UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

JÚLIA EIDELWEIN

Active Selection of Continuous Training Tasks to Learn Generalizable Policies

Work presented in partial fulfillment
of the requirements for the degree of
Bachelor in Computer Science

Advisor: Prof. Dr. Anderson Rocha Tavares
Coadvisor: Prof. Dr. Bruno Castro da Silva

Porto Alegre
December 2021

ABSTRACT

A key challenge when deploying reinforcement learning (RL) algorithms in robotics settings is the large number of interactions between the agent and its environment that are necessary for an optimal behavior to be learned. An alternative to training a robot in the real world is to execute the training process in a simulator. However, for an RL agent to be successfully deployed in real-life, one needs to guarantee that the characteristics of the environment where it will operate are accurately modeled by the simulator. This might be difficult to ensure and, as a result, the robot may have to—when deployed in real-life—interact with environments whose dynamics are *different* than those experienced during training in simulation. This typically results in policies that were optimal in simulation but that perform poorly in the real world.

In this work we investigate how to design learning agents that are robust to settings such as this—i.e., settings where the agent may have to interact with different types of environments, and where it is not capable of directly identifying with which environment it is interacting. This corresponds to the setting where the agent's *state* (which stores all information collected by its sensors) might not be sufficiently complete or powerful to characterize all properties of the environment (or task) being tackled by the agent. In particular, we wish to design novel training algorithms that result in control policies that are robust despite latent changes to the dynamics of the environment with which the agent is interacting at any given moment in time.

Our proposed method is capable of *(i)* learning a model capable of mapping trajectories collected from a given environment (with some type of unknown/unobservable dynamics) to a *representation* of said environment; *(ii)* using the estimated representation of the current environment to augment the agent's state, thus allowing the agent to learn a single policy that generalizes across many different tasks; and *(iii) actively training* the above-mentioned models. In other words, we introduce a technique by which the agent can autonomously decide (in simulation) with which types of environments/tasks/dynamics it wishes to interact to more rapidly learn a generalizable policy. After introducing the formalism that underlies our novel method, we evaluate its behavior and performance when tasked with learning a single policy that generalizes across a family of continuous control tasks.

Keywords: Reinforcement Learning. Generalizable Policies. Active Learning.

Seleção Ativa de Tarefas de Treinamento Contínuas para Construção de Políticas Generalizáveis

RESUMO

Um dos principais desafios de se utilizar técnicas de aprendizado por reforço (RL) em problemas de robótica diz respeito ao grande número de interações entre o robô e o seu ambiente, necessárias para que um comportamento ótimo possa ser aprendido. Uma alternativa à ideia de treinar o robô diretamente no mundo real é treiná-lo, primeiro, em um simulador. Infelizmente, para que isso resulte em comportamentos eficazes, o projetista do sistema precisa garantir que todas as características relevantes do ambiente sejam modeladas de forma precisa no simulador. Isso é, via de regra, difícil de garantir. Como resultado, o robô pode ter que—ao ser posto em uso no mundo real—interagir com ambientes cuja dinâmica difere daquelas vivenciadas durante o processo de treinamento em simulação. Isso tipicamente faz com que políticas que eram ótimas em simulações tenham performance sub-ótima quando executas em um ambiente real.

Nesse trabalho, nós propomos uma técnica para treinar agentes de RL que sejam robustos a problemas desse tipo—i.e., problemas nos quais um agente precisa interagir com diferentes tipos de ambientes, e nos quais o agente não é capaz de diretamente inferir ou estimar com qual ambiente ele está atualmente interagindo. Isso corresponde a um *setting* no qual o estado do agente (o qual armazena toda informação coletada por seus sensores) não é completo ou expressivo suficiente a fim de caracterizar todas as propriedades do ambiente (ou tarefa) sendo observada atualmente pelo agente. Tendo isso em vista, nosso objetivo, em particular, é desenvolver um novo algoritmo de treinamento que permita a construção de políticas de controle que sejam robustas mesmo na presença de mudanças não-observáveis na dinâmica do ambiente.

Nosso método é capaz de *(i)* aprender um modelo capaz de mapear trajetórias colectadas em um ambiente (com algum tipo de dinâmica desconhecida e não-observável) para uma *representação* de tal ambiente; *(ii)* utilizar a representação estimada do ambiente a fim de estender o estado do agente, dessa forma permitindo com que ele aprenda uma política única capaz de generalizar para vários tipos de tarefas; e *(iii) treinar de forma ativa* os modelos mencionados acima. Em outras palavras, isso implica em uma nova técnica através da qual o agente pode autonomamente decidir (em simulação) com quais tipos de ambientes/tarefas/tipos de dinâmica deseja interagir, a fim de mais rapidamente aprender

uma política generalizável. Após introduzirmos o formalismo sob o qual nosso método é construído, iremos avaliar o seu comportamento e performance quando utilizado em um problema no qual o agente precisa aprender uma política generalizável única, capaz de lidar com uma família de tarefas de controle contínuas—cada qual com um tipo diferente de dinâmica.

Palavras-chave: Aprendizado por Reforço. Políticas Generalizáveis. Aprendizado Ativo..

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

RL	Reinforcement Learning
MDP	Markov Decision Process
GP	Gaussian Process
GPR	Gaussian Process Regression
EPI	Environment Probing Interaction

CONTENTS

1 INTRODUCTION

Among the many current applications of reinforcement learning (RL), one of the most popular and promising ones is robotics control. A key challenge in this setting is the large number of interactions between the agent/robot and its environment that are necessary for an optimal behavior (or *policy*) to be identified. This often makes the cost of training physical robots prohibitive. Furthermore, executing a suboptimal policy (which may be evaluated by an RL algorithm during training) on a physical agent may result in exploratory behaviors that could jeopardize the robot's integrity—causing, e.g., collisions with objects or wear-and-tear of expensive parts.

An alternative to training a robot directly in the real world—and motivated by the desire to avoid risks such as the ones described above—is to execute the training process in a *simulator*. A key challenge here, however, is that for an RL agent to be successfully deployed in real-life, one needs to guarantee that the characteristics of the environment where the robot will operate are correctly and accurately modeled by the simulator. This may be difficult to ensure. Consider, for instance, that many latent/unobserved elements of the real-world, such as the friction floor, might be hard to be accurately estimated and modeled in a simulator. As a result, the robot may have to—when deployed in real-life—interact with an environment whose dynamics are different than those experienced while it was trained in simulation. This typically results in policies that were optimal in simulation to perform poorly in the real world.

In this work we investigate how to design learning agents that are robust to settings such as this—i.e., settings where the agent may have to interact with different types of environments, and where it is not capable of directly identifying with which environment it is interacting. This corresponds to the setting where the agent's *state* (which stores all information collected by its sensors) might not be sufficiently complete or powerful to characterize all properties of the environment (or task) being tackled by the agent. By doing so, even if the robot does not have, say, a sensor to identify the friction floor, it should nonetheless be capable of adapting its behavior/policy accordingly, as it moves from a room with a low friction coefficient to a room with a high friction coefficient. In other words, we wish to design novel training algorithms that result in control policies that are robust despite latent changes to the dynamics of the environment with which the agent is interacting at any given moment in time.

Existing techniques address this problem in different ways. The problem described

above is related to the setting of *non-stationary RL*, where the dynamics of the environment may change (from the point of view of the agent) due to unobserved changes to properties of that environment. These methods typically learn a library of policies—one per "type" of environment dynamics. In this work, by constrast, we propose:

1. To learn a model capable of mapping trajectories collected from a given environment (with some type of unknown/unobservable dynamics) to a *representation* of said environment;
2. To use the estimated representation of the current environment with which the agent is interacting in order to augment its state. This effectively results in a process that *learns* to identify/estimate the current environment/task the agent is interacting with, and that *uses* this learned representation to inform the agent about how to adapt its behavior, so that it is more effective in that particular environment;
3. And, importantly, to perform the above training process in an *active* way. In particular, we will introduce a novel *actively learning mechanism* by which the agent can autonomously decide (in simulation) with which types of environments/tasks/dynamics it wishes to interact, in order to more rapidly learn a policy that can generalize to novel and different environments or tasks. Another contribution is that our method, unlike existing similar ones (ZHOU; PINTO; GUPTA, 2019) allows the agent to select training tasks from a continuous family of tasks; some of the previous techniques were only capable of dealing with a finite, discrete number of possible variations of an environment.

Effectively, the above components can be combined in a way that results in a method by which the agent learns a robust policy that generalizes across many types of tasks and environments. First, during a training phase, the agent *actively selects* with which tasks or settings it wants to interact, in order to more rapidly acquire a model capable of constructing/estimating a *representation* of a given environment, with some unknown/unobservable type of dynamics. Once this model is trained, the agent can later—when deployed on a series of unknown environments—estimate with which environment it is currently interacting, and augment its state with such information. This effectively allows the agent to learn a policy over an augmented version of the state space such that—by construction—the policy can determine appropriate behaviors, for different tasks, on-the-fly.

Previous works that address this problem—the problem of learning policies that

generalize to different environments, with different dynamics—have generally been studied within the context of Sim2Real techniques. We discuss these algorithms in more detail in Section 3. We can summarize the main differences between our proposed method and the existing techniques as follows: *(i)* our method does not require explicit access to an identifier characterizing the current task with which the agent is interacting; *(ii)* our method does not try to *adjust* a policy so that it can be fine-tuned to work on a novel problem; instead, it learns a single policy that generalizes across many tasks; and *(iii)* our method allows the agent to actively decide on which tasks or settings it wishes to train, in order to more rapidly learn a policy that generalizes over many different problems (i.e., environments with different dynamics).

This work is organized as follows. In Chapter 2, we discuss the main mathematical frameworks upon which our method will be build. In Chapter 3, we discuss existing related work and constrast them with our method. In Chapter 4, we introduce our proposed approach for tackling this challenging learning setting. In Chapter 5, we present experiments that allow us to evaluate the behavior and performance of our method when tasked with learning a single policy that generalizes across a family of continuous control tasks. Finally, in Chapter 6 we present our conclusions and discuss future work.

2 BACKGROUND

In this chapter, we discuss the two main learning frameworks that will be necessary to design our novel algorithm: Reinforcement Learning, and Gaussian Process Regression.

2.1 Reinforcement Learning

Reinforcement Learning (RL) is a framework for learning how to act in complex environments while maximizing a reward signal (SUTTON; BARTO, 2018). An RL agent interacts with its environment in discrete time steps. At each time, it observes the current state of the environment and selects an action. After executing an action, the agent observes the new state of the environment and receives a reward signal. RL problems are typically modeled as Markov Decision Processes (MDP). An MDP is a tuple (S, A, R, T) , where S is a set of environment states; A is a set of actions that the agent may choose to execute; $R : S \times A \to R$ is the expected reward function, mapping a state and an action to a scalar reward signal, and $T : S \times A \times S \to [0, 1]$ is a transition function specifying the probability of the agent transitioning to a state after taking an action in a given state.

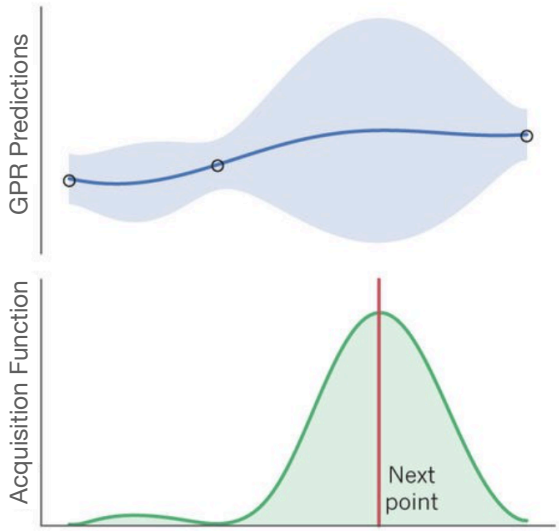
The behavior of an agent is encoded by a policy $\pi : S \times A \to [0, 1]$, that indicates which action it should take in each state. The goal of an RL agent is to learn a policy that allows it to accumulate as much reward as possible. In other words, solving an MDP consists of finding an optimal policy π^* that maximizes the agent's expected return, given by the expression $\sum_{i=0}^{\infty} \gamma^i r_{t+i}$, where r_t is the reward received at time t and γ is a discount factor, indicating the extent to which the agent prefers immediate over delayed rewards.

2.2 Gaussian Processes

Different methods have been proposed in the literature in the context of performing regression—within the broader problem of supervised learning. In this work, we emphasize one particular regression algorithm known as *Gaussian Process Regression* (GPR). This technique, which we will describe below, is relevant in the context of our research project for three main reasons:

1. It is a *non-parametric* regression algorithm. Unlike, e.g., neural networks or linear

Figure 2.1: An example of a Gaussian Process Regression model and a associated acquisition function. Here, the acquisition function is designed in a way that it chooses to query the unknown function being modeled (i.e., chooses to collect an additional training point) at the location in which the current GPR model has maximal uncertainty regarding its prediction.



Source: Adapted from (ZHOU; PINTO; GUPTA, 2019).

regression models, GPR does not require the user to make design decisions such as which features should be used, or how many layers or neurons a neural network should have. Instead, the entire learning process itself is *data-driven*: predictions are made based solely on observed training instances, and no model parameters (such as neural network weights) need to be estimated directly. This is relevant because it results in a domain-agnostic regression technique that requires less fine-tuning, for example, regarding the most effective neural network architecture to use in a particular application. The common downside of GPR models is that they typically do not perform well when making predictions over high-dimensional input data; this will not be the case in our work, though, as we will discuss in subsequent chapters.

2. It results not only in a regression model capable of producing mean/expected predictions, y, for a given input, x, but it also estimates the *uncertainty* of the model regarding such predictions. This is relevant in cases where the regression algorithm is being used in an *active learning* setting, such as the one being tackled in this work (for more details, see Section 4). Consider, for instance, that knowledge about a model's uncertainty regarding different types of inputs may be used to guide

the agent's decisions regarding which new training instances it should sample next, in order to actively improve the model's accuracy. The agent could, e.g., choose to query the unknown function being modeled by GPR (f), at a particular input location (x), thus observing its associated output ($y = f(x)$). Here, x could be selected, for example, because it corresponds to the input location for which the current GPR regression model is maximally uncertain regarding its prediction.

3. Related to the point above, GPR models can be combined with *acquisition functions*. Acquisition functions denote different criteria that a Gaussian Processes algorithm may use when determining the most promising point, x, to sample next, in order to improve (for example) the accuracy of the resulting regression model.

As an example, consider Figure 2.1. In the uppermost graph of this figure, we see one possible GPR model trained over three data points. The blue curve shows the mean prediction made by the model for different inputs, and the shaded blue region surrounding the mean prediction indicates the model's uncertainty regarding its predictions at those locations. In the graph immediately below, we show the values produced by an acquisition function used to determine which points the agent should sample next in order to improve the accuracy to the GPR model. Here, the acquisition function is designed in a way that it chooses to query the unknown function being modeled by GPR at the input location in which the current GPR model has maximal uncertainty about its prediction. This location is indicated as a vertical red line in the bottom graph, and it is aligned with the input location of the regression model where the shaded blue region indicates highest uncertainty. After collecting such new training instance, at the selected location, the data point is added to the training set of the GPR model, the model is adjusted, and then the acquisition function is queried once again to determine the next most promising point to sample. Notice, therefore, that combining GPR and acquisition functions naturally results in a Bayesian regression algorithm (Bayesian in the sense that it produces uncertainty estimates) that can be *actively trained*, online, thus allowing an agent to autonomously construct its training set.

Mathematically, a Gaussian Process (GP) extends the idea of a Gaussian distribution—a distribution over real numbers—to the idea of a distribution over *functions*. Concretely, a GP models a distribution over the functions that may have generated the observed training points. We say that a GP is fully characterized by its prior mean predicted output, $\mu(x)$, which describes the model's predictions when little or no data is available; and a kernel function, $k(x, y)$, which quantifies how similar two given inputs (x and y) are to

the regression model; if x and y are deemed similar, the GP model will tend to produce similar outputs/predictions for such input points. Concretely, to fully specify a GP one must define a mean function μ and a positive-definite kernel k:

$$\mu(x) = \mathbb{E}[f(x)]$$

$$k(x, y) = \mathbb{E}[(f(x) - \mu(x))(f(y)) - \mu(y))^\top],$$

where f is the unknown function being modeled by the Gaussian Process regression model. A kernel specifies properties of f such as smoothness and periodicity.

Given k and a training set $D = \{(x_1, f(x_1), \ldots, (x_N, f(x_N))\}$, the Gaussian Process regression model can be constructed in a way that allows it to make *mean predictions*, $\mu(x)$, associated with any new input point, x, as well as predictions about the model's uncertainty, $\sigma^2(x)$, regarding its predictions about $f(x)$. Formally, the new GPR model will be a function whose mean predictions and associated uncertainty values are given by

$$\mu(x) = \mathbf{k}^\top (\mathbf{C_D} + \sigma^2 I)^{-1} \mathbf{y_D} \tag{2.1}$$

$$\sigma^2(x) = k(x, x) + \sigma^2 - \mathbf{k}^\top (\mathbf{C_D} + \sigma^2 I)^{-1} \mathbf{k}, \tag{2.2}$$

with $\mathbf{k} = [k(x, x_1) \ldots k(x, x_N)]^\top$, $\mathbf{C_D}$ is an $N \times N$ matrix with entries $(\mathbf{C_D})_{ij} = k(x_i, x_j)$, $\mathbf{y_D} = [f(x_1) \ldots f(x_N)]^\top$ and σ^2 is the additive noise we assume affects measurements of the outputs of f. One commonly-used kernel function is the exponential kernel, $k(x, y) = \exp\Big(-\frac{1}{2l^2}||x - y||^2 \Big)$, where l regulates the *width* of the kernel; intuitively, larger values of l result in smoother (less "spiky") regression models.

3 RELATED WORK

The issue of deploying an agent in environments different from expected during training is widely discussed in an area named Sim2Real. The problem is named Reality Gap. Since simulations are usually imperfect representations of the real world, where details and characteristics that could influence the environment's response to a performed action are disregarded. These inaccuracies can lead to failed attempts to deploy phisical robots, since their policies were trained in a different environment. Simulation errors can be related to an incorrect representation of the agent's observations or in the simulation of collisions and frictional dynamics. Works tackling both issues are discussed in the sections that follow.

3.1 Imperfect simulation of observations

Mismatches in simulated observation frequently happen when the state includes visual information. Since the computational and the design costs of high fidelity simulations can be prohibitive, simplified and imperfect simulators are often used for pixel-to-torque tasks, where controller's input is an image.

In this scenario, Tzeng et al. (2015) proposes a supervised domain adaption approach to map objective environment states to a given imperfect simulation state. The technique uses an adversarial approach to align the distribution of both source and target environments. This class of problems, known as domain confusion, uses a domain classifier to predict the domain of a representation of a given image. At the same time, a generator tries to find a representation able to trick the discriminator into classifying both domains as being the same. This way, the learned representation is indistinguishable in the feature space, and both domains lie in the same general neighborhood in the representation space.

However, the confusion loss may align domains in the same representation space, but it does not guarantee that similar states have a close representation in the state space. To tackle this, the authors also propose a pairwise loss to match labeled images from the goal environment to the corresponding state in the simulator. Matching the representations for the same state encourages the network to disregard domain-specific features in favor of relevant features to the task. The pairs, which are weak labels, are matched automatically by selecting the closest neighbor of the sampled images.

Viereck, Saenko and Jr. (2018) also use a pairwise loss to approximate the representation of the same state of source and target environments. The authors use siamese networks to predict the offset of a robotic gripper given the chosen action for the state of a task where the claw must place a cap on its matching container. The gripper has a depth sensor, and one of the networks receives the depth image of the target environment while the other network's input is the simulated depth image. The pairwise loss is calculated from the intermediate representation extracted from the last pooling layer of both convolutional networks. Noise (different objects cluttered together with the task object) and salt and pepper noise (sparse black and white pixels randomly distributed over the image), can be added to the scene to increase the robustness of the final model. The disadvantage of this strategy is that the labels need manual pairing and each paired image requires the measurement of the gripper's offset in the real-world setting.

Bonatti et al. (2019) also deal with simplified simulations for visual policies. Their approach to mapping states to a common latent representation is closer to our work, even though the embedded representation is reconstructed to be applied to visual mismatches instead of dynamics variations. Bonatti et al. (2019) use an autoencoder to compress the image into a vector, which deliberately causes loss of information. The goal is to maintain only the necessary information, discarding details of distracting elements in the background. This shrunk representation then passes through the decoder to reconstruct the image with diminished noise and is finally fed to the policy.

Hansen et al. (2020) combine an encoder as a feature extractor and an forward dynamics predictor — a neural network aiming to predict the action a_t that produced the transition from s_t to s_{t+1} — to be able to calibrate the policy without having rewards available in the deployment phase. At the training phase, the authors train a task policy and the forward dynamics predictor, both receiving as input the image representation obtained by the feature extractor. The policy is kept fixed at deployment, and the forward dynamics predictor is updated over the goal environment, backpropagating the updates to the encoder, thus calibrating the representation to match the new environment particularities.

3.2 Imperfect dynamics simulation

Approximation of physics rules, simplified collision models, and other implementation choices that reduce the computational cost of simulations incur in inaccuracies regarding the dynamics of the real world. Several research fronts dedicate efforts to miti-

gate quality drops on the deployment of policies learned in such simulators. A commonly investigated approach focuses on adjusting the learned policy when the agent comes into contact with the goal environment. Silver et al. (2018) augments the initial policy learning with a residual function that maps a state to a function and can be learned like an MDP over samples of the target environment, needing fewer samples than a policy trained from scratch. The final policy sums the initial policy with the residual function to obtain an action calibrated to suit the new environment.

Similarly, Higuera, Meger and Dudek (2017) uses an adjustment policy to correct the action chosen by the original policy. Unlike the residual function from Silver et al. (2018), the adjustment policy receives the original action a_{source} as input alongside the state s_{target}. The objective of the adjustment policy is to replicate in the target environment trajectories from the source domain, which is achieved by learning an additive change to a_{source}. Christiano et al. (2016) maps a_{source} to a_{target} with the help of forward dynamics models. Instead of a policy that directly outputs an action, a history of observations in the target domain is passed to the source policy, which chooses a corresponding action, a_{source}. a_{source} is passed to a model of the forward dynamics of the target domain, obtaining an estimated next state for the target environment. Having both the next step estimate and the history of observations from the target, the forward dynamics model trained over target predicts the final action, a_{target}.

An interesting alternative approach, presented in Cutler, Walsh and How (2015) takes advantage of an array of available simulators, with varying degrees of accuracy, to reduce the number of samples needed from slower, higher-fidelity environments. The authors leverage the aspects of an environment that can be captured even by less complex simulations, using the valuable information the agent can learn at a lower cost as a base for the final policy. During training, the agent starts learning at the lowest fidelity simulator and can choose whether execute the next step on a higher fidelity simulation or maintain the training at the current simulator. Bidirectionally, it is also possible to go back a level and learn new behaviors on simpler settings. As the agent incrementally explores the simulations, it aims to use the highest-fidelity environments only to access states that other simulators were unable to model.

As opposed to changing the policy to adapt to the target environment, some authors argue that bringing the simulation closer to the target can also lead to a policy that works as expected when deployed on the final environment. A requirement on that approach is to have parametrized simulators that can be changed to emulate the goal do-

mains' physics better. Hanna and Stone (2017) proposes an iterative approach to do the simulation grounding – as the process of tweaking simulation parameters is named. A policy trained on the simulator is used to collect trajectories with the physical robot. The algorithm tries to find the set of simulation parameters that minimize the distance of the collected trajectory to the same sequence of actions executed from the same initial state of the simulator. The best set of parameters is applied to the simulator to train a set of candidate policies evaluated on the physical robot. The policy with the best performance is chosen and used to collect trajectories to repeat the whole process until the stop criterion is reached. After simulation grounding, the authors apply policy adjustment similarly as Higuera, Meger and Dudek (2017).

Allevato et al. (2019) presents a one-shot version of simulation grounding that is trained over pairs of observations of environments with known distinctive parameters. The neural network predicts the difference of the two groups of parameters θ_1 and θ_2. For the task policy, a single observation is collected from the target and it is used to obtain $\Delta\theta$ of the matching source observation; this process requires that the simulation allows reseting to a specific state. Δ is added to the original simulation parameters, and the agent learns the final policy in the grounded simulation.

In Lopez-Guevara et al. (2017), Δ is given by the difference in the predicted liquid spillage in a task where the agent must fill a container with water. The task is trained in a fast simulator (frequently used in videogames) that approximates fluid dynamics by moving groups of particles according to the position of their neighbors. Simulation parameters grounded by Δ aim to compensate for the mismatch of the fluid dynamics model. Golemo et al. (2018) also learns the difference of matching states of each environment, but Δ represents the immediate difference of the observations, which, at each state, is predicted again and applied simulation's state at training time.

Another branch of research related to this work uses the idea that, instead of learning a unique policy to solve different environments, a policy should receive enough information from the agent to perform different actions when it finds itself in the same state of distinct environments.

Yu et al. (2017) uses system identification (SysID) to augment the policy input. SysID is a technique used to identify the dynamics rules of a system using data collected on it. Yu et al. (2017) learns to predict the explicit dynamics parameters of an environment by using sequences of state-action pairs (trajectories) collected on simulators with known parameters. The task policy receives the parameters inferred by the SysID mod-

ule and trains on various environments, pursuing a policy that generalizes accordingly to the dynamics parameters. The system identification process continuously executes over the most recent history of visited states and actions, inferring the dynamics parameters at each step.

To learn a generalizable policy, Liang, Saxena and Kroemer (2020) trains an exploration policy exclusively to collect trajectories for system identification. As in Yu et al. (2017), the task policy receives the explicit dynamics parameters as an extra input.

However, explicitly inferring the dynamics parameters requires all relevant parameters to be identified and modeled as outputs of the system identification (PENG et al., 2018). Furthermore, as happens with simulators, the explicit set of parameters may not capture all the elements that influence real-world dynamics. As an alternative, the explicit system identification can be replaced by an implicit representation of the domain dynamics.

Peng et al. (2018) implements the policy as a recurrent neural network and augments the input with the last action taken. This way, they capture the sequence of states and actions and implicitly learn to differentiate environments by their trajectories and choose the action taking the dynamics into account.

In order to have more control over the representation of the dynamics, Zhou, Pinto and Gupta (2019) augments the generalizable policy with an embedding of trajectories rather than leaving the latent representation learning for the policy network. Alongside the embedding, an environment probing interaction policy is dedicated to exploring the environment and collecting trajectories, similar to Liang, Saxena and Kroemer (2020). The authors defend that collecting trajectories with a random policy may not expose nuances of the environment, and using the task policy for probing is not ideal as a good task policy may not be a helpful policy to understand the environment. This work is strongly based on Zhou, Pinto and Gupta (2019) and builds the methodology from the original EPI.

3.3 Summary

We can summarize the main ways in which our proposed method differs with respect to existing algorithms as follows:

1. Some existing algorithms assume, as a black box, that the agent has direct access to

information precisely characterizing the current environment/task with which the agent is interacting; we, by contrast, assume that the agent needs to *learn* such representations;

2. Some existing techniques try to accelerate the process of adjusting/improving a policy, so that it can be fine-tuned to work well on a novel environment; we, by contrast, train a *single* policy that generalizes to different environments or tasks. When a new environment is encountered (or a previously-visited environment is revisited), the policy is capable of deploying effective behaviors directly, without requiring a re-learning process to take place;
3. Some existing techniques make use of SysID algorithms, which try to estimate previously-unknown values of some parameters of the environment (such as the latent friction with the ground). We, by contrast, do not assume that the agent knows which parameters are unobservable, in the first place. Instead, our method learns *abstract* representations of different environments based solely on trajectories collected by these environment with different dynamics;
4. Other techniques *do* try to learn abstract representations of the latent environment/dynamics, but do so by collecting training data in a way that is completely random. That is, the agent does not control with which types of environments or tasks it will interact while trying to learn a generalizable policy. We, by contrast, allow for an active learning strategy by which the agent can *actively select* the tasks on which it wants to train in order to more rapidly learn a policy that is effective over many different tasks or settings.
5. Our method, unlike existing similar techniques (ZHOU; PINTO; GUPTA, 2019), allows the agent to select training tasks from a continuous family of tasks. Some of the previous, such as the EPI algorithm, on which our work is based, were only capable of dealing with a finite, discrete number of possible variations of an environment.

4 PROPOSED APPROACH

In this chapter, we introduce our proposed method—Active Learning via Environment Probing Interaction Policies, or AL-EPI. This method is built upon the EPI technique, discussed in Chapter 2 (ZHOU; PINTO; GUPTA, 2019). AL-EPI is a method for constructing generalizable policies via active selection of training tasks. AL-EPI is divided into two phases: *(i)* training a *probing policy*, responsible for collecting training data (trajectories) to more rapidly learn a network capable of mapping such trajectories to a representation of the environment with which the agent is interacting. ***This corresponds to a process that learns to characterize the latent environment with which the agent is interacting***; and *(ii)* deploying this model to estimate the current environment/task being tackled by the agent, and use such information to augment the agent's state; this effectively allows the agent to learn a single *task policy*—a policy that generalizes across many different tasks or environments with different dynamics. ***This corresponds to a process that learns a generalizable policy via augmented states.***

4.1 Learning to Characterize the Latent Environment

During the first phase of AL-EPI, the agent interacts with its environment by following a given (fixed) *probing policy*. This policy is executed until the agent collects trajectories of up to 10 steps, which will be used to try to characterize the current unknown environment with which the agent is interacting. More precisely, these trajectories are given as input to an *embedding network*: a network that maps trajectories of states, actions, and next states, to a latent representation of the environment's dynamics. Notice that because we assume that the agent does *not* know with which environment it is interacting, this embedding network cannot be trained in a supervised manner—the agent does not know the ground-truth environment identifier that should be associated with a given trajectory. In order to address this challenge, AL-EPI trains the embedding network using a proxy objective function—one that assesses the *quality* of the learned embeddings.

Here, we define the *quality* of learned embeddings as a measure of how effective they are in characterizing the underlying (unknown, unobservable) dynamics of the current environment. More precisely, we quantify:

1. whether a learned transition model (e.g., a forward dynamics model) is more ac-

curate when estimating the next state (s') based only on the current state (s) and current action (a); or

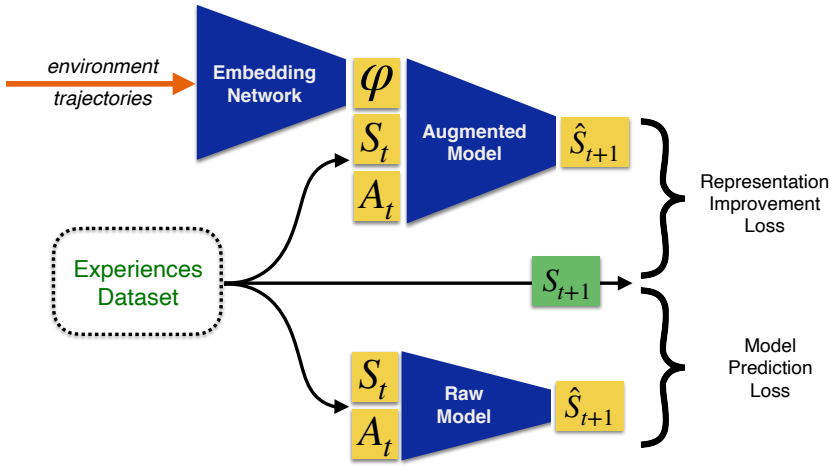
2. whether a learned transition model is more accurate when estimating the next state (s') based on the current action (a) and based on augmented state information; i.e., s concatenated with the agent's estimated representation of the environment, as given by the embeddding network.

If the embedding network is capable of producing effective representations of the environment, then they should help the agent to determine more accurately how the environment's state will evolve over time—after all, the learned representation of the environment is built precisely in order to summarize all latent/non-observable properties of the environment, which might be required to identify the possible next states. In order to quantify the quality of a learned embedding network, thus, we learn forward dynamics models using two types of models: one that only uses information about states and actions as input, and one that uses information about actions and about the the augmented state, as described above. The first model, f, is a traditional forward dynamics predictor: the input is an action a_t and a state s_t, and $f(s_t, a_t)$ is used to predict the next state, s_{t+1}. The second model, f_{epi} , receives as input the action a_t and the state, s_t, augmented with an embedding produced by the embedding network—where the embedding network is given as input a trajectory sampled from the current environment. The actions and states used to train f and f_{epi} are sampled from a dataset of transitions, $D_{transitions}$, which is constructed offline, before training, by collecting annotated transition tuples from different environments.

To compute the performance of a given embedding network, then, both prediction models are given as input a same state-action pair, with f_{epi} also receiving the trajectory embedding information produced by the embedding network. Then, the next state predicted by each model is compared with the actual next state. This allows us to compute the Mean Squared Error of each model—this will be the loss function we wish to minimize. Finally, given the MSE loss of each model, we *define* the quality of an embedding by quantifying how much better the performance of f_{epi} (the predictor that received the embedding information) is compared to f (the standard forward dynamics predictor that only analyzes states and actions). The difference between the loss of the two predictors indicates how helpful the embedding was in fully characterizing the dynamics of the (unknown) environment. The difference between these losses—which can be seen as a type of *information gain* resulting from the use of trajectory embeddings to estimate an ac-

curate forward model—is then use to train the embedding network[1]. This is done using standard backpropagation. The overall process is depicted in Figure 4.1.

Figure 4.1: Diagram depicting the process of training the embedding network.



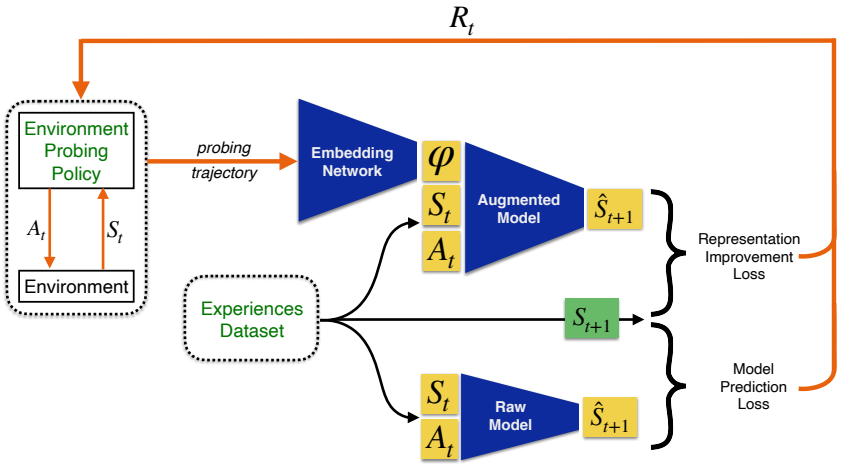
Source: Adapted from (ZHOU; PINTO; GUPTA, 2019)

Notice, importantly, that for the embedding network to produce meaningful embeddings, it needs to be given trajectories that are representative of the current environment being tackled by the agent. These trajectories, as previously mentioned, are collected by running a *probing policy*. We propose *training* this probing policy, so that it can actively decide which actions to taken in a given environment, in order to collect trajectories of states and actions that are more useful to characterize the current environment's dynamics. In order to train this policy, we define a reward function that corresponds to the original reward function of the current task, but where we add the information gain mentioned above—the quality of the current embeddings—as a type of reward bonus. In other words: if the probing policy is producing experiences/trajectories that help the embedding network to better characterize the environment, then the probing policy's behavior is rewarded more strongly. This results, effectively, in an adaptive probing policy that better determines how the agent should interact with a given novel, unknown environment, in order to more rapidly characterize its dynamics. The training process—which adapts both the embedding network and the probing policy—continues until the probing policy and the embedding encoder converge. The overall process of using the predic-

[1]To ensure that the embeddings associated with different environments are sufficiently spaced apart—i.e., that the trajectories from environments with different dynamics are mapped to sufficiently different embeddings, the embedding network has a regularization-separation loss. This loss penalizes the network whenever it produces similar embeddings for trajectories arising from different tasks/environments.

tion model losses to augment the reward of the probing policy, thus resulting in a more effective/informative trajectory-collecting mechanism, is depicted in Figure 4.2.

Figure 4.2: Diagram depicting the process of training the probing policy.



Source: Adapted from (ZHOU; PINTO; GUPTA, 2019)

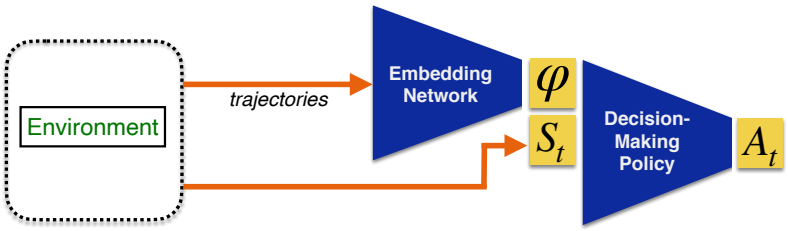
4.2 Actively Learning a Generalizable Policy via Augmented States

In the second phase of AL-EPI, a *task policy* is trained—a single policy that generalizes across many tasks. The forward dynamics predictors are discarded, and we keep only the embedding network and the learned probing policy. In order to learn a task policy, the agent will select a task/environment on which to train; execute the probing policy to collect a trajectory from the environment; provide that trajectory as input to the embedding network; and use this embedding to augment the state provided to the task policy, so that the agent can make decisions based not only on usual state information, but also based on its learned representation of the latent environment[2]. This process is depicted in Figure 4.3.

In order to reduce the number of interactions with the environment that are needed

[2]Notice that this same process is used after the task policy is learned in simulation, and when it gets deployed in the real world. In that case, when the task policy is deployed in order to solve a novel, unknown task, the agent—as usual—executes its probing policy to collect a sample trajectory from the unknown environment; the resulting trajectory is passed to the embedding network to infer the environment's representation; and this representation is used to augment the state information given to the previously-learned task policy. Notice, then, that during deployment time, the only additional step that is required to run generalizable policy it to collect a small trajectory from the new environment, in order to estimate its representation as given by the embedding network.

Figure 4.3: Diagram depicting the use of a trained embedding network to compute outputs (representations/estimates of the current environment/task being tackled by the agent) in order to augment the state given to the generalizable policy over tasks.



Source: Adapted from (ZHOU; PINTO; GUPTA, 2019)

to learn the probing policy, AL-EPI modifies the training to allow the agent to make *informed decisions* about which environment to explore next. We hypothesize that actively deciding on which tasks to train at each moment in time, to more rapidly learn a policy that generalizes well, is a more effective strategy than randomly selecting training tasks—as was done in the original EPI method (ZHOU; PINTO; GUPTA, 2019).

We model this as an active learning problem. Here, the agent first constructs a model of expected performance of the probing policy across different tasks, along with its uncertainty about those predictions. Then, it chooses to train next on the task with the lowest predicted performance. This is determined by computing a lower confidence bound on the expected performance of the probing policy over the possible tasks. Intuitively, this results in a mechanism that incentivizes the agent to practice on tasks for which its task policy—the policy that generalizes across different problems—does not yet perform well.

In order to construct a model of expected performance of the probing policy across different tasks, we use Gaussian Process Regression (GPR; see Section 2 for more details). The inputs to GPR correspond to the parameters of a given candidate task on which the agent may choose to practice. The output of the GPR model corresponds to the prediction of the mean performance of the probing policy on each task, along with the uncertainty associated with those predictions. Recall that the performance of the probing policy on a given problem is defined as the original reward produced by the environment (which we refer to as *task reward*), plus a bonus reward reflecting how useful the trajectories collected by that policy were in terms of properly characterizing the unknown environment dynamics. In other words: the model above can be used to quantify how "useful" each candidate task is for the agent, in terms of the information it provides for improving the agent's representation of the dynamics of the different tasks it may face. By actively

selecting tasks this way, the agent effectively learns better representations of different environments, and, thus, better augmented state representations to provide to the task policy. When provided with this more complete information about the state and the current environment, the task policy becomes more capable of selecting an appropriate behaviors that take into account not only the current state of the agent, but also on the agent's learned representation of the current (unknown) environment it is facing. This helps the agent to more rapidly acquire a policy that is effective in generalizing across diferent tasks, or environments with different dynamics.

As previously mentioned, we construct the model of expected performance of the probing policy across different tasks using Gaussian Process Regression. The uncertainty estimates provided by GPR are useful because they indicate regions of the task space that have not yet been properly explored by the agent. When a new task is selected (and the agent trains on that task), its uncertainty about the probing policy's performance on that environment—and on surrounding environment, with similar dynamics–is reduced. This allows the agent to dynamically explore *regions* of the task space based on how informative they are to the construction of an effective input representation for the generalizable policy.

Notice that the original EPI method was only capable of operating over discrete task spaces. Since GPR performs regression over functions with continuous inputs (i.e., tasks described by continuous parameters, in the context of our work), our method can naturally deal with the actively learning problem even when the agent might face an infinite number of possible environments[3]. This—combined with the active task selection mechanisitself—is an important contribution of our method over the originally-proposed EPI technique.

By constructing AL-EPI in a way that supports the active selection of tasks described by continuous parameters, applying GPR becomes possible. At each iteration, AL-EPI selects a task to practice (more details on the acquisition function used in this step are provided later). It then collects sample trajectories from this task, by execut-

[3]In order to adapt the previously-mentioned separation loss to the case of infinite tasks, a few adaptations are needed. Informally, we discretize task parameters and assign tasks to discrete bins in order to ensure proper embedding separation. Notice that this discretization process is only needed to regularize the training process of the embedding network; it does *not* affect the capability of the agent of searching over an infinite space of candidate tasks to select the most promissing one on which to train next. Other minor changes to the way EPI works are needed to support the new setting where an infinite number of tasks exist. For instance, in the original EPI, a transitions dataset was collected offline, prior to the training process. This is no longer possible now since the agent is actively selecting tasks on-the-fly—and thus collecting trajectories from those tasks online, on-demand.

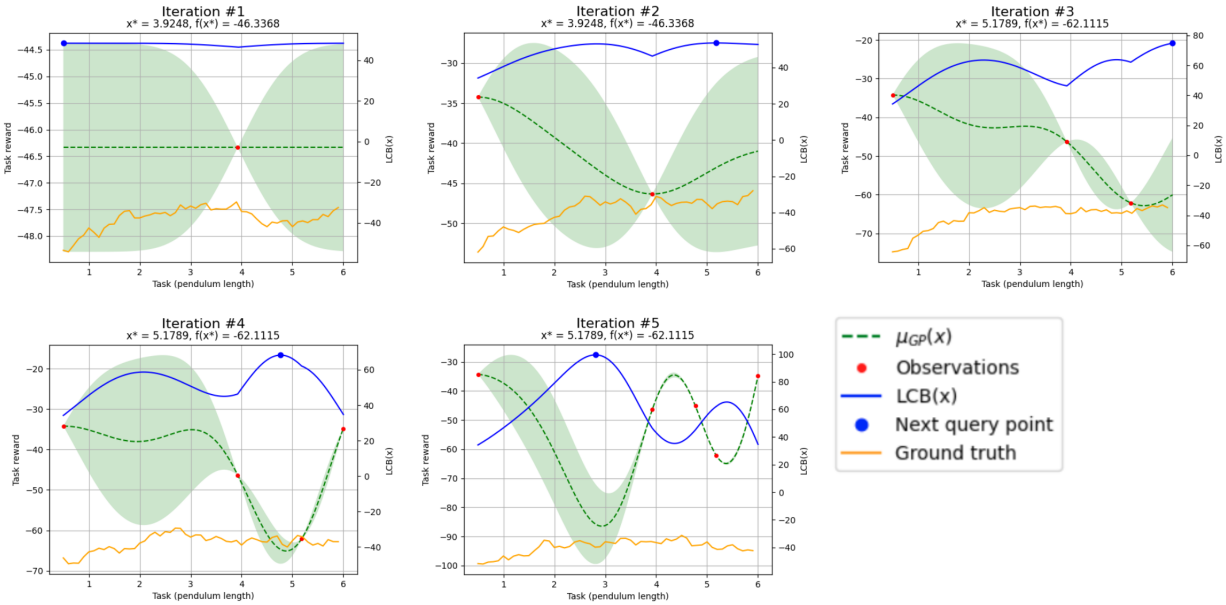
ing the probing policy; the probing policy is updated based on the augmented reward function described previously. Recall that the objective of GPR, in this case, it to model the performance of the probing policy across different tasks. Notice, however, that as the agent's generalizable policy improves, the performance of the probing policy also changes—which means that the GPR model will need to track/model a non-stationary performance function. Different strategies have been proposed to adapt GPR to settings where the function being modeled is non-stationary. In this work, we simply apply a sliding window over the training set used by the Gaussian Process. Recent performance data about the probing policy is maintained, and all observations older than a given threshold are discarded in a FIFO order.

Two final design decisions need to be made when using GPR in our setting. The first decision is regarding which kernel function to use. This is a key decision because the kernel function encodes the mechanism that quantifies how similar different tasks/environments are based on their parameters. The GPR model will predict that the probing policy's performance will be similar in case environments are deemed to be similar—consider, e.g., a robot learning to balance under two similar values of gravity. The GP kernel of choice for the domain we explore in Chapter 5 is the RBF kernel. This is a Gaussian kernel that has been empirically shown to be appropriate over a wide range of practical problems. It has hyper-parameters that allow a designer to control, e.g., the smoothness of the function being estimated.

The second design decision that needs to be made when using GPR in our setting relates to which *acquisition function* to use when actively sampling tasks on which to practice. This choice will inform the sampling and exploration strategy employed by the agent, while learning a policy that generalizes across different tasks. In this work, we used a Lower Confidence Bound acquisition function. This function analyzes the mean performance predicted by the GPR model for different tasks, as well as its uncertainty about said predictions, and identifies the task with the lowest confidence bound on its performance. Intuitively, this results in a mechanism that incentivizes the agent to practice on tasks for which its task policy—the policy that generalizes across different problems—does not yet perform well. Fig. 4.4 shows the acquisition function, alongside the trained Gaussian Process Regression model, evolving over time as the agent actively selects tasks to train on. Here, the dashed green line represents the mean predicted performance of the probing policy across different tasks. The shaded green region represents the uncertainty of the GPR model about those predictions. Red points indicate tasks on which the agent

has already practiced, and whose measured performances are used as training set for the Gaussian Process. The Lower Confidence Bound acquisition function is shown as the blue curve. Notice that the most promising point, at each iteration, as identified by the acquisition function, corresponds to the task practiced by the agent at the subsequent iteration of the algorithm. This demonstrates how AL-EPI uses the GPR model (and its uncertainty estimates) to identify the most relevant tasks to practice on—thus resulting in the above mentioned active learning process for collecting data to train a generalizable policy.

Figure 4.4: A depiction of the GPR model being updated as the agent trains on different tasks, as well as the corresponding acquisition function at each moment in time.



Source: The authors

Notice that the active learning process described above allows the agent to collect data about *one* task/environment at a time—always the task that seems the most promising according to the GPR acquisition function. If we were to sample multiple tasks at once (i.e., by selecting the top N most promising tasks, and to practice them all in parallel), the resulting behavior would approximate—for large values of N—a near-random selection of training tasks. This would accelerate learning from a computational point of view, but would hinder our objective of improving sampling efficiency in terms of the number of interactions between the agent and the environment.

In this chapter we have described how AL-EPI can *(i)* learn a model capable of

mapping trajectories collected from a given environment (with some type of unknown/unobservable dynamics) to a *representation* of said environment; *(ii)* use the estimated representation of the current environment to augment the agent's state, thus allowing the agent to learn a single policy that generalizes across many different tasks; and *(iii) actively train* the above-mentioned models. In other words, we have introduce a technique by which the agent can autonomously decide (in simulation) with which types of environments/tasks/dynamics it wishes to interact to more rapidly learn a generalizable policy. In the next chapter, we evaluate its behavior and performance when tasked with learning a single policy that generalizes across a family of continuous control tasks.

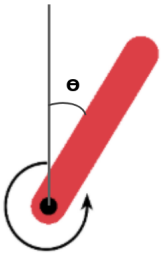
5 EXPERIMENTS

This chapter discusses the experiments performed to compare and understand the differences between the behavior of our proposed method (AL-EPI) and the original, discretized, passive-learner version of EPI.

5.1 Setting

All experiments were executed in the inverted pendulum environment from OpenAI Gym (BROCKMAN et al., 2016). The pendulum domain is a toy problem modeling motorized rod with one a fixed end, as shown in Figure 5.1. The goal is to swing the rod to the upright position and maintain it balanced. The rod starts in a random state/position, described by the pendulum angle and speed; actions correspond to the torques applied to the rod, and are continuous values between $[-2, +2]$.

Figure 5.1: The Pendulum domain.



Source: OpenAI Gym (BROCKMAN et al., 2016)

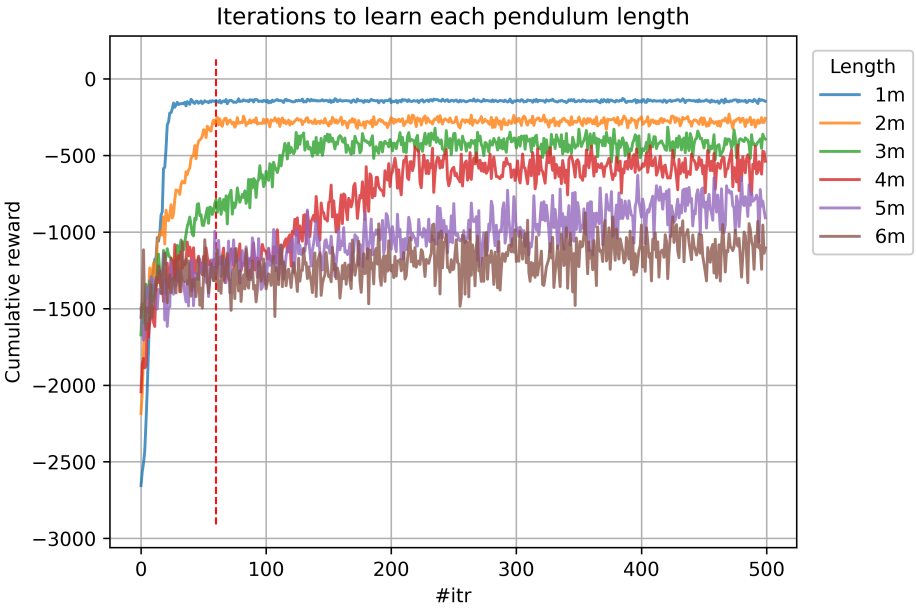
To evaluate the capability of EPI and AL-EPI to learn generalizable policies, we defined a set of related but different tasks/environments within the pendulum setting, by varying a single dynamics parameter: the length of the rod. This directly influences the environment's transition dynamics. An agent tasked with controlling a longer pendulum, e.g., may have to learn to swing back and forth a few times to gain enough momentum, prior to being able to reach the upright position. By contrast, an agent tasked with controlling a short rod may be able to continuously apply torque in the same direction and reach the upright position with a single continuous motion.

5.2 Training Generalizable Policies Under a Budget

The pendulum domain has an important property. If the agent practices, first, on a more challenging environment (say, on longer rods), its generalizable policy will most likely be capable of controlling shorter pendulums. To make the learning problem more challenging and properly emphasize the advantages of actively learning training tasks, we modify the learning setting and add a *budget*—a maximum number of training iterations during which the generalizable policy will be allowed to train on any one task. For the following experiments, we set the budget in a way that the only way for the agent to learn to control challenging pendulums is for it to deploy a carefully constructed learning curriculum, where it first practices easier versions of the problem. If the agent were to training tasks uniformly at random, by contrast, it could end up "wasting" time trying to solve complex variations of the environment, while not having had previous experience in simpler pendulum-control settings.

To calibrate the training budget, we trained the agent to learn (from scratch) to control pendulums with different lengths. By observing the resulting learning curves, we then defined the budget to be the average number of iterations required for the agent's policy to converge for the 2-meter pendulum. This ensures that if the agent were to train tasks randomly (e.g., by selecting longer pendulums at first), it would fail often, since the learning process would timeout. Thus, the agent would end up interacting many times with the environment—thus incurring risks and various costs associated with training—while not significantly improving its generalizable policy. This, as previously argued, ensures that the only way for the agent to learn to control challenging pendulums is for it to carefully and actively choose an appropriate order on which to practice tasks. In particular, the agent will only master more complex versions of the pendulum control domain if it first practices on easier versions of the problem.

Figure 5.2: Learning curves used to identify a training budget. The horizontal axis shows the number of training iterations and the vertical axis shows the cumulate reward of the learned policy. The vertical dashed red line shows the identified budget/cutoff. This is the number of iterations that allows the agent to learn effective policies for simpler pendulums (with length equal to or smaller than 2 meters), but that is insufficient for the agent to *directly* learn (from scratch) to solve more challenging varitions of this problem.

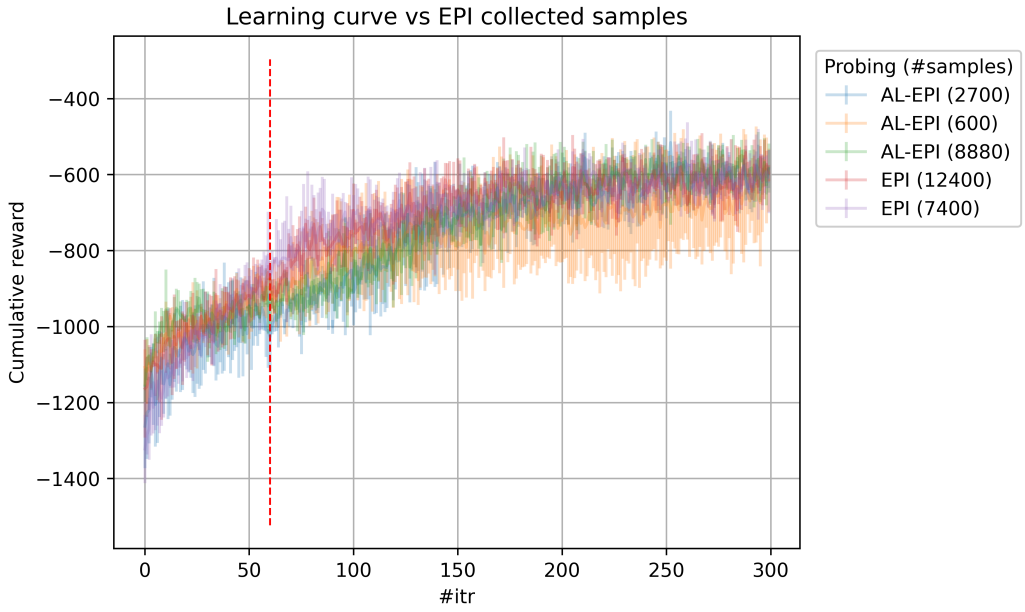


Source: The authors

After identifying a proper training budget, we proceeded to investigate how informative the learned embeddings may be as a function of the number of interactions between the agent and the environment. For this experiment, we set a series of sample thresholds as checkpoints during the training of the probing policy and the embedding network. The lowest amount of samples in figure 5.3 — 600 for AL-EPI and 7400 for EPI — is given by the first checkpoint in which each method was able to save the models.

The original EPI method and our method (AL-EPI) differ only in the training process of the probing policy and the embedding network—our method actively selects on which task to practice at each iteration. After EPI and AL-EPI learn a probing policy and an embedding network, both methods discard the learned transition dynamics predictors (f and f_{epi}). Both methods, then, when tasked with tackling a novel and unknown environment, first collect sample trajectories from that environment; use them along with the trained embedding network to construct/estimate a representation of the latent environment; and use that representation/embedding to augment the state vector provided as

Figure 5.3: Performance of the generalizable policy learned by EPI and AL-EPI as a function of the number of samples.



Source: The authors

input to the task policy. Figure 5.3 shows the performance of the generalizable policy learned by EPI and AL-EPI as a function of the number of samples; i.e., interactions between the agent and its environment. All learning curves overlapped unless for AL-EPI with 600 samples, which was not enough for the generalizable policy to converge as with the other embeddings. The generalizable policy trained using the embedding learned by AL-EPI with 2700 samples already converged similarly to the embeddings with more interactions with the environment.

5.3 Results

After using EPI and AL-EPI to learn a generalizable policy (i.e., a *task policy*), we evaluated their performances when used to solve tasks corresponding to pendulums whose rod lengths were drawn from three different ranges, as depicted in Table 5.1. The goal of this experiment is to evaluate whether the embeddings learned by EPI (a method that selects training tasks completely at random) and AL-EPI (our method, which actively selects training tasks) would be effective in learning a task policy that could generalize across pendulums of different difficulties.

Table 5.1: Pendulum lengths used when evaluating the generalizable policies learned by EPI and AL-EPI.

	Short	**Medium**	**Long**
Length range (meters)	0.5 to 1.3	2.5 to 3.7	4.9 to 6

We show the results of this experiment in Figure 5.4. The horizontal axis shows the number of samples (interactions between the agent and its environment), and the vertical axis shows the performance of the learned generalizable policy over pendulums of differents lenghts (*short*, *medium*, and *long*). Notice that we chose to compare the algorithms' performances as a function of the number of collected *samples*, instead of as a function of the number of *iterations*. That was done because the meaning of "iteration" is different for EPI and AL-EPI. While AL-EPI samples and explores only one selected task/environment per iteration (thus collecting a small number of trajectories from it), the original discretized EPI algorithm collects data—in parallel—from dozens of environments at each iteration. As a result, EPI requires fewer iterations to explore many different environments, but such a metric (number of iterations) masks the fact that, while exploring a large number of environments, in parallel, during one iteration, EPI actually requires a significantly larger number of *agent-environment interactions* than AL-EPI.

By analyzing Figure 5.4, one can observe the advantage of our method. In particular, notice that AL-EPI masters short pendulums (i.e., achieves a performance of approximately -1500 units of reward) after about 2000 samples. The original EPI algorithm, by contrast, requires between 7400 and 12,400 samples to achieve the same level of performance. Similar observations can be made when we evaluate the generalizable policies learned by EPI and AL-EPI on medium and long pendulums. In particular, given a desired level of performance (i.e., cumulative reward), AL-EPI is always capable of achieving it by using approximately *three times fewer* samples than the original EPI algorithm. This confirms our hypothesis that AL-EPI, coupled with GPR and an acquisition function, is capable of carefully and actively selecting the most promising tasks to practice on, in order to more rapidly learn a generalizable policy.

Figure 5.4: Performance of the generalizable policy learned by EPI and AL-EPI on domains corresponding to pendulum with different rod lengths.

6 CONCLUSION

In this work we introduced a new method (AL-EPI) for learning generalizable policies that are robust to variations in an environment's dynamics. The proposed algorithm is based on constructing a model capable of *learning* a representation that characterizes the unknown dynamics of novel environments with which the agent may need to interact. Such a representation is then used to augment the agent's state, thus allowing it to learn a *single* generalizable policy that encodes adjustable behaviors that depend not only on the agent's state, but also on its estimate of the latent dynamics of the environment it is currently tackling. Our algorithm also extends the state-of-the-art by allowing the agent to reason about which tasks to practice next by considering an infinite space of candidate tasks, instead of finite, discrete tasks spaces, as done in previous techniques (ZHOU; PINTO; GUPTA, 2019). Finally, our proposed method makes use of Gaussian Process Regression and acquisition functions—used, here, to compute lower confidence bounds on the performance of a policy across different tasks—in order to identify the most promising tasks on which to practice next. These contributions allow AL-EPI to to work on continuous domains and add an informed sampling component to actively direct the domain randomization, thereby reducing the amount of data required to learn effective generalizable policies.

There are many future search directions we find interesting. First, our proposed method was tested in a toy environment. A next step would be to evaluate its performance on different, more complex domains. Furthermore, we believe that other types of kernel functions may be constructed for use with GPR, in the context of our method's objective, in order to exploit domain knowledge about how to quantify the similarity between tasks/MDPs. Similarly, we also believe that acquisition functions other than Lower Confidence Bound may be used and would result in qualitatively different exploration strategies. One promising acquisition function of interest, which has been explored in the context of intrinsic motivation algorithms in RL, is the *competence improvement* acquisition function. This function would guide the agent not to practice on tasks on which its policy does not yet perform well; instead, it would guide the agent to practice on tasks in which it is making *faster learning progress*. Finally, we also highlight the challenge of deploying GPR on non-stationary functions. In this work, we used a simple strategy for disposing of old observations. A more principled approach may be able to retain older (but still relevant) observations instead of "blindly" discarding all older experiences.

REFERENCES

ALLEVATO, A. et al. TuneNet: One-Shot Residual Tuning for System Identification and Sim-to-Real Robot Task Transfer. Jul 2019. Available from Internet: <https://arxiv.org/abs/1907.11200>.

ANTONOVA, R. et al. Reinforcement Learning for Pivoting Task. Mar 2017. Available from Internet: <http://arxiv.org/abs/1703.00472>.

BONATTI, R. et al. Learning Visuomotor Policies for Aerial Navigation Using Cross-Modal Representations. Sep 2019. Available from Internet: <https://arxiv.org/abs/1909.06993http://arxiv.org/abs/1909.06993>.

BROCHU, E.; CORA, V. M.; FREITAS, N. de. A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning. 2010. Available from Internet: <http://arxiv.org/abs/1012.2599>.

BROCKMAN, G. et al. OpenAI Gym. **arXiv preprint arXiv:1606.01540**, 2016.

CHRISTIANO, P. et al. Transfer from Simulation to Real World through Learning Deep Inverse Dynamics Model. Oct 2016. Available from Internet: <http://arxiv.org/abs/1610.03518>.

CUTLER, M.; HOW, J. P. Efficient reinforcement learning for robots using informative simulated priors. In: **Proceedings - IEEE International Conference on Robotics and Automation**. [S.l.]: Institute of Electrical and Electronics Engineers Inc., 2015. v. 2015-June, n. June, p. 2605–2612. ISSN 10504729.

CUTLER, M.; WALSH, T. J.; HOW, J. P. Reinforcement learning with multi-fidelity simulators. In: **2014 IEEE International Conference on Robotics and Automation (ICRA)**. IEEE, 2014. p. 3888–3895. ISBN 978-1-4799-3685-4. Available from Internet: <http://ieeexplore.ieee.org/document/6907423/>.

CUTLER, M.; WALSH, T. J.; HOW, J. P. Real-World Reinforcement Learning via Multi-Fidelity Simulators. **IEEE Transactions on Robotics**, v. 31, n. 3, p. 655–671, 2015. Available from Internet: <https://ieeexplore.ieee.org/document/7106543>.

EBDEN, M. Gaussian Processes: A Quick Introduction. n. August, 2015. Available from Internet: <http://arxiv.org/abs/1505.02965>.

EYSENBACH, B. et al. Off-Dynamics Reinforcement Learning: Training for Transfer with Domain Classifiers. Jun 2020. Available from Internet: <http://arxiv.org/abs/2006.13916>.

FU, J.; LEVINE, S.; ABBEEL, P. One-shot learning of manipulation skills with online dynamics adaptation and neural network priors. In: **IEEE International Conference on Intelligent Robots and Systems**. [s.n.], 2016. v. 2016-Novem, p. 4019–4026. ISBN 9781509037629. ISSN 21530866. Available from Internet: <http://arxiv.org/abs/1509.06841>.

GOLEMO, F. et al. Sim-to-Real Transfer with Neural-Augmented Robot Simulation. **Proceedings of The 2nd Conference on Robot Learning**, v. 87, n. CoRL, p. 817–828, 2018.

HANNA, J. P.; STONE, P. Grounded action transformation for robot learning in simulation. **31st AAAI Conference on Artificial Intelligence, AAAI 2017**, p. 4931–4932, 2017.

HANSEN, N. et al. Self-Supervised Policy Adaptation during Deployment. Jul 2020. Available from Internet: <https://arxiv.org/abs/2007.04309>.

HIGUERA, J. C. G.; MEGER, D.; DUDEK, G. Adapting learned robotics behaviours through policy adjustment. In: **Proceedings - IEEE International Conference on Robotics and Automation**. [S.l.: s.n.], 2017. p. 5837–5843. ISBN 9781509046331. ISSN 10504729.

HOFFMAN, J. et al. Simultaneous deep transfer across domains and tasks. In: **Advances in Computer Vision and Pattern Recognition**. [s.n.], 2017. p. 173–187. Available from Internet: <http://arxiv.org/abs/1510.02192>.

LIANG, J.; SAXENA, S.; KROEMER, O. Learning Active Task-Oriented Exploration Policies for Bridging the Sim-to-Real Gap. Jun 2020. Available from Internet: <http://arxiv.org/abs/2006.01952>.

LOPEZ-GUEVARA, T. et al. Adaptable Pouring: Teaching Robots Not to Spill using Fast but Approximate Fluid Simulation. **Proceedings of the 1st Annual Conference on Robot Learning**, v. 78, n. CoRL, p. 77–86, 2017. Available from Internet: <http://proceedings.mlr.press/v78/lopez-guevara17a/lopez-guevara17a.pdf>.

MURATORE, F. et al. Data-efficient Domain Randomization with Bayesian Optimization. **IEEE Robotics and Automation Letters**, Institute of Electrical and Electronics Engineers Inc., Mar 2020. Available from Internet: <http://arxiv.org/abs/2003.02471>.

PENG, X. B. et al. Sim-to-Real Transfer of Robotic Control with Dynamics Randomization. In: **Proceedings - IEEE International Conference on Robotics and Automation**. [s.n.], 2018. p. 3803–3810. ISBN 9781538630815. ISSN 10504729. Available from Internet: <http://dx.doi.org/10.1109/ICRA.2018.8460528>.

RASMUSSEN, C. E. Gaussian Processes in machine learning. **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, v. 3176, p. 63–71, 2004. ISSN 16113349.

RUSU, A. A. et al. Sim-to-Real Robot Learning from Pixels with Progressive Nets. Oct 2016. Available from Internet: <http://arxiv.org/abs/1610.04286>.

SILVER, T. et al. Residual Policy Learning. Dec 2018. Available from Internet: <http://arxiv.org/abs/1812.06298>.

SONG, X. et al. Rapidly Adaptable Legged Robots via Evolutionary Meta-Learning. Mar 2020. Available from Internet: <http://arxiv.org/abs/2003.01239>.

SUTTON, R. S.; BARTO, A. G. **Reinforcement learning: An introduction**. [S.l.]: MIT press, 2018.

TZENG, E. et al. Adapting Deep Visuomotor Representations with Weak Pairwise Constraints. Nov 2015. Available from Internet: <https://arxiv.org/abs/1511.07111>.

VIERECK, U. et al. Learning a visuomotor controller for real world robotic grasping using simulated depth images. **CoRL**, Jun 2017. Available from Internet: <http://arxiv.org/abs/1706.04652>.

VIERECK, U.; SAENKO, K.; JR., R. P. Adapting control policies from simulation to reality using a pairwise loss. **CoRR**, abs/1807.1, 2018. Available from Internet: <http://arxiv.org/abs/1807.10413>.

YU, W. et al. Preparing for the Unknown: Learning a Universal Policy with Online System Identification. **Robotics: Science and Systems**, MIT Press Journals, v. 13, Feb 2017. Available from Internet: <http://arxiv.org/abs/1702.02453>.

ZHOU, W.; PINTO, L.; GUPTA, A. Environment Probing Interaction Policies. Jul 2019. Available from Internet: <https://arxiv.org/abs/1907.11740>.