Towards Characterizing Curriculum Reinforcement Learning in Sparse Robotics Tasks

A THESIS PRESENTED
BY
THOMAS KAMINSKY
TO
THE DEPARTMENT OF STATISTICS

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
BACHELOR OF ARTS (HONORS)
IN THE SUBJECT OF
STATISTICS AND MATHEMATICS

HARVARD UNIVERSITY
CAMBRIDGE, MASSACHUSETTS
APRIL 2024

Thesis advisor: Gabriel Kreiman Thomas Kaminsky

Towards Characterizing Curriculum Reinforcement Learning in Sparse Robotics Tasks

Abstract

Nearly every living human has benefited from curriculum training. Curricula, which enable learners to acquire complex skills by training them on a series of simpler subtasks that slowly increase in difficulty, are ubiquitous in both education and psychology, where such methods are generally referred to as shaping. In recent years, many attempts have been made to use ideas from curriculum learning within the realm of machine learning to train better models in a more sample-efficient manner.

We show that, in complex multi-task robotic reinforcement learning problems where rewards are sparse and the probability of successfully executing a task at random is negligible, some form of curriculum training is necessary to achieve strong performance in a sample-efficient manner. Moreover, we demonstrate that successful curricula provide two distinct services:

Most importantly, they must serve as a form of *Signal Engineering*, granting the agent a meaningful reward signal early in training which serves as a substitute for a dense reward. Second, to continue to learn later in training, curricula must provide a source of *guided exploration*, presenting the agent with regions in task space which are difficult but learnable and reinforcing existing skills to avoid forgetting.

Through experiments, we show that failing to provide either of these services results in a less effective curriculum, with inadequate signal engineering resulting in no meaningful learning in the more complex setting. These results encourage a more principled development of sparse-reward curricula which explicitly solve both subproblems of curriculum learning, allowing ever more general agents to solve pressing robotics problems.

Contents

Listing of figures

Acknowledgments

An enormous thank you to my advisor, Gabriel Kreiman, for taking me on and helping me develop the ideas for this proejct; to Joshua Smith and Karthik Desingh for your incredible support and mentorship over the years; to Boling Yang for introducing me to curriculum learning and bearing with me as I blundered through code and AWS documentation (and still do); to Markus Grotz for your support and encouragement, and for helping me get swipe access to the espresso room; to Abhishek Gupta, Henri Fung, Soofiyan Atar, Chahyon Ku, Nirmal Raj, Bahaa Aldeeb, and all those at UW and UMN whom I worked with and learned from.

Thank you to all of my teachers and mentors here at Harvard, both in the statistics department and beyond; to my friends, for providing, at times, both a support system and a diversion; to Andrew Rossi, for patiently listening as I ranted about this project more times than I can count; and, of course, to my family, my parents Jane and Jake, and my brother and sister, Eddie and Lily, for being infinitely supportive and understanding—I wouldn't be here without you.

This willingness continually to revise one's own location in order to place oneself in the path of beauty is the basic impulse underlying education. One submits oneself to other minds (teachers) in order to increase the chance that one will be looking in the right direction when a comet makes its sweep through a certain patch of sky.

Elaine Scarry

1 Introduction

If you are reading this, you have almost certainly been trained using a curriculum.

In the United States, every child enrolled Public School is tasked with developing English fluency through an ordered set of standards. By one model [29], students in Kindergarten learn to describe objects by color or texture, and to predict the meanings of words through context in conversation. The next year, they must extend these abilities to simple texts, and to begin the process of learning abstract reasoning by sorting objects into ephemeral categories like 'living things' or 'birds'. By the time they graduate high school, they should hopefully posses a complex understanding of the language, enabling them to study specialized fields, intuit new words' meanings, understand idioms and references to significant works, and to communicate complex ideas to others.

In other words, at each stage of their development students are given subtasks, related to but distinct from the end goal of developing general competence in the English language. This set of ordered tasks, which we call a *curriculum*, has enabled modern humans to quickly learn a variety of subjects, resulting for most in better results than if self-taught.

It is unsurprising, then, that those interested in teaching machines have also considered whether ideas from curriculum learning can improve learning speed and performance in their own domains. In this thesis, we hope to describe in some detail the ways which this structured view of skill acquisition has been applied to machine learning domains, focusing in particular on methods unique to reinforcement learning. At the core of this paper are questions relevant to modern robotics research:

- What causes curriculum methods to succeed or fail?
- How does reward sparsity impact the learning process?
- How can we gain an intuitive sense of whether a curriculum method will perform well on relevant robotics tasks?

We focus on sparse-reward robotics tasks, which are entering increasing prominence in modern robotics research due to being incredibly general and simple to define, even for complex systems. Our results show that, in this setting, successful curricula can be seen to provide two services: *signal engineering* and *guided exploration.* We show that, when either of these components is missing from an autocurriculum method, performance on the goal task suffers, and these problems compound when the complexity of the environment increases.

Thank you for reading.

2 Background

In the following chapter, we provide background on reinforcement learning in the single- and multi-task settings. We also describe the trajectory of curriculum learning research, beginning with supervised learning domains and proceeding to the sparse-reward multi-task reinforcement learning setting, which will encompass our experiments.

Note that throughout this paper, we refer to the set of distributions over a (possibly infinite) set $\mathcal{S}$ as $\Delta\mathcal{S}$. In the discrete case where $|\mathcal{S}| = n \in \mathbb{N}$, this is simply the set of categorical distributions

$$[p_1, p_2, \ldots, p_n] \text{ s.t. } \forall i \in \{1, \ldots, n\} p_i \geq 0, \sum_{i=1}^{n} p_i = 1.$$

When sampling from a distribution with probability density f, we overload notation by letting $f(x)$ denote the density of x and $x \sim f$

denote a variable sampled from a distribution with PDF f.

2.1 Markov Decision Processes and Reinforcement Learning

Central to this work is the *Markov Decision Process (MDP)*, a model for describing general tasks that require an agent to repeatedly recieve signals from the environment and interact with the world via actions.

Formally, we define an MDP as follows:

Definition 2.1.1 (Markov Decision Process)**.** A Markov Decision Process is a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \rho, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ are sets called the *state space* and *action space*, $P(s, a) : \mathcal{S} \times \mathcal{A} \to \Delta\mathcal{S}$ and $r(s, a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ are called the *transition* and *reward* functions, respectively, $\rho \in \Delta\mathcal{S}$ defines an *initial state distribution*, and $\gamma \in (0, 1)$ is called the *discount factor*.

Intuitively, $\mathcal{S}$ and $\mathcal{A}$ describe the space of all states and actions available to the agent, $P(\cdot|s, a)$ describes the transition dynamics upon taking action a in state s, $r(s, a)$ assigns a reward for taking action a in state s, ρ is a distribution describing where the agent begins in the environemnt, and γ is a discount factor weighting how heavily future rewards should be considered in decision-making (see figure 2.1.1).

One can consider this tuple as specifying the *environment* of the MDP—where and how agent may move, and what dynamics govern how the state changes when actions are taken—but it does not describe the behavior of the agent itself. To do so, we must also introduce the idea of a *policy*:

Definition 2.1.2 (Policy)**.** A *policy* π is a function $\pi : \mathcal{S} \to \Delta\mathcal{A}$. Intuitively, π encodes the actions of an agent operating in an MDP.

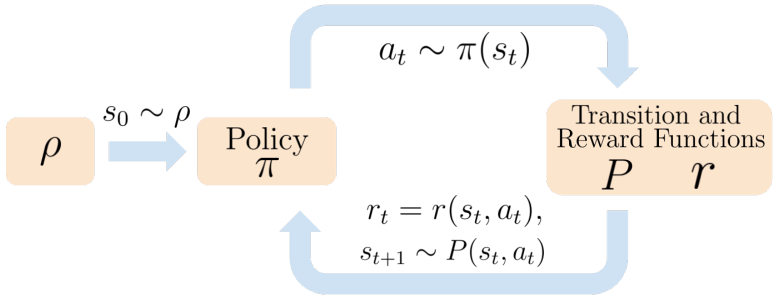

Figure 2.1.1: A graph describing the loop for collecting a trajectory in an MDP. After an initial state is sampled from ρ, the policy predicts for the current state an action a_t, which is used to compute rewards and the subsequent state. This process repeats indefinitely, or until a terminal state is reached.

When at state s, $\pi(s)$ describes the chance of the agent taking each available action. Having now an environment and an agent to act in it, we can begin to collect experience samples from the MDP:

Definition 2.1.3 (Trajectory)**.** For any MDP $m = (\mathcal{S}, \mathcal{A}, P, r, \rho, \gamma)$ and policy π, we can collect a *trajectory*

$$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \dots)$$

by the following scheme:

1. Sample an initial state $s_0 \sim \rho$.
2. For $t = 0, 1, 2, \dots$:
   (a) Sample an action from the policy $a_t \sim \pi(s_t)$.
   (b) Collect a reward $r_t = r(s_t, a_t)$.
   (c) Transition to the next state by sampling from the transition distribution $s_{t+1} \sim P(s_t, a_t)$.

As an abuse of notation, we will often write $\tau \sim (\pi, \mathcal{M})$, or simply

$\tau \sim \pi$ when the MDP is clear from context, to describe collecting a trajectory sample by this process.

We can also describe the *value* of a state under a policy as follows:

Definition 2.1.4 (Value)**.** The *value* of a state s under policy π is the expected total discounted reward collected by a policy beginning in that state. Formally:

$$V^{\pi}(s) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t, a_t)\middle| s_0 = s\right].$$

This definition captures two important notions of value. First, the quality of a state depends, not only on how good the state is in isolation, but also on how it may lead to more rewards in the future. Second, future states are discounted by a γ^t term, causing rewards far in the future to impact the value less than rewards in the present. How much these rewards are discounted depends on γ, with choices close to 1 causing long-term rewards to be more impactful, and with choices close to 0 yielding a myopic value assignment.

We can also define a related notion of the state-value, encoded by the Q function:

Definition 2.1.5 (Q-Function)**.** The *state-value* $Q^{\pi}(s, a)$ of a state-action pair (s, a) under policy π is the expected total discounted reward collected by a policy beginning in state s and taking action a. Formally:

$$Q^{\pi}(s, a) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t, a_t)\middle| s_0 = s, a_0 = a\right].$$

The Q-function is connected to the Value function by the relationship

$$V^{\pi}(s) = \mathbb{E}_{a\sim\pi}\left[Q^{\pi}(s, a)\right].$$

That is, the value of a state is equal to the weighted average of the state-action value over all possible actions, weighted by the probability of policy π taking each action.

With the notion of value defined, we can now state the complete reinforcement learning objective:

Definition 2.1.6 (Reinforcement Learning (RL) Objective)**.** The goal of Reinforcement Learning is to find a policy which maximizes the expected value when placed in states sampled from the initial distribution ρ.

That is, given some *policy space* Π, we seek

$$\operatorname*{argmax}_{\pi \in \Pi} \mathbb{E}_{s_0 \sim \rho} \left[V^{\pi}(s_0) \right].$$

2.2 Multi-Task Reinforcement Learning

We can generalize the above problem to describe settings with multiple tasks.

Definition 2.2.1 (Multi-Task Markov Decision Process (MTMDP))**.** A Multi-Task MDP is a tuple $(\mathcal{T}, g)$, where $\mathcal{T}$ is a set of MDPs called the *task space* and $g \in \Delta\mathcal{T}$ specifies a distribution over the task space, called the *ground truth distribution*.

In this setting, a policy is now a function $\texttt{POLICY} : \mathcal{T} \longrightarrow \Pi_{\mathcal{T}}$, where $\Pi_{\mathcal{T}}$ describes the union of all policy spaces for each task in $\mathcal{T}$. For any task $m \in \mathcal{T}$, $\texttt{POLICY}(m) = \pi_m$ provides a policy for the corresponding MDP.

It is worth noting that, so long as the discount factor γ is constant across every task in $\mathcal{T}$, a Multi-Task MDP $(\mathcal{T}, g)$ can be reduced to a single MDP by defining

- $\mathcal{S} = \bigcup_{m \in \mathcal{T}} \mathcal{S}_m$,
- $\mathcal{A} = \bigcup_{m \in \mathcal{T}} \mathcal{A}_m$,
- $P(s, a) = P_m(s, a)$ for all $s, a \in m$, $m \in \mathcal{T}$, and 0 otherwise.
- $r(s, a) = r_m(s, a)$ for all $s, a \in m$, $m \in \mathcal{T}$, and 0 otherwise.
- $\rho(s) = g(m)\rho_m(s)$ for all $s \in \mathcal{S}_m$, $m \in \mathcal{T}$, and 0 otherwise.

Even if the discount factors are unequal, the problem still nearly reduces to the standard RL problem, though the value function must instead be defined with task-dependent discounting. In general, we have the following objective:

Definition 2.2.2 (Multi-Task Reinforcement Learning (MTRL) Objective)**.** Analogous to the single-task setting, Multi-Task RL seeks to find a policy which maximizes the value of trajectories collected by the policy, with task importance weighted by the ground truth distribution. That is, given a policy space $\mathcal{P} \subseteq \{f : \mathcal{T} \to \Pi_{\mathcal{T}}\}$ we seek

$$\underset{\texttt{POLICY} \in \mathcal{P}}{\operatorname{argmax}} \, \mathbb{E}_{m \sim g, s_0 \sim \rho_m} \left[V_m^{\texttt{POLICY(m)}}(s_0) \right].$$

Some papers which focus on exploration or zero-shot generalization rather than performance on a single goal task may forgo g entirely. In these cases, some other metric is used to analyze how expressive or generalizable the policy is, rather than the metric given above. However, for the purposes of our problem, we take performance to ultimately depend on some ground-truth distribution of tasks at test time.

Having described the relevant formalism for our problem, we now proceed with a review of curriculum learning literature in the supervised and reinforcement learning settings.

2.3 Curriculum Learning in Supervised Domains

Most early work on curriculum learning sought to create analogues for human curricula in supervised learning domains. That is, in settings where one is given a sample of training data drawn from some ground truth distribution of features $\{x_i\}_{i \in I} \sim \mathcal{D}_{GT}$ and regression targets $\{y_i\}_{i \in I} \sim f(y|x_i)_\epsilon$, a curriculum seeks to *order* these data so that the agent can learn a function $\hat{f}(x_i) \approx y_i$—for example, by training a deep neural network—more efficiently than if trained on the full dataset in random order.

Bengio et al. (2009) [6] first describe the standard formulation of the curriculum learning paradigm, noting two potential benefits:

1. A curriculum can act as a *continuation method*, or a strategy for minimizing non-convex criteria by iteratively optimizing smoother objectives which slowly approach the full objective.

2. Curricula can serve as a regularizers, smoothing out bad local minima and thus yielding a better final optimum.

Wang et al. summarize these two motivations in curriculum learning as *guiding* and *denoising* [33]. A curriculum *guides* in that it proposes surrogate objectives which hopefully have optima in parameter space that are close to the optimal parameters for the full task, and it *denoises* by making these surrogate objectives simpler than the final task, either by smoothing the function, removing ambiguous samples, or so on (see figure 2.3.1).

Still, though curriculum methods are intuitively appealing and have seen success in data-rich domains such as noisy computer vision tasks (e.g. [12]) and complex language tasks like machine translation ([22]), other work casts doubt on their efficacy as a general tool.

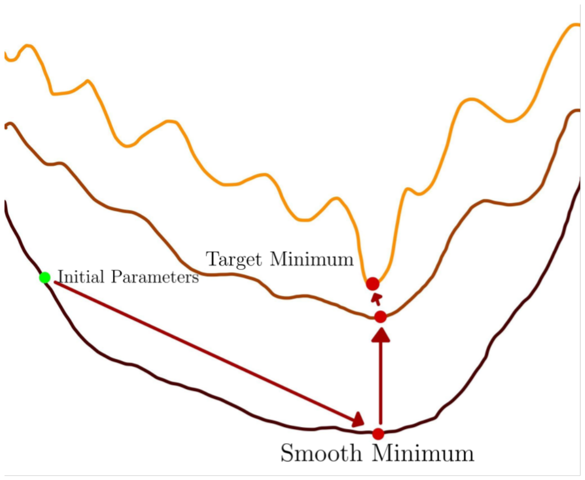

Figure 2.3.1: A visualization of curriculum guiding and denoising, copied with minor alterations from [33]. Data subsets early in the curriculum (bottom) have smooth objectives which move the parameters to good regions of parameter space. Later datasets (middle) guide the model's parameters to regions closer to the global optimum of the final task (top).

Wu et al. (2021) [34], for instance, untertook a large-scale evaluation of supervised curriculum methods on standard image classification tasks (CIFAR10, CIFAR100, FOOD101, FOOD101N), finding that curricula produced mixed results.

In settings with high label noise or limited compute budgets, curriculum methods did indeed dramatically outperform standard training. However, in the 'standard setting' where training is not limited by compute or annotation quality, curriculum learning resulted in no performance benefits relative to a random ordering, or even to so-called 'anti-curriculum' learning, where data is presented in the reverse of the order proposed by the curriculum.

Thus, for many supervised learning tasks, curriculum learning does not seem to yield a significant benefit to final performance, keeping it from becoming a ubiquitous module added to supervised learning tasks. When it is used in general settings, it is most likely to help by increasing sample efficiency, such as during active learning [28].

2.4 Curriculum Reinforcement Learning

In the reinforcement learning setting, results have been more dramatic. However, its potential usefulness has been qualified by the fact that, compared to the supervised setting, a much more diverse set of algorithms can all be seen as training agents via curricula. Thus, it is difficult to develop a useful theory for analyzing curricula in general.

Paraphrasing a survey of curriculum learning by Narvekar et al. (2021) [19], we define curriculum learning in the MTRL setting as follows:

Definition 2.4.1 (Multi-Task RL Curriculum)**.** Let $(\mathcal{T}, g)$ be an MTMDP, $\mathcal{D}^{\mathcal{T}}$ the set of all transition samples (s, a, r, s') from tasks in $\mathcal{T}$, and $\mathcal{P}(\mathcal{D}^{\mathcal{T}})$ the power set of of $\mathcal{D}^{\mathcal{T}}$. A *curriculum* $C = (\mathcal{V}, \mathcal{E}, f, \mathcal{T})$ is a directed acyclic graph, with $\mathcal{V}$ being the set of vertices, $\mathcal{E} \subseteq \{(x, y) \in \mathcal{V}^2 \text{ s.t. } x \neq y\}$ a set of directed edges, and $f : \mathcal{V} \to \mathcal{P}(\mathcal{D}^{\mathcal{T}})$ a function mapping vertices to subsets of transition samples in the task space. There must also exist a single sink node $v_s \in \mathcal{V}$ corresponding to the goal task.

This dense formalism has a reasonable visual interpretation—each vertex describes a set of samples on which to train a policy. These samples could be specific sets of transition samples, entire trajectories, or even groups of samples from different tasks. We can train an agent using such a curriculum by alternating between walking to a vertex, training using some algorithm on the corresponding experience samples, and then moving along a directed edge to another vertex until reaching the final node corresponding to the goal task.

Still, this definition encompasses a range of possible algorithms. We describe a few common categories below:

2.4.1 Experience Augmentation

Some of the lowest-intervention curriculum strategies seek to adjust an agent's learning by adjusting its *experience* of the world, without changing anything in the environment itself. In these settings, an agent often contains a memory B (called a replay buffer), and optimization is performed on batches sampled from this buffer.

In standard RL, this sample is chosen randomly. Experience Augmentation methods instead impact the batch choice by, for example:

- Weighting some samples over others.
- Altering the rewards from some experience samples.
- Duplicating or removing experience samples.

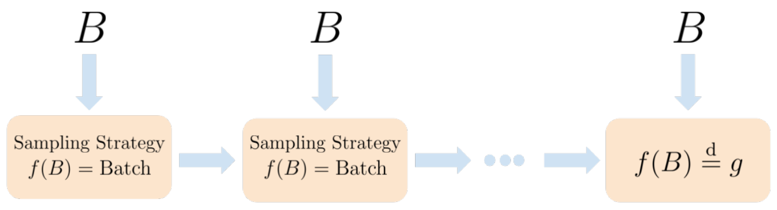

Figure 2.4.1: A graph representation of experience augmentation.

Approaches like Prioritized Experience Replay [27]), for example, make it more likely to sample experiences related to high TD-errors. Others, like Hindsight Experience Replay (see chapter 3 for details) [3] and its successor Curriculum-Guided Hindsight Experience Replay [10] also alter the rewards to reflect successful goals which differ from those sought during training.

In all cases, a curriculum is formed by giving to the agent at each step of optimization samples which are judged, by some metric, to cause the agent to learn better than a randomly-generated batch.

2.4.2 Self-Play

Self-Play methods propose tasks by reframing goal selection as an adversarial game between two agents in the same body. Though this is especially common in multiplayer competitive games like hide-and-seek [4] and fencing [35], there also exist related methods for single-player tasks ([9], [20]).

Sukhbaatar et al. (2018) [30], for instance, instantiate two agents which they call Alice and Bob. Bob is the agent which we seek to learn a multi-goal MTMDP, where a 'goal' is a desirable subset of the state space. Play alternates between Alice, who tries to manipulate the state space into a position that Bob will fail to reach, and Bob, who nevertheless attempts to maneuver to the same place as Alice.

Tasks are clearly solvable since both agents share the same body, but they grow more difficult as one learns to outpace the other. Thus, a curriculum emerges in which one agent's success leads the other to become better at foiling them, until they each can set and reach many goals.

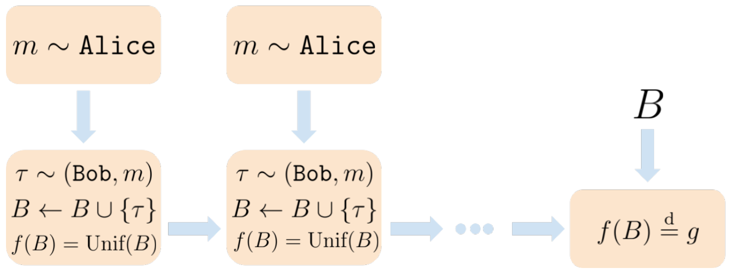
$m \sim$ Alice
$m \sim$ Alice
B
$\tau \sim (\text{Bob}, m)$
$B \leftarrow B \cup \{\tau\}$
$f(B) = \text{Unif}(B)$
$\tau \sim (\text{Bob}, m)$
$B \leftarrow B \cup \{\tau\}$
$f(B) = \text{Unif}(B)$
$f(B) \overset{d}{=} g$

Figure 2.4.2: A graph representation of [30].

2.4.3 Unsupervised Environment Design (UED)

Unsupervised Environment Design [8] tackles the curriculum learning problem by adding a new system, the **environment designer**, whose goal is to propose tasks $m \in \mathcal{T}$ which challenge the learner while still remaining feasible. It is a more significant intervention than either experience augmentation or self-play because it requires the ability to construct an environment with certain features at will, usually based on a continuous task vector.

Some methods, like ALP-GMM [24] and RIAC [5] (see chapter 3), accomplish this by proposing tasks based on a measure of Absolute Learning Progress (ALP), a statistic which is high for some task when the current policy receives different rewards on it than it has in the past. Thus, these methods keep a model of task difficulty which is used to generate new environments.

Others use a more complex learning loop. Dennis et al. (2020) [8], for instance, add an environment adversary which is itself a reinforcement learning agent. The environment adversary seeks to generate tasks which another agent, the antagonist, can learn, but which the protagonist (our goal model) cannot. Both the environment adversary and antagonist are granted reward equal to the regret between the protagonist and antagonist's achieved awards on the task, and the protagonist is granted the negation of this reward. Thus, a curriculum emerges where, as the antagonist and protagonist become better at existing tasks, the environment adversary suggests tasks at which it thinks only one agent will succeed.

Jiang et al. (2020) [13] also notes that, in settings where the environment is stochastic or partially observable, aleatoric parameters—parameters which remain uncertain regardless of curriculum experience—may cause a policy to perform suboptimally in ways which may not have been the case if trained on the full problem.

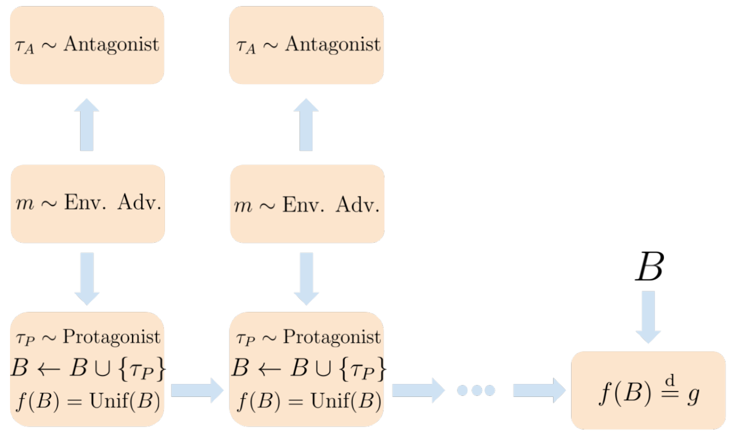

Figure 2.4.3: A graph representation of [8]. It is worth noting that, while the horizontal arrows refer to directed edges, the vertical arrows reflect the structure of the sampling function f.

2.5 Benchmarks for Curriculum Learning

Because of the diversity of curriculum methods, it is difficult to evaluate whether a curriculum method is likely to be successful on a particular problem.

Currently, the most comprehensive comparison of multiple curriculum learning methods is a 2021 benchmark by Romac et al. (2021) [26]. It tests a large number of autocurriculum methods on two Gym [32] environments which are survival-based autoscrollers, with tasks specifying various environment parameters, such as the distance between obstacles (see figure 2.5.1). In both cases, the goal is to move as far right as possible in 2000 timesteps while minimizing applied torque.

Though these problems offer a strong testbed for evaluating environment design methods, they each differ from the sparse robotics setting in meaningful ways. First, both environments employ a dense,

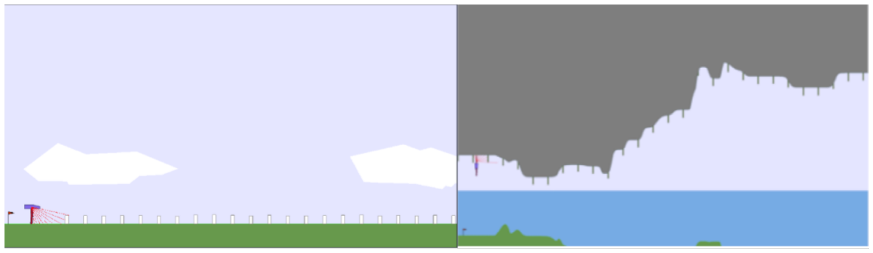

Figure 2.5.1: Examples of two example environments from TeachMyAgent [26]. Both tasks require agents of various embodiments to traverse a obstacle-riddled scene for as long as possible, while simultaneously minimizing torque.

survival-based reward, being rewarded for moving forward and penalized for using torque. Thus, they are a poor analogue for tasks where locating a reward signal may be difficult.

Second, because the task space impacts the transition dynamics of the system, a number of methods which have proven useful in robotics applications, including self-play and experience augmentation, cannot be applied to these environments. In general, any method which conceptualizes tasks as 'goal' positions which can be altered independently of the rest of the environment cannot be tested using a framework like the above. Even in curriculum learning papers centering robotics, algorithms are typically evaluated against a small number of methods which are similar in approach. Thus, it is not clear where new methods stand compared to all potential curricula.

Therefore, in the following experiments, we attempt to dissect how curriculum methods impact performance on two sparse-reward robotics tasks, paying special attention to environment design and experience augmentation algorithms.

3 Methods

In the following chapter, we describe the methods used in this thesis, including environments, algorithms, and experimental design.

3.1 Multi-Goal Reinforcement Learning

In our experiments, we consider a special case of the MTMDP setting in which the state space, action space, transition dynamics, and discount factor are constant across all tasks. Thus, only the rewards, and hence the value function, change from task to task.

We can think of this problem as goal-conditioned reinforcement learning, where the goal is given to be some desirable subset of the state space that the agent must reach (see figure 3.1.1 for a concrete

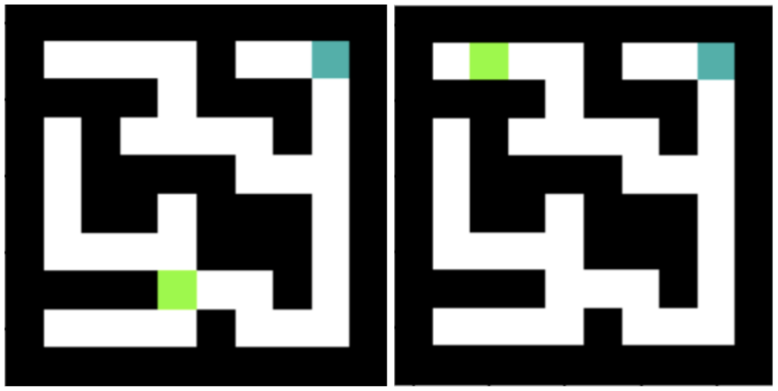
Figure 3.1.1: An example goal-conditioned MDP, with initial agent position in blue and goal in green. For two different tasks, the goal may change position, but the environment otherwise remains unchanged.

example).

As Schaul et al. (2017) [29] show, one can learn in such goal-conditioned settings a single *Universal Value Function Approximator* $V(s, m, \theta)$ and corresponding goal-conditioned policy $\pi(m, s)$, rather than a set of independent policies and value functions. This is not surprising, given that this problem is equivalent to a regular MDP as described in chapter 2. More details on the practical implementation of these functions is provided below.

3.2 The `panda-gym` Environment

There are a number of common libraries used for simulating robotic reinforcement learning tasks in Python, including MuJoCo [31], IsaacGym [16], and PyBullet [7]. For this paper, we use a library called `panda-gym` [11], which is a lightweight package implementing multiple goal-conditioned robotics tasks using the Gymnasium API [32] and simulating physics through PyBullet (see figure 3.2.1).

Each task uses the Franka Emika Panda robotic arm, a popular

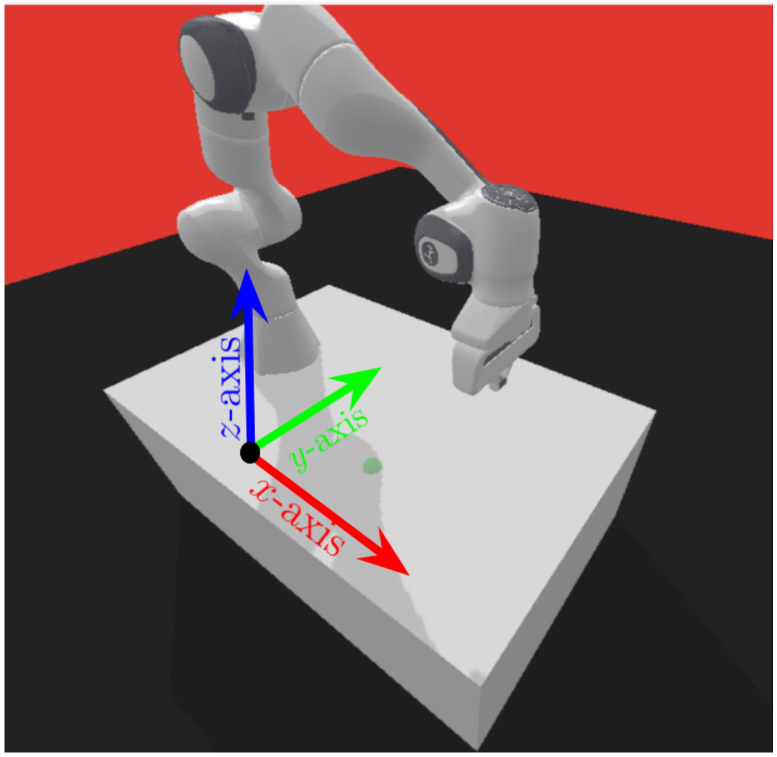

Figure 3.2.1: The `panda-gym` environment [11] The robotic arm is placed on a table, which serves as its workspace. For later experiments, note that the axes of the task space are given above.

robotic arm for manipulation tasks. Actions can be taken to be either joint motions or end-effector displacement—for the sake of simplicity, the latter is used throughout our experiments. The task-specific state spaces are described in the following subsections.

We consider two tasks, chosen due to differences in task complexity and reward sparsity.

3.2.1 curriculumReach

The `Reach` task requires the robot end-effector to navigate to a specified region in space, called the goal, within 50 timesteps.

In our experiments, we parametrize the state as a 6-dimensional vector

$$\Big[r_x, r_y, r_z, g_x, g_y, g_z\Big]$$

where $[r_x, r_y, r_z]$ is the 3D-coordinate of the arm's end-effector, and $[g_x, g_y, g_z]$ is the location of the goal. At each timestep, the agent receives a reward of -1 if it is more than `threshold` $= 0.05$ away from the goal, and otherwise it receives a reward of 0 and the episode terminates.

The default `PandaReach-v3` environment randomly sets goals, so some minor alterations were made to enable curriculum learning. In particular, rather than randomly generating goal positions on reset, a *task vector* $m \in B \subseteq \mathbb{R}^3$, where B is a box of reachable goals, must be submitted upon reset to specify the goal location for the next trajectory. For our experiments, we take the task space to be the default box $[-.15, .15] \times [-.15, .15] \times [0, .3]$.

3.2.2 curriculumPush

The `Push` task requires the robot to push a small box placed on a flat surface from a start position $[o_x^{(0)}, o_y^{(0)}, 0]$ to a goal position $[g_x, g_y, 0]$ within some square of feasible positions B. This task is more complex than the reach task in that the agent must learn to perform two distinct behaviors to succeed. First, in order to learn how its position can impact the object's position, it must learn how to move reach the box. Then, second, it must learn how to manipulate the box to reach the goal position.

We represent the state with a 12-dimensional vector

$$\Big[r_x, r_y, r_z, v_x, v_y, v_z, o_x, o_y, o_z, g_x, g_y, g_z\Big]$$

where $[r_x, r_y, r_z]$ is the robot end-effector position, $[v_x, v_y, v_z]$ is its velocity, $[o_x, o_y, o_z]$ is the current object location, and $[g_x, g_y, g_z]$ is the goal location. It is worth noting that $g_z = 0$ for all goals. However, since o_z may be non-zero as a result of the robot's moves, both parameters were kept for the sake of completeness. Likewise, the agent receives a reward of -1 until the distance between the box center and goal position is less than `threshold` $= 0.05$, at which point it receives a 0 reward and the episode terminates.

As in `curriculumReach`, we alter the `PandaPush-v3` environment so that we may specify a task by a 4-dimensional vector $[o_x^{(0)}, o_y^{(0)}, g_x, g_y]$ specifying the object start position and goal. Thus, it also reflects a larger, more challenging task space than `curriculumPush`. Like above, we let both the object and goal locations range anywhere in the default box $[-.15, .15] \times [-.15, .15]$.

3.3 Off-Policy Reinforcement Learning

Many algorithms for Multi-Task curriculum learning allow the agent to be trained by an off-policy algorithm, which we define as follows:

Definition 3.3.1 (Off-Policy Agent)**.** A learning algorithm is called off-policy if it can learn the optimal value (and a policy which optimizes that value) independently of the policy used to take actions in the world.

These algorithms learn from a memory of samples (s_t, a_t, r_t, s_{t+1}) collected under multiple policies stored in a common replay buffer, and thus any agent can add additional samples to this buffer without compromising training. Off-policy algorithms include Deep Q Networks (DQN), Deep Deterministic Policy Gradient (DDPG), and Soft Actor-Critic (SAC).

Because the original `panda-gym` paper finds that DDPG significantly outperforms SAC and TD3 over all environments in terms of sample complexity and final performance, we use DDPG as our learner across all experiments. We describe the algorithm in more detail below:

3.3.1 Deep Deterministic Policy Gradient (DDPG)

DDPG [15] is an off-policy learning algorithm intended for continuous action spaces. Paraphrasing the guide from Spinning Up in Deep RL [1], it seeks to learn the optimal Q function

$$Q^*(s,a) = \mathbb{E}_{s'}\left[r(s,a) + \gamma \max_{a' \in \mathcal{A}} Q^*(s',a')\right]$$

and use this to find a policy by taking

$$\pi^*(s) = \max_{a \in \mathcal{A}} Q^*(s,a).$$

To approximate Q^* given the replay buffer $\mathcal{D}$, we perform steps of gradient descent with the following loss:

$$L(\phi,\mathcal{D}) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\left(Q_\phi(s,a) - \left(r(s,a) + \gamma \mathbb{I}(s' \text{ is terminal}) \max_{a' \in \mathcal{A}} Q_\phi(s',a')\right)\right)^2\right]$$

By the Bellman equations, the left and right terms should be equal. Thus, minimizing their squared distance should move Q towards an optimum.

To improve training stability, we instantiate two networks for Q, one of whose parameters ϕ_{targ} (the target network) are updated at a delayed rate. When the first network updates its parameters, the target network is updated by *polyak averaging* [23], or taking only a weighted proportion of the new parameters for updating:

$$\phi_{\text{targ}} \leftarrow p\phi_{\text{targ}} + (1-p)\phi_{\text{targ}}.$$

To choose our policy, we perform gradient descent on its parameters to optimize

$$\max_{\theta} \mathbb{E}_{s\sim\rho}\left[Q_{\phi}(s, \pi_{\theta}(s))\right]$$

where the expectation is approximated using a batch of experience samples from trajectories sampled from the environment. Also, as with the Q network, we add a target policy network with parameters θ_{targ} to increase training stability when calculating the Q function loss, which is also updated via Polyak averaging.

Finally, to encourage exploration, we add during training noise ϵ to our policy, so that actions are carried out with

$$\pi'(s) = \pi_{\theta}(s) + \epsilon.$$

Using the recommendation from the original DDPG paper, time-correlated Ornstein Uhlenbeck (OU) Action Noise (an approximation of Brownian motion) is used as our noise term. At test time, the deterministic policy is used.

3.4 Curriculum Methods

In the following section, we describe the algorithms used for curriculum learning. Each takes the agent to be a DDPG agent, as specified above. For specific hyperparameters, see section 3.5.

3.4.1 Hindsight Experience Replay (HER)

Hindsight Experience Replay (HER) [3], which falls under the *experience augmentation* umbrella described in chapter 2, has proven to be a powerful method for generating reward signals in sparse-reward tasks, such as those common in robotics. All of the baseline results in the original `panda-gym` paper use HER, and initial experiments using only DDPG yield no meaningful reward signal over a large number of episodes in the `curriculumPush` task.

HER works by augmenting an agent's replay buffer with successful trajectories, even when the policy does not reach its intended goal. For instance, in the `curriculumReach` task, after collecting a trajectory, we may not reach the intended goal position. However, if we imagine that the goal was instead wherever the end effector ended up, we would have gathered a successful demonstration of this goal. HER works by adding both the real experience and the imagined experience to the dataset buffer, allowing for the agent to always gain a sample with a meaningful reward (for a concrete example, see figure 3.4.1).

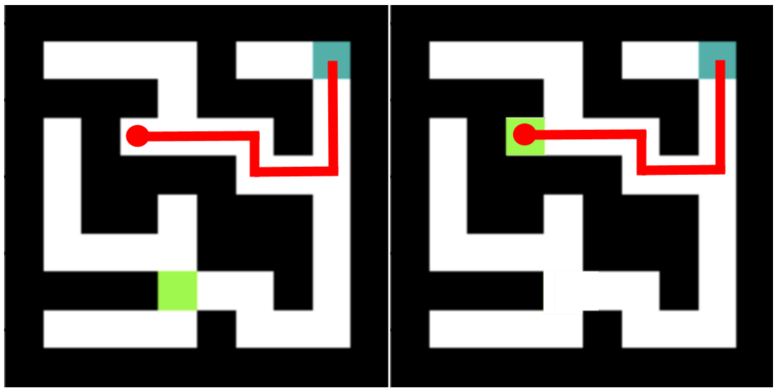

Figure 3.4.1: An example of a HER-augmented trajectory. An agent rolls out a trajectory with some goal m (left), potentially failing to reach it. However, this sample *does* demonstrate a successful trajectory for a different goal m' (right). Then we may add both each real transition sample $(m, s_t, a_t, r_m(s_t, a_t), s_{t+1})$ and each reconstructed transition sample $(m', s_t, a_t, r_{m'}(s_t, a_t), s_{t+1})$, with the latter representing a successful trajectory.

The complete algorithm is laid out below (alg. 1). This algorithm forms an implicit curriculum by populating the agent's memory with samples corresponding to ‘easy’ goals, in the sense that they are reachable by the agent moving under its current policy. Thus, as the agent masters more tasks, its buffer will begin to reflect increasingly difficult problems.

Still, it is worth noting that this algorithm is not without limitations. First, it requires that the user be able to supply and annotate rewards for trajectories relative to arbitrary tasks without actually running them in the environment. In some settings, where the explicit form of the reward function is unknown, this may be difficult.

Second, in some settings, there is not always a way to transform every unsuccessful trajectory into a meaningful successful one. In the `curriculumPush` environment, for instance, a trajectory that fails to move the box at all can only represent a task where the initial object position is already the goal, and thus cannot inform how the robot should manipulate the box.

3.4.2 Absolute Learning Progress Gaussian Mixture Model (ALP-GMM)

ALP-GMM [24] is an unsupervised environment design method that samples tasks based on a measure called the **Absolute Learning Progress (ALP)**. Formally, the ALP of a task that has been seen at least twice, with most recent cumulative rewards r_{new} and r_{old}, is said to have

$$alp_{new} = |r_{\text{new}} - r_{\text{old}}|.$$

In continuous settings, we loosen this definition to accept the newest r_{old} within some ball of the new task when determining its ALP.

ALP-GMM works by keeping track of the learning progress associated with each sampled task. In an initial ‘bootstrap phase,’ the

Algorithm 1 `Hindsight Experience Replay (HER)`

```
Input: An off-policy RL algorithm $\mathbb{A}$, sampling strategy $\mathbb{S}$, and
reward function $r$ :

Initialize $\mathbb{A}$ and replay buffer $R$.

for episode= $1, \dots, M$ do
    Sample a task $m \sim \mathcal{T}$ and initial state $s_0 \sim \rho_m$.
    Let $\pi_m = \texttt{POLICY}(m)$
    for $t = 0, \dots, T-1$ do
        Sample action $a_t \sim \pi_m$.
        Execute $a_t$ and observe new state $s_{t+1}$.
    end for
    for $t = 0, \dots, T-1$ do
        Calculate $r_t = r_m(s_t, a_t)$.
        Store transition $(m, s_t, a_t, r_t, s_{t+1})$ in $R$.
        Sample a set of replay tasks $G \sim \mathbb{S}(\textbf{episode})$.
        for $m' \in G$ do
            $r' = r_{m'}(s_t, a_t)$
            Store $(m', s_t, a_t, r', s_{t+1})$ in $R$.
        end for
    end for
    for $n = 1, \dots, N$ do
        Sample a batch $B$ from $R$.
        Perform one step of optimization using $\mathbb{A}$ and $B$.
    end for
end for
return POLICY.
```

algorithm randomly selects tasks from the space via the ground truth distribution g. This initializes a history of task-reward pairs which can be used to calculate learning progress for new tasks.

After this bootstrap phase is over, ALP-GMM instead constructs a probability distribution fitting the (normalized) ALP over the task space. A set of Gaussian Mixture Models [25] of between 2 and k_{max} Gaussians are fit using standard algorithms (the EM Algorithm, see [18]).

The model used for sampling is chosen by the Akaike Information Criterion (AIC) [2], a common heuristic for model selection that assigns a model f a score based on the complexity of the model (n_{params}) and the likelihood of the data in B:

$$AIC = 2n_{\text{params}} - ln(L(B|f)).$$

ALP-GMM's sampling scheme causes tasks which are suddenly successful after a long period of failure, or conversely which fail after being learned for some time, to be focused on during training. It forms a curriculum by always supplying tasks which are both possible and challenging to the agent, updating its model by recomputing ALP as the agent masters different region in space.

3.4.3 Robust Intelligent Adaptive Curiosity (RIAC)

Robust Intelligent Adaptive Curiosity (RIAC) also seeks to maximize learning progress, though rather than fitting a Gaussian mixture, it instead iteratively splits the task space into subregions differentiated by learning progress. Like ALP-GMM, it keeps a history of learning progress across different points in parameter space, but initially all samples are contained in a single region $R_0 = \mathcal{T}$.

The algorithm collects ALP samples for different tasks like

Algorithm 2 `Absolute Learning Progress Gaussian Mixture Model (ALP-GMM)`

```
Initialize a buffer $B$ of size $N$, probability of random sampling $p_{rnd}$,
and maximum number of Gaussians $k_max$. Initialize history $\mathcal{H}$

for $i = 1, \ldots, N$ do
    Sample a random $m \in \mathcal{T}$, and collect $\tau \sim (\pi, m)$.
    Compute ALP of $m$, and store $m, ALP_m$ in $B$, $(m, r_m)$ in $\mathcal{H}$.
end for
for $\ell = 1, 2, \ldots$ do
    Fit GMMS with 2 to $k_{max}$ Gaussians on $B$.
    Take the Gaussian with the highest AIC.
    for $i = 1, \ldots, N$ do
        $p_{rnd}\%$ of the time, sample a random $m \in \mathcal{T}$. Otherwise, sample
        $m$ from GMM proportionally to mean ALP.
        Collect a new trajectory $\tau \sim (\pi, m)$.
        Compute ALP of $m$, and store $m, ALP_m$ in $B$, $(m, r_m)$ in $\mathcal{H}$.
    end for
end for
```

ALP-GMM, but it dynamically splits its subregions into smaller ones when they reach a critical threshold of samples rather than forming a single model. These splits are chosen to maximize the difference in ALP between the newly-formed regions, while maintaining a minimum area in each.

The goal is for uncertainty to be organized into small, highly-contested regions of task space which are sampled from disproportionately. We implement the algorithm by [24], which has some minor modifications from the original algorithm to improve curriculum quality and reduce the chance of oversplitting the task space.

3.5 Experimental Details

With minor modifications, we use the parameter settings described in appendix A of the `panda-gym` paper [11], which are themselves taken from Plappert et al. (2018) [21], a similar paper implementing common robotics tasks on a Fetch robot in Gym with MuJoCo. DDPG was implemented following a guide by Machine Learning with Phil [17].

The following section describes the hyperparameters and implementation details used in the experiments:

3.5.1 Networks

The actor and critic networks of DDPG each parametrized by a fully-connected network (FCN) with three layers of 256 nodes each with ReLU activations. The magnitude of each displacement is constrained to lie in $(-1, 1)$ by the tanh function. DDPG Polyak averaging of the target network is weighted by $p = 0.95$. Policy noise is OU Action noise, following the recommendation given in the original DDPG paper [15].

3.5.2 HER

Unlike [11], we use the following HER sampling strategy:

1. If the episode was successful, add only the successful transition to the buffer.

2. If the episode was unsuccessful, add a single HER replay corresponding to the goal achieved by the final state.

The authors do not describe in detail their sampling strategy, specifying only that they generate 4 HER replays for each episode. In early experiments, adding HER replays from 4 randomly-selected goals within each trajectory led to catastrophic forgetting in the `curriculumPush` task. This is potentially due to the fact that, as mentioned above, reconstructed goals in early experiences, where the block does not move throughout the episode, are uninformative. Populating the majority of the buffer with these samples thus yields low-quality training data.

After each episode, we perform $N = 50$ steps of optimization on the actor and critic. This is chosen to reflect the DDPG training loop, which performs one step of optimization for each step taken in the environment (50 for unsuccessful episodes).

3.5.3 Training

We train models using Adam Optimization [14] with a learning rate $LR = 0.001$ for both the actor and critic and batch size of $B = 256$. Replay buffers store 10^6 transitions. To encourage exploration, we take a random action with probability 0.3 at each timestep.

For the `curriculumReach` task, we train for 50 epochs (sets of 50 episodes). In practice, this is sufficient for the best of our algorithms (ALP-GMM+HER) to converge to near perfect success rate. For the `curriculumPush` task, we train for 1000 epochs.

With the above parameters, `curriculumReach` models take around 1 hour to train, and `curriculumPush` models take around 2 days on a single NVIDIA RTX 4070 GPU.

4 Results

In the following chapter, we present our results evaluating various curriculum methods on the `curriculumPush` and `curriculumReach` environments. We consider five possible curriculum learning strategies:

1. HER
2. ALP-GMM
3. RIAC
4. ALP-GMM+HER
5. RIAC+HER

All models are trained using the DDPG agent described in chapter 3. For each experiment, the model parameters used in evaluation is taken to be those which achieved the highest evaluation performance throughout training, rather than the learned parameters at the end of training. Final success rate is calculated by randomly sampling $N = 1000$ tasks from the ground truth distribution Unif($\mathcal{T}$) and evaluating each method on this set of tasks.

We analyze performance by three criteria:

1. We compare the success rate of each model against samples drawn from the ground truth distribution (figures 4.1.1, 4.2.1).

2. We visualize as a heatmap the reward achieved by each agent as a function of the task (figures 4.1.2, 4.2.2). This allows us to see qualitatively which regions in task space are mastered by each policy, and which remain challenging.

3. We assess qualitatively the differences in training curves between models (figures 4.1.3, 4.2.3).

To find the learning curves, we evaluate every 50 episodes (1 epoch) on 80 randomly-generated tasks during training. In the `curriculumReach` task, 5 experiments are conducted for each algorithm, and the maximum, mean, and minimum are all shown. For `curriculumPush`, only a single run is shown for each algorithm, as training as few as 3 models with each algorithm would take nearly a month in continuous compute time, which was unfeasible on a single laptop.

To generate the heatmaps, a random model was chosen from each experiment, and a trajectory was a collected for each of the 100^2 goal positions, holding in each row a constant z-coordinate or initial object position, depending on the task. Because the learned policy is deterministic, it is not necessary to evaluate the model on samples from each task setting. Since a failed task incurs a -50 reward, the dark purple regions reflect tasks which the agent still cannot complete.

To see tables containing the performance statistics captured in the bar plots below, see figures A.0.3 and A.0.2 in the appendix.

4.1 `curriculumReach` Results

Below we describe the results of the `curriculumReach` experiments.

4.1.1 Overall Performance

Evaluation Success Proportion and Average Reward
For `curriculumReach` Models

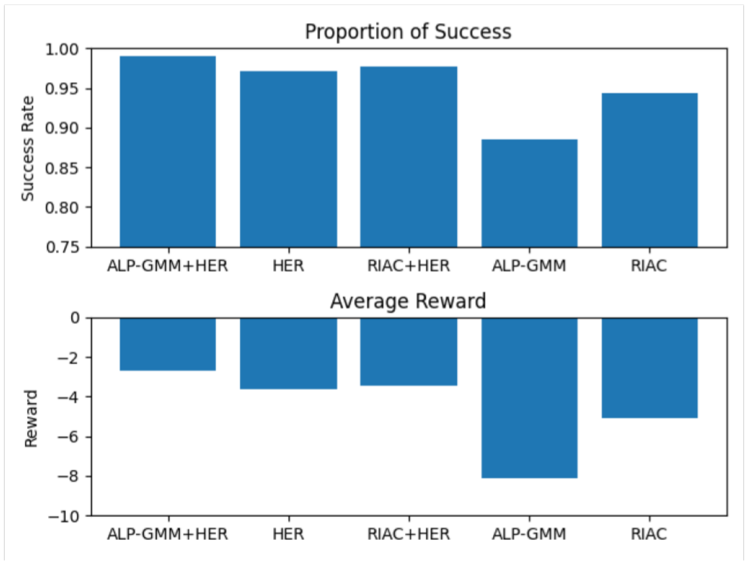

Figure 4.1.1: (Top) Percent of successful trajectories and (Bottom) average reward incurred over 1000 random tasks in `curriculumReach`.

Figure 4.1.1 shows a bar plot of the final performance for each model.

We see that, in all cases, the learned models achieve nearly or greater than 90% success on tasks drawn from the ground truth distribution, regardless of whether HER is used. The worst-performing model is ALP-GMM without HER, which achieves an 88.6% success rate. The model with the highest performance, ALP-GMM+HER, successfully completes 99% of tasks. In all cases, HER-augmented algorithms achieve higher performance than those trained without it.

There does not seem to be a meaningful difference in reward that is not explained by the difference in success rate. Since this reward is equal to the negative time of completion in the case of a successful episode, or -50 in an unsuccessful one, we find that no policy learns a significantly faster strategy for reaching the goal than any other.

4.1.2 Goal Generalization

Figure 4.1.2 shows the reward incurred by each policy for different goal positions. Each row represents a fixed z-position (height) above the table. Within each row, the heatmap shows the reward incurred by the agent when the x and y position of the goal is made the corresponding point on the heatmap. Visually, we can consider each heatmap as a slice of the performance in height space, looking directly downwards onto the table.

We find that, for intermediate heights in the task space, all five methods yield a policy that can perform well. Nearing the extremes of the task space, however, RIAC and ALP-GMM+HER perform most consistently, whereas ALP-GMM in particular suffers.

It is worth noting that, in general, all models perform worse at extreme low heights than extreme high ones. This may be because these positions correspond to points touching the table, so only half of the sphere of valid end-effector states specifying the goal are potential destinations for the agent.

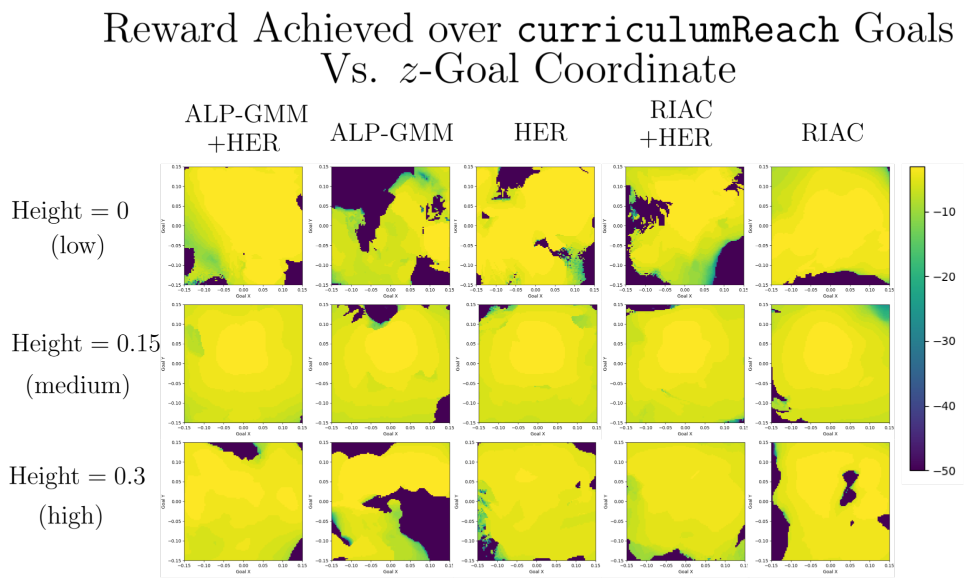

Figure 4.1.2: Each heatmap shows the achieved reward for each method's policy for 100^2 (x, y) goal positions in the task space for for three heights z—low height (row 1), medium height (row 2), and hight height (row 3).

When models do fail, they tend to do so in contiguous regions, rather than at random points in space. This supports our intuition that learning goals in one region should generally help a policy learn similar goals nearby.

4.1.3 Training Efficiency

Figure 4.1.3 shows the learning curves for each model. While all models trained with HER have quite similar curves, both ALP-GMM and RIAC alone improve more slowly, especially in early epochs.

Regardless, it appears as though all algorithms are converging to the optimal policy, though at different rates. This is most likely a feature of the simple task, as even random policies are likely to incur some successful demonstrations due to luck, allowing any agent to

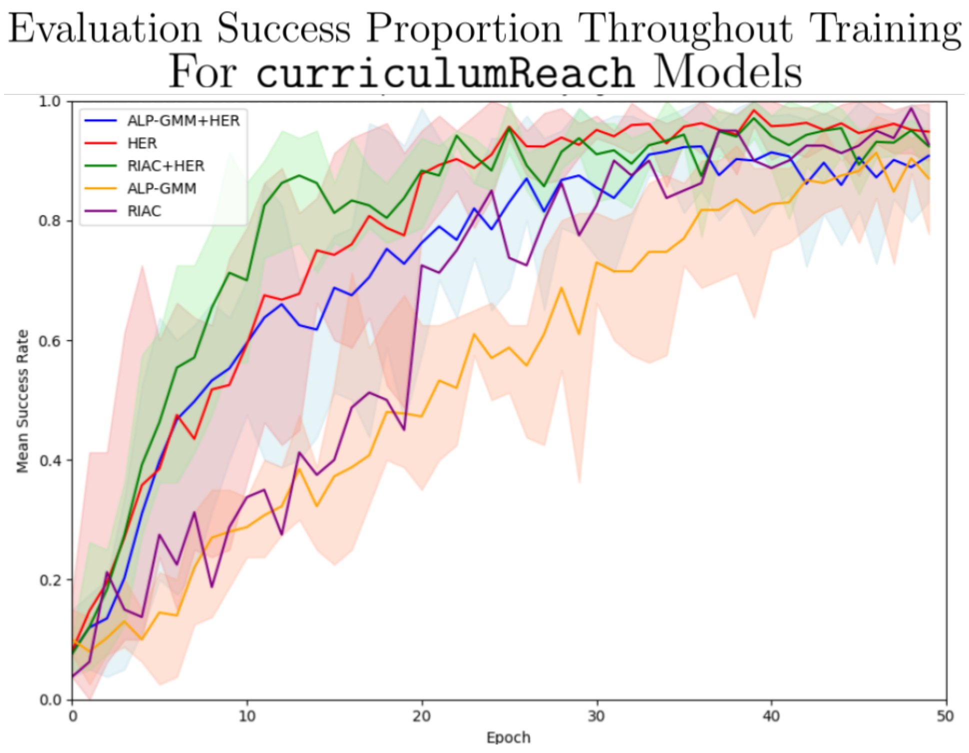

Figure 4.1.3: The learning curves for each algorithm on the `curriculumReach` task. Dark lines reflect the mean evaluation performance of 5 independent trials, with shaded regions denoting the maximum and minimum success rate achieved at each epoch. over five trials

receive a learning signal relatively early in training.

4.2 curriculumPush Results

Below we describe the results of the `curriculumPush` experiments.

4.2.1 Overall Performance

The final performance of each model is shown in figure 4.2.1.

We see that, unlike the `curriculumReach` task, only curricula

Evaluation Success Proportion and Average Reward
For `curriculumPush` Models

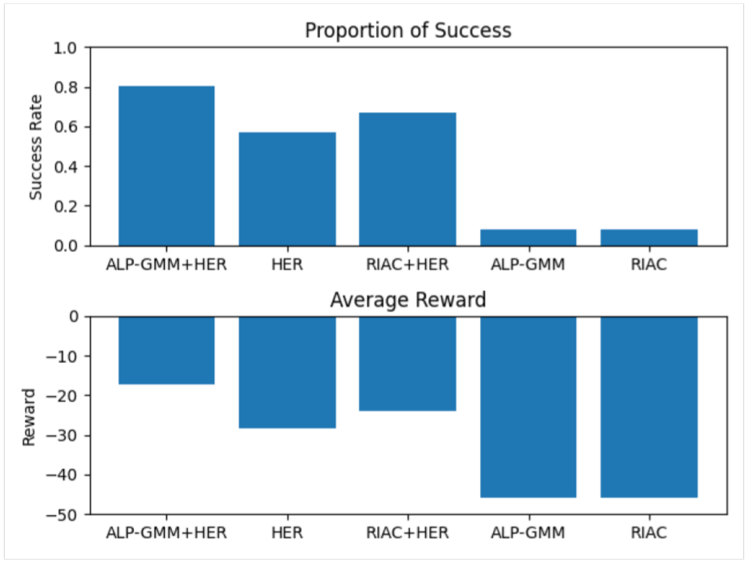

Figure 4.2.1: (Top) Percent of successful trajectories and (Bottom) average reward incurred over 1000 random tasks in `curriculumPush`.

utilizing HER are able to learn a successful policy. ALP-GMM+HER once again learns the best policy, achieving a success rate of 80.3%.

Note that there is always a chance of a randomly-generated object being placed within threshold = .05 of the goal automatically and immediately winning (see the yellow circles in figure 4.2.2). Thus, some of the successes of ALP-GMM and RIAC should be considered to be essentially chance. These results suggest that HER is a critical component for learning the `curriculumPush` task in a sample-efficient manner, as we discuss further in chapter 5.

RIAC+HER and ALP-GMM+HER also outperform HER on its own, with ALP-GMM solving over 20% more tasks. Thus, while HER is necessary for ensuring good performance, it also benefits from an additional environment designer adaptively-sampling environments.

Like in `curriculumReach`, the variation in average reward appears to be explainable by the success rate, rather than a significant difference in speed when solving each task.

4.2.2 Goal Generalization

As above, in figure 4.2.2 we see the reward accrued by each successful policy for fixed initial positions, varying only the goal location to generate the rewards in the heatmap (see the appendix A for a visualization of the reward over more initial object positions). Note that the circular yellow region in each heatmap reflects goals which are immediately achieved because the object is initialized within the 0.05 threshold.

As the figure shows, all policies perform best in general on samples where the object begins near the center of the task space. As the object is placed in more extreme regions of the space, all three methods struggle, though HER in particular completely fails at each task. As we will discuss in chapter 5, both ALP-GMM and RIAC may adaptively weight samples in these difficult regions during training until they are mastered, whereas HER continues to randomly sample tasks from (mostly) central regions in task space late in training.

It is worth noting that, though both ALP-GMM and RIAC perform well in larger regions of goal space than HER alone across all object initializations, the structure of their performance is qualitatively different. In particular, where ALP-GMM is successful, it tends to be similarly successful in contiguous regions in parameter space, and conversely it tends to fail in coherent regions. In comparison, RIAC exhibits more ‘holes’ in otherwise good regions of task space. This may be because, where ALP-GMM fits a smooth function to learning progress, boosting the likelihood of sampling all trajectories in a ellipsoid region, RIAC splits the parameter space into arbitrarily small subregions based on differences in learning progress. It has been

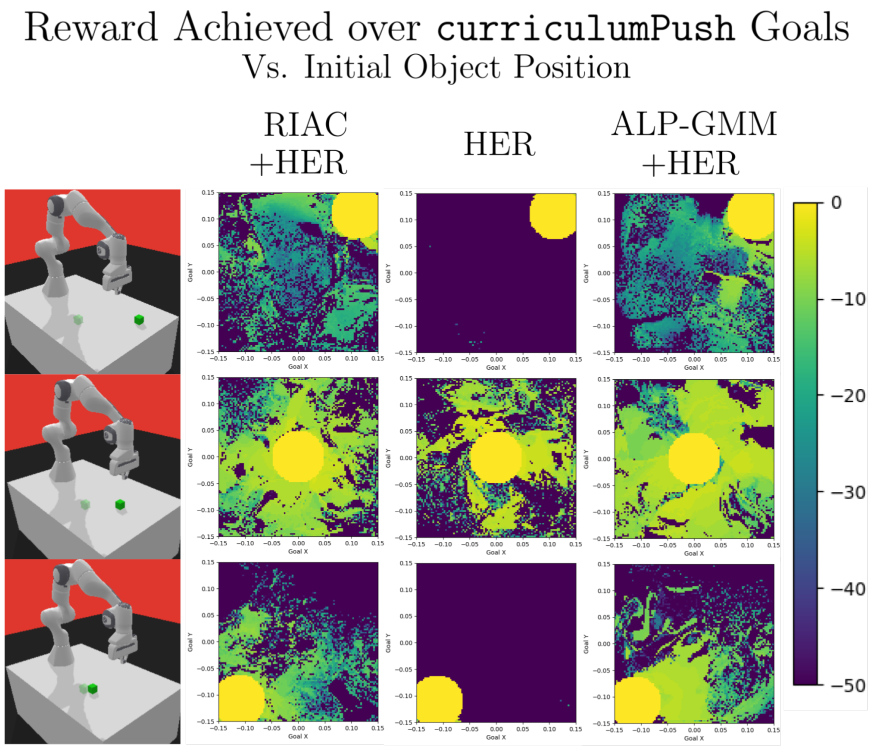

Figure 4.2.2: Each heatmap shows the achieved reward for each method's policy on 100^2 goal positions in the task space, with initial object position on the top-left of the task space (first row), center of the task space (second row), and bottom-right of the task space(third row).

noted that RIAC also tends to oversplit the task space, yielding at times suboptimal curricula as a result [26].

4.2.3 Training Efficiency

The training curves for each model are visualized in figure 4.2.3.

Unlike `curriculumReach`, we see that each model converges to a different success rate, rather than to optimal performance. This reflects many difficult RL tasks, in which curricula can be the difference between a good and a bad policy, rather than simply yielding a difference in sample efficiency.

Initially, RIAC+HER performs significantly worse than the other HER methods. The cause of this performance difference is not obvious, though perhaps it is a function of early reward sparsity. Before a reasonable policy is learned, the ALP landscape is exclusively 0, and so both RIAC and ALP-GMM sample uniformly. However, when a success is attained, each method adjust their models differently. ALP-GMM continues to look mostly uniform, with potentially a small single peak. However, RIAC is likely to split this region into a smaller subregion, focusing too much attention on that area of the parameter space rather than continuing to explore.

Evaluation Success Proportion Throughout Training For `curriculumPush` Models

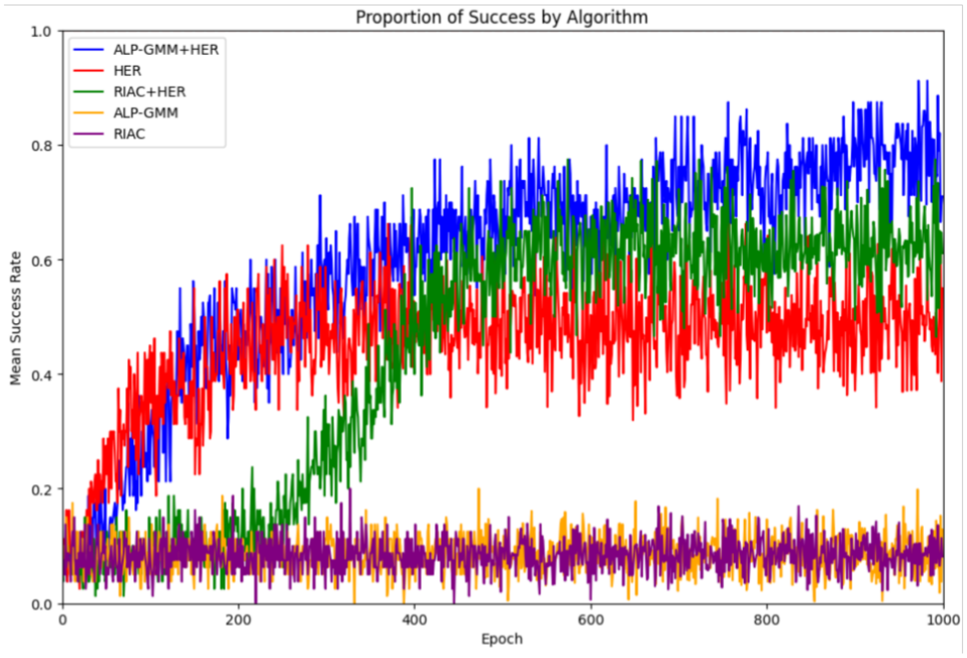

Figure 4.2.3: The learning curve for each method on the more complex `curriculumPush` environment.

5
Discussion

In the following chapter, we discuss the significance of the results presented in chapter 4, as well as any limitations and directions for future research.

5.1 Discussion

In both experiments, we find that HER assists training in ways qualitatively-distinct from the UED algorithms ALP-GMM and RIAC. This can be seen by performance, with both tasks succeeding more often with HER and a UED method than with either alone, and also by examining the set of mastered tasks in the heatmaps for both environments. In the more complex `curriculumPush` setting, in particular, incorporating HER is necessary for the model to learn anything about how to act in the environment, and RIAC and ALP-GMM are necessary for it to learn how to perform for a large set of goals, as evidenced by the heatmaps of reward at extreme regions of space.

These data suggest that an effective curriculum in sparse-reward settings must perform two distinct functions:

5.1.1 Signal Engineering

First, it must provide a mechanism for receiving any learning signal at all. This *signal engineering* capacity dramatically improves training efficiency by providing a direction for model improvement when randomly training on tasks would provide no signal.

In simple cases such as problems with sufficiently dense rewards, this capacity is less likely to be necessary to achieve success, and instead assists learning by yielding faster convergence to the optimal policy. Even in the sparse `curriculumReach` setting, the likelihood of randomly solving a task is sufficiently high that a UED method on its own can still extract a useful signal. However, we see that incorporating HER significantly improves sample efficiency, as the buffer immediately receives demonstrations of successful trajectories, rather than wasting time providing failing examples.

However, when the probability of randomly solving a task becomes lower, such as in `curriculumPush`, it becomes difficult to extract a meaningful reward signal early in training. Though other curriculum methods like ALP-GMM and RIAC can model the task space after a signal has been achieved, the ALP will remain 0 for all states until a successful task is located out of a potentially harsh task space, hampering training.

5.1.2 Guided Exploration

Second, to continue to learn later in training without falling into a local optimum, the curriculum must provide a source of *guided exploration*, recommending at later timesteps useful tasks that will allow the agent to learn difficult edge cases of the environment and to avoid forgetting. The value of this component of curriculum learning is especially clear in `curriculumPush`, where the agent trained using HER alone plateaus at a 56.7% success rate. As the heatmaps in figure 4.2.2 show, HER does learn a broadly successful policy for pushing the block from central regions of the task space, but it struggles when it is placed in less common regions. UED methods constantly supply examples which push the model into uncertain regions of parameter space, yielding as a result a more robust and general policy.

5.2 Analogies to Supervised Curricula

These two capacities serve as natural analogues for the functions of *guiding* and *denoising* discussed by [33] in chapter 2. From this perspective, long-horizon sparse RL tasks can be seen as a highly noisy dataset, with many qualitatively-distinct categories being collapsed into two classes: successful and unsuccessful. Thus, signal engineering works to denoise trajectory data by reducing the set of highly ambiguous failures into clearer successful examples.

Likewise, guided exploration serves to lead the agent to challenging but learnable regions of parameter space, improving as a result its learning relative to a random set of tasks. It serves to regularly move the model parameters to nearby regions in space which are more capable of performing well on general tasks.

Many existing RL curricula, like in the supervised setting, prioritize only one of these objectives. However, where in the supervised setting this can still yield improvements over training without a curriculum, such RL curricula may fail to perform well in complex sparse-reward tasks, as our experiments show. This effect of curricula yielding a large difference in asymptotic performance is unusual in the supervised setting, where in general curriculum strategies impact sample efficiency more than the final network. Thus, the reinforcement learning problem is somewhat unique in that many sparse-reward long horizon tasks could see such an increase in performance by adopting such a general curriculum approach.

5.3 Limitations and Future Directions

It is worth noting some of the limitations of the current approach. For one, only a small subset of curriculum methods were used, and, though the results are consistent across two distinct UED algorithms, they are unified by seeking to produce samples based on a shared metric—absolute learning progress. It remains future work to perform a more comprehensive analysis across a greater number of tasks, though for such an undertaking a compute cluster will surely be necessary.

In addition, no self-play methods were considered in this work. This is in part because it is difficult to compare them on an even playing field with UED and Experience Augmentation methods—self-play methods often require double or more the number of demonstrations in the environment, and it is unclear then how to compare sample complexity between methods. Many self-play methods also are more difficult to incorporate with experience augmentation methods like HER due to the presence of multiple agents with different, interlocking policies. Thus, in such cases, it is difficult to compare these methods in terms of the metrics used in this thesis.

They are perhaps best suited for real-world robotics training, in contexts where it is difficult or impossible to design the environment. Nevertheless, some of the most dramatic results in curriculum reinforcement learning can be described as self-play-based, and so additional work evaluating such methods on robotics tasks, and perhaps creating self-play methods that more explicitly denoise the reward space, could be very useful.

5.4 Conclusion

In short, this thesis addresses the problem of curriculum learning in complex robotics settings, where a single policy must learn to perform a large set of related tasks.

By evaluating curricula which perform different functions throughout training, we show that reinforcement learning curricula can be seen to benefit learning in two complementary ways—*signal engineering* and *guided exploration.* These functions are analogous to the common strategies for describing curricula in the supervised setting, though they can be far more impactful in the reinforcement learning setting due to the inherent size and complexity of the domain, yielding policies which perform better at test time than similar models trained without a curriculum.

We show that, for two classes of state-of-the-art curriculum learning methods, *signal engineering* is vital for learning successful policies on difficult multi-goal tasks where reward signals are sufficiently sparse. Moreover, we show that adding an additional component of the curriculum which focuses on improving exploration improves the reward further, resulting in a policy which continues to learn more generalizable skills throughout training, unlike models trained using only one component of the curriculum.

These results suggest that, for an RL curriculum to be effective at such tasks, it must provide both of these services. With any luck, a model trained with such a curriculum will allow machine learning research to tackle ever more complex challenges, enabling robots that can solve a huge variety of problems in the world.

A
Appendix

Below is the full visualization of the learned `curriculumPush` policy, on 5, rather than 3, initial object positions, as well as tables containing the final success states and rewards acheived by each model.

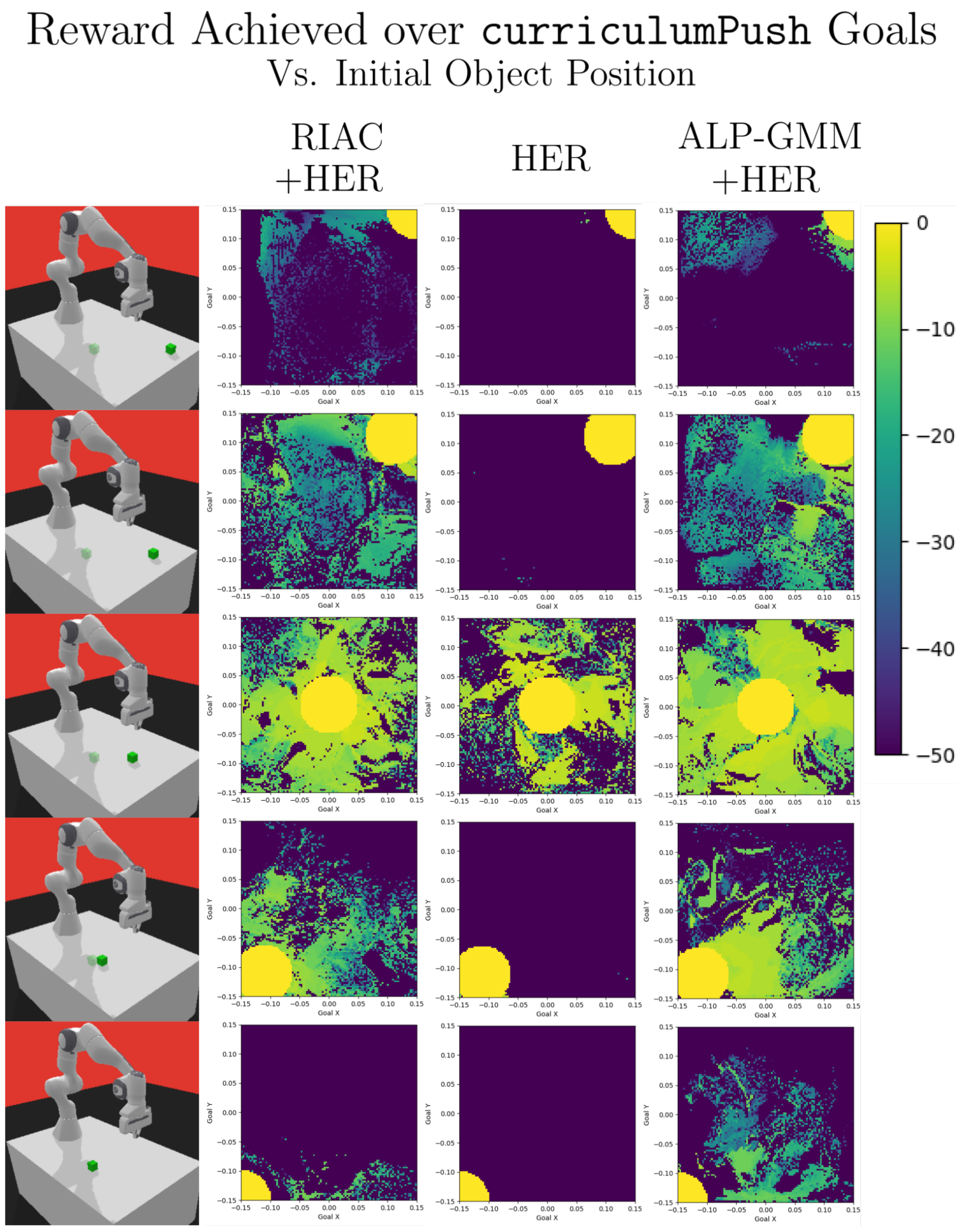

Figure A.0.1: Each heatmap shows the achieved reward for each method's policy for 100 goal positions in the task space with initial object position on the top-left (first row), mid-top-left (second), center (third), mid-bottom right (fourth), and bottom-right of the task space(fifth).

Final Success Rate Attained by Each method		
	`curriculumReach`	`curriculumPush`
ALP-GMM+HER	**0.990**	**0.803**
HER	0.972	0.568
RIAC+HER	0.977	0.669
ALP-GMM	0.886	0.081
RIAC	0.944	0.081

Figure A.0.2: Final evaluation success rate for each model on the `curriculumReach` and `curriculumPush` tasks. Models were evaluated on 1000 random goals, with the proportion of successful episodes being presented above.

Final Rewards Attained by Each method		
	`curriculumReach`	`curriculumPush`
ALP-GMM+HER	**-2.688**	**-17.401**
HER	-3.613	-28.474
RIAC+HER	-3.471	-23.94
ALP-GMM	-8.143	-45.95
RIAC	-5.101	-45.95

Figure A.0.3: Final evaluation rewards for each model on the `curriculumReach` and `curriculumPush` tasks. Models were evaluated on 1000 random goals, and rewards range from -50 to 0.

References

[1] Joshua Achiam. Spinning Up in Deep Reinforcement Learning. 2018.

[2] Hirotugu Akaike. Akaike's information criterion. *International encyclopedia of statistical science*, pages 25–25, 2011.

[3] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017.

[4] Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula. *arXiv preprint arXiv:1909.07528*, 2019.

[5] Adrien Baranes and Pierre-Yves Oudeyer. R-iac: Robust intrinsically motivated exploration and active learning. *IEEE Transactions on Autonomous Mental Development*, 1(3):155–169, 2009.

[6] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

[7] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016–2021.

[8] Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. *Advances in neural information processing systems*, 33:13049–13061, 2020.

[9] Yuqing Du, Pieter Abbeel, and Aditya Grover. It takes four to tango: Multiagent selfplay for automatic curriculum generation. *arXiv preprint arXiv:2202.10608*, 2022.

[10] Meng Fang, Tianyi Zhou, Yali Du, Lei Han, and Zhengyou Zhang. Curriculum-guided hindsight experience replay. *Advances in neural information processing systems*, 32, 2019.

[11] Quentin Gallouédec, Nicolas Cazin, Emmanuel Dellandréa, and Liming Chen. panda-gym: Open-Source Goal-Conditioned Environments for Robotic Learning. *4th Robot Learning Workshop: Self-Supervised and Lifelong Learning at NeurIPS*, 2021.

[12] Lu Jiang, Deyu Meng, Teruko Mitamura, and Alexander G Hauptmann. Easy samples first: Self-paced reranking for zero-example multimedia search. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 547–556, 2014.

[13] Minqi Jiang, Michael Dennis, Jack Parker-Holder, Andrei Lupu, Heinrich Küttler, Edward Grefenstette, Tim Rocktäschel, and Jakob Foerster. Grounding aleatoric uncertainty for unsupervised environment design. *Advances in Neural Information Processing Systems*, 35:32868–32881, 2022.

[14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

[16] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac gym: High performance gpu-based physics simulation for robot learning, 2021.

[17] MLWithPhil. Reinforcement learning in continuous action spaces | ddpg tutorial (pytorch), Jun 2019.

[18] Todd K Moon. The expectation-maximization algorithm. *IEEE Signal processing magazine*, 13(6):47–60, 1996.

[19] Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *Journal of Machine Learning Research*, 21(181):1–50, 2020.

[20] OpenAI OpenAI, Matthias Plappert, Raul Sampedro, Tao Xu, Ilge Akkaya, Vineet Kosaraju, Peter Welinder, Ruben D'Sa, Arthur Petron, Henrique P d O Pinto, et al. Asymmetric self-play for automatic goal discovery in robotic manipulation. *arXiv preprint arXiv:2101.04882*, 2021.

[21] Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, et al. Multi-goal

reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*, 2018.

[22] Emmanouil Antonios Platanios, Otilia Stretcu, Graham Neubig, Barnabas Poczos, and Tom M Mitchell. Competence-based curriculum learning for neural machine translation. *arXiv preprint arXiv:1903.09848*, 2019.

[23] Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM journal on control and optimization*, 30(4):838–855, 1992.

[24] Rémy Portelas, Cédric Colas, Katja Hofmann, and Pierre-Yves Oudeyer. Teacher algorithms for curriculum learning of deep rl in continuously parameterized environments. In *Conference on Robot Learning*, pages 835–853. PMLR, 2020.

[25] Douglas A Reynolds et al. Gaussian mixture models. *Encyclopedia of biometrics*, 741(659-663), 2009.

[26] Clément Romac, Rémy Portelas, Katja Hofmann, and Pierre-Yves Oudeyer. Teachmyagent: a benchmark for automatic curriculum learning in deep RL. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 9052–9063. PMLR, 2021.

[27] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[28] Burr Settles. Active learning literature survey. 2009.

[29] Sandra Stotsky. An english language arts curriculum framework for american public schools: A model for use by any state or

school district without charge. *Nonpartisan Education Review*, 13(2):1–83, 2017.

[30] Sainbayar Sukhbaatar, Zeming Lin, Ilya Kostrikov, Gabriel Synnaeve, Arthur Szlam, and Rob Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. *arXiv preprint arXiv:1703.05407*, 2017.

[31] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[32] Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, March 2023.

[33] Xin Wang, Yudong Chen, and Wenwu Zhu. A survey on curriculum learning. *IEEE transactions on pattern analysis and machine intelligence*, 44(9):4555–4576, 2021.

[34] Xiaoxia Wu, Ethan Dyer, and Behnam Neyshabur. When do curricula work? *arXiv preprint arXiv:2012.03107*, 2020.

[35] Boling Yang, Liyuan Zheng, Lillian J Ratliff, Byron Boots, and Joshua R Smith. Stackelberg games for learning emergent behaviors during competitive autocurricula. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5501–5507. IEEE, 2023.