we assume a smooth, *i.e.*, infinitely differentiable, loss function $\mathcal{L}_\theta = \mathbb{E}_{\mathbf{x}_i} L_\theta(\mathbf{x}_i)$, where $\theta \in \mathbb{R}^n = \Theta$ are typically the weights of a DNN. In the supervised setting, $\mathbf{x}_i = \{x_i, y_i\}$ is an input and label for a training data point, while $L_\theta(\mathbf{x}_i)$ could be the cross-entropy between the true label y_i and the prediction $f_\theta(x_i)$.

We use $\nabla_\theta \mathcal{L}$ for the gradient and H for the Hessian, ie. $H = \nabla^2_\theta \mathcal{L}$, representing the matrix of second order derivatives. The eigenvalues (EVals) and eigenvectors

(EVecs) of H are λ_i and e_i respectively. The computation of the full Hessian is prohibitively expensive for all but the smallest models, so we assume that an automatic differentiation library is available from which vector-Hessian products, Hv, can be computed efficiently [221]:

$$Hv = \nabla_\theta\Big((\nabla_\theta \mathcal{L})v\Big).$$

We begin with a description of symmetries and invariances, before moving to the intuition behind RR where we present both exact and approximate (scalable) algorithms. Indeed, real world problems commonly contain symmetries and invariances. For example, in coordination games, the payout is unchanged if all players jointly update their strategy to another equilibrium with the same payout. We formalize this as follows: A symmetry, ϕ, of the loss function is a bijection on the parameter space such that $\mathcal{L}_\theta = \mathcal{L}_{\phi(\theta)}$, for all $\theta \in \Theta$.

If there are N non-overlapping sets of m parameters each, $k_i = \{\theta_{k_i^1}, ..\theta_{k_i^m}\}, i \in \{1, ...N\}$ and the loss function is invariant under all permutations of these N sets, we call these sets "N-equivalent". In a card-game which is invariant to color/suit, this corresponds to permuting both the color-dependent part of the input layer and the output layer of the DNN simultaneously for all players.

The goal of RR is as follows. Given a smooth loss function, $\mathcal{L}(\theta)$, discover qualitatively different local minima, θ^*, while grouping together those that are equivalent or, alternatively, only exploring one of each of these equivalent policies.

While symmetries are problem specific, in general *equivalent parameter sets* lead to repeated eigenvalues (EVals). If a given loss function in $\mathbb{R}^n$ has N-equivalent parameter sets ($N > 2$) of size m and a given θ is invariant under all the associated permutations, then the Hessian at θ has at most $n - m(N-2)$ distinct eigenvalues (proof in Appendix C.2.6). Thus, rather than having to explore up to n orthogonal directions, in some settings it is sufficient to explore one member of each of the groups of distinct EVals to obtain different non-symmetry-equivalent solutions. Further, the eigenvectors (EVecs) can be ordered by the corresponding EVal, providing a numbering scheme for the classes of solution.

5. The Need for Diversity

To ensure that there is at least one negative EVal and that all negative curvature directions are locally loss-reducing, we start at or near a strict saddle point θ^{MIS}. For example, in supervised settings, we can accomplish this by initializing the network near zero. In RL, we wish to begin at a point that facilitates exploration, thus we optimize θ^{MIS} to maximize entropy in the state distribution. We refer to this point as the Maximally Invariant Saddle (MIS), but note that in high dimensional (e.g. deep RL) settings it may be challenging to find. Formally,

$$\theta^{\text{MIS}} = \arg\min_{\theta} |\nabla_\theta \mathcal{L}(\theta)|, \text{ s.t. } \phi(\theta) = \theta, \ \forall \phi,$$

for all symmetry maps ϕ.

In tabular RL problems the MIS can be obtained by optimizing the following objective (Proof in Appendix C.2.7):

$$\theta^{\text{MIS}} = \arg\min_{\theta} |\nabla_\theta \mathcal{L}(\theta)| - \lambda \mathcal{H}(\pi_\theta(\mathbf{a})), \lambda > 0$$

where $\mathcal{H}$ corresponds to the entropy of the policy. From θ^{MIS}, RR proceeds as follows: We branch to create replicas, which are each updated in the direction of a different EVec of the Hessian (which we refer to as "ridges"). While this is locally loss reducing, a single step typically does not solve a problem. Therefore, at all future timesteps, rather than choosing a new EVec, each replicate follows the updated version of its original ridge until a break-condition is met, at which point the branching process is repeated. For any ridge this procedure is repeated until a locally convex region without any negative curvature is found, at which point gradient descent could be performed to find a local optimum or saddle. In the latter case, we can in principle apply RR again starting at the saddle (although we did not find this to be necessary in our setting). We note that RR effectively turns a continuous optimization problem, over θ, into a discrete search process, *i.e.*, which ridge to explore in what order.

During RR we keep track of a *fingerprint*, Ψ, containing the indices of the EVecs chosen at each of the preceding branching points. Ψ uniquely describes θ^Ψ up

to repeated EVals. In Algorithm 7 we show pseudocode for RR, the functions it takes as input are described in the next paragraphs.

Algorithm 7 Ridge Rider

Input: θ^{MIS}, α, ChooseFromArchive, UpdateRidge, EndRide, GetRidges
 # Initialize Archive of Solutions
Archive $A = [\{\theta^{\Psi=[i]}, e_i, \lambda_i\}$ for $i, e_i, \lambda_i \in \text{GetRidges}(\theta^{\text{MIS}})]$
while $|A| > 0$ **do**
  # Select a Ridge from the Archive
  $\{e_i, \theta_0^\Psi, \lambda_i\}, A = \text{ChooseFromArchive}(A)$
  while True **do**
    # Step along the Ridge with learning rate α
    $\theta_t^\Psi = \theta_{t-1}^\Psi - \alpha e_i$
    # Get updated Ridge
    $e_i, \lambda_i = \text{UpdateRidge}(\theta_t^\Psi, e_i, \lambda_i)$
    # Check Break Condition
    if $\text{EndRide}(\theta_t^\Psi, e_i, \lambda_i)$ **then**
      break
    end if
  end while
  # Add new Ridges to Archive
  $A = A \cup \left[\{\theta^{\Psi.\text{append}(i)}, e_i, \lambda_i\}$ for $i, e_i, \lambda_i \in \text{GetRidges}(\theta^\Psi)\right]$
end while

UpdateRidge computes the updated version of e_i at the new parameters. The EVec e_i is a continuous function of H ([127], pg 110-111).[1] This allows us to "follow" $e_i(\theta_t^\Psi)$ (our "ridge") for a number of steps even though it is changing. In the exact version of RR, we recompute the spectrum of the Hessian after every parameter update, find the EVec with greatest overlap, and then step along this updated direction. While this updated EVec might no longer correspond to the i-th EVal, we maintain the subscript i to index the ridge. The dependency of $e_i(\theta)$ and $\lambda_i(\theta)$ on θ is entirely implicit: $H(\theta)e_i(\theta) = \lambda_i(\theta)e_i(\theta)$, $|e_i| = 1$.

EndRide is a heuristic that determines how long we follow a given ridge for. For example, this can consider whether the curvature is still negative, the loss is decreasing and other factors.

[1] We rely on the fact that the EVecs are continuous functions of θ; this follows from the fact that $\mathcal{L}(\theta)$ is continuous in θ, and that $H(\mathcal{L})$ is a continuous function of $\mathcal{L}$.

GetRidges determines which ridges we explore from a given branching point and in what order. Note that from a saddle, one can explore in *opposite* directions along any negative EVec. Optionally, we select the N most negative EVals.

ChooseFromArchive provides the search-order over all possible paths. For example, we can use breadth first search (BFS), depth first search (DFS) or random search. In BFS, the archive is a FIFO queue, while in DFS it is a LIFO queue. Other orderings and heuristics can be used to make the search more efficient, such as ranking the archive by the current loss achieved.

It can be shown that, under mild assumptions, RR maintains a descent direction: At θ, its EVals are $\lambda_1(\theta) \geq \lambda_2(\theta) \cdots \geq \lambda_d(\theta)$ with EVecs $e_1(\theta), \cdots, e_d(\theta)$. We denote the eigengaps as $\Delta_{i-1} := \lambda_{i-1}(\theta) - \lambda_i(\theta)$ and $\Delta_i := \lambda_i(\theta) - \lambda_{i+1}(\theta) = \Delta_i$, with the convention $\lambda_0 = \infty$.

5.4.2 Proof of Concept Experiments

In this section we seek to evaluate Ridge Rider in toy problems to assess its effectiveness in a controlled setting. We begin with a simple synthetic function, before moving to tabular RL.

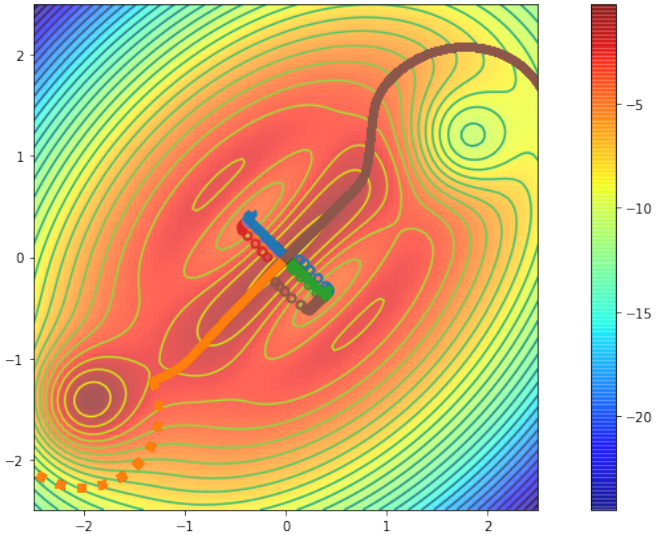


Figure 5.3: Comparison of Ridge Rider (solid squares) with gradient descent (hollow circles) on a two-dimensional loss surface.

In Figure 5.3 we show a comparison of gradient descent (GD, hollow circles) and Ridge Rider (RR, solid circles) on a two-dimensional loss surface. GD starting near the origin only finds the two local minima whose basin have large gradient near the origin. RR starts at the maximum and explores along four paths based on the two eigenvectors of $\mathcal{L}$. Two paths (blue and green) find the local minima while the other two explore the lower-curvature ridge and find global minima. Following the eigenvectors leads RR around a local minimum (orange), while causing it to halt at a local maximum (brown) where either a second ride (dotted) or GD may find a minimum.

Moving to our next experiment, we test whether we can find diverse solutions in RL. We use a toy binary tree environment with a tabular policy (see Fig. 5.4). The agent begins at s_1, selects actions $a \in \{\text{left}, \text{right}\}$, receiving reward $r \in \{-1, 10\}$ upon reaching a terminal node. For the loss function, we compute the expectation of a policy as the sum of the rewards of each terminal node, weighted by the cumulative probability of reaching that node. The maximum reward is 10.

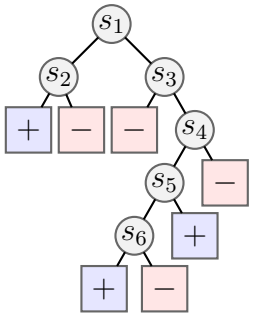


Figure 5.4: A tabular tree environment. Decisions (left or right) are made at each round grey state and the episode terminates at a square leaf node.

We first use the exact version of RR and find θ^{MIS}, by maximizing entropy. For the ChooseFromArchive precedure, we use BFS. In this case we have access to exact gradients so can cheaply re-compute the Hessian and EVecs. As such, when we call UpdateRidge we adapt the learning rate α online to take the largest possible step while preserving the ridge, similar to Backtracking Line Search [205]. We begin with a large α, take a step, recompute the EVecs of the Hessian and the maximum overlap δ. We then and sequentially halve α until we find a ridge satisfying the δ_{break} criteria

(or α gets too small). In addition, we use the following criteria for EndRide: (1) If the dot product between e_i and e_i' is less than δ_{break} (2) If the policy stops improving.

We run RR with a maximum budget of $T = 10^5$ iterations similarity $\delta_{\text{break}} = 0.95$, and take only the top $N = 6$ in GetRidges. As baselines, we use gradient descent (GD) with random initialization, GD starting from the MIS, and random norm-one vectors starting from the MIS. All baselines are run for the same number of timesteps as used by RR for that tree. For each depth $d \in \{4, 6, 8, 10\}$ we randomly generate 20 trees and record the percentage of positive solutions found.

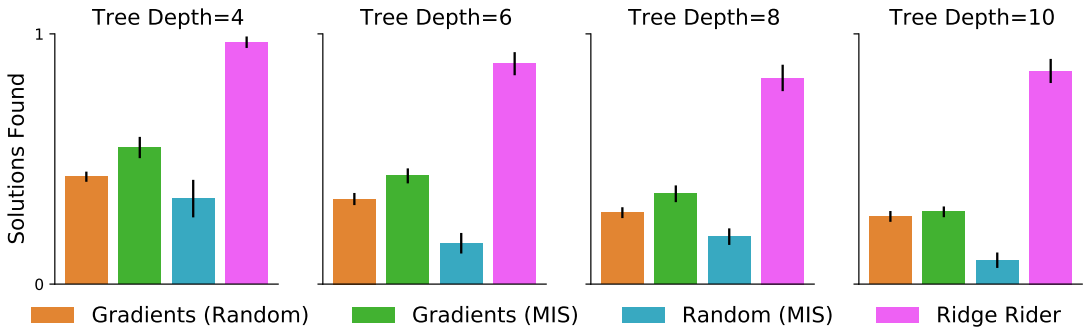


Figure 5.5: Results from running each method on trees with depths. A "solution" corresponds to a leaf node with a positive reward, and the y axis here shows the proportion of the total number of solutions found. Each plot shows the mean and standard error of 20 independent runs.

In Figure 5.5, we see that RR outperforms all three baselines. While RR often finds over 90% of the solutions, GD finds at most 50% in each setting. Importantly, following random directions performs poorly, indicating the importance of using the EVecs to explore the parameter space. To run this experiment, see the notebook at https://bit.ly/2XvEmZy.

5.4.3 Discussion

We have introduced Ridge Rider, a novel method for finding specific types of solutions, which shows promising results in a diverse set of problems. There is clearly a long way to go before RR is practical for real world or larger scale settings. Scaling RR to more difficult problems will require a way to deal with the noise and

stochasticity of these settings. It will also require more efficient ways to compute eigenvalues and eigenvectors far from the extreme points of the spectrum, as well as better understanding of how to follow them robustly. Finally, RR hints at conceptual connections between generalization in supervised learning and zero-shot coordination, which we are just beginning to understand. Clearly, symmetries and invariances in the loss surface play a crucial, but under-explored, role in this connection.

While overall RR is largely a proof of concept, it is important to consider ideas that are interesting and different even if not yet state of the art. Indeed, it is possible that the challenge of achieving open-endedness will require entirely new classes of methods that we haven't even considered yet. Thus, in this light, it is easy to see the importance of pursuing directions (or ridges) such as RR, where we can find diverse solutions without a hybrid objective or a means for *measuring* diversity. As Goodhart's law suggests, doing so may be counterproductive.

5.5 Future Work

In this chapter we have discussed two distinct approaches for discovering diverse solutions. First we proposed DvD, which seeks to find a set of policies that fill a behavior space, defined as the actions taken by a set of policies. Then we introduced Ridge Rider, which rather than proposing a new diversity-inducing loss function, optimizes for the *original objective* but updates the policy by following eigenvectors of the Hessian in the parameter space.

The goal of this part of the thesis is not to present finished methods that work off the shelf, but instead to propose future directions that can increase the open-endedness of our algorithms. Thus, the question is, how can we use DvD or Ridge Rider to make the system described in Part I unbounded in its potential for discovering interesting new behaviors? The most obvious application is seeking the divergent properties of POET by utilizing methods such as DvD in our agent populations. This could be especially useful for multi-agent learning, where it is crucial to generate a diverse set of co-players [164, 329] and RR has already been shown to discover diverse equilibria [174]. Further, we may wish to develop

representations that make it possible to use multi-objective approaches such as DvD for producing diverse levels as part of a UED curriculum. Doing this requires us to define behavior embeddings for levels rather than agents, which is likely non trivial. Finally, it may be possible to combine insights from each of these approaches, for example by following ridges of the original loss, but using the subsequent behavioral diversity to guide search. Given the challenge of producing diverse, high quality solutions, it is likely that we need multiple diverse methods [65]. This chapter (and many other works [235]) is just the start.

6
Learning World Models

"It aint the things you don't know that get you into trouble, its the things you know for sure that just aint so."

— Attributed to Mark Twain

Contents

6.1 Introduction

The methods described thus far in the thesis provide us with the tools required to produce an agent that can solve a diverse range of challenging tasks, possibly even solving every solvable level produced by an environment (denoted by a UPOMDP).

However, what if *the environment itself* is the bottleneck for AGI [198, 302]? In some cases simulated environments may be of sufficient quality to transfer agents to the real world [146, 189, 209, 298]. However, in many prominent settings the human-designed simulator is likely at best an approximation, and it may not be possible to fully incorporate all of the intricacies of the real world into a human-designed environment.

Consider the example of autonomous vehicles. It is incredibly challenging for an environment to cover every possible edge case, including the physical properties of every road, the nuance of each individual intersection, and the complexity of the other agent behaviors. To produce agents that can perform safely, and preferably at a super-human level in these settings, it is almost certain that we would require our environment to *learn* from real world data. How may this work? There have been works in the field already exploring learned simulators [93, 250], but thus far they remain limited in scope, requiring components to be manually defined. Further, these often focus on more accurate simulation rather than increasing the diversity of possible scenarios in an environment.

In this chapter we push forward the notion that learning a *world model* offers the potential for open-ended learning beyond the confines of a human-designed environment. In particular, we focus on Dyna-style model-based RL [289], whereby a policy is learned entirely in "imagination" (see Section 2.1.6). From now on when we refer to imagination, we refer to an agent learning in an environment with a parameterized neural network producing both the transition and reward functions, i.e. $\mathcal{M}_\psi = (\mathcal{S}, \mathcal{A}, P_\psi, R_\psi, \rho_0)$. This approach has led to impressive performance for RL, both from proprioceptive states [30, 41, 119, 148] and pixels [94, 97, 99, 125]. However, all of these works rely on an agent collecting data from a human-designed RL environment.

The key goal for this chapter is to reduce our reliance on training in simulation in a human-designed environment, instead learning from data, as in many other prominent fields of machine learning such as self-supervised learning [29]. In an RL context this is often referred to as *offline* RL [165]. In this paradigm, world

models have once again been shown to be highly effective [132, 238, 332], making use of penalized objectives to train policies where the model is confident [175]. Interestingly, if a world model has sufficiently diverse data, then it can be used (without a simulator) for zero-shot generalization to novel, unseen tasks [9, 188, 269].

But first, a useful dataset must be acquired. We begin by discussing approaches for *actively* acquiring useful samples from a human-designed simulated environment, which we would hope can generalize in future work to unstructured datasets such as videos. We show that by doing so, we can learn general models that facilitate learning effective policies for previously unspecified tasks in an offline fashion. We then focus on this second part of the problem in more detail. Given a fixed dataset we introduce an approach for generating synthetic transitions that make it possible for a policy to generalize to unseen dynamics.

6.2 World Building via Active Learning

In this section we discuss a general approach for training policies that can acquire useful data for building a world model. We begin in the sequential setting, given access to a reward function, before moving to the batch, reward-free paradigm.

6.2.1 Information Theoretic Exploration

We consider the problem whereby we have a world model, producing an imaginary MDP $\mathcal{M}_\psi$ (see: Section 2.1.6). In this section we consider the case where our world model is an ensemble of individual models, such that we can produce epistemic uncertainty estimates [150]. We wish to train an exploration policy π_{EXP} such that the data we acquire is useful for our desired objective. If given access to no other information, a sensible approach is to seek data maximizing *expected information gain* as follows [269]:

$$\pi_{\text{EXP}} = \operatorname*{argmax}_\pi \mathcal{I}\left(d^\pi_{\mathcal{M}_\psi}; \mathcal{M}_\psi\right) = \mathcal{H}(d^\pi_{\mathcal{M}_\psi}) - \mathcal{H}(d^\pi_{\mathcal{M}_\psi}|\mathcal{M}_\psi) \tag{6.1}$$

where $\mathcal{I}$ is the mutual information, $\mathcal{H}$ is the entropy and $d^\pi_{\mathcal{M}_\psi}$ is the distribution of states visited by the policy π in the imaginary MDP $\mathcal{M}_\psi$. This objective produces

π_{EXP}, a policy whose visitation distribution has a high entropy when computed over model samples, but has low entropy for each individual MDP model (i.e., high epistemic/reducable uncertainty). To make this objective more general, we represent the trajectory data collected by a policy with a "summary" embedding space [193, 212]. Let $\Phi : \Gamma \to \Omega$ be a summary function mapping trajectories from a trajectory space Γ into an embedding (or summary) space Ω. $\mathbb{P}^{\Phi}_{\pi}[\mathcal{M}_{\psi}]$ denotes the embedding distribution generated by policy π in imaginary MDP $\mathcal{M}_{\psi}$. We can now write the objective from Eq. 6.1 as follows:

$$\pi_{\text{EXP}} = \underset{\pi}{\text{argmax}}\, \mathcal{I}\left(\mathbb{P}^{\Phi}_{\pi}[\mathcal{M}_{\psi}]; \mathcal{M}_{\psi}\right) = \mathcal{H}(\mathbb{P}^{\Phi}_{\pi}[\mathcal{M}_{\psi}]) - \mathcal{H}(\mathbb{P}^{\Phi}_{\pi}[\mathcal{M}_{\psi}]|\mathcal{M}_{\psi}) \tag{6.2}$$

This more general framework allows us to consider multiple representations for trajectories, in a similar fashion to behavioral characterizations in quality diversity algorithms [235].

6.2.2 Ready Policy One

In the first setting we consider, we have access to a single environment with a specified reward function. The goal is thus to collect batches of data to maximally improve our model with the reward function in mind. Concretely, we will try to maximize the performance of our model with only a limited number of samples from the environment. As such, we choose a behavioral embedding Φ that is simply the predicted reward of our policy for each timestep.

Online Active Learning

We consider the setting where we jointly optimize π_{EXP} for both reward and information gain, since we initially focus on environments where a reward function is provided so we wish to avoid focusing on stochastic or challenging regions of the state space which have no impact on the task at hand. Therefore policies used for data collection also perform well in the true environment, and means we can use these policies for evaluation. Consequently, a separate "exploit" agent does

not need to be trained. Concretely, since we also have the reward function, we optimize our policy for a hybrid objective as follows:

$$\pi_{\text{EXP}} = \underset{\pi}{\text{argmax}}\, \lambda \mathcal{I}\left(\mathbb{P}_{\pi}^{\Phi}[\mathcal{M}_{\psi}]; \mathcal{M}_{\psi}\right) + (1-\lambda)\mathbb{E}_{\tau\sim\pi}\left[\mathcal{R}_{\psi}(\tau)\right] \tag{6.3}$$

where $\mathcal{R}_{\psi}(\tau)$ is the cumulative (undiscounted) reward from an imagined trajectory and $\lambda \in [0, 1]$ is chosen before training the policy in the $\mathcal{M}_{\psi}$ (see: Section 2.1.6).

Indeed, we can see that this objective takes very different forms depending on the value of λ. If $\lambda = 0$, we will simply explore using a greedy policy, maximizing expected return inside the imagined MDP. By contrast, $\lambda = 1$ means we maximize the disagreement from our model ensemble, analogous to acquiring data maximizing expected information gain [178, 179]. As we will show next this can be framed as an online active learning problem, where we pose the choice of λ as the choice of policy for exploration.

We use the Exponential Weights [11] framework to derive an adaptive algorithm for the selection of λ. In this setup we consider k experts making recommendations at the beginning of each round. After sampling a decision $i_t \in \{1, \cdots, k\}$ from a distribution $\mathbf{p}^t \in \Delta_k$ with the form $\mathbf{p}^t(i) \propto \exp\left(\ell_t(i)\right)$ the learner experiences a loss $l_{i_t}^t \in \mathbb{R}$. The distribution $\mathbf{p}^t$ is updated by updating ℓ_t as follows:

$$\ell_{t+1}(i) = \begin{cases} \ell_t(i) + \eta \frac{l_i^t}{\mathbf{p}^t(i)} & \text{if } i = i_t \\ \ell_t(i) & \text{o.w.} \end{cases} \tag{6.4}$$

For some step size parameter η.

In our case we consider the case when the selection of λ_t is thought as choosing among k experts which we identify as the different values $\{\lambda_i\}_{i=1}^k$. The loss we consider is of the form $l_{i_t} = \hat{G}_{\psi_t}(\theta_{t+1})$, where $G_{\psi_t}(\theta_{t+1})$ is the RMSE of the model under parameters ψ_t on data collected using $\pi_{\theta_{t+1}}$, and θ_{t+1} is the parameter of the policy trained under the choice λ_{i_t}, after incorporating into the model the data collected using the previous policy π_{θ_t}. We then perform a normalization of G[1], hence $\hat{G}$. Henceforth we denote by $\mathbf{p}_{\lambda}^t$ the exponential weights distribution over λ values at time t.

[1] We first subtract the mean of the past five observations then divide by the previous observation.

Algorithm 8 Online Learning Mechanism

Input: step size η, number of timesteps T.
Initialize: $\mathbf{p}_\lambda^1$ as a uniform distribution.
for $t = 1, \ldots, T-1$ **do**
1. Select $i_t \sim \mathbf{p}_\lambda^t$ and $\lambda_t = \lambda_{i_t}$.
2. Use Equation 6.4 to update $\mathbf{p}_\lambda^t$ with

$$l_{i_t}^t = \hat{G}_{\psi_t}(\theta_{t+1})$$

Our version of Exponential Weights algorithm also known as Exponentially Weighted Average Forecaster [35] is outlined in Algorithm 8.

In practice and in order to promote more effective exploration over λ values we sample from a mixture distribution where $\mathbf{p}_\lambda^t$ is not proportional to $\exp(\ell_t)$ but it is a mixture between this exponential weights distribution and the uniform over $[k]$. In other words, let $\epsilon > 0$ be a small parameter. With probability $1 - \epsilon$ the produce i_t as a sample from an exponential weights distribution proportional to $\exp(\ell_t)$, and with probability ϵ it equals a uniform index from $1, \cdots, k$.

Diverse Sample Collection

Consider the problem of online data acquisition from a policy in an environment. Let x refer to the concatenation of a state s and action a, and H refer to the horizon of an episode. At each timestep we receive a set of datapoints $\{x_1, \ldots, x_H\} \sim \pi_{\text{EXP}}$ corresponding to the concatenation of each state and action in a trajectory. At timestep t we have a dataset $\mathbf{X}_t = \{x_1, \ldots, x_n\} \subset \mathbb{R}^d$, where $\mathbf{X}_t \in \mathbb{R}^{d \times n}$ sampled from the sampling policy. We represent this data in the form of the Singular Value Decomposition (SVD) of the symmetric matrix, $\text{Cov}_t = \frac{1}{n}\mathbf{X}_t\mathbf{X}_t^\top = \mathbf{Q}_t^\top \Sigma_t \mathbf{Q}_t \in \mathbb{R}^d$.

Equipped with this representation, we take the top k eigenvalues e_i of Cov_t, where k is smallest such that: $\sum_{i=1}^k e_i \geq (1-\delta)\sum_{i=1}^d e_i$ for some parameter $\delta \in [0, 1]$, and take $n_t = k$. Next we take the corresponding eigenvectors $\mathbf{u}_1, ..., \mathbf{u}_k \in \mathbb{R}^d$ and let $\mathbf{U} \in \mathbb{R}^{d \times k}$ be obtained by stacking them together. We define the Active Subspace [46] $\mathbf{U}^{\text{act}} \in \mathbb{R}^{d \times k}$ as $\mathcal{L}_{\text{active}} \stackrel{\text{def}}{=} \text{span}\{\mathbf{u}_1, ..., \mathbf{u}_k\}$. $\mathbf{U}^{\text{act}}$ is an orthonormal basis of $\mathcal{L}_{\text{active}}$.

We use $\mathbf{U}^{\text{act}}$ to evaluate new data. After we collect n' new samples $\mathbf{V}_{t+1} \in \mathbb{R}^{d \times n'}$, we form a covariance matrix with this new data as $\frac{1}{n'}\mathbf{V}_{t+1}\mathbf{V}_{t+1}^T \in \mathbb{R}^{d \times d}$ and project

it onto $\mathbf{U}^{\text{act}}$. We define the residual at timestep t, RESIDUAL_t, as follows:

$$\text{RESIDUAL}_t = \frac{\text{tr}(\mathbf{V}_{t+1}\mathbf{V}_{t+1}^T - \mathbf{U}\mathbf{U}^T\mathbf{V}_{t+1}\mathbf{V}_{t+1}^T\mathbf{U}\mathbf{U}^T)}{\text{tr}(\mathbf{V}_{t+1}\mathbf{V}_{t+1}^T)} \tag{6.5}$$

Where tr denotes the trace operator. After evaluating RESIDUAL_t we append the new data $\mathbf{V}_{t+1}$ to $\mathbf{X}_t$ to form $\mathbf{X}_{t+1} \in \mathbb{R}^{d\times(n+n')}$. In words, RESIDUAL_t tells us how much of the new data could not be explained by the principal components in the data collected thus far. We stop collecting data and proceed to retrain the model once $\text{RESIDUAL}_t < \alpha$. The full procedure is presented in Algorithm 9.

Let q_t be the probability that at timestep t, step 4. of Algorithm 9 is executed (i.e., $q_t = \text{P}(\text{RESIDUAL}_t < \alpha)$). The evolution of q_t operates in roughly two phases. First, the algorithm tries to collect data to form an accurate estimate of the covariance matrix Cov_t and a stable estimator $\mathbf{U}^{\text{act}}$. During this phase, q_t is small as it is roughly the probability of two random samples $\mathbf{X}, \mathbf{X}' \in \mathbb{R}^{d\times n'}$ aligning. After t_0 steps when the algorithm has built a stable estimator of $\mathbf{U}^{\text{act}}$, the stopping probability stabilizes to a value q^* that solely depends on the trajectories intrinsic noise. Both the values of t_0 and q^* scale with trajectory noise. If the trajectories have little noise, both t_0 and q^* are small. On the other hand, if the trajectories have high noise, the early stopping mechanism will take longer to trigger.

Algorithm 9 Early Stopping Mechanism

Input: thresholds α, δ, maximum number of samples T.
Initialize: training set $\mathbf{X} = \emptyset$
Collect initial samples $\mathbf{X}_1 = \{x_1, \ldots, x_H\}$.
for $t = 2, \ldots, T-1$ **do**
1. Compute Active Subspace $\mathbf{U}^{\text{act}}$ as the result of stacking together some orthonormal basis of $\mathcal{L}_{\text{active}} \stackrel{\text{def}}{=} \text{span}\{\mathbf{u}_1, ..., \mathbf{u}_r\}$ where the vectors $\mathbf{u}_i$ correspond to the top k eigenvalues of the covariance matrix Cov_t.
2. Produce samples $\mathbf{V}_{t+1}$ via the sampling policy
3. Calculate the residual RESIDUAL_t using Equation 6.5.
4. Stop collecting data if $\text{RESIDUAL}_t < \alpha$

This dynamic approach to determining the effective "batch size" of the incoming labeled data is similar to [36], whereby feedback from unsupervised learning is used to control the amount of data collected per batch. However, we do this in a more instantaneous fashion, leveraging data collected so far to determine when to stop.

The Algorithm

We now present our algorithm: Ready Policy One (RP1). At each iteration we select a policy objective (λ) to maximize sample utility, and train a policy on this objective. The policies are trained using the original PPO [267] loss function, but we use the training approach in [265] as this combination delivered more robust policy updates.

Once the policy has been trained inside the model, we use it to generate samples in the real environment. These samples continue until our early stopping mechanism is triggered, and we have sufficiently diverse data to retrain the model. The full procedure is outlined in Algorithm 10.

The overall aim is to therefore determine which part of the model space is both high value and unknown, so that our trained sampling policy can obtain enough data samples pertaining to those regions of the environment.

Algorithm 10 RP1: Ready Policy One

Input: Number of initial samples N_0, number of ongoing samples N_t, number of policies in the ensemble M, number of time steps T.
Initialize: Initial World Model $\hat{f}_0$ comprised of M models.
Collect N_0 samples with a random policy and initialize data set $\mathcal{D}_0 = \{(s_t, a_t), s_{t+1}\}_{t=1}^{N_0}$.
for $t = 1, \ldots, T-1$ **do**

1. Train $\hat{f}_{t-1}$ with $\mathcal{D}_t$ to derive $\hat{f}_t$.
2. Select λ using Algorithm 8.
3. Train exploration policy π_{EXP} using Equation 6.3 in $\hat{f}_t$.
4. Collect new samples $\mathcal{D}_{new} = \{(s_t, a_t), s_{t+1}\}_{t=1}^{N_t}$ in the environment with π_{EXP}, where N_t is defined as the number of time steps required for Algorithm 9 to return.
5. $\mathcal{D}_{t+1} \leftarrow \mathcal{D}_t \cup \mathcal{D}_{new}$
6. Set $\alpha_t = \mathcal{L}(\hat{f}_t(\mathcal{D}_{new}))$

Experiments

Recall that the goal of this section is to efficiently train a world model from a human-designed simulated environment. Thus, the primary goal of our experiments it to evaluate whether our active learning approach for learning world models is more sample efficient than existing approaches. In particular, we test RP1 on

the following MuJoCo continuous control tasks [299]: `HalfCheetah`, `Ant`, `Swimmer` and `Hopper`. Rather than individual algorithms, we compare against the two approaches most commonly used in MBRL:

- **Greedy**: We train a policy to maximize reward in a learned model, and subsequently add noise to discover previously unseen states. This is the approach used in many recent works [148].
- **Variance + Reward (V+R)**: We train a policy with $\lambda = 0.5$, producing a fixed degree of priority for reward and model entropy. This resembles methods with information theoretic exploration bonuses such as [105].

In all cases model-based approaches are learning both the model and policy from scratch in each environment, using the same base RL algorithm and model architecture. Thus, the only difference is the objective of the RL agent when learning inside the imaginary MDP induced by the model. We also compare against a model free baseline, which we train for 10^6, 5×10^6 and 10^7 timesteps. This provides an indication of the asymptotic performance of our policy, if trained in the true environment. For more details on our implementation, see the Appendix (Section D.1).

Table 6.1: Median best performance at a given timestep for ten seeds. Bold indicates the best performing algoirthm. T1 corresponds to the t-stat for RP1 vs. Greedy, T2 corresponds to the t-stat for RP1 vs. V+R. * indicates $p < 0.05$.

	Timesteps	Greedy	V+R	RP1	T1	T2
`HalfCheetah`	10^4	-0.95	-1.1	**100.51**	5.39*	4.34*
`Ant`	10^4	94.72	95.07	**113.63**	4.02*	3.08*
`Swimmer`	10^4	1.08	1.07	**3.24**	1.2	1.98
`Hopper`	10^4	76.5	139.03	**322.22**	4.32*	3.42*
`HalfCheetah`	10^5	260.92	283.27	**390.49**	3.89*	2.62*
`Ant`	10^5	186.36	217.08	**238.4**	3.83*	3.11*
`Swimmer`	10^5	61.76	62.89	**64.19**	-0.11	-1.09
`Hopper`	10^5	487.25	570.09	**619.73**	3.52*	2.21

Table 6.1 shows the main results, where RP1 outperforms both the greedy baseline and the fixed variance maximizing (V+R) approach. Furthermore, we

perform Welch's unequal variances t-test (two-sided) and see that in most cases the results are statistically significant (i.e. $p < 0.05$), aside from `Swimmer` at 10^5 timesteps where all three methods have converged to the optimal solution. In addition, we observe that RP1 is able to achieve strong performance vs. model-free in fewer timesteps than the existing baselines.

Interestingly, we see that simply adding a fixed entropy term (V+R) into the reward function gives improved performance over the baseline greedy approach. This corroborates with findings in [78], where there is, in most tasks, a correlation between regions that are difficult to predict and regions of high reward. However our findings also suggest that this is not always the case, and having the ability to adjust how much we focus on such regions of disagreement is vital. We hypothesize that the fixed V+R approach may collect too many high-variance samples for certain tasks, since we do not tune λ, nor limit batch size. As a result, the trajectories gathered do not necessarily result in strong policy performance, unlike RP1, which aims to maximize policy performance through the data collected.

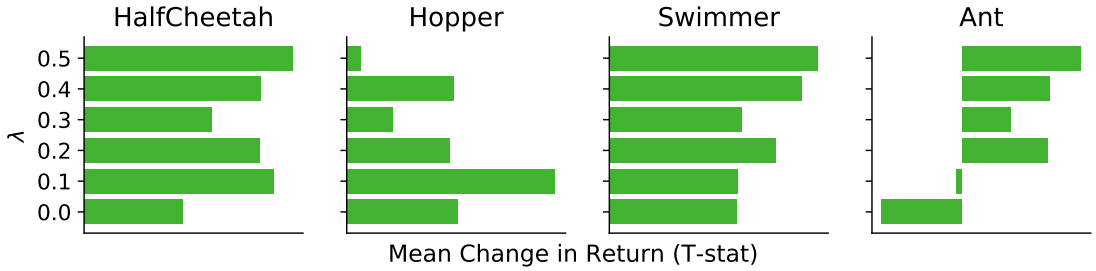


Figure 6.1: Mean one-step policy improvement after a given λ value, for all ten seeds of RP1. Plots show the t-statistic for all runs.

We support this hypothesis with Figure 6.1 where, we show the normalized change in reward for different λ values in each task. In particular, we observe for `HalfCheetah`, `Swimmer` and `Ant`, a greater focus on uncertainty appears positively related to faster learning. However, the opposite is true for `Hopper`. The benefit of our mechanism is the ability to dynamically learn this preference, and thus adapt to the current optimization landscape.

6.2.3 Coordinated Active Sample Collection

We now consider a setting that is more closely aligned with our goal of making open-ended reinforcement learning systems [326]. We seek to design a system that continuously collects diverse batches of data without a specific task in mind. This can, in theory, make it possible to learn policies that can transfer to *any* MDP within a family of MDPs, for example those with differing reward functions, dynamics or state spaces. In this work we focus on the case where the reward is unknown, and use the formalism of a *Contextual* MDP [136], where the observations and dynamics are consistent but the reward function is decided by the context. If the context is unknown at training time, we must collect data that *sufficiently covers the space of possible reward functions*.

To facilitate scalability, we operate in the deployment efficient paradigm [184], whereby policy learning and exploration are completely separate, and during a given *deployment*, we gather a large quantity of data without further policy retraining (c.f. online approaches like DER [308], which take multiple gradient steps *per* exploration timestep in the real environment). Taken together, we consider the *reward-free deployment efficiency* problem.

This differs from previous work as follows: 1) unlike previous deployment efficiency work, our exploration is task agnostic; 2) unlike previous reward-free RL work, we cannot update our exploration policy π_{EXP} during deployment. Thus, the focus of our work is on how to train π_{EXP} offline such that it gathers heterogeneous and informative data which facilitate zero-shot transfer to unknown tasks.

A Population of Diverse Explorers

Recall that in Equation. 6.2 we provided a formulation for maximizing information gain with respect to a specified policy embedding. For the rest of this discussion, we will use the final state embedding as our summary representation, whereby $\Phi(\tau) = h_H$, since in the case of the RSSM, the final latent state from the imagined trajectory is a compact representation of the entire trajectory collected by the policy [251, 287].

We now consider a population-based version of Equation. 6.2, using B agents:

$$\{\pi^{(i)}_{\text{EXP}}\}_{i=1}^{B} = \underset{\boldsymbol{\pi}^B \in \Pi^B}{\operatorname{argmax}} \mathcal{I}\left(\prod_{i=1}^{B} \mathbb{P}^{\Phi}_{\pi^{(i)}}[\mathcal{M}_\psi]; \mathcal{M}_\psi\right) = \mathcal{H}\left(\prod_{i=1}^{B} \mathbb{P}^{\Phi}_{\pi^{(i)}}[\mathcal{M}_\psi]\right) - \mathcal{H}\left(\prod_{i=1}^{B} \mathbb{P}^{\Phi}_{\pi^{(i)}}[\mathcal{M}_\psi] \middle| \mathcal{M}_\psi\right) \tag{6.6}$$

where $\boldsymbol{\pi}^B = \pi^{(1)}, \cdots, \pi^{(B)}$ and $\prod_{i=1}^{B} \mathbb{P}^{\Phi}_{\pi^{(i)}}[\mathcal{M}_\psi]$ is the product measure of the policies' embedding distributions in $\mathcal{M}_\psi$. By definition, the conditional entropy factorizes as:

$$\mathcal{H}\left(\prod_{i=1}^{B} \mathbb{P}^{\Phi}_{\pi^{(i)}}[\mathcal{M}_\psi] \middle| \mathcal{M}_\psi\right) = \sum_{i=1}^{B} \mathcal{H}\left(\mathbb{P}^{\Phi}_{\pi^{(i)}}[\mathcal{M}_\psi] \middle| \mathcal{M}_\psi\right). \tag{6.7}$$

It is now possible to show that maximum information gain is achieved with a *diverse* set of agents:

Lemma 8. *When all models $\mathcal{M}_\psi$ in the support of the model posterior are deterministic and tabular, and the space of policies Π consists only of deterministic policies, there always exists a solution $\{\pi^{(i)}_{\text{EXP}}\}_{i=1}^{B}$ satisfying $\pi^{(i)}_{\text{EXP}} \neq \pi^{(j)}_{\text{EXP}} \forall i \neq j$. Moreover, there exists a family of tabular MDP models, such that the maximum cannot be achieved by setting $\pi_{\text{EXP}}(i) = \pi$ for a fixed π.*

The proof of Lemma 8 is in the Appendix (see Section D.2).

Since the objective in Equation 6.6 is submodular, a greedy algorithm yields a $(1 - \frac{1}{e})$ approximation of the optimum [199]. Leveraging this insight, let us assume that we already have a set of policies $\pi^{(1)}, \cdots, \pi^{(i-1)}$; we then select the next policy $\pi^{(i)}$ based on the following greedy objective:

$$\begin{aligned}\pi^{(i)} &= \underset{\tilde{\pi}^{(i)} \in \Pi}{\operatorname{argmax}} \mathcal{I}\left(\prod_{j=1}^{i} \mathbb{P}^{\Phi}_{\tilde{\pi}^{(j)}}[\mathcal{M}_\psi]; \mathcal{M}_\psi \middle| \tilde{\pi}^{(j)} = \pi^{(j)} \;\; \forall j \leq i-1\right) \\ &= \mathcal{H}\left(\prod_{j=1}^{i} \mathbb{P}^{\Phi}_{\tilde{\pi}^{(j)}}[\mathcal{M}_\psi] \middle| \tilde{\pi}^{(j)} = \pi^{(j)} \;\; \forall j \leq i-1\right) - \mathcal{H}\left(\prod_{j=1}^{i} \mathbb{P}^{\Phi}_{\pi^{(j)}}[\mathcal{M}_\psi] \middle| \mathcal{M}_\psi, \tilde{\pi}^{(j)} = \pi^{(j)} \;\; \forall j \leq i-1\right)\end{aligned}$$

Which can be factorized in similar fashion to Equation 6.7 (See Appendix D.2).

A Tractable Objective for Deep RL

Inspired by [269], we make approximations to derive a tractable objective for $\{\pi_{\text{EXP}}\}_{i=1}^{B}$ in the deep RL setting. First, we assume that the final state embedding distributions are Gaussian with means that depend on the policies and sampled worlds, and variances that depend on the worlds, i.e. $\mathbb{P}^{\Phi}_{\pi}[\mathcal{M}_\psi = w] =$

$\mathcal{N}(\mu(w,\pi),\Sigma(w))$. In this case, $\mathcal{H}(\mathbb{P}^{\Phi}_{\pi}[\mathcal{M}_\psi]|\mathcal{M}_\psi = w) = \rho(w)$, and Eq. 6.8 reduces to solving $\pi^{(i)} = \arg\max_{\tilde{\pi}^{(i)}\in\Pi} \mathcal{H}\left(\prod_{j=1}^{i} \mathbb{P}^{\Phi}_{\tilde{\pi}^{(j)}}[\mathcal{M}_\psi]\middle|\tilde{\pi}^{(j)} = \pi^{(j)} \;\; \forall j \leq i-1\right)$ for a policy that maximizes the resulting joint entropy of the embedding distribution when added to the policy population. This produces the following surrogate objective, maximizing a quadratic *cascading* disagreement:

$$\text{PopDiv}^{\Phi}(\pi|\pi^{(1)},\cdots,\pi^{(i-1)}) = \mathbb{E}_{\tau\sim\mathbb{P}^{\pi}[\mathcal{M}_\psi]}\left[\frac{1}{|\mathcal{D}^{(i-1)}|-1}\sum_{\tilde{\tau}\in\mathcal{D}^{(i-1)}}\|\Phi(\tau)-\Phi(\tilde{\tau})\|^2\right]$$

where $\mathcal{D}^{(i-1)}$ is a dataset of imagined trajectories sampled from policies $\pi^{(1)},\cdots,\pi^{(i-1)}$ in the model, and PopDiv is short for Population Diversity.

Finally, following [16, 269], we also add a per state model-disagreement/information gain component to each policy's reward to encourage a richer landscape for data acquisition: $\text{InfoGain}(\pi) = \mathbb{E}_{\tau\sim\mathbb{P}^{\pi}[\mathcal{M}_\psi]}\left[\sum_{(s,a)\in\tau}\sigma(s,a)\right]$ where $\sigma(\cdot,\cdot)$ is the variance across the ensemble latent state predictions (for details see the Appendix, Section D.2). Taken together, these objectives form our approach, which we call ***C**oordinated **A**ctive **S**ample **C**ollection via **D**iverse **E**xplorers* or CASCADE. CASCADE trains agents to optimize: 1) a diversity term (PopDiv) that takes into account the behaviors of the other agents in the population; 2) an information gain term (InfoGain) that encourages an individual agent to sample states that maximally improve the model:

$$\pi^{(i)} = \underset{\pi\in\Pi}{\text{argmax}}\left[\lambda\text{PopDiv}^{\Phi}(\pi|\{\pi^{(\text{j})}_{\text{EXP}}\}^{\text{i}-1}_{\text{j}=1}) + (1-\lambda)\text{InfoGain}(\pi)\right] \tag{6.9}$$

where $0 \geq \lambda \geq 1$ is a weighting hyperparameter that trades off whether we favor individual model information gain or population diversity. Finally, we train the B agents in *parallel* using (policy) gradient descent over θ, which makes it possible to achieve the same wall clock time as training a single agent [48].

Experiments

In our experiments we test whether CASCADE facilitates the learning of generalist agents by evaluating their zero-shot transfer to unseen tasks, given a limited number of reward-free deployments. In all experiments, agents cannot train online

in the environment, and instead execute a fixed exploration policy to gather new data during each deployment. We focus on training from visual domains to test our methods in a high dimensional setting that is closer to real-world problems. All methods make use of a DreamerV2 world model [99] and use the same hyperparameters for model and agent training (more details in the Appendix, Section D.2).

We test against three baselines. The first is random exploration, which is used in the majority of RL publications. Our next baseline is Plan2Explore [269] (P2E), which trains a single exploration policy that optimizes for expected information gain inside a world model. This resembles RP1, and was published concurrently, but the main difference is that P2E considers does not train with rewards and therefore uses state-based disagreement. We then include a population-based version of P2E, which we call *Population Plan2Explore* (PP2E). PP2E trains a set of randomly initialized agents that independently seek to maximize expected information gain.

We test all methods in two separate settings: *Exploring Worlds* where we seek to collect data in challenging exploration environments. We test how well the explorers cover the state space and discover sparse rewards (also known as "deep exploration" [206, 211]). Next we consider *Exploring Behaviors* where we consider collecting data in continuous control environments. In both settings we test zero-shot transfer as follows: 1) we provide reward labels to the model; 2) we train a separate reward head; 3) we train a task specific behavior policy and test it in the environment. This tests whether our model is general enough to facilitate learning downstream policies for previously unknown tasks [151, 269, 330].

Exploring Worlds In many cases we may not know a priori which states in an environment are highly rewarding. Thus, if we wish to use our world model to subsequently train agents to solve tasks, it is crucial that we accurately model as much of state space as possible. Here we consider exploring worlds in three different environments: MiniGrid [38], Atari [20] and Crafter [96]. In all settings, we start with a *random* deployment before beginning reward-free exploration.

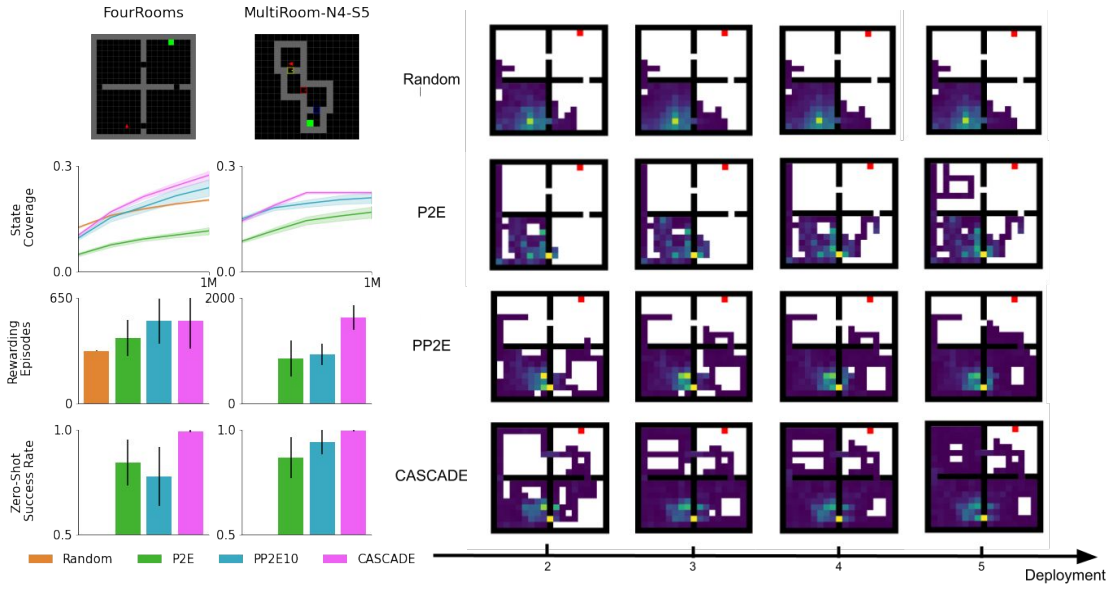


Figure 6.2: MiniGrid Results. Left: Performance statistics for both `FourRooms` and `MultiRoom`. For reward-free exploration we show both "state coverage", which corresponds to the the percentage of states visited on a fixed set of levels, and "rewarding episodes" which is the count of total solved episodes found after 1M steps. Finally for zero-shot transfer we show the success rate of a task-specific behavior policy trained with labels provided only at test time. All plots show the mean of ten seeds with SEM shaded. (Random achieves 0 zero-shot success rate). Right: we show the state coverage on a fixed level in `FourRooms`, using the exploration policy π_{EXP} for each deployment (Test time goal state indicated by the red dot).

We begin with MiniGrid, a set of sparse-reward, partially observable navigation environments commonly used to test state-of-the-art exploration methods [32, 72, 241, 342]. MiniGrid environments are procedurally generated, which provides an additional generalization challenge [124, 248]. We consider the canonical `FourRooms` and the more challenging `MultiRoom`. In each case we are limited to 5 deployments of 200k transitions, for a total of 1M environment interactions.[2] We score the reward-free exploration in two ways: 1) state coverage, where we evaluate the percentage of the states visited by the exploration policies after each deployment on the same set of held out test levels; and 2) rewarding episodes, where we show

[2]Zero-shot success rates of P2E, PP2E and CASCADE increase with more training steps, and CASCADE achieves a near 100% success rate at 1M steps in both `FourRooms` and `MultiRoom`.

the number of rewarding episodes collected during the training process. Note that these two objectives are not useful in isolation, as solved levels may only include a narrow set of observations, while state coverage alone could be maximized without performing deep, goal-seeking exploration in the environment.

We show the results in Figure 6.2, where we see the CASCADE exploration policies cover the state space far more effectively than all baselines. In `FourRooms`, both CASCADE and PP2E find the goal an equal number of times, while CASCADE finds almost double the number of rewarding episodes in the `MultiRoom` environment. We also test zero-shot performance, where CASCADE achieves almost 100% success rate. Interestingly, in both settings the Random policy does cover a large proportion of the state space, but finds fewer rewarding episodes and subsequently does not have a sufficient quantity to train a policy to solve the task (and accordingly achieves 0% success rate).

Next we consider four Atari games: `Montezuma's Revenge`, `Frostbite`, `Hero` and `Freeway`, often used as barometers for exploration capabilities [61]. We use 15 reward-free deployments, each consisting of 200k steps, and use the final model to train a behavior policy with task reward. We test these agents in a zero-shot fashion, with results shown in the first three columns of Figure 6.3. We see that CASCADE gets statistically significantly stronger zero-shot returns compared to the baselines. Furthermore, the baseline methods display no clear trend, with random exploration sometimes offering the strongest performance. On the other hand, we see that CASCADE performs consistently well, providing further evidence of it being a strong general exploration strategy.

Finally, we consider the `Crafter` environment [96], a procedurally generated grid world based on the game of Minecraft. This more complex setting requires the agent to master a variety of compositional skills to fully explore all possible behaviors. Online P2E represents the state of the art reward-free agent, achieving a "Crafter Score" of 2.1. To convert this environment to our reward-free deployment efficiency setting, we deploy only 20 distinct exploration policies, each collecting 50k timesteps, which leads to weaker performance for P2E vs. the reported result

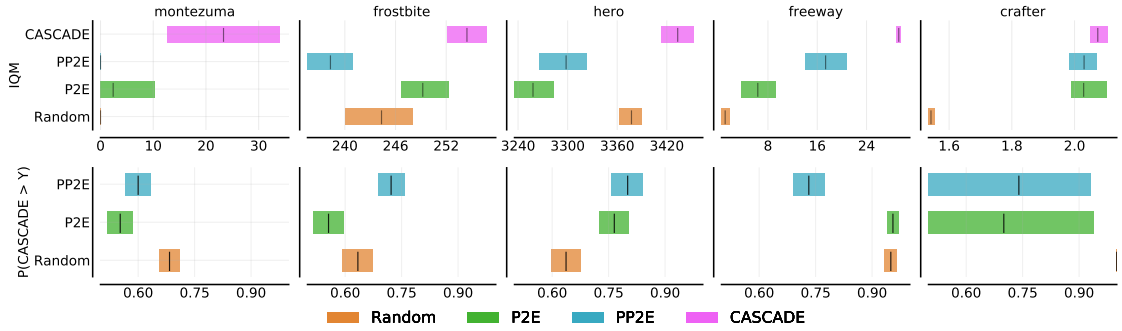


Figure 6.3: Exploring Worlds: Atari and Crafter: Plots show RLiable metric performance aggregated across all seeds. Note that the 'Probability of Improvement' lower CIs all exceed 0.5 for Atari, indicating CASCADE provides a statistically significant improvement (under the Mann-Whitney U test) over all baseline methods [2]. For Atari we show five seeds of zero-shot test performance, for Crafter we show ten seeds of Crafter Score, the geomean of skills discovered in a purely reward-free fashion.

in [99]. However, as we see in Figure 6.3, adding a population of diverse explorers recovers the majority of this performance. We find this result encouraging as the behaviors required to achieve a higher Crafter score are non-trivial. We believe this opens the door to further gains with more active tuning of the behavioral representation used in CASCADE.

Exploring Behaviors Now we have shown our agents are capable of exploring the state space in challenging navigation settings, the next thing we wish to test is whether they sufficiently explore a range of behaviors. To test this, we move from discrete to continuous control, using the DeepMind Control Suite [305]. In these experiments we operate from pixels, and assume access to an initial dataset of 200k transitions. We then conduct two further reward-free deployments, each collecting 200k additional transitions. To test the ability of our methods to improve the generality of a model when given access to prior data, we test three different initial datasets commonly used in the offline RL literature: `random`, `medium` and `expert`. When using random data, this represents learning from scratch. We focus

on the `walker` environment and evaluate our agents on the four tasks proposed in Unsupervised RL Benchmark (URLB, Laskin *et al.* [155]).

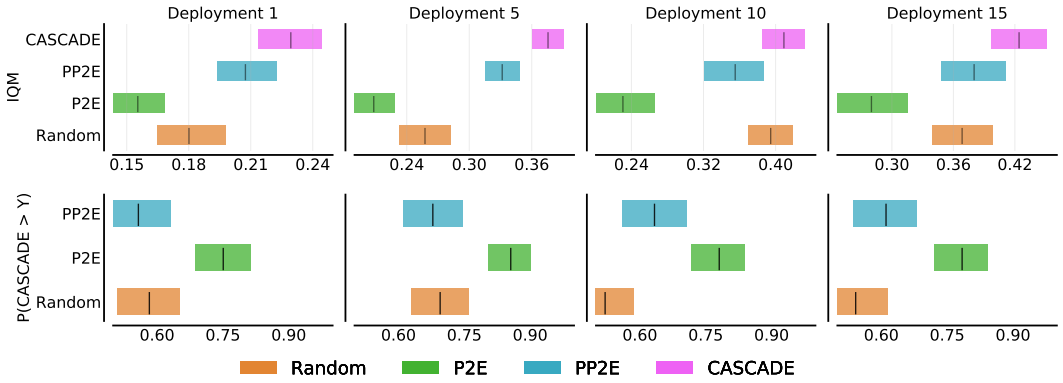


Figure 6.4: DMC Zero-Shot Aggregate Results: Plots show RLiable metric performance aggregated across all three initial datasets (`random`, `medium`, `expert`) for a total of 30 seeds. Note that the 'Probability of Improvement' lower CIs all exceed 0.5, indicating CASCADE provides a statistically significant improvement (under the Mann-Whitney U test) over all baseline methods [2], even after a single deployment.

After each deployment, we test the generality of the world model by training a separate agent from scratch for each of the four individual tasks: `stand`, `walk`, `run`, `flip`. This demonstrates whether the world model is capable of producing an imaginary MDP that facilitates learning specific behaviors, i.e. it is sufficiently accurate at modelling the dynamics. To compare the performance of CASCADE and the baselines, in Figure 6.4 we show the aggregated statistics from the `rliable` library [2] for a given number of deployments. To do this, we combine the results across all datasets (`random`, `medium`, `expert`) and all downstream tasks (`stand`, `walk`, `run`, `flip`) and compute the following: 1) the Inter-Quartile Mean (IQM) of the normalized scores (the robust statistic recommended in [2]); 2) the Probability of Improvement, the likelihood of one method outperforming another on a new, unseen task. We show the results for each deployment in Figure 6.4. As we see, CASCADE shows clear and consistent gains over the next best baseline (PP2E) throughout 15 deployments, showing statistical significance under the Mann-Whitney U test

[181]. Note that here the single agent P2E performs poorly, likely due to only collecting data with a single mode of exploratory behavior.

6.2.4 Discussion and Limitations

In this section we have introduced two new approaches for actively acquiring new data. First, RP1 sought to combine an information gain bonus alongside the task reward when training a policy for a specific environment. This makes learning significantly faster on a series of simple environments, but it does not lead to open-ended learning for a generally capable agent. To address this, CASCADE takes us a large step further by both using *reward-free* exploration and focusing on the more scalable *deployment efficient* setting.

Across a variety of environments we have shown that CASCADE is the most effective method in training generally capable agents. This is because CASCADE provides the best of both worlds: deep exploration by seeking the frontier of the current knowledge (via InfoGain), while covering the space of possible behaviors (via PopDiv). Each of our baselines lacked in one of these two key properties. For example, Random exploration fared poorly when exploring worlds, which requires deep exploration. However, it can be a highly effective method for exploring behaviors. This is likely why boosting entropy in the action space is so effective for robotics tasks [95], but less so in deep exploration tasks [206]. Furthermore, the information theoretic baselines were more effective than Random in Exploring Worlds, but the disappointing performance for P2E in DMC is likely caused by a lack of data diversity, while despite diversity from random initialization, the PP2E populations were still not sufficiently heterogeneous to match the coverage of CASCADE.

Returning to the goal of this chapter: how does CASCADE help us remove the reliance on human-designed simulated environments? We now have a potentially open-ended process whereby diverse data can be collected in parallel to train a world model that is sufficiently general to learn a wide range of downstream behaviors. However, the main limitation here is clear—we still rely on a human-designed

simulator to both collect data and test our agents. In practice if we have access to this then our methods remain constrained to the confines of a potentially inaccurate simulator and a narrow set of environments. In future work we can instead seek to design active learning systems that can scour the *Internet* for useful video data, to produce a model that can learn about a diverse range of real world tasks in an unbounded fashion. Once we have access to this type of open-ended data acquisition process, we can build world models that span a wide range of possible tasks. The next question is, what do we do with the model?

6.3 Augmented World Models

It is likely that once we have a world model, we may wish to use it to train agents that can generalize to a wide range of unseen tasks from the same distribution [31]. However, our goal in this thesis is to produce an open-ended RL system, and we have shown that if we have access to a *parameterized environment* represented as a UPOMDP (see: Chapter 4) we can automatically generate increasingly complex problems. But how do we do this if our agent is no longer training in a parameterized environment, but instead a learned world model?

In this section we describe an initial approach for doing just that—concretely, we focus on the problem where tasks can vary with respect to their underlying dynamics, but share the same state space and reward function. We seek to train a world model from a single offline dataset, and *augment* the model dynamics such that the policy trained inside it is robust to a variety of unseen dynamics. We call this method *Augmented World Models* or AugWM [17]. AugWM can be seen as the first method to propose domain randomization inside a learned model, which it does through a naive, random scaling approach. This essentially boils down to sampling "imaginary" tasks that do not exist, to robustify an agent against variations that do arise when transferring to a new setting. As we will see this is enough to get non-trivial gains when seeking to generalize from an offline dataset to a range of unseen dynamics.

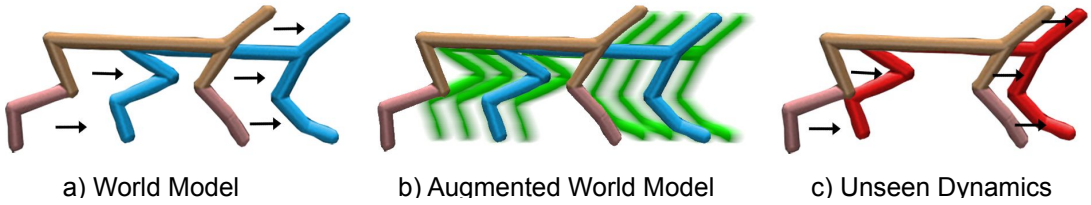


Figure 6.5: An illustration of our approach, each figure shows a transition $P(s,a) \to s'$. In a) we show the World Model dynamics $\widehat{P}$, trained from $\mathcal{D}_{\text{env}}$, a single offline dataset. In b) we show the Augmented World Model, blue represents $\widehat{P}$, while each green agent illustrates an instantiation of augmented dynamics, which is sampled at each timestep: $\widehat{P}_z, z \sim \mathcal{Z}$, where z is a latent context vector and $\widehat{P}_z$ is a new dynamics function consistent with this context. The goal is to approximate c) where we show an unseen test environment, with transition dynamics $P^\star$ (the "optimal" or "true" dynamics of the test task).

6.3.1 Random Dynamics Augmentations

Recall that we are training an agent fully offline inside an imaginary MDP defined by a model, which we denote $\mathcal{M}_\psi$. The goal of this work is to increase the breadth of tasks that can be generated by the model via synthetic, augmented dynamics functions. After training a model to fit an offline dataset, we consider three approaches to subsequently augment the dynamics function (and corresponding imaginary MDP) when training the policy (see Figure 6.5).

We begin with **Random Amplitude Scaling** as in Laskin *et al.* [154], which we refer to as RAD. RAD scales both s_t and s_{t+1} as follows:

$$\mathcal{P}_z : (s_t, a_t, r_t, s_{t+1}) \mapsto (z \odot s_t, a_t, r_t, z \odot s_{t+1}) \tag{6.10}$$

for $z \sim \text{Unif}([c,d]^{|\mathcal{S}|})$, for constants c and d, where $\odot$ refers to the Hadamard product. Given that our focus is on changing dynamics (vs. observational overfitting), we also propose to scale only the next state, i.e., **Random Amplitude Nextstate Scaling** (RANS):

$$\mathcal{P}_z : (s_t, a_t, r_t, s_{t+1}) \mapsto (s_t, a_t, r_t, z \odot s_{t+1}) \tag{6.11}$$

for $z \sim \text{Unif}([c,d]^{|\mathcal{S}|})$. Note that while RANS is more focused on augmenting dynamics than RAS, it still suffers from a dependence on the magnitude of s_{t+1}. As such, we further propose a more targeted augmentation, which we call **Dynamics Amplitude Sampling** (DAS). Rather than directly scale the state, DAS scales the *change in the state* which we denote with $\delta_t = s_{t+1} - s_t$, as follows:

$$\mathcal{P}_z : (s_t, a_t, r_t, s_{t+1}) \mapsto (s_t, a_t, r_t, s_t + z \odot \delta_t) \tag{6.12}$$

for $z \sim \text{Unif}([c,d]^{|\mathcal{S}|})$. The full training procedure is shown in Algorithm 11.

Algorithm 11 Augmented World Models: Training

Input: Offline data $\mathcal{D}_{\text{env}}$, Penalty λ, horizon h, batchsize B, augmentation $\mathcal{Z}$.
Initialize: Ensemble of N dynamics models $\widehat{P}$, policy π. Empty replay buffer $\mathcal{D}_{\widehat{\text{env}}}$
1. Train $\widehat{P}$ in a supervised fashion using $\mathcal{D}_{\text{env}}$.
for *epoch* $= 1, 2, \ldots$ **do**
  (i) Sample initial states: $\{s_1^1, \ldots, s_1^B\} \sim \mathcal{D}_{\text{env}}$
  (ii) Rollout policy (in parallel) in model MDP, using λ-penalized reward, storing all data in $\mathcal{D}_{\widehat{\text{env}}}$
  (iii) Train policy using $\mathcal{D}_{\text{env}} \cup \mathcal{D}_{\widehat{\text{env}}}$. For each (s, a, r, s'), sample $z \sim \mathcal{Z}$, apply $\mathcal{P}_z$.
Return: Policy π

One crucial addition to our method is the use of *context*. Concretely, when we are optimizing the policy using a batch of data, we concatenate the next state with the augmentation vector z. This allows our policy to be informed of the specific augmentation that was applied to the environment and thus behave accordingly. We show the full procedure in Figure 6.6.

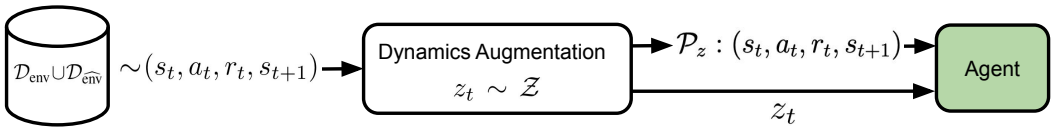


Figure 6.6: Training policies in Augmented World Models. For each state-action pair sampled from the buffer, a new augmentation $z_t \sim \mathcal{Z}$ is sampled to produce an augmentation operator $\mathcal{P}_z$, which is applied to the transition. The policy is then trained with the new tuple of data, with the context concatenated to the state.

However, at test time we do not know z, so what can we use to condition the policy? Next we propose a solution to this problem, approximating the context on the fly for zero-shot adaptation.

6.3.2 Self-Supervised Policy Selection

In the meta-learning literature there have been many recent successes making use of a *learned context* to adapt a policy at test time to a new environment [242, 345, 346], typically using a blackbox model with a latent state. Crucially, these approaches require several episodes to adapt at test time, making them unfeasible if we wish for our learned policy to be able to adapt in a zero-shot fashion.

What makes our setting unique is we explicitly know what the context represents: a linear transformation of s_t, or δ_t. Using this insight, we are able to learn an effective context on the fly at test time. Concretely, we observe that from a state s_t drawn from $M^\star$, we can sample an action $a_t \sim \pi$ and then compute an approximate $\widehat{s}_{t+1}$ using our model $\widehat{P}$. With $\widehat{s}_{t+1}$, we have a sample estimate of the *state change under* $\widehat{P}$, i.e. $\widehat{\delta}_t = \widehat{s}_{t+1} - s_t$. We can make this approximation of the next state without interacting with the environment, but once we do take the action a_t in the environment, we then receive the *true next state* s_{t+1} and can store the true difference $\delta_t = s_{t+1} - s_t$. Using the DAS augmentation, we can approximate z as ${}^{\delta_t}/_{\hat{\delta}_t}$.

Algorithm 12 Augmented World Models: Testing

Input: Initial state s_1, horizon H, policy π, world model $\widehat{P}$, initial context $\hat{z} = \mathbf{1}^{|\mathcal{S}|}$
Initialize: Linear model f_ψ, dataset $\mathcal{D} = \emptyset$, return $R_0 = 0$.
for *step* $= 1, 2, \ldots H$ **do**
  Select action: $a_t \sim \pi(s_t, \widehat{z}_t)$
  Take action: $s_{t+1} \sim P^\star(s_t, a_t)$
  Update return: $R_{t+1} = R_t + R^\star(s_t, a_t, s_{t+1})$
  Update Dataset: $\mathcal{D} \cup (s_t, \delta_t)$.
  Update Linear model by minimizing $\mathcal{L}_{\text{MSE}}(\psi, \mathcal{D})$.
  Predict new context using $\widehat{z}_t = {}^{\delta_t}/_{\hat{\delta}_t}$.
Return: R_T

This however is retrospective; we can only approximate z having already seen the next state, by which time our agent has already acted. Furthermore, we believe under changed dynamics the true z likely depends on s, thus we cannot

use a previous z for future timesteps. Therefore, we learn a forward dynamics model using the data collected *during* the test rollout. After h timesteps in the environment, we have the following dataset: $\mathcal{D} = \{(s_1, s_2), \ldots, (s_{h-1}, s_h)\}$. This allows us to learn a simple dynamics model $f_\psi : (s_t) \mapsto \delta_t = s_{t+1} - s_t$, by minimizing the mean squared error $\mathcal{L}_{\text{MSE}}(\psi, \mathcal{D})$. Notably, since we never actually plan with this model, it does not need to be as accurate as a typical dynamics model in MBRL. Instead, it is crucial that the model learns quickly enough (in the order of hundreds of timesteps) such that we can use it in a zero-shot evaluation. Thus, we choose to use a simple linear model for f.

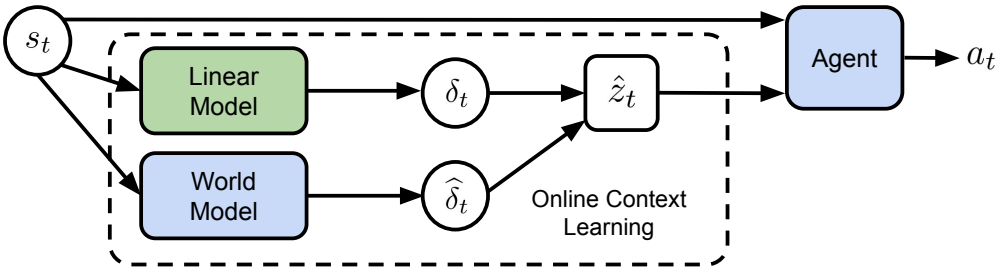


Figure 6.7: Self-supervised policy adaptation via learned context. At each timestep, the state s_t is fed into a linear model (trained online at each timestep) to predict the change in state δ_t, and also passed to the fixed World Model (trained on the offline data) to predict the change in state under $\widehat{P}$, $\widehat{\delta}_t$. The approximate context is then $z_t = {}^{\delta_t}/_{\widehat{\delta}_t}$, which is concatenated with s_t and passed to the agent to produce an action.

6.3.3 Empirical Results

In our experiments we seek to evaluate whether Augmented World Models provide a sufficiently rich and effective distribution of imaginary MDPs. To test this we train Augmented World Models from a single offline environment and evaluate the subsequent ability of agents trained in them to generalize to unseen dynamics. To answer this question we perform a detailed analysis, using multiple benchmarks from the D4RL dataset [79]. Namely, we consider the Random, Medium, Mixed and Medium-Expert datasets for both `Walker2d` and `HalfCheetah`. For each dataset

we train on the original dynamics, but then test the trained policy on a set of varied dynamics where we change both the mass of the agent and damping coefficient by a multiplicative factor. In this work we consider a grid of the following values for `HalfCheetah`: $\{0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 1.75\}$ and $\{0.5, 0.75, 1.0, 1.25, 1.5\}$ for `Walker2d`, representing a significant out-of-distribution shift compared to the standard environment (both in Gym and D4RL) which corresponds to both mass and damping being set to 1.0.

In each setting, we compare AugWM against a state-of-the-art offline model-based RL approach, *Model-based Offline Policy Optimization* (MOPO, Yu *et al.* [332]). All results will be zero-shot performance from a single offline environment, i.e. training solely using the data provided, and not seeing the test environment until evaluation. The results are shown as a total return number averaged over both dimensions in Table 6.2. For additional implementation details (e.g., hyperparameters) see Appendix D.3. Using Welch's t-test, AugWM provides *statistically significant* improvements in zero-shot performance v.s. MOPO in many cases, achieving successful policies where MOPO fails.

Table 6.2: Each entry for `HalfCheetah` is the mean of 49 different dynamics, while for `Walker2d` it is over 25 dynamics. Results are mean $\pm$1std. $\star$ indicates $p < 0.05$ for Welch's t-test for gain over MOPO (5 seeds).

Dataset Type	Environment	MOPO	AugWM (Ours)
Random	`HalfCheetah`	2303 ± 112	2818 ± 197 $\star$
Random	`Walker2d`	569 ± 103	706 ± 139
Mixed	`HalfCheetah`	3447 ± 218	3948 ± 122 $\star$
Mixed	`Walker2d`	946 ± 95	1317 ± 206 $\star$
Medium	`HalfCheetah`	2954 ± 89	2967 ± 106
Medium	`Walker2d`	1477 ± 337	1614 ± 440
Med-Expert	`HalfCheetah`	1590 ± 766	2885 ± 432 $\star$
Med-Expert	`Walker2d`	1062 ± 334	2521 ± 316 $\star$

In addition, we note that dynamics may change *during* an episode; consider a robot that suffers a motor fault, reducing the power delivered to its joints. Evidently the underlying dynamics have been altered, and being robust to such changes when

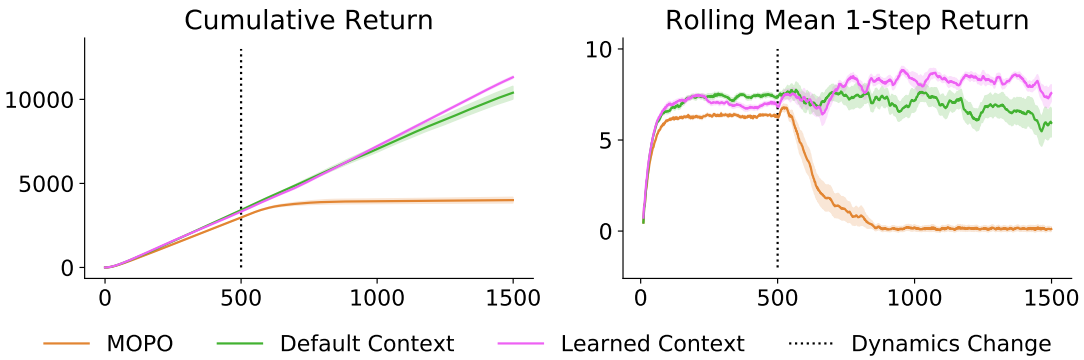


Figure 6.8: Performance under *changing* dynamics. We alter the dynamics by reducing the mass and damping coefficients during an episode, after 500 timesteps. Note that this corresponds to an "easier" test task but the baseline MOPO agent is unable to adapt so falls over. Instead, AugWM is able to adapt its gait and run faster with the reduced mass. Left = cumulative returns, right = rolling average single step reward. All results are averaged over twenty seeds, shaded area shows the standard error of the mean.

only training from a single dataset of offline experience is challenging. To illustrate this, we perform a 1500 step rollout in the `HalfCheetah` environment, starting with offline dynamics (mass/damping = 1), before changing to mass = 0.75, damping = 0.5 after 500 steps; performance is shown in Figure 6.8. Observe that after 500 steps, MOPO performance is dramatically reduced. This is because the agent continues to apply the same force and thus falls forward with lighter mass. For our AugWM agent, performance initially drops, then when the new context is learned we achieve *higher* performance than before, making use of the lighter torso.

6.3.4 Discussion

We believe that our experiments provide significant support to the claim that training with an *Augmented World Model* improves zero-shot dynamics generalization from a single offline environment. In a broad set of commonly used datasets, and with a wide range of out-of-distribution dynamics, our algorithm learns good policies where a state-of-the-art baseline fails.[3] This is due to two novel contributions: 1)

[3]For videos see: https://sites.google.com/view/augmentedworldmodels/

using *dynamics augmentation* rather than *observation augmentation*; 2) training and testing with a context-based policy. In essence, Augmented World Models represents the first step towards learning in truly imaginary or synthetic tasks. It is possible that none of the random augmentations produce genuinely reasonable dynamics, and yet, training on them produces an agent that is robust to unseen physics at test time.

Regarding limitations, we note that in more complex settings we may not see the same performance for this simple approach. This could be because the dynamics changes are out of the distribution of the augmentations proposed in this work, or due to the difficulty of modeling the task with a linear model. We note that linear models have achieved success in planning [91] and meta learning [225], and are effective in our case due to their data efficiency, but can be replaced by more flexible models to deal with different augmentations. Indeed, given our work is the first of its kind, we believe significant improvements are possible, such as using more complex and problem-specific augmentations.

6.4 Future Work

In this chapter we have focused on both building world models and then subsequently using them to generate new, synthetic tasks. In both cases we have only achieved proof of concept results, thus, the future work is clear—we need to scale these methods to larger problems. Doing so will likely be a challenge, requiring many future innovations.

For active world building, we need to use larger models trained on much larger, more diverse datasets, for example from the Internet. When training agents inside a world model, we can consider building upon Augmented World Models with more sophisticated means to sample environments (or levels) from the world that can robustify our trained agent. We could also consider sampling different styles or distracting backgrounds to robustify vision-based agents from observational overfitting [176, 276]. In the next chapter we conclude the thesis with a discussion of how these approaches could combine with other contributions from the thesis to produce a truly open-ended system.

7
Conclusion

"We should expect surprises."

— Kenneth O. Stanley, *Why Open-Endedness Matters*

The core belief in this thesis is that reinforcement learning (RL), the paradigm of agents learning from autonomous interaction in an environment, may indeed be "enough" for training generally capable agents. However, as we argue, the key missing ingredient is open-endedness.

We began in Part I by discussing the tendency for RL systems to rely on static, handcrafted configurations. We introduced methods that can learn and adapt multiple agent hyperperameters on the fly in a single training run. We also proposed an approach to automatically evolve an increasingly complex curriculum of environments, producing a generally capable agent within the given domain. This shifts the emphasis from manually designing learning algorithms and human-engineered environments to designing systems which coevolve both environments and agent configurations. However, as described, this alone cannot be considered truly open-ended, since we can only maximize the agent performance in a single environment distribution. This process may produce interesting new things for a moderate amount of time, but would likely eventually plateau. Indeed, if we wish

for our systems to be *truly* open-ended, and give them the chance of discovering things that can surprise us, we need to go even further.

In Part II we propose two pathways to fully remove the shackles from our RL agents. First, in Chapter 5 we posit that diversity will play a crucial role, both in the form of diverse exploration agents that can capture a wide range of experiences, and subsequently by learning diverse behaviors such that our agent is robust to possible changes in the world. Indeed, diversity may be the only way to ensure our agents can cover all possible degrees of variability they may encounter in the real world. Further, without diversity in our environments and agent behaviors we may miss vital "stepping stones" that would be ignored with greedy optimization [163, 280], but play a key role in future progress towards more sophisticated forms of intelligence [268, 315].

Finally, in Chapter 6 we argue that one path to true open-endedness is via learning *world models* and training agents inside imaginary MDPs they induce. To build these world models we propose new methods capable of collecting diverse batches of task-agnostic data. Once trained, these models could could endow our agents with the ability to generate their own imaginary worlds, possibly via an infinite array of creative interpretations of the real world [243]. We showed the first proof of concept for this approach by training agents inside Augmented World Models, which make it possible to generalize to new, unseen dynamics.

7.1 A Recipe for Open-Ended RL

To conclude, we seek to unify many of the ideas from this thesis into a single vision for a large scale, open-ended learning system. We could combine our approaches in Part II to both collect new data for a world model and then subsequently use it to generate imaginary environments. Naturally, with this ability to sample environments, we can use Unsupervised Environment Design to create an adaptive distribution over the samples (see: Chapter 4). Further, we can also utilize methods from Chapter 3 to adapt agent configurations inside the model, which could be computationally efficient given that models run on hardware accelerators [99].

Combining these pieces of the puzzle, we can envisage a system whereby a world model gets more accurate on a broader set of environments, and an agent is robustified against all possible imagined worlds. We can seek to scale this system as compute continues to get cheaper, data becomes more abundant and novel architectures continue to emerge [3, 29, 56, 243]. In theory, if data can automatically be collected from a sufficiently rich source then this process could have no bounds, as both the agent and world model coevolve indefinitely by seeking to surprise each other. In practice, this could produce increasingly novel, interesting things as well as a generally capable agent. It might even be open-ended.

Appendices

A

Appendix for Population Based Bandits

A.1 Implementation Details

Continuous Control Experiments For all experiments we set $\beta_t = c_1 + \log(c_2 t)$ with $c_1 = 0.2$ and $c_2 = 0.4$, as in the traffic speed data experiment from [25]. In Table A.1 and we show hyperparameters for the PPO experiments in the BipedalWalker and LunarLanderContinuous environments. In Table A.2 we show the bounds for the hyperparameters learned by PBT and PB2. All methods were initialized by randomly sampling from these bounds.

Table A.1: Fixed

Parameter	Value
Filter	$MeanStdFilter$
SGD Iterations	10
Architecture	32-32
ready	5×10^4

Table A.2: Learned

Parameter	Value
Batch Size	$[1000, 60000]$
GAE λ	$[0.9, 0.99]$
PPO Clip ϵ	$[0.1, 0.5]$
Learning Rate η	$[10^{-5}, 10^{-3}]$

Procgen Experiments In Table A.3 we show the the fixed parameters, taken from UCB-DrAC [240]. In Table A.4 we show the continuous hyperparameters optimized by our methods. For the categorical variables, we use the data augmentations used in [141, 154, 240]: *crop, grayscale, cutout, cutout-color, flip, rotate, random convolution and color-jitter*. For PB2 use the same UCB hyperparameters

as in the continuous control experiments, which come from [25]. Concretely, we set $c1 = 0.2$, $c2 = 0.4$. We believe performance may be increased by selecting these parameters more carefully. However, it is promising to see that this fine tuning is not crucial for performance.

Table A.3: Fixed

Parameter	Value
Bootstrapping γ	0.999
GAE λ	0.95
# timesteps per rollout	256
# epochs per rollout	3
# minibatches per epoch	8
optimizer	Adam

Table A.4: Learned

Parameter	Value
PPO Clip (ϵ)	$[0.01, 0.5]$
Learning Rate	$[10^{-5}, 10^{-3}]$
Entropy Coeff	$[0, 0.2]$
Regularization (α_r)	$[0.01, 0.5]$

A.2 PB2 Theoretical Results

A.2.1 Derivation for Lemma 1

Proof. We have a reward at the starting iteration $F_1(x_1)$ as a constant that allows us to write the objective function as:

$$F_T(x_T) - F_1(x_1) = F_T(x_T) - F_{T-1}(x_{T-1}) + \cdots + F_3(x_3) - F_2(x_2) + F_2(x_2) - F_1(x_1) \tag{A.1}$$

Therefore, maximizing the left of Eq. (A.20) is equivalent to minimizing the cummulative regret as follows:

$$\max\left[F_T(x_T) - F_1(x_1)\right] = \max \sum_{t=1}^{T} F_t(x_t) - F_{t-1}(x_{t-1}) = \max \sum_{t=1}^{T} f_t(x_t) = \min \sum_{t=1}^{T} r_t(x_t)$$

where we define $f_t(x_t) = F_t(x_t) - F_{t-1}(x_{t-1})$, the regret $r_t = f_t(x_t^*) - f_t(x_t)$ and $f_t(x_t^*) := \max_{\forall x} f_t(x)$ is an unknown constant. □

A.2.2 Convergence Analysis

We minimize the cumulative regret R_T by sequentially suggesting an $\mathbf{x}_t$ to be used in each iteration t. We shall derive the upper bound in the cumulative regret and show that it asymptotically goes to zero as T increases, i.e., $\lim_{T\to\infty} \frac{R_T}{T} = 0$.

A. Appendix for Population Based Bandits

We make the following smoothness assumption to derive the regret bound of the proposed algorithm.

Assumptions. We will assume that the kernel k holds for some (a, b) and $\forall L_t \geq 0$. The joint kernel satisfies for all dimensions $j = 1, ..., d$,

$$\forall L_t \geq 0, t \leq \mathcal{T}, p(\sup \left| \frac{\partial f_t(\mathbf{x})}{\partial \mathbf{x}^{(j)}} \right| \geq L_t) \leq a e^{-(L_t/b)^2}. \tag{A.2}$$

These assumptions are achieved by using a time-varying kernel $k_{time}(t, t') = (1-\omega)^{\frac{|t-t'|}{2}}$ [25] with the smooth functions [25]. For completeness, we restate Lemma 9, 10, 11, 12 from [25, 278], then present our new theoretical results in Lemma 13, 14, 15 and Theorem 16.

Lemma 9 ([278]). *Let* $L_t = b\sqrt{\log 3da\frac{\pi_t}{\delta}})$ *where* $\sum_{t=1}^{T} \frac{1}{\pi_t} = 1$, *we have with probability* $1 - \frac{\delta}{3}$,

$$|f_t(\mathbf{x}) - f_t(\mathbf{x}')| \leq L_t ||\mathbf{x} - \mathbf{x}'||_1, \forall t, \mathbf{x}, \mathbf{x}' \in D. \tag{A.3}$$

Lemma 10 ([278]). *We define a discretization* $D_t \subset D \subseteq [0, r]^d$ *of size* $(\tau_t)^d$ *satisfying* $||\mathbf{x} - [\mathbf{x}]_t||_1 \leq \frac{d}{\tau_t}, \forall \mathbf{x} \in D$ *where* $[\mathbf{x}]_t$ *denotes the closest point in* D_t *to* $\mathbf{x}$. *By choosing* $\tau_t = \frac{t^2}{L_t d} = rdbt^2\sqrt{\log(3da\pi_t/\delta)}$, *we have*

$$|f_t(\mathbf{x}) - f_t([\mathbf{x}]_t)| \leq \frac{1}{t^2}.$$

Lemma 11 ([278]). *Let* $\beta_t \geq 2\log\frac{3\pi_t}{\delta} + 2d\log\left(rdbt^2\sqrt{\log\frac{3da\pi_t}{2\delta}}\right)$ *where* $\sum_{t=1}^{T} \pi_t^{-1} = 1$, *then with probability at least* $1 - \frac{\delta}{3}$, *we have*

$$|f_t(\mathbf{x}_t) - \mu_t(\mathbf{x}_t)| \leq \sqrt{\beta_t}\sigma_t(\mathbf{x}_t), \forall t, \forall \mathbf{x} \in \mathcal{D}.$$

Proof. We note that conditioned on the outputs $(y_1, ..., y_{t-1})$, the sampled points $(\mathbf{x}_1, ..., \mathbf{x}_t)$ are deterministic, and $f_t(\mathbf{x}_t) \sim \mathcal{N}(\mu_t(\mathbf{x}_t), \sigma_t^2(\mathbf{x}_t))$. Using Gaussian tail bounds [313], a random variable $f \sim \mathcal{N}(\mu, \sigma^2)$ is within $\sqrt{\beta}\sigma$ of μ with probability at least $1 - \exp\left(-\frac{\beta}{2}\right)$. Therefore, we first claim that if $\beta_t \geq 2\log\frac{3\pi_t}{\delta}$ then the selected points $\{\mathbf{x}_t\}_{t=1}^{T}$ satisfy the confidence bounds with probability at least $1 - \frac{\delta}{3}$

$$|f_t(\mathbf{x}_t) - \mu_t(\mathbf{x}_t)| \leq \sqrt{\beta_t}\sigma_t(\mathbf{x}_t), \forall t. \tag{A.4}$$

This is true because the confidence bound for individual $\mathbf{x}_t$ will hold with probability at least $1 - \frac{\delta}{3\pi_t}$ and taking union bound over $\forall t$ will lead to $1 - \frac{\delta}{3}$.

We show above that the bound is applicable for the selected points $\{\mathbf{x}_t\}_{t=1}^{T}$. To ensure that the bound is applicable for all points in the domain D_t and $\forall t$, we can set $\beta_t \geq 2\log\frac{3|D_t|\pi_t}{\delta}$ where $\sum_{t=1}^{T}\pi_t^{-1} = 1$ e.g., $\pi_t = \frac{\pi^2 t^2}{6}$

$$p(|f_t(\mathbf{x}_t) - \mu_t(\mathbf{x}_t)| \leq \sqrt{\beta_t}\sigma_t(\mathbf{x}_t) \geq 1 - |D_t|\sum_{t=1}^{T}\exp(-\beta_t/2) = 1 - \frac{\delta}{3.}. \tag{A.5}$$

By discretizing the domain D_t in Lem. 10, we have a connection to the cardinality of the domain that $|D_t| = (\tau_t)^d = \left(rdbt^2\sqrt{\log(3da\pi_t/\delta)}\right)^d$. Therefore, we need to set β_t such that both conditions in Eq. (A.4) and Eq. (A.5) are satisfied. We simply take a sum of them and get $\beta_t \geq 2\log\frac{3\pi_t}{\delta} + 2d\log\left(rdbt^2\sqrt{\log\frac{3da\pi_t}{2\delta}}\right)$. □

We use TB to denote the batch setting where we will run the algorithm over T iterations with a batch size B. The mutual information is defined as $\tilde{\mathbf{I}}(f_{TB}; y_{TB}) = \frac{1}{2}\log\det\left(\mathbf{I}_{TB} + \sigma_f^{-2}\tilde{K}_{TB}\right)$ and the maximum information gain is as $\tilde{\gamma}_T := \max\tilde{\mathbf{I}}(\mathbf{f}_{TB};_{TB})$ where $\mathbf{f}_{TB} := f_{TB}(\mathbf{x}_{TB}) = (f_{t,b}(\mathbf{x}_{t,b}), ..., f_{T,B}(\mathbf{x}_{T,B}))\,, \forall b = 1....B, \forall t = t...T$ for the time variant GP f. Using the result presented in [25], we can adapt the bound on the time-varying information gain into the parallel setting using a population size of B below.

Lemma 12. *(adapted from [25] with a batch size B) Let ω be the forgetting-remembering trade-off parameter and consider the kernel for time $1 - K_{time}(t,t') \leq \omega|t - t'|$, we bound the maximum information gain that*

$$\tilde{\gamma}_{TB} \leq \left(\frac{T}{\tilde{N}\times B} + 1\right)\left(\gamma_{\tilde{N}\times B} + \sigma_f^{-2}\left[\tilde{N}\times B\right]^3\omega\right).$$

Uncertainty Sampling (US). We next derive an upper bound over the maximum information gain obtained from a batch $\mathbf{x}_{t,b}, \forall b = 1, ..., B$. In other words, we want to show that the information gain by our chosen points $\mathbf{x}_{t,b}$ will not go beyond the ones by maximizing the uncertainty. For this, we define an uncertainty sampling (US) scheme which fills in a batch $\mathbf{x}_{t,b}^{\text{US}}$ by maximizing the GP predictive variance. Particularly, at iteration t, we select $\mathbf{x}_{t,b}^{\text{US}} = \arg\max_{\mathbf{x}}\sigma_t(\mathbf{x} \mid D_{t,b-1}), \forall b \leq B$ and

the data set is augmented over time to include the information of the new point, $D_{t,b} = D_{t,b-1} \cup \mathbf{x}_{t,b}^{\text{US}}$. We note that we use $\mathbf{x}_{t,b}^{\text{US}}$ to derive the upper bound, but this is not used by our PB2 algorithm.

Lemma 13. *Let $\mathbf{x}_{t,b}^{PB2}$ be the point chosen by our algorithm and $\mathbf{x}_{t,b}^{US}$ be the point chosen by uncertainty sampling (US) by maximizing the GP predictive variance $\mathbf{x}_{t,b}^{US} = \arg\max_{\mathbf{x}\in D} \sigma_t(\mathbf{x} \mid D_{t,b-1}), \forall b = 1, ...B$ and $D_{t,b} = D_{t,b-1} \cup \mathbf{x}_{t,b}$. We have*

$$\sigma_{t+1,1}\left(\mathbf{x}_{t+1,1}^{PB2}\right) \leq \sigma_{t+1,1}\left(\mathbf{x}_{t+1,1}^{US}\right) \leq \sigma_{t,b}\left(\mathbf{x}_{t,b}^{US}\right), \forall t \in \{1, ..., T\}, \forall b \in \{1, ...B\}.$$

Proof. The first inequality is straightforward that the point chosen by uncertainty sampling will have the highest uncertainty $\sigma_{t+1,1}\left(\mathbf{x}_{t+1,1}^{\text{PB2}}\right) \leq \sigma_{t+1,1}\left(\mathbf{x}_{t+1,1}^{\text{US}}\right) = \arg\max_{\mathbf{x}} \sigma_t(\mathbf{x} \mid D_{t,b-1})$.

The second inequality is obtained by using the principle of "information never hurts" [143], we know that the GP uncertainty for all locations $\forall \mathbf{x}$ decreases with observing a new point. Therefore, the uncertainty at the future iteration σ_{t+1} will be smaller than that of the current iteration σ_t, i.e., $\sigma_{t+1,b}\left(\mathbf{x}_{t+1,b}^{\text{US}}\right) \leq \sigma_{t,b}\left(\mathbf{x}_{t,b}^{\text{US}}\right), \forall b \leq B, \forall t \leq T$. We thus conclude the proof $\sigma_{t+1,1}\left(\mathbf{x}_{t+1,1}^{\text{US}}\right) \leq \sigma_{t,b}\left(\mathbf{x}_{t,b}^{\text{US}}\right), \forall t \in \{1, ..., T\}, \forall b \in \{1, ...B\}$. □

Lemma 14. *The sum of variances of the points selected by the our PB2 algorithm $\sigma()$ is bounded by the sum of variances by uncertainty sampling $\sigma^{US}()$. Formally, w.h.p., $\sum_{t=2}^{T} \sigma_{t,1}(\mathbf{x}_{t,1}) \leq \frac{1}{B}\sum_{t=1}^{T}\sum_{b=1}^{B} \sigma_{t,b}\left(\mathbf{x}_{t,b}^{US}\right)$.*

Proof. By the definition of uncertainty sampling in Lem. 13, we have $\sigma_{t+1,1}(\mathbf{x}_{t+1,1}) \leq \sigma_{t,b}\left(\mathbf{x}_{t,b}^{\text{US}}\right), \forall t \in \{1, ..., T\}, \forall b \in \{2, ...B\}$ and $\sigma_{t,1}(\mathbf{x}_{t,1}) \leq \sigma_{t,1}\left(\mathbf{x}_{t,1}^{\text{US}}\right)$ where $\mathbf{x}_{t,1}$ is the point chosen by our PB2 and $\mathbf{x}_{t,1}^{\text{US}}$ is from uncertainty sampling. Summing all over B, we obtain

$$\sigma_{t,1}(\mathbf{x}_{t,1}) + (B-1)\,\sigma_{t+1,1}(\mathbf{x}_{t+1,1}) \leq \sigma_{t,1}\left(\mathbf{x}_{t,1}^{\text{US}}\right) + \sum_{b=2}^{B} \sigma_{t,b}\left(\mathbf{x}_{t,b}^{\text{US}}\right)$$

$$\sum_{t=1}^{T} \sigma_{t,1}(\mathbf{x}_{t,1}) + (B-1)\sum_{t=1}^{T} \sigma_{t+1,1}(\mathbf{x}_{t+1,1}) \leq \sum_{t=1}^{T}\sum_{b=1}^{B} \sigma_{t,b}\left(\mathbf{x}_{t,b}^{\text{US}}\right) \qquad \text{by summing over } T$$

$$\sum_{t=2}^{T} \sigma_{t,1}(\mathbf{x}_{t,1}) \leq \frac{1}{B}\sum_{t=1}^{T}\sum_{b=1}^{B} \sigma_{t,b}\left(\mathbf{x}_{t,b}^{\text{US}}\right).$$

The last equation is obtained because of $\sigma_{1,1}\left(\mathbf{x}_{1,1}\right) \geq 0$ and $(B-1)\,\sigma_{T+1,1}\left(\mathbf{x}_{T+1,1}\right) \geq 0$. □

Lemma 15. *Let* $C_1 = \frac{32}{\log\left(1+\sigma_f^{-2}\right)}$, σ_f^2 *be the measurement noise variance and* $\tilde{\gamma}_{TB} := \max \tilde{\mathbf{I}}$ *be the maximum information gain of time-varying kernel, we have* $\sum_{t=1}^{T}\sum_{b=1}^{B}\sigma_{t,b}^2(\mathbf{x}_{t,b}^{US}) \leq \frac{C_1}{16}\tilde{\gamma}_{TB}$ *where* $\mathbf{x}_{t,b}^{US}$ *is the point selected by uncertainty sampling (US).*

Proof. We show that $\sigma_{t,b}^2(\mathbf{x}_{t,b}^{\mathrm{US}}) = \sigma_f^2\left(\sigma_f^{-2}\sigma_{t,b}^2(\mathbf{x}_{t,b}^{\mathrm{US}})\right) \leq \sigma_f^2 C_2 \log\left(1+\sigma_f^{-2}\sigma_{t,b}^2\left(\mathbf{x}_{t,b}^{\mathrm{US}}\right)\right), \forall b \leq B, \forall t \leq T$ where $C_2 = \frac{\sigma_f^{-2}}{\log\left(1+\sigma_f^{-2}\right)} \geq 1$ and σ_f^2 is the measurement noise variance. We have the above inequality because $s^2 \leq C_2 \log\left(1+s^2\right)$ for $s \in \left[0, \sigma_f^{-2}\right]$ and $\sigma_f^{-2}\sigma_{t,b}^2\left(\mathbf{x}_{t,b}^{\mathrm{US}}\right) \leq \sigma^{-2}k\left(\mathbf{x}_{t,b}^{\mathrm{US}}, \mathbf{x}_{t,b}^{\mathrm{US}}\right) \leq \sigma_f^{-2}$. We then use Lemma 5.3 of [55] to have the information gain over the points chosen by a time-varying kernel $\tilde{\mathbf{I}} = \frac{1}{2}\sum_{t=1}^{T}\sum_{b=1}^{B}\log\left(1+\sigma_f^{-2}\sigma_{t,b}^2\left(\mathbf{x}_{t,b}^{\mathrm{US}}\right)\right)$. Finally, we obtain

$$\sum_{t=1}^{T}\sum_{b=1}^{B}\sigma_{t,b}^2(\mathbf{x}_{t,b}^{\mathrm{US}}) \leq \sigma_f^2 C_2 \sum_{t=1}^{T}\sum_{b=1}^{B}\log\left(1+\sigma_f^{-2}\sigma_{t,b}^2\left(\mathbf{x}_{t,b}^{\mathrm{US}}\right)\right) = 2\sigma_f^2 C_2 \tilde{\mathbf{I}} = \frac{C_1}{16}\tilde{\gamma}_{TB}$$

where $C_1 = \frac{2}{\log\left(1+\sigma_f^{-2}\right)}$ and $\tilde{\gamma}_{TB} := \max \tilde{\mathbf{I}}$ is the definition of maximum information gain given by $T \times B$ data points from a GP for a specific time-varying kernel. □

Theorem 16. *Let the domain* $\mathcal{D} \subset [0,r]^d$ *be compact and convex where* d *is the dimension and suppose that the kernel is such that* $f \sim GP(0,k)$ *is almost surely continuously differentiable and satisfies Lipschitz assumptions for some* a, b. *Fix* $\delta \in (0,1)$ *and set* $\beta_T = 2\log\frac{\pi^2 T^2}{2\delta} + 2d\log rdbT^2\sqrt{\log\frac{da\pi^2T^2}{2\delta}}$. *Defining* $C_1 = 32/\log(1+\sigma_f^2)$, *the PB2 algorithm satisfies the following regret bound after* T *time steps:*

$$R_{TB} = \sum_{t=1}^{T} f_t(\mathbf{x}_t^*) - \max_{b=1,\ldots,B} f_t(\mathbf{x}_{t,b}) \leq \sqrt{C_1 T \beta_T\left(\frac{T}{\tilde{N}B}+1\right)\left(\gamma_{\tilde{N}B} + \left[\tilde{N}B\right]^3\omega\right)} + 2$$

with probability at least $1-\delta$, *the bound holds for any* $\tilde{N} \in \{1,\ldots,T\}$ *and* $B \ll T$.

Proof. Let $\mathbf{x}_t^* = \arg\max_{\forall \mathbf{x}} f_t(\mathbf{x})$ and $\mathbf{x}_{t,b}$ be the point chosen by our algorithm at iteration t and batch element b, we define the (time-varying) instantaneous regret

as $r_{t,b} = f_t(\mathbf{x}_t^*) - f_t(\mathbf{x}_{t,b})$ and the (time-varying) batch instantaneous regret over B points is as follows:

$$
\begin{aligned}
r_t^B &= \min_{b \le B} r_{t,b} = \min_{b \le B} f_t(\mathbf{x}_t^*) - f_t(\mathbf{x}_{t,b}), \forall b \le B \\
&\le f_t(\mathbf{x}_t^*) - f_t(\mathbf{x}_{t,1}) \le \mu_t(\mathbf{x}_t^*) + \sqrt{\kappa_t}\sigma_t(\mathbf{x}_t^*) + \frac{1}{t^2} - f_t(\mathbf{x}_{t,1}) && \text{by Lem. 10} \\
&\le \mu_t(\mathbf{x}_{t,1}) + \sqrt{\kappa_t}\sigma_t(\mathbf{x}_{t,1}) + \frac{1}{t^2} - f_t(\mathbf{x}_{t,1}) \le 2\sqrt{\kappa_t}\sigma_t(\mathbf{x}_{t,1}) + \frac{1}{t^2} && \text{(A.6)}
\end{aligned}
$$

where we have used the property that $\mu_t(\mathbf{x}_{t,1}) + \sqrt{\beta_t}\sigma_t(\mathbf{x}_{t,1}) \ge \mu_t(\mathbf{x}_t^*) + \sqrt{\beta_t}\sigma_t(\mathbf{x}_t^*)$ by the definition of selecting $\mathbf{x}_{t,1} = \arg\max_{\mathbf{x}} \mu_t(\mathbf{x}) + \sqrt{\beta_t}\sigma_t(\mathbf{x})$. Next, we bound the cumulative batch regret as:

$$
\begin{aligned}
R_{TB} &= \sum_{t=1}^{T} r_t^B \le \sum_{t=1}^{T} \left(2\sqrt{\kappa_t}\sigma_t(\mathbf{x}_{t,1}) + \frac{1}{t^2}\right) && \text{by Eq. (A.6)} \\
&\le 2\sqrt{\kappa_T}\sigma_1(\mathbf{x}_{1,1}) + \frac{2\sqrt{\kappa_T}}{B}\sum_{t=1}^{T}\sum_{b=1}^{B}\sigma_{t,b}\left(\mathbf{x}_{t,b}^{\text{US}}\right) + \sum_{t=1}^{T}\frac{1}{t^2} && \text{by Lem. 13 and } \kappa_T \ge \kappa_t, \forall t \le T \\
&\le \frac{4\sqrt{\kappa_T}}{B}\sum_{t=1}^{T}\sum_{b=1}^{B}\sigma_{t,b}\left(\mathbf{x}_{t,b}^{\text{US}}\right) + \sum_{t=1}^{T}\frac{1}{t^2} && \text{(A.7)} \\
&\le \frac{4}{B}\sqrt{\kappa_T \times TB\sum_{t=1}^{T}\sum_{b=1}^{B}\sigma_{t,b}^2(\mathbf{x}_{t,b}^{\text{US}})} + 2 \le \sqrt{C_1\frac{T}{B}\kappa_T\tilde{\gamma}_{TB}} + 2 && \text{(A.8)} \\
&\le \sqrt{C_1\frac{T}{B}\kappa_T\left(\frac{T}{\tilde{N}B}+1\right)\left(\gamma_{\tilde{N}B} + \frac{1}{\sigma_f^2}\left[\tilde{N}B\right]^3\omega\right)} + 2 && \text{(A.9)}
\end{aligned}
$$

where $C_1 = 32/\log(1+\sigma_f^2)$, $\mathbf{x}_{t,b}^{\text{US}}$ is the point chosen by uncertainty sampling – used to provide the upper bound in the uncertainty. In Eq. (A.7), we take the upper bound by considering two possible cases: either $\sigma_1(\mathbf{x}_{1,1}) \ge \frac{1}{B}\sum_{t=1}^{T}\sum_{b=1}^{B}\sigma_{t,b}\left(\mathbf{x}_{t,b}^{\text{US}}\right)$ or $\frac{1}{B}\sum_{t=1}^{T}\sum_{b=1}^{B}\sigma_{t,b}\left(\mathbf{x}_{t,b}^{\text{US}}\right) \ge \sigma_1(\mathbf{x}_{1,1})$. It results in:

$$
\frac{2}{B}\sum_{t=1}^{T}\sum_{b=1}^{B}\sigma_{t,b}\left(\mathbf{x}_{t,b}^{\text{US}}\right) \ge \frac{1}{B}\sum_{t=1}^{T}\sum_{b=1}^{B}\sigma_{t,b}\left(\mathbf{x}_{t,b}^{\text{US}}\right) + \sigma_1(\mathbf{x}_{1,1}).
$$

In Eq. (A.8) we have used $\sum_{t=1}^{\infty}\frac{1}{t^2} \le \pi^2/6 \le 2$ and $||z||_1 \le \sqrt{T}||z||_2$ for any vector $z \in \mathcal{R}^T$. In Eq. (A.9), we utilize Lem. 12.

Finally, given the squared exponential (SE) kernel defined, $\gamma_{\tilde{N}B}^{SE} = \mathcal{O}(\left[\log \tilde{N}B\right]^{d+1})$, the bound is $R_{TB} \le \sqrt{C_1\frac{T}{B}\beta_T\left(\frac{T}{\tilde{N}B}+1\right)\left((d+1)\log\left(\tilde{N}B\right) + \frac{1}{\sigma_f^2}\left[\tilde{N}B\right]^3\omega\right)} + 2$ where $\tilde{N} \le T$ and $B \ll T$. □

A. Appendix for Population Based Bandits

In our time-varying setting, if the time-varying function is highly correlated, i.e., the information between $f_1(.)$ and $f_T(.)$ does not change significantly, we have $\omega \to 0$ and $\tilde{N} \to T$. Then, the regret bound grows sublinearly with the number of iterations T, i.e., $\lim_{T\to\infty} \frac{R_{TB}}{T} = 0$. This bound suggests that the gap between $f_t(\mathbf{x}_t)$ and the optimal $f_t(\mathbf{x}_t^*)$ vanishes asymptotically using PB2. In addition, our regret bound is tighter and better with increasing batch size B.

On the other hand in the worst case, if the time-varying function is not correlated, such as $\tilde{N} \to 1$ and $\omega \to 1$, then PB2 achieves the linear regret [25].

A.3 PB2-Mix Theoretical Results

Table A.5: Notation used in the theoretical analysis.

Variable	Domain	Meaning
C	$\mathcal{N}$	number of categories (number of arms)
T	$\mathcal{N}$	maximum number of bandit update (the number of t_{ready} in PB2)
V	$\mathcal{N}$	number of time segments, how many times the function has been shifted
B	$\mathcal{N}$	batch size (number of parallel agents)
S_t	*list*	a list of B selected categories $[c_{t,1}, c_{t,2}, ..., c_{t,B}]$
e	$\mathcal{R}^+$	this is Euler's number 2.71828
$\left[p_t^1, ...p_t^C\right]$	*list*	probability vector at iteration t and category $c = 1....C$
w_c	$\mathcal{R}$	weight for categorical c (or arm c)
$W_t = \sum_{c=1}^{C} w_c$	$\mathcal{R}^+$	sum of the weight vector at iteration t
$c_{t,b} \in \{1...C\}$	$\mathcal{N}$	a selection at iteration t at agent b
$c_t = [c_{t,1}, ..., c_{t,B}]$	*list*	a batch of B categorical selections at iteration t
$c_t^* \in \{1...C\}$	*list*	an optimal selection at iteration t
A_v^*	*list*	a list of B elements taking the (same) optimal selection c_t^* at iteration/segment v
$g_t(c)$	$[0, 1]$	a gain (reward) occurred at iteration t by pulling an arm c
$\hat{g}_t(c) = \frac{g_t(c)}{p_t^c + \gamma}(c = h_t)$	$\mathcal{R}^+$	a normalized gain
$\alpha = \frac{1}{T}, \eta = 2\gamma = \sqrt{\frac{2 \ln C}{CT}}$	$\mathcal{R}^+$	hyperparameters, set by Theorem 3
$\gamma = \sqrt{\frac{\ln C}{2CT}} \in [0, 1]$	$\mathcal{R}^+$	is the exploration parameter
$G_T(c) = \sum_{t=1}^{T} g_t(c)$	$\mathcal{R}^+$	a total gain if we select an arm c entirely

A.3.1 Time-varying EXP3 Multiple play (TV.EXP3.M)

We present a new algorithm for parallel (or multiple play) multi-armed bandits in the time-varying setting with adversarial feedback. Particularly, we extend the multiple play EXP3 algorithm (or EXP3.M, Uchiya *et al.* [306]), to the time-varying setting where the unknown reward distribution of each arm can change arbitrarily, but the total number of change points is no more than V times. We refer to Table A.5 for the notations used in the proofs.

We summarize the TV.EXP3.M in Algorithm 13 – this is complementary to the brief Algorithm 2 in the main paper. In Algorithm 13, we maintain a set of probability vectors for each arm which will be specified and weighted in the optimal way derived by the theory. Then, at each round we select a batch of B arms for parallel evaluations, observe the reward and update the model. The

normalization steps are to prevent from being biased toward a single arm which performs overwhelmingly well, thus overly exploiting.

Algorithm 13 TV.EXP3.M algorithm

Input: $\gamma = \sqrt{\frac{C\ln(C/B)}{(e-1)BT}}$, $\alpha = \frac{1}{T}$,C #categorical choice, T #max iteration, B # multiple play choice
Initialize: $\omega_c = 1, \forall c = 1...C$ and denote $\eta = (\frac{1}{B} - \frac{\gamma}{C})\frac{1}{1-\gamma}$
for $t = 1$ *to* T **do**
  if $\arg\max_{c\in[C]} \omega_c \geq \eta \sum_{c=1}^{C} \omega(c)$ **then**
    ν s.t. $\frac{\nu}{\eta} = \sum_{\omega_t(c)\geq\nu} \nu + \sum_{\omega_t(c)<\omega_t(c)} \omega_t(c)$
    Set $S_0 = (c : \omega_t(c) \geq \nu)$ and $\omega_t(S_0) = \nu$
  else
    Set $S_0 = \emptyset$
  end
  1. Compute $p_t^c = B\left((1-\gamma)\frac{\omega_c}{\sum_{c=1}^{C}\omega_c} + \frac{\gamma}{C}\right), \forall c$
  2. Select a batch of B choices: $S_t = [c_t^1, c_t^2, ..., c_t^B] = \text{DepRound}\left(B, \left[p_t^1 p_t^2 ....p_t^C\right]\right)$
  3. Observe the reward after evaluating S_t
  4. $\hat{g}_t(c) = \frac{g_t(c)}{p_t^c}, \forall c \in S_t$ and $\hat{g}_t(c) = 0$ otherwise
  5. $\forall c \notin S_0$: update $\omega_c = \omega_c \times \exp\left(B\gamma\hat{g}_t(c)/C\right) + \frac{e\alpha}{C}\sum_{i=1}^{C}\omega_c$,
  6. $\forall c = S_0$: update $\omega_c = \omega_c + \frac{e\alpha}{C}\sum_{i=1}^{C}\omega_c$
end

Next we will provide proofs for the main theoretical results in the paper.

Theorem 17. *(Theorem 3 in the main paper) Let $T > 0$, $C > 0$, set $\alpha = \frac{1}{T}$ and $\gamma = \min\left\{1, \sqrt{\frac{C\ln(C/B)}{(e-1)BT}}\right\}$, we assume the reward distributions to change at arbitrary time instants, but the total number of change points is no more than V times. The expected regret gained by TV.EXP3.M in a batch satisfies the following sublinear regret bound*

$$\mathbb{E}\left[R_{TB}\right] \leq [1 + e + V]\sqrt{(e-1)\frac{CT}{B}\ln\frac{CT}{B}}.$$

Proof. We refer to Table A.5 for notations. We follow the proof technique presented in [11] and [306] to derive the regret bound. Let $W_t = \sum_{c=1}^{C}\omega_t^c$ and using step 9 in

A. Appendix for Population Based Bandits

Algorithm 13, we have $\frac{\frac{p_t^c}{B}-\frac{\gamma}{C}}{1-\gamma} = \frac{\omega_t^c}{W_t}$. This equation will be used below.

$$
\begin{aligned}
\frac{W_{t+1}}{W_t} &= \frac{\sum_{c=1}^{C}\omega_{t+1}^c}{W_t} = \frac{\sum_{c\notin S_t(0)}\omega_{t+1}^c}{W_t} + \frac{\sum_{c\in S_t(0)}\omega_{t+1}^c}{W_t} + \frac{\sum_{c=1}^{C}\frac{e\alpha W_t}{C}}{W_t} \\
&= \sum_{c\notin S_t(0)}\frac{\omega_t^c}{W_t}\times\exp\left(\frac{B\gamma}{C}\hat{g}_t(c)\right) + \frac{\sum_{c\in S_t(0)}\omega_t^c}{W_t} + e\alpha \\
&\leq e\alpha + \sum_{c\notin S_t(0)}\frac{\omega_t^c}{W_t}\left[1+\frac{B\gamma}{C}\hat{g}_t(c) + (e-2)\left(\frac{B\gamma}{C}\right)^2\hat{g}_t^2(c)\right] + \frac{\sum_{c\in S_t(0)}\omega_{t+1}^c}{W_t} \quad \text{by } e^x \leq 1+x+(e-2)x^2 \\
&= e\alpha + \underbrace{\sum_{c\notin S_t(0)}\frac{\omega_t^c}{W_t} + \sum_{c\in S_t(0)}\frac{\omega_{t+1}^c}{W_t}}_{1} + \sum_{c\notin S_t(0)}\frac{\frac{p_t^c}{B}-\frac{\gamma}{C}}{1-\gamma}\left[\frac{B\gamma}{C}\hat{g}_t(c) + (e-2)\left(\frac{B\gamma}{C}\right)^2\hat{g}_t^2(c)\right] \\
&\leq e\alpha + 1 + \frac{B\gamma}{C(1-\gamma)}\sum_{c\notin S_t(0)}\left(\frac{p_t^c}{B}-\frac{\gamma}{C}\right)\hat{g}_t(c) + \frac{(e-2)}{1-\gamma}\left(\frac{B\gamma}{C}\right)^2\sum_{c\notin S_t(0)}\left(\frac{p_t^c}{B}-\frac{\gamma}{C}\right)\hat{g}_t^2(c) \\
&= e\alpha + 1 + \frac{\gamma}{C(1-\gamma)}\sum_{c\notin S_t(0)}p_t^c\hat{g}_t(c) - \frac{B\gamma^2}{C^2(1-\gamma)}\sum_{c\notin S_t(0)}\hat{g}_t(c) \\
&+ \frac{(e-2)\gamma^2 B}{(1-\gamma)C^2}\sum_{c\notin S_t(0)}p_t^c\hat{g}_t^2(c) - \frac{(e-2)\gamma^3B^2}{(1-\gamma)C^2}\sum_{c\notin S_t(0)}\hat{g}_t^2(c) \\
&\leq e\alpha + 1 + \frac{\gamma}{C(1-\gamma)}\sum_{c\notin S_t(0)}p_t^c\hat{g}_t(c) + \frac{(e-2)\gamma^2B}{(1-\gamma)C^2}\sum_{c\notin S_t(0)}p_t^c\hat{g}_t^2(c) \\
&\leq e\alpha + 1 + \frac{\gamma}{C(1-\gamma)}\sum_{c\in S_t - S_t(0)}g_t(c) + \frac{(e-2)\gamma^2B}{(1-\gamma)C^2}\sum_{c\notin S_t(0)}\hat{g}_t(c) \quad \text{by } x^2\leq x, \forall x\in[0,1] \\
\frac{W_{t+1}}{W_t} &\leq e\alpha + 1 + \frac{\gamma}{C(1-\gamma)}\sum_{c\in S_t - S_t(0)}g_t(c) + \frac{(e-2)\gamma^2B}{(1-\gamma)C^2}\sum_{c\in[C]}\hat{g}_t(c). \qquad \text{(A.10)}
\end{aligned}
$$

To demonstrate the time-varying property in our bandit problem, we assume the reward distributions change at arbitrary time instants, but the total number of change points is no more than V times. We split the sequence of total decisions T into V segments such that within each segment we have the *time-invariant* reward function.

We can write V segments as $[T_1, ...T_2), [T_2, ..., T_3), [T_V, ..., T_{V+1})$ where T_v indicates the starting index of the v-th segment, T_{v+1} is the ending index of the v-th segment which is also the starting index of the $v+1$-th segment – using the same notation in EXP3.S [11]. Similarly, we can write the optimal sequence $\underbrace{[c_{T_1}^*, ...c_{T_2}^*)}_{=c_{T_1}^*}, \underbrace{[c_{T_2}^*, ..., c_{T_3}^*)}_{=c_{T_2}^*}, \underbrace{[c_{T_V}^*, ..., c_{T_{V+1}}^*)}_{=c_{T_V}^*}$ where the reward function does not change in each segment, thus takes the same optimal choice $c_{T_v}^*$.

We consider an arbitrary segment v and denote the length $\Delta_v = T_{v+1} - T_v$. Furthermore, let define the cumulative gain (or reward) achieved by using TV.EXP3.M strategy within a segment v that the indices are ranging from T_v to $T_{v+1}-1$

$$G_{TV.EXP3.M}(v) = \sum_{t=T_v}^{T_{v+1}-1}\sum_{c\in S_t} g_t(c_t = c).$$

A. Appendix for Population Based Bandits

Taking ln of Eqn. (A.10), we get:

$$\begin{aligned}\ln \frac{W_{t+1}}{W_t} &\leq \ln\left(e\alpha + 1 + \frac{\gamma/C}{1-\gamma}\sum_{c\in S_t - S_t(0)} g_t(c) + \frac{(e-2)\left(\frac{\gamma}{C}\right)^2 B}{1-\gamma}\sum_{c=1}^{C}\hat{g}_t(c)\right) \\ &\leq e\alpha + \frac{\gamma/C}{1-\gamma}\sum_{c\in S_t - S_t(0)} g_t(c) + \frac{(e-2)\left(\frac{\gamma}{C}\right)^2 B}{1-\gamma}\sum_{c=1}^{C}\hat{g}_t(c) \qquad \text{by } 1+a\leq e^a.\end{aligned}$$

Summing over all indices $t = T_v, ..., T_{v+1} - 1$ within a segment v-th:

$$\begin{aligned}\ln W_{T_{v+1}} - \ln W_{T_v} &\leq \Delta_v e\alpha + \frac{\gamma/C}{1-\gamma}\sum_{t=T_v}^{T_{v+1}-1}\sum_{c\in S_t - S_t(0)} g_t(c) + \frac{(e-2)\left(\frac{\gamma}{C}\right)^2 B}{1-\gamma}\sum_{t=T_v}^{T_{v+1}-1}\sum_{c=1}^{C}\hat{g}_t(c) \\ &\leq \Delta_v e\alpha + \frac{\gamma/C}{1-\gamma}\sum_{t=T_v}^{T_{v+1}-1}\sum_{c\in S_t - S_t(0)} g_t(c) + \frac{(e-2)\left(\frac{\gamma}{C}\right)^2 B}{1-\gamma}\sum_{t=T_v}^{T_{v+1}-1}\sum_{c=1}^{C}\hat{g}_t(c).\end{aligned}$$

Let $j = c^*_{T_v} = ... = c^*_{T_{v+1}-1}$ be the optimal choice in the sequence v, we have $\omega_j^{(T_{v+1})} = \omega_j^{(T_{v+1}-1)} \times \exp\left(\frac{B\gamma}{C}\hat{g}_t(j)\right) + \frac{e\alpha}{C}W_{T_{v+1}-1}, \forall j \notin S_{T_{v+1}-1}(0)$ and $\omega_j^{(T_{v+1})} = \omega_j^{(T_{v+1}-1)}, \forall j \in S_{T_{v+1}-1}(0)$

$$\begin{aligned}\omega_j^{(T_{v+1})} &= \omega_j^{(T_{v+1}-1)} \times \exp\left(\frac{B\gamma}{C}\hat{g}_t(j)\right) + \frac{e\alpha}{C}W_{T_{v+1}-1} \\ &\geq \omega_j^{(T_{v+1}-1)} \times \exp\left(\frac{B\gamma}{C}\hat{g}_t(j)\right) \\ &\geq \omega_j^{(T_{v+1}-2)} \times \exp\left(\frac{B\gamma}{C}\hat{g}_t(j)\right)\exp\left(\frac{B\gamma}{C}\hat{g}_{(T_{v+1}-2)}(j)\right) \\ &\geq \omega_j^{(T_v)} \times \exp\left(\frac{B\gamma}{C}\sum_{t=T_v|t:j\notin S_t(0)}^{T_{v+1}-1}\hat{g}_t(j)\right) \\ &\geq \frac{e\alpha}{C}W_{T_v} \times \exp\left(\frac{B\gamma}{C}\sum_{t=T_v|t:j\notin S_t(0)}^{T_{v+1}-1}\hat{g}_t(j)\right) \\ &\geq \frac{\alpha}{C}W_{T_v} \times \exp\left(\frac{B\gamma}{C}\sum_{t=T_v|t:j\notin S_t(0)}^{T_{v+1}-1}\hat{g}_t(j)\right).\end{aligned}$$

We now consider the lower bound by taking the optimal (batch) set $A^*_v \subset [C]$ for B elements with the maximum total of reward within the segment v: $\sum_{c\in A^*_v}\sum_{t=1}^{T} g_t(c)$. Since we have the fact that $\sum_{j\in A^*_v}\omega_j^{(T_{v+1})} \geq B\left(\prod_{j\in A^*_v}\omega_j^{(T_{v+1})}\right)^{1/B}$ by the Cauchy

A. Appendix for Population Based Bandits

Schwarz inequality, we continue the above formula

$$\begin{aligned}
\ln\left(W_{T_{v+1}-1}\right) - \ln W_{T_v} \geq & \ln B + \frac{1}{B}\sum_{j\in A_v^*} \ln \omega_j^{(T_{v+1})} - \ln W_{T_v} \\
= & \ln B + \frac{1}{B}\sum_{j\in A_v^*} \ln \frac{\alpha}{C} W_{T_v} + \frac{1}{B}\sum_{j\in A_v^*} \frac{B\gamma}{C} \sum_{t=T_v|t:j\notin S_t(0)}^{T_{v+1}-1} \hat{g}_t(j) - \ln W_{T_v} \\
= & \ln B + \frac{1}{B}\sum_{j\in A_v^*} \ln \frac{\alpha}{C} + \sum_{j\in A_v^*} \frac{\gamma}{C} \sum_{t=T_v|t:j\notin S_t(0)}^{T_{v+1}-1} \hat{g}_t(j) \\
= & \ln \frac{B\alpha}{C} + \sum_{j\in A_v^*} \frac{\gamma}{C} \sum_{t=T_v|t:j\notin S_t(0)}^{T_{v+1}-1} \hat{g}_t(j).
\end{aligned}$$

Combining both the lower bound and upper bound, we get

$$\begin{aligned}
& \ln \frac{B\alpha}{C} + \sum_{j\in A_v^*} \frac{\gamma}{C} \sum_{t=T_v|t:j\notin S_t(0)}^{T_{v+1}-1} \hat{g}_t(j) \leq \Delta_v e\alpha + \frac{\gamma/C}{1-\gamma} \sum_{t=T_v}^{T_{v+1}-1} \sum_{c\in S_t - S_t(0)} g_t(c) \\
& + \frac{(e-2)\left(\frac{\gamma}{C}\right)^2 B}{1-\gamma} \sum_{t=T_v}^{T_{v+1}-1} \sum_{c=1}^{C} \hat{g}_t(c).
\end{aligned}$$

Summing over all segments $v = 1....V$

$$\begin{aligned}
V \ln \frac{B\alpha}{C} + \sum_{v=1}^{V} \sum_{j\in A_v^*} \frac{\gamma}{C} \sum_{t=T_v|t:j\notin S_t(0)}^{T_{v+1}-1} \hat{g}_t(j) \leq & Te\alpha + \frac{\gamma/C}{1-\gamma} \sum_{v=1}^{V} \sum_{t=T_v}^{T_{v+1}-1} \sum_{c\in S_t - S_t(0)} g_t(c) \\
& + \frac{(e-2)\left(\frac{\gamma}{C}\right)^2 B}{1-\gamma} \sum_{v=1}^{V} \sum_{t=T_v}^{T_{v+1}-1} \sum_{c=1}^{C} \hat{g}_t(c) \\
\frac{VC}{\gamma} \ln \frac{B\alpha}{C} + \sum_{v=1}^{V} \sum_{j\in A_v^*} \sum_{t=T_v|t:j\notin S_t(0)}^{T_{v+1}-1} \hat{g}_t(j) \leq & \frac{C}{\gamma} Te\alpha + \frac{1}{1-\gamma} \sum_{v=1}^{V} \sum_{t=T_v}^{T_{v+1}-1} \sum_{c\in S_t - S_t(0)} g_t(c) \\
& + \frac{(e-2)\left(\frac{\gamma B}{C}\right)}{1-\gamma} \sum_{v=1}^{V} \sum_{t=T_v}^{T_{v+1}-1} \sum_{c=1}^{C} \hat{g}_t(c) \\
\sum_{v=1}^{V} \sum_{t=T_v|j\in S_t(0)}^{T_{v+1}-1} \sum_{j\in A_v^*} g_t(j) + \sum_{v=1}^{V} \sum_{j\in A_v^*} \sum_{t=T_v|j\notin S_t(0)}^{T_{v+1}-1} \hat{g}_t(j) \leq & \frac{C}{\gamma} Te\alpha + \frac{1}{1-\gamma} \sum_{v=1}^{V} \sum_{t=T_v}^{T_{v+1}-1} \sum_{c\in S_t(0)} g_t(c) \\
& + \frac{1}{1-\gamma} \sum_{v=1}^{V} \sum_{t=T_v}^{T_{v+1}-1} \sum_{c\in S_t - S_t(0)} g_t(c) \\
& + \frac{(e-2)\left(\frac{\gamma B}{C}\right)}{1-\gamma} \sum_{v=1}^{V} \sum_{t=T_v}^{T_{v+1}-1} \sum_{c=1}^{C} \hat{g}_t(c) \\
& - \frac{VC}{\gamma} \ln \frac{B\alpha}{C}. \qquad \text{(A.11)}
\end{aligned}$$

Let us denote the optimal gain over all iterations and all parallel agents $G_{TB}^* = \sum_{v=1}^{V} \sum_{t=T_v}^{T_{v+1}-1} \sum_{j\in A_v^*} g_t(j)$. We also denote the cumulative gain achieved by using

TV.EXP3.M algorithms $G_{\text{TV.EXP3.M}} = \sum_{v=1}^{V}\sum_{t=T_v}^{T_{v+1}-1}\sum_{c\in S_t} g_t(c)$. Then, we continue Eqn. (A.11) as

$$G^*_{TB} \leq \frac{C}{\gamma}Te\alpha + \frac{1}{1-\gamma}G_{\text{TV.EXP3.M}} + \frac{(e-2)\left(\frac{\gamma B}{C}\right)}{1-\gamma}\sum_{v=1}^{V}\sum_{t=T_v}^{T_{v+1}-1}\sum_{c=1}^{C}\hat{g}_t(c) - \frac{VC}{\gamma}\ln\frac{B\alpha}{C} \tag{A.12}$$

where the number of element in a batch $|S_t| = |A^*_j| = B$. Let take expectation both sides of Eqn. (A.12), we have $\mathbb{E}\left[\hat{g}_t(c) \mid S_1, S_2, ..., S_{i-1}\right] = g_t(c)$ from the fact that DepRound [81] selects action c with probability $p_c(t)$, we obtain

$$G^*_{TB} \leq \frac{1}{(1-\gamma)}\mathbb{E}\left[G_{\text{TV.EXP3.M}}\right] + \frac{(e-2)\left(\frac{\gamma B}{C}\right)}{1-\gamma}\sum_{v=1}^{V}\sum_{t=T_v}^{T_{v+1}-1}\sum_{c=1}^{C}g_t(c) + \frac{C}{\gamma}Te\alpha - \frac{VC}{\gamma}\ln\frac{B\alpha}{C}.$$

We finally have

$$(1-\gamma)G^*_{TB} \leq \mathbb{E}\left[G_{\text{TV.EXP3.M}}\right] + (e-2)\left(\frac{\gamma B}{C}\right)\frac{C}{B}G^*_{TB} + \frac{(1-\gamma)}{\gamma}CTe\alpha - \frac{(1-\gamma)VC}{\gamma}\ln\frac{B\alpha}{C} \tag{A.13}$$

$$\begin{aligned}
G^*_{TB} - \mathbb{E}\left[G_{\text{TV.EXP3.M}}\right] \leq& (e-1)\gamma G^*_{TB} + \frac{(1-\gamma)}{\gamma}CTe\alpha + \frac{(1-\gamma)VC}{\gamma}\ln\frac{C}{B\alpha} \\
\leq& (e-1)\gamma TB + \frac{(1-\gamma)}{\gamma}CTe\alpha + \frac{VC}{\gamma}\ln\frac{C}{B\alpha} - VC\ln\frac{C}{B\alpha} \qquad \text{by } G^*_{TB} \leq TB \\
\leq& (e-1)\gamma TB + \frac{1}{\gamma}CTe\alpha + \frac{VC}{\gamma}\ln\frac{C}{B\alpha}. \qquad \text{(A.14)}
\end{aligned}$$

In Eqn. (A.13), we use the fact that:
$\sum_{v=1}^{V}\sum_{t=T_v}^{T_{v+1}-1}\sum_{c=1}^{C} g_t(c) \leq \frac{C}{B}G^*_{TB} = \frac{C}{B}\sum_{v=1}^{V}\sum_{t=T_v}^{T_{v+1}-1}\sum_{c\in A^*} g_t(c)$.

Set $\gamma = \min\left\{1, \sqrt{\frac{C\ln(C/B)}{(e-1)BT}}\right\}$ and $\alpha = \frac{1}{T}$. Then, we have the cumulative regret over all iterations and considering the best element in a batch

$$\mathbb{E}\left[R_{TB}\right] = \mathbb{E}\left[\sum_{v=1}^{V}\sum_{t=T_v}^{T_{v+1}-1}\overbrace{\max_{\forall c\leq C} g_t(c)}^{\text{optimal arms}}\right] - \mathbb{E}\left[\sum_{v=1}^{V}\sum_{t=T_v}^{T_{v+1}-1}\overbrace{\max_{\forall c\in S_t} g_t(c)}^{\text{best arm in a batch } S_t}\right] \tag{A.15}$$

$$\leq \frac{1}{B}G^*_{TB} - \frac{1}{B}\mathbb{E}\left[G_{\text{TV.EXP3.M}}\right] \tag{A.16}$$

$$\leq \frac{1}{B}\sqrt{\frac{C(e-1)\ln(C/B)}{BT}}T + \frac{1}{B}\frac{\sqrt{C(e-1)BT}}{\sqrt{\ln(C/B)}}e + \frac{1}{B}\frac{VC}{\sqrt{\frac{C\ln(C/B)}{(e-1)BT}}}\ln\frac{CT}{B} \tag{A.17}$$

$$\leq [1+e+V]\sqrt{(e-1)\frac{CT}{B}\ln\frac{CT}{B}} \tag{A.18}$$

where we obtain Eqn. (A.16) because the best gain should be greater than the average gain of a batch $\sum_{v=1}^{V}\sum_{t=T_v}^{T_{v+1}-1}\max_{\forall c\in S_t} g_t(c) \geq \frac{1}{B}\sum_{v=1}^{V}\sum_{t=T_v}^{T_{v+1}-1}\sum_{c\in S_t} g_t(c)$.

Given the number of changing points in our reward function is bounded $V \ll \sqrt{T}$ and the number of category C is a constant, our regret bound achieves sublinear regret rate with the number of iterations T, i.e., $\lim_{T\to\infty}\frac{\mathbb{E}[R_{TB}]}{T} = 0$.

□

In the above derivation, T refers to the number of bandit updates which is equivalent to the number of t_{ready} in PB2 setting [217].

A.3.2 Theoretical Derivations for PB2-Mult

We adapt Lemma 1 to handle categorical variables. We first restate some notations used in the proofs for the original PB2 (above). Let $F_t^{c_t}(x_t)$ be an objective function under a given set of continuous hyperparameters x_t and categorical variable c_t at timestep t. An example of $F_t^{c_t}(x_t)$ could be the reward for a deep RL agent. When training for a total of T steps, our goal is to maximize the final performance $F_T^{c_T}(x_T)$. We formulate this problem as optimizing the time-varying black-box reward function f_t, over $\mathcal{D}$.

Lemma 18. *Maximizing the final performance F_T of a model with respect to a given continuous hyperparameter schedule $\{x_t\}_{t=1}^T$ and a categorical hyperparameter schedule $\{c_t\}_{t=1}^T$ is equivalent to maximizing the time-varying black-box function $f_t(x_t)$ and minimizing the corresponding cumulative regret $r_t(x_t)$,*

$$\max F_T^{c_T}(x_T) = \max \sum_{t=1}^{T} f_{c_t}(x_t) = \min \sum_{t=1}^{T} r_t(x_t). \tag{A.19}$$

Proof. At each t_{ready}, we select a categorical variable $c_t \in \{1, ..., C\}$ and a continuous variable $x_t \in \mathcal{D}^{c_t}$. We would emphasize that the continuous variables are conditioned on the choice of c_t. We observe and record noisy observations, $y_t = f_{c_t}(x_t) + \epsilon_t$, where $\epsilon_t \sim \mathcal{N}(0, \sigma^2\mathbf{I})$ for some fixed σ^2. The function f_t represents the change in F_t after training for t_{ready} steps, i.e. $f_{c_t}(x_t) = F_t^{c_t}(x_t) - F_{t-t_{\text{ready}}}^{c_{t-t_{\text{ready}}}}(x_{t-t_{\text{ready}}})$. We define the best choice at each timestep as $x_t^* = \arg\max_{x_t \in \mathcal{D}^{c_t}} f_{c_t}(x_t)$, and $c_t^* = \arg\max_{c_t, x_t} f_{c_t}(x_t)$. The intermediate regret of each decision is defined as $r_t = f_{c_t^*}(x_t^*) - f_{c_t}(x_t)$ where $f_{c_t^*}(x_t^*)$ is an unknown constant.

We have a reward at the starting iteration $F_1^{c_1}(x_1)$ as a constant that allows us to write the objective function as:

$$F_T^{c_T}(x_T) - F_1^{c_1}(x_1) = \underbrace{F_T^{c_T}(x_T) - F_{T-1}^{c_{T-1}}(x_{T-1})}_{f_{c_{T-1}}(x_{T-1})} + ... + \underbrace{F_3^{c_3}(x_3) - F_2^{c_2}(x_2)}_{f_{c_2}(x_2)} + \underbrace{F_2^{c_2}(x_2) - F_1^{c_1}(x_1)}_{f_{c_1}(x_1)}. \tag{A.20}$$

Therefore, maximizing the left of Eqn. (A.20) is equivalent to minimizing the cummulative regret as follows:

$$\max\left[F_T^{c_T}(x_T) - F_1^{c_1}(x_1)\right] = \max\sum_{t=1}^{T} F_t^{c_t}(x_t) - F_{t-1}^{c_{t-1}}(x_{t-1}) = \max\sum_{t=1}^{T-1} f_{c_t}(x_t) = \min\sum_{t=1}^{T-1} r_t(x_t)$$

where we define $f_{c_t}(x_t) = F_t^{c_t}(x_t) - F_{t-1}^{c_{t-1}}(x_{t-1})$, the regret $r_t = f_{c_t^*}(x_t^*) - f_{c_t}(x_t)$. □

Next, we are going to derive the main theorem of the paper, building on the main regret bound for the original PB2 (Theorem 2).

Theorem 19. *(Theorem 4 in the main paper) Let the domain for continuous variables $\mathcal{D} \subset [0, r]^d$ be compact and convex where d is the dimension and suppose that the kernel is such that $f_t \sim GP(0, k)$ is almost surely continuously differentiable and satisfies Lipschitz assumptions $\forall L_t \geq 0, t \leq \mathcal{T}, \forall j \leq d, p(\sup\left|\frac{\partial f_t(\mathbf{x})}{\partial \mathbf{x}^{(j)}}\right| \geq L_t) \leq ae^{-(L_t/b)^2}$ for some a, b.*

Assume the reward distributions to change at arbitrary time instants, but the total number of change points is no more than V. Set $\alpha = \frac{1}{T}$, $\gamma = \min\left\{1, \sqrt{\frac{C\ln(C/B)}{(e-1)BT}}\right\}$, $\beta_T = 2\log\frac{\pi^2T^2}{2\delta} + 2d\log rdbT^2\sqrt{\log\frac{da\pi^2T^2}{2\delta}}$, pick $\delta \in (0, 1)$ and define $C_1 = 32/\log(1+\sigma_f^2)$, the PB2-Mult algorithm satisfies the following regret bound after T time steps over B parallel agents with probability at least $1 - \delta$:

$$\mathbb{E}\left[R_{TB}\right] \leq [1 + e + V]\sqrt{(e-1)\frac{CT}{B}\ln\frac{CT}{B}} + \sqrt{C_1 T_{c_t^*}\beta_{T_{c_t^*}}\left(\frac{T_{c_t^*}}{\tilde{N}B} + 1\right)\left(\gamma_{\tilde{N}B} + \left[\tilde{N}B\right]^3\omega\right)} + 2.$$

The bound holds for any $\tilde{N} \in \{1, ..., T_{c_t^*}\}$ and $V \ll \sqrt{T}$.

Proof. We expand the cumulative regret and optimize it using time-varying GP bandit optimization [25, 217]:

$$\begin{aligned} R_{TB} &= \sum_{t=1}^{T} f^* - \max_{b=1...B}\sum_{t=1}^{T} f_{c_{t,b}}(x_{t,b}) \\ &= \max_{b=1,...,B}\sum_{t=1}^{T} f_{c_{t,b}^*}(x_{t,b}) - \max_{b=1,...,B}\sum_{t=1}^{T} f_{c_{t,b}}(x_{t,b}) + \sum_{t=1}^{T} f^* - \max_{b=1,...,B}\sum_{t=1}^{T} f_{c_{t,b}^*}(x_{t,b}) \end{aligned}$$

where $b \in \{1, ..., B\}$ is an agent's index being trained in parallel, $f_{c_t^*}(x_t) = \max_{\forall c_t \in \{1,...,C\}} f_{c_t}(x_t)$ and $c_t^* = \arg\max_{\forall c \in \{1,...C\}} f_c(x_t)$ is the optimal categorical choice at iteration t.

We bound the two terms separately as follows. The first term is bounded by Theorem 3. We assume the process of generating the arm c_t's reward $f_{c_t}(.) := f_{c_t}(x_t)$

is by the "adversary" that TV.EXP3.M does not have the direct control on the selection of x_t. Particularly, x_t will be chosen by TV-GP-BUCB (as part of PB2) in Eqn. (3.3). We take the expectation of the first term to have:

$$\mathbb{E}\left[\max_{b=1...B}\sum_{t=1}^{T}\underbrace{f_{c^*_{t,b}}(.)}_{\text{pull optimal arm}}\right]-\mathbb{E}\left[\max_{b=1...B}\sum_{t=1}^{T}\underbrace{f_{c_{t,b}}(.)}_{\text{pull arm } c_t}\right]=\mathbb{E}\left[\sum_{t=1}^{T}\underbrace{f_{c^*_t}(.)}_{\text{pull optimal arm}}\right]-\mathbb{E}\left[\max_{b=1...B}\sum_{t=1}^{T}\underbrace{f_{c_{t,b}}(.)}_{\text{pull arm } c_t}\right]$$
$$=\mathbb{E}\left[\tilde{R}_{TB}\right]$$
$$\leq [1+e+V]\sqrt{(e-1)\frac{CT}{B}\ln\frac{CT}{B}}$$

where R_{TB} is cumulative regret of the TV.EXP3.M, defined in Theorem 3.

Assuming the best arm (the best categorical choice) c^*_t can be identified by TV.EXP3.M in Theorem 3. The second term is the regret bound presented in Theorem 2:

$$\sum_{t=1}^{T}f^*_t-\max_{b=1...B}\sum_{t=1}^{T}f_{c^*_{t,b}}(x_{t,b})=\sum_{t=1}^{T}f^*_t-\max_{b=1...B}\sum_{t=1}^{T}f_{c^*_t}(x_{t,b})$$
$$\leq\underbrace{\sqrt{C_1T_{c^*_t}\beta_{T_{c^*_t}}\left(\frac{T_{c^*_t}}{\tilde{N}B}+1\right)\left(\gamma_{\tilde{N}B}+\left[\tilde{N}B\right]^3\omega\right)}+2}_{\mathcal{O}(PB2)}$$

where $T_{c^*_t}$ denotes the number of times the optimal category c^*_t (or optimal arm) is selected. We follow [25, 217] to denote a block length $\tilde{N}\in\{1,....T_{c^*_t}\}$ in which the function does not change significantly. In the above equation, $\omega\in[0,1]$ is a time-varying hyperparameter which is estimated by maximizing the GP log marginal likelihood, B is a batch size or the number of population agents, $\gamma_{\tilde{N}B}$ is the maximum information gain [278] defined within a block $\tilde{N}$ over B parallel agents.

Note that the theoretical result for PB2 comes with the additional smoothness assumption on the kernel k that holds for some (a,b) and $\forall L_t\geq 0$. The kernel satisfies for all dimensions $j=1,...,d$

$$\forall L_t\geq 0, t\leq\mathcal{T}, p(\sup\left|\frac{\partial f_t(\mathbf{x})}{\partial\mathbf{x}^{(j)}}\right|\geq L_t)\leq ae^{-(L_t/b)^2}. \tag{A.21}$$

We combine the two terms to obtain the final regret bound of the PB2-Mult algorithm:

$$\mathbb{E}\left[R_{TB}\right]\leq[1+e+V]\sqrt{(e-1)\frac{CT}{B}\ln\frac{CT}{B}}+\sqrt{C_1T_{c^*_t}\beta_{T_{c^*_t}}\left(\frac{T_{c^*_t}}{\tilde{N}B}+1\right)\left(\gamma_{\tilde{N}B}+\left[\tilde{N}B\right]^3\omega\right)}+2.$$

The final regret bound is a summation of two sub-linear terms. Therefore, the regret bound grows sublinearly with the number of iterations T, i.e., $\lim_{T\to\infty}\frac{\mathbb{E}[R_{TB}]}{T}=0$. This is under a common assumption [25, 217] that the time-varying function is correlated, i.e., we have $\omega\to 0$, thus $\tilde{N}\to T$. We also note that when $T\to\infty$, then $T_{c^*_T}\to\infty$ due to Theorem 3. □

A.3.3 Kernel Hyperparameter Gradients

We optimize the GP hyperparameters by maximizing the log marginal likelihood [245]. We fit the GP hyperparameters by maximizing their posterior probability (MAP), $p\left(\sigma_x, \sigma_t \mid \mathbf{x}, \mathbf{t}, \right) \propto p\left(\sigma_x, \sigma_t, \mathbf{x}, \mathbf{t}, \right)$, which, thanks to the Gaussian likelihood, is available in closed form

$$\mathcal{L} := \ln p\left(, \mathbf{x}, \mathbf{t}, \theta\right) = \frac{1}{2}^{T}\left(+\sigma_y^2 \mathbf{I}_N\right)^{-1} - \frac{1}{2}\ln\left|+\sigma_y^2 \mathbf{I}_N\right| + \ln p_{hyp}\left(\theta\right) + \text{const} \quad \text{(A.22)}$$

where $\mathbf{I}_N$ is the identity matrix in dimension N (the number of points in the training set), and $p_{hyp}(\theta)$ is the prior over hyperparameters. We optimize Eqn. (A.22) with a gradient-based optimizer, providing the analytical gradient to the algorithm.

Our time-varying CoCaBO kernel is defined as

$$k_z(z, z') = (1-\lambda)\left(k_{xt} + k_{ct}\right) + \lambda k_{xt} k_{ct}$$

where $k_{xt} = k_{\text{Continuous}}(x, x') \times k_{\text{time1}}(t, t')$, $k_{ct} = k_{\text{Categorical}}(c, c') \times k_{\text{time2}}(t, t')$, $k_{\text{Continuous}}(x, x') = \sigma_1 \times \exp\left(-\frac{||x-x'||^2}{l}\right)$, $k_{\text{Categorical}}(c, c') = \frac{\sigma_2}{C}\sum(c, c')$, $k_{\text{time1}}(t, t') = (1-\epsilon_1)^{\frac{|t-t'|}{2}}$ and $k_{\text{time2}}(t, t') = (1-\epsilon_2)^{\frac{|t-t'|}{2}}$.

Our hyperparameters include $\theta = \{\epsilon_1, \epsilon_2, l, \sigma_1, \sigma_2, \lambda\}$. We need to compute the gradients of $\frac{\partial \mathcal{L}}{\partial \epsilon_1}, \frac{\partial \mathcal{L}}{\partial \epsilon_2}, \frac{\partial \mathcal{L}}{\partial l}, \frac{\partial \mathcal{L}}{\partial \sigma_1}, \frac{\partial \mathcal{L}}{\partial \sigma_2}, \frac{\partial \mathcal{L}}{\partial \lambda}$ as follows:

- The gradient of $\frac{\partial \mathcal{L}}{\partial \epsilon_1}$

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \epsilon_1} &= \frac{\partial \mathcal{L}}{\partial k_z} \times \frac{\partial k_z}{\partial \epsilon_1} \\
\frac{\partial k_z}{\partial \epsilon_1} &= (1-\lambda)\, k_{\text{Continuous}}(x, x') \frac{\partial k_{\text{time1}}}{\partial \epsilon_1} + \lambda k_{ct} k_{\text{Continuous}}(x, x') \frac{\partial k_{\text{time1}}}{\partial \epsilon_1} \\
\frac{\partial k_{\text{time1}}}{\partial \epsilon_1} &= -\frac{|t-t'|}{2}(1-\epsilon_1)^{\frac{|t-t'|}{2}-1}
\end{aligned}$$

- The gradient of $\frac{\partial \mathcal{L}}{\partial \epsilon_2}$

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \epsilon_2} &= \frac{\partial \mathcal{L}}{\partial k_z} \times \frac{\partial k_z}{\partial \epsilon_2} \\
\frac{\partial k_z}{\partial \epsilon_2} &= (1-\lambda)\, k_{\text{Categorical}}(c, c') \frac{\partial k_{\text{time2}}}{\partial \epsilon_2} + \lambda k_{xt} k_{\text{Categorical}}(c, c') \frac{\partial k_{\text{time2}}}{\partial \epsilon_2} \\
\frac{\partial k_{\text{time2}}}{\partial \epsilon_2} &= -\frac{|t-t'|}{2}(1-\epsilon_2)^{\frac{|t-t'|}{2}-1}
\end{aligned}$$

A. Appendix for Population Based Bandits

- The gradient of $\frac{\partial \mathcal{L}}{\partial l}$

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial l} =& \frac{\partial \mathcal{L}}{\partial k_z} \times \frac{\partial k_z}{\partial l} \\
\frac{\partial k_z}{\partial l} =& (1-\lambda)\, k_{\text{time1}}(t,t') \frac{\partial k_{\text{Continuous}}}{\partial l} + \lambda k_{ct} k_{\text{time1}}(t,t') \frac{\partial k_{\text{Continuous}}}{\partial l} \\
\frac{\partial k_{\text{Continuous}}}{\partial l} =& \frac{||x-x'||^2}{l^2} k_{\text{Continuous}}(x,x')
\end{aligned}$$

- The gradient of $\frac{\partial \mathcal{L}}{\partial \sigma_1}$

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \sigma_1} =& \frac{\partial \mathcal{L}}{\partial k_z} \times \frac{\partial k_z}{\partial \sigma_1} \\
\frac{\partial k_z}{\partial \sigma_1} =& (1-\lambda)\, k_{\text{time1}}(t,t') \frac{\partial k_{\text{Continuous}}}{\partial \sigma_1} + \lambda k_{ct} k_{\text{time1}}(t,t') \frac{\partial k_{\text{Continuous}}}{\partial \sigma_1} \\
\frac{\partial k_{\text{Continuous}}}{\partial \sigma_1} =& k_{\text{Continuous}}(x,x')
\end{aligned}$$

- The gradient of $\frac{\partial \mathcal{L}}{\partial \sigma_2}$

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \sigma_2} =& \frac{\partial \mathcal{L}}{\partial k_z} \times \frac{\partial k_z}{\partial \sigma_2} \\
\frac{\partial k_z}{\partial \sigma_2} =& (1-\lambda)\, k_{\text{time2}}(t,t') \frac{\partial k_{\text{Categorical}}}{\partial \sigma_2} + \lambda k_{xt} k_{\text{time2}}(t,t') \frac{\partial k_{\text{Categorical}}}{\partial \sigma_2} \\
\frac{\partial k_{\text{Categorical}}}{\partial \sigma_2} =& k_{\text{Categorical}}(c,c')
\end{aligned}$$

- The gradient of $\frac{\partial \mathcal{L}}{\partial \lambda}$

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \lambda} =& \frac{\partial \mathcal{L}}{\partial k_z} \times \frac{\partial k_z}{\partial \lambda} \\
\frac{\partial k_z}{\partial \lambda} =& -(k_{ct} + k_{xt}) + k_{ct} k_{xt}
\end{aligned}$$

B

Appendix for Evolving Curricula with Regret-Based Environment Design

B.1 Full Experimental Results

Partially-Observable Navigation Next we show the extended results for the MiniGrid experiments. We use a series of challenging zero-shot environments (see Figure B.1), introduced in the UED literature [54, 120]. We include the full results in Table B.1.

We also include an additional version of ACCEL using a the same generator as the DR and PLR baselines, thus editing more complex base levels. This removes the prior that it is beneficial to begin with simple empty rooms. As we see, both versions of ACCEL significantly outperforms the baselines. Particularly in the more complex environments like Labyrinth we see large gains vs. the baselines. Also note that PLR outperforms all other baselines, and ACCEL outperforms PLR. We show this in the Table with a statistically significant point estimate, but also using the more robust metrics in `rliable` [2].

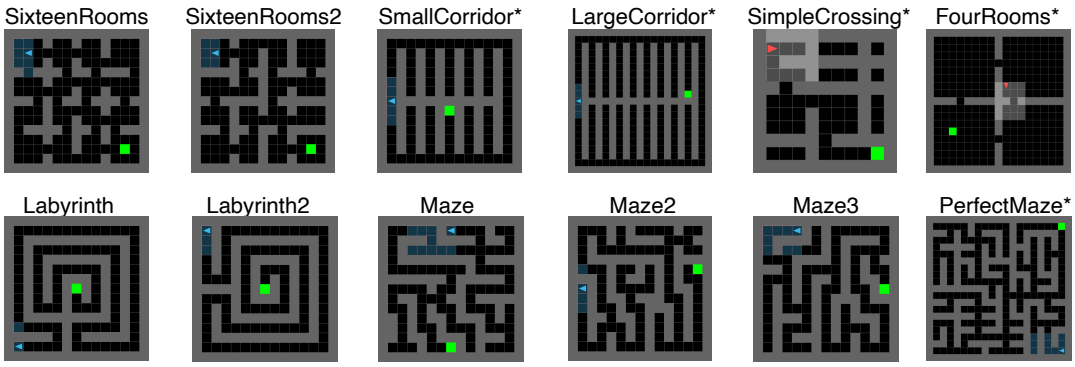


Figure B.1: MiniGrid Zero-Shot Environments. Those with an asterisk are procedurally generated: For Small and Large Corridor, the position of the goal can be in any of the corridors, for SimpleCrossing and Four Rooms see Chevalier-Boisvert *et al.* [38] and for PerfectMaze see Jiang *et al.* [120].

Table B.1: Zero-Shot transfer to human-designed environments. Each data point corresponds to the mean (and standard error) of five independent runs, where each run is evaluated for 100 trials on each environment. † indicates the generator first samples the number of blocks to place, between zero and sixty, then places that many. ‡ indicates the generator produces empty rooms. Bold indicates being within one standard error of the best mean. $\star$ indicates $p < 0.05$ in Welch's t-test against PLR. Note that all methods are evaluated after 20k student updates, aside from PAIRED and Minimax which have 30k updates.

Environment	PAIRED	Minimax	DR†	PLR†	ACCEL†	ACCEL‡
16Rooms	0.63 ± 0.14	0.01 ± 0.01	0.87 ± 0.06	0.95 ± 0.03	$\mathbf{1.0 \pm 0.0}$	$\mathbf{1.0 \pm 0.0}$
16Rooms2	0.53 ± 0.15	0.0 ± 0.0	0.53 ± 0.18	0.49 ± 0.17	0.62 ± 0.22	$\mathbf{0.92 \pm 0.06}$
SimpleCrossing	0.55 ± 0.11	0.11 ± 0.04	0.57 ± 0.15	$\mathbf{0.87 \pm 0.05}$	$\mathbf{0.92 \pm 0.08}$	$\mathbf{0.84 \pm 0.16}$
FourRooms	0.46 ± 0.06	0.14 ± 0.03	0.77 ± 0.1	0.64 ± 0.04	$\mathbf{0.9 \pm 0.08}$	0.72 ± 0.07
SmallCorridor	0.37 ± 0.09	0.14 ± 0.09	$\mathbf{1.0 \pm 0.0}$	0.89 ± 0.05	0.88 ± 0.11	$\mathbf{1.0 \pm 0.0}$
LargeCorridor	0.27 ± 0.08	0.14 ± 0.09	0.64 ± 0.05	0.79 ± 0.13	0.94 ± 0.05	$\mathbf{1.0 \pm 0.0}$
Labyrinth	0.45 ± 0.14	0.0 ± 0.0	0.45 ± 0.23	0.55 ± 0.23	$\mathbf{0.97 \pm 0.03}$	0.86 ± 0.14
Labyrinth2	0.38 ± 0.12	0.0 ± 0.0	0.54 ± 0.18	0.66 ± 0.18	$\mathbf{1.0 \pm 0.01}$	$\mathbf{1.0 \pm 0.0}$
Maze	0.02 ± 0.01	0.0 ± 0.0	0.43 ± 0.23	$\mathbf{0.54 \pm 0.19}$	$\mathbf{0.52 \pm 0.26}$	$\mathbf{0.72 \pm 0.24}$
Maze2	0.37 ± 0.13	0.0 ± 0.0	0.49 ± 0.16	0.74 ± 0.13	0.93 ± 0.04	$\mathbf{1.0 \pm 0.0}$
Maze3	0.3 ± 0.12	0.0 ± 0.0	0.69 ± 0.19	0.75 ± 0.12	$\mathbf{0.94 \pm 0.06}$	0.8 ± 0.1
PerfectMaze (M)	0.32 ± 0.06	0.01 ± 0.0	0.45 ± 0.1	0.62 ± 0.09	$\mathbf{0.88 \pm 0.12}$	$\mathbf{0.93 \pm 0.07}$
Mean	0.39 ± 0.03	0.05 ± 0.01	0.62 ± 0.05	0.71 ± 0.04	$\mathbf{0.88 \pm 0.04}^{\star}$	$\mathbf{0.9 \pm 0.03}^{\star}$

BipedalWalker For the BipedalWalker environment we test agents on each of the individual challenges from the environment parameterization. Concretely, we use the following four environments:

- `Stairs`: the stair height parameters are set to [2,2] with the number of steps set to 5.
- `PitGap`: the PitGap parameter is set to [5,5].
- `Stump`: the Stump parameter is set to [2,2].
- `Roughness`: the Roughness parameter is set to 5.

Each of these is visualized in the main body of the paper, in Figure 4.10. We also test agents on the simple `BipedalWalker-v3` environment and the more challenging `BipedalWalkerHardcore-v3` environment. For the "Hardcore" version, we note that none of our agents *officially* solve the environment, which is considered to be a reward over 300 for 100 independent evaluations. To test whether it is possible with our setup, we trained an identical PPO agent directly on the environment for 1B steps. The reward achieved was 239—indistinguishable from what is achieved by ACCEL using a far more challenging design space. In addition, the ACCEL agent is robust to individual challenges.

Table B.2: Test performance on a variety of challenges. Each data point corresponds to the mean (and standard error) of ten independent runs, where each run is evaluated for 100 trials on each environment. † indicates the generator first samples the number of blocks to place, between zero and sixty, then places that many. ‡ indicates using the low value generator. Bold indicates being within one standard error of the best mean. All methods are evaluated after 30k updates.

Environment	PAIRED	Minimax	ALP-GMM	DR†	PLR†	ACCEL†	ACCEL‡
Basic	206.5 ± 30.3	154.3 ± 59.2	301.5 ± 11.6	261.9 ± 19.3	304.1 ± 1.8	316.9 ± 2.1	$\mathbf{318.1 \pm 1.0}$
Hardcore	-47.2 ± 10.6	-44.3 ± 1.6	29.7 ± 9.9	23.8 ± 8.3	82.6 ± 8.5	163.3 ± 30.9	$\mathbf{236.0 \pm 8.9}$
Stairs	-27.4 ± 12.1	-2.6 ± 2.6	22.1 ± 6.3	23.3 ± 4.4	48.0 ± 4.3	59.4 ± 10.5	$\mathbf{91.7 \pm 8.9}$
PitGap	-68.2 ± 9.7	-79.3 ± 0.5	$\mathbf{98.8 \pm 24.9}$	11.0 ± 7.6	46.2 ± 11.3	49.6 ± 12.6	$\mathbf{133.3 \pm 39.1}$
Stump	-76.0 ± 10.3	-65.0 ± 18.4	-22.4 ± 17.2	-5.4 ± 5.5	7.5 ± 6.4	44.6 ± 49.8	$\mathbf{188.8 \pm 10.9}$
Roughness	-5.1 ± 25.9	-1.2 ± 7.7	44.7 ± 11.6	52.3 ± 9.0	126.7 ± 7.3	211.7 ± 21.5	$\mathbf{248.9 \pm 12.3}$
Mean	-2.9 ± 14.5	-6.3 ± 24.6	79.1 ± 17.5	61.1 ± 12.6	102.5 ± 13.0	140.9 ± 23.0	$\mathbf{202.8 \pm 13.6}$

POET Generated Levels We also evaluated our agents on the six "Extremely Challenging" environments highlighted in the original POET paper. These represent some of the most difficult environment parameterizations produced by POET. Since each run of POET has a population of 20 agents, it is not clear if a single agent from any of their runs can solve more than one of these challenges. In addition, POET only solves a single fixed seed of these environments. Instead, we report the mean performance over a hundred random samples for each given parameterization, for all ten runs of ACCEL with the mean and max performance (across seeds) shown in Table B.3.

Table B.3: Test performance on levels produced by POET. In each case we run 100 trials with different random seeds. Mean shows the mean performance across all ten ACCEL runs. Max shows the best from the ten runs per environment.

	1a	1b	2a	2b	3a	3b
Mean	0.01	0.01	0.00	0.03	0.01	0.12
Max	0.03	0.05	0.00	0.08	0.03	0.31

B.2 Additional Experiments

In this section we show a series of additional experiments to understand the performance of ACCEL

Ablation Studies To investigate the design choices that led to the success of ACCEL, we consider a variety of ablations:

- **ACCEL** Using the DR generator.
- **Edit Batch** The same ACCEL algorithm but editing the entire replay batch rather than the "easy" levels from the batch.
- **Learned Editor** The same algorithm changing only the editor, which is optimized for positive value loss.
- **No Editor** This is an ablation on the algorithm structure, we use Algorithm 6 but sample more DR levels instead of editing.

Table B.4: Zero-Shot transfer to human-designed environments. Each data point corresponds to the mean (and standard error) of five independent runs, where each run is evaluated for 100 trials on each environment. All methods use a DR generator which places between zero and sixty blocks.

Test Environment	ACCEL	Edit Batch	Learned Editor	No Editor
16Rooms	1.0 ± 0.0	0.76 ± 0.19	0.9 ± 0.07	0.84 ± 0.06
16Rooms2	0.51 ± 0.28	0.23 ± 0.16	0.41 ± 0.19	0.68 ± 0.18
SimpleCrossing	0.8 ± 0.05	1.0 ± 0.0	0.9 ± 0.1	0.75 ± 0.05
FourRooms	0.85 ± 0.05	0.85 ± 0.06	0.88 ± 0.04	0.88 ± 0.05
SmallCorridor	0.72 ± 0.1	0.74 ± 0.1	0.6 ± 0.17	0.7 ± 0.18
LargeCorridor	0.91 ± 0.05	0.75 ± 0.08	0.56 ± 0.18	0.63 ± 0.18
Labyrinth	0.98 ± 0.02	0.85 ± 0.11	0.99 ± 0.01	0.67 ± 0.19
Labyrinth2	0.97 ± 0.03	0.83 ± 0.11	0.7 ± 0.15	0.48 ± 0.2
Maze	0.78 ± 0.21	0.87 ± 0.05	0.57 ± 0.18	0.15 ± 0.08
Maze2	0.5 ± 0.24	0.67 ± 0.18	0.65 ± 0.15	0.23 ± 0.15
Maze3	0.79 ± 0.14	0.9 ± 0.08	0.95 ± 0.05	0.56 ± 0.17
Mean	0.79 ± 0.04	0.76 ± 0.04	0.74 ± 0.04	0.58 ± 0.05

Each of these uses the same exact structure, sampling levels from the DR distribution 10% of the time, and editing levels after replaying from the curator. For the first, we edit the entire replay batch, rather than just the top four easiest (high return minus positive value loss) that we use in the main experiments. The results after 10k updates are shown in TableB.4. As we see, Editing the batch levels performs slightly worse than easy, while there is also a decrease in performance for the learned editor. Note however that all of these ablations still outperform the next best baseline (PLR, mean = 0.69). Finally, the ablation using additional DR levels instead of edited ones performs poorly, showing the strong performance for ACCEL comes from editing rather than the structure of the algorithm.

B.3 Implementation Details

In this section we detail the training procedure for all our experiments. Training for all methods is conducted on a single V100 GPU, using ten CPUs. The codebase is built on top of open source repos. For PAIRED and Minimax in

the partially-observable navigation environment we use results from the authors of Jiang *et al.* [120].

B.3.1 Environment Details

Partially-Observable Navigation Each maze consists of a 15×15 grid, where each cell can contain a wall, the goal, the agent, or navigable space. The student agent receives a reward of $1 - T/T_{\max}$ upon reaching the goal, where T is the episode length and $T_{\max}$ is the maximum episode length (set to 250). Otherwise, the agent receives a reward of 0 if it fails to reach the goal.

BipedalWalker The BipedalWalker environment we use is a modified version of the `BipedalWalkerHardcore` environment from OpenAI Gym. The agent learns from a proprioceptive state with 24 dimensions consisting of lidar sensors, angles and contacts but notably the agent does not have access to its map coordinates. The action space is four continuous values controlling the torques of its 4 motors. The environment design space is shown in Table B.5, where we show the value of the initialization for ACCEL, the edit size, and the maximum values. DR levels are sampled between zero and the maximum value. For PLR, we combine the environment parameterization with the specific seed of the sampled level, making the replay deterministic. For ACCEL we make an edit by randomly sampling one of the eight parameters and then adding or subtracting the quantity shown in the "Edit Size" row of Table B.5.

Table B.5: Environment design space for the BipedalWalker environment. Where values are shown as ranges, both the low and high end of the range are learned parameters, and when the level is created each obstacle size is sampled from the range.

	Stump Height	**Stair Height**	**Stair Steps**	**Roughness**	**Pit Gap**
Easy Init	[0,0.4]	[0,0.4]	1	Unif(0.6)	[0,0.8]
Edit Size	0.2	0.2	1	Unif(0.6)	0.4
Max Value	[5,5]	[5,5]	9	10	[10,10]

Table B.6: Total number of environment interactions for a given number of student PPO updates.

Environment	**PPO Updates**	PLR	ACCEL
MiniGrid	20k	327M	369M
BipedalWalker	30k	1.96B	2.07B

B.3.2 Environment Design Procedure

The environment design procedure works as follows: at each timestep the adversary agent receives an observation consisting of a map of the entire level and then takes a two dimensional action, consisting of an object and a location, which can be anywhere in the grid. This is similar to several recent works [54, 120, 121, 131]. For MiniGrid the object is always a wall. For both methods, the goal and agent location are placed in the final two steps.

When editing, the editor has five steps to alter the environment. For the lava environment we only edit to add or remove lava tiles, while for MiniGrid we allow the editor to also change the goal location. If lava or walls (or lava) are placed in the current location of the goal or agent, then these replace the goal or agent, which must then be relocated in the final two steps of the editing episode. If the editor attempts to remove the goal or agent location in the final two steps then the action is void.

B.3.3 Hyperparameters

The majority of our hyperparameters are inherited from previous works such as [54, 120, 121], with a few small changes.

For MiniGrid we maintain the replay buffer size from Jiang *et al.* [120] and only conduct the ACCEL grid search over the edit objective, running both {random, positive value loss}, as well as the levels to edit from {easy, batch} and replay rate from {0.8, 0.9}. For MiniGrid, we follow the protocol from Jiang *et al.* [120] and select the best hyperparameters using the validation levels {16Rooms, Labyrinth, Maze}. The final hyperparameters chosen are shown in Table B.7.

B. Appendix for Evolving Curricula with Regret-Based Environment Design

For BipedalWalker we used the continuous control policy from the open source implementation of PPO from Kostrikov [140], as well as many of the hyperparameters used in the recommended settings for MuJoCo. This involves a simple feedforward neural network with two hidden layers of size 64 and Tanh activations. We tuned the hyperparameters for our base agent using domain randomization, and conducted a sweep over the learning rate {3e-4, 3e-5}, PPO epochs {5,20}, entropy coefficient {0, 1e-3} and number of minibatches {4,32}, using the validation performance on BipedalWalkerHardcore. We then used these base agent configurations for all UED algorithms. For PLR we further conducted a sweep over the buffer size {1000, 5000}, replay rate {0,9, 0.5} and staleness coefficient {0.3, 0.5, 0.7}, using the same settings found for both PLR and ACCEL. Finally, for ACCEL we tuned both the number of edits {1, 2, 3, 4} and whether to edit the easy or batch levels.

Table B.7: Hyperparameters used for training each method in the maze and car racing environments.

Parameter	MiniGrid	BipedalWalker
PPO		
γ	0.995	0.99
λ_{GAE}	0.95	0.9
PPO rollout length	256	2000
PPO epochs	5	5
PPO minibatches per epoch	1	32
PPO clip range	0.2	0.2
PPO number of workers	32	16
Adam learning rate	1e-4	3e-4
Adam ϵ	1e-5	1e-5
PPO max gradient norm	0.5	0.5
PPO value clipping	yes	no
return normalization	no	yes
value loss coefficient	0.5	0.5
student entropy coefficient	0.0	1e-3
generator entropy coefficient	0.0	0.0
ACCEL		
Edit rate, q	1.0	1.0
Replay rate, p	0.8	0.9
Buffer size, K	4000	1000
Scoring function	positive value loss	positive value loss
Edit method	random	random
Levels edited	easy	easy
Prioritization	rank	rank
Temperature, β	0.3	0.1
Staleness coefficient, ρ	0.5	0.5
Robust PLR		
Scoring function	positive value loss	positive value loss
Replay rate, p	0.5	0.5
Buffer size, K	4000	1000

C

Appendix for Diversity Algorithms

C.1 Appendix for Diversity via Determinants

In the case of deterministic behavioral embeddings, it is possible to compute gradients through the whole objective. Notice that $\det(\mathbf{K})$ is then differentiable as a function of the policy parameters. In the case of trajectory based embeddings, differentiating through the determinant is not that simple. Actually it may make sense in this case to use a $\log(\det(\mathbf{K}))$ score instead. This is because of the following lemma:

Lemma 20. *The gradient of* $\log(\det(\mathbf{K}))$ *with respect to* $\theta = \theta^1, \cdots, \theta^M$ *equals:*

$$\nabla_\theta \log(\det(\mathbf{K})) = -(\nabla_\theta \Phi(\theta))(\nabla_\Phi \mathbf{K})\mathbf{K}^{-1}$$

Where $\Phi(\theta) = \Phi(\theta^1) \cdots \Phi(\theta^M)$

Proof. We start with:

$$\nabla_{\mathbf{K}} \log(\det(\mathbf{K})) = \mathbf{K}^{-1}$$

This is a known result[1].

Consequently,

[1] see for example https://math.stackexchange.com/questions/38701/how-to-calculate-the-gradient-of-log-det-matrix-inverse

$$\nabla_\theta \log\left(\det(\mathbf{K})\right) = \left(\nabla_\theta \Phi(\theta)\right)\left(\nabla_\Phi \mathbf{K}\right)\left(\nabla_{\mathbf{K}} \log\left(\det(\mathbf{K})\right)\right)$$
$$= \left(\nabla_\theta \Phi(\theta)\right)\left(\nabla_\Phi \mathbf{K}\right)\mathbf{K}^{-1}$$

Each of the other gradients can be computed exactly.

□

C.2 Appendix for Ridge Rider

C.2.1 Behavior of gradient descent near a saddle point

We will illustrate how gradient descent dynamics near a saddle point moves towards the most negative eigenvector of the Hessian via two different derivations. These are not novel results but provided as an illustration of a well known fact.

First, let θ_0 be a saddle point of $\mathcal{L}(\theta)$, and consider T steps of gradient descent $\theta_1, ..., \theta_T$. Let $H(\theta_0)$ be the Hessian of $\mathcal{L}$ at θ_0. We will use the first-order Taylor expansion, $\nabla_\theta \mathcal{L}(\theta_t) = H(\theta_t - \theta_0) + o(\epsilon^2)$ ignoring the error term to approximate the gradient close to θ_0.

We can decompose $\theta_t - \theta_0$ into the basis of eigenvectors of $H(\theta_0)$: $\theta_t - \theta_0 = \sum_i a_{i,t} e_i(\theta_0)$ where $\{e_i(\theta_0)\}$ are the eigenvectors of $H(\theta_0)$. After one step of gradient descent with learning rate α,

$$\theta_{t+1} = \theta_0 + \sum_i a_{i,t} e_i(\theta_0) - \alpha \sum_i \lambda_i a_{i,t} e_i(\theta_0) \tag{C.1}$$
$$= x_0 + \sum_i \left(1 - \alpha\lambda_i(\theta_0)\right) a_{i,t} e_i(\theta_0) \tag{C.2}$$

i.e. $a_{i,t+1} = (1 - \alpha\lambda_i(\theta_0)) a_{i,t}$

It follows by simple induction that if T isn't too large so the displacement $\theta_T - \theta_0$ is still small, $a_{i,T} = (1 - \alpha\lambda_i)^T$. In other words, the component of $\theta_1 - \theta_0$ corresponding to more negative eigenvalues of $H(\theta_0)$ will be amplified relative to less negative eigenvalues by a ratio that grows exponentially in T.

In the limit of small step sizes, we can also consider the differential limit of approximate (up to first order terms as defined above) gradient descent dynamics

$$\frac{d\theta}{dt} = -\alpha H(\theta_0)\theta,$$

assuming the saddle is at $\theta_0 = 0$ wlog. The solution to this system of equations is $\theta(t) = \theta(0)\exp(-\alpha H(\theta_0)t)$. If we write the eigendecomposition of $H(\theta_0)$, $H(\theta_0) = Q\Lambda Q^{-1}$, then $\exp(-H(\theta_0)) = -Q\exp(\Lambda)Q^{-1}$. So if $\theta(0) = \sum_i a_i e_i(\theta_0)$, then $\theta(t) = \sum_i e^{-\alpha\lambda_i(\theta_0)t} a_i e_i(\theta_0)$.

C.2.2 Structural properties of the eigenvalues and eigenvectors of Smooth functions

Definition 21. *We say a function $f : \mathbb{R}^d \to \mathbb{R}$ is $\beta-$smooth if for all pairs of points $\theta, \theta' \in \mathbb{R}^d$:*

$$|\nabla^2_\theta f(\theta) - \nabla^2_\theta f(\theta')| \leq \beta\|\theta - \theta'\|$$

We show that for $\beta-$smooth functions f, the $i-$th eigenvalue function:

$$\lambda_i(\theta) = i - \text{th largest eigenvalue of } \nabla^2_\theta f(\theta) \tag{C.3}$$

And (an appropriate definition of) the $i-$th eigenvector function:

$$e_i(\theta) = i - \text{th largest normalized eigenvector of } \nabla^2_\theta f(\theta) \tag{C.4}$$

are continuous functions of θ.

Lemma 22. *If f is $\beta-$smooth' $\lambda_i(\theta)$ is continuous.*

Proof. We show the result for the case $i = 1$, the proof for $i \neq 1$ follows the same structure, albeit making use of the more complex variational characterization of the $i-$th eigenvalue / eigenvector pair. Recall the variational formulation of λ_1:

$$\lambda_1(\theta) = \max_{v \in \mathcal{S}_d(1)} v^\top \nabla^2_\theta f(\theta) v \tag{C.5}$$

Let $\theta' = \theta + \Delta_\theta$. It is enough to show that:

C. Appendix for Diversity Algorithms

$$\lim_{\Delta_\theta \to 0} \lambda_1(\theta') = \lambda_1(\theta)$$

Let v_1 be a unit vector achieving the max in Equation C.5 and let v_1' be the maximizer for the corresponding variational equation for θ', then:

$$\begin{aligned} |\lambda_1(\theta) - v_1^\top \nabla_\theta^2 f(\theta') v_1| &= |v_1^\top \left(\nabla_\theta^2 f(\theta) - \nabla_\theta^2 f(\theta') \right) v_1| \\ &\overset{(i)}{\leq} \beta \|\Delta_\theta\| \end{aligned}$$

Inequality (i) follows by $\beta-$smoothness. Similarly:

$$|\lambda_1(\theta') - (v_1')\nabla_\theta^2 f(\theta) v_1'| \leq \beta \|\Delta_\theta\|$$

Since by definition:

$$\lambda_1(\theta) \geq (v_1')^\top \nabla_\theta^2 f(\theta) v_1'$$

And:

$$\lambda_1(\theta') \geq v_1^\top \nabla_\theta^2 f(\theta') v_1$$

We conclude that:

$$\lambda_1(\theta') \geq \lambda_1(\theta) - \beta \|\Delta_\theta\|$$

And:

$$\lambda_1(\theta) \geq \lambda_1(\theta') - \beta \|\Delta_\theta\|$$

Consequently:

$$\lambda_1(\theta) + \beta \|\Delta_\theta\| \geq \lambda_1(\theta') \geq \lambda_1(\theta) - \beta \|\Delta_\theta\|$$

The result follows by taking the limit as $\Delta_\theta \to 0$. □

In fact the proof above shows even more:

If L is $\beta-$smooth and $\theta, \theta' \in \mathbb{R}^d$ then the eigenvalue function is β-Lipschitz:

$$|\lambda_i(\theta') - \lambda_i(\theta)| \leq \beta \|\theta - \theta'\|$$

And therefore:

$$\|\nabla_\theta \lambda_i(\theta)\| \leq \beta$$

Proof. With the exact same sequence of steps as in the proof of Lemma 22, we conclude:

$$\lambda_i(\theta) + \beta\|\Delta_\theta\| \geq \lambda_i(\theta') \geq \lambda_i(\theta) - \beta\|\Delta_\theta\|$$

And:

$$\lambda_i(\theta') + \beta\|\Delta_\theta\| \geq \lambda_i(\theta) \geq \lambda_i(\theta') - \beta\|\Delta_\theta\|$$

The result follows. The gradient bound is an immediate consequence of the Lipschitz property of $\lambda_1(\cdot)$. □

Let's define a canonical $i-$th eigenvector path $\ell : [0, \infty) \to \mathbb{R}$ starting at θ as follows:

$$\begin{aligned} \ell(0) &= \theta \\ \frac{\partial \ell(t)}{\partial t} &= \begin{cases} e_i(\theta) & \text{if } \lim_{l \to t} \langle e_i(\theta), \frac{\partial \ell(l)}{\partial l} \ell(l) \rangle > 0 \\ -e_i(\theta) & \text{o.w.} \end{cases} \end{aligned}$$

We proceed to show that the curve traced by ℓ is continuous.

Lemma 23. *Let θ be such that $\lambda_{i-1}(\theta) - \lambda_i(\theta) = \Delta_{i-1} > 0$ and $\lambda_i(\theta) - \lambda_{i+1}(\theta) = \Delta_i > 0$. Then:*

$$\min(\|e_i(\theta) - e_i(\theta')\|, \|e_i(\theta) + e_i(\theta')\|) \leq \sqrt{\frac{4\beta\|\theta - \theta'\|}{\min(\Delta_i, \Delta_{i-1})}} \tag{C.6}$$

For all θ' such that $\|\theta' - \theta\| \leq \frac{\min(\Delta_{i-1}, \Delta_i)}{4\beta}$

Proof. By Proposition C.2.2, for all θ' such that $\|\theta' - \theta\| \leq \frac{\min(\Delta_{i-1}, \Delta_i)}{4\beta}$:

$$\max(|\lambda_i(\theta') - \lambda_i(\theta)|, |\lambda_{i-1}(\theta') - \lambda_{i-1}(\theta)|, |\lambda_{i+1}(\theta') - \lambda_{i+1}(\theta)|) \leq \beta\|\theta' - \theta\| \tag{C.7}$$

In other words, it follows that:

$$\lambda_{i+1}(\theta), \lambda_{i+1}(\theta') < \lambda_i(\theta), \lambda_i(\theta') < \lambda_{i-1}(\theta), \lambda_{i-1}(\theta')$$

Where this sequence of inequalities implies for example that $\lambda_{i+1}(\theta') < \lambda_i(\theta)$.

C. Appendix for Diversity Algorithms

Let $H = \nabla^2_\theta f(\theta)$ and $H' = \nabla^2_\theta f(\theta')$. Let $\Delta = H - H'$. By $\beta-$smoothness we know that $\|\Delta\| \leq \beta\|\theta - \theta'\|$.

Again for simplicity we restrict ourselves to the case $i = 1$. The argument for $i \neq 1$ uses the same basic ingredients, but takes into account the more complex variational characterization of the $i-$th eigenvector.

W.l.o.g. define $e_1(\theta)$ and $e_1(\theta')$ such that $\langle e_1(\theta), e_1(\theta')\rangle = \alpha \geq 0$. We can write $e_1(\theta') = \alpha e_1(\theta) + \left(\sqrt{1-\alpha^2}\right) v$ with $\|v\|_2 = 1$ and $\langle v, e_1(\theta)\rangle = 0$. Notice that $\|e_1(\theta) - e_1(\theta')\| = \|(1-\alpha)e_1(\theta) - \left(\sqrt{1-\alpha^2}\right) v\| \leq (1-\alpha) + \sqrt{1-\alpha^2}$. We now show α is close to 1. Recall:

$$e_1(\theta) = \arg\max_{v \in \mathbb{S}_d} v^\top H v$$

and

$$e_1(\theta') = \arg\max_{v' \in \mathbb{S}_d} (v')^\top H' v'$$

Equation C.7 implies $\lambda_1(\theta) \geq \lambda_2(\theta') + \Delta_1 - \beta\|\theta - \theta'\|$ and $\lambda_1(\theta') \geq \lambda_2(\theta) + \Delta_1 - \beta\|\theta - \theta'\|$. The following inequalities hold:

$$\begin{aligned}
\lambda_1(\theta') &\geq e_1(\theta)^\top H' e_1(\theta) && \text{(C.8)}\\
&= e_1(\theta)^\top [H - \Delta] e_1(\theta)\\
&= e_1(\theta)^\top H e_1(\theta) - e_1(\theta)^\top \Delta e_1(\theta)\\
&= \lambda_1(\theta) - e_1(\theta)^\top \Delta e_1(\theta)\\
&\geq \lambda_1(\theta) - \|\Delta\|\\
&\geq \lambda_1(\theta) - \beta\|\theta - \theta'\| && \text{(C.9)}
\end{aligned}$$

Write $e_1(\theta) = \sum_i \alpha_i e_i(\theta')$ with $\sum_i \alpha_i^2 = 1$. Notice that:

$$e_1(\theta)^\top H' e_1(\theta) = \sum_i \alpha_i^2 \lambda_i(\theta')$$

Since $\lambda_i(\theta') < \lambda_1(\theta) - \frac{3\Delta_1}{4}$ for all $i > 1$:

$$\begin{aligned}\sum_i \alpha_i^2 \lambda_i(\theta') &\leq \alpha_1^2 \lambda_1(\theta') + \left(\sum_{i=2}^{d} \alpha_i^2\right)(\lambda_1(\theta) - \Delta_1 + \beta\|\theta - \theta'\|) \\ &\leq \alpha_1^2(\lambda_1(\theta) + \beta\|\theta - \theta'\|) + (\sum_{i=2}^{d} \alpha_i^2)(\lambda_1(\theta) - \Delta_1 + \beta\|\theta - \theta'\|) \\ &\leq \lambda_1(\theta) + \beta\|\theta - \theta'\| - \Delta_1(1 - \alpha_1^2) \end{aligned} \tag{C.10}$$

And therefore, combining Equation C.9 and C.10:

$$\lambda_1(\theta) - \beta\|\theta - \theta'\| \leq e_1(\theta)^\top H' e_1(\theta) \leq \lambda_1(\theta) + \beta\|\theta - \theta'\| - \Delta_1(1 - \alpha_1^2)$$

Therefore:

$$\alpha_1^2 \geq \frac{\Delta_1 - 2\beta\|\theta - \theta'\|}{\Delta_1} = 1 - \frac{2\beta\|\theta - \theta'\|}{\Delta_1}$$

This in turn implies that $\sum_{i=2}^{d} \alpha_i^2 \leq \frac{2\beta\|\theta-\theta'\|}{\Delta_1}$ and that $1 - \alpha \leq 1 - \alpha^2 \leq \frac{2\beta\|\theta-\theta'\|}{\Delta_1}$. Therefore:

$$\begin{aligned}\|e_1(\theta) - e_1(\theta')\|^2 &= (1 - \alpha_1)^2 + \sum_{i=2}^{d} \alpha_2^2 \\ &\leq (1 - \alpha_1)^2 + \frac{2\beta\|\theta - \theta'\|}{\Delta_1} \\ &\leq \frac{4\beta^2\|\theta - \theta'\|^2}{\Delta_1^2} + \frac{2\beta\|\theta - \theta'\|}{\Delta_1} \\ &\leq \frac{4\beta\|\theta - \theta'\|}{\Delta_1}\end{aligned}$$

The result follows.

□

As a direct implication of Lemma 23, we conclude the eigenvector function is continuous.

C.2.3 Convergence rates for finding a new eigenvector, eigenvalue pair

Let $\theta' = \theta + \Delta_\theta$, ridge riding minimizes the following loss w.r.t e and λ to find a candidate e' and λ':

$$L(e, \lambda; \theta') = \|(1/\lambda)H(\theta')e/\|e\| - e/\|e\|\|^2 \tag{C.11}$$

C. Appendix for Diversity Algorithms

Notice that:

$$L(e,\lambda;\theta') = \frac{1}{\lambda^2\|e\|^2}e^\top H(\theta')^2 e + 1 - 2\frac{1}{\lambda\|e\|^2}e^\top H(\theta')e$$

Therefore:

$$\begin{aligned}\nabla_e L(e,\lambda;\theta') &= \frac{1}{\lambda^2}\nabla_e\left(\frac{1}{\|e\|^2}e^\top H(\theta')^2 e\right) - 2\frac{1}{\lambda}\nabla_e\left(\frac{1}{\|e\|^2}e^\top H(\theta')e\right)\\ &= \frac{2}{\lambda^2\|e\|}\left(H^2(\theta') - \tilde{e}^\top H^2(\theta')\tilde{e}I\right)\tilde{e} - \frac{4}{\lambda\|e\|}\left(H(\theta') - \tilde{e}^\top H(\theta')\tilde{e}I\right)\tilde{e}\\ &= \left(\frac{2}{\lambda^2\|e\|}H^2(\theta') - \frac{4}{\lambda\|e\|}H(\theta')\right)\tilde{e} + \left(\frac{4}{\lambda\|e\|}\tilde{e}^\top H(\theta')\tilde{e}I - \frac{2}{\lambda^2\|e\|}\tilde{e}^\top H^2(\theta')\tilde{e}I\right)\tilde{e}\end{aligned}$$

Where $\tilde{e} = \frac{e}{\|e\|}$.

Now let's compute the following gradient:

$$\nabla_\lambda L(e,\lambda;\theta') = -\frac{2}{\lambda^3\|e\|^2}e^\top H(\theta')^2 e + \frac{2}{\lambda^2\|e\|^2}e^\top H(\theta')e = \frac{2}{\lambda^2\|e\|}\left(e^\top H(\theta')e - \frac{e^\top H(\theta')^2 e}{\lambda}\right)$$

We consider the following algorithm:

1. Start at (e,λ).
2. Take a gradient step $e \to e - \alpha_e\nabla_e L(e,\lambda;\theta')$.
3. Take a gradient step $\lambda \to \lambda - \alpha_\lambda\nabla_\lambda L(e,\lambda;\theta')$.
4. Normalize $e \to \frac{e}{\|e\|}$.

It is easy to see that the update for e takes the form:

$$e \to \underbrace{\left(\left(1 + \alpha_e\left(\frac{2e^\top H(\theta')^2 e}{\lambda^2} - \frac{4e^\top H(\theta')e}{\lambda}\right)\right)I + \alpha_e\left(\frac{4H(\theta')}{\lambda} - \frac{2H(\theta')^2}{\lambda^2}\right)\right)}_{U} e$$

$$e \to \frac{e}{\|e\|}$$

Where we think of U as an operator acting on the vector e. In fact if we consider T consecutive steps of this algorithm, yielding normalized eigenvector candidates

$e_0, \cdots, e_T$ and eigenvalue candidates $\lambda_0, \cdots, \lambda_T$, and name the corresponding $U-$operators as $U_1, \cdots, U_T$ it is easy to see that:

$$e_T = \frac{E_T}{\|E_T\|}$$

Where $E_T = \left(\prod_{i=1}^{T} U_T\right) e_0$. In other words, the normalization steps can be obviated as long as we normalize at the very end. This observation will prove useful in the analysis.

Let's assume L is $\beta-$smooth and let's say we are trying to find the $i-$th eigenvalue eigenvector pair for θ': $(e_i(\theta'), \lambda_i(\theta'))$. Furthermore let's assume we start our optimizaation at the $(e_i(\theta), \lambda_i(\theta))$ pair. Furthermore, assume that θ' is such that:

$$\|e_i(\theta) - e_i(\theta')\| \leq \min(\frac{\Delta_i}{4}, \frac{\Delta_{i-1}}{4}) \text{ and } \|\lambda_i(\theta) - \lambda_i(\theta')\| \leq \min(\frac{\Delta_i}{4}, \frac{\Delta_{i-1}}{4}) \quad \text{(C.12)}$$

Where $\lambda_i(\theta) - \lambda_i(\theta) = \Delta_{i-1} > 0$ and $\lambda_i(\theta) - \lambda_{i+1}(\theta) = \Delta_i > 0$. The existence of such θ' as in C.12 can be guaranteed by virtue of Lemmas C.2.2 and 23.

Notice that as long as $\alpha_e = \min(1/4, \Delta_i, \Delta_{i-1})$ is small enough the operator U attains the form:

$$U = AI + \alpha_e \left(\frac{4H(\theta')}{\lambda} - \frac{2H(\theta')^2}{\lambda^2}\right)$$

Where α_e/A is small.

Notice that the operator $\left(\frac{4H(\theta')}{\lambda} - \frac{2H(\theta')^2}{\lambda^2}\right)$ has the following properties:

1. $\left(\frac{4H(\theta')}{\lambda} - \frac{2H(\theta')^2}{\lambda^2}\right)$ has the exact same eigenvectors set $\{e_j(\theta')\}_{j=1}^{T}$ as $H(\theta')$.
2. The eigenvalues of $\left(\frac{4H(\theta')}{\lambda} - \frac{2H(\theta')^2}{\lambda^2}\right)$ equal $\{\frac{4\lambda_j(\theta')}{\lambda} - \frac{2\lambda_j(\theta'^2}{\lambda^2}\}_{j=1}^{d}$.

Consequently, if $|\lambda - \lambda_i(\theta')| < \min(\frac{\Delta_i'}{4}, \frac{\Delta_{i-1}'}{4})$) we conclude that the maximum eigenvalue of $\left(\frac{4H(\theta')}{\lambda} - \frac{2H(\theta')^2}{\lambda^2}\right)$ equals $\frac{4\lambda_i(\theta')}{\lambda} - \frac{2\lambda_i(\theta')^2}{\lambda^2}$ with eigenvector $e_i(\theta')$.

Furthermore, the eigen-gap between the maximum eigenvalue and any other one is lower bounded by $\frac{\min(\Delta_i, \Delta_{i-1})}{2}$. Therefore, after taking a gradient step on e, the dot product $\langle e_i(\theta'), e_t \rangle = \gamma_t$ satsifies $\gamma_{t+1}^2 \to \gamma_t^2 + \gamma_t^2 * (1 - \alpha_e^2 \frac{\min(\Delta_i, \Delta_{i-1}^2)}{4})$

If $|\lambda_t - \lambda_i(\theta')| < \min(\frac{\Delta'_i}{4}, \frac{\Delta'_{i-1}}{4})$), and the eigenvalue update satisfied the properties above, then λ_{t+1} is closer to $\lambda_i(\theta')$ than λ_t, thus maintaining the invariance. We conclude that the convergence rate is the rate at which $\gamma_t \to 1$, which is captured by the following theorem:

Theorem 24. *If L is $\beta-$smooth, $\alpha_e = \min(1/4, \Delta_i, \Delta_{i-1})$, and $\|\theta - \theta'\| \leq \frac{\min(1/4, \Delta_i, \Delta_{i-1})}{\beta}$ then $|\langle e_t, e_i(\theta')\rangle| \geq 1 - \left(1 - \frac{\min(1/4, \Delta_i, \Delta_{i-1})}{4}\right)^t$*

C.2.4 Staying on the ridge

In this section, we show that under the right assumptions on the step sizes, Ridge Riding stays along a descent direction.

We analyze the following setup. Starting at θ, we move along negative eigenvector $e_i(\theta)$ to $\theta' = \theta - \alpha e_i(\theta)$. Once there we move to $\theta'' = \theta' - \alpha e_i(\theta')$. Let $L : \Theta \to \mathbb{R}$ be the function we are trying to optimize. We show that:

Theorem 25. *Let $L : \Theta \to \mathbb{R}$ have $\beta-$smooth Hessian, let α be the step size. If at θ RR satisfies: $\langle \nabla L(\theta), e_i(\theta)\rangle \geq \|\nabla L(\theta)\|\gamma$, and $\alpha \leq \frac{\min(\Delta_i, \Delta_{i-1})\gamma^2}{16\beta}$ then after two steps of RR:*

$$L(\theta'') \leq L(\theta) - \gamma\alpha\|\nabla L(\theta)\|$$

Proof. Since L is $\beta-$smooth, the third order derivatives of L are uniformly bounded. Let's write $L(\theta')$ using a Taylor expansion:

$$\begin{aligned} L(\theta') &= L(\theta - \alpha e_i(\theta)) \\ &\overset{(i)}{\leq} L(\theta) + \langle \nabla L(\theta), -\alpha e_i(\theta)\rangle + \frac{1}{2}(-\alpha e_i(\theta)^\top H(\theta)(-\alpha e_i(\theta)) + c'\alpha^3\beta \\ &= L(\theta) - \alpha\langle \nabla L(\theta), e_i(\theta)\rangle + \alpha^2\frac{\lambda_i(\theta)}{2} + c'\alpha^3\beta \end{aligned}$$

Inequality (i) follows by Hessian smoothness.

Let's expand $L(\theta'')$:

$$\begin{aligned} L(\theta'') &= L(\theta' - \alpha e_i(\theta')) \\ &\leq L(\theta') - \alpha\langle \nabla L(\theta'), e_i(\theta')\rangle + \frac{\alpha^2}{2}\lambda_i(\theta') + c''\alpha^3\beta \\ &\leq L(\theta) - \alpha\langle \nabla L(\theta), e_i(\theta)\rangle - \alpha\langle \nabla L(\theta'), e_i(\theta')\rangle + \frac{\alpha^2\lambda_i(\theta)}{2} + \frac{\alpha^2\lambda_i(\theta')}{2} + (c' + c'')\alpha^3\beta \end{aligned}$$

Notice that for any $v \in \mathbb{R}^d$ it follows that $\langle \nabla L(\theta'), v\rangle \leq \langle \nabla L(\theta), v\rangle - \alpha v^\top \nabla^2 L(\theta) e_i(\theta) + c'''\beta\alpha^2 = \langle \nabla L(\theta), v\rangle - \alpha\lambda_i(\theta) v^\top e_i(\theta) + c'''\beta\alpha^2$. Plugging this in the sequence of inequalities above:

$$
\begin{aligned}
&\mathrm{L}(\theta'') \leq L(\theta) - \alpha\langle \nabla L(\theta), e_i(\theta)\rangle - \alpha\langle \nabla L(\theta'), e_i(\theta')\rangle + \tfrac{\alpha^2\lambda_i(\theta)}{2} + \tfrac{\alpha^2\lambda_i(\theta')}{2} + (c' + c'')\alpha^3\beta \\
&\leq L(\theta) - \alpha\langle \nabla L(\theta), e_i(\theta)\rangle - \alpha\langle \nabla L(\theta), e_i(\theta')\rangle + \tfrac{3\alpha^2\lambda_i(\theta)+\alpha^2\lambda_i(\theta')}{2} + (c' + c'' + c''')\alpha^3\beta \\
&= L(\theta) - 2\alpha\langle \nabla L(\theta), e_i(\theta)\rangle + \alpha\langle \nabla L(\theta), e_i(\theta) - e_i(\theta')\rangle + \tfrac{3\alpha^2\lambda_i(\theta)+\alpha^2\lambda_i(\theta')}{2} + (c' + c'' + c''')\alpha^3\beta \\
&\leq L(\theta) - 2\alpha\langle \nabla L(\theta), e_i(\theta)\rangle + \alpha\langle \nabla L(\theta), e_i(\theta) - e_i(\theta')\rangle + \tfrac{3\alpha^2\lambda_i(\theta)+\alpha^2\lambda_i(\theta')}{2} + (c' + c'' + c''')\alpha^3\beta \\
&\overset{(i)}{\leq} L(\theta) - 2\alpha\langle \nabla L(\theta), e_i(\theta)\rangle + \alpha\|\nabla L(\theta\| \sqrt{\tfrac{4\beta\|\theta-\theta'\|}{\min(\Delta_i,\Delta_{i-1})}} + \tfrac{3\alpha^2\lambda_i(\theta)+\alpha^2\lambda_i(\theta')}{2} + (c' + c'' + c''')\alpha^3\beta
\end{aligned}
$$

Where inequality (i) follows from Cauchy-Schwarz and Lemma 23 since:
$\|e_i(\theta) - e_i(\theta')\| \leq \sqrt{\frac{4\beta\|\theta-\theta'\|}{\min(\Delta_i,\Delta_{i-1})}} = \sqrt{\frac{4\beta\alpha}{\min(\Delta_i,\Delta_{i-1})}}$

Recall that by assumption $\langle \nabla L(\theta), e_i(\theta)\rangle \geq \|\nabla L(\theta)\|\gamma$ and $\gamma \in (0,1)$ and that $\alpha \leq \frac{\min(\Delta_i,\Delta_{i-1})\gamma^2}{16\beta}$. Applying this to inequality C.2.4 :

$$
\begin{aligned}
L(\theta'') &\leq L(\theta) - 2\alpha\langle \nabla L(\theta), e_i(\theta)\rangle + \alpha\|\nabla L(\theta\| \sqrt{\frac{4\beta\alpha}{\min(\Delta_i,\Delta_{i-1})}} \\
&\quad + \frac{3\alpha^2\lambda_i(\theta) + \alpha^2\lambda_i(\theta')}{2} + (c' + c'' + c''')\alpha^3\beta \\
&\leq L(\theta) - 2\alpha\langle \nabla L(\theta), e_i(\theta)\rangle + \frac{\gamma\alpha}{2}\|\nabla L(\theta\| + \frac{3\alpha^2\lambda_i(\theta) + \alpha^2\lambda_i(\theta')}{2} + (c' + c'' + c''')\alpha^3\beta \\
&\leq L(\theta) - \gamma\alpha\|\nabla L(\theta)\| + \frac{3\alpha^2\lambda_i(\theta) + \alpha^2\lambda_i(\theta')}{2} + (c' + c'' + c''')\alpha^3\beta \\
&\leq L(\theta) - \gamma\alpha\|\nabla L(\theta)\|
\end{aligned}
$$

The last inequality follows because term $\frac{3\alpha^2\lambda_i(\theta)+\alpha^2\lambda_i(\theta')}{2} \leq 0$ and of order less than the third degree terms at the end.

□

C.2.5 Behavior of RR near a saddle point

The discussion in this section is intended to be informal and has deliberately been written in this way. First, let θ_0 be a saddle point of $\mathcal{L}(\theta)$, and consider the steps of RR $\theta_1, ..., \theta_t, \cdots$. Let H be the Hessian of $\mathcal{L}$ at θ_0. We will start by using the first-order Taylor expansion, $\nabla_\theta \mathcal{L}(\theta_t) = H(\theta_{t-1})(\theta_t - \theta_{t-1}) + o(\epsilon^2)$ ignoring the error term to approximate the gradient close to θ_{t-1}.

We will see that $\nabla_\theta \mathcal{L}(\theta_t) = \alpha \sum_{l=0}^{t-1} \lambda_i(\theta_l) e_i(\theta_l) + o(t\epsilon^2)$ for all t. We proceed by induction. Notice that for $t = 1$, this is true since $\nabla_\theta \mathcal{L}(\theta_1) = \mathcal{H}(\theta_0)(\theta_1 - \theta_0) + o(\epsilon^2) = \alpha\lambda_i(\theta_0))e_i(\theta_0) + o(\epsilon^2)$ for some lower order error term ϵ.

Now suppose that for some $t \geq 1$ we have $\nabla_\theta \mathcal{L}(\theta_t) = \alpha \sum_{l=0}^{t-1} \lambda_i(\theta_l) e_i(\theta_l) + o(t\epsilon^2)$, this holds for $t = 0$. By a simple Taylor expansion around θ_t:

$$\begin{aligned}
\nabla_\theta \mathcal{L}(\theta_{t+1}) &= \nabla_\theta \mathcal{L}(\theta_t) + H(\theta_t)(\theta_{t+1} - \theta_t) + o(\epsilon^2) \\
&\overset{(i)}{=} \alpha \sum_{l=0}^{t-1} \lambda_i(\theta_l) e_i(\theta_l) + o(t\epsilon^2) + H(\theta_t)(\theta_{t+1} - \theta_t) + o(\epsilon^2) \\
&= \alpha \sum_{l=0}^{t} \lambda_i(\theta_l) e_i(\theta_l) + o((t+1)\epsilon^2)
\end{aligned}$$

Equality (i) follows from the inductive assumption. The last inequality follows because by definition $H(\theta_t)(\theta_{t+1} - \theta_t) = \alpha\lambda_i(\theta_t)e_i(\theta_t)$. The result follows.

C.2.6 Symmetries lead to repeated eigenvalues

Let $\mathcal{L} : {}^d \to {}^d$ be a twice-differentiable loss function and write $[n] = \{1, \ldots, n\}$ for $n \in \mathbb{N}$. For any permutation $\phi \in S_d$ (the symmetric group on d elements), consider the group action

$$\phi({}^{\text{th}}1, \ldots, {}^{\text{th}}d) = \left({}^{\text{th}}\phi(1), \ldots, {}^{\text{th}}\phi(d)\right)$$

and abuse notation by also writing $\phi : {}^d \to {}^d$ for the corresponding linear map. For any $N, m \in \mathbb{N}$, define the group of permutations

$$\Phi_N^m = \left\{ \prod_{i=1}^{m} (i, i + mk) \mid k \in [N-1] \right\}^2.$$

Now assume there are N non-overlapping sets

$$\{\theta_{k_i^1}, \ldots, \theta_{k_i^m}\}$$

of m parameters each, with $i \in [N]$, which we can reindex (by reordering parameters) to

$$\{\theta_{1+m(i-1)}, \ldots, \theta_{mi}\}$$

[2] For $m = 1$ we have $\Phi_N^1 = \{(1,2), \ldots, (1,N)\}$, which together generate all $N!$ permutations on N elements. For larger m, Φ_N^m also generates $N!$ permutations, but the m elements within each set are tied to each other. For instance, $\Phi_3^2 = \{(1,3)(2,4), (1,5)(2,6)\}$ which together generate $\{(1), (1,3)(2,4), (1,5)(2,6), (3,5)(4,6), (1,3,5)(2,4,6), (1,5,3)(2,6,4)\}$.

for convenience. Assume the loss function is invariant under all permutations of these N sets, namely, $\mathcal{L} \circ \phi = \mathcal{L}$ for all $\phi \in \Phi_N^m$. Our main result is that such parameter symmetries reduce the number of distinct Hessian eigenvalues, cutting down the number of directions to explore by ridge riding.

Theorem 26. *Assume that for some N, m we have $\mathcal{L} \circ \phi = \mathcal{L}$ and $\phi(^{th}) = {}^{th}$ for all $\phi \in \Phi_N^m$. Then $\nabla^2 \mathcal{L}(^{th})$ has at most $d - m(N-2)$ distinct eigenvalues.*

We first simplify notation and prove a few lemmata.

Definition 27. *Write $\Phi = \Phi_N^m$ and $H = \nabla^2 \mathcal{L}(^{th})$. We define an eigenvector v of H to be* trivial *if $\phi(v) = v$ for all $\phi \in \Phi$. We call an eigenvalue* trivial *if all corresponding eigenvectors are trivial.*

Lemma 28. *Assume $\mathcal{L} \circ \phi = \mathcal{L}$ for some $\phi \in \Phi$ and $\phi(^{th}) = {}^{th}$. If $(v,)$ is an eigenpair of H then so is $(\phi(v),)$.*

Proof. First notice that $D\phi$ (also written $\nabla\phi$) is constant by linearity of ϕ, and orthogonal since

$$(D\phi^T D\phi)_{ij} = \sum_k D\phi_{ki} D\phi_{kj} = \sum_k \delta_{\phi(k)i}\delta_{\phi(k)j} = \delta_{ij} \sum_k \delta_{\phi(k)i} = \delta_{ij} = I_{ij}\,.$$

Now applying the chain rule to $\mathcal{L} = \mathcal{L} \circ \phi$ we have

$$D\mathcal{L} = D\mathcal{L}|_\phi \circ D\phi$$

and applying the product rule and chain rule again,

$$D^2\mathcal{L} = D(D\mathcal{L}|_\phi \circ D\phi) = D\phi^T D^2\mathcal{L}|_\phi D\phi + 0$$

since $D^2\phi = 0$. If $\phi(^{\text{th}}) = {}^{\text{th}}$ then we obtain

$$H = D\phi^T H D\phi\,, \quad \text{or equivalently,} \quad H = D\phi H D\phi^T$$

by orthogonality of $D\phi$. Now notice that ϕ acts linearly as a matrix-vector product

$$\phi(v) = D\phi \cdot v\,,$$

so any eigenpair $(v,)$ of H must induce

$$H\phi(v) = (D\phi H D\phi^T)(D\phi v) = D\phi H v = D\phi v = D\phi v = \phi(v)$$

as required. □

Lemma 29. *Assume v is a non-trivial eigenvector of H with eigenvalue λ. Then has multiplicity at least $N-1$.*

Proof. Since v is non-trivial, there exists $\phi \in \Phi$ such that $\phi(v)_i \neq v_i$ for some $i \in [d]$. Without loss of generality, by reordering the parameters, assume $i = 1$. Since $\phi = \prod_{i=1}^{m}(i, i+mk)$ for some $k \in [N-1]$, we can set k to $N-1$ after reindexing of the N sets. Now $u = v - \phi(v)$ is an eigenvector of H with $u_1 \neq 0$ and zeros everywhere except the first and last m entries, since $k = N-1$ implies that ϕ keeps other entries fixed. We claim that

$$\left\{\prod_{i=1}^{m}(i, i+mk)u\right\}_{k=0}^{N-2}$$

are $N-1$ linearly independent vectors. Assume there are real numbers $a_0, \ldots, a_{N-2}$ such that

$$\sum_{k=0}^{N-2} a_k \prod_{i=1}^{m}(i, i+mk)u = 0\,.$$

In particular, noticing that $u_{1+mj} = 0$ for all $1 \leq j \leq N-2$ and considering the $(1+mj)$th entry for each such j yields

$$0 = \sum_{k=0}^{N-2} a_k \left(\prod_{i=1}^{m}(i, i+mk)u\right)_{1+mj} = \sum_{k=0}^{N-2} a_{kjk}u_1 = a_j u_1\,.$$

This implies $a_j = 1$ for all $1 \leq j \leq N-2$. Finally we are left with

$$0 = a_0 \prod_{i=1}^{m}(i, i)u = a_0 u$$

which implies $a_0 = 0$, so the vectors are linearly independent. By the previous lemma, each vector is an eigenvector with eigenvalue , so the eigenspace has dimension at least $N-1$ as required. □

Lemma 30. *There are at most $d-m(N-1)$ linearly independent trivial eigenvectors.*

Proof. Assume v is a trivial eigenvector, namely, $\phi(v) = v$ for all $\phi \in \Phi$. Then $v_i = v_{i+mk}$ for all $1 \leq i \leq m$ and $1 \leq k \leq N-1$, so v is fully determined by its first m entries $v_1, \ldots, v_m$ and its last $d - mN$ entries $v_{mN+1}, \ldots, v_d$. This implies that trivial eigenvectors have at most $m + d - mN = d - m(N-1)$ degrees of freedom, so there can be at most $d - m(N-1)$ linearly independent such vectors. □

The theorem now follows easily.

Proof. Let k and l respectively be the number of distinct trivial and non-trivial eigenvalues. Eigenvectors with distinct eigenvalues are linearly independent, so $k \leq d - m(N-1)$ by the previous lemma. Now assuming for contradiction that $k + l > d - m(N-2)$ implies

$$d - m(N-2) < k + l \leq d - m(N-1) + l \quad \Longrightarrow \quad l > m\,.$$

On the other hand, each non-trivial eigenvalue has multiplicity at least $N-1$, giving $k + l(N-1) \leq d$ linearly independent eigenvectors. We obtain the contradiction

$$d \geq k + l(N-1) = k + l + l(N-2) > d - m(N-2) + l(N-2) > d$$

and conclude that $k + l \leq d - m(N-2)$, as required. □

C.2.7 Maximally Invariant Saddle

In this section we show that for the case of tabular RL problems, the Maximally Invariant Saddle (MIS) corresponds to the parameter achieving the optimal reward and having the largest entropy.

We consider $\theta \in \mathbb{R}^{|S|\times|A|}$ the parametrization of a policy π_θ over an MDP with states S and actions A. We assume $\theta = \{\theta_s\}_{s\in S}$ with $\theta_s \in \mathbb{R}^{|A|}$ and (for simplicity) satisfying[3] $\sum_{a\in A} \theta_{s,a} = 1$ and $\theta_{s,a} \geq 0$.

Let Φ denote the set of symmetries over parameter space. In other words, $\phi \in \Phi$ if ϕ is a permutation over $|S| \times |A|$ and for all θ a valid policy parametrization, we have that $J(\theta) = J(\phi(\theta))$ such that $\phi(\theta) = \{\phi(\theta_s)\}_{s\in S}$ acting per state.

[3] A similar argument follows for a softmax parametrization.

We also assume the MDP is episodic in its state space, meaning the MDP has a horizon length of H and each state $s \in S$ is indexed by a horizon position h. No state is visited twice during an episode.

We show the following theorem:

Theorem 31. *Let Θ_b be the set of parameters that induce policies satisfying $J(\theta) = b$ for all $\theta \in \Theta_b$. Let $\theta^* \in \Theta_b$ be the parameter satisfying $\theta^* = \arg\max_{\theta \in \Theta_b} \sum_s \mathcal{H}(\pi_\theta(\mathbf{a}|s))$. Then for all $\phi \in \Phi$ it follows that $\phi(\theta^*) = \theta_*$.*

Proof. Let $\theta \in \Theta_b$ and let's assume there is a $\theta' \in \Theta_b$ such that $\phi(\theta') \neq \theta$. We will show there must exist $\theta'' \in \Theta_b$ such that:

$$\sum_s \mathcal{H}(\pi_{\theta''}(\mathbf{a}|s)) > \max\left(\sum_s \mathcal{H}(\pi_\theta(\mathbf{a}|s)), \sum_s \mathcal{H}(\pi_{\theta'}(\mathbf{a}|s))\right)$$

Let s be a state such that $\theta_s \neq \theta'_s$ and having maximal horizon position index h. In this case, all states s' with a horizon index larger than h satisfy $\theta_{s'} = \phi(\theta_{s'})$. Therefore for any s' having index $h+1$ (if any) it follows that the value function $V_\theta(s') = V_{\theta'}(s')$. Since the symmetries hold over any policy and specifically for delta policies, it must be the case that at state s and for any $a, a' \in A$ such that there is a $\phi' \in \Phi$ with (abusing notation) $\phi'(s, a) \to s, a'$ it must hold that $Q_\theta(s, a) = Q_\theta(s, a')$. Therefore the whole orbit of a under ϕ' for any $\phi' \in A$ has the same Q value under θ and θ'. Since the entropy is maximized when all the probabilities of these actions are the same, this implies that if θ does not correspond to a policy acting uniformly over the orbit of a at state s we can increase its entropy by turning it into a policy that acts uniformly over it. Applying this argument recursively down the different layers of the episodic MDP implies that for any $a \in A$, the maximum entropy $\theta \in \Theta_b$ assigns a uniform probability over all the actions on a's orbit. It is now easy to see that such a policy must satisfy $\phi(\theta) = \theta$ for all $\phi \in \Phi$.

□

We now show a result relating the entropy regularized gradient-norm objective:

$$\arg\min_\theta |\nabla_\theta J(\theta)| - \lambda \mathcal{H}(\pi_\theta(\mathbf{a})), \lambda > 0$$

C. Appendix for Diversity Algorithms

In this discussion we will consider a softmax parametrization for the policies. Let's start with the following lemma:

Lemma 32. *Let* $p_i(\theta) = \frac{\exp(\theta_i)}{\sum_j \exp(\theta_j))}$ *parametrize a policy over* K *reward values* $\{r_i\}_{i=1}^K$. *The value function's gradient satisfies:*

$$\left(\nabla \sum_{j=1}^{K} p_j(\theta)\right)_i = p_\theta(i)(r_i - \bar{r})$$

Where $\bar{r} = \sum_{j=1}^K p_i(\theta) r_i$.

Proof. Let $Z(\theta) = \sum_{j=1}^K \exp(\theta_j)$. The following equalities hold:

$$\begin{aligned}
\left(\nabla \sum_{j=1}^{K} p_j(\theta)\right)_i &= \frac{Z(\theta)\exp(\theta_i) r_i - \exp^2(\theta_i) r_i}{Z^2(\theta)} + \sum_{j \neq i} \frac{-\exp(\theta_j) r_j \exp(\theta_i)}{Z^2(\theta)} \\
&= \frac{Z(\theta)\exp(\theta_i) r_i}{Z^2(\theta)} - \sum_j \exp(\theta_i) \frac{\exp(\theta_j)}{Z^2(\theta)} \\
&= \frac{\exp(\theta_i) r_i}{Z(\theta)} - \frac{\exp(\theta_i)}{Z(\theta)} \left(\sum_j \frac{\exp(\theta_j) r_j}{Z(\theta)}\right) \\
&= p_i(\theta)\left(r_i - \bar{r}\right).
\end{aligned}$$

The result follows. □

We again consider an episodic MDP with horizon length of H and such that each state $s \in S$ is indexed by a horizon position h. No state is visited twice during an episode. Recall the set of symmetries is defined as $\phi \in \Phi$ if for any policy $\pi : \mathcal{S} \to \Delta_A$, with $Q-$function $Q_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, it follows that:

$$Q_\pi(\phi(s), \phi(a)) = Q_\pi(s, a). \tag{C.13}$$

We abuse notation and for any policy parameter $\theta \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|}$ we denote the parameter vector resulting of the action of a permutation on ϕ on the indices of a parameter vector θ by $\phi(\theta)$.

We show the following theorem:

Theorem 33. *Let Θ_b be the set of parameters that induce policies satisfying $\|\nabla J(\theta)\| = b$ for all $\theta \in \Theta_b$. Let $\theta^* \in \Theta_b$ be a parameter satisfying $\theta^* = \arg\max_{\theta \in \Theta_b} \sum_s H(\pi_\theta(\mathbf{a}|s))$ (there could be multiple optima). Then for all $\phi \in \Phi$ it follows that $\phi(\theta^*) = \theta_*$.*

Proof. Let $\theta \in \Theta_b$ and let's assume there is a $\phi \in \Phi$ such that $\theta' = \phi(\theta) \neq \theta$.

We will show there must exist $\theta'' \in \Theta_b$ such that:

$$\sum_s \mathcal{H}(\pi_{\theta''}(\mathbf{a}|s)) > \max\left(\sum_s \mathcal{H}(\pi_\theta(\mathbf{a}|s)), \sum_s \mathcal{H}(\pi_{\theta'}(\mathbf{a}|s))\right)$$

Since we are assuming a softmax parametrization and ϕ is a symmetry of the MDP, it must hold that for any two states s and s' with (abusing notation) $s' = \phi(s)$:

$$E_{a\sim\pi_\theta}[Q_{\pi(\theta)}(s,a)] = E_{a\sim\pi_{\phi(\theta)}}[Q_{\pi(\phi(\theta))}(s,a)]$$

We conclude that the gradient norm $\|\nabla J(\phi(\theta))\|$ must equal that of $\|\nabla J(\theta)\|$. This implies that if $\phi(s) \neq \phi(s')$ and wlog $\mathcal{H}(\pi_\theta(\mathbf{a}|s)) > \mathcal{H}(\pi_\theta(\mathbf{a}|\phi(s)))$, then we can achieve the same gradient norm but larger entropy by substituting $\theta_{\phi(s)}$ with θ_s. Where θ_s denotes the $|\mathcal{A}|$-dimensional vector of the policy parametrization for state s. The gradient norm would be preserved and the total entropy of the resulting policy would be larger of that achieved by θ and θ'. This finalizes the proof.

□

D

Appendix for World Model Algorithms

D.1 World Building: Ready Policy One

Table D.1: Hyperparameters used in the Policy

Hyperparameter Name	Value
Optimizer	Adam
Learning Rate	3e-4
Loss	PPO
Discount Factor	0.99
Batch Size	50,000
Epochs per Batch	10
ϵ-clip	0.2
Default Action σ	0.5
Hidden Layer Size	32
Number of Hidden Layers	2
Activation Function	ReLU

In terms of implementation, we follow ME-TRPO [148] with some adjustments. Instead of using the TRPO loss [265], we leverage the first-order approximation loss in PPO [267]. We do not apply parameter noise, and we modify policy training slightly by ensuring at least 10 updates are performed before termination is considered, which we find helps to improve convergence. We do not apply GAE nor the overall training approach in Schulman *et al.* [267], as this introduced instabilities. We instead found that generating large batch sizes gave better and

Table D.2: Hyperparameters used in the World Model

Hyperparameter Name	Value
Optimizer	Adam
Learning Rate	1e-3
Train/Validation Split	2:1
Number of Models	5
Batch Size	1,024
Hidden Layer Size	1,024
Number of Hidden Layers	2
Activation Function	ReLU
Early Stopping α in PCA	0.0005

more consistent performance, which corroborates the findings in Ilyas *et al.* [113] with respect to true policy gradients.

We augment each environment with an additional state which contains velocity information. This is also done in the 'FixedSwimmer' environment in Wang *et al.* [317], and allows us to infer the reward from the states directly. It must also be noted that in the original 'rllab' [58] environments used in Kurutach *et al.* [148], one of the observable states was the velocity state used to calculate rewards, and we therefore mirror this in our OpenAI Gym [28] implementation; we do not anticipate there to be any problem integrating reward prediction with our framework. Furthermore, we provide this state in both the model-free and model-based benchmarks to ensure there is no advantage, and do not notice any noticeable improvement in the model-free setting when this is provided; we hypothesize that some close proxy to the true velocity state already exists in the original state-space. We remove contact information from all environments, and instead of 'Swimmer-v2' we use the aforementioned 'FixedSwimmer', since this can be solved by our policy in a model-free regime. We remove early stopping from Hopper since we found it was necessary for convergence, but left the early stopping in for Ant since it was possible to train performant policies. We train the policy for 100 time steps in HalfCheetah and Ant, and for 200 time steps in Swimmer and Hopper. In experiments without the early stopping mechanism, data collection defaults to 3,000 timesteps per iteration. Full hyperparameter values can be found in Table D.2.

We use the following approach to normalize G to produce $\hat{G}$; for convenience, we write $\hat{G}_{\phi_t}(\theta_{t+1})$ as $\hat{G}_t$.

$$\hat{G}_t = \frac{G_t - \frac{1}{5}\sum_{\tau=t-5}^{t-1}(G_\tau)}{l_{\text{val}}} \tag{D.1}$$

where l_{val} is the final model validation loss from the iteration $t-1$.

Attention should be drawn to the α parameter used to determine early stopping in Algorithm 9. For the tasks we test on, we choose to fix this to 0.0005, and therefore do not tune it to be task specific. We found that at this setting of α, the early stopping mechanism generally collects significantly fewer than the default 3,000 samples (which acts as an upper bound in RP1), but can still expand the batch size collected to the full amount where appropriate (i.e., under policies that provide non-homogenous trajectories).

D.2 World Building: Coordinated Active Sample Collection

Implementation Details

DreamerV2 consists of an image encoder that uses a Convolutional Neural Network (CNN, [156]), a Recurrent State-Space Model (RSSM [98]) that learns the dynamics, and predictors for the image, reward, and discount factor. The RSSM uses a sequence of deterministic recurrent states h_t. At each step, it computes a posterior state z_t conditioned on the current image x_t, as well as a prior state $\hat{z}_t$ without the current image. During world model training, the concatenation of h_t and z_t is used to reconstruct the current image x_t, and predict the reward r_t and discount factor γ_t. Once the world model is trained, it can be used to roll out "imaginary" trajectories where the model state is instead the concatenation of the deterministic state h_t and prior stochastic state $\hat{z}_t$.

$$\text{RSSM} \begin{cases} \text{Recurrent model:} & h_t = f_\psi(h_{t-1}, z_{t-1}, a_{t-1}) \\ \text{Representation model:} & z_t \sim q_\psi(z_t|h_t, x_t) \\ \text{Transition predictor:} & \hat{z}_t \sim p_\psi(\hat{z}_t|h_t) \end{cases}$$

Image predictor: $\hat{x}_t \sim p_\psi(\hat{x}_t|h_t, z_t)$

Reward predictor: $\hat{r}_t \sim p_\psi(\hat{r}_t|h_t, z_t)$

Discount predictor: $\hat{\gamma}_t \sim p_\psi(\hat{\gamma}_t|h_t, z_t)$.

Our implementation (https://github.com/ainomearod/divwm) was built on top of the official DreamerV2 repository https://github.com/danijar/dreamerv2. We used the default hyperparameter values in the DreamerV2 repository. Table D.3 lists the additional hyperparameters used in our experiments.

Table D.3: CASCADE hyperparameters

Environment	Parameter	Value
MiniGrid-FourRooms	CASCADE weight (λ)	0.1
	batch size	200k
	deployments	5
	explorer train steps	10k
	offline model train steps	20k
MiniGrid-MultiRoom-N4S5	CASCADE weight (λ)	0.3
	batch size	200k
	deployments	5
	explorer train steps	10k
	offline model train steps	20k
Montezuma's Revenge	CASCADE weight (λ)	0.8
	batch size	200k
	deployments	15
	explorer train steps	10k
	offline model train steps	5k
Walker	CASCADE weight(λ)	0.1 (Random) 0.3 (Medium/Expert)
	batch size	200k
	deployments	2
	explorer train steps	250
	offline model train steps	5k

We also incorporate a latent disagreement ensemble, following Plan2Explore [269]. This involves training, alongside the RSSM model, an MLP ensemble with 10 members (each having 4 hidden layers, 400 units per layer, and different parameter initializations). These one-step models take action a_t and latents z_t, h_t as inputs, and try to predict the next stochastic latent z_{t+1}. The variance across these outputs during imagined rollouts is then used as the reward signal that forms the Information

Gain (InfoGain) objective. Note that this ensemble is otherwise unused, and is solely trained for the purposes of the exploration objective.

Proof of Lemma 8

In the proof of Lemma 8 we will make use of the following supporting result,

Lemma 34. *Let $p \in (0,1]$ and $p_1, p_2 > 0$ satisfying $p_1 + p_2 = p$ then,*

$$p \log(1/p) < p_1 \log(1/p_1) + p_2 \log(1/p_2).$$

Proof. Let $p_1 = \alpha p$ with $\alpha \neq 0, 1$, then,

$$\begin{aligned} p_1 \log(1/p_1) + p_2 \log(1/p_2) &= \alpha p \log(1/(\alpha p)) + (1-\alpha) p \log(1/((1-\alpha)p)) \\ &= p \log(1/p) + p \left(\underbrace{\alpha \log(1/\alpha) + (1-\alpha) \log(1/(1-\alpha))}_{>0} \right) \\ &> p \log(1/p). \end{aligned}$$

□

Lemma 34 implies the following result,

Lemma 35. *Let $\in [0,1]^K$ be a vector satisfying $\sum_{i=1}^K {}_i = 1$ and let $\mathcal{C}$ be a partition of $[K]$ such that $|\mathcal{C}| \leq K - 1$. For all $C \in \mathcal{C}$ denote $(C) = \sum_{i \in C} {}_i$. The following inequality holds,*

$$\sum_{C \in \mathcal{C}} (C) \log(1/(C)) \leq \sum_{i=1}^K {}_i \log(1/{}_i).$$

If ${}_i > 0$ the inequality is strict,

$$\sum_{C \in \mathcal{C}} (C) \log(1/(C)) < \sum_{i=1}^K {}_i \log(1/{}_i).$$

Proof. W.l.o.g we call C_1 the partition set containing 1 and assume ${}_1 > 0$. If ${}_i > 0$ for only one element of C_1, it must be the case that ${}_1 \log(1/{}_1) = {}_{C_1} \log(1/{}_{C_1})$.

We now assume that $|C_1| > 1$ and that C_1 has at least two indices $i \neq j$ such that ${}_{i,j} > 0$. Let l denote the size of C_1 and w.l.o.g. let's say $C_1 = \{1, 2, \cdots, l\}$. Lemma 34 implies the following inequalities,

$$
\begin{aligned}
(C_1)\log(1/(C_1)) &<_1 \log(1/_1) + (C_1\backslash\{1\})\log(1/(C_1\backslash\{1\})) \\
&< \cdots \\
&< \sum_{i\in C_1} {}_i \log(1/p_i).
\end{aligned}
$$

Applying this reasoning to each element in the partition $\mathcal{C}$ yields the desired result.

□

Lemma 8. *When all models $\mathcal{M}_\psi$ in the support of the model posterior are deterministic and tabular, and the space of policies Π consists only of deterministic policies, there always exists a solution $\{\pi^{(i)}_{\text{EXP}}\}_{i=1}^B$ satisfying $\pi^{(i)}_{\text{EXP}} \neq \pi^{(j)}_{\text{EXP}} \forall i \neq j$. Moreover, there exists a family of tabular MDP models, such that the maximum cannot be achieved by setting $\pi_{\text{EXP}}(i) = \pi$ for a fixed π.*

Proof. First let's observe that when the models and policies are all deterministic, the conditional entropies satisfy,

$$
\mathcal{H}\left(\prod_{i=1}^B \mathbb{P}^\Phi_{\pi^{(i)}}[\mathcal{M}_\psi]\middle|\mathcal{M}_\psi\right) = \sum_{i=1}^B \mathcal{H}\left(\mathbb{P}^\Phi_{\pi^{(i)}}[\mathcal{M}_\psi]\middle|\mathcal{M}_\psi\right) = 0.
$$

We assume the set of policies Π is of size at least B. Otherwise, the result cannot be true. The first result follows immediately by Lemma 35. For any fixed π, the product distribution $\prod_{i=1}^B \mathbb{P}^\Phi_\pi[\mathcal{M}_\psi]$ is a distribution over 'diagonal' tuples of the form $\underbrace{(b,\cdots,b)}_{\text{size } B}$ where b is an embedding.

Consider an arbitrary set of policies $\pi^{(2)},\cdots,\pi^{(B)}$ satisfying $\pi^{(i)} \neq \pi^{(j)} \neq \pi$ for all $i,j \in \{2,\cdots,K\}$. Call $\pi^{(1)} = \pi$. The distribution over embeddings induced by the product distribution $\prod_{i=1}^B \mathbb{P}^\Phi_{\pi^{(i)}}[\mathcal{M}_\psi]$ is made of tuples of the form $(b_1,\cdots,b_B)$. Call $(b_1,\cdots,b_B)$ the probability under the product measure $\prod_{i=1}^B \mathbb{P}^\Phi_{\pi^{(i)}}[\mathcal{M}_\psi]$ of the tuple $(b_1,\cdots,b_B)$. Notice the 'projection measure' onto the first coordinate satisfies

$$
\prod_{i=1}^B \mathbb{P}^\Phi_{\pi^{(i)}}[\mathcal{M}_\psi](b_1) = \prod_{i=1}^B \mathbb{P}^\Phi_\pi[\mathcal{M}_\psi](b_1,\cdots,b_1) :=_1 (b_1)
$$

and that $\sum_{b_2,\cdots,b_K}(b_1, b_2, \cdots, b_K) =_1 (b_1)$. This induces a partition over the outcome space $(b_1, \cdots, b_K)$ corresponding to $C(b_1) = \{(b_1, b_2, \cdots, b_K)\}_{b_2,\cdots,b_K}$.

A direct use of Lemma 35 implies the entropy of the product distribution over finer grained tuples is larger than the entropy over the diagonal of the product distribution having all coordinates equal to each other. In fact this result also informs when the inequality will be strict. This happens when (for example the measure over $\mathcal{M}_\psi$ is the counting measure) there exists two worlds $\mathcal{M}_\psi^{(1)}$ and $\mathcal{M}_\psi^{(2)}$ such that the embedding tuples $(b_1^{(1)}, \cdots, b_B^{(1)})$ and $(b_1^{(2)}, \cdots, b_B^{(2)})$ satisfy $b_1^{(1)} = b_1^{(2)}$ and $(b_2^{(1)}, \cdots, b_B^{(1)}) \neq (b_2^{(2)}, \cdots, b_B^{(B)})$.

We will use this observation to prove the second claim. Consider a family of MDPs formed of depth L binary trees. In this family, the paths leading to the $L-1$ layer are the same, but the connections from layer $L-1$ to layer L are unknown. The 'posterior' distribution is assumed to be uniform over all plausible trees. Layer $L-1$ has size 2^{L-1} and layer L has size 2^L. W.l.o.g. we assume the set of nodes in layer L is labeled as $[1, 2, \cdots, 2^L]$. The distribution over models is supported over the set of partitions of size 2^{L-1} of $[1, \cdots, 2^L]$ where each partition set has size 2. We assume $2^L \geq B$ so that there are at least B distinct policies.

Let π be a fixed policy. It is enough to show there exist two worlds $\mathcal{M}_\psi^{(1)}$ and $\mathcal{M}_\psi^{(2)}$ such that policy π ends in the same state for both $\mathcal{M}_\psi^{(1)}$ and $\mathcal{M}_\psi^{(2)}$ but that there exist distinct policies $\pi^{(2)}, \cdots, \pi^{(B)}$ such that their endpoints $(b_2^{(1)}, \cdots, b_B^{(1)})$ and $(b_2^{(2)}, \cdots, b_B^{(2)})$ in worlds $\mathcal{M}_\psi^{(1)}$ and $\mathcal{M}_\psi^{(2)}$ satisfy $(b_2^{(1)}, \cdots, b_B^{(1)}) \neq (b_2^{(2)}, \cdots, b_B^{(2)})$. The latter always holds because among the set of models that maintain the same endpoint for π, there is a pair of models that has different endpoints for $\pi^{(2)}$ for any arbitrary $\pi^{(2)} \neq \pi$. This suffices to show the entropy of the ensemble of distinct policies is strictly larger than the entropy of the 'diagonal choice' $\underbrace{(\pi, \cdots, \pi)}_{\text{size } B}$.

□

D.3 Augmented World Models

D.3.1 Hyperparameters

Our algorithm is based on MOPO [332] with values for the rollout length h and penalty coefficient λ shown in Table D.4.

Table D.4: Hyperparameters used in the D4RL datasets.

Dataset Type	Environment	MOPO (h, λ)
random	halfcheetah	5, 0.5
random	walker2d	1, 1
medium	halfcheetah	1, 1
medium	walker2d	1, 1
mixed	halfcheetah	5, 1
mixed	walker2d	1, 1
med-expert	halfcheetah	5, 1
med-expert	walker2d	1, 2

AugWM specific hyperparameters are listed in Table D.5. For each evaluation rollout, we clear the buffer of stored true modified environment transitions to measure zero-shot performance. We adapt using the context after a set number of steps, k, in the environment to train the linear model. The two ranges used for the context z during training and test time are different. At test time, the estimated context is clipped to remain within the given bounds.

Table D.5: AugWM Hyperparameters

Parameter	Value
evaluation rollouts	5
MOPO offline epochs	400
AugWM offline epochs	900
k, steps for adaptation	300
z train range	[0.5, 1.5]
z test range	[0.93, 1.07]

D.3.2 D4RL dataset

We evaluate our method on D4RL [79] datasets based on the MuJoCo continuous control tasks (halfcheetah and walker2d). The four dataset types we evaluate on are:

- **random:** roll out a randomly initialized policy for 1M steps.
- **medium:** partially train a policy using SAC, then roll it out for 1M steps.
- **mixed:** train a policy using SAC until a certain (environment-specific) performance threshold is reached, and take the replay buffer as the batch.
- **medium-expert:** combine 1M samples of rollouts from a fully-trained policy with another 1M samples of rollouts from a partially trained policy or a random policy.

This gives us a total of 8 experiments.

Bibliography

1. Agarwal, R., Machado, M. C., Castro, P. S. & Bellemare, M. G. *Contrastive Behavioral Similarity Embeddings for Generalization in Reinforcement Learning* in *International Conference on Learning Representations* (2021).
2. Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A. & Bellemare, M. G. *Deep Reinforcement Learning at the Edge of the Statistical Precipice* in *Advances in Neural Information Processing Systems* (2021).
3. Alayrac, J.-B., Donahue, J., Luc, P., Miech, A., Barr, I., Hasson, Y., Lenc, K., Mensch, A., Millican, K., Reynolds, M., *et al.* Flamingo: a visual language model for few-shot learning. *arXiv preprint arXiv:2204.14198* (2022).
4. Alvi, A., Ru, B., Calliess, J.-P., Roberts, S. & Osborne, M. A. *Asynchronous Batch Bayesian Optimisation with Improved Local Penalisation* in *Proceedings of the 36th International Conference on Machine Learning* (2019), 253–262.
5. Andrychowicz, M., Crow, D., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P. & Zaremba, W. *Hindsight Experience Replay* in *Advances in Neural Information Processing Systems 30* (eds Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N. & Garnett, R.) (2017).
6. Andrychowicz, M., Raichuk, A., Stańczyk, P., Orsini, M., Girgin, S., Marinier, R., Hussenot, L., Geist, M., Pietquin, O., Michalski, M., Gelly, S. & Bachem, O. *What Matters for On-Policy Deep Actor-Critic Methods? A Large-Scale Study* in *International Conference on Learning Representations* (2021).
7. Andrychowicz, O. M., Baker, B., Chociej, M., Józefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., Schneider, J., Sidor, S., Tobin, J., Welinder, P., Weng, L. & Zaremba, W. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research* **39,** 3–20 (2020).
8. Angeline, P. J. *Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences* in *Evolutionary Programming VII* (Springer Berlin Heidelberg, Berlin, Heidelberg, 1998), 601–610. ISBN: 978-3-540-68515-9.
9. Argenson, A. & Dulac-Arnold, G. *Model-Based Offline Planning* in *International Conference on Learning Representations* (2021).

10. Arulkumaran, K., Cully, A. & Togelius, J. *AlphaStar: An Evolutionary Computation Perspective* in *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (Association for Computing Machinery, Prague, Czech Republic, 2019), 314–315. ISBN: 9781450367486.

11. Auer, P., Cesa-Bianchi, N., Freund, Y. & Schapire, R. E. The Nonstochastic Multiarmed Bandit Problem. *SIAM journal on computing* **32,** 48–77 (2002).

12. Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, D. & Blundell, C. *Agent57: Outperforming the Atari Human Benchmark* in *Proceedings of the 37th International Conference on Machine Learning* (2020).

13. Badia, A. P., Sprechmann, P., Vitvitskyi, A., Guo, D., Piot, B., Kapturowski, S., Tieleman, O., Arjovsky, M., Pritzel, A., Bolt, A. & Blundell, C. *Never Give Up: Learning Directed Exploration Strategies* in *International Conference on Learning Representations* (2020).

14. Baker, B., Kanitscheider, I., Markov, T. M., Wu, Y., Powell, G., McGrew, B. & Mordatch, I. Emergent Tool Use From Multi-Agent Autocurricula. *CoRR* **abs/1909.07528.** arXiv: 1909.07528 (2019).

15. Balduzzi, D., Garnelo, M., Bachrach, Y., Czarnecki, W., Perolat, J., Jaderberg, M. & Graepel, T. *Open-ended learning in symmetric zero-sum games* in *International Conference on Machine Learning* (2019), 434–443.

16. Ball, P., Parker-Holder, J., Pacchiano, A., Choromanski, K. & Roberts, S. *Ready Policy One: World Building Through Active Learning* in *Proceedings of the 37th International Conference on Machine Learning* (2020).

17. Ball, P. J., Lu, C., Parker-Holder, J. & Roberts, S. J. *Augmented World Models Facilitate Zero-Shot Dynamics Generalization From a Single Offline Environment* in *The International Conference on Machine Learning* (2021).

18. Baranes, A. & Oudeyer, P.-Y. *Robust Intrinsically Motivated Exploration and Active Learning* in (July 2009), 1 –6.

19. Bellemare, M., Candido, S., Castro, P., Gong, J., Machado, M., Moitra, S., Ponda, S. & Wang, Z. Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature* **588,** 77–82 (2020).

20. Bellemare, M. G., Naddaf, Y., Veness, J. & Bowling, M. The Arcade Learning Environment: An Evaluation Platform for General Agents. *CoRR* **abs/1207.4708** (2012).

21. Bergstra, J. & Bengio, Y. *Random Search for Hyper-Parameter Optimization* in *Journal of Machine Learning Research* (2012).

22. Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., de Oliveira Pinto, H. P., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F. & Zhang, S. Dota 2 with Large Scale Deep Reinforcement Learning. *CoRR* **abs/1912.06680** (2019).

23. Berthouze, L. & Lungarella, M. Motor Skill Acquisition Under Environmental Perturbations: On the Necessity of Alternate Freezing and Freeing of Degrees of Freedom. *Adapt. Behav.* **12,** 47–64 (2004).

24. Bhaumik, D., Khalifa, A., Green, M. C. & Togelius, J. *Tree Search versus Optimization Approaches for Map Generation* in *AAAI 2020* (2020).

25. Bogunovic, I., Scarlett, J. & Cevher, V. *Time-Varying Gaussian Process Bandit Optimization* in *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics* (2016).

26. Brant, J. C. & Stanley, K. O. *Minimal Criterion Coevolution: A New Approach to Open-Ended Search* in *Proceedings of the Genetic and Evolutionary Computation Conference* (Association for Computing Machinery, Berlin, Germany, 2017), 67–74. ISBN: 9781450349208.

27. Brochu, E., Cora, V. M. & de Freitas, N. A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning. *CoRR* **abs/1012.2599** (2010).

28. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J. & Zaremba, W. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).

29. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., *et al.* Language models are few-shot learners. *Advances in neural information processing systems* **33,** 1877–1901 (2020).

30. Buckman, J., Hafner, D., Tucker, G., Brevdo, E. & Lee, H. *Sample-efficient reinforcement learning with stochastic ensemble value expansion* in *Advances in neural information processing systems* **31** (2018).

31. Byravan, A., Springenberg, J. T., Abdolmaleki, A., Hafner, R., Neunert, M., Lampe, T., Siegel, N., Heess, N. M. O. & Riedmiller, M. A. *Imagined Value Gradients: Model-Based Policy Optimization with Transferable Latent Dynamics Models* in *CoRL* (2019).

32. Campero, A., Raileanu, R., Kuttler, H., Tenenbaum, J. B., Rocktäschel, T. & Grefenstette, E. *Learning with AMIGo: Adversarially Motivated Intrinsic Goals* in *International Conference on Learning Representations* (2021).

33. Castillo, P. A., Rivas, V., Merelo, J. J., Gonzalez, J., Prieto, A. & Romero, G. *G-Prop-II: global optimization of multilayer perceptrons using GAs* in *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)* (1999).

34. Ceron, J. S. O. & Castro, P. S. *Revisiting Rainbow: Promoting more insightful and inclusive deep reinforcement learning research* in *Deep Reinforcement Learning Workshop, NeurIPS 2020* (2020).

35. Cesa-Bianchi, N. & Lugosi, G. *Prediction, learning, and games* (Cambridge university press, 2006).

36. Chakraborty, S., Balasubramanian, V. & Panchanathan, S. *Dynamic batch mode active learning* English (US). in *2011 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2011* (2011), 2649–2656. ISBN: 9781457703942.

37. Chen, Y., Huang, A., Wang, Z., Antonoglou, I., Schrittwieser, J., Silver, D. & de Freitas, N. Bayesian Optimization in AlphaGo. *CoRR* **abs/1812.06855** (2018).

38. Chevalier-Boisvert, M., Willems, L. & Pal, S. *Minimalistic Gridworld Environment for OpenAI Gym* https://github.com/maximecb/gym-minigrid. 2018.

39. Choromanski, K., Rowland, M., Sindhwani, V., Turner, R. E. & Weller, A. *Structured Evolution with Compact Architectures for Scalable Policy Optimization* in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018* (2018), 969–977.

40. Chua, K., Calandra, R., McAllister, R. & Levine, S. *Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models* in *Advances in Neural Information Processing Systems 31* (2018), 4754–4765.

41. Clavera, I., Rothfuss, J., Schulman, J., Fujita, Y., Asfour, T. & Abbeel, P. *Model-Based Reinforcement Learning via Meta-Policy Optimization* in *2nd Annual Conference on Robot Learning, CoRL 2018, Zürich, Switzerland, 29-31 October 2018, Proceedings* **87** (PMLR, 2018), 617–629.

42. Clune, J. AI-GAs: AI-generating algorithms, an alternate paradigm for producing general artificial intelligence. *CoRR* **abs/1905.10985** (2019).

43. Co-Reyes, J. D., Miao, Y., Peng, D., Le, Q. V., Levine, S., Lee, H. & Faust, A. *Evolving Reinforcement Learning Algorithms* in *International Conference on Learning Representations* (2021).

44. Cobbe, K., Hesse, C., Hilton, J. & Schulman, J. *Leveraging Procedural Generation to Benchmark Reinforcement Learning* in *Proceedings of the 37th International Conference on Machine Learning* (2020), 2048–2056.

45. Cobbe, K., Klimov, O., Hesse, C., Kim, T. & Schulman, J. Quantifying Generalization in Reinforcement Learning. *CoRR* **abs/1812.02341** (2018).

46. Constantine, P. G., Dow, E. & Wang, Q. Active Subspace Methods in Theory and Practice: Applications to Kriging Surfaces. *SIAM Journal on Scientific Computing* **36,** A1500–A1524 (2014).

47. Conti, E., Madhavan, V., Such, F. P., Lehman, J., Stanley, K. O. & Clune, J. *Improving Exploration in Evolution Strategies for Deep Reinforcement Learning via a Population of Novelty-seeking Agents* in *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (Curran Associates Inc., Montréal, Canada, 2018), 5032–5043.

48. Cui, B., Hu, H., Pineda, L. & Foerster, J. N. *K-level Reasoning for Zero-Shot Coordination in Hanabi* in *Advances in Neural Information Processing Systems* (eds Beygelzimer, A., Dauphin, Y., Liang, P. & Vaughan, J. W.) (2021).

49. Cully, A. *Autonomous Skill Discovery with Quality-Diversity and Unsupervised Descriptors* in *Proceedings of the Genetic and Evolutionary Computation Conference* (Association for Computing Machinery, Prague, Czech Republic, 2019), 81–89. ISBN: 9781450361118.

50. Cully, A., Clune, J., Tarapore, D. & Mouret, J.-B. Robots that can adapt like animals. *Nature* **521,** 503–507 (2015).

51. De Jong, E. D., Watson, R. A. & Pollack, J. B. *Reducing Bloat and Promoting Diversity Using Multi-Objective Methods* in *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation* (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001), 11–18. ISBN: 1558607749.

52. Deb, K. & Kalyanmoy, D. *Multi-Objective Optimization Using Evolutionary Algorithms* ISBN: 047187339X (John Wiley Sons, Inc., USA, 2001).

53. Degrave, J., Felici, F., Buchli, J., Neunert, M., Tracey, B., Carpanese, F., Ewalds, T., Hafner, R., Abdolmaleki, A., Casas, D., Donner, C., Fritz, L., Galperti, C., Huber, A., Keeling, J., Tsimpoukelli, M., Kay, J., Merle, A., Moret, J.-M. & Riedmiller, M. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature* **602,** 414–419 (Feb. 2022).

54. Dennis, M., Jaques, N., Vinitsky, E., Bayen, A., Russell, S., Critch, A. & Levine, S. *Emergent Complexity and Zero-shot Transfer via Unsupervised Environment Design* in *Advances in Neural Information Processing Systems* **33** (2020).

55. Desautels, T., Krause, A. & Burdick, J. W. Parallelizing Exploration-Exploitation Tradeoffs in Gaussian Process Bandit Optimization. *Journal of Machine Learning Research* **15,** 4053–4103 (2014).

56. Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

57. Downey, C. & Sanner, S. *Temporal Difference Bayesian Model Averaging: A Bayesian Perspective on Adapting Lambda* in *ICML* (2010), 311–318.

58. Duan, Y., Chen, X., Houthooft, R., Schulman, J. & Abbeel, P. Benchmarking Deep Reinforcement Learning for Continuous Control. *CoRR* **abs/1604.06778** (2016).

59. Duvenaud, D. The Kernel cookbook: Advice on covariance functions. https://www.cs.toronto.edu/\~duvenaud/cookbook (2014).

60. Earle, S., Edwards, M., Khalifa, A., Bontrager, P. & Togelius, J. *Learning Controllable Content Generators* in *IEEE Conference on Games (CoG)* (2021).

61. Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O. & Clune, J. First return then explore. *Nature* **590 7847,** 580–586 (2021).

62. Eimer, T., Biedenkapp, A., Hutter, F. & Lindauer, M. *Self-Paced Context Evaluation for Contextual Reinforcement Learning* in *The International Conference on Machine Learning* (2021).

63. Engstrom, L., Ilyas, A., Santurkar, S., Tsipras, D., Janoos, F., Rudolph, L. & Madry, A. *Implementation Matters in Deep RL: A Case Study on PPO and TRPO* in *International Conference on Learning Representations* (2020).

64. Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S. & Kavukcuoglu, K. *IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures* in *Proceedings of the 35th International Conference on Machine Learning* (2018).

65. Etcheverry, M., Moulin-Frier, C. & Oudeyer, P.-Y. *Hierarchically Organized Latent Modules for Exploratory Search in Morphogenetic Systems* in *Advances in Neural Information Processing Systems* (eds Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. & Lin, H.) **33** (Curran Associates, Inc., 2020), 4846–4859.

66. Everett, R., Cobb, A., Markham, A. & Roberts, S. *Optimising Worlds to Evaluate and Influence Reinforcement Learning Agents* in *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems* (2019).

67. Eysenbach, B., Gupta, A., Ibarz, J. & Levine, S. *Diversity is All You Need: Learning Skills without a Reward Function* in *International Conference on Learning Representations* (2019).

68. Falkner, S., Klein, A. & Hutter, F. *BOHB: Robust and Efficient Hyperparameter Optimization at Scale* in *Proceedings of the 35th International Conference on Machine Learning, ICML, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018* **80** (PMLR, 2018), 1436–1445.

69. Fang, K., Zhu, Y., Savarese, S. & Li, F.-F. *Adaptive Procedural Task Generation for Hard-Exploration Problems* in *International Conference on Learning Representations* (2021).

70. Finn, C., Abbeel, P. & Levine, S. *Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks* in *Proceedings of the 34th International Conference on Machine Learning, ICML, Sydney, NSW, Australia, 6-11 August* **70** (PMLR, 2017), 1126–1135.

71. Flennerhag, S., Schroecker, Y., Zahavy, T., van Hasselt, H., Silver, D. & Singh, S. *Bootstrapped Meta-Learning* in *The International Conference on Learning Representations* (2022).

72. Flet-Berliac, Y., Ferret, J., Pietquin, O., Preux, P. & Geist, M. *Adversarially Guided Actor-Critic* in *International Conference on Learning Representations* (2021).

73. Florensa, C., Held, D., Geng, X. & Abbeel, P. *Automatic Goal Generation for Reinforcement Learning Agents* in *Proceedings of the 35th International Conference on Machine Learning* (eds Dy, J. & Krause, A.) **80** (PMLR, 2018), 1515–1528.

74. Florensa, C., Held, D., Wulfmeier, M., Zhang, M. & Abbeel, P. *Reverse Curriculum Generation for Reinforcement Learning* in *1st Annual Conference on Robot Learning, CoRL 2017, Mountain View, California, USA, November 13-15, 2017, Proceedings* **78** (PMLR, 2017), 482–495.

75. Fortunato, M., Azar, M. G., Piot, B., Menick, J., Hessel, M., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., Blundell, C. & Legg, S. *Noisy Networks For Exploration* in *International Conference on Learning Representations* (2018).

76. Franke, J. K., Koehler, G., Biedenkapp, A. & Hutter, F. *Sample-Efficient Automated Deep Reinforcement Learning* in *International Conference on Learning Representations* (2021).

77. Freeman, C. D., Frey, E., Raichuk, A., Girgin, S., Mordatch, I. & Bachem, O. *Brax - A Differentiable Physics Engine for Large Scale Rigid Body Simulation* in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)* (2021).

78. Freeman, D., Ha, D. & Metz, L. *Learning to Predict Without Looking Ahead: World Models Without Forward Prediction* in *Advances in Neural Information Processing Systems 32* (Curran Associates, Inc., 2019), 5380–5391.

79. Fu, J., Kumar, A., Nachum, O., Tucker, G. & Levine, S. *D4{RL}: Datasets for Deep Data-Driven Reinforcement Learning* 2021.

80. Fujimoto, S., van Hoof, H. & Meger, D. *Addressing Function Approximation Error in Actor-Critic Methods* in *Proceedings of the 35th International Conference on Machine Learning* (eds Dy, J. & Krause, A.) **80** (PMLR, Stockholmsmässan, Stockholm Sweden, 2018), 1587–1596.

81. Gandhi, R., Khuller, S., Parthasarathy, S. & Srinivasan, A. Dependent rounding and its applications to approximation algorithms. *Journal of the ACM (JACM)* **53,** 324–360 (2006).

82. Garnelo, M., Czarnecki, W. M., Liu, S., Tirumala, D., Oh, J., Gidel, G., van Hasselt, H. & Balduzzi, D. Pick your battles: Interaction graphs as population-level objectives for strategic diversity. *arXiv preprint arXiv:2110.04041* (2021).

83. Geirhos, R., Jacobsen, J.-H., Michaelis, C., Zemel, R., Brendel, W., Bethge, M. & Wichmann, F. A. Shortcut learning in deep neural networks. *Nature Machine Intelligence* **2,** 665–673 (2020).

84. Ghosh, D., Rahme, J., Kumar, A., Zhang, A., Adams, R. P. & Levine, S. Why Generalization in RL is Difficult: Epistemic POMDPs and Implicit Partial Observability. *arXiv preprint arXiv:2107.06277* (2021).

85. Goldberg, D. E. *Simple Genetic Algorithms and the Minimal, Deceptive Problem* in (1987).

86. Gonzalez, J., Dai, Z., Hennig, P. & Lawrence, N. *Batch Bayesian Optimization via Local Penalization* in *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics* (2016), 648–657.

87. Goodfellow, I. J., Bengio, Y. & Courville, A. C. *Deep Learning* ISBN: 978-0-262-03561-3 (MIT Press, 2016).

88. GPy. *GPy: A Gaussian process framework in python* http://github.com/SheffieldML/GPy. since 2012.

89. Grbic, D., Palm, R. B., Najarro, E., Glanois, C. & Risi, S. *Evocraft: A new challenge for open-endedness* in *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)* (2021), 325–340.

90. Gregory, R. L. & Zangwill, O. L. *The Oxford companion to the mind.* (Oxford university press, 1987).

91. Gu, S., Lillicrap, T., Sutskever, I. & Levine, S. *Continuous Deep Q-Learning with Model-based Acceleration* in *Proceedings of The 33rd International Conference on Machine Learning* **48** (PMLR, 2016), 2829–2838.

92. Gur, I., Jaques, N., Miao, Y., Choi, J., Tiwari, M., Lee, H. & Faust, A. *Environment Generation for Zero-Shot Compositional Reinforcement Learning* in *Advances in Neural Information Processing Systems* (2021).

93. Gur, I., Nachum, O. & Faust, A. *Targeted Environment Design from Offline Data* 2022.

94. Ha, D. & Schmidhuber, J. *Recurrent World Models Facilitate Policy Evolution* in *Proceedings of the 32Nd International Conference on Neural Information Processing Systems* (2018), 2455–2467.

95. Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P. & Levine, S. Soft Actor-Critic Algorithms and Applications. *CoRR* **abs/1812.05905** (2018).

96. Hafner, D. *Benchmarking the Spectrum of Agent Capabilities* in *International Conference on Learning Representations* (2022).

97. Hafner, D., Lillicrap, T., Ba, J. & Norouzi, M. *Dream to Control: Learning Behaviors by Latent Imagination* in *International Conference on Learning Representations* (2020).

98. Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H. & Davidson, J. *Learning Latent Dynamics for Planning from Pixels* in *Proceedings of the 36th International Conference on Machine Learning* (eds Chaudhuri, K. & Salakhutdinov, R.) (2019), 2555–2565.

99. Hafner, D., Lillicrap, T. P., Norouzi, M. & Ba, J. *Mastering Atari with Discrete World Models* in *International Conference on Learning Representations* (2021).

100. Hambro, E., Mohanty, S. P., Babaev, D., Byeon, M.-R., Chakraborty, D., Grefenstette, E., Jiang, M., Jo, D., Kanervisto, A., Kim, J., Kim, S., Kirk, R., Kurin, V., Kuttler, H., Kwon, T., Lee, D., Mella, V., Nardelli, N., Nazarov, I., Ovsov, N., Parker-Holder, J., Raileanu, R., Ramanauskas, K., Rocktaschel, T., Rothermel, D., Samvelyan, M., Sorokin, D., Sypetkowski, M. & Sypetkowski, M. *Insights From the NeurIPS 2021 NetHack Challenge* in (2022).

101. He, K., Zhang, X., Ren, S. & Sun, J. Deep Residual Learning for Image Recognition. *CoRR* **abs/1512.03385** (2015).

102. Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D. & Meger, D. Deep Reinforcement Learning that Matters. *AAAI* (2018).

103. Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M. G. & Silver, D. Rainbow: Combining Improvements in Deep Reinforcement Learning. *AAAI* (2018).

104. Hong, Z.-W., Shann, T.-Y., Su, S.-Y., Chang, Y.-H., Fu, T.-J. & Lee, C.-Y. *Diversity-Driven Exploration Strategy for Deep Reinforcement Learning* in *Advances in Neural Information Processing Systems 31* (2018).

105. Houthooft, R., Chen, X., Duan, Y., Schulman, J., De Turck, F. & Abbeel, P. Vime: Variational information maximizing exploration. *Advances in neural information processing systems* **29** (2016).

106. Hu, H., Peysakhovich, A., Lerer, A. & Foerster, J. *"Other-Play"for Zero-Shot Coordination* in *Proceedings of Machine Learning and Systems 2020* (2020), 9396–9407.

107. Husken, M., Gayko, J. E. & Sendhoff, B. *Optimization for problem classes-neural networks that learn to learn* in *2000 IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks. Proceedings of the First IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks (Cat. No.00* (2000).

108. Hutter, F., Hoos, H. H. & Leyton-Brown, K. *Sequential Model-Based Optimization for General Algorithm Configuration* in *Learning and Intelligent Optimization* (ed Coello, C. A. C.) (Springer Berlin Heidelberg, 2011).

109. *Automated Machine Learning: Methods, Systems, Challenges* (eds Hutter, F., Kotthoff, L. & Vanschoren, J.) In press, available at http://automl.org/book. (Springer, 2018).

110. Hutter, M. *Universal Artificial Intellegence - Sequential Decisions Based on Algorithmic Probability* ISBN: 978-3-540-22139-5 (Springer, 2005).

111. Igl, M., Ciosek, K., Li, Y., Tschiatschek, S., Zhang, C., Devlin, S. & Hofmann, K. in *Advances in Neural Information Processing Systems* (2019).

112. Igl, M., Farquhar, G., Luketina, J., Boehmer, W. & Whiteson, S. *Transient Non-stationarity and Generalisation in Deep Reinforcement Learning* in *International Conference on Learning Representations* (2021).

113. Ilyas, A., Engstrom, L., Santurkar, S., Tsipras, D., Janoos, F., Rudolph, L. & Madry, A. Are Deep Policy Gradient Algorithms Truly Policy Gradient Algorithms? *CoRR* **abs/1811.02553** (2018).

114. Islam, R., Henderson, P., Gomrokchi, M. & Precup, D. Reproducibility of Benchmarked Deep Reinforcement Learning Tasks for Continuous Control. *CoRR* **abs/1708.04133** (2017).

115. Jaderberg, M., Czarnecki, W. M., Dunning, I., Marris, L., Lever, G., Castañeda, A. G., Beattie, C., Rabinowitz, N. C., Morcos, A. S., Ruderman, A., Sonnerat, N., Green, T., Deason, L., Leibo, J. Z., Silver, D., Hassabis, D., Kavukcuoglu, K. & Graepel, T. Human-level performance in 3D multiplayer games with population-based reinforcement learning. *Science* **364,** 859–865 (2019).

116. Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., Fernando, C. & Kavukcuoglu, K. Population Based Training of Neural Networks. *CoRR* **abs/1711.09846** (2017).

117. Jakobi, N. Evolutionary Robotics and the Radical Envelope-of-Noise Hypothesis. *Adaptive Behavior* **6,** 325–368 (1997).

118. James, S., Davison, A. J. & Johns, E. *Transferring End-to-End Visuomotor Control from Simulation to Real World for a Multi-Stage Task* in *CoRL* (2017).

119. Janner, M., Fu, J., Zhang, M. & Levine, S. *When to Trust Your Model: Model-Based Policy Optimization* in *Advances in Neural Information Processing Systems* (2019).

120. Jiang, M., Dennis, M., Parker-Holder, J., Foerster, J., Grefenstette, E. & Rocktäschel, T. *Replay-Guided Adversarial Environment Design* in *Advances in Neural Information Processing Systems* (2021).

121. Jiang, M., Grefenstette, E. & Rocktäschel, T. *Prioritized level replay* in *The International Conference on Machine Learning* (2021).

122. Juliani, A., Khalifa, A., Berges, V., Harper, J., Teng, E., Henry, H., Crespi, A., Togelius, J. & Lange, D. *Obstacle Tower: A Generalization Challenge in Vision, Control, and Planning* in *IJCAI* (2019).

123. Jung, W., Park, G. & Sung, Y. *Population-Guided Parallel Policy Search for Reinforcement Learning* in *International Conference on Learning Representations* (2020).

124. Justesen, N., Torrado, R. R., Bontrager, P., Khalifa, A., Togelius, J. & Risi, S. Procedural Level Generation Improves Generality of Deep Reinforcement Learning. *CoRR* **abs/1806.10729** (2018).

125. Kaiser, L., Babaeizadeh, M., Milos, P., Osiński, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., Mohiuddin, A., Sepassi, R., Tucker, G. & Michalewski, H. *Model Based Reinforcement Learning for Atari* in *International Conference on Learning Representations* (2020).

126. Kalashnikov, D., Irpan, A., Pastor, P., Ibarz, J., Herzog, A., Jang, E., Quillen, D., Holly, E., Kalakrishnan, M., Vanhoucke, V. & Levine, S. *Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation* in *Proceedings of The 2nd Conference on Robot Learning* (eds Billard, A., Dragan, A., Peters, J. & Morimoto, J.) **87** (PMLR, 2018), 651–673.

127. Kato, T. *Perturbation Theory for Linear Operators* 2nd ed. ISBN: 3-540-58661-X (Springer, 1995).

128. Kearns, M. J. & Singh, S. P. *Bias-Variance Error Bounds for Temporal Difference Updates* in *Proceedings of the Thirteenth Annual Conference on Computational Learning Theory* (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000), 142–147. ISBN: 155860703X.

129. Keren, S., Pineda, L., Gal, A., Karpas, E. & Zilberstein, S. in *Proceedings of the International Conference on Automated Planning and Scheduling* (2017).

130. Keren, S., Pineda, L., Gal, A., Karpas, E. & Zilberstein, S. *Efficient heuristic search for optimal environment redesign* in *Proceedings of the International Conference on Automated Planning and Scheduling* **29** (2019), 246–254.

131. Khalifa, A., Bontrager, P., Earle, S. & Togelius, J. PCGRL: Procedural Content Generation via Reinforcement Learning. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* **16,** 95–101 (2020).

132. Kidambi, R., Rajeswaran, A., Netrapalli, P. & Joachims, T. *MOReL : Model-Based Offline Reinforcement Learning* in *Advances in Neural Information Processing Systems* (2020).

133. Killian, T., Konidaris, G. & Doshi-Velez, F. Robust and Efficient Transfer Learning with Hidden Parameter Markov Decision Processes. *Proceedings of the AAAI Conference on Artificial Intelligence* **31** (2017).

134. Kingma, D. P. & Ba, J. *Adam: A Method for Stochastic Optimization* in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (eds Bengio, Y. & LeCun, Y.) (2015).

135. Kingma, D. P. & Welling, M. *Auto-encoding variational bayes* in *ICLR* (2014).

136. Kirk, R., Zhang, A., Grefenstette, E. & Rocktäschel, T. A Survey of Generalisation in Deep Reinforcement Learning. *CoRR* **abs/2111.09794** (2021).

137. Kirsch, L., van Steenkiste, S. & Schmidhuber, J. *Improving Generalization in Meta Reinforcement Learning using Learned Objectives* in *International Conference on Learning Representations* (2020).

138. Klink, P., Abdulsamad, H., Belousov, B. & Peters, J. *Self-Paced Contextual Reinforcement Learning* in *Conference on Robot Learning* (Oct. 2019).

139. Kolmogorov, A. N. On Tables of Random Numbers (Reprinted from "Sankhya: The Indian Journal of Statistics", Series A, Vol. 25 Part 4, 1963). *Theor. Comput. Sci.* **207,** 387–395 (1998).

140. Kostrikov, I. *PyTorch Implementations of Reinforcement Learning Algorithms* https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail. 2018.

141. Kostrikov, I., Yarats, D. & Fergus, R. *Image Augmentation Is All You Need: Regularizing Deep Reinforcement Learning from Pixels* in *International Conference on Learning Representations* (2021).

142. Krause, A. & Ong, C. S. *Contextual gaussian process bandit optimization* in *Advances in Neural Information Processing Systems* (2011), 2447–2455.

143. Krause, A., Singh, A. & Guestrin, C. in *Journal of Machine Learning Research* (2008).

144. Krizhevsky, A., Sutskever, I. & Hinton, G. E. *ImageNet Classification with Deep Convolutional Neural Networks* in *Advances in Neural Information Processing Systems 25* (eds Pereira, F., Burges, C. J. C., Bottou, L. & Weinberger, K. Q.) (2012).

145. Kulesza, A. & Taskar, B. *Determinantal Point Processes for Machine Learning* ISBN: 1601986289, 9781601986283 (Now Publishers Inc., Hanover, MA, USA, 2012).

146. Kumar, A., Fu, Z., Pathak, D. & Malik, J. Rma: Rapid motor adaptation for legged robots. *arXiv preprint arXiv:2107.04034* (2021).

147. Kumar, S., Kumar, A., Levine, S. & Finn, C. *One Solution is Not All You Need: Few-Shot Extrapolation via Structured MaxEnt RL* in *Advances in Neural Information Processing Systems* (eds Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. & Lin, H.) (2020).

148. Kurutach, T., Clavera, I., Duan, Y., Tamar, A. & Abbeel, P. *Model-Ensemble Trust-Region Policy Optimization* in *International Conference on Learning Representations* (2018).

149. Küttler, H., Nardelli, N., Miller, A. H., Raileanu, R., Selvatici, M., Grefenstette, E. & Rocktäschel, T. *The NetHack Learning Environment* in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)* (2020).

150. Lakshminarayanan, B., Pritzel, A. & Blundell, C. *Simple and Scalable Predictive Uncertainty Estimation Using Deep Ensembles* in *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Curran Associates Inc., Long Beach, California, USA, 2017), 6405–6416. ISBN: 9781510860964.

151. Lambert, N., Wulfmeier, M., Whitney, W. F., Byravan, A., Bloesch, M., Dasagi, V., Hertweck, T. & Riedmiller, M. A. The Challenges of Exploration for Offline Reinforcement Learning. *CoRR* (2022).

152. Lanctot, M., Zambaldi, V., Gruslys, A., Lazaridou, A., Tuyls, K., Perolat, J., Silver, D. & Graepel, T. *A Unified Game-Theoretic Approach to Multiagent Reinforcement Learning* in *Advances in Neural Information Processing Systems* (eds Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S. & Garnett, R.) **30** (Curran Associates, Inc., 2017).

153. Land, A. H. & Doig, A. G. An Automatic Method of Solving Discrete Programming Problems. *Econometrica* **28,** 497–520. ISSN: 00129682, 14680262 (1960).

154. Laskin, M., Lee, K., Stooke, A., Pinto, L., Abbeel, P. & Srinivas, A. *Reinforcement Learming with Augmented Data* in *Advances in Neural Information Processing Systems 33* (2020).

155. Laskin, M., Yarats, D., Liu, H., Lee, K., Zhan, A., Lu, K., Cang, C., Pinto, L. & Abbeel, P. *URLB: Unsupervised Reinforcement Learning Benchmark* in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)* (2021).

156. LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. & Jackel, L. D. *Backpropagation Applied to Handwritten Zip Code Recognition* in *Neural Computation* **1** (1989), 541–551.

157. Legg, S. & Hutter, M. *A Collection of Definitions of Intelligence* in *Advances in Artificial General Intelligence: Concepts, Architectures and Algorithms - Proceedings of the AGI Workshop 2006 [May 20-21, 2006, Washington DC, USA]* (eds Goertzel, B. & Wang, P.) **157** (IOS Press, 2006), 17–24.

158. Legg, S. & Hutter, M. Universal Intelligence: A Definition of Machine Intelligence. *Minds Mach.* **17,** 391–444. ISSN: 0924-6495 (2007).

159. Lehman, J., Gordon, J., Jain, S., Ndousse, K., Yeh, C. & Stanley, K. O. Evolution through Large Models. *arXiv* (2022).

160. Lehman, J. & Stanley, K. O. *Exploiting open-endedness to solve problems through the search for novelty* in *Proceedings of the Eleventh International Conference on Artificial Life (Alife XI* (MIT Press, 2008).

161. Lehman, J. & Stanley, K. O. *Evolving a Diversity of Virtual Creatures through Novelty Search and Local Competition* in *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation* (Association for Computing Machinery, Dublin, Ireland, 2011), 211–218. ISBN: 9781450305570.

162. Lehman, J. & Stanley, K. O. Abandoning Objectives: Evolution Through the Search for Novelty Alone. *Evolutionary Computation* **19,** 189–223 (2011).

163. Lehman, J., Wilder, B. & Stanley, K. O. On the Critical Role of Divergent Selection in Evolvability. *Frontiers Robotics AI* **3,** 45 (2016).

164. Leibo, J. Z., Hughes, E., Lanctot, M. & Graepel, T. Autocurricula and the Emergence of Innovation from Social Interaction: A Manifesto for Multi-Agent Intelligence Research. *ArXiv* **abs/1903.00742** (2019).

165. Levine, S., Kumar, A., Tucker, G. & Fu, J. *Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems* 2020.

166. Levine, S., Pastor, P., Krizhevsky, A., Ibarz, J. & Quillen, D. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research* **37,** 421–436 (2018).

167. Li, A., Spyra, O., Perel, S., Dalibard, V., Jaderberg, M., Gu, C., Budden, D., Harley, T. & Gupta, P. in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2019).

168. Li, L., Jamieson, K. G., Rostamizadeh, A., Gonina, E., Hardt, M., Recht, B. & Talwalkar, A. Massively Parallel Hyperparameter Tuning. *CoRR* (2018).

169. Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A. & Talwalkar, A. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *J. Mach. Learn. Res.* **18,** 6765–6816. ISSN: 1532-4435 (Jan. 2017).

170. Liang, E., Liaw, R., Nishihara, R., Moritz, P., Fox, R., Goldberg, K., Gonzalez, J. E., Jordan, M. I. & Stoica, I. *RLlib: Abstractions for Distributed Reinforcement Learning* in *International Conference on Machine Learning (ICML)* (2018).

171. Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J. E. & Stoica, I. Tune: A Research Platform for Distributed Model Selection and Training. *arXiv preprint* (2018).

172. Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. & Wierstra, D. *Continuous control with deep reinforcement learning* in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings* (eds Bengio, Y. & LeCun, Y.) (2016).

173. Liu, S., Lever, G., Heess, N., Merel, J., Tunyasuvunakool, S. & Graepel, T. *Emergent Coordination Through Competition* in *International Conference on Learning Representations* (2019).

174. Lorraine, J., Vicol, P., Parker-Holder, J., Kachman, T., Metz, L. & Foerster, J. *Lyapunov Exponents for Diversity in Differentiable Games* in *AAMAS* (2022).

175. Lu, C., Ball, P., Parker-Holder, J., Osborne, M. & Roberts, S. J. *Revisiting Design Choices in Offline Model Based Reinforcement Learning* in *International Conference on Learning Representations* (2022).

176. Lu, C., Ball, P. J., Rudner, T. G., Parker-Holder, J., Osborne, M. A. & Teh, Y. W. Challenges and opportunities in offline reinforcement learning from visual observations. *arXiv preprint arXiv:2206.04779* (2022).

177. Lupu, A., Cui, B., Hu, H. & Foerster, J. *Trajectory Diversity for Zero-Shot Coordination* in *Proceedings of the 38th International Conference on Machine Learning* (eds Meila, M. & Zhang, T.) **139** (PMLR, 2021), 7204–7213.

178. MacKay, D. J. C. Information-Based Objective Functions for Active Data Selection. *Neural Computation* **4,** 590–604. ISSN: 0899-7667 (1992).

179. MacKay, D. J. C. *Information Theory, Inference & Learning Algorithms* ISBN: 0521642981 (Cambridge University Press, USA, 2002).

180. Mahfoud, S. *Handbook of Evolutionary Computation, chapter Niching Methods* 1997.

181. Mann, H. B. & Whitney, D. R. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics* **18,** 50 –60 (1947).

182. Masood, M. A. & Doshi-Velez, F. *Diversity-Inducing Policy Gradient: Using Maximum Mean Discrepancy to Find a Set of Diverse Policies* in *IJCAI* (2019), 5923–5929.

183. Matiisen, T., Oliver, A., Cohen, T. & Schulman, J. Teacher-Student Curriculum Learning. *IEEE Trans. Neural Networks Learn. Syst.* **31,** 3732–3740 (2020).

184. Matsushima, T., Furuta, H., Matsuo, Y., Nachum, O. & Gu, S. *Deployment-Efficient Reinforcement Learning via Model-Based Offline Optimization* in *International Conference on Learning Representations* (2021).

185. Mauldin, M. L. *Maintaining Diversity in Genetic Search* in *AAAI* (1984).

186. Mazumdar, E., Ratliff, L. J. & Sastry, S. S. On Gradient-Based Learning in Continuous Games. *SIAM J. Math. Data Sci.* **2,** 103–131 (2020).

187. Mehta, B., Diaz, M., Golemo, F., Pal, C. J. & Paull, L. *Active Domain Randomization* in *Proceedings of the Conference on Robot Learning* (2020).

188. Mendonca, R., Rybkin, O., Daniilidis, K., Hafner, D. & Pathak, D. *Discovering and Achieving Goals via World Models* in *Advances in Neural Information Processing Systems* (2021).

189. Miki, T., Lee, J., Hwangbo, J., Wellhausen, L., Koltun, V. & Hutter, M. Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science Robotics* **7,** eabk2822 (2022).

190. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A., Veness, J., Bellemare, M., Graves, A., Riedmiller, M., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. & Hassabis, D. Human-level control through deep reinforcement learning. *Nature* **518,** 529–33 (Feb. 2015).

191. Modi, A., Jiang, N., Singh, S. & Tewari, A. in *Algorithmic Learning Theory* (2017).

192. Mortenson, M. E. *Mathematics for Computer Graphics Applications* (Industrial Press Inc., 1999).

193. Moskovitz, T., Arbel, M., Huszar, F. & Gretton, A. *Efficient Wasserstein Natural Gradients for Reinforcement Learning* in *International Conference on Learning Representations* (2021).

194. Moskovitz, T. H., Parker-Holder, J., Pacchiano, A., Arbel, M. & Jordan, M. I. *Tactical Optimism and Pessimism for Deep Reinforcement Learning* in *Advances in Neural Information Processing Systems* (2021).

195. Mouret, J.-B. & Doncieux, S. *Using Behavioral Exploration Objectives to Solve Deceptive Problems in Neuro-Evolution* in *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation* (Association for Computing Machinery, Montreal, Québec, Canada, 2009), 627–634. ISBN: 9781605583259.

196. Mouret, J.-B. & Doncieux, S. Encouraging Behavioral Diversity in Evolutionary Robotics: an Empirical Study. *Evolutionary Computation* **20,** 91–133 (2012).

197. Mu, J., Zhong, V., Raileanu, R., Jiang, M., Goodman, N., Rocktäschel, T. & Grefenstette, E. *Improving Intrinsic Exploration with Language Abstractions* 2022.

198. Nachum, O., Ahn, M., Ponte, H., Gu, S. S. & Kumar, V. *Multi-Agent Manipulation via Locomotion using Hierarchical Sim2Real* in *Proceedings of the Conference on Robot Learning* (2020), 110–121.

199. Nemhauser, G. L., Wolsey, L. A. & Fisher, M. L. An analysis of approximations for maximizing submodular set functions—I. *Mathematical programming* **14,** 265–294 (1978).

200. Nesterov, Y. & Spokoiny, V. Random Gradient-Free Minimization of Convex Functions. *Found. Comput. Math.* **17,** 527–566. ISSN: 1615-3375 (Apr. 2017).

201. Nguyen, V., Schulze, S. & Osborne, M. A. *Bayesian optimization for iterative learning* in *Advances in Neural Information Processing Systems* (2020).

202. Nichol, A., Pfau, V., Hesse, C., Klimov, O. & Schulman, J. Gotta Learn Fast: A New Benchmark for Generalization in RL. *CoRR* **abs/1804.03720** (2018).

203. Nilsson, O. & Cully, A. *Policy Gradient Assisted MAP-Elites* in *Proceedings of the Genetic and Evolutionary Computation Conference* (Association for Computing Machinery, Lille, France, 2021), 866–875. ISBN: 9781450383509.

204. Nix, D. A. & Weigend, A. S. *Estimating the mean and variance of the target probability distribution* in *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)* **1** (1994), 55–60 vol.1.

205. Nocedal, J. & Wright, S. J. *Numerical Optimization* second (Springer, New York, NY, USA, 2006).

206. O'Donoghue, B., Osband, I. & Ionescu, C. *Making Sense of Reinforcement Learning and Probabilistic Inference* in *International Conference on Learning Representations* (2020).

207. Oh, J., Hessel, M., Czarnecki, W. M., Xu, Z., van Hasselt, H., Singh, S. & Silver, D. *Discovering Reinforcement Learning Algorithms* in *Advances in Neural Information Processing Systems 33* (2020).

208. OpenAI. *Spinning Up in Deep RL* https://spinningup.openai.com/en/latest/. 2018.

209. OpenAI, Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., Schneider, J., Tezak, N., Tworek, J., Welinder, P., Weng, L., Yuan, Q., Zaremba, W. & Zhang, L. Solving Rubik's Cube with a Robot Hand. *CoRR* **abs/1910.07113** (2019).

210. OpenAI, O., Plappert, M., Sampedro, R., Xu, T., Akkaya, I., Kosaraju, V., Welinder, P., D'Sa, R., Petron, A., de Oliveira Pinto, H. P., Paino, A., Noh, H., Weng, L., Yuan, Q., Chu, C. & Zaremba, W. *Asymmetric self-play for automatic goal discovery in robotic manipulation* 2021.

211. Osband, I., Roy, B. V., Russo, D. J. & Wen, Z. *Deep Exploration via Randomized Value Functions* in *Journal of Machine Learning Research* **20** (2019), 1–62.

212. Pacchiano, A., Parker-Holder, J., Tang, Y., Choromanska, A., Choromanski, K. & Jordan, M. I. *Learning to Score Behaviors for Guided Policy Optimization* in *Proceedings of the 37th International Conference on Machine Learning* (2020).

213. Packard, N., Bedau, M. A., Channon, A., Ikegami, T., Rasmussen, S., Stanley, K. O. & Taylor, T. An Overview of Open-Ended Evolution: Editorial Introduction to the Open-Ended Evolution Ii Special Issue. *Artif. Life* **25,** 93–103. ISSN: 1064-5462 (2019).

214. Packer*, C., Gao*, K., Kos, J., Krahenbuhl, P., Koltun, V. & Song, D. *Assessing Generalization in Deep Reinforcement Learning* 2019.

215. Parker-Holder, J., Jiang, M., Dennis, M., Samvelyan, M., Foerster, J., Grefenstette, E. & Rocktäschel, T. *Evolving Curricula with Regret-Based Environment Design* in *The International Conference on Machine Learning* (2022).

216. Parker-Holder, J., Nguyen, V., Desai, S. & Roberts, S. *Tuning Mixed Input Hyperparameters on the Fly for Efficient Population Based AutoRL* in *Advances in Neural Information Processing Systems* (eds Beygelzimer, A., Dauphin, Y., Liang, P. & Vaughan, J. W.) (2021).

217. Parker-Holder, J., Nguyen, V. & Roberts, S. J. *Provably Efficient Online Hyperparameter Optimization with Population-Based Bandits* in *Advances in Neural Information Processing Systems* (eds Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F. & Lin, H.) **33** (Curran Associates, Inc., 2020), 17200–17211.

218. Parker-Holder, J., Pacchiano, A., Choromanski, K. & Roberts, S. *Effective Diversity in Population-Based Reinforcement Learning* in *Advances in Neural Information Processing Systems 33* (2020).

219. Parker-Holder, J., Rajan, R., Song, X., Biedenkapp, A., Miao, Y., Eimer, T., Zhang, B., Nguyen, V., Calandra, R., Faust, A., Hutter, F. & Lindauer, M. *Automated Reinforcement Learning (AutoRL): A Survey and Open Problems* in *JAIR* (2022).

220. Paul, S., Kurin, V. & Whiteson, S. Fast Efficient Hyperparameter Tuning for Policy Gradients. *Advances in Neural Information Processing Systems (NeurIPS)* (2019).

221. Pearlmutter, B. A. Fast exact multiplication by the Hessian. *Neural computation* **6,** 147–160 (1994).

222. Pelikan, M. & Goldberg, D. E. in *Scalable optimization via probabilistic modeling* 63–90 (Springer, 2006).

223. Pelikan, M., Goldberg, D. E. & Cantú-Paz, E. *BOA: The Bayesian Optimization Algorithm* in *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 1* (1999).

224. Pelikan, M., Goldberg, D. E. & Lobo, F. G. A survey of optimization by building and using probabilistic models. *Computational optimization and applications* **21,** 5–20 (2002).

225. Peng, M., Zhu, B. & Jiao, J. Linear Representation Meta-Reinforcement Learning for Instant Adaptation. *CoRR* (2021).

226. Peng, X. B., Andrychowicz, M., Zaremba, W. & Abbeel, P. *Sim-to-real transfer of robotic control with dynamics randomization* in *2018 IEEE international conference on robotics and automation (ICRA)* (2018), 3803–3810.

227. Perez-Liebana, D., Liu, J., Khalifa, A., Gaina, R. D., Togelius, J. & Lucas, S. M. General video game ai: A multitrack framework for evaluating agents, games, and content generation algorithms. *IEEE Transactions on Games* **11,** 195–214 (2019).

228. Peters, J. & Schaal, S. Reinforcement learning of motor skills with policy gradients. *Neural Networks* **21,** 682–697. ISSN: 0893-6080 (2008).

229. Pierrot, T., Macé, V., Chalumeau, F., Flajolet, A., Cideron, G., Beguir, K., Cully, A., Sigaud, O. & Perrin-Gilbert, N. *Diversity Policy Gradient for Sample Efficient Quality-Diversity Optimization* in *ICLR Workshop on Agent Learning in Open-Endedness* (2022).

230. Pinto, L., Davidson, J. & Gupta, A. *Supervision via competition: Robot adversaries for learning tasks* in *2017 IEEE International Conference on Robotics and Automation (ICRA)* (2017), 1601–1608.

231. Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R. Y., Chen, X., Asfour, T., Abbeel, P. & Andrychowicz, M. *Parameter Space Noise for Exploration* in *International Conference on Learning Representations* (2018).

232. Pong, V. H., Dalal, M., Lin, S., Nair, A., Bahl, S. & Levine, S. in *The International Conference on Machine Learning* (2020).

233. Portelas, R., Colas, C., Hofmann, K. & Oudeyer, P. *Teacher algorithms for curriculum learning of Deep RL in continuously parameterized environments* in *3rd Annual Conference on Robot Learning, CoRL 2019, Osaka, Japan, October 30 - November 1, 2019, Proceedings* (eds Kaelbling, L. P., Kragic, D. & Sugiura, K.) **100** (PMLR, 2019), 835–853.

234. Portelas, R., Colas, C., Weng, L., Hofmann, K. & Oudeyer, P.-Y. Automatic curriculum learning for deep rl: A short survey. *arXiv preprint arXiv:2003.04664* (2020).

235. Pugh, J. K., Soros, L. B. & Stanley, K. O. Quality Diversity: A New Frontier for Evolutionary Computation. *Frontiers in Robotics and AI* **3,** 40. ISSN: 2296-9144 (2016).

236. Puterman, M. L. Markov decision processes. *Handbooks in operations research and management science* **2,** 331–434 (1990).

237. Racaniere, S., Lampinen, A., Santoro, A., Reichert, D., Firoiu, V. & Lillicrap, T. *Automated curriculum generation through setter-solver interactions* in *International Conference on Learning Representations* (2020).

238. Rafailov, R., Yu, T., Rajeswaran, A. & Finn, C. *Offline Reinforcement Learning from Images with Latent Space Models* in *Proceedings of the 3rd Conference on Learning for Dynamics and Control* **144** (PMLR, 2021), 1154–1168.

239. Raileanu, R. & Fergus, R. *Decoupling Value and Policy for Generalization in Reinforcement Learning* in *Proceedings of the 38th International Conference on Machine Learning* (2021).

240. Raileanu, R., Goldstein, M., Yarats, D., Kostrikov, I. & Fergus, R. *Automatic Data Augmentation for Generalization in Deep Reinforcement Learning* in *Advances in Neural Information Processing Systems* (2021).

241. Raileanu, R. & Rocktäschel, T. *RIDE: Rewarding Impact-Driven Exploration for Procedurally-Generated Environments* in *International Conference on Learning Representations* (2020).

242. Rakelly, K., Zhou, A., Finn, C., Levine, S. & Quillen, D. *Efficient Off-Policy Meta-Reinforcement Learning via Probabilistic Context Variables* in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019* **97** (PMLR, 2019), 5331–5340.

243. Ramesh, A., Dhariwal, P., Nichol, A., Chu, C. & Chen, M. Hierarchical Text-Conditional Image Generation with CLIP Latents. *arXiv preprint arXiv:2204.06125* (2022).

244. Raparthy, S. C., Mehta, B., Golemo, F. & Paull, L. Generating Automatic Curricula via Self-Supervised Active Domain Randomization. *CoRR* **abs/2002.07911** (2020).

245. Rasmussen, C. E. & Williams, C. K. I. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)* ISBN: 026218253X (The MIT Press, 2005).

246. Real, E., Liang, C., So, D. R. & Le, Q. V. *AutoML-Zero: Evolving Machine Learning Algorithms From Scratch* in *Proceedings of the 37th International Conference on Machine Learning* (2020).

247. Riquelme, C., Penedones, H., Vincent, D., Maennel, H., Gelly, S., Mann, T. A., Barreto, A. & Neu, G. *Adaptive Temporal-Difference Learning for Policy Evaluation with Per-State Uncertainty Estimates* in *Advances in Neural Information Processing Systems* **32** (2019).

248. Risi, S. & Togelius, J. Increasing generality in machine learning through procedural content generation. *Nature Machine Intelligence* **2** (Aug. 2020).

249. Ru, B., Alvi, A. S., Nguyen, V., Osborne, M. A. & Roberts, S. J. *Bayesian optimisation over multiple continuous and categorical inputs* in *International Conference on Machine Learning (ICML)* (2020).

250. Ruiz, N., Schulter, S. & Chandraker, M. *Learning To Simulate* in *International Conference on Learning Representations* (2019).

251. Rumelhart, D. E., Hinton, G. E. & Williams, R. J. Learning representations by back-propagating errors. **323,** 533–536 (Oct. 1986).

252. Rumelhart, D. E., Hinton, G. E. & Williams, R. J. in *Neurocomputing: Foundations of Research* 696–699 (MIT Press, Cambridge, MA, USA, 1988). ISBN: 0262010976.

253. Runge, F., Stoll, D., Falkner, S. & Hutter, F. *Learning to Design RNA* in *7th International Conference on Learning Representations, ICLR, New Orleans, LA, USA, May 6-9* (OpenReview.net, 2019).

254. Sadeghi, F. & Levine, S. *CAD2RL: Real Single-Image Flight Without a Single Real Image* in *Robotics: Science and Systems XIII, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, July 12-16, 2017* (eds Amato, N. M., Srinivasa, S. S., Ayanian, N. & Kuindersma, S.) (2017).

255. Salimans, T., Ho, J., Chen, X., Sidor, S. & Sutskever, I. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. *CoRR* (2017).

256. Samvelyan, M., Kirk, R., Kurin, V., Parker-Holder, J., Jiang, M., Hambro, E., Petroni, F., Kuttler, H., Grefenstette, E. & Rocktäschel, T. *MiniHack the Planet: A Sandbox for Open-Ended Reinforcement Learning Research* in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track* (2021).

257. Sareni, B. & Krahenbuhl, L. Fitness sharing and niching methods revisited. *IEEE Transactions on Evolutionary Computation* **2,** 97–106 (1998).

258. Savage, L. J. The Theory of Statistical Decision. *Journal of the American Statistical association* (1951).

259. Schaul, T., Borsa, D., Ding, D., Szepesvari, D., Ostrovski, G., Dabney, W. & Osindero, S. Adapting Behaviour for Learning Progress. *CoRR* **abs/1912.06910** (2019).

260. Schaul, T., Quan, J., Antonoglou, I. & Silver, D. *Prioritized Experience Replay* in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings* (eds Bengio, Y. & LeCun, Y.) (2016).

261. Schmidhuber, J. Formal Theory of Creativity, Fun, and Intrinsic Motivation (1990–2010). *IEEE Transactions on Autonomous Mental Development* **2,** 230–247 (2010).

262. Schmidhuber, J. PowerPlay: Training an Increasingly General Problem Solver by Continually Searching for the Simplest Still Unsolvable Problem. *Frontiers in Psychology* **4,** 313. ISSN: 1664-1078 (2013).

263. Schmitt, S., Hudson, J. J., Zídek, A., Osindero, S., Doersch, C., Czarnecki, W. M., Leibo, J. Z., Küttler, H., Zisserman, A., Simonyan, K. & Eslami, S. M. A. Kickstarting Deep Reinforcement Learning. *CoRR* **abs/1803.03835** (2018).

264. Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T. P. & Silver, D. Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model. *Nature* (2020).

265. Schulman, J., Levine, S., Abbeel, P., Jordan, M. & Moritz, P. *Trust region policy optimization* in *International Conference on Machine Learning (ICML)* (2015).

266. Schulman, J., Moritz, P., Levine, S., Jordan, M. & Abbeel, P. *High-Dimensional Continuous Control Using Generalized Advantage Estimation* in *Proceedings of the International Conference on Learning Representations (ICLR)* (2016).

267. Schulman, J., Wolski, F., Dhariwal, P., Radford, A. & Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2016-2018).

268. Secretan, J., Beato, N., D Ambrosio, D. B., Rodriguez, A., Campbell, A. & Stanley, K. O. *Picbreeder: Evolving Pictures Collaboratively Online* in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Association for Computing Machinery, Florence, Italy, 2008), 1759–1768. ISBN: 9781605580111.

269. Sekar, R., Rybkin, O., Daniilidis, K., Abbeel, P., Hafner, D. & Pathak, D. *Planning to Explore via Self-Supervised World Models* in *ICML* (2020).

270. Shahriari, B., Swersky, K., Wang, Z., Adams, R. P. & De Freitas, N. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE* **104,** 148–175 (2015).

271. Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T. P., Leach, M., Kavukcuoglu, K., Graepel, T. & Hassabis, D. Mastering the game of Go with deep neural networks and tree search. *Nature* **529,** 484–489 (2016).

272. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T. P., Simonyan, K. & Hassabis, D. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *Science* (2017).

273. Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T. P., Hui, F., Sifre, L., van den Driessche, G., Graepel, T. & Hassabis, D. Mastering the game of Go without human knowledge. *Nat.* **550,** 354–359 (2017).

274. Silver, D., Singh, S., Precup, D. & Sutton, R. S. Reward is enough. *Artificial Intelligence* **299,** 103535. ISSN: 0004-3702 (2021).

275. Song, X., Du, Y. & Jackson, J. An Empirical Study on Hyperparameters and their Interdependence for RL Generalization. *ICML 2019 Workshop "Understanding and Improving Generalization in Deep Learning"* (2019).

276. Song, X., Jiang, Y., Tu, S., Du, Y. & Neyshabur, B. *Observational Overfitting in Reinforcement Learning* in *8th International Conference on Learning Representations, ICLR, Addis Ababa, Ethiopia, April 26-30* (OpenReview.net, 2020).

277. Soros, L. & Stanley, K. Identifying Necessary Conditions for Open-Ended Evolution through the Artificial Life World of Chromaria. *Artificial Life Conference Proceedings,* 793–800 (2014).

278. Srinivas, N., Krause, A., Kakade, S. & Seeger, M. *Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design* in *Proceedings of the 27th International Conference on International Conference on Machine Learning* (2010).

279. Stanley, K., Clune, J., Lehman, J. & Miikkulainen, R. Designing neural networks through neuroevolution. *Nature Machine Intelligence* **1** (Jan. 2019).

280. Stanley, K. O. & Lehman, J. *Why Greatness Cannot Be Planned: The Myth of the Objective* ISBN: 3319155237 (Springer Publishing Company, Incorporated, 2015).

281. Stanley, K. O., Lehman, J. & Soros, L. Open-endedness: The last grand challenge you've never heard of. *O'Reilly* (2017).

282. Strathern, M. 'Improving ratings': audit in the British University system. *European review* **5,** 305–321 (1997).

283. Strouse, D., McKee, K., Botvinick, M., Hughes, E. & Everett, R. *Collaborating with Humans without Human Data* in *Advances in Neural Information Processing Systems* (eds Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P. & Vaughan, J. W.) **34** (Curran Associates, Inc., 2021), 14502–14515.

284. Sturtevant, N., Decroocq, N., Tripodi, A. & Guzdial, M. *The unexpected consequence of incremental design changes* in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* **16** (2020), 130–136.

285. Sukhbaatar, S., Lin, Z., Kostrikov, I., Synnaeve, G., Szlam, A. & Fergus, R. *Intrinsic Motivation and Automatic Curricula via Asymmetric Self-Play* in *International Conference on Learning Representations* (2018).

286. Summerville, A., Snodgrass, S., Guzdial, M., Holmgård, C., Hoover, A. K., Isaksen, A., Nealen, A. & Togelius, J. Procedural Content Generation via Machine Learning (PCGML). *IEEE Trans. Games* **10,** 257–270 (2018).

287. Sutskever, I., Vinyals, O. & Le, Q. V. *Sequence to Sequence Learning with Neural Networks* in *Advances in Neural Information Processing Systems* (eds Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. & Weinberger, K.) **27** (Curran Associates, Inc., 2014).

288. Sutton, R. & Singh, S. P. *On Step-Size and Bias in Temporal-Difference Learning* in *Center for Systems Science, Yale University* (1994), 91–96.

289. Sutton, R. S. Dyna, an Integrated Architecture for Learning, Planning, and Reacting. *SIGART Bull.* **2,** 160–163. ISSN: 0163-5719 (July 1991).

290. Sutton, R. S. & Barto, A. G. *Introduction to Reinforcement Learning* 1st. ISBN: 0262193981 (MIT Press, Cambridge, MA, USA, 1998).

291. Sutton, R. S., McAllester, D., Singh, S. & Mansour, Y. *Policy Gradient Methods for Reinforcement Learning with Function Approximation* in *Advances in Neural Information Processing Systems* (1999).

292. Swersky, K., Snoek, J. & Adams, R. P. *Freeze-Thaw Bayesian Optimization* 2014.

293. Tam, A. C., Rabinowitz, N. C., Lampinen, A. K., Roy, N. A., Chan, S. C. Y., Strouse, D., Wang, J. X., Banino, A. & Hill, F. *Semantic Exploration from Language Abstractions and Pretrained Representations* 2022.

294. Tang, Y., Nguyen, D. & Ha, D. *Neuroevolution of Self-Interpretable Agents* in *Proceedings of the Genetic and Evolutionary Computation Conference* (2020).

295. Taylor, T., Bedau, M., Channon, A., Ackley, D., Banzhaf, W., Beslon, G., Dolson, E., Froese, T., Hickinbotham, S., Ikegami, T., McMullin, B., Packard, N., Rasmussen, S., Virgo, N., Agmon, E., Clark, E., McGregor, S., Ofria, C., Ropella, G., Spector, L., Stanley, K. O., Stanton, A., Timperley, C., Vostinar, A. & Wiser, M. Open-Ended Evolution: Perspectives from the OEE Workshop in York. *Artificial Life* **22,** 408–423. ISSN: 1064-5462 (Aug. 2016).

296. Team, O. E. L., Stooke, A., Mahajan, A., Barros, C., Deck, C., Bauer, J., Sygnowski, J., Trebacz, M., Jaderberg, M., Mathieu, M., McAleese, N., Bradley-Schmieg, N., Wong, N., Porcel, N., Raileanu, R., Hughes-Fitt, S., Dalibard, V. & Czarnecki, W. M. Open-Ended Learning Leads to Generally Capable Agents. *CoRR* **abs/2107.12808** (2021).

297. Tjanaka, B., Fontaine, M. C., Togelius, J. & Nikolaidis, S. *Differentiable Quality Diversity for Reinforcement Learning by Approximating Gradients* in *ICLR Workshop on Agent Learning in Open-Endedness* (2022).

298. Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W. & Abbeel, P. *Domain randomization for transferring deep neural networks from simulation to the real world* in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2017, Vancouver, BC, Canada, September 24-28, 2017* (IEEE, 2017), 23–30.

299. Todorov, E., Erez, T. & Tassa, Y. *MuJoCo: A physics engine for model-based control* in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2012), 5026–5033.

300. Toffolo, A. & Benini, E. Genetic Diversity as an Objective in Multi-Objective Evolutionary Algorithms. *Evol. Comput.* **11,** 151–167. ISSN: 1063-6560 (2003).

301. Togelius, J. & Schmidhuber, J. *An experiment in automatic game design* in *2008 IEEE Symposium On Computational Intelligence and Games* (2008), 111–118.

302. Tolani, V., Bansal, S., Faust, A. & Tomlin, C. J. Visual Navigation Among Humans With Optimal Control as a Supervisor. *IEEE Robotics Autom. Lett.* **6,** 2288–2295 (2021).

303. Trujillo, L., Olague, G., Lutton, E. & de Vega, F. F. *Behavior-Based Speciation for Evolutionary Robotics* in *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation* (Association for Computing Machinery, Atlanta, GA, USA, 2008), 297–298. ISBN: 9781605581309.

304. Trujillo, L., Olague, G., Lutton, E., Fernández de Vega Francisco", e. M., Brabazon, A., Cagnoni, S., Di Caro, G. A., Drechsler, R., Ekárt, A., Esparcia-Alcázar, A. I., Farooq, M., Fink, A., McCormack, J., O'Neill, M., Romero, J., Rothlauf, F., Squillero, G., Uyar, A. Ş. & Yang, S. *Discovering Several Robot Behaviors through Speciation* in *Applications of Evolutionary Computing* (2008).

305. Tunyasuvunakool, S., Muldal, A., Doron, Y., Liu, S., Bohez, S., Merel, J., Erez, T., Lillicrap, T., Heess, N. & Tassa, Y. dm control: Software and tasks for continuous control. *Software Impacts* **6,** 100022. ISSN: 2665-9638 (2020).

306. Uchiya, T., Nakamura, A. & Kudo, M. *Algorithms for adversarial bandit problems with multiple plays* in *International Conference on Algorithmic Learning Theory* (2010), 375–389.

307. Van Hasselt, H., Guez, A. & Silver, D. *Deep Reinforcement Learning with Double Q-Learning* in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA* (eds Schuurmans, D. & Wellman, M. P.) (AAAI Press, 2016), 2094–2100.

308. Van Hasselt, H., Hessel, M. & Aslanides, J. *When to use parametric models in reinforcement learning?* in *Advances in Neural Information Processing Systems* (2019), 14322–14333.

309. Vapnik, V. *Principles of Risk Minimization for Learning Theory* in *Advances in Neural Information Processing Systems 4* (eds Moody, J. E., Hanson, S. J. & Lippmann, R. P.) (Morgan-Kaufmann, 1992), 831–838.

310. Veeriah, V., Hessel, M., Xu, Z., Rajendran, J., Lewis, R. L., Oh, J., van Hasselt, H. P., Silver, D. & Singh, S. *Discovery of Useful Questions as Auxiliary Tasks* in *Advances in Neural Information Processing Systems* **32** (2019).

311. Veeriah, V., Zahavy, T., Hessel, M., Xu, Z., Oh, J., Kemaev, I., van Hasselt, H., Silver, D. & Singh, S. Discovery of Options via Meta-Learned Subgoals. *CoRR* **abs/2102.06741** (2021).

312. Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., Vezhnevets, A. S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T. L., Gülçehre, Ç., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T. P., Kavukcuoglu, K., Hassabis, D., Apps, C. & Silver, D. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nat.* **575,** 350–354 (2019).

313. Wainwright, M. Basic tail and concentration bounds. *URl: https://www. stat. berkeley. edu/.../Chap2_TailBounds_Jan22_2015. pdf (visited on 12/31/2017)* (2015).

314. Wan, X., Lu, C., Parker-Holder, J., Ball, P. J., Nguyen, V., Ru, B. & Osborne, M. *Bayesian Generational Population-Based Training* in *AutoML Conference* (2022).

315. Wang, R., Lehman, J., Clune, J. & Stanley, K. O. Paired Open-Ended Trailblazer (POET): Endlessly Generating Increasingly Complex and Diverse Learning Environments and Their Solutions. *CoRR* **abs/1901.01753** (2019).

316. Wang, R., Lehman, J., Rawal, A., Zhi, J., Li, Y., Clune, J. & Stanley, K. *Enhanced POET: Open-ended Reinforcement Learning through Unbounded Invention of Learning Challenges and their Solutions* in *Proceedings of the 37th International Conference on Machine Learning* (eds III, H. D. & Singh, A.) **119** (PMLR, 2020), 9940–9951.

317. Wang, T., Bao, X., Clavera, I., Hoang, J., Wen, Y., Langlois, E., Zhang, S., Zhang, G., Abbeel, P. & Ba, J. Benchmarking Model-Based Reinforcement Learning. *CoRR* **abs/1907.02057** (2019).

318. Watkins, C. J. & Dayan, P. Q-learning. *Machine learning* **8,** 279–292 (1992).

319. Welch, B. L. The Generalization of 'Student's' Problem when Several Different Population Variances are Involved. *Biometrika* **34,** 28–35. ISSN: 00063444. (2022) (1947).

320. White, M. & White, A. *A greedy approach to adapting the trace parameter for temporal difference learning* in *AAMAS* (2016).

321. Whiteson, S., Tanner, B., Taylor, M. E. & Stone, P. *Generalized domains for empirical evaluations in reinforcement learning* 2009.

322. Whitley, D., Gordon, V. S. & Mathias, K. *Lamarckian evolution, the Baldwin effect and function optimization* in *Parallel Problem Solving from Nature — PPSN III* (eds Davidor, Y., Schwefel, H.-P. & Männer, R.) (1994).

323. Wierstra, D., Schaul, T., Glasmachers, T., Sun, Y., Peters, J. & Schmidhuber, J. Natural Evolution Strategies. *Journal of Machine Learning Research* **15,** 949–980 (2014).

324. Williams, R. J. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Mach. Learn.* **8,** 229–256. ISSN: 0885-6125 (1992).

325. Woolley, B. G. & Stanley, K. O. *On the Deleterious Effects of a Priori Objectives on Evolution and Representation* in *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation* (Association for Computing Machinery, Dublin, Ireland, 2011), 957–964. ISBN: 9781450305570.

326. Xu, Y., Parker-Holder, J., Pacchiano, A., Ball, P. J., Rybkin, O., Roberts, S. J., Rocktäschel, T. & Grefenstette, E. *Learning General World Models in a Handful of Reward-Free Deployments* in *Advances in Neural Information Processing Systems* (2022).

327. Xu, Z., van Hasselt, H. & Silver, D. *Meta-Gradient Reinforcement Learning* in *Advances in Neural Information Processing Systems 31: NeurIPS, December 3-8, Montréal, Canada* (2018), 2402–2413.

328. Xu, Z., van Hasselt, H. P., Hessel, M., Oh, J., Singh, S. & Silver, D. *Meta-Gradient Reinforcement Learning with an Objective Discovered Online* in *Advances in Neural Information Processing Systems* **33** (2020), 15254–15264.

329. Yang, Y., Luo, J., Wen, Y., Slumbers, O., Graves, D., Bou-Ammar, H., Wang, J. & Taylor, M. E. *Diverse Auto-Curriculum is Critical for Successful Real-World Multiagent Learning Systems* in *AAMAS '21: 20th International Conference on Autonomous Agents and Multiagent Systems, Virtual Event, United Kingdom, May 3-7, 2021* (eds Dignum, F., Lomuscio, A., Endriss, U. & Nowé, A.) (ACM, 2021), 51–56.

330. Yarats, D., Brandfonbrener, D., Liu, H., Laskin, M., Abbeel, P., Lazaric, A. & Pinto, L. Don't Change the Algorithm, Change the Data: Exploratory Data for Offline Reinforcement Learning. *CoRR* **abs/2201.13425** (2022).

331. Yarats, D., Fergus, R., Lazaric, A. & Pinto, L. *Mastering Visual Continuous Control: Improved Data-Augmented Reinforcement Learning* in *International Conference on Learning Representations* (2022).

332. Yu, T., Thomas, G., Yu, L., Ermon, S., Zou, J., Levine, S., Finn, C. & Ma, T. *MOPO: Model-based Offline Policy Optimization* in *Advances in Neural Information Processing Systems* (2020).

333. Zahavy, T., Barreto, A., Mankowitz, D. J., Hou, S., O'Donoghue, B., Kemaev, I. & Singh, S. *Discovering a set of policies for the worst case reward* in *International Conference on Learning Representations* (2021).

334. Zahavy, T., O'Donoghue, B., Barreto, A., Mnih, V., Flennerhag, S. & Singh, S. Discovering Diverse Nearly Optimal Policies with Successor Features. *arXiv preprint arXiv:2106.00669* (2021).

335. Zahavy, T., Xu, Z., Veeriah, V., Hessel, M., Oh, J., van Hasselt, H., Silver, D. & Singh, S. *A Self-Tuning Actor-Critic Algorithm* in *Advances in Neural Information Processing Systems* (2020).

336. Zech, J. R., Badgeley, M. A., Liu, M., Costa, A. B., Titano, J. J. & Oermann, E. K. Variable generalization performance of a deep learning model to detect pneumonia in chest radiographs: a cross-sectional study. *PLoS medicine* **15,** e1002683 (2018).

337. Zhang, A., Ballas, N. & Pineau, J. A Dissection of Overfitting and Generalization in Continuous Reinforcement Learning. *CoRR* **abs/1806.07937** (2018).

338. Zhang, B., Rajan, R., Pineda, L., Lambert, N., Biedenkapp, A., Chua, K., Hutter, F. & Calandra, R. *On the Importance of Hyperparameter Optimization for Model-based Reinforcement Learning* in *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics* (2021).

339. Zhang, C., Vinyals, O., Munos, R. & Bengio, S. A Study on Overfitting in Deep Reinforcement Learning. *CoRR* **abs/1804.06893** (2018).

340. Zhang, H., Chen, Y. & Parkes, D. *A General Approach to Environment Design with One Agent* in *Proceedings of the 21st International Jont Conference on Artifical Intelligence* (Pasadena, California, USA, 2009), 2002–2008.

341. Zhang, H. & Parkes, D. *Value-Based Policy Teaching with Active Indirect Elicitation* in *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 1* (AAAI Press, 2008), 208–214. ISBN: 9781577353683.

342. Zhang, T., Xu, H., Wang, X., Wu, Y., Keutzer, K., Gonzalez, J. E. & Tian, Y. *NovelD: A Simple yet Effective Exploration Criterion* in *Advances in Neural Information Processing Systems* (eds Beygelzimer, A., Dauphin, Y., Liang, P. & Vaughan, J. W.) (2021).

343. Zhang, Y., Abbeel, P. & Pinto, L. *Automatic Curriculum Learning through Value Disagreement* in *Advances in Neural Information Processing Systems* (eds Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F. & Lin, H.) **33** (Curran Associates, Inc., 2020), 7648–7659.

344. Zhou, Z., Fu, W., Zhang, B. & Wu, Y. *Continuously Discovering Novel Strategies via Reward-Switching Policy Optimization* in *International Conference on Learning Representations* (2022).

345. Zintgraf, L., Shiarlis, K., Igl, M., Schulze, S., Gal, Y., Hofmann, K. & Whiteson, S. *VariBAD: A Very Good Method for Bayes-Adaptive Deep RL via Meta-Learning* in *International Conference on Learning Representations* (2020).

346. Zintgraf, L. M., Feng, L., Lu, C., Igl, M., Hartikainen, K., Hofmann, K. & Whiteson, S. *Exploration in Approximate Hyper-State Space for Meta Reinforcement Learning* in *Proceedings of the 38th International Conference on Machine Learning* **139** (PMLR, 2021), 12991–13001.