What Supervision Scales? Practical Learning Through Interaction

Carlos Florensa Campo

Acknowledgement

This PhD has been a long endeavour that started many years before arriving to Berkeley, and I want to dedicate this dissertation to the early beacons that have guided my steps in this direction. See the dissertation file for full acknowledgments.

What Supervision Scales? Practical Learning Through Interaction

by

Carlos Florensa

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Pieter Abbeel, Chair
Assistant Professor Sergey Levine
Professor Ken Goldberg
Professor John Canny

Spring 2020

What Supervision Scales? Practical Learning Through Interaction

Abstract

What Supervision Scales? Practical Learning Through Interaction

by

Carlos Florensa

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Pieter Abbeel, Chair

To have an agent learn useful behaviors, we must be able to specify what are the desired outcomes. This supervision can come in many forms, like the reward in Reinforcement Learning (RL), the target state-action pairs in Imitation Learning, or the dynamics model in motion-planning. Each form of supervision needs to be evaluated along three cost dimensions that dictate how well it scales: how much domain knowledge is required to provide that supervision form, what is the total volume of interaction under such supervision needed to learn a task, and how does this amount increases for every new task that the agent has to learn. For example, guiding rewards provided at every time-step might speed up an RL algorithm, but it is hard to design such dense rewards that are easy-to-provide and induce a solution to the task at hand, and the design process must be repeated for every new task; On the other hand, a completion signal is a weaker form of supervision because non-expert users can specify the objective to many tasks in this way, but unfortunately standard RL algorithms struggle with such sparse rewards. In the first part of this dissertation we study how overcome this limitation by means of learning hierarchies over re-usable skills. In the second part of this dissertation, we extend the scope to explicitly minimize the supervision needed to learn distributions of tasks. This paradigm shifts the focus away from the complexity of learning a single task, hence paving the way towards more general agents that efficiently learn from multiple tasks. To achieve this objective, we propose two automatic curriculum generation methods. In the third part of the dissertation, we investigate how to leverage different kinds of partial experts as supervision. First we propose a method that does not require any reward, and is still able to largely surpass the performance of the demonstrator in goal-reaching tasks. This allows to leverage sub-optimal “experts", hence lowering the cost of the provided supervision. Finally we explore how to exploit a rough description of a task and an “expert" able to operate in only parts of the state-space. This is a common setting in robotic applications where the model provided by the manufacturer allows to execute efficient motion-planning as long as there’s no contacts or perception errors, but fails to complete the last contact-rich part of the task, like inserting a key. These are all key pieces to provide supervision that scales to generate robotic behavior for practical tasks.

To my family.

Contents

List of Figures

List of Tables

Acknowledgments

This PhD has been a long endeavour that started many years before arriving to Berkeley, and I want to dedicate this dissertation to the early beacons that have guided my steps in this direction. Obviously, it all starts with my parents Senén Florensa and Maria Antonia Campo, and my twin brother Luis. With their love, blind faith, and high expectations, they made me push myself harder every day. I am grateful to my whole family, who has always been there to encourage me in the path to research, specially my uncle Joan Guinovart. Beyond my relatives, I want to acknowledge how important it was to meet other young research enthusiasts that proved me how enjoyable and thrilling it can be to work in science, like Miguel Picallo, many colleagues at the CFIS, my cohort at Xlab, or the other summer fellows at ICFO. This last institution gave me one of the strongest motivation thrusts, thanks to its inspiring director Lluis Torner and my supervisor Silvia Carrasco.

My first in-depth research experiences came under the wing of Victor Zavala at Argonne National Lab, who offered me the most intellectually thriving summers of my undergrad. He also propelled me to work with Ignacio Grossmann and Pablo Garcia-Herreros at CMU, who consolidated my eagerness to devote myself to cutting edge science.

Combined with the generous La Caixa fellowship, I was ready to set sails for California! I want to highlight how key the community around this fellowship has been: it provided role-models to look up to and raise my own bar, but most importantly, it provided a warm family in Berkeley, the famous "Berkeley Gang", including Vicenç Rubies, Carlitos Ruiz, Julia Gómez, and Jaime Fernández Fisac. Another important source of support for the challenges of the first years in the PhD were all the friends from the I-House: Xabi Azaguirre, Maria Eckstein, Kate Beck, Signe Kristensen, and many others. Finally, there is one person I owe living in the best house in Berkeley, with the best housemate himself: Adam Stooke! Becoming close friends has been the best side-effect of struggling together to join Pieter's lab from a non-Computer-Science background. We made sure to compensate every hard time with a huge paella, or with an outdoors trip with the rest of the close group: Kate Rakelly, Rebecca Sarto, Kim, Dan, etc.

Academically, I must thank my advisor Pieter Abbeel for granting me the opportunity to get involved in AI and Robotics, for providing great mentorship the years I needed it the most, and for his generous support throughout the PhD. His research group was a thriving place full of ideas that fueled my determination to contribute to the field of Reinforcement Learning. Many late-night discussions with Abhishek Gupta, Sandy Huang, Greg Khan or Ignasi Clavera kept me dreaming about autonomous agents. My direct collaborators in the group have also been some of the people I've learned the most from, like Yan Duan (yes, the legendary Rocky) or David Held. I also want to give a huge shut-out to the two undergrads I've mentored: Yiming Ding and Alexander Li, now starting their own PhD adventures at Stanford and CMU.

Beyond Berkeley, I've been fortunate to do internships at DeepMind, Nvidia, and Covariant. In each place I've gained a new point of view on my research field, all complementing each other. I want to particularly praise the contagious vision of Nicolas Heess at DeepMind, the insightful discussions with Nathan Ratliff and Dieter Fox, and all the fun I had collaborating with Michelle Lee and John Tremblay at Nvidia. Finally, I'm grateful to Covariant for providing an electrifying environment to start pushing AI robotics research into the real world!

Chapter 1

Introduction

In this dissertation we study autonomous learning agents from the perspective of the cost of the supervision they require to perform a range of tasks, and how to minimize it.

In the first part of this dissertation, Chapter 2, we focus on perhaps the most common kind of supervision for autonomous agents: rewards. Specifying a task through rewards holds the promise to allow users to describe "what" they want a complex system to do, but without needing to know "how" it should perform it. From this perspective, a system able to learn only through rewards and interactions with its environments requires a supervision that can be provided by non-experts, and hence might scale better. In many cases, the most straight-forward way to specify a task is through a completion signal, giving the agent a reward of 1 when it completes the task and 0 otherwise. For example, in navigation tasks, the agent is not rewarded until it finds the target. This type of reward is commonly referred to as "sparse", and does not require the user to have any knowledge of the system to shape the reward appropriately. Unfortunately, most RL algorithms struggle to find solutions under such weak supervision. In this chapter of the dissertation we propose two methods for training policies for a collection of tasks with sparse rewards. The first framework starts by learning a span of skills in a pre-training environment, where it is employed with nothing but a proxy reward signal, whose design only requires very minimal domain knowledge. This proxy reward can be understood as a form of intrinsic motivation that encourages the agent to explore its own capabilities, without any goal information or sensor readings specific to each downstream task. Then a hierarchical architecture is proposed to leverage these skills in a variety of downstream tasks. The second method we introduce allows to train both levels of a hierarchical end-to-end, therefore enhancing many prior HRL works like the one described above. Having fixed skills can considerably cap the final performance on the new task, and little work has been done on adapting pre-trained sub-policies to be optimal for a new task.

In the second part of this dissertation, Chapter 3, we focus on efficiently learning sets of tasks so that the total supervision needed to learn all tasks is minimized. The most natural question when trying to train an agent to perform well on a set of tasks is *whether there is an ordering of the tasks* that makes the process more efficient than randomly training on one task at a time. But the gain in sample complexity shouldn't come to the expense of more expert supervision designing this ordering. Therefore in this section we propose two different kinds of *automatic curriculum* that don't require any additional supervision and hence scale better to large sets of tasks. Our first curriculum method uses a generator network to propose tasks for the agent to try to accomplish, each task being specified as reaching a certain parametrized subset of the state-space. The generator network is optimized using a Goal Generative Adversarial Network (Goal GAN), to produce tasks that are always at the appropriate level of difficulty for the agent. We show that, by using this framework, an agent can efficiently and automatically learn to perform a wide set of tasks without requiring any prior knowledge of its environment, even when only sparse rewards are available Our second approach is particularly geared towards reaching a single extremely hard-to-reach goal from anywhere, like when a seven DoF robotic arm has to robustly place a ring onto a peg. Therefore the "set of tasks" is now parameterized by the start-state rather than the goal-state, that is the same for all tasks. Past approaches tackle these problems by exploiting expert demonstrations or by manually designing a task-specific reward shaping function to guide the learning agent. Instead, we propose a method to learn these tasks without requiring any prior knowledge other than obtaining a single state in which the task is achieved. The robot is trained in "reverse", gradually learning to reach the goal from a set of start states increasingly far from the goal. Our method automatically generates a curriculum of start states that adapts to the agent's performance, leading to efficient training on goal-oriented tasks. We demonstrate our approach on difficult simulated navigation and fine-grained manipulation problems, not solvable by state-of-the-art reinforcement learning methods.

The third part of this dissertation, Chapter 4, broadens the spectrum of possible supervisions that can be leveraged to obtain a desired behavior. We strive for other types of supervision because designing rewards for Reinforcement Learning (RL) can be challenging: it needs to convey the desired task, be efficient to optimize, and be easy to compute. The latter is particularly problematic when applying RL to robotics, where detecting whether the desired configuration is reached might require considerable supervision and instrumentation. Therefore, we deepen into two possible methods to exploit two different kinds of partial experts. First, we propose a novel algorithm *goalGAIL*, which incorporates sub-optimal demonstrations to drastically speed up the convergence to a policy able to reach any goal, surpassing the performance of an agent trained with other Imitation Learning algorithms. Furthermore, we show our method can also be used when the available expert trajectories do not contain the actions, which makes it applicable when only kinesthetic, third-person or noisy demonstrations are available. Finally we explore how to exploit a rough description of a task and an "expert" able to operate in only parts of the state-space. This is a common setting in robotic applications where the model provided by the manufacturer allows to execute efficient motion-planning as long as there's no contacts or perception errors, but fails to complete the last contact-rich part of the task, like manipulating a key. This method is shown to be very efficient in real-world tasks like peg insertion.

This dissertation composes the work published in the following articles:

- Carlos Florensa, Yan Duan, and Pieter Abbeel. "Stochastic Neural Networks for Hierarchical Reinforcement Learning". In: *International Conference in Learning Representations* (2017), pp. 1–17
- Alexander C. Li*, Carlos Florensa*, Ignasi Clavera, and Pieter Abbeel. "Sub-policy Adaptation for Hierarchical Reinforcement Learning". In: *International Conference on Learning Representations* (2020)
- Carlos Florensa*, David Held*, Xinyang Geng*, and Pieter Abbeel. "Automatic Goal Generation for Reinforcement Learning Agents". In: *International Conference in Machine Learning* (2018)
- Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. "Reverse Curriculum Generation for Reinforcement Learning". In: *Conference on Robot Learning* (2017)
- Yiming Ding*, Carlos Florensa*, Mariano Phielipp, and Pieter Abbeel. "Goal-conditioned Imitation Learning". In: *Advances in Neural Information Processing Systems* (2019)
- Michelle A Lee*, Carlos Florensa*, Jonathan Tremblay, Nathan Ratliff, Animesh Garg, Fabio Ramos, and Dieter Fox. "Guided Uncertainty-Aware Policy Optimization: Combining Learning and Model-Based Strategies for Sample-Efficient Policy Learning". In: *International Conference on Robotics and Automation* (2020)

If interested in the full body of work by Carlos Florensa, visit:
`https://sites.google.com/view/carlosflorensa`.

Chapter 2

Hierarchical RL to Learn with Sparse Rewards

2.1 Introduction

Specifying a task through rewards holds the promise to allow users to describe "what" they want a complex system to do, but not "how" it should perform it. From this perspective, a system able to learn only through rewards and interactions with its environments requires a supervision that can be provided by non-experts, and hence might scale better. In many cases, the most straight-forward way to specify a task is through a completion signal, giving the agent a reward of 1 when it completes the task and 0 otherwise. For example, in navigation tasks, the agent is not rewarded until it finds the target. This type of reward is commonly referred to as "sparse", and does not require the user to have any knowledge of the system to shape the reward appropriately.

Reinforcement learning (RL) has made great progress at learning only from a reward and interaction, from playing games such as Pong and Go [103, 142] to automating robotic locomotion [138, 54], dexterous manipulation [36, 3], and perception [106, 38]. Despite these success stories, these deep RL algorithms typically employ naive exploration strategies such as ϵ-greedy or uniform Gaussian exploration noise, which have been shown to perform poorly in tasks with sparse rewards [30, 62, 11]. The challenge is further complicated by long horizons, where naive exploration strategies can lead to exponentially large sample complexity [110]. Furthermore, most work in RL is still learn from scratch when faced with a new problem. This is particularly inefficient when tackling multiple related tasks that are hard to solve due to sparse rewards or long horizons. Therefore, to fully leverage that sparse rewards scale well in terms of not requiring expert knowledge, we need to overcome the optimization challenges that they pose.

To tackle these challenges, two main strategies have been pursued: The first strategy is to design a hierarchy over the actions [111, 156, 28]. By composing low-level actions into high-level primitives, the search space can be reduced exponentially. However, these approaches require domain-specific knowledge and careful hand-engineering. The second strategy uses intrinsic rewards to encourage exploration [130, 131, 62, 11]. The computation of these intrinsic rewards does not require domain-specific knowledge. However, when facing a collection of tasks, these methods do not provide a direct answer about how knowledge about solving one task may transfer to other tasks, and by solving each of the tasks from scratch, the overall sample complexity may still be high.

In this chapter, we propose two methods for training policies for a collection of tasks with sparse rewards. The first framework, starts by learning a span of skills in a pre-training environment, where it is employed with nothing but a proxy reward signal, whose design only requires very minimal domain knowledge about the downstream tasks. This proxy reward can be understood as a form of intrinsic motivation that encourages the agent to explore its own capabilities, without any goal information or sensor readings specific to each downstream task. The set of skills can be used later in a wide collection of different tasks, by training a separate high-level policy for each task on top of the skills, thus reducing sample complexity uniformly. To learn the span of skills, we propose to use Stochastic Neural Networks (SNNs) [108, 107, 160], a class of neural networks with stochastic units in the computation graph. This class of architectures can easily represent multi-modal policies, while achieving weight sharing among the different modes. We parametrize the stochasticity in the network by feeding latent variables with simple distributions as extra input to the policy. In the experiments, we observed that direct application of SNNs does not always guarantee that a wide range of skills will be learned. Hence, we propose an information-theoretic regularizer based on Mutual Information (MI) in the pre-training phase to encourage diversity of behaviors of the SNN policy. Our experiments[1] show[2] that this combination is effective in learning a wide span of interpretable skills in a sample-efficient way, and can significantly boost the learning performance uniformly across a wide range of downstream tasks. We call our full method SNN4HRL.

The second method we introduce allows to train both levels of a hierarchical end-to-end, therefore enhancing many prior HRL works like the one described above. Having fixed skills can considerably cap the final performance on the new task, and little work has been done on adapting pre-trained sub-policies to be optimal for a new task. To develop a new framework for simultaneously adapting all levels of temporal hierarchies we do several contributions. First, we derive an efficient approximated hierarchical policy gradient. The key insight is that, despite the decisions of the manager being unobserved latent variables from the point of view of the Markovian environment, from the perspective of the sub-policies they can be considered as part of the observation. We show that this provides a decoupling of the manager and sub-policy gradients, which greatly simplifies the computation in a principled way. It also theoretically justifies a technique used in other prior works [40]. Second, we introduce a sub-policy specific baseline for our hierarchical policy gradient. We prove that this baseline is unbiased, and our experiments reveal faster convergence, suggesting efficient gradient variance reduction. Then, we introduce a more stable way of using this gradient, Hierarchical Proximal Policy Optimization (HiPPO). This method helps us take more conservative steps in our policy space [137], critical in hierarchies because of the interdependence of each layer. Results show that HiPPO is highly efficient both when learning from scratch, i.e. adapting randomly initialized skills, and when adapting pretrained skills on a new task. Finally, we evaluate the benefit

[1]Code available at: `https://github.com/florensacc/snn4hrl`

[2]Videos available at: `http://bit.ly/snn4hrl-videos`

of randomizing the time-commitment of the sub-policies, and show it helps both in terms of final performance and zero-shot adaptation on similar tasks.

2.2 Related Work

One of the main appealing aspects of hierarchical reinforcement learning (HRL) is to use skills to reduce the search complexity of the problem [111, 156, 28]. However, specifying a good hierarchy by hand requires domain-specific knowledge and careful engineering, hence motivating the need for learning skills automatically. Prior work on automatic learning of skills has largely focused on learning skills in discrete domains [20, 174]. A popular approach there is to use statistics about state transitions to identify bottleneck states [150, 98, 144]. It is however not clear how these techniques can be applied to settings with high-dimensional continuous spaces, and only recently HRL has been applied to high-dimensional continuous domains as we do in this chapter [79, 24].

To obtain the lower level policies, or skills, most methods exploit some additional assumptions, like access to demonstrations [85, 101, 121, 139], policy sketches [1], or task decomposition into sub-tasks [45, 147]. Other methods use a different reward for the lower level, often constraining it to be a "goal reacher" policy, where the signal from the higher level is the goal to reach [104, 93, 172]. These methods are very promising for state-reaching tasks, but might require access to goal-reaching reward systems not defined in the original MDP, and are more limited when training on tasks beyond state-reaching. Our methods does not require any additional supervision, and the obtained skills are not constrained to be goal-reaching.

There has also been work on learning skills in tasks with continuous actions [128, 76, 22, 122]. These methods extract useful skills from successful trajectories for the same (or closely related) tasks, and hence require first solving a comparably challenging task or demonstrations. Guided Policy Search (GPS) [90] leverages access to training in a simpler setting. GPS trains with iLQG [167] in state space, and in parallel trains a neural net that only receives raw sensory inputs that has to agree with the iLQG controller. The neural net policy is then able to generalize to new situations.

Another line of work on skill discovery particularly relevant to our approaches is the HiREPS algorithm by [23] where, instead of having a mutual information bonus like in our proposed approach, they introduce a constraint on an equivalent metric. The solution approach is nevertheless very different as they cannot use policy gradients. Furthermore, although they do achieve multimodality like us, they only tried the episodic case where a single option is active during the rollout. Hence the hierarchical use of the learned skills is less clear. Similarly, the Option-critic architecture [7] can learn interpretable skills, but whether they can be reuse across complex tasks is still an open question. Recently, [55] have independently proposed to learn a range of skills in a pre-training environment that will be useful for the downstream tasks, which is similar to our framework. However, their pre-training setup requires a set of goals to be specified. In comparison, we use a signle proxy reward as the only signal to the agent during the pre-training phase, the construction of which only requires minimal domain knowledge or instrumentation of the environment.

When transferring skills to a new environment, most HRL methods keep them fixed and simply train a new higher-level on top [53, 56]. Other work allows for building on previous skills by constantly supplementing the set of skills with new ones [141], but they require a hand-defined curriculum of tasks, and the previous skills are never fine-tuned. Our algorithm allows for seamless adaptation of the skills, showing no trade-off between leveraging the power of the hierarchy and the final performance in a new task. Other methods use invertible functions as skills [50], and therefore a fixed skill can be fully overwritten when a new layer of hierarchy is added on top. This kind of "fine-tuning" is promising, although similar to other works [114], they do not apply it to temporally extended skills as we do here.

One of the most general frameworks to define temporally extended hierarchies is the options framework [157], and it has recently been applied to continuous state spaces [6]. One of the most delicate parts of this formulation is the termination policy, and it requires several regularizers to avoid skill collapse [52, 173]. This modification of the objective may be difficult to tune and affects the final performance. Instead of adding such penalties, we propose to have skills of a random length, not controlled by the agent during training of the skills. The benefit is two-fold: no termination policy to train, and more stable skills that transfer better. Furthermore, these works only used discrete action MDPs. We lift this assumption, and show good performance of our algorithm in complex locomotion tasks. There are other algorithms recently proposed that go in the same direction, but we found them more complex, less principled (their per-action marginalization cannot capture well the temporal correlation within each option), and without available code or evidence of outperforming non-hierarchical methods [145].

The closest work to ours in terms of final algorithm structure is the one proposed by Frans et al. [40]. Their method can be included in our framework, and hence benefits from our new theoretical insights. We introduce a modification that is shown to be highly beneficial: the random time-commitment mentioned above, and find that our methods can learn in difficult environments without their complicated training scheme.

2.3 Hierarchical Policies

Basic Definitions

We define a discrete-time finite-horizon discounted Markov decision process (MDP) by a tuple $M = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \rho_0, \gamma, T)$, in which $\mathcal{S}$ is a state set, $\mathcal{A}$ an action set, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_+$ a transition probability distribution, $r : \mathcal{S} \times \mathcal{A} \to [-R_{\max}, R_{\max}]$ a bounded reward function, $\rho_0 : \mathcal{S} \to \mathbb{R}_+$ an initial state distribution, $\gamma \in [0, 1]$ a discount factor, and T the horizon. When necessary, we attach a suffix to the symbols to resolve ambiguity, such as $\mathcal{S}^M$. In policy search methods, we typically optimize a stochastic policy $\pi_\theta : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_+$ parametrized by θ. The objective is to maximize its expected discounted return, $\eta(\pi_\theta) = \mathbb{E}_\tau[\sum_{t=0}^{T} \gamma^t r(s_t, a_t)]$, where $\tau = (s_0, a_0, \ldots)$ denotes the whole trajectory, $s_0 \sim \rho_0(s_0)$, $a_t \sim \pi_\theta(a_t|s_t)$, and $s_{t+1} \sim \mathcal{P}(s_{t+1}|s_t, a_t)$.

In this chapter, we propose two methods to learn a hierarchical policy to efficiently perform a new task. We study hierarchical policies composed of a higher level, or manager $\pi_{\theta_h}(z_t|s_t)$, and a

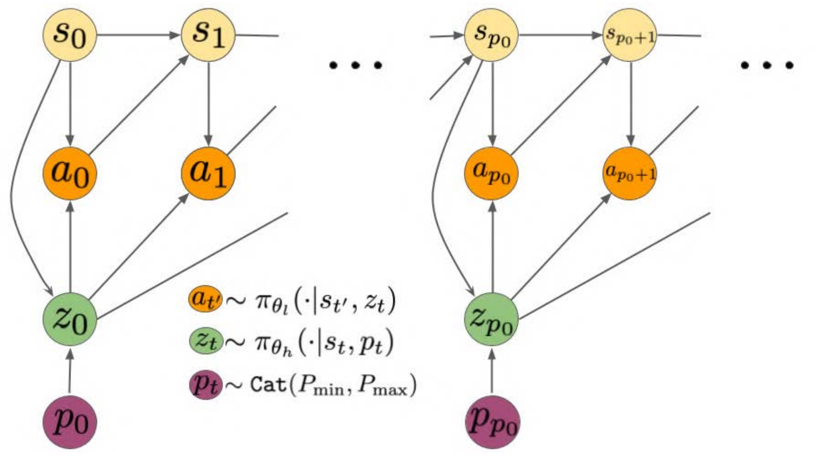

s_0
s_1
s_{p_0}
s_{p_0+1}
a_0
a_1
a_{p_0}
a_{p_0+1}
z_0
z_{p_0}
p_0
p_{p_0}
$a_{t'} \sim \pi_{\theta_l}(\cdot|s_{t'}, z_t)$
$z_t \sim \pi_{\theta_h}(\cdot|s_t, p_t)$
$p_t \sim \texttt{Cat}(P_{\min}, P_{\max})$

Figure 2.1: Temporal hierarchy studied in this section. A latent code z_t is sampled from the manager policy $\pi_{\theta_h}(z_t|s_t)$ every p time-steps, using the current observation s_{kp}. The actions a_t are sampled from the sub-policy $\pi_{\theta_l}(a_t|s_t, z_{kp})$ conditioned on the same latent code from $t = kp$ to $(k+1)p - 1$

lower level, or sub-policy $\pi_{\theta_l}(a_{t'}|z_t, s_{t'})$. The higher level does not take actions in the environment directly, but rather outputs a command, or latent variable $z_t \in \mathcal{Z}$, that conditions the behavior of the lower level. We focus on the common case where $\mathcal{Z} = \mathbb{Z}_n$ making the manager choose among n sub-policies, or skills, to execute. The manager typically operates at a lower frequency than the sub-policies, only observing the environment every p time-steps. When the manager receives a new observation, it decides which low level policy to commit to for p environment steps by the means of a latent code z. Figure 2.1 depicts this framework where the high level frequency p is a random variable, which is one of the contribution we introduce in our HiPPO algorithm as described in Section 2.6. In most works, including the first one we present in 2.4, this is a deterministic hyper-parameter. Note that the class of hierarchical policies we work with is more restrictive than others like the Options framework, where the time-commitment is also decided by the policy. Nevertheless, we show that this loss in policy expressivity acts as a regularizer and does not prevent our algorithms from surpassing other state-of-the art methods.

Problem statement for leveraging reusable skills

Our main interest is in solving a collection of downstream tasks, specified via a collection of MDPs $\mathcal{M}$. If these tasks do not share any common structure, we cannot expect to acquire a set of skills that will speed up learning for all of them. On the other hand, we want the structural assumptions to be minimal, to make our problem statement more generally applicable.

For each MDP $M \in \mathcal{M}$, we assume that the state space $\mathcal{S}^M$ can be factored into two components, $\mathcal{S}_{\text{agent}}$, and $\mathcal{S}^M_{\text{rest}}$, which only weakly interact with each other. The $\mathcal{S}_{\text{agent}}$ should be the same for all MDPs in $\mathcal{M}$. We also assume that all the MDPs share the same action space. Intuitively, we consider a robot who faces a collection of tasks, where the dynamics of the robot are shared across tasks, and are covered by $\mathcal{S}_{\text{agent}}$, but there may be other components in a task-specific state space, which will be denoted by $\mathcal{S}_{\text{rest}}$. For instance, in a grasping task M, $\mathcal{S}^{M}_{\text{rest}}$ may include the positions of objects at interest or the new sensory input associated with the task. This specific structural assumption has been studied in the past as sharing the same *agent-space* [75].

Given a collection of tasks satisfying the structural assumption, our objective for designing the algorithm is to minimize the total sample complexity required to solve these tasks. This has been more commonly studied in the past in sequential settings, where the agent is exposed to a sequence of tasks, and should learn to make use of experience gathered from solving earlier tasks to help solve later tasks [161, 178, 83, 27]. However, this requires that the earlier tasks are relatively easy to solve (e.g., through good reward shaping or simply being easier) and is not directly applicable when all the tasks have sparse rewards, which is a much more challenging setting. In the next section, we describe our formulation, which takes advantage of a pre-training task that can be constructed with minimal domain knowledge, and can be applied to the more challenging scenario.

2.4 Learning Diverse Skills with Stochastic Neural Networks

In this section, we describe our formulation to solve a collection of tasks, exploiting the structural assumption articulated above. In Sec. 2.4, we describe the pre-training environment, where we use proxy rewards to learn a useful span of skills. In Sec. 2.4, we motivate the usage of Stochastic Neural Networks (SNNs) and discuss the architectural design choices made to tailor them to skill learning for RL. In Sec. 2.4, we describe an information-theoretic regularizer that further improves the span of skills learned by SNNs. In Sec. 2.4, we describe the architecture of high-level policies over the learned skills, and the training procedure for the downstream tasks with sparse rewards. Finally, in Sec. 2.4, we describe the policy optimization for both phases of training.

Constructing the pre-training environment

Given a collection of tasks, we would like to construct a pre-training environment, where the agent can learn a span of skills, useful for enhancing exploration in downstream tasks. We achieve this by letting the agent freely interact with the environment in a minimal setup. For a mobile robot, this can be a spacious environment where the robot can first learn the necessary locomotion skills; for a manipulator arm which will be used for object manipulation tasks, this can be an environment with many objects that the robot can interact with.

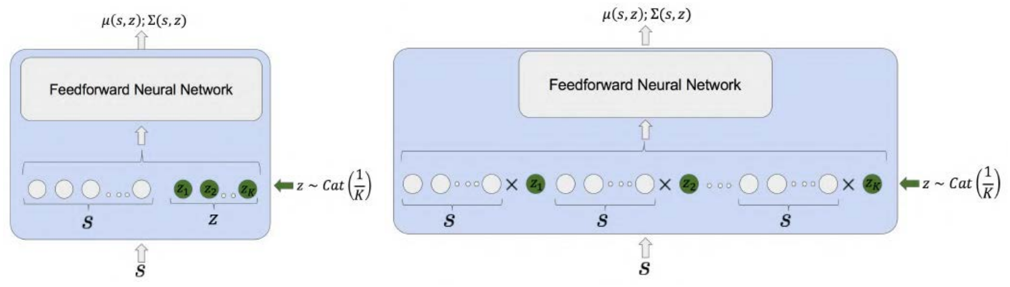
The skills learned in this environment will depend on the reward given to the agent. Rather than setting different rewards in the pre-training environment corresponding to the desired skills, which requires precise specification about what each skill should entail, we use a generic proxy reward as the only reward signal to guide skill learning. The design of the proxy reward should encourage the existence of locally optimal solutions, which will correspond to different skills the agent should learn. In other words, it encodes the prior knowledge about what high level behaviors might be useful in the downstream tasks, rewarding all of them roughly equally. For a mobile robot, this reward can be as simple as proportional to the magnitude of the speed of the robot, without constraining the direction of movement. For a manipulator arm, this reward can be the successful grasping of any object.

Every time we train a usual uni-modal gaussian policy in such environment, it should converge towards a potentially different skill. As seen in Sec. 2.5, applying this approach to mobile robots gives a different direction (and type) of locomotion every time. But it is an inefficient procedure: training each policy from scratch is not sample efficient (sample complexity grows linearly with the number of skills to be learned) and nothing encourages the individual skills to be distinct. The first issue will be addressed in the next subsection by using Stochastic Neural Networks as policies and the second issue will be addressed in Sec. 2.4 by adding an information-theoretic regularizer.

Stochastic Neural Networks for Skill Learning

To learn several skills at the same time, we propose to use Stochastic Neural Networks (SNNs), a general class of neural networks with stochastic units in the computation graph. There has been a large body of prior work on special classes of SNNs, such as Restricted Boltzmann Machines (RBMs) [146, 58], Deep Belief Networks (DBNs) [59], and Sigmoid Belief Networks (SBNs) [108, 160]. They have rich representation power and can in fact approximate any well-behaved probability distributions [84, 21]. Policies modeled via SNNs can hence represent complex action distributions, especially multi-modal distributions.

For our purpose, we use a simple class of SNNs, where latent variables with fixed distributions are integrated with the inputs to the neural network (here, the observations from the environment) to form a joint embedding, which is then fed to a standard feed-forward neural network (FNN) with deterministic units, that computes distribution parameters for a uni-modal distribution (e.g. the mean and variance parameters of a multivariate Gaussian). We use simple categorical distributions with uniform weights for the latent variables, where the number of classes, K, is an hyperparameter that upper bounds the number of skills that we would like to learn.


$\mu(s,z); \Sigma(s,z)$
Feedforward Neural Network
z_1 z_2 z_K
$z \sim Cat\left(\frac{1}{K}\right)$
s
z
$\times$

(a) Concatenation (b) Bilinear integration

Figure 2.2: Different architectures for the integration of the latent variables

The simplest joint embedding, as shown in Figure 2.2a, is to concatenate the observations and the latent variables directly.[3] However, this limits the expressiveness power of integration between the observation and latent variable. Richer forms of integrations, such as multiplicative integrations and bilinear pooling, have been shown to have greater representation power and improve the optimization landscape, achieving better results when complex interactions are needed [42, 179]. Inspired by this work, we study using a simple bilinear integration, by forming the outer product between the observation and the latent variable (Fig. 2.2b). Note that the concatenation integration effectively corresponds to changing the bias term of the first hidden layer depending on the latent code z sampled, and the bilinear integration to changing all the first hidden layer weights. As shown in the experiments, the choice of integration greatly affects the quality of the span of skills that is learned. Our bilinear integration already yields a large span of skills, hence no other type of SNNs is studied in this work.

To obtain temporally extended and consistent behaviors associated with each latent code, in the pre-training environment we sample a latent code at the beginning of every rollout and keep it constant throughout the entire rollout. After training, each of the latent codes in the categorical distribution will correspond to a different, interpretable skill, which can then be used for the downstream tasks.

Compared to training separate policies, training a single SNN allows for flexible weight-sharing schemes among different policies. It also takes a comparable amount of samples to train an SNN than a regular uni-modal gaussian policy, so the sample efficiency of training K skills is effectively K times better. To further encourage the diversity of skills learned by the SNN we introduce an information theoretic regularizer, as detailed in the next section.

Information-Theoretic Regularization

Although SNNs have sufficient expressiveness to represent multi-modal policies, there is nothing in the optimization that prevents them from collapsing into a single mode. We have not observed this worst case scenario in our experiments, but sometimes different latent codes correspond to very similar skills. It is desirable to have direct control over the diversity of skills that will be learned. To achieve this, we introduce an information-theoretic regularizer, inspired by recent success of similar objectives in encouraging interpretable representation learning in InfoGAN [18].

Concretely, we add an additional reward bonus, proportional to the mutual information (MI) between the latent variable and the current state. We only measure the MI with respect to a relevant subset of the state. For a mobile robot, we choose this to be the $c = (x, y)$ coordinates of its center of mass (CoM). Formally, let C be a random variable denoting the current CoM coordinate of the agent, and let Z be the latent variable. Then the MI can be expressed as $I(Z; C) = H(Z) - H(Z|C)$, where H denotes the entropy function. In our case, $H(Z)$ is constant since the probability distribution of the latent variable is fixed to uniform during this stage of training. Hence, maximizing the MI is equivalent to minimizing the conditional entropy $H(Z|C)$. As entropy is a measure of uncertainty, another interpretation of this bonus is that, given where the robot is, it should be easy to infer which

[3]In scenarios where the observation is very high-dimensional such as images, we can form a low-dimensional embedding of the observation alone, say using a neural network, before jointly embedding it with the latent variable.

skill the robot is currently performing. To penalize $H(Z|C) = -\mathbb{E}_{z,c}\log p(Z = z|C = c)$, we modify the reward received at every step as specified in Eq. (2.1), where $\hat{p}(Z = z^n|c_t^n)$ is an estimate of the posterior probability of the latent code z^n sampled on rollout n, given the coordinates c_t^n at time t of that rollout.

$$R_t^n \leftarrow R_t^n + \alpha_H \ \log \hat{p}(Z = z^n|c_t^n) \tag{2.1}$$

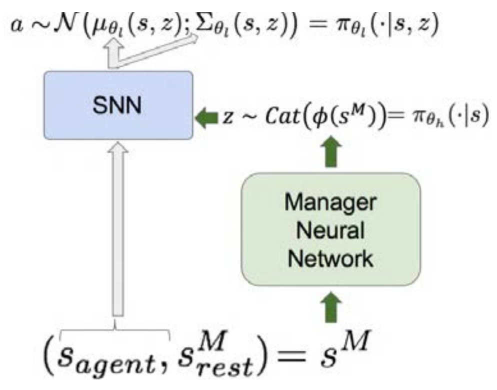
To estimate the posterior $\hat{p}(Z = z^n|c_t^n)$, we apply the following discretization: we partition the (x, y) coordinate space into cells and map the continuous-valued CoM coordinates into the cell containing it. Overloading notation, now c_t is a discrete variable indicating the cell where the CoM is at time t. This way, calculation of the empirical posterior only requires maintaining visitation counts $m_c(z)$ of how many times each cell c is visited when latent code z is sampled. Given that we use a batch policy optimization method, we use all trajectories of the current batch to compute all the $m_c(z)$ and estimate $\hat{p}(Z = z^n|c_t^n)$, as shown in Eq. (2.2). If c is more high dimensional, the posterior $\hat{p}(Z = z^n|c_t^n)$ can also be estimated by fitting a MLP regressor by Maximum Likelihood.

$$\hat{p}\big(Z = z\big|(x, y)\big) \approx \hat{p}(Z = z|c) = \frac{m_c(z)}{\sum_{z'} m_c(z')} \tag{2.2}$$

Learning High-level Policies

Given a span of K skills learned during the pre-training task, we now describe how to use them as basic building blocks for solving tasks where only sparse reward signals are provided. Instead of learning from scratch the low-level controls, we leverage the provided skills by freezing them and training a high-level policy (Manager Neural Network) that operates by selecting a skill and committing to it for a fixed amount of steps $\mathcal{T}$. The imposed temporal consistency and the quality of the skills (in our case given by optimizing the proxy reward in the pre-train environment) yield an enhanced exploration allowing to solve the downstream tasks with sparse rewards.

For any given task $M \in \mathcal{M}$ we train a new Manager NN on top of the common skills. Given the factored representation of the state space $\mathcal{S}^M$ as $\mathcal{S}_{\text{agent}}$ and $\mathcal{S}^M_{\text{rest}}$, the high-level policy receives the full state as input, and outputs the parametrization of a categorical distribution from which we sample a discrete action z out of K possible choices, corresponding to the K available skills. If those skills are independently trained uni-modal policies, z dictates the policy to use during the following $\mathcal{T}$ time-steps. If the skills are encapsulated in a SNN, z is used in place of the latent variable. The architecture is depicted in Fig. 2.3.



$a \sim \mathcal{N}(\mu_{\theta_l}(s,z); \Sigma_{\theta_l}(s,z)) = \pi_{\theta_l}(\cdot|s,z)$
SNN
$z \sim Cat(\phi(s^M)) = \pi_{\theta_h}(\cdot|s)$
Manager Neural Network
$(s_{agent}, s^M_{rest}) = s^M$

Figure 2.3: Hierarchical SNN architecture to solve downstream tasks

The weights of the low level and high level neural networks could also be jointly optimized to adapt the skills to the task at hanad. This end-to-end training of

a policy with discrete latent variables in the Stochastic
Computation Graph could be done using straight-through estimators like the one proposed by [64] or [97]. Nevertheless, we show in our experiments that frozen low-level policies are already sufficient to achieve good performance in the studied downstream tasks, so these directions are left as future research.

Policy Optimization

For both the pre-training phase and the training of the high-level policies, we use Trust Region Policy Optimization (TRPO) as the policy optimization algorithm [138]. We choose TRPO due to its excellent empirical performance and because it does not require excessive hyperparameter tuning. The training on downstream tasks does not require any modifications, except that the action space is now the set of skills that can be used. For the pre-training phase, due to the presence of categorical latent variables, the marginal distribution of $\pi(a|s)$ is now a mixture of Gaussians instead of a simple Gaussian, and it may even become intractable to compute if we use more complex latent variables. To avoid this issue, we consider the latent code as part of the observation. Given that $\pi(a|s, z)$ is still a Gaussian, TRPO can be applied without any modification.

Algorithm 1: Skill training for SNNs with MI bonus

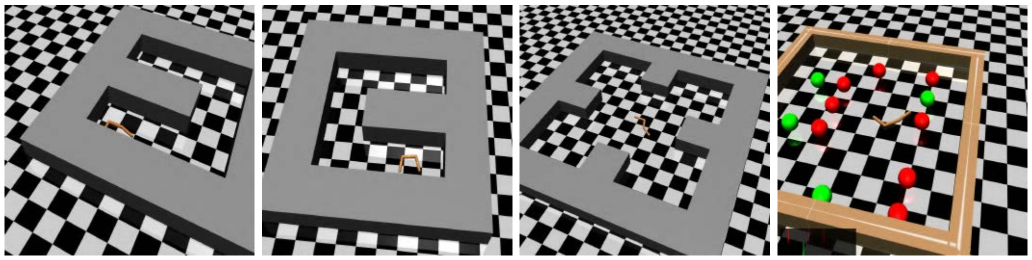
```
Initialize: Policy $\pi_\theta$; Latent dimension $K$;
while Not trained do
    for $n \leftarrow 1$ to $N$ do
        Sample $z_n \sim \text{Cat}\left(\frac{1}{K}\right)$;
        Collect rollout with $z_n$ fixed;
    end
    Compute $\hat{p}(Z = z|c) = \frac{m_c(z)}{\sum_{z'} m_c(z')}$;
    Modify $R_t^n \leftarrow R_t^n + \alpha_H \log \hat{p}(Z = z^n|c_t^n)$;
    Apply TRPO considering $z$ part of the observation;
end
```

2.5 Experimental Results for SNN4HRL

We have applied our framework to the two hierarchical tasks described in the benchmark by [30]: Locomotion + Maze and Locomotion + Food Collection (Gather). The observation space of these tasks naturally decompose into S_{agent} being the robot and S_{rest}^M the task-specific attributes like walls, goals, and sensor readings. Here we report the results using the Swimmer robot, also described in the benchmark paper. In fact, the swimmer locomotion task described therein corresponds exactly to our pretrain task, as we also solely reward speed in a plain environment.

To increase the variety of downstream tasks, we have constructed four different mazes. Maze 0 is the same as the one described in the benchmark [30] and Maze 1 is its reflection, where the robot has to go backwards-right-right instead of forward-left-left. These correspond to Figs. 2.4a-2.4b, where the robot is shown in its starting position and has to reach the other end of the U turn. Mazes 2 and 3 are different instantiations of the environment shown in Fig. 2.4c, where the goal has been placed in the North-East or in the South-West corner respectively. A reward of 1 is only granted when the robot reaches the goal position. In the Gather task depicted in Fig. 2.4d, the robot gets a reward of 1 for collecting green balls and a reward of -1 for the red ones, all of which are positioned randomly at the beginning of each episode. Apart from the robot joints positions and velocities, in these tasks the agent also receives LIDAR-like sensor readings of the distance to walls, goals or balls that are within a certain range.

In the benchmark of continuous control problems [30] it was shown that algorithms that employ naive exploration strategies could not solve them. More advanced intrinsically motivated explorations [62] do achieve some progress, and we report our stronger results with the exact same setting in Appendix A.1. Our hyperparameters for the neural network architectures and algorithms are detailed in the Appendix A.1 and the full code is available[4].



(a) Maze 0 (b) Maze 1 (c) Maze 2 or 3 (d) Food Gather

Figure 2.4: Illustration of the sparse reward tasks studied in SNN4HRL

We evaluate every step of the skill learning process, showing the relevance of the different pieces of our architecture and how they impact the exploration achieved when using them in a hierarchical fashion. Then we report the results[5] on the sparse environments described above. We seek to answer the following questions:

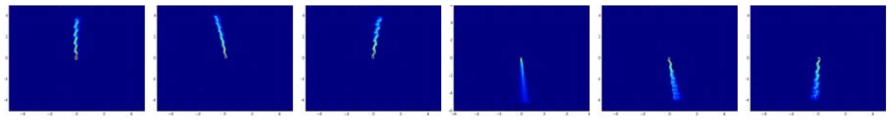
- Can the multimodality of SNNs and the MI bonus consistently yield a large span of skills?
- Can the pre-training experience improve the exploration in downstream environments?
- Does the enhanced exploration help to efficiently solve sparse complex tasks?

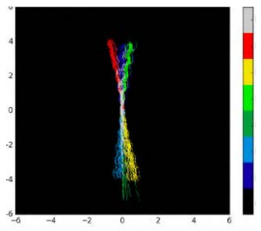
[4]Code available at: `https://github.com/florensacc/snn4hrl`

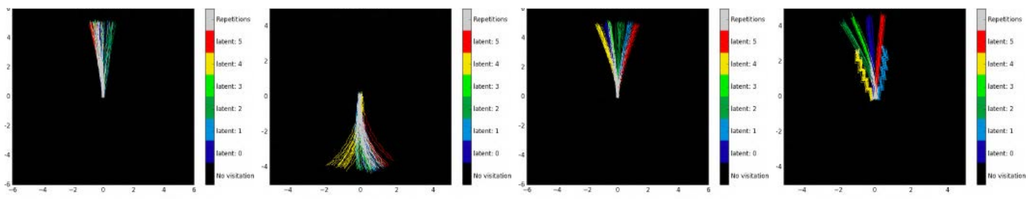
[5]Videos available at: `http://bit.ly/snn4hrl-videos`

Skill learning in pretrain

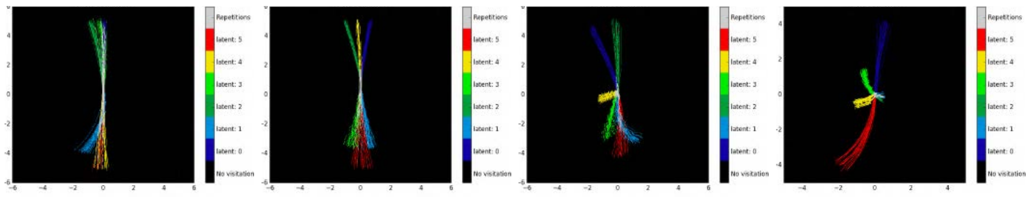
To evaluate the diversity of the learned skills we use "visitation plots", showing the (x, y) position of the robot's Center of Mass (CoM) during 100 rollouts of 500 time-steps each. At the beginning of every rollout, we reset the robot to the origin, always in the same orientation (as done during training). In Fig. 2.5a we show the visitation plot of six different feed-forward policies, each trained from scratch in our pre-training environment. For better graphical interpretation and comparison with the next plots of the SNN policies, Fig. 2.5b superposes a batch of 50 rollout for each of the 6 policies, each with a different color. Given the morphology of the swimmer, it has a natural preference for forward and backward motion. Therefore, when no extra incentive is added, the visitation concentrates heavily on the direction it is always initialized with. Note nevertheless that the proxy reward is general enough so that each independently trained policy yields a different way of advancing, hence granting a potentially useful skills to solve downstream tasks when embedded in our described Multi-policy hierarchical architecture.


(a) Independently trained policies in the pre-train MDP with the proxy reward of the CoM speed norm


(b) Superposed policy visitations from (a)



(c) SNN *without* bilinear integration and increasing $\alpha_H = 0, 0.001, 0.01, 0.1$



(d) SNN *with* bilinear integration and increasing $\alpha_H = 0, 0.001, 0.01, 0.1$

Figure 2.5: Span of skills learn by different methods and architectures

Next we show that, using SNNs, we can learn a similar or even larger span of skills without training several independent policies. The number of samples used to train a SNN is the same as a single feed-forward policy from Fig. 2.5a, therefore this method of learning skills effectively reduces the sample complexity by a factor equal to the numbers of skills being learned. With an adequate architecture and MI bonus, we show that the span of skills generated is also richer. In the two rows of Figs. 2.5c-2.5d we present the visitation plots of SNNs policies obtained for different design choices. Now the colors indicate the latent code that was sampled at the beginning of the rollout. We observe that each latent code generates a particular, interpretable behavior. Given that the initialization orientation is always the same, the different skills are truly distinct ways of moving: forward, backwards, or sideways. In the following we analyze the impact on the span of skills of the integration of latent variables (concatenation in first row and bilinear in the second) and the MI bonus (increasing coeficient α_H towards the right). Simple concatenation of latent variables with the observations rarely yields distinctive behaviors for each latent code sampled. On the other hand, we have observed that 80% of the trained SNNs with bilinear integration acquire at least forward and backward motion associated with different latent codes. This can be further improved to 100% by increasing α_H and we also observe that the MI bonus yields less overlapped skills and new advancing/turning skills, independently of the integration of the latent variables.

Hierarchical use of skills

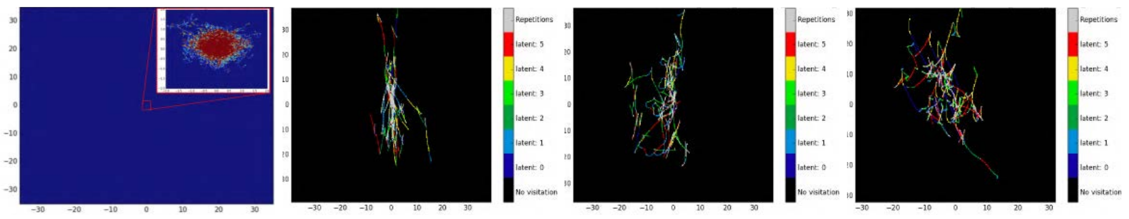
The hierarchical architectures we propose have a direct impact on the areas covered by random exploration. We will illustrate it with plots showing the visitation of the $(x - y)$ position of the Center of Mass (CoM) during single rollouts of one million steps, each generated with a different architecture.

On one hand, we show in Fig. 2.6a the exploration obtained with actions drawn from a Gaussian with $\mu = 0$ and $\Sigma = I$, similar to what would happen in the first iteration of training a Multi-Layer Perceptron with normalized random initialization of the weights. This noise is relatively large as the swimmer robot has actions clipped to $[-1, 1]$. Still, it does not yield good exploration as all point reached by the robot during the one million steps rollout lay within the $[-2, 2] \times [-2, 2]$ box around the initial position. This exploration is not enough to reach the first reward in most of the downstream sparse reward environments and will never learn, as already reported by Duan et al. [30].

On the other hand, using our hierarchical structures with pretrained policies introduced in Sec. 2.4 yields a considerable increase in exploration, as reported in Fig. 2.6b-2.6d. These plots also show a possible one millions steps rollout, but under a randomly initialized Manager Network, hence outputting uniformly distributed one-hot vectors every $\mathcal{T} = 500$ steps. The color of every point in the CoM trajectory corresponds to the latent code that was fixed at that time-step, clearly showing the “skill changes". In the following we describe what specific architecture was used for each plot.

The rollout in Fig. 2.6b is generated following a policy with the *Multi-policy* architecture. This hierarchical architecture is our strong baseline given that it has used six times more samples for pretrain because it trained six policies independently. As explained in Sec. 2.4, every $\mathcal{T} = 500$ steps it uses the sampled one-hot output of the Manager Network to select one of the six pre-trained policies. We observe that the exploration given by the *Multi-policy* hierarchy heavily concentrates around upward and downward motion, as is expected from the six individual pre-trained policies composing it (refer back to Fig. 2.5a).

Finally, the rollouts in Fig.2.6c and 2.6d use our hierarchical architecture with a SNN with bilinear integration and $\alpha_H = 0$ and 0.01 respectively. As described in Sec. 2.4, the placeholder that during the SNN training received the categorical latent code now receives the one-hot vector output of the Manager Network instead. The exploration obtained with SNNs yields a wider coverage of the space as the underlying policy usually has a larger span of skills. Note how most of the changes of latent code yield a corresponding change in direction. Our simple pre-train setup and the interpretability of the obtained skills are key advantages with respect to previous hierarchical structures.



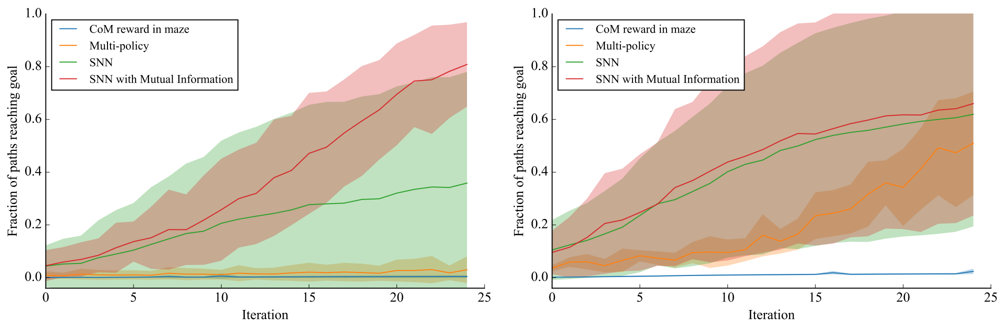
(a) Gaussian noise with covariance $\Sigma = I$ (b) Hierarchy with *Multi-policy* (c) Hierarchy with Bil-SNN $\alpha_H = 0$ (d) Hierarchy with Bil-SNN $\alpha_H = 0.01$

Figure 2.6: Visitation plots for different randomly initialized architectures (one rollout of 1M steps). All axis are in scale [-30,30] and we added a zoom for the Gaussian noise to scale [-2,2]

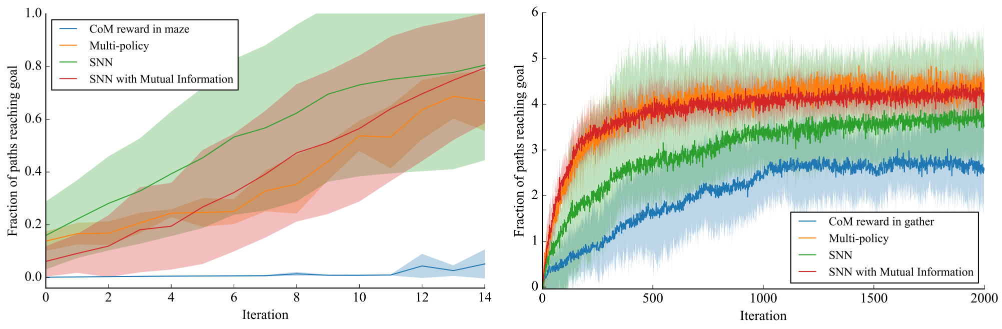
Mazes and Gather tasks

In Fig. 2.7 we evaluate the learning curves of the different hierarchical architectures proposed. Due to the sparsity of these tasks, none can be properly solved by standard reinforcement algorithms [30]. Therefore, we compare our methods against a better baseline: adding to the downstream task the same Center of Mass (CoM) proxy reward that was granted to the robot in the pre-training task. This baseline performs quite poorly in all the mazes, Fig. 2.7a-2.7c. This is due to the long time-horizon needed to reach the goal and the associated credit assignment problem. Furthermore, the proxy reward alone does not encourage diversity of actions as the MI bonus for SNN does. The proposed hierarchical architectures are able to learn much faster in every new MDP as they effectively shrink the time-horizon by aggregating time-steps into useful primitives. The benefit of using SNNs with bilinear integration in the hierarchy is also clear over most mazes, although pretraining with MI bonus does not always boost performance. Observing the learned trajectories that solve the mazes, we realize that the turning or sideway motions (more present in SNNs pretrained with MI bonus) help on maze 0, but are not critical in these tasks because the robot may use a specific forward motion against the wall of the maze to reorient itself. We think that the extra skills will shine more in other tasks requiring more precise navigation. And indeed this is the case in the Gather task as seen in Fig. 2.7d, where not only the average return increases, but also the variance of the learning curve is lower for the algorithm using SNN pretrained with MI bonus, denoting a more consistent learning across different SNNs. For more details on the Gather task, refer to Appendix A.1.

It is important to mention that each curve of SNN or Multi-policy performance corresponds to a total of 10 runs: 2 random seeds for 5 different SNNs obtained with 5 random seeds in the pretrain task. There is no cherry picking of the pre-trained policy to use in the sparse environment, as done in most previous work. This also explains the high variance of the proposed hierarchical methods in some tasks. This is particularly hard on the mazes as some pretrained SNN may not have the particular motion that allows to reach the goal, so they will have a 0 reward. See Appendix A.1 for more details.



(a) Maze 0 (b) Maze 1



(c) Aggregated results for Mazes 2 and 3 (d) Gather task

Figure 2.7: Faster learning of the SNN4HRL architectures in the sparse downstream MDPs

2.6 Efficient Adaptation with End-to-End Hierarchical Policy Gradients

Most methods in Hierarchical Reinforcement Learning, like the one presented in the previous section, still decouple the lower-level skill acquisition process and the training of a higher level that controls the skills in a new task. Leaving the skills fixed can lead to sub-optimality in the transfer setting.

One of the obstacles for training end-to-end hierarchical policies with policy gradient algorithms is that the gradient is considerably harder to compute. This is due to the fact that the intermediate decision taken by the higher level is not directly applied in the environment. Therefore, technically it should not be incorporated into the trajectory description as an observed variable, like the actions, and we should marginalize over this latent variable. In this section we first prove that, under mild assumptions, the hierarchical policy gradient can be accurately approximated without needing to marginalize over this latent variable. Then, we derive an unbiased baseline for the policy gradient that can reduce the variance of its estimate. With these findings, we present a new method, Hierarchical Proximal Policy Optimization (HiPPO), an on-policy algorithm for hierarchical policies, allowing learning at all levels of the policy end-to-end and preventing sub-policy collapse. We also propose a method for training time-abstractions that improves the robustness of the obtained skills to environment changes. Code and videos for the work presented in this section are available[6].

Approximate Hierarchical Policy Gradient

Policy gradient algorithms are based on the likelihood ratio trick [177] to estimate the gradient of returns with respect to the policy parameters as

$$\nabla_\theta \eta(\pi_\theta) = \mathbb{E}_\tau \big[\nabla_\theta \log P(\tau) R(\tau)\big] \approx \frac{1}{N}\sum_{i=1}^{n} \nabla_\theta \log P(\tau_i) R(\tau_i) \tag{2.3}$$

$$= \frac{1}{N}\sum_{i=1}^{n} \frac{1}{H}\sum_{t=1}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau_i) \tag{2.4}$$

In a temporal hierarchy, a hierarchical policy with a manager $\pi_{\theta_h}(z_t|s_t)$ selects every p time-steps one of n sub-policies to execute. These sub-policies, indexed by $z \in \mathbb{Z}_n$, can be represented as a single conditional probability distribution over actions $\pi_{\theta_l}(a_t|z_t, s_t)$. This allows us to not only use a given set of sub-policies, but also leverage skills learned with Stochastic Neural Networks (SNNs) [35]. Under this framework, the probability of a trajectory $\tau = (s_0, a_0, s_1, \ldots, s_H)$ can be written as

$$P(\tau) = \left(\prod_{k=0}^{H/p}\Big[\sum_{j=1}^{n} \pi_{\theta_h}(z_j|s_{kp}) \prod_{t=kp}^{(k+1)p-1} \pi_{\theta_l}(a_t|s_t, z_j)\Big]\right)\Big[P(s_0)\prod_{t=1}^{H} P(s_{t+1}|s_t, a_t)\Big]. \tag{2.5}$$

The mixture action distribution, which presents itself as an additional summation over skills, prevents additive factorization when taking the logarithm, as from Eq. 2.3 to 2.4. This can yield numerical instabilities due to the product of the p sub-policy probabilities. For instance, in the case where all the skills are distinguishable all the sub-policies' probabilities but one will have small values, resulting in an exponentially small value. In the following Lemma, we derive an approximation of the policy gradient, whose error tends to zero as the skills become more diverse, and draw insights on the interplay of the manager actions.

[6] `sites.google.com/view/hippo-rl`

Lemma 1. *If the skills are sufficiently differentiated, then the latent variable can be treated as part of the observation to compute the gradient of the trajectory probability. Let $\pi_{\theta_h}(z|s)$ and $\pi_{\theta_l}(a|s,z)$ be Lipschitz functions w.r.t. their parameters, and assume that $0 < \pi_{\theta_l}(a|s,z_j) < \epsilon \, \forall j \neq kp$, then*

$$\nabla_\theta \log P(\tau) = \sum_{k=0}^{H/p} \nabla_\theta \log \pi_{\theta_h}(z_{kp}|s_{kp}) + \sum_{t=0}^{H} \nabla_\theta \log \pi_{\theta_l}(a_t|s_t, z_{kp}) + \mathcal{O}(nH\epsilon^{p-1}) \tag{2.6}$$

Proof. See Appendix. □

Our assumption can be seen as having diverse skills. Namely, for each action there is just one sub-policy that gives it high probability. In this case, the latent variable can be treated as part of the observation to compute the gradient of the trajectory probability. Many algorithms to extract lower-level skills are based on promoting diversity among the skills [35, 31], therefore usually satisfying our assumption. We further analyze how well this assumption holds in our experiments section and Table 2.2.

Unbiased Sub-Policy Baseline

The policy gradient estimate obtained when applying the log-likelihood ratio trick as derived above is known to have large variance. A very common approach to mitigate this issue without biasing the estimate is to subtract a baseline from the returns [115]. It is well known that such baselines can be made state-dependent without incurring any bias. However, it is still unclear how to formulate a baseline for all the levels in a hierarchical policy, since an action dependent baseline does introduce bias in the gradient [169]. It has been recently proposed to use latent-conditioned baselines [176]. Here we go further and prove that, under the assumptions of Lemma 1, we can formulate an unbiased latent dependent baseline for the approximate gradient (Eq. 2.6).

Lemma 2. *For any functions $b_h : \mathcal{S} \to \mathbb{R}$ and $b_l : \mathcal{S} \times \mathcal{Z} \to \mathbb{R}$ we have:*

$$\mathbb{E}_\tau[\sum_{k=0}^{H/p} \nabla_\theta \log P(z_{kp}|s_{kp}) b_h(s_{kp})] = 0 \quad \textit{and} \quad \mathbb{E}_\tau[\sum_{t=0}^{H} \nabla_\theta \log \pi_{\theta_l}(a_t|s_t, z_{kp}) b_l(s_t, z_{kp})] = 0$$

Proof. See Appendix. □

Now we apply Lemma 1 and Lemma 2 to Eq. 2.3. By using the corresponding value functions as the function baseline, the return can be replaced by the Advantage function $A(s_{kp}, z_{kp})$ (see details in Schulman et al. [136]), and we obtain the following approximate policy gradient expression:

$$\hat{g} = \mathbb{E}_\tau\Big[(\sum_{k=0}^{H/p} \nabla_\theta \log \pi_{\theta_h}(z_{kp}|s_{kp}) A(s_{kp}, z_{kp})) + (\sum_{t=0}^{H} \nabla_\theta \log \pi_{\theta_l}(a_t|s_t, z_{kp}) A(s_t, a_t, z_{kp}))\Big]$$

This hierarchical policy gradient estimate can have lower variance than without baselines, but using it for policy optimization through stochastic gradient descent still yields an unstable algorithm. In the next section, we further improve the stability and sample efficiency of the policy optimization by incorporating techniques from Proximal Policy Optimization [137].

Algorithm 2: `HiPPO Rollout`

```
Input  : skills $\pi_{\theta_l}(a|s,z)$, manager $\pi_{\theta_h}(z|s)$, time-commitment bounds $P_{\min}$ and $P_{\max}$,
         horizon $H$
Reset environment: $s_0 \sim \rho_0$, $t = 0$.
while $t < H$ do
    Sample time-commitment $p \sim \texttt{Cat}([P_{\min}, P_{\max}])$ ;
    Sample skill $z_t \sim \pi_{\theta_h}(\cdot|s_t)$ ;
    for $t' = t \dots (t+p)$ do
        Sample action $a_{t'} \sim \pi_{\theta_l}(\cdot|s_{t'}, z_t)$ ;
        Observe new state $s_{t'+1}$ and reward $r_{t'}$ ;
    end
    $t \leftarrow t + p$ ;
end
Output : $(s_0, z_0, a_0, s_1, a_1, \dots, s_H, z_H, a_H, s_{H+1})$
```

Algorithm 3: HiPPO

```
Input  : skills $\pi_{\theta_l}(a|s,z)$, manager $\pi_{\theta_h}(z|s)$, horizon $H$, learning rate $\alpha$
while not done do
    for actor = 1, 2, ..., N do
        Obtain trajectory with HiPPO Rollout ;
        Estimate advantages $\hat{A}(a_{t'}, s_{t'}, z_t)$ and $\hat{A}(z_t, s_t)$ ;
    end
    $\theta \leftarrow \theta + \alpha \nabla_\theta L^{CLIP}_{HiPPO}(\theta)$ ;
end
```

Hierarchical Proximal Policy Optimization

Using an appropriate step size in policy space is critical for stable policy learning. Modifying the policy parameters in some directions may have a minimal impact on the distribution over actions, whereas small changes in other directions might change its behavior drastically and hurt training efficiency [69]. Trust region policy optimization (TRPO) uses a constraint on the KL-divergence between the old policy and the new policy to prevent this issue [138]. Unfortunately, hierarchical policies are generally represented by complex distributions without closed form expressions for the KL-divergence. Therefore, to improve the stability of our hierarchical policy gradient we turn towards Proximal Policy Optimization (PPO) [137]. PPO is a more flexible and compute-efficient algorithm. In a nutshell, it replaces the KL-divergence constraint with a cost function that achieves the same trust region benefits, but only requires the computation of the likelihood. Letting $w_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$, the PPO objective is:

$$L^{CLIP}(\theta) = \mathbb{E}_t \min \left\{ w_t(\theta) A_t,\ \texttt{clip}(w_t(\theta), 1-\epsilon, 1+\epsilon) A_t \right\}$$

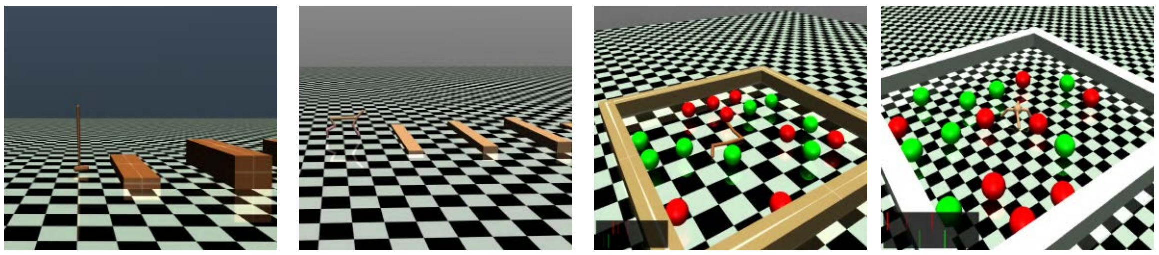

(a) Block Hopper (b) Block Half Cheetah (c) Snake Gather (d) Ant Gather

Figure 2.8: Environments used to evaluate the performance of HiPPO. Every episode has a different configuration: wall heights for (a)-(b), ball positions for (c)-(d)

We can adapt our approximated hierarchical policy gradient with the same approach by letting $w_{h,kp}(\theta) = \frac{\pi_{\theta_h}(z_{kp}|s_{kp})}{\pi_{\theta_{h,old}}(z_{kp}|s_{kp})}$ and $w_{l,t}(\theta) = \frac{\pi_{\theta_l}(a_t|s_t,z_{kp})}{\pi_{\theta_{l,old}}(a_t|s_t,z_{kp})}$, and using the super-index `clip` to denote the clipped objective version, we obtain the new surrogate objective:

$$L^{CLIP}_{HiPPO}(\theta) = \mathbb{E}_\tau \Big[\sum_{k=0}^{H/p} \min \left\{ w_{h,kp}(\theta) A(s_{kp}, z_{kp}), w^{\texttt{clip}}_{h,kp}(\theta) A(s_{kp}, z_{kp}) \right\}$$
$$+ \sum_{t=0}^{H} \min \left\{ w_{l,t}(\theta) A(s_t, a_t, z_{kp}), w^{\texttt{clip}}_{l,t}(\theta) A(s_t, a_t, z_{kp}) \right\} \Big]$$

We call this algorithm Hierarchical Proximal Policy Optimization (HiPPO). Next, we introduce a critical additions: a switching of the time-commitment between skills.

Varying Time-commitment

Most hierarchical methods either consider a fixed time-commitment to the lower level skills [35, 40], or implement the complex options framework [118, 6]. In this work we propose an in-between, where the time-commitment to the skills is a random variable sampled from a fixed distribution `Categorical`$(T_{\text{min}}, T_{\text{max}})$ just before the manager takes a decision. This modification does not hinder final performance, and we show it improves zero-shot adaptation to a new task. This approach to sampling rollouts is detailed in Algorithm 2. The full algorithm is detailed in Algorithm 3.

2.7 Experimental Results for HiPPO

We designed our experiments to answer the following questions:

1. How does HiPPO compare against a flat policy when learning from scratch?

2. Does it lead to policies more robust to environment changes?

3. How well does it adapt already learned skills?
4. Does our skill diversity assumption hold in practice?

Tasks

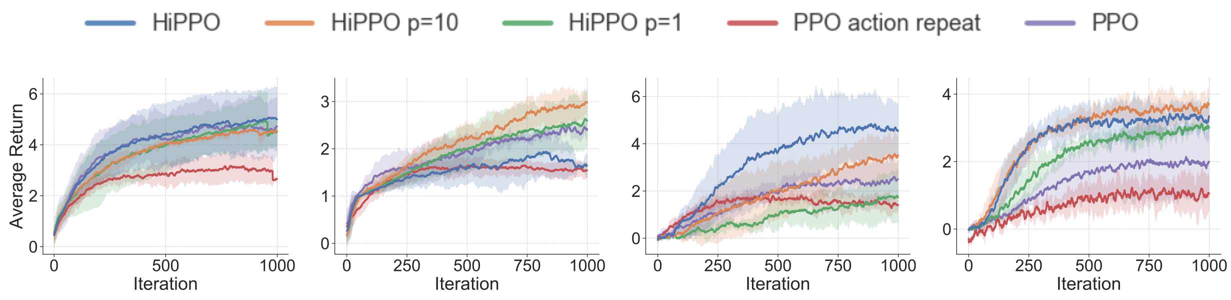
We evaluate our approach on a variety of robotic locomotion and navigation tasks. The Block environments, depicted in Fig. 2.8a-2.8b, have walls of random heights at regular intervals, and the objective is to learn a gait for the Hopper and Half-Cheetah robots to jump over them. The agents observe the height of the wall ahead and their proprioceptive information (joint positions and velocities), receiving a reward of +1 for each wall cleared. The Gather environments, described by Duan et al. [30], require agents to collect apples (green balls, +1 reward) while avoiding bombs (red balls, -1 reward). The only available perception beyond proprioception is through a LIDAR-type sensor indicating at what distance are the objects in different directions, and their type, as depicted in the bottom left corner of Fig. 2.8c-2.8d. This is challenging hierarchical task with sparse rewards that requires simultaneously learning perception, locomotion, and higher-level planning capabilities. We use the Snake and Ant robots in Gather. As described in the previous section, we apply the same partition of the state-space into $\mathcal{S}^M_{agent}$ and $\mathcal{S}^M_{rest}$. Details for all robotic agents are provided in Appendix A.2.

Learning from Scratch and Time-Commitment

In this section, we study the benefit of using our HiPPO algorithm instead of standard PPO on a flat policy [137]. The results, reported in Figure 2.9, demonstrate that training from scratch with HiPPO leads to faster learning and better performance than flat PPO. Furthermore, we show that the benefit of HiPPO does not just come from having temporally correlated exploration: PPO with action repeat converges at a lower performance than our method. HiPPO leverages the time-commitment more efficiently, as suggested by the poor performance of the ablation where we set $p = 1$, when the manager takes an action every environment step as well. Finally, Figure 2.10 shows the effectiveness of using the presented skill-dependent baseline.

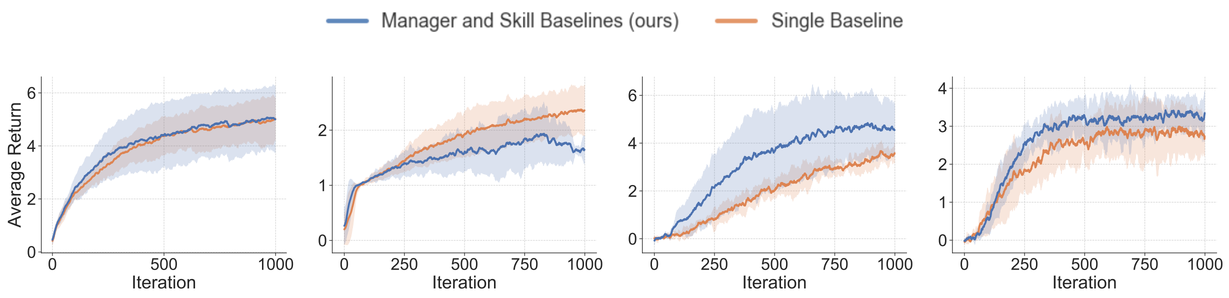
Comparison to Other Methods

We compare HiPPO to current state-of-the-art hierarchical methods. First, we evaluate HIRO [104], an off-policy RL method based on training a goal-reaching lower level policy. Fig. 2.11 shows that HIRO achieves poor performance on our tasks. As further detailed in Appendix A.2, this algorithm is sensitive to access to ground-truth information, like the exact (x, y) position of the robot in Gather. In contrast, our method is able to perform well directly from the raw sensory inputs described in Section 2.7. We evaluate Option-Critic [6], a variant of the options framework [157] that can be used for continuous action-spaces. It fails to learn, and we hypothesize that their algorithm provides less time-correlated exploration and learns less diverse skills. We also compare against MLSH [40], which repeatedly samples new environment configurations to learn primitive skills. We take these



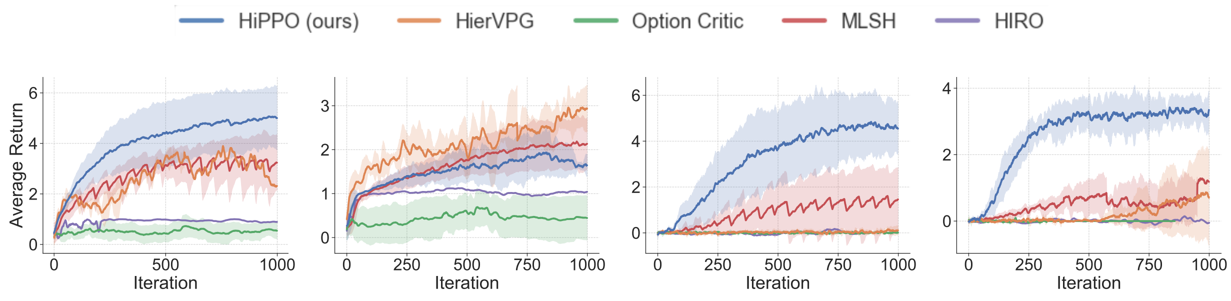
(a) Block Hopper (b) Block Half Cheetah (c) Snake Gather (d) Ant Gather

Figure 2.9: Analysis of different time-commitment strategies on learning from scratch.



(a) Block Hopper (b) Block Half Cheetah (c) Snake Gather (d) Ant Gather

Figure 2.10: Using a skill-conditioned baseline, as defined in Section 2.6, generally improves performance of HiPPO when learning from scratch.



(a) Block Hopper (b) Block Half Cheetah (c) Snake Gather (d) Ant Gather

Figure 2.11: Comparison of HiPPO and HierVPG to prior hierarchical methods on learning from scratch.

hyperparameters from their Ant Twowalk experiment: resetting the environment configuration every 60 iterations, a warmup period of 20 during which only the manager is trained, and a joint training period of 40 during which both manager and skills are trained. Our results show that such a training scheme does not provide any benefits. Finally, we provide a comparison to a direct application of our Hierarchical Vanilla Policy Gradient (HierVPG) algorithm, and we see that the algorithm is unstable without PPO's trust-region-like technique.

Robustness to Dynamics Perturbations

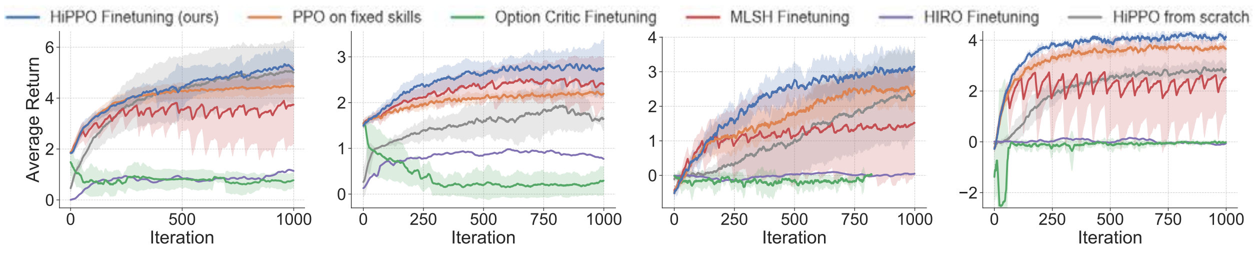
We investigate the robustness of HiPPO to changes in the dynamics of the environment. We perform several modifications to the base Snake Gather and Ant Gather environments. One at a time, we change the body mass, dampening of the joints, body inertia, and friction characteristics of both robots. The results, presented in Table 2.1, show that HiPPO with randomized period `Categorical`$([T_{\min}, T_{\max}])$ is able to better handle these dynamics changes. In terms of the drop in policy performance between the training environment and test environment, it outperforms HiPPO with fixed period on 6 out of 8 related tasks. These results suggest that the randomized period exposes the policy to a wide range of scenarios, which makes it easier to adapt when the environment changes.

Gather	Algorithm	Initial	Mass	Dampening	Inertia	Friction
Snake	Flat PPO	2.72	3.16 (+16%)	2.75 (+1%)	2.11 (-22%)	2.75 (+1%)
	HiPPO, $p = 10$	4.38	3.28 (-25%)	3.27 (-25%)	3.03 (-31%)	3.27 (-25%)
	HiPPO random p	5.11	**4.09** (-20%)	**4.03** (-21%)	**3.21** (-37%)	**4.03** (-21%)
Ant	Flat PPO	2.25	2.53 (+12%)	2.13 (-5%)	2.36 (+5%)	1.96 (-13%)
	HiPPO, $p = 10$	3.84	3.31 (-14%)	**3.37** (-12%)	2.88 (-25%)	**3.07** (-20%)
	HiPPO random p	3.22	**3.37** (+5%)	2.57 (-20%)	**3.36** (+4%)	2.84 (-12%)

Table 2.1: Zero-shot transfer performance. The final return in the initial environment is shown, as well as the average return over 25 rollouts in each new modified environment.

Adaptation of Pre-Trained Skills

For the Block task, we use DIAYN [31] to train 6 differentiated subpolicies in an environment without any walls. Here, we see if these diverse skills can improve performance on a downstream task that's out of the training distribution. For Gather, we take 6 pretrained subpolicies encoded by a Stochastic Neural Network [159] that was trained in a diversity-promoting environment [35]. We fine-tune them with HiPPO on the Gather environment, but with an extra penalty on the velocity of the Center of Mass. This can be understood as a preference for cautious behavior. This requires adjustment of the sub-policies, which were trained with a proxy reward encouraging them to move



(a) Block Hopper (b) Block Half Cheetah (c) Snake Gather (d) Ant Gather

Figure 2.12: Benefit of adapting some given skills when the preferences of the environment are different from those of the environment where the skills were originally trained. Adapting skills with HiPPO has better learning performance than leaving the skills fixed or learning from scratch.

as far as possible (and hence quickly). Fig. 2.12 shows that using HiPPO to simultaneously train a manager and fine-tune the skills achieves higher final performance than fixing the sub-policies and only training a manager with PPO. The two initially learn at the same rate, but HiPPO's ability to adjust to the new dynamics allows it to reach a higher final performance. Fig. 2.12 also shows that HiPPO can fine-tune the same given skills better than Option-Critic [6], MLSH [40], and HIRO [104].

Skill Diversity Assumption

In Lemma 1, we derived a more efficient and numerically stable gradient by assuming that the sub-policies are diverse. In this section, we empirically test the validity of our assumption and the quality of our approximation. We run the HiPPO algorithm on Ant Gather and Snake Gather both from scratch and with given pretrained skills, as done in the previous section. In Table 2.2, we report the average maximum probability under other sub-policies, corresponding to ϵ from the assumption. In all settings, this is on the order of magnitude of 0.1. Therefore, under the $p \approx 10$ that we use in our experiments, the term we neglect has a factor $\epsilon^{p-1} = 10^{-10}$. It is not surprising then that the average cosine similarity between the full gradient and our approximation is almost 1, as reported in Table 2.2.

2.8 Conclusions and Future Work

In this Chapter we have studied the use of hierarchical policies to tackle problems with very weak supervision, like sparse rewards. This type of supervision scales very well from the "providing" end, because sparse rewards are closer to how a non-expert operator could specify a task, without needing domain knowledge to shape the reward of every new task. There is still a lot of work to improve the scaling from the "optimization" end, or how to efficiently use this type of supervision. In this Chapter we introduced two new methods pushing the needle in that direction.

Gather	Algorithm	Cosine Sim.	$\max_{z' \neq z_{kp}} \pi_{\theta_l}(a_t \mid s_t, z')$	$\pi_{\theta_l}(a_t \mid s_t, z_{kp})$
Snake	HiPPO on given skills	0.98 ± 0.01	0.09 ± 0.04	0.44 ± 0.03
	HiPPO on random skills	0.97 ± 0.03	0.12 ± 0.03	0.32 ± 0.04
Ant	HiPPO on given skills	0.96 ± 0.04	0.11 ± 0.05	0.40 ± 0.08
	HiPPO on random skills	0.94 ± 0.03	0.13 ± 0.05	0.31 ± 0.09

Table 2.2: Empirical evaluation of Lemma 1. In the middle and right columns, we evaluate the quality of our assumption by computing the largest probability of a certain action under other skills (ϵ), and the action probability under the actual latent. We also report the cosine similarity between our approximate gradient and the exact gradient from Eq. 2.5. The mean and standard deviation of these values are computed over the full batch collected at iteration 10.

The first method, SNN4HRL, first learns a diverse set of skills using Stochastic Neural Networks trained with minimum supervision, and then utilizes these skills in a hierarchical architecture to solve the downstream tasks. This framework successfully combines two parts, firstly an unsupervised procedure to learn a large span of skills using proxy rewards and secondly a hierarchical structure that encapsulates the latter span of skills and allows to re-use them in future tasks. The span of skills learning can be greatly improved by using Stochastic Neural Networks as policies and their additional expressiveness and multimodality. The bilinear integration and the mutual information bonus are key to consistently yield a wide, interpretable span of skills. As for the hierarchical structure, our experiments demonstrate it can significantly boost the exploration of an agent in a new environment and we demonstrate its relevance for solving complex tasks as mazes or gathering.

The second method, HiPPO, reveals how to effectively adapt temporal hierarchies, overcoming one of the main limitations of SNN4HRL and many other HRL methods. We began by deriving a hierarchical policy gradient and its approximation. We then show that HiPPO can stably train multiple layers of a hierarchy jointly. The adaptation experiments suggest that we can optimize pretrained skills for downstream environments, as well as learn skills without any unsupervised pre-training. We also demonstrate that HiPPO with randomized period can learn from scratch on sparse-reward and long time horizon tasks, while outperforming non-hierarchical methods on zero-shot transfer.

Still, we have many different possible avenues of future work. We only used SNNs to encode our subpolicies for both experiments. How would we perform if we were provided n separate neural networks to finetune? We still need to explore this setting, as finetuning for SNNs has the problem where training one subpolicy will change the behavior of the other. While HiPPO does learn good emergent skills, their transferability is not guaranteed. As HiPPO is simply a policy architecture and gradient expression, we could explore using meta-learning on top in order to learn better skills that are more useful on a distribution of different tasks. Another avenue of research is having extended hierarchies with multiple layers of managers, each operating at broader and broader timescales.

Chapter 3

Automatic Curriculum Generation

3.1 Introduction

In the previous chapter we introduced two hierarchical reinforcement learning methods to be able to learn from the weak supervision of sparse rewards. The reusability of skills was key to speed up the learning on new tasks that shared a common agent but had otherwise arbitrarily different objectives or even different dynamics depending on the interaction with the environment. Nevertheless each new downstream task was still considered independently. The skills were re-used for all tasks, but the efficiency of learning a new downstream task could not leverage the training from the other downstream tasks.

In this chapter we focus on efficiently learning sets of tasks so that the total supervision needed to learn all tasks is minimized. The most natural question when trying to train an agent to perform well on a set of tasks is *whether there is an ordering of the tasks* that makes the process more efficient than randomly training on one task at a time. But the gain in sample complexity shouldn't come to the expense of more expert supervision designing this ordering. Therefore in this section we propose two different kinds of *automatic curriculum* that don't require any additional supervision and hence scale better to large sets of tasks.

Throughout this chapter we will consider *goal-conditioned tasks* that require an agent to reach a certain states or to manipulate objects into desired configurations. For example, we might want a robot to navigate to any positions in a room, moving objects to varying locations, or insert and turn a key in a lock. These goal-oriented tasks present a considerable challenge for reinforcement learning, since their natural reward function is sparse and prohibitive amounts of exploration are required to reach the goal and receive some learning signal. To tackle this challenge, we propose two methods that allows an agent to automatically discover the range of tasks that it is capable of performing in its environment.

Our first method uses a generator network to propose tasks for the agent to try to accomplish, each task being specified as reaching a certain parametrized subset of the state-space. The generator network is optimized using a Goal Generative Adversarial Network (Goal GAN), a variation of to the GANs introduced by Goodfellow et al. [47], to produce tasks that are always at the appropriate level of difficulty for the agent. We show that, by using this framework, an agent can efficiently and automatically learn to perform a wide set of tasks without requiring any prior knowledge of its environment, even when only sparse rewards are available. Videos and code available at: `https://sites.google.com/view/goalgeneration4rl`.

Our second approach is particularly geared towards reaching a single extremely hard-to-reach goal from anywhere, like when a seven DoF robotic arm has to robustly place a ring onto a peg, as shown in Fig. 3.8c. Therefore the “set of tasks" is now parameterized by the start-state rather than the goal-state, that is the same for all tasks. Past approaches tackle these problems by exploiting expert demonstrations or by manually designing a task-specific reward shaping function [109] to guide the learning agent. Instead, we propose a method to learn these tasks without requiring any prior knowledge other than obtaining a single state in which the task is achieved. The robot is trained in “reverse", gradually learning to reach the goal from a set of start states increasingly far from the goal. Our method automatically generates a curriculum of start states that adapts to the agent's performance, leading to efficient training on goal-oriented tasks. We demonstrate our approach on difficult simulated navigation and fine-grained manipulation problems, not solvable by state-of-the-art reinforcement learning methods.

3.2 Related Work

The problem that we are exploring has been referred to as “multi-task policy search" [26] or “contextual policy search," in which the task is viewed as the context for the policy [25, 32]. Unlike the work of Deisenroth et al. [26], our work uses a curriculum to perform efficient multi-task learning, even in sparse reward settings. In contrast to Fabisch and Metzen [32], which trains from a small number of discrete contexts / tasks, our method generates a training curriculum directly in continuous task space.

Curriculum-based approaches with manually designed schedules have been explored in supervised learning [13, 183, 12, 48] to split particularly complex tasks into smaller, easier-to-solve sub-problems. One particular type of curriculum learning explicitly enables the learner to reject examples which it currently considers too hard [80, 65]. This type of adaptive curriculum has mainly been applied to supervised tasks, and most practical curriculum approaches in RL rely on pre-specified task sequences [5, 72]. Some very general frameworks have been proposed to generate increasingly hard problems [132, 148], although challenges remain to apply the idea to difficult robotics tasks. A similar line of work uses intrinsic motivation based on learning progress to obtain “developmental trajectories" that focus on increasingly difficult tasks [10]. Nevertheless, their method requires iteratively partitioning the full task space, which strongly limits the application to fine-grain manipulation tasks like the ones presented in our work.

More recent work in using a curriculum for RL assumes that baseline performances for several tasks are given, and it uses them to gauge which tasks are the hardest (furthest behind the baseline) and require more training [140]. However, this framework can only handle finite sets of tasks and requires each task to be learnable on its own. On the other hand, our method trains a policy that generalizes to a set of continuously parameterized tasks, and it is shown to perform well even under sparse rewards by not allocating training effort to tasks that are too hard for the current performance of the agent.

Closer to our method of adaptively generating the tasks to train on, an interesting asymmetric self-play strategy has recently been proposed [152]. Contrary to our approach, which aims to generate and train on all tasks that are at the appropriate level of difficulty, the asymmetric component of their method can lead to biased exploration concentrating on only a subset of the tasks that are at the appropriate level of difficulty, as the authors and our own experiments suggests. This problem and their time-oriented metric of hardness may lead to poor performance in continuous state-action spaces, which are typical in robotics. Furthermore, their approach is designed as an exploration bonus for a single target task; in contrast, we define a new problem of efficiently optimizing a policy across a range of start states, which is considered relevant to improve generalization [120].

Our approach can be understood as sequentially composing locally stabilizing controllers by growing a tree of stabilized trajectories backwards from the goal state, similar to work done by Tedrake et al. [162]. This can be viewed as a "funnel" which takes start states to the goal state via a series of locally valid policies [16]. Unlike these methods, our approach does not require any dynamic model of the system. An RL counterpart, closer to our approach, is the work by Bagnell et al. [8], where a policy search algorithm in the spirit of traditional dynamic programming methods is proposed to learn a non-stationary policy: they learn what should be done in the last time-step and then "back it up" to learn the previous time-step and so on. Nevertheless, they require the stronger assumption of having access to baseline distributions that approximate the optimal state-distribution at every time-step.

The idea of directly influencing the start state distribution to accelerate learning in a Markov Decision Process (MDP) has drawn attention in the past. Kakade and Langford [70] studied the idea of exploiting the access to a 'generative model' [73] that allows training the policy on a fixed 'restart distribution' different from the one originally specified by the MDP. If properly chosen, this is proven to improve the policy training and final performance on the original start state distribution. Nevertheless, no practical procedure is given to choose this new distribution (only suggesting to use a more uniform distribution over states, which is what our baseline does), and they don't consider adapting the start state distribution during training, as we do. Other researchers have proposed to use expert demonstrations to improve learning of model-free RL algorithms, either by modifying the start state distribution to be uniform among states visited by the provided trajectories [117], or biasing the exploration towards relevant regions [151]. Our method works without any expert demonstrations, so we do not compare against these lines of research.

3.3 Goal-Reaching Policy Learning

Goal-parameterized Reward Functions

In the traditional RL framework, at each timestep t, the agent in state $s_t \in \mathcal{S} \subseteq \mathbb{R}^n$ takes an action $a_t \in \mathcal{A} \subseteq \mathbb{R}^m$, according to some policy $\pi(a_t \mid s_t)$ that maps from the current state s_t to a probability distribution over actions. Taking this action causes the agent to enter into a new state s_{t+1} according to a transition distribution $p(s_{t+1}|s_t, a_t)$, and receive a reward $r_t = r(s_t, a_t, s_{t+1})$. The objective of the agent is to find the policy π that maximizes the expected return, defined as the sum of rewards $R = \sum_{t=0}^{T} r_t$, where T is a maximal time given to perform the task. The learned policy corresponds to maximizing the expected return for a single reward function.

In our framework, instead of learning to optimize a single reward function, we consider a range of reward functions r^g indexed or parametrized by a goal $g \in \mathcal{G}$. Each goal g corresponds to a set of states $S^g \subset \mathcal{S}$ such that goal g is considered to be achieved when the agent is in any state $s_t \in S^g$. Then the objective is to learn a policy that, given any goal $g \in \mathcal{G}$, acts optimally with respect to r^g. We define a very simple reward function that measures whether the agent has reached the goal:

$$r^g(s_t, a_t, s_{t+1}) = \mathbb{1}\{s_{t+1} \in S^g\}\,, \tag{3.1}$$

where $\mathbb{1}$ is the indicator function. In our case, we use $S^g = \{s_t : d(f(s_t), g) \leq \epsilon\}$, where $f(\cdot)$ is a function that projects a state into goal space $\mathcal{G}$, $d(\cdot, \cdot)$ is a distance metric in goal space, and ϵ is the acceptable tolerance that determines when the goal is reached. However, our method can handle generic binary rewards (as in Eq. (3.1)) and does not require a distance metric for learning.

Furthermore, we define our MDP such that each episode terminates when $s_t \in S^g$. Thus, the return $R^g = \sum_{t=0}^{T} r_t^g$ is a binary random variable whose value indicates whether the agent has reached the set S^g in at most T time-steps. Policies $\pi(a_t \mid s_t, g)$ are also conditioned on the current goal g (as in Schaul et al. [129]). The expected return obtained when we take actions sampled from the policy can then be expressed as the probability of success on that goal within T time-steps, as shown in Eq. (3.2).

$$\begin{aligned} R^g(\pi) &= \mathbb{E}_{\pi(\cdot \mid s_t, g)}\, \mathbb{1}\big\{\exists\, t \in [1 \ldots T] : s_t \in S^g\big\} \\ &= \mathbb{P}\Big(\exists\, t \in [1 \ldots T] : s_t \in S^g \,\Big|\, \pi, g\Big) \end{aligned} \tag{3.2}$$

The sparse indicator reward function of Eq. (3.1) is not only simple but also represents a property of many real-world goal problems: in many settings, it may be difficult to tell whether the agent is getting closer to achieving a goal, but easy to tell when a goal has been achieved (e.g. in a maze). In theory, one could hand-engineer a meaningful distance function for each task that could be used to create a dense reward function. Instead, our method is able to learn simply using the indicator function of Eq. (3.1).

Overall Objective

We desire to find a policy $\pi(a_t \mid s_t, g)$ that achieves a high reward for many goals g. We assume that there is a test distribution of goals $p_g(g)$ that we would like to perform well on. For simplicity, we assume that the test distribution samples goals uniformly from the set of goals $\mathcal{G}$, although in practice any distribution can be used. The overall objective is then to find a policy π^* such that

$$\pi^*(a_t \mid s_t, g) = \arg\max_{\pi} \mathbb{E}_{g \sim p_g(\cdot)}\, R^g(\pi)\,. \tag{3.3}$$

Recall from Eq. (3.2) that $R^g(\pi)$ is the probability of success for each goal g. Thus the objective of Eq. (3.3) measures the average probability of success over all goals sampled from $p_g(g)$. We refer to the objective in Eq. (3.3) as the *coverage*.

3.4 Automatic Goal Generation for Reinforcement Learning Agents

In order to efficiently maximize the objective from the previous section, the algorithm must intelligently choose which goals to focus on at every training stage: goals should be at the appropriate level of difficulty for the current policy. To do so, our algorithm allows an agent to generate its own reward functions, defined with respect to target subsets of the state space, called goals. We generate such goals using a Goal Generative Adversarial Network (Goal GAN), a variation of to the GANs introduced by Goodfellow et al. [47]. A goal discriminator is trained to evaluate whether a goal is at the appropriate level of difficulty for the current policy, and a goal generator is trained to generate goals that meet this criteria. We show that such a framework allows an agent to quickly learn a policy that reaches all feasible goals in its environment, with no prior knowledge about the environment or the tasks being performed. The method described in this section automatically creates a curriculum, in which, at each step, the generator generates goals that are only slightly more difficult than the goals that the agent already knows how to achieve.

Similar to previous work [129, 81, 32, 26], we require a continuous goal-space representation such that a goal-conditioned policy can efficiently generalize over the goals. In particular, we assume that:

1. A policy trained on a sufficient number of goals in some area of the goal-space will learn to interpolate to other goals within that area.
2. A policy trained on some set of goals will provide a good initialization for learning to reach close-by goals, meaning that the policy can occasionally reach them but maybe not consistently.

Furthermore, we assume that if a goal is reachable, there exists a policy that does so reliably. This is a reasonable assumption for any practical robotics problem, and it will be key for our method, as it strives to train on every goal until it is consistently reached.

The approach described in this section can be broken down into three parts: First, we label a set of goals based on whether they are at the appropriate level of difficulty for the current policy. Second, using these labeled goals, we train a generator to output new goals at the appropriate level of difficulty. Finally, we use these new goals to efficiently train the policy, improving its coverage objective. We iterate through each of these steps until the policy converges.

Goal Labeling

As shown in our experiments, sampling goals from $p_g(g)$ directly, and training our policy on each sampled goal may not be the most sample efficient way to optimize the coverage objective of Eq. (3.3). Instead, we modify the distribution from which we sample goals during training to be uniform over the set of *Goals of Intermediate Difficulty* (*GOID*):

$$GOID_i := \{g : R_{\min} \leq R^g(\pi_i) \leq R_{\max}\} \subseteq \mathcal{G}. \tag{3.4}$$

The justification for this is as follows: due to the sparsity of the reward function, for most goals g, the current policy π_i (at iteration i) obtains no reward. Instead, we wish to train our policy on goals g for which π_i is able to receive some minimum expected return $R^g(\pi_i) > R_{\mathrm{m}in}$ such that the agent receives enough reward signal for learning. On the other hand, with this single restriction, we might sample repeatedly from a small set of already mastered goals. To force our policy to train on goals that still need improvement, we also ask for $R^g(\pi_i) \leq R_{\max}$, where $R_{\max}$ is a hyperparameter setting a maximum level of performance above which we prefer to concentrate on new goals. Note that from Eq. (3.2), $R_{\mathrm{m}in}$ and R_{max} can be interpreted as a minimum and maximum probability of reaching a goal over T time-steps. Training our policy on goals in $GOID_i$ allows us to efficiently maximize the coverage objective of Eq. (3.3). The justification for this is as follows: due to the sparsity of the reward function, for most goals g, the current policy π_i (at iteration i) obtains no reward. Instead, we wish to train our policy on goals g for which π_i is able to receive some minimum expected return $R^g(\pi_i) > R_{\mathrm{m}in}$ such that the agent receives enough reward signal for learning. On the other hand, with this single restriction, we might sample repeatedly from a small set of already mastered goals. To force our policy to train on goals that still need improvement, we also ask for $R^g(\pi_i) \leq R_{\max}$, where $R_{\max}$ is a hyperparameter setting a maximum level of performance above which we prefer to concentrate on new goals. Note that from Eq. (3.2), $R_{\mathrm{m}in}$ and R_{max} can be interpreted as a minimum and maximum probability of reaching a goal over T time-steps. Training our policy on goals in $GOID_i$ allows us to efficiently maximize the coverage objective of Eq. (3.3). Therefore, we need to approximate the sampling from $GOID_i$. We propose to first estimate the label $y_g \in \{0, 1\}$ that indicates whether $g \in GOID_i$ for all goals g used in the previous training iteration, and then use these labels to train a generative model from where we can sample goals to train on the next iteration. We estimate the label of a goal g by computing the fraction of success among all trajectories that had this goal g during the previous training iteration, and then check whether this estimate is in between $R_{\mathrm{m}in}$ and R_{max}. In all our experiments we use 0.1 and 0.9 respectively, although the algorithm is very robust to these hyperparameters (any value of $R_{\mathrm{m}in} \in (0, 0.25)$ and $R_{max} \in (0.75, 1)$ would yield basically the same result, as shown in Appendix A.3)

Adversarial Goal Generation

In order to sample new goals g uniformly from $GOID_i$, we introduce an adversarial training procedure called "goal GAN", which is a modification of the procedure used for training Generative Adversarial Networks (GANs) [47]. The modification allows us to train the generative model both with positive examples from the distribution we want to approximate and negative examples sampled from a distribution that does not share support with the desired one. This improves the accuracy of the generative model despite being trained with few positive samples. Our choice of GANs for goal generation is motivated both from this training from negative examples, as well as their ability to generate very high dimensional samples such as images [47] which is important for scaling up our approach to goal generation in high-dimensional goal spaces. Other generative models like Stochastic Neural Networks [159] don't accept negative examples, and don't scale to higher dimensions.

We use a “goal generator" neural network $G(z)$ to generate goals g from a noise vector z. We train $G(z)$ to uniformly output goals in $GOID_i$ using a second “goal discriminator" network $D(g)$. The latter is trained to distinguish goals that are in $GOID_i$ from goals that are not in $GOID_i$. We optimize our $G(z)$ and $D(g)$ in a manner similar to that of the Least-Squares GAN (LSGAN) [100], which we modify by introducing the binary label y_g allowing us to train from “negative examples" when $y_g = 0$:

$$\begin{aligned} &\min_D V(D) = \mathbb{E}_{g\sim p_{\mathrm{data}}(g)}\left[y_g(D(g)-b)^2\ + (1-y_g)(D(g)-a)^2\right]\ + \mathbb{E}_{z\sim p_z(z)}[(D(G(z))-a)^2] \\ &\min_G V(G) = \mathbb{E}_{z\sim p_z(z)}[D(G(z))-c)^2] \end{aligned} \tag{3.5}$$

We directly use the original hyperparameters reported in Mao et al. [100] in all our experiments (a = -1, b = 1, and c = 0). The LSGAN approach gives us a considerable improvement in training stability over vanilla GAN, and it has a comparable performance to WGAN [4]. However, unlike in the original LSGAN paper [100], we have three terms in our value function $V(D)$ rather than the original two. For goals g for which $y_g = 1$, the second term disappears and we are left with only the first and third terms, which are identical to that of the original LSGAN framework. Viewed in this manner, the discriminator is trained to discriminate between goals from $p_{\mathrm{data}}(g)$ with a label $y_g = 1$ and the generated goals $G(z)$. Looking at the second term, our discriminator is also trained with “negative examples" with a label $y_g = 0$ which our generator should not generate. The generator is trained to “fool" the discriminator, i.e. to output goals that match the distribution of goals in $p_{\mathrm{data}}(g)$ for which $y_g = 1$.

Policy Optimization

Algorithm 4: Generative Goal Learning

```
Input  : Policy $\pi_0$
Output : Policy $\pi_N$
$(G, D) \leftarrow$ initialize_GAN()
$goals_{\mathrm{old}} \leftarrow \varnothing$
for $i \leftarrow 1$ to $N$ do
    $z \leftarrow$ sample_noise($p_z(\cdot)$);
    $goals \leftarrow G(z) \cup$ sample($goals_{\mathrm{old}}$);
    $\pi_i \leftarrow$ update_policy($goals, \pi_{i-1}$);
    $returns \leftarrow$ evaluate_policy($goals, \pi_i$);
    $labels \leftarrow$ label_goals($returns$)
    $(G, D) \leftarrow$ train_GAN($goals, labels, G, D$);
    $goals_{\mathrm{old}} \leftarrow$ update_replay($goals$)
end
```

Our full algorithm for training a policy $\pi(a_t \mid s_t, g)$ to maximize the coverage objective in Eq. (3.3) is shown in Algorithm 4. At each iteration i, we generate a set of goals by first using `sample_noise` to obtain a noise vector z from $p_z(\cdot)$ and then passing this noise to the generator $G(z)$. We use these goals to train our policy using RL, with the reward function given by Eq. (3.1) (`update_policy`). Any RL algorithm can be used for training; in our case we use TRPO with GAE [136]. Our policy's empirical performance on these goals (`evaluate_policy`) is used to determine each goal's label y_g (`label_goals`), as described in Section 3.4. Next, we use these labels to train our goal generator and our goal discriminator (`train_GAN`), as described in Section 3.4. The generated goals from the previous iteration are used to compute the Monte Carlo estimate of the expectations with respect to the distribution $p_{data}(g)$ in Eq. (3.5). By training on goals within $GOID_i$ produced by the goal generator, our method efficiently finds a policy that optimizes the coverage objective. For details on how we initialize the goal GAN (`initialize_GAN`), and how we use a replay buffer to prevent "catastrophic forgetting" (`update_replay`), see Appendix A.3.

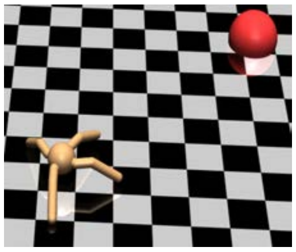
The algorithm described above naturally creates a curriculum. The goal generator is updated along with the policy to generate goals in $GOID_i$, for which our current policy π_i obtains an intermediate level of return. Thus such goals are always at the appropriate level of difficulty. However, the curriculum occurs as a by-product via our optimization, without requiring any prior knowledge of the environment or the tasks that the agent must perform.

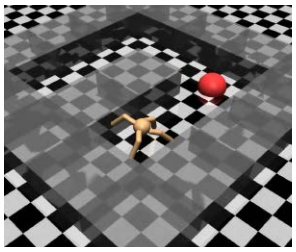
3.5 Experimental Results for goalGAN

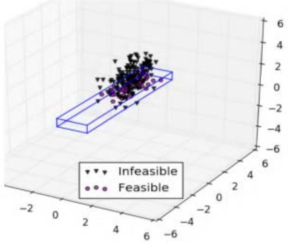
In this section we provide the experimental results to answer the following questions:

- Does our automatic curriculum yield faster maximization of the coverage objective?
- Does our Goal GAN dynamically shift to sample goals of the appropriate difficulty (i.e. in $GOID_i$)?
- Can our Goal GAN track complex multimodal goal distributions $GOID_i$?
- Does it scale to higher-dimensional goal-spaces with a low-dimensional space of feasible goals?

To answer the first two questions, we demonstrate our method in two challenging robotic locomotion tasks, where the goals are the (x, y) position of the Center of Mass (CoM) of a dynamically complex quadruped agent. In the first experiment the agent has no constraints (see Fig. 3.1a) and in the second one the agent is inside a U-maze (see Fig. 3.1b). To answer the third question, we train a point-mass agent to reach any point within a multi-path maze (see Fig. 3.1d). To answer the final question, we study how our method scales with the dimension of the goal-space in an environment where the feasible region is kept of approximately constant volume in an embedding space that grows in dimension (see Fig. 3.1c for the 3D case). We compare our *Goal GAN* method against four baselines. *Uniform Sampling* is a method that does not use a curriculum at all, training at every iteration on goals uniformly sampled from the goal-space. To demonstrate that a straight-forward distance reward can be prone to local minima, *Uniform Sampling with L2 loss* samples goals in the same fashion as the first baseline, but instead of the indicator reward that our method uses, it receives the negative L2 distance to the goal as a reward at every step. We have also adapted two methods from the literature to our setting: Asymmetric Self-play [152] and SAGG-RIAC [9]. Finally, we


(a) Free Ant Locomotion


(b) Maze Ant Locomotion



(c) Point-mass 3D

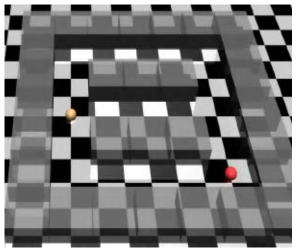

(d) Multi-path point-mass

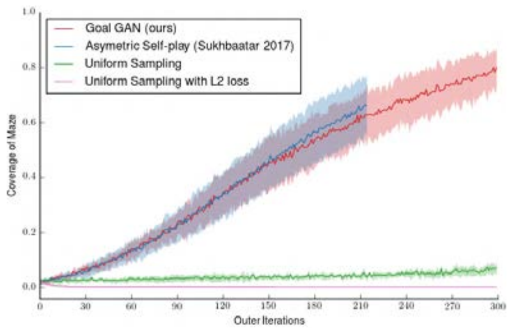
Figure 3.1: In 3.1a-3.1d, the red areas are goals reachable by the orange agent. In 3.1c any point within the blue frame is a feasible goal (purple balls) and the rest are unfeasible (black triangles).

provide an ablation and an oracle for our method to better understand the importance of sampling goals of intermediate difficulty $g \in GOID_i$. The ablation *GAN fit all* consists on training the GAN not only on the goals $g \in GOID_i$ but rather on every goal attempted in the previous iteration. Given the noise injected at the output of the GAN this generates a gradually expanding set of goals - similar to any hand-designed curriculum. The oracle consists in sampling goals uniformly from the feasible state-space, but only keeping them if they satisfy the criterion in Eq. (3.4) defining $GOID_i$. This *Rejection Sampling* method is orders of magnitude more expensive in terms of labeling, but it serves to estimate an upper-bound for our method in terms of performance.

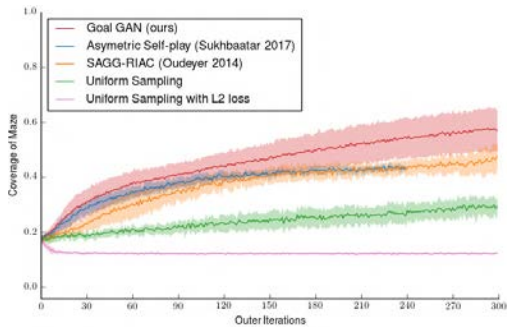
Ant Locomotion

We test our method in two challenging environments of a complex robotic agent navigating either a free space (Free Ant, Fig. 3.1a) or a U-shaped maze (Maze Ant, Fig. 3.1b). Duan et al. [30] describe the task of trying to reach the other end of the U-turn, and they show that standard RL methods are unable to solve it. We further extend the task to ask to be able to reach any given point within the maze, or within the $[-5, 5]^2$ square for Free Ant. The reward is still a sparse indicator function being 1 only when the (x, y) CoM of the Ant is within $\epsilon = 0.5$ of the goal. Therefore the goal space is 2 dimensional, the state-space is 41 dimensional, and the action space is 8 dimensional (see Appendix A.3).

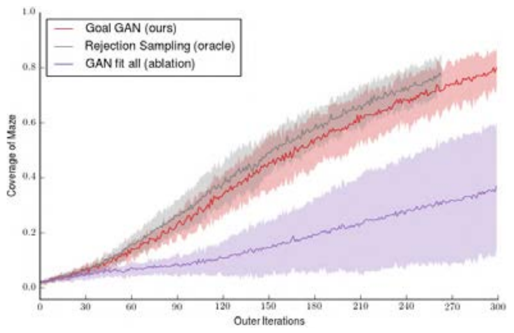
We first explore whether, by training on goals that are generated by our Goal GAN, we are able to improve our policy's training efficiency, compared to the baselines described above. In Figs. 3.2a-Fig. 3.2b we see that our method leads to faster training compared to the baselines. The *Uniform Sampling* baseline does very poorly because too many samples are wasted attempting to train on goals that are infeasible or not reachable by the current policy - hence not receiving any learning signal. If an L2 loss is added to try to guide the learning, the agent falls into a poor local optima of not moving to avoid further negative rewards. The two other baselines that we compare against perform better, but still do not surpass the performance of our method. In particular, Asymmetric Self-play needs to train the goal-generating policy (Alice) at every outer iteration, with



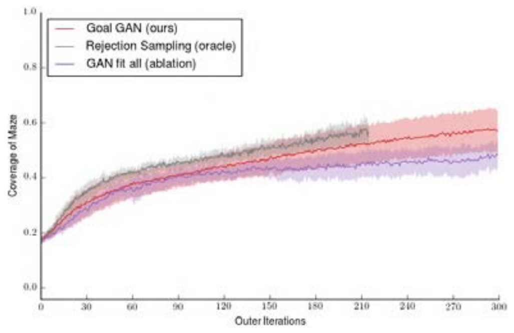
(a) Free Ant - Baselines



(b) Maze Ant - Baselines



(c) Free Ant - Variants

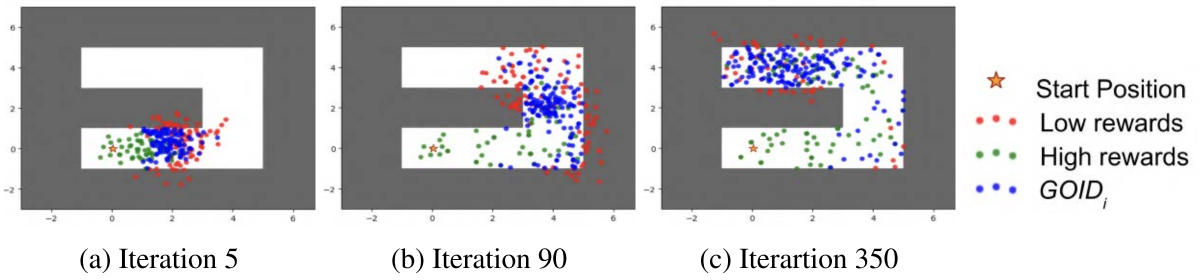


(d) Maze Ant - Variants

Figure 3.2: Learning curves comparing the training efficiency of our Goal GAN method and different baselines (first row) and variants (second row), for the Free Ant (left column) and the Maze Ant (right column). The y-axis indicates the average return over all feasible goals. The x-axis shows the number of times that new goals have been sampled. All plots average over 10 random seeds.

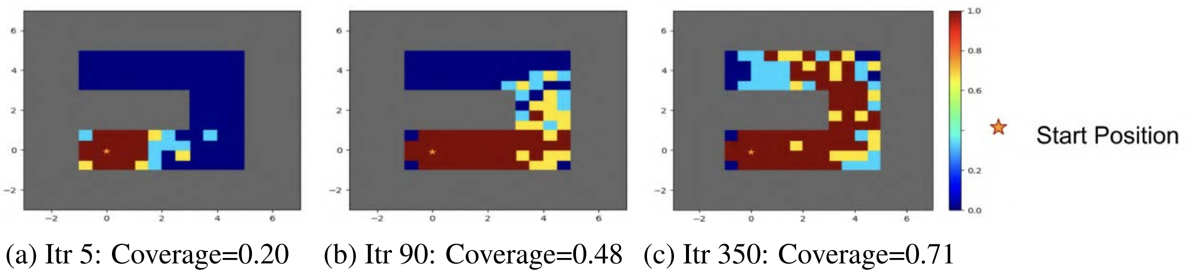
an amount of rollouts equivalent to the ones used to train the goal-reaching policy. This additional burden is not represented in the plots, being therefore at least half as sample-efficient as the plots indicate. SAGG-RIAC maintains an ever-growing partition of the goal-space that becomes more and more biased towards areas that already have more sub-regions, leading to reduced exploration and slowing down the expansion of the policy's capabilities. Details of our adaptation of these two methods to our problem, as well as further study of their failure cases, is provided in the Appendices A.3 and A.3.

To better understand the efficiency of our method, we analyze the goals generated by our automatic curriculum. In these Ant navigation experiments, the goal space is two dimensional, allowing us to study the shift in the probability distribution generated by the Goal GAN (Fig. 3.3 for the Maze Ant) along with the improvement of the policy coverage (Fig. 3.4 for the Maze Ant). We have indicated the difficulty of reaching the generated goals in Fig. 3.3. It can be observed in these figures that the location of the generated goals shifts to different parts of the maze, concentrating on the area where the current policy is receiving some learning signal but needs more improvement. The percentage of generated goals that are at the appropriate level of difficulty (in $GOID_i$) stays around 20% even as the policy improves. The goals in these figures include a mix of newly generated goals from the Goal GAN as well as goals from previous iterations that we use to prevent our policy from “forgetting" (Appendix A.3). Overall it is clear that our Goal GAN dynamically shift to sample goals of the appropriate difficulty. See Appendix A.3 and Fig. A.11-A.12 therein for the analogous analysis of Free Ant, where we observe that Goal GAN produces a growing ring of goals around the origin.



(a) Iteration 5 (b) Iteration 90 (c) Iterartion 350

Figure 3.3: Goals that, at iterations i, our algorithm trains on - 200 sampled from Goal GAN, 100 from replay. Green goals satisfy $\bar{R}^g(\pi_i) \geq R_{\max}$. Blue ones have appropriate difficulty for the current policy $R_{\min} \leq \bar{R}^g(\pi_i) \leq R_{\max}$. The red ones have $R_{\min} \geq \bar{R}^g(\pi_i)$.



(a) Itr 5: Coverage=0.20 (b) Itr 90: Coverage=0.48 (c) Itr 350: Coverage=0.71

Figure 3.4: Visualization of the policy performance (same policy training as in Fig. 3.3). For illustration purposes, each grid cell is colored according to the expected return achieved when fixing its center as goal: Red indicates 100% success; blue indicates 0% success.

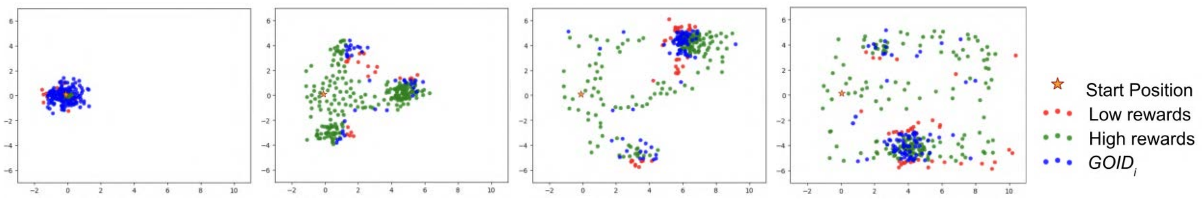
It is interesting to analyze the importance of generating goals in $GOID_i$ for efficient learning. This is done in Figs. 3.2c-3.2d, where we first show an ablation of our method *GAN fit all*, that disregards the labels. This method performs worse than ours, because the expansion of the goals is not related to the current performance of the policy. Finally, we study the *Rejection Sampling* oracle. As explained in Section 3.4, we wish to sample from the set of goals $GOID_i$, which we approximate by fitting a Goal GAN to the distribution of good goals observed in the previous policy optimization step. We evaluate now how much this approximation affects learning by comparing the learning performance of our Goal GAN to a policy trained on goals sampled uniformly from

$GOID_i$ by using rejection sampling. This method is orders of magnitude more sample inefficient, but gives us an upper bound on the performance of our method. Figs. 3.2c-3.2d demonstrate that our performance is quite close to the performance of this much less efficient baseline.

Multi-path point-mass maze

In this section we show that our Goal GAN method is efficient at tracking clearly multi-modal distributions of goals $g \in GOID_i$. To this end, we introduce a new maze environment with multiple paths, as can be seen in Fig. 3.1d. To keep the experiment simple we replace the Ant agent by a point-mass, which actions are the velocity vector (2 dim). As in the other experiments, our aim is to learn a policy that can reach any feasible goal corresponding to ϵ-balls in state space, like the one depicted in red.

Similar to the experiments in Figures 3.3 and 3.4, here we show the goals that our algorithm generated to train the Mutli-path point-mass agent. Figures 3.5 and 3.6 show the results. It can be observed that our method produces a multi-modal distribution over goals, tracking all the areas where goals are at the appropriate level of difficulty. Note that the samples from the regularized replay buffer are responsible for the trailing spread of “High Reward" goals and the Goal GAN is responsible for the more concentrated nodes (see only Goal GAN samples in Appendix Fig. A.13a). A clear benefit of using our Goal GAN as a generative model is that no prior knowledge about the distribution to fit is required (like the number of modes). Finally, having several possible paths to reach a specific goal does not hinder the learning of our algorithm that consistently reaches full coverage in this problem (see Appendix Fig. A.13b).

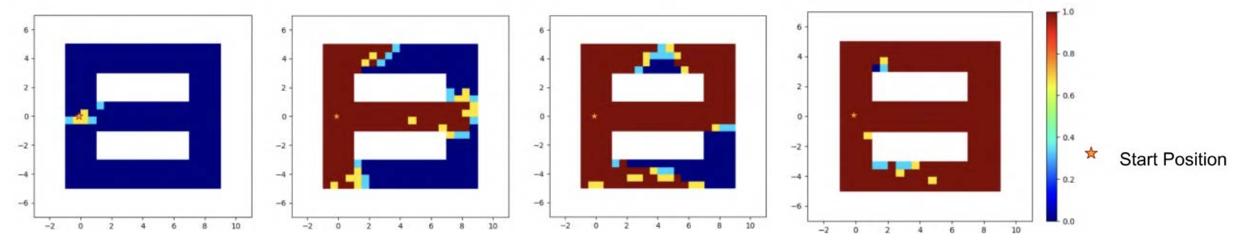


(a) Iteration 1 (b) Iteration 10 (c) Iteration 30 (d) Iteration 100

Figure 3.5: Goals that, at iterations i, our algorithm trains on - 200 sampled from Goal GAN, 100 from replay. Green goals satisfy $\bar{R}^g(\pi_i) \geq R_{\max}$. Blue ones have appropriate difficulty for the current policy $R_{\min} \leq \bar{R}^g(\pi_i) \leq R_{\max}$. The red ones have $R_{\min} \geq \bar{R}^g(\pi_i)$.

N-dimensional Point Mass

In many real-world RL problems, the set of feasible states is a lower-dimensional subset of the full state space, defined by the constraints of the environment. For example, the kinematic constraints of a robot limit the set of feasible states that the robot can reach. In this section we use an N-



(a) Itr 1: Coverage=0.014 (b) Itr 10: Coverage=0.53 (c) Itr 30: Coverage=0.78 (d) Itr 100: Coverage=0.98

Figure 3.6: Visualization of the policy performance (same policy training as in Fig. 3.5). For illustration purposes, each grid cell is colored according to the expected return achieved when fixing its center as goal: Red indicates 100% success; blue indicates 0% success.

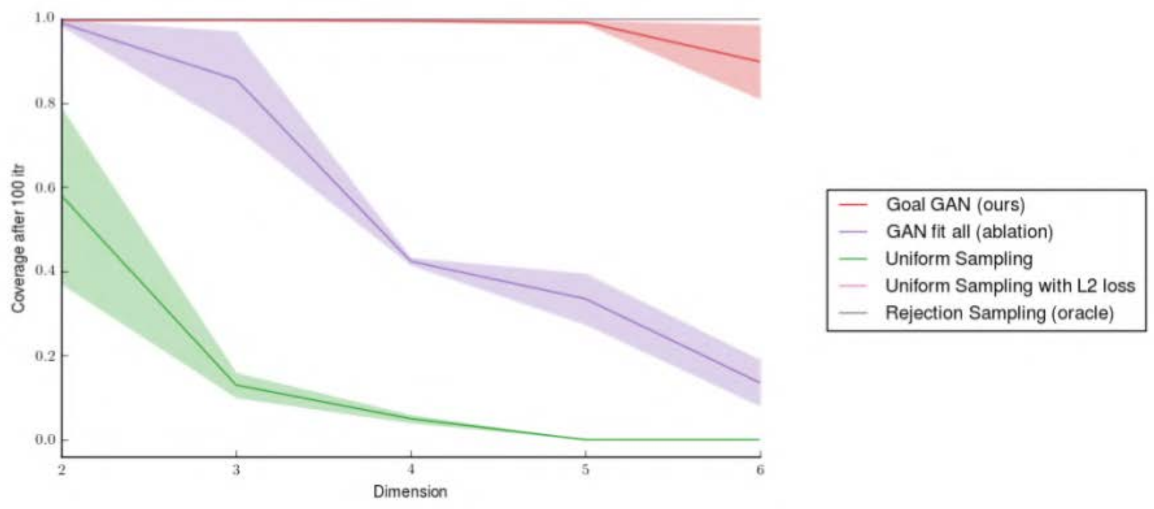


Figure 3.7: Final goal coverage obtained after 200 outer iterations on the N-dim point mass environment. All plots average over 5 random seeds.

dimensional Point Mass to demonstrate the performance of our method as the embedding dimension increases.

In this experiments, the full state-space of the N-dimensional Point Mass is the hypercube $[-5, 5]^N$. However, the Point Mass can only move within a small subset of this state space. In the two-dimensional case, the set of feasible states corresponds to the $[-5, 5] \times [-1, 1]$ rectangle, making up 20% of the full space. For $N > 2$, the feasible space is the Cartesian product of this 2D strip with $[-\epsilon, \epsilon]^{N-2}$, where $\epsilon = 0.3$. In this higher-dimensional environment, our agent receives a reward of 1 when it moves within $\epsilon_N = 0.3\frac{\sqrt{N}}{\sqrt{2}}$ of the goal state, to account for the increase in average $L2$ distance between points in higher dimensions. The fraction of the volume of the feasible space decreases as N increases (e.g. 0.00023:1 for $N = 6$).

We compare the performance of our method to the baselines in Fig. 3.7. The uniform sampling baseline has poor performance as the number of dimensions increases because the fraction of feasible states within the full state space decreases as the dimension increases. Thus, sampling uniformly results in sampling an increasing percentage of unfeasible goals, leading to poor learning signal. In contrast, the performance of our method does not decay as much as the state space dimension increases, because our Goal GAN always generates goals within the feasible portion of the state space. The *GAN fit all* variation of our method suffers from the increase in dimension because it is not encouraged to track the narrow feasible region. Finally, the oracle and the baseline with an L2 distance reward have perfect performance, which is expected in this task where the optimal policy is just to go in a straight line towards the goal. Even without this prior knowledge, the Goal GAN discovers the feasible subset of the goal space.

3.6 Reverse Curriculum Generation for Reinforcement Learning

In this section we focus on distributions of tasks where there is a single goal to reach, but we want to reach it robustly from any starting state. We are particularly interested in hard-to-reach goals that would be too inefficient to learn how to reach with the previous hierarchical methods or the automatic goal generation method above. In this section, we avoid costly supervisions like reward engineering or use of demonstrations by exploiting two key insights. First, it is easier to reach the goal from states nearby the goal, or from states nearby where the agent already knows how to reach the goal. Second, applying random actions from one such state leads the agent to new feasible nearby states, from where it is not too much harder to reach the goal. This can be understood as requiring a minimum degree of reversibility, which is usually satisfied in many robotic manipulation tasks like assembly and manufacturing.

We take advantage of these insights to develop a “reverse learning" approach for solving such difficult manipulation tasks. The robot is first trained to reach the goal from start states nearby a given goal state. Then, leveraging that knowledge, the robot is trained to solve the task from increasingly distant start states. All start states are automatically generated by executing a short random walk from the previous start states that got some reward but still require more training. This method of learning in reverse, or growing outwards from the goal, is inspired by dynamic programming methods like value iteration, where the solutions to easier sub-problems are used to compute the solution to harder problems.

Therefore, our method automatically generates a curriculum of initial positions from which to learn to achieve the task. Our method requires no prior knowledge of the task other than providing a single state that achieves the task (i.e. is at the goal). The contributions of this section include:

- Formalizing a novel problem definition of finding the optimal start-state distribution at every training step to maximize the overall learning speed.
- A novel and practical approach for sampling a start state distribution that varies over the course of training, leading to an automatic curriculum of start state distributions.
- Empirical experiments showing that our approach solves difficult tasks like navigation or fine-grained robotic manipulation, not solvable by state-of-the-art learning methods.

Formalization and assumptions

We denote by $R(\pi, s_0) := \mathbb{E}_{\tau|s_0}[\sum_{t=0}^{T} r(s_t, a_t)]$ the expected cumulative reward starting when starting from a $s_0 \sim \rho_0$, where $\tau = (s_0, a_0, , \dots, a_{T-1}, s_T)$ denotes a whole trajectory, with $a_t \sim \pi_\theta(a_t|s_t)$, and $s_{t+1} \sim \mathcal{P}(s_{t+1}|s_t, a_t)$.

In our work we propose to instead use a different start-state distribution ρ_i at every training iteration i to maximize the learning rate. Learning progress is still evaluated based on the original distribution ρ_0. Convergence of ρ_i to ρ_0 is desirable but not required as an optimal policy $\pi_i^\star$ under a start distribution ρ_i is also optimal under any other ρ_0, as long as their support coincide. In the case of approximately optimal policies under ρ_i, bounds on the performance under ρ_0 can be derived [70].

In this work we study how to exploit three assumptions that hold true in a wide range of practical learning problems (especially if learned in simulation):

Assumption 1. *We can arbitrarily reset the agent into any start state $s_0 \in \mathcal{S}$ at the beginning of all trajectories.*

Assumption 2. *At least one state s^g is provided such that $s^g \in S^g$.*

Assumption 3. *The Markov Chain induced by taking uniformly sampled random actions has a communicating class*[1] *including all start states S^0 and the given goal state s^g.*

The first assumption has been considered previously (e.g. access to a generative model in Kearns, Mansour, and Ng [73]) and is deemed to be a considerably weaker assumption than having access to the full transition model of the MDP. Kakade and Langford [70] proved that Assumption 1 can be used to improve the learning in MDPs that require large exploration. Nevertheless, they do not propose a concrete procedure to choose a distribution ρ from which to sample the start states in order to maximally improve on the objective coverage objective. In our case, combining Assumption 1 with Assumption 2, we are able to reset the state to s^g, which is critical in our method to initialize the start state distribution to concentrate around the goal space at the beginning of learning. For Assumption 2, note that we only assume access to one state s^g in the goal region; we do not require a description of the full region nor trajectories leading to it. Finally, Assumption 3 ensures that

[1]A *communicating class* is a maximal set of states C such that every pair of states in C communicates with each other. Two states *communicate* if there is a non-zero probability of reaching one from the other.

the goal can be reached from any of the relevant start states, and that those start states can also be reached from the goal; this assumption is satisfied by many robotic problems of interest, as long as there are no major irreversibilities in the system. In the next sections we detail our automatic curriculum generation method based on continuously adapting the start state distribution to the current performance of the policy. We demonstrate the value of this method for challenging robotic manipulation tasks.

In a wide range of goal-oriented RL problems, reaching the goal from an overwhelming majority of start states in S^0 requires a prohibitive amount of on-policy or undirected exploration. On the other hand, it is usually easy for the learning agent (i.e. our current policy π_i) to reach the goal S^g from states nearby a goal state $s^g \in S^g$. Therefore, learning from these states will be fast because the agent will perceive a strong signal, even under the indicator reward introduced in Section 3.7. Once the agent knows how to reach the goal from these nearby states, it can train from even further states and bootstrap its already acquired knowledge. This reverse expansion is inspired by classical RL methods like Value Iteration or Policy Iteration [155], although in our case we do not assume knowledge of the transition model and our environments have high-dimensional continuous action and state spaces. In the following subsections we propose a method that leverages the assumptions from the previous section and the idea of reverse expansion to automatically adapt the start state distribution, generating a curriculum of start state distributions that can be used to tackle problems unsolvable by standard RL methods.

Policy Optimization with modified start state distribution

Policy gradient strategies are well suited for robotic tasks with continuous and high dimensional action-spaces [25]. Nevertheless, applying them directly on the original MDP does poorly in tasks with sparse rewards and long horizons like our challenging manipulation tasks. If the goal is not reached from the start states in S^0, no reward is received, and the policy cannot improve. Therefore, we propose to adapt the distribution ρ_i from where start states s_0 are sampled to train policy π_i.

Analogously to the previous section, here we also postulate that in goal-oriented environments, a strong learning signal is obtained when training on start states $s_0 \sim \rho_i$ from where the agent reaches the goal sometimes, but not always. We call these start states “good starts". More formally, at training iteration i, we would like to sample from $\rho_i = \text{Unif}(S_i^0)$ where $S_i^0 = \{s_0 : R_{\min} < R(\pi_i, s_0) < R_{\max}\}$. The hyper-parameters $R_{\min}$ and $R_{\max}$ are easy to tune due to their interpretation as bounds on the probability of success. Unfortunately, sampling uniformly from S_i^0 is intractable. Nevertheless, at least at the beginning of training, states nearby a goal state s^g are more likely to be in S_i^0. Then, after some iterations of training on these start states, some will be completely mastered (i.e. $R(\pi_{i+1}, s_0) > R_{max}$ and s_0 is no longer in S_{i+1}^0), but others will still need more training. To find more “good starts", we follow the same reasoning: the states nearby these remaining $s \in S_{i+1}^0$ are likely to also be in S_{i+1}^0. In the rest of the section we describe an effective way of sampling feasible nearby states and we layout the full algorithm.

Algorithm 5: Policy Training

```
Input  : $\pi_0$, $s^g$, $\rho_0$, $N_{\text{new}}$, $N_{\text{old}}$, $R_{\min}$, $R_{\max}$, $Iter$
Output : Policy $\pi_N$
$starts_{old} \leftarrow [s^g]$;
$starts,\ rews \leftarrow [s^g],\ [1]$;
for $i \leftarrow 1$ to $Iter$ do
    $starts \leftarrow$ SampleNearby($starts$, $N_{new}$);
    $starts$.append[sample($starts_{old}$, $N_{old}$)];
    $\rho_i \leftarrow$ Unif($starts$);
    $\pi_i$, $rews \leftarrow$ train_pol($\rho_i$, $\pi_{i-1}$);
    $starts \leftarrow$
      select($starts$, $rews$, $R_{\min}$, $R_{\max}$);
    $starts_{old}$.append[$starts$];
end
```

Procedure 6: `SampleNearby`

```
Input  : $starts$, $N_{\text{new}}$, $\Sigma$, $T_B$,
         $M$
Output : $starts_{\text{new}}$
while $len(starts) < M$ do
    $s_0 \sim Unif(starts)$;
    for $t \leftarrow 1$ to $T_B$ do
        $a_t = \epsilon_t,\ \epsilon_t \sim \mathcal{N}(0, \Sigma)$;
        $s_t \sim \mathcal{P}(s_t | s_{t-1}, a_t)$;
        $starts$.append($s_t$);
    end
end
$starts_{\text{new}} \leftarrow$
  sample($starts$, $N_{\text{new}}$)
```

Sampling “nearby" feasible states

For robotic manipulation tasks with complex contacts and constraints, applying noise in state-space $s' = s + \epsilon,\ \epsilon \sim \mathcal{N}$ may yield many infeasible states s'. For example, even small random perturbations of the joint angles of a seven degree-of-freedom arm generate large modifications to the end-effector position, potentially placing it in an infeasible state that intersects with surrounding objects. For this reason, the concept of “nearby" states might be unrelated to the Euclidean distance $||s' - s||^2$ between these states. Instead, we have to understand proximity in terms of how likely it is to reach one state from the other by taking actions in the MDP.

Therefore, we choose to generate new states s' from a certain seed state s by applying noise in action space. This means we exploit Assumption 1 to reset the system to state s, and from there we execute short “Brownian motion" rollouts of horizon T_B taking actions $a_{t+1} = \epsilon_t$ with $\epsilon_t \sim \mathcal{N}(0, \Sigma)$. This method of generating “nearby" states is detailed in Procedure 6. The total sampled states M should be large enough such that the N_{new} desired states $starts_{\text{new}}$, obtained by subsampling, extend in all directions around the input states $starts$. All states visited during the rollouts are guaranteed to be feasible and can then be used as start states to keep training the policy.

Detailed Algorithm

Our generic algorithm is detailed in Algorithm 5. We first initialize the policy with π_0 and the “good start" states list $starts$ with the given goal state s^g. Then we perform $Iter$ training iterations of our RL algorithm of choice `train_pol`. In our case we perform 5 iterations of Trust Region Policy Optimization (TRPO) [138] but any on-policy method could be used. At every iteration, we set the start state distribution ρ_i to be uniform over a list of start states obtained by sampling N_{new} start states from nearby the ones in our “good starts" list $starts$ (see `SampleNearby` in previous section), and N_{old} start states from our replay buffer of previous “good starts" $starts_{\text{old}}$. As already shown by Florensa* et al. [39], the replay buffer is an important feature to avoid catastrophic forgetting. Technically, to check which of the states $s_0 \in starts$ are in S_i^0 (i.e. the "good starts") we should execute some trajectories from each of those states to estimate the expected returns $R(s_0, \pi_{i-1})$, but this considerably increases the sample complexity. Instead, we use the trajectories collected by `train_pol` to estimate $R(\pi_{i-1}, s_0)$ and save it in the list $rews$. These are used to `select` the "good" start states for the next iteration - picking the ones with $R_{\min} \leq R(\pi_{i-1}, s_0) \leq R_{\max}$. We found this heuristic to give a good enough estimate and not drastically decrease learning performance of the overall algorithm.

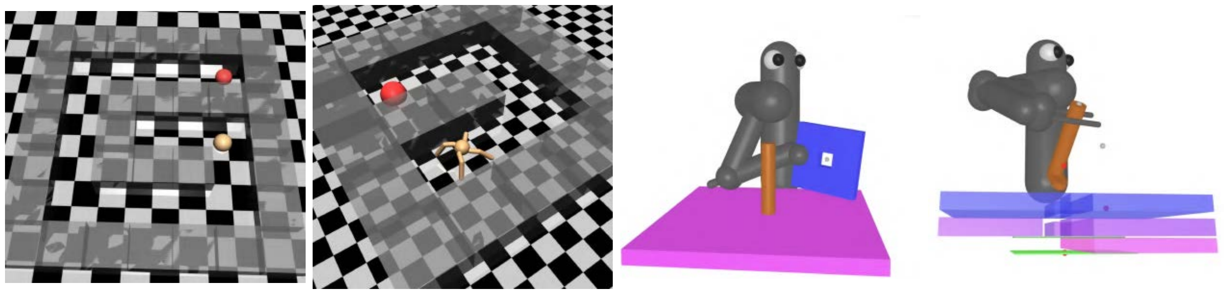
Our method keeps expanding the region of the state-space from which the policy can reach the goal reliably. It samples more heavily nearby the start states that need more training to be mastered and avoiding start states that are yet too far to receive any reward under the current policy. Then, thanks to Assumption 3, the Brownian motion that is used to generate further and further start states will eventually reach all start states in S^0, and therefore our method improves the metric η_{ρ_0} defined in Sec. 3.3 (see also Sec. A.4 for details on how we evaluate our progress on this metric).

3.7 Experimental Results for Reverse Curriculum

We investigate the following questions in our experiments:

- Does the performance of the policy on the target start state distribution ρ_0 improve by training on distributions ρ_i growing from the goal?
- Does focusing the training on "good starts" speed up learning?
- Is Brownian motion a good way to generate "good starts" from previous "good starts"?

We use the below task settings to explore these questions. All are implemented in MuJoCo [166] and the hyperparameters used in our experiments are described in Appendix A.4.



(a) Point-mass maze task (b) Ant maze task (c) Ring on Peg task (d) Key insertion task

Figure 3.8: Task images. Source code and videos of the performance obtained by our algorithm are available here: `http://bit.ly/reversecurriculum`

Point-mass maze: (Fig. 3.8a) A point-mass agent (orange) must navigate within 30cm of the goal position $(4m, 4m)$ at the end of a G-shaped maze (red). The target start state distribution from which we seek to reach the goal is uniform over all feasible (x, y) positions in the maze.

Ant maze: (Fig. 3.8b) A quadruped robot (orange) must navigate its Center of Mass to within 50cm of the goal position $(0m, 4m)$ at the end of a U-shaped maze (red). The target start state distribution from which we seek to reach the goal is uniform over all feasible ant positions inside the maze.

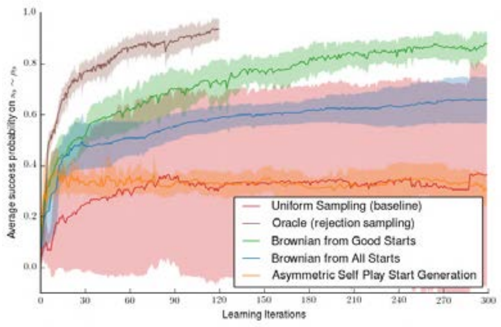
Ring on Peg: (Fig. 3.8c) A 7 DOF robot must learn to place a “ring" (actually a square disk with a hole in the middle) on top of a tight-fitting round peg. The task is complete when the ring is within 3 cm of the bottom of the 15 cm tall peg. The target start state distribution from which we seek to reach the goal is uniform over all feasible joint positions for which the center of the ring is within 40 cm of the bottom of the peg.

Key insertion: (Fig. 3.8d) A 7 DOF robot must learn to insert a key into a key-hole. The task is completed when the distance between three reference points at the extremities of the key and its corresponding targets is below 3cm. In order to reach the target, the robot must first insert the key at a specific orientation, then rotate it 90 degrees clockwise, push forward, then rotate 90 degrees counterclockwise. The target start state distribution from which we seek to reach the goal is uniform over all feasible joint positions such that the tip of the key is within 40 cm of key-hole.

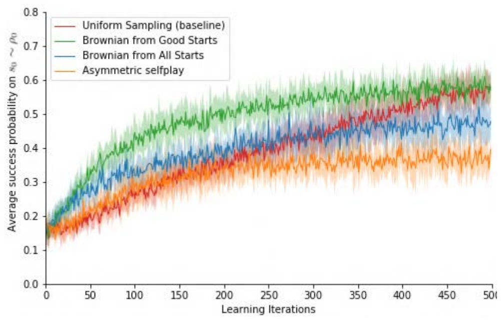
Effect of start state distribution

In Figure 3.9, the *Uniform Sampling (baseline)* red curves show the average return of policies learned with TRPO without modifying the start state distribution. The green and blue curves correspond to our method and an ablation, both exploiting the idea of modifying the start state distribution at every learning iteration. These approaches perform consistently better across the board. In the case of the point-mass maze navigation task in Fig. 3.9a, we observe that *Uniform Sampling* has a very high variance because some policies only learn how to perform well from one side of the goal (see Appendix A.4 for a thorough analysis). The Ant-maze experiments in Fig. 3.9b also show a considerable slow-down of the learning speed when using plain TRPO, although the effect is less drastic as the start state distribution ρ_0 is over a smaller space.

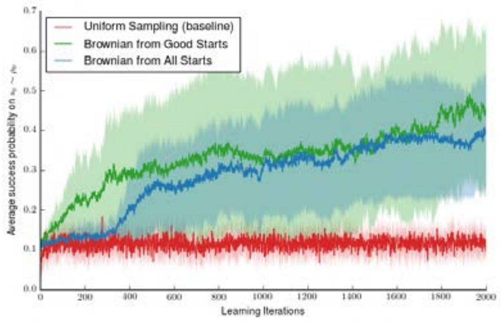
In the more complex manipulation tasks shown in Fig. 3.9c-3.9d, we see that the probability of reaching the goal with *Uniform Sampling* is around 10% for the ring task and 2% for the key task. These success probabilities correspond to reliably reaching the goal only from very nearby positions: when the ring is already on the peg or when the key is initialized very close to the final position. None of the learned policies trained on the original ρ_0 learn to reach the goal from more distant start states. On the other hand, our methods do succeed at reaching the goal from a wide range of far away start states. The underlying RL training algorithm and the evaluation metric are the same. We conclude that training on a different start state distribution ρ_i can improve training or even allow learning at all.



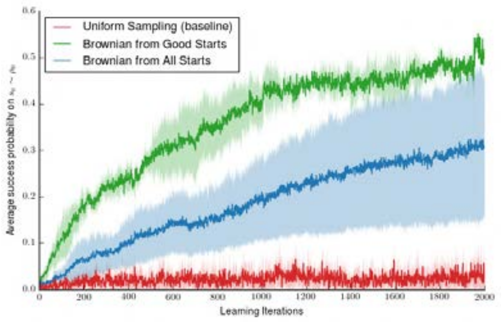
(a) Point-mass Maze task



(b) Ant Maze task



(c) Ring on Peg task



(d) Key insertion task

Figure 3.9: Learning curves for goal-oriented tasks (mean and variance over 5 random seeds).

Effect of “good starts"

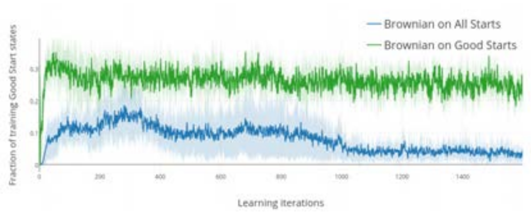
In Figure 3.9 we see how applying our Algorithm 5 to modify the start state distribution considerably improves learning (*Brownian on Good Starts*, in green) and final performance on the original MDP. Two elements are involved in this improvement: first, the backwards expansion from the goal, and second, the concentration of training efforts on “good starts". To test the relevance of this second element, we ablate our method by running our `SampleNearby` Procedure 6 on all states from which the policy was trained in the previous iteration. In other words, the `select` function in Algorithm 5 is replaced by the identity, returning all *starts* independently of the rewards *rews* they obtained during the last training iteration. The resulting algorithm performance is shown as the *Brownian from All Starts* blue curve in Figures 3.9. As expected, this method is still better than not modifying the start state distribution but has a slower learning rate than running `SampleNearby` around the estimated good starts.

Now we evaluate an upper bound of the benefit provided by our idea of sampling “good starts". As mentioned in Sec. 3.6, we would ideally like to sample start states from $\rho_i = \text{Unif}(S_i^0)$, but it is intractable. Instead, we evaluate states in S_{i-1}^0, and we use Brownian motion to find nearby states, to approximate S_i^0. We can evaluate how much this approximation hinders learning by exhaustively sampling states in the lower dimensional point-mass maze task. To do so, at every iteration we can sample states s_0 uniformly from the state-space $\mathcal{S}$, empirically estimate their return $R(s_0, \pi_i)$, and reject the ones that are not in the set $S_i^0 = \{s_0 : R_{\min} < R(\pi_i, s_0) < R_{\max}\}$. This exhaustive sampling method is orders of magnitude more expensive in terms of sample complexity, so it would not be of practical use. In particular, we can only run it in the easier point-mass maze task. Its performance is shown in the brown curve of Fig. 3.9a, called “Oracle (rejection sampling)"; training on states sampled in such a manner further improves the learning rate and final performance. Thus we can see that our approximation of using states in S_{i-1}^0 to find states in S_i^0 leads to some loss in performance, at the benefit of a greatly reduced computation time.

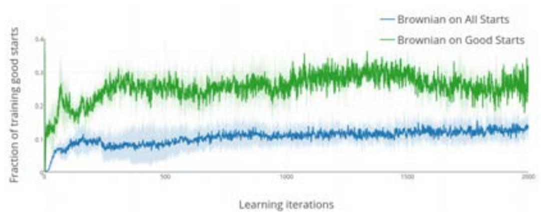
Finally, we compare to another way of generating start states based on the asymmetric self-play method of Sukhbaatar et al. [152]. The basic idea is to train another policy, “Alice", that proposes start states to the learning policy, “Bob". As can be seen, this method performs very poorly in the point-mass maze task, and our investigation shows that “Alice" often gets stuck in a local optimum, leading to poor start states suggestions for “Bob". In the original paper, the method was demonstrated only on discrete action spaces, in which a multi-modal distribution for Alice can be maintained; even in such settings, the authors observed that Alice can easily get stuck in local optima. This problem is exacerbated when moving to continuous action spaces defined by a unimodal Gaussian distribution. See a detailed analysis of these failure modes in Appendix A.4.

Brownian motion to generate good starts “nearby" good starts

Here we evaluate if running our Procedure 6 `SampleNearby` with the start states estimated as “good" from the previous iteration yields more good starts than running `SampleNearby` from all start states used in the previous iteration. This can clearly be seen in Figs. 3.10b-3.10a for the robotic manipulation tasks.



(a) Ring on Peg task



(b) Key insertion task

Figure 3.10: Fraction of “good starts" generated during training for the robotic manipulation tasks

3.8 Conclusions and Future Directions

In this chapter we have proposed two methods within the RL paradigm where the objective is to train a single policy to succeed on a variety of tasks, under sparse rewards. To solve this problem first we develop a method for automatic curriculum generation that dynamically adapts to the current performance of the agent. The curriculum is obtained without any prior knowledge of the environment or of the tasks being performed. We use generative adversarial training to automatically generate goals for our policy that are always at the appropriate level of difficulty (i.e. not too hard and not too easy). In the future we want to combine our goal-proposing strategy with recent multi-goal approaches like HER [2] that could greatly benefit from better ways to select the next goal to train on. Another promising line of research is to build hierarchy on top of the multi-task policy that we obtain with our method by training a higher-level policy that outputs the goal for the lower level multi-task policy [56, 35]. The hierarchy could also be introduced by replacing our current feed-forward neural network policy by an architecture that learns to build implicit plans [102, 158], or by leveraging expert demonstrations to extract sub-goals [184], although none of these approaches tackles yet the multi-task learning problem formulated in this chapter.

We also propose a method to automatically adapt the start state distribution on which an agent is trained, such that the performance on the original problem is efficiently optimized. We leverage three assumptions commonly satisfied in simulated tasks to tackle hard goal-oriented problems that state of the art RL methods cannot solve. A limitation of the current approach is that it generates start states that grow from a single goal uniformly outwards, until they cover the original start state distribution Unif(S^0). Nevertheless, if the target set of start states S^0 is far from the goal and we have some prior knowledge, it would be interesting to bias the generated start distributions ρ_i towards the desired start distribution. A promising future line of work is to combine both present automatic curriculum methods presented in this chapter, jointly sampling starts and goals similarly to classical results in planning [78]. It can be observed in the videos of our final policy for the manipulation tasks that the agent has learned to *exploit* the contacts instead of avoiding them. Therefore, the learning based aspect of the presented method has a huge potential to tackle problems that classical motion planning algorithms could struggle with, such as environments with non-rigid objects or with uncertainties in the task geometric parameters. We also leave as future work to combine our curriculum-generation approach with domain randomization methods [165] to obtain policies that are transferable to the real world.

Chapter 4

Leveraging Partial Expert Demonstrations

4.1 Introduction

Designing rewards for Reinforcement Learning (RL) is challenging because it needs to convey the desired task, be efficient to optimize, and be easy to compute. The latter is particularly problematic when applying RL to robotics, where detecting whether the desired configuration is reached might require considerable supervision and instrumentation. Furthermore, we are often interested in being able to reach a wide range of configurations, hence setting up a different reward every time might be unpractical. Methods like Hindsight Experience Replay (HER) have recently shown promise to learn policies able to reach many goals, without the need of a reward. Unfortunately, without tricks like resetting to points along the trajectory, HER might require many samples to discover how to reach certain areas of the state-space. In this work we propose a novel algorithm *goalGAIL*, which incorporates demonstrations to drastically speed up the convergence to a policy able to reach any goal, surpassing the performance of an agent trained with other Imitation Learning algorithms. Furthermore, we show our method can also be used when the available expert trajectories do not contain the actions or when the expert is suboptimal, which makes it applicable when only kinesthetic, third-person or noisy demonstrations are available. Our code is open-source [1].

Reinforcement Learning (RL) has shown impressive results in a plethora of simulated tasks, ranging from attaining super-human performance in video-games [103, 175] and board-games [142], to learning complex locomotion behaviors [54, 35]. Nevertheless, these successes are shyly echoed in real world robotics [125, 185]. This is due to the difficulty of setting up the same learning environment that is enjoyed in simulation. One of the critical assumptions that are hard to obtain in the real world are the access to a reward function. Self-supervised methods have the power to overcome this limitation.

A very versatile and reusable form of self-supervision for robotics is to learn how to reach any previously observed state upon demand. This problem can be formulated as training a goal-conditioned policy [68, 129] that seeks to obtain the indicator reward of having the observation exactly match the goal. Such a reward does not require any additional instrumentation of the

[1]https://sites.google.com/view/goalconditioned-il/

environment beyond the sensors the robot already has. But in practice, this reward is never observed because in continuous spaces like the ones in robotics, it is extremely rare to observe twice the exact same sensory input. Luckily, if we are using an off-policy RL algorithm [95, 51], we can “relabel" a collected trajectory by replacing its goal by a state actually visited during that trajectory, therefore observing the indicator reward as often as we wish. This method was introduced as Hindsight Experience Replay [2] or HER, although it used special resets, and the reward was in fact an ϵ-ball around the goal, which is only easy to interpret and use in lower-dimensional state-spaces. More recently the method was shown to work directly from vision with a special reward [106], and even only with the indicator reward of exactly matching observation and goal [38].

In theory these approaches could learn how to reach any goal, but the breadth-first nature of the algorithm makes that some areas of the space take a long time to be learned [39]. This is specially challenging when there are bottlenecks between different areas of the state-space, and random motion might not traverse them easily [36]. Some practical examples of this are pick-and-place, or navigating narrow corridors between rooms, as illustrated in Fig. 4.2 depicting the diverse set of environments we work with. In both cases a specific state needs to be reached (grasp the object, or enter the corridor) before a whole new area of the space is discovered (placing the object, or visiting the next room). This problem could be addressed by engineering a reward that guides the agent towards the bottlenecks, but this defeats the purpose of trying to learn without direct reward supervision. In this work we study how to leverage a few demonstrations that traverse those bottlenecks to boost the learning of goal-reaching policies.

Learning from Demonstrations, or Imitation Learning (IL), is a well-studied field in robotics [71, 126, 15]. In many cases it is easier to obtain a few demonstrations from an expert than to provide a good reward that describes the task. Most of the previous work on IL is centered around trajectory following, or doing a single task. Furthermore it is limited by the performance of the demonstrations, or relies on engineered rewards to improve upon them. In this work we first illustrate how IL methods can be extended to the goal-conditioned setting, and study a more powerful relabeling strategy that extracts additional information from the demonstrations. We then propose a novel algorithm, *goalGAIL*, and show it can outperform the demonstrator without the need of any additional reward. We also investigate how our method is more robust to sub-optimal experts. Finally, the method we develop is able to leverage demonstrations that do not include the expert actions. This considerably broadens its application in practical robotics, where demonstrations might be given by a motion planner, by kinesthetic demonstrations [99] (moving the agent externally, instead of using its own controls), or even by another agent [149]. To our knowledge, this is the first framework that can boost goal-conditioned policy learning with only state demonstrations.

4.2 Related Work

Related Work for Goal-Conditioned Imitation Learning

Imitation Learning is an alternative to reward crafting to train a desired behaviors. There are many ways to leverage demonstrations, from Behavioral Cloning [116] that directly maximizes the likelihood of the expert actions under the training agent policy, to Inverse Reinforcement Learning that extracts a reward function from those demonstrations and then trains a policy to maximize it [188, 33, 41]. Another formulation close to the latter is Generative Adversarial Imitation Learning (GAIL), introduced by Ho and Ermon [60]. GAIL is one of the building blocks of our own algorithm, and is explained in more details in the Preliminaries section.

Unfortunately most work in the field cannot outperform the expert, unless another reward is available during training [171, 44, 153], which might defeat the purpose of using demonstrations in the first place. Furthermore, most tasks tackled with these methods consist of tracking expert state trajectories [187, 113], but cannot adapt to unseen situations.

In this work we are interested in goal-conditioned tasks, where the objective is to reach any state upon demand [68, 129]. This kind of multi-task learning is pervasive in robotics [3, 96], but challenging and data-hungry if no reward-shaping is available. Relabeling methods like Hindsight Experience Replay [2] unlock the learning even in the sparse reward case [38]. Nevertheless, the inherent breath-first nature of the algorithm might still produce inefficient learning of complex policies. To overcome the exploration issue we investigate the effect of leveraging a few demonstrations. The closest prior work is by Nair et al. [105], where a Behavioral Cloning loss is used with a Q-filter. We found that a simple annealing of the Behavioral Cloning loss [119] works well and allows the agent to outperform demonstrator. Furthermore, we introduce a new relabeling technique of the expert trajectories that is particularly useful when only few demonstrations are available. Finally we propose a novel algorithm *goalGAIL*, leveraging the recently shown compatibility of GAIL with off-policy algorithms.

Related Work for GUAPO

In robotic manipulation there are two dominating paradigms to perform a task: leveraging model of the environment (model-based method) or leveraging data to learn (learning-based method). The first category of methods relies on a precise description of the task, such as object CAD models, as well as powerful and sophisticated perception systems [133, 180]. With an accurate model, a well engineered solution can be designed for that particular task [170, 74], or the model can then be combined with some search algorithm like motion planning [164]. This type of model-based approach is limited by the ingenuity of the roboticist, and could lead to irrecoverable failure if the perception system has un-modeled noise and error.

On the other hand, learning-based approaches in manipulation [89, 49] do not require such detailed description, but rather require access to interaction with the environment, as well as a reward that indicates success. Such binary rewards are easy to describe, but unfortunately they render Reinforcement Learning methods extremely sample-inefficient. Hence many prior works use shaped rewards [86], which requires considerable tuning. Other works use low-dimensional state spaces [186] instead of image inputs, which requires either precise perception systems or specially-designed hardware with sensors. There are some proposed methods that manage to deal directly with the sparse rewards, like automatic curriculum generation [39, 36] or the use of demonstrations [171, 29, 105], but these approaches still require large amounts of interaction with the environment. Furthermore, if the position of the objects in the scene changes or there are new distractors in the scene, these methods need to be fully retrained. On the other hand, our method is extremely sample-efficient with a sparse success reward, and is robust to these variations thanks to the model-based component.

Recent works can also be understood as combining model-based and learning-based approaches. One such method [67] uses a reinforcement learning algorithm to find the best parameters that describe the behavior of the agent based on a model-based template. The learning is very efficient, but at the cost of an extremely engineered pre-solution that also relies on an accurate perception system. Another line of work that allows to combine model-based and learning-based methods is Residual Learning [66, 143], where RL is used to learn an additive policy that can potentially fully over-write the original model-based policy and does not require any further structure. Nevertheless, these methods are hard to tune, and hardly preserve any of the benefits of the underlying model-based method once trained.

The problem of known object pose estimation is a vibrant subject within the robotics and computer vision communities [168, 57, 61, 182, 181, 63, 112, 154, 163]. Regressing to keypoints on the object or on a cuboid encompassing the object seems to have become the defacto approach for the problem. Keypoints are first detected by a neural network, then PnP [88] is used to predict the pose of the object. Peng *et al.* [112] also explored the problem of using uncertainty by leveraging a ransac voting algorithm to find regions where a keypoint could be detected. This approach differs from ours as they do not directly regress to a keypoint probability map, they regress to a vector voting map, where line intersection is then used to find keypoints. Moreover their method does not carry pose uncertainty in the final prediction.

4.3 Background on Imitation Learning

We define a discrete-time finite-horizon discounted Markov decision process (MDP) by a tuple $M = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \rho_0, \gamma, H)$, where $\mathcal{S}$ is a state set, $\mathcal{A}$ is an action set, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_+$ is a transition probability distribution, $\gamma \in [0, 1]$ is a discount factor, and H is the horizon. Our objective is to find a stochastic policy π_θ that maximizes the expected discounted reward within the MDP, $\eta(\pi_\theta) = \mathbb{E}_\tau[\sum_{t=0}^{T} \gamma^t r(s_t, a_t, s_{t+1})]$. We denote by $\tau = (s_0, a_0, ...,)$ an entire state-action trajectory, where $s_0 \sim \rho_0(s_0)$, $a_t \sim \pi_\theta(\cdot|s_t)$, and $s_{t+1} \sim \mathcal{P}(\cdot|s_t, a_t)$. In the goal-conditioned setting that we use here, the policy and the reward are also conditioned on a “goal" $g \in \mathcal{S}$. The reward is $r(s_t, a_t, s_{t+1}, g) = \mathbb{1}\big[s_{t+1} == g\big]$, and hence the return is the γ^h, where h is the number of time-steps to the goal. Given that the transition probability is not affected by the goal, g can be “relabeled" in hindsight, so a transition $(s_t, a_t, s_{t+1}, g, r = 0)$ can be treated as $(s_t, a_t, s_{t+1}, g' = s_{t+1}, r = 1)$. Finally, we also assume access to D trajectories $\big\{(s_0^j, a_0^j, s_1^j, ...)\big\}_{j=0}^{D} \sim \tau_{\text{expert}}$ that were collected by an expert attempting to reach the goals $\{g_j\}_{j=0}^{D}$ sampled uniformly among the feasible goals. Those trajectories must be approximately geodesics, meaning that the actions are taken such that the goal is reached as fast as possible.

In GAIL [60], a discriminator D_ψ is trained to distinguish expert transitions $(s, a) \sim \tau_{\text{expert}}$, from agent transitions $(s, a) \sim \tau_{\text{agent}}$, while the agent is trained to "fool" the discriminator into thinking itself is the expert. Formally, the discriminator is trained to minimize $\mathcal{L}_{GAIL} =$

$\mathbb{E}_{(s,a)\sim\tau_{\text{agent}}}[\log D_\psi(s,a)] + \mathbb{E}_{(s,a)\sim\tau_{\text{expert}}}[\log(1 - D_\psi(s,a))]$; while the agent is trained to maximize $\mathbb{E}_{(s,a)\sim\tau_{\text{agent}}}[\log D_\psi(s,a)]$ by using the output of the discriminator $\log D_\psi(s,a)$ as reward. Originally, the algorithms used to optimize the policy are on-policy methods like Trust Region Policy Optimization [138], but recently there has been a wake of works leveraging the efficiency of off-policy algorithms without loss in stability [14, 127, 134, 77]. This is a key capability that we exploit in our *goalGAIL* algorithm.

4.4 Goal-conditioned Imitation Learning

In this section we describe methods to incorporate demonstrations into Hindsight Experience Replay [2] for training goal-conditioned policies. First we revisit adding a Behavioral Cloning loss to the policy update as in [105], then we propose a novel expert relabeling technique, and finally we formulate for the first time a goal-conditioned GAIL algorithm termed *goalGAIL*, and propose a method to train it with state-only demonstrations.

Goal-conditioned Behavioral Cloning

The most direct way to leverage demonstrations $\big\{(s_0^j, a_0^j, s_1^j, ...)\big\}_{j=0}^{D}$ is to construct a data-set $\mathcal{D}$ of all state-action-goal tuples (s_t^j, a_t^j, g^j), and run supervised regression. In the goal-conditioned case and assuming a deterministic policy $\pi_\theta(s,g)$, the loss is:

$$\mathcal{L}_{\text{BC}}(\theta, \mathcal{D}) = \mathbb{E}_{(s_t^j, a_t^j, g^j)\sim\mathcal{D}}\Big[\|\pi_\theta(s_t^j, g^j) - a_t^j\|_2^2\Big] \tag{4.1}$$

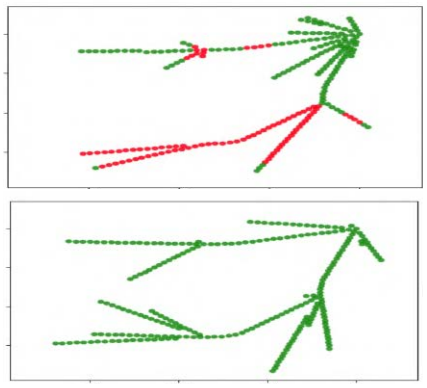
This loss and its gradient are computed without any additional environments samples from the trained policy π_θ. This makes it particularly convenient to combine a gradient descend step based on this loss together with other policy updates. In particular we can use a standard off-policy Reinforcement Learning algorithm like DDPG [95], where we fit the $Q_\phi(a,s,g)$, and then estimate the gradient of the expected return as:

$$\nabla_\theta \hat{J} = \frac{1}{N}\sum_{i=1}^{N} \nabla_a Q_\phi(a,s,g)\nabla_\theta\pi_\theta(s,g) \tag{4.2}$$

In our goal-conditioned case, the Q fitting can also benefit from “relabeling“ like done in HER [2]. The improvement guarantees with respect to the task reward are lost when we combine the BC and the deterministic policy gradient updates, but this can be side-stepped by either applying a Q-filter $\mathbb{1}\big\{Q(s_t, a_t, g) > Q(s_t, \pi(s_t, g), g)\big\}$ to the BC loss as proposed in [105], or by annealing it as we do in our experiments, which allows the agent to eventually outperform the expert.

Relabeling the expert

The expert trajectories have been collected by asking the expert to reach a specific goal g^j. But they are also valid trajectories to reach any other state visited within the demonstration! This is the key


(a) Performance on reaching states visited in the 20 given demonstrations. The states are green if reached by the policy when attempting so, and red otherwise.

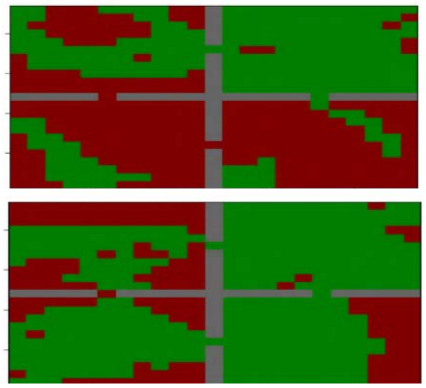

(b) Performance on reaching any possible state. Each cell is colored green if the policy can reach the center of it when attempting so, and red otherwise.

Figure 4.1: Policy performance on reaching different goals in the four rooms, when training with standard Behavioral Cloning (top row) or with our expert relabeling (bottom).

motivating insight to propose a new type of relabeling: if we have the transitions $(s_t^j, a_t^j, s_{t+1}^j, g^j)$ in a demonstration, we can also consider the transition $(s_t^j, a_t^j, s_{t+1}^j, g' = s_{t+k}^j)$ as also coming from the expert! Indeed that demonstration also went through the state s_{t+k}^j, so if that was the goal, the expert would also have generated this transition. This can be understood as a type of data augmentation leveraging the assumption that the given demonstrations are geodesics (they are the faster way to go from any state in it to any future state in it). It will be particularly effective in the low data regime, where not many demonstrations are available. The effect of expert relabeling can be visualized in the four rooms environment as it's a 2D task where states and goals can be plotted. In Fig. 4.1 we compare the final performance of two agents, one trained with pure Behavioral Cloning, and the other one also using expert relabeling.

Goal-conditioned GAIL with Hindsight

The compounding error in Behavioral Cloning might make the policy deviate arbitrarily from the demonstrations, and it requires too many demonstrations when the state dimension increases. The first problem is less severe in our goal-conditioned case because in fact we do want to visit and be able to purposefully reach all states, even the ones that the expert did not visit. But the second drawback will become pressing when attempting to scale this method to practical robotics tasks where the observations might be high-dimensional sensory input like images. Both problems can be mitigated by using other Imitation Learning algorithms that can leverage additional rollouts collected by the learning agent in a self-supervised manner, like GAIL [60]. In this section we extend the formulation of GAIL to tackle goal-conditioned tasks, and then we detail how it can be combined with HER [2], which allows the agent to outperform the demonstrator and generalize to reaching all goals. We call the final algorithm *goalGAIL.*

We describe the key points of *goalGAIL* below. First of all, the discriminator needs to also be conditioned on the goal which results in $D_\psi(a, s, g)$, and be trained by minimizing

$$\begin{aligned}\mathcal{L}_{GAIL}(D_\psi, \mathcal{D}, \mathcal{R}) \; = \; & \mathbb{E}_{(s,a,g)\sim\mathcal{R}}[\log D_\psi(a, s, g)] \; + \\ & \mathbb{E}_{(s,a,g)\sim\mathcal{D}}[\log(1 - D_\psi(a, s, g))].\end{aligned} \tag{4.3}$$

Once the discriminator is fitted, we can run our favorite RL algorithm on the reward $\log D_\psi(a_t^h, s_t^h, g^h)$. In our case we used the off-policy algorithm DDPG [95] to allow for the relabeling techniques outlined above. In the goal-conditioned case we intepolate between the GAIL reward described above and an indicator reward $r_t^h = \mathbb{1}\left[s_{t+1}^h == g^h\right]$. This combination is slightly tricky because now the fitted Q_ϕ does not have the same clear interpretation it has when only one of the two rewards is used [38] . Nevertheless, both rewards are pushing the policy towards the goals, so it shouldn't be too conflicting. Furthermore, to avoid any drop in final performance, the weight of the reward coming from GAIL δ_{GAIL} can be annealed. The final proposed algorithm *goalGAL*, together with the expert relabeling technique is formalized in Algorithm 7.

Use of state-only Demonstrations

Both Behavioral Cloning and GAIL use state-action pairs from the expert. This limits the use of the methods, combined or not with HER, to setups where the exact same agent was actuated to reach different goals. Nevertheless, much more data could be cheaply available if the action was not required. For example, non-expert humans might not be able to operate a robot from torque instructions, but might be able to move the robot along the desired trajectory. This is called a kinesthetic demonstration. Another type of state-only demonstration could be the one used in third-person imitation [149], where the expert performed the task with an embodiment different from the agent that needs to learn the task. This has mostly been applied to the trajectory-following case. In our case every demonstration might have a different goal.

Furthermore, we would like to propose a method that not only leverages state-only demonstrations, but can also outperform the quality and coverage of the demonstrations given, or at least generalize to similar goals. The main insight we have here is that we can replace the action in the GAIL formulation by the next state s', and in most environments this should be as informative as having access to the action directly. Intuitively, given a desired goal g, it should be possible to determine if a transition $s \to s'$ is taking the agent in the right direction. The loss function to train a discriminator able to tell apart the current agent and expert demonstrations (always transitioning towards the goal) is simply:

$$\mathcal{L}_{GAIL^s}(D_\psi^s, \mathcal{D}, \mathcal{R}) = \mathbb{E}_{(s,s',g)\sim\mathcal{R}}[\log D_\psi^s(s, s', g)] + \mathbb{E}_{(s,s',g)\sim\mathcal{D}}[\log(1 - D_\psi^s(s, s', g))].$$

4.5 Experimental Results for Goal-Conditioned IL

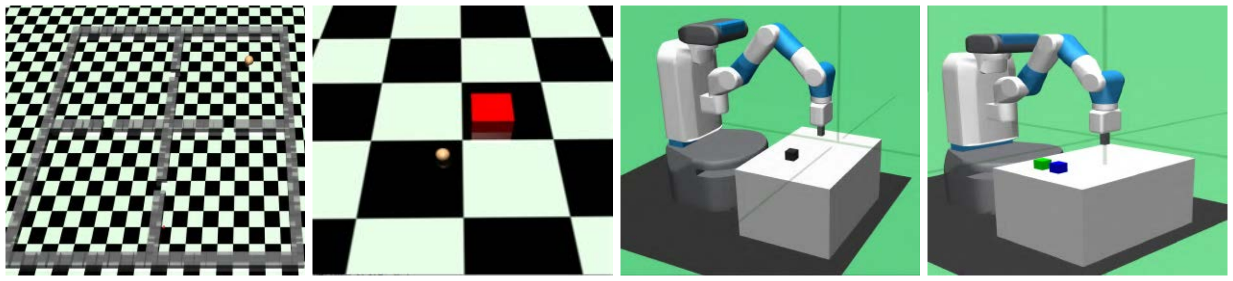
We are interested in answering the following questions:

Algorithm 7: Goal-conditioned GAIL with Hindsight: *goalGAIL*

```
Input  :Demonstrations $\mathcal{D} = \left\{(s_0^j, a_0^j, s_1^j, ..., g^j)\right\}_{j=0}^D$, replay buffer $\mathcal{R} = \{\}$, policy
        $\pi_\theta(s, g)$, discount $\gamma$, hindsight probability $p$
while not done do
    # Sample rollout
    $g \sim \texttt{Uniform}(\mathcal{S})$
    $\mathcal{R} \leftarrow \mathcal{R} \cup (s_0, a_0, s_1, ...)$ sampled using $\pi(\cdot, g)$
    # Sample from expert buffer and replay buffer
    $\left\{(s_t^j, a_t^j, s_{t+1}^j, g^j)\right\} \sim \mathcal{D}$, $\left\{(s_t^i, a_t^i, s_{t+1}^i, g^i)\right\} \sim \mathcal{R}$
    # Relabel agent transitions
    for each $i$, with probability $p$ do
        $g^i \leftarrow s_{t+k}^i, \quad k \sim \text{Unif}\{t+1, \dots, T^i\}$ ;     // Use future HER strategy
    end
    # Relabel expert transitions
    $g^j \leftarrow s_{t+k'}^j, \quad k' \sim \text{Unif}\{t+1, \dots, T^j\}$
    $r_t^h = \mathbb{1}\left[s_{t+1}^h == g^h\right]$
    $\psi \leftarrow \min_\psi \mathcal{L}_{GAIL}(D_\psi, \mathcal{D}, \mathcal{R})$ (Eq. 4.3)
    $r_t^h = (1 - \delta_{GAIL}) r_t^h + \delta_{GAIL} \log D_\psi(a_t^h, s_t^h, g^h)$ ;        // Add annealed GAIL
     reward
    # Fit $Q_\phi$
    $y_t^h = r_t^h + \gamma Q_\phi(\pi(s_{t+1}^h, g^h), s_{t+1}^h, g^h)$ ;    // Use target networks $Q_{\phi'}$ for
     stability
    $\phi \leftarrow \min_\phi \sum_h \|Q_\phi(a_t^h, s_t^h, g^h) - y_t^h\|$
    # Update Policy
    $\theta + = \alpha \nabla_\theta \hat{J}$ (Eq. 4.2)
    Anneal $\delta_{GAIL}$ ;                    // Ensures outperforming the expert
end
```

- Without supervision from reward, can *goalGAIL* use demonstrations to accelerate the learning of goal-conditioned tasks and outperform the demonstrator?
- Is the *Expert Relabeling* an efficient way of doing data-augmentation on the demonstrations?
- Compared to Behavioral Cloning methods, is *goalGAIL* more robust to expert action noise?
- Can *goalGAIL* leverage state-only demonstrations equally well as full trajectories?

We evaluate these questions in four different simulated robotic goal-conditioned tasks that are detailed in the next subsection along with the performance metric used throughout the experiments section. All the results use 20 demonstrations reaching uniformly sampled goals. All curves have 5 random seeds and the shaded area is one standard deviation.



(a) Four rooms (b) Block pusher (c) Fetch Pick & Place (d) Fetch Stack Two

Figure 4.2: Four continuous goal-conditioned environments where we tested the effectiveness of the proposed algorithm *goalGAIL* and expert relabeling technique.

Tasks

Experiments are conducted in four continuous environments in MuJoCo [166]. The performance metric we use in all our experiments is the percentage of goals in the feasible goal space the agent is able to reach. We call this metric coverage. To estimate this percentage we sample feasible goals uniformly, and execute a rollout of the current policy. It is consider a success if the agent reaches within ϵ of the desired goal.

Four rooms environment: This is a continuous twist on a well studied problem in the Reinforcement Learning literature. A point mass is placed in an environment with four rooms connected through small openings as depicted in Fig. 4.2a. The action space of the agent is continuous and specifies the desired change in state space, and the goals-space exactly corresponds to the state-space.

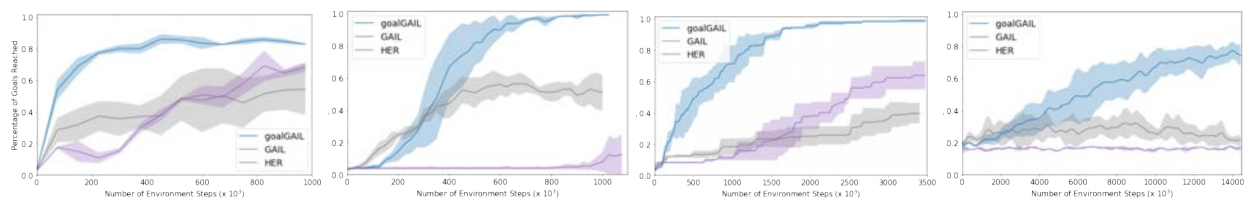
Pointmass Block Pusher: In this task, a Pointmass needs to navigates itself to the block, push the block to a desired position (x, y) and then eventually stops a potentially different spot (a, b). The action space is two dimensional as in four rooms environment. The goal space is four dimensional and specifies (x, y, a, b).

Fetch Pick and Place: This task is the same as the one described by Nair et al. [105], where a fetch robot needs to pick a block and place it in a desired point in space. The control is four-dimensional, corresponding to a change in (x, y, z) position of the end-effector as well as a change in gripper opening. The goal space is three dimensional and is restricted to the position of the block.

Fetch Stack Two: A Fetch robot stacks two blocks on a desired position, as also done in Nair et al. [105]. The control is the same as in Fetch Pick and Place while the goal space is now the position of two blocks, which is six dimensional.

Goal-conditioned GAIL with Hindsight: goalGAIL

In goal-conditioned tasks, HER [2] should eventually converge to a policy able to reach any desired goal. Nevertheless, this might take a long time, specially in environments where there are bottlenecks that need to be traversed before accessing a whole new area of the goal space.



(a) Four rooms (b) Block pusher (c) Fetch Pick & Place (d) Fetch Stack Two

Figure 4.3: In all four environments, the proposed algorithm *goalGAIL* takes off and converges faster than HER by leveraging demonstrations. It is also able to outperform the demonstrator unlike standard GAIL, the performance of which is capped.

In this section we show how the methods introduced in the previous section can leverage a few demonstrations to improve the convergence speed of HER. This was already studied for the case of Behavioral Cloning by [105], and in this work we show we also get a benefit when using GAIL as the Imitation Learning algorithm, which brings considerable advantages over Behavioral Cloning as shown in the next sections. In all four environments, we observe that our proposed method *goalGAIL* considerably outperforms the two baselines it builds upon: HER and GAIL. HER alone has a very slow convergence, although as expected it ends up reaching the same final performance if run long enough. On the other hand GAIL by itself learns fast at the beginning, but its final performance is capped. This is because despite collecting more samples on the environment, those come with no reward of any kind indicating what is the task to perform (reach the given goals). Therefore, once it has extracted all the information it can from the demonstrations it cannot keep learning and generalize to goals further from the demonstrations. This is not an issue anymore when combined with HER, as our results show.

Expert relabeling

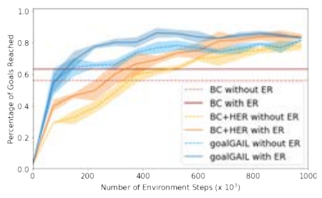
Here we show that the Expert Relabeling technique introduced in Section 4.4 is an effective means of data augmentation on demonstrations. We show the effect of Expert Relabeling on three methods: standard behavioral cloning (*BC*), HER with a behavioral cloning loss (*BC+HER*) and *goalGAIL*. For *BC+HER*, the gradient of the behavior cloning loss $\mathcal{L}_{\text{BC}}$ (Equation 4.1) is combined with the gradient of the policy objective $\nabla_\theta \hat{J}$ (Equation 4.2). The resulting gradient for the policy update is:

$$\nabla_\theta \hat{J}_{\text{BC+HER}} = \nabla_\theta \hat{J} - \beta \nabla_\theta \mathcal{L}_{\text{BC}}$$

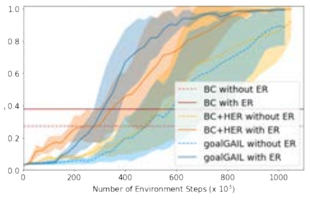
where β is the weight of the BC loss and is annealed to enable the agent to outperform the expert.

As shown in Fig. 4.4, our expert relabeling technique brings considerable performance boosts for both Behavioral Cloning methods and *goalGAIL* in all four environments.

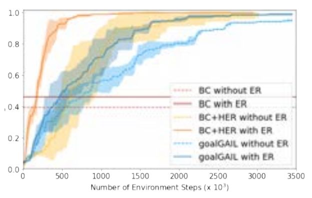
We also perform a further analysis of the benefit of the expert relabeling in the four-rooms environment because it is easy to visualize in 2D the goals the agent can reach. We see in Fig. 4.1



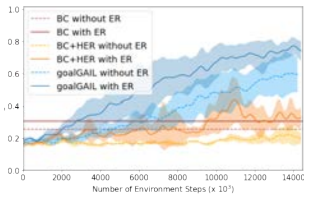
(a) Four Rooms



(b) Block Pusher



(c) Fetch Pick & Place



(d) Fetch Stack Two

Figure 4.4: Our Expert Relabeling technique boosts final performance of standard BC. It also accelerates convergence of BC+HER and *goalGAIL* on all four environments.

that without the expert relabeling, the agent fails to learn how to reach many intermediate states visited in demonstrations.

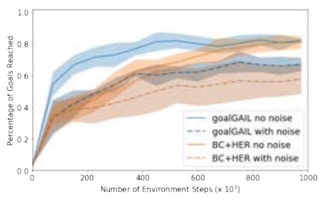
The performance of running pure Behavioral Cloning is plotted as a horizontal dotted line given that it does not require any additional environment steps. We observe that combining BC with HER always produces faster learning than running just HER, and it reaches higher final performance than running pure BC with only 20 demonstrations.

Robustness to sub-optimal expert

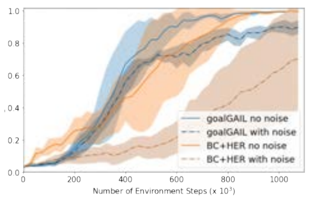
In the above sections we were assuming access to perfectly optimal experts. Nevertheless, in practical applications the experts might have a more erratic behavior, not always taking the best action to go towards the given goal. In this section we study how the different methods perform when a sub-optimal expert is used. To do so we collect sub-optimal demonstration trajectories by adding noise α to the optimal actions, and making it be ϵ-greedy. Thus, the sub-optimal expert is $a = \mathbb{1}[r < \epsilon]u + \mathbb{1}[r > \epsilon](\pi^*(a|s,g) + \alpha)$, where $r \sim \texttt{Unif}(0,1)$, $\alpha \sim \mathcal{N}(0, \sigma_\alpha^2 I)$ and u is a uniformly sampled random action in the action space.

In Fig. 4.5 we observe that approaches that directly try to copy the action of the expert, like Behavioral Cloning, greatly suffer under a sub-optimal expert, to the point that it barely provides any improvement over performing plain Hindsight Experience Replay. On the other hand, methods based on training a discriminator between expert and current agent behavior are able to leverage much noisier experts. A possible explanation of this phenomenon is that a discriminator approach can give a positive signal as long as the transition is "in the right direction", without trying to exactly enforce a single action. Under this lens, having some noise in the expert might actually improve the performance of these adversarial approaches, as it has been observed in many generative models literature [46].

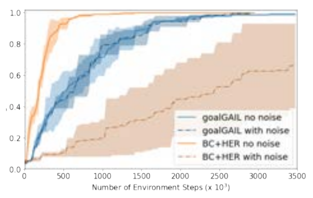
Using state-only demonstrations



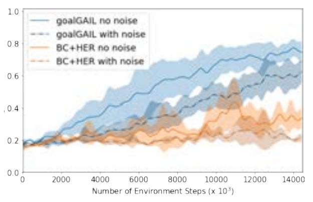
(a) Four Rooms



(b) Block Pusher



(c) Fetch Pick & Place



(d) Fetch Stack Two

Figure 4.5: Effect of sub-optimal demonstrations on *goalGAIL* and Behavorial Cloning method. We produce sub-optimal demonstrations by making the expert ϵ-greedy and adding Gaussian noise to the optimal actions.

Behavioral Cloning and standard GAIL rely on the state-action (s, a) tuples coming from the expert. Nevertheless there are many cases in robotics where we have access to demonstrations of a task, but without the actions. In this section we want to emphasize that all the results obtained with our *goalGAIL* method and reported in Fig. 4.3 and Fig. 4.4 do *not* require any access to the action that the expert took. Surprisingly, in all environments but Fetch Pick & Place, despite the more restricted information *goalGAIL* has access to, it outperforms BC combined with HER. This might be due to the superior imitation learning performance of GAIL, and also to the fact that these tasks are solvable by only matching the state-distribution of the expert. We run the experiment of training GAIL only conditioned on the current state, and not the action (as also done in other non-goal-conditioned works [41]), and we observe that the discriminator learns a very well shaped reward that clearly encourages the agent to go towards the goal, as pictured in Fig. 4.6. See the Appendix for more details.

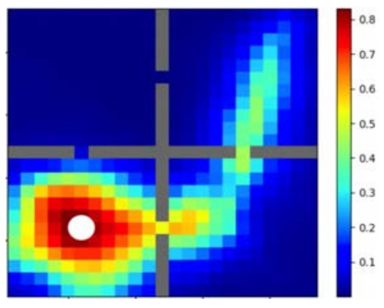


Figure 4.6: Output of the Discriminator $D(\cdot, g)$ for the four rooms environment. The goal is the lower left white dot, and the start is at the top right.

4.6 Guided Uncertainty-Aware Policy Optimization

Traditional robotic approaches rely on an accurate model of the environment, a detailed description of how to perform the task, and a robust perception system to keep track of the current state. On the other hand, reinforcement learning approaches can operate directly from raw sensory inputs with only a reward signal to describe the task, but are extremely sample-inefficient and brittle. In this work, we combine the strengths of model-based methods with the flexibility of learning-based methods to obtain a general method that is able to overcome inaccuracies in the robotics perception/actuation pipeline, while requiring minimal interactions with the environment. This is achieved by leveraging uncertainty estimates to divide the space in regions where the given model-based policy is reliable, and regions where it may have flaws or not be well defined. In these uncertain regions, we show that a locally learned-policy can be used directly with raw sensory inputs. We test our algorithm, Guided Uncertainty-Aware Policy Optimization (GUAPO), on a real-world robot performing peg insertion. Videos are available at: `https://sites.google.com/view/guapo-rl`.

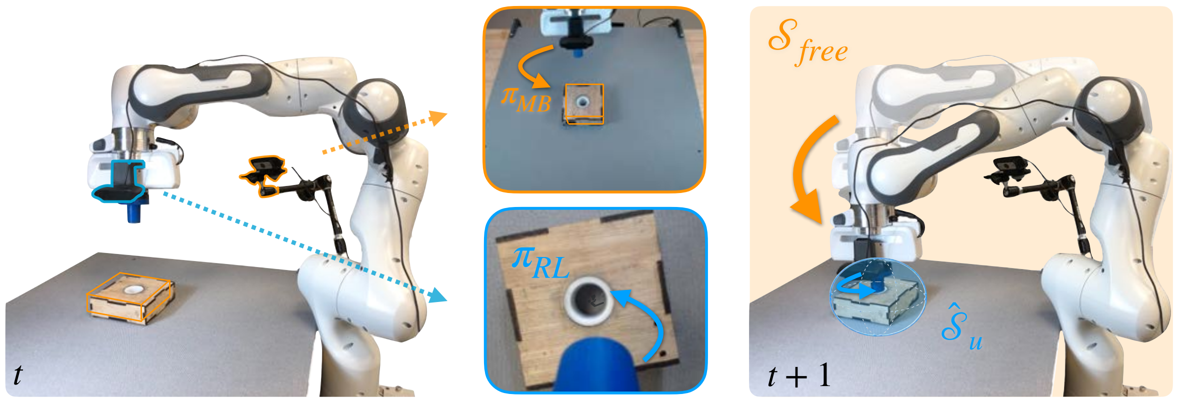


Figure 4.7: Real-world setup for peg insertion: one perception system (in orange) gives the approximate position of the relevant objects. The *model-based* method, π_{MB}, drives the system from free space, $\mathcal{S}_{free}$, to the uncertainty area, $\hat{\mathcal{S}}_u$ (in blue). Once inside this area, the model can't be trusted, and a *reinforcement learning* policy, π_{RL}, is then learned directly from the raw sensory inputs of another "local" camera (in blue) that gives enough information to complete the task.

Modern robots rely on extensive systems to accomplish a given task, such as a perception module to monitor the state of the world [133, 19, 181]. Simple perception failure in this context is catastrophic for the robot, since its motion generator relies on it. Moreover, classic motion generators are quite rigid in how they accomplish a task, *e.g.*, the robot has to pick an object in a specific way and might not recover if the grasp fails. These problems make robotics systems unstable, and hard to scale to new domains. In order to expand robotics reach we need more robust, adaptive, and flexible systems.

Learning-based method, such as Reinforcement Learning (RL) has the capacity to adapt, and deal directly with raw sensory inputs, which are not subject to estimation errors [106, 89]. The strength of RL stems from its capacity to define a task at a higher level through a reward function indicating *what* to do, not through an explicit set of control actions describing *how* the task should be performed. RL does not need specific physical modelling as they implicitly learn a data-driven model from interacting with the environment [17], allowing the method to be deployed in different settings. These characteristics are desired but come with different limitations: 1) randomly interacting with an environment can be quite unsafe for the human users as well as for the equipment, 2) RL is not recognized for being sample efficient. As such, introducing RL to a new environment can be time consuming and difficult.

Classic robotic approaches have mastered generating movements within free space, where there are no contacts with other elements in the environment [123]. We refer to these accessible methods as Model Based (MB) methods. One of their main limitations is that they normally do not handle perception errors and physical interactions naturally, *e.g.*, grasping an object, placing an object, object insertion, *etc.* As such this limits the expressiveness of roboticists and the reliability of the system.

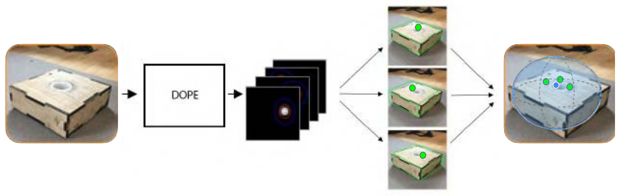
In this work we present an algorithmic framework that is aware of its own uncertainty in the perception and actuation system. As such a MB guides the agent to the relevant region, hence reducing the area where the RL policy needs to be optimized and making it more invariant to the absolute goal location. Our novel algorithm combines the strengths from MB and RL. We leverage the efficiency of MB to move in free-space, and the capacity of RL to learn from its environment from a loosely defined goal. In order to efficiently fuse MB and RL, we introduce a perception system that provides uncertainty estimates of the region where contacts might occur. This uncertainty is used to determine the region where the MB method shouldn't be applied, and an RL policy should be learned instead. Therefore, we call our algorithm Guided Uncertainty Aware Policy Optimization (GUAPO).

Figure 4.7 shows an overview of our system, the task is initialized with MB where it guides the robot within the range of the uncertainties of the object of interest, *e.g.*, the box where to insert the peg. Once we have reached that region, we switch to RL to complete the task. At learning time, we leverage information from task completion by the RL policy to reduce our perception system's uncertainties. This work makes the following contributions:

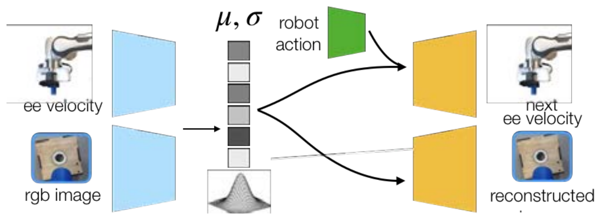
- We demonstrate that GUAPO outperforms pure RL, pure MB, as well as a Residual policy baseline [143, 66] that combines MB and RL for peg insertion;
- We present a simple and yet efficient way to express pose uncertainties for a keypoint based pose estimator;
- We show that our approach is sample efficient for learning methods on real-world robots.

Definitions and Formulation

We tackle the problem of learning to perform an operation, unknown *a-priori*, in an area of which we only have an estimated location and no accurate model. We formalize this problem as a Markov Decision Process (MDP), where we want to find a policy $\pi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}_+$ that is a probability distribution over actions $a \in \mathcal{A}$ conditioned on a given state $s \in \mathcal{S}$. We optimize this policy to maximize the expected return $\sum_{t=0}^{H} \gamma^t r(s_t, a_t)$, where $r : \mathcal{S} \rightarrow \mathbb{R}$ is a given reward function, H is the horizon of each rollout, and γ is the discount factor. The first assumption we leverage in this work can be expressed as having a partial knowledge of the transition function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}_+$ dictating the probability over next states when executing a certain action in the current state. Specifically, we assume this transition function is available only within a sub-space



(a) DOPE perception and uncertainty to estimate $\hat{\mathcal{S}}_u$



(b) Variational autoencoder for π_{RL}

Figure 4.8: Perception modules for the model-based component (left) and reinforcement learning component (right).

of the state-space $\mathcal{S}_{free} \subset \mathcal{S}$. This is a common case in robotics, where it is perfectly known how the robot moves while it is in free-space, but there are no reliable and accurate models of general contacts and interactions with its surrounding [43]. This partial model can be combined with well established methods able to plan and execute trajectories that traverse $\mathcal{S}_{free}$ [124, 135, 82], but these methods cannot successfully complete tasks that require acting in $\mathcal{S}_u = \mathcal{S} \setminus \mathcal{S}_{free}$. It is usually easy for the user to specify this split relative to the objects in the scene, *e.g.*, all points around or in contact with an object are not free space. If we call a point of interest, g, in that region we can therefore express that region relative to it as $\mathcal{S}_u(g)$. We do not assume perfect knowledge of the absolute coordinates of that point g nor of $\mathcal{S}_u$, but rather only a noisy estimate of them as described in Section 4.6.

The tasks we consider consist on reaching a particular state or configuration through interaction with the environment, like peg-insertion, switch-toggling, or grasping. These tasks are conveniently defined by a binary reward function $r(s) = \mathbb{1}[s \in \mathcal{S}_g]$ that indicates having successfully reached a goal set $\mathcal{S}_g \subset \mathcal{S}_u$, usually described with respect to a point g [129, 39, 36]. Unfortunately this reward is extremely sparse, and random actions can take an prohibitive amount of samples to discover it [30, 35]. Therefore this paper addresses how to leverage the partial model described above to efficiently learn to solve the full task through interactions with the environment, overcoming an imperfect perception system and dynamics.

In the following subsections, we describe the different components of GUAPO. We define $\hat{\mathcal{S}}_u$ as a super-set of $\mathcal{S}_u$ generated from the perception system uncertainty estimation. We use this set to partition the space into the regions where the MB method is used, and regions where the RL policy is trained. Then we describe a MB method that can now confidently be used outside of $\hat{\mathcal{S}}_u$ to bring the robot within that set. Finally we define the RL policy, and how the learning can be more efficient by making its inputs local. We also outline our algorithm in Algorithm 8.

From coarse perception to the RL workspace

Coarse perception systems are usually cheaper and faster to setup because they might require simpler hardware like RGB cameras, and can be used out-of-the-box without excessive tuning and calibration efforts [168]. If we use such a system to directly localize $\mathcal{S}_u$, the perception errors might misleadingly indicate that a certain area belongs to $\mathcal{S}_{free}$, hence trying to apply the MB method and potentially not being able to learn how to recover from there. Instead, we propose to use a perception system that also gives an *uncertainty estimate*. Many methods can represent the uncertainty by a nonparametric distribution, with n possible descriptions of the region $\{\mathcal{S}_u^i\}_{i=1}^n$ and their associated weights $p(\mathcal{S}_u^i)$. By interpreting these weights as the likelihoods $P(\mathcal{S}_u^i = \mathcal{S}_u)$, we can express the likelihood of a certain state s belonging to $\mathcal{S}_u$ as:

$$p(s \in \mathcal{S}_u) = \sum_{i=1}^{n} \mathbb{1}\big[s \in \mathcal{S}_u^i\big] p(\mathcal{S}_u^i). \tag{4.4}$$

If the perception system provides a parametric distribution, the above probability can be computed by integration, or approximated in a way such that the set $\hat{\mathcal{S}}_u = \{s : p(s \in \mathcal{S}_u) > \epsilon\}$ is a super-set of $\mathcal{S}_u$ for an appropriate ϵ set by the user. A more accurate perception system would make $\hat{\mathcal{S}}_u$ a tighter super-set of $\mathcal{S}_u$. Now that we have an over-approximation of the area where we cannot use our model-based method, we define a function $\alpha(s) = \mathbb{1}[s \in \hat{\mathcal{S}}_u]$ indicating when to apply an *RL* policy $\pi_{RL}(a|s)$ instead of the given *MB* one π_{MB}. In short, GUAPO uses a hybrid policy presented in 4.5.

$$\pi(a|s) = (1 - \alpha(s)) \cdot \pi_{MB}(a|s) + \alpha(s) \cdot \pi_{RL}(a|s), \tag{4.5}$$

Therefore we use a switch between these two policies, based on the uncertainty estimate. A lower perception uncertainty reduces the area where the *reinforcement learning* method is required, and improves the overall efficiency. We now detail how each of these policies is obtained.

Model-based actuation

In the previous section we defined $\hat{\mathcal{S}}_u$, the region containing the goal set $\mathcal{S}_g$ and hence the agent's reward. In our problem statement we assume that outside that region, the environment model is well known, and therefore it is amenable to use a *model-based* approach. Therefore, whenever we are outside of $\hat{\mathcal{S}}_u$, the MB approach corrects any deviations.

Our formulation can be extended for obstacle avoidance. Using a similar approach used to over-estimate the set $\hat{\mathcal{S}}_u$, we can over-estimate the obstacle set to be avoided by $\hat{\mathcal{S}}_u^{obst}$, and remove that space from where the MB method can be applied, $\mathcal{S}_{free}$. An obstacle-avoiding MB method can be used to get to the area where the goal is, while avoiding the regions where the obstacle might be, as shown in our videos[2].

From Model-Based to Reinforcement learning

Once π_{MB} has brought the system within $\hat{\mathcal{S}}_u$, the control is handed-over to π_{RL} as expressed in Eq. 4.5. Note that our switching definition goes both ways, and therefore if π_{RL} takes exploratory actions that move it outside of $\hat{\mathcal{S}}_u$, the MB method will act again to funnel the state to the area of

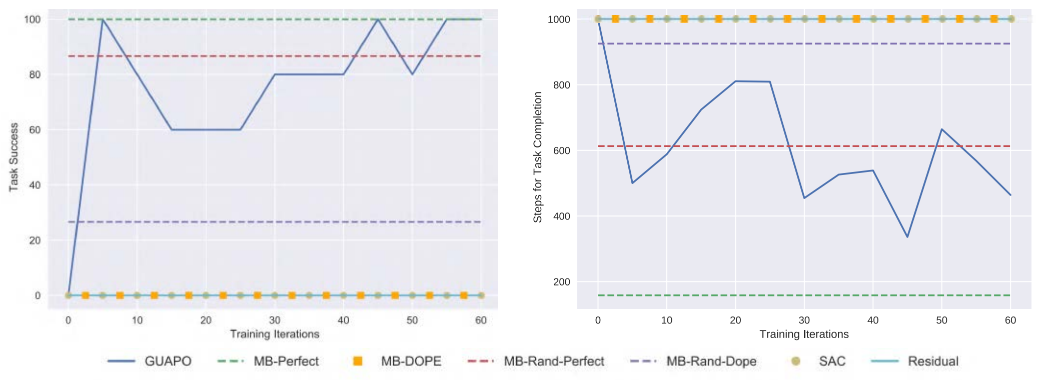
[2]https://sites.google.com/view/guapo-rl

interest. This also provides a framework for safe learning [34] in case there are obstacles to avoid as introduced in the section above. There are several advantages to having a more restricted area where the RL policy needs to learn how to act: first the exploration becomes easier, second, the policy can be local. In particular, we only feed to π_{RL} the images from a wrist-mounted camera and its current velocities, as depicted in Fig. 4.8b. Not using global information from our perception system in Fig. 4.8a can make our RL policy generalize better across locations of $\hat{\mathcal{S}}_u$. Finally, we propose to use an off-policy RL algorithm, so all the observed transitions can be added in the replay buffer, no matter if they come from π_{MB} or π_{RL}.

Closing the MB-RL loop

This framework also allows to use any newly acquired experience to reduce $\hat{\mathcal{S}}_u$ such that successive rollouts can use the *model-based* method in larger areas of the state-space. For example, in the peg-insertion task, once the reward of fully inserting the peg is received, the location of the opening is immediately known. Since we no longer need to rely on the noisy perception system to estimate the location of the hole, we can update $\hat{\mathcal{S}}_u = \mathcal{S}_u$, where now the reinforcement learning algorithm only needs to do the actual insertion and not keep looking for the opening.

4.7 Experimental Results for GUAPO



(a) Real Robot Task Success

(b) Steps for Peg Insertion

Figure 4.9: GUAPO is compared with five other methods: (1) Model-based policy with perfect goal estimate (MB-Perfect), (2) Model-based policy with additive random actions and perfect goal estimate (MB-Rand-Perfect), (3) Model-Based policy with additive random actions using DOPE goal estimates (MB-Rand-DOPE), (4) Reinforcement learning algorithm Soft Actor Critic (SAC), and (5) Residual policy. We train the policy for over 60 iterations, each with two episodes, 1000 steps long.

Algorithm 8: GUAPO

```
Input   : $s_0 \in \mathcal{S}$: reset state
$o^g$: global observation $\rightarrow$ RGB workspace camera
$o^l$: local observations $\rightarrow$ wrist-mounted camera, robot velocity observations
$\mathcal{S}_u$: model uncertain region containing the goal region $\mathcal{S}_g$, both unknown a-priori until
reward is obtained.
Output : $a_t$: robot actions

goal_localized $\leftarrow$ False
for each episode do
    $s_{t=0} \leftarrow s_0$ ;                                  // Reset robot
    if not goal_localized then
        $\hat{\mathcal{S}}_u \leftarrow$ DOPE($o^g$) ;                  // Run perception
    end
    for $t = 0 \ldots H$ do
        if robot not in $\hat{\mathcal{S}}_u$ then
            $a_t \leftarrow \pi_{MB}$ ;                          // Takes robot to $\hat{\mathcal{S}}_u$
        end
        else
            $a_t \leftarrow \pi_{RL}(o^l_t)$ ;
        end
        Apply action $a_t$ to environment ;
        $r_t = \mathbb{1}[s_t \in \mathcal{S}_g]$
        Add $(o^l_t, a_t, o^l_{t+1}, r_t)$ to replay buffer ;
        if $r_t$ = 1 then
            $\hat{\mathcal{S}}_u \leftarrow \mathcal{S}_u(s_t)$ ;                      // No DOPE uncert.
            goal_localized $\leftarrow$ True ;
        end
    end
end
```

Table 4.1: Real world peg insertion results out of 30 trials. All learning policies (SAC, Residual, and Guapo) trained for 120 episodes (which takes around 90 minutes). The first row indicates percentage success for a full peg insertion. The second row depicts the speed of insertion for the trained policies. The last two rows indicate the percentage the method enters $\mathcal{S}_u$ and $\hat{\mathcal{S}}_u$.

	MB-Perfect	MB-DOPE	MB-Rand-Perfect	MB-Rand-DOPE	SAC [51]	RESIDUAL [66]	GUAPO (ours)
Success Rate	100%	0%	86.67%	26.6%	0%	0%	93%
Avg. Steps for Task Completion	158.3	n/a	554.1	925.4	n/a	n/a	469.6
In $\mathcal{S}_u$	100%	0%	100%	70.0%	0%	0%	100%
In $\hat{\mathcal{S}}_u$	100%	100%	100%	93.3%	0%	100%	100%

In this section we seek to answer the following questions: How does our method compares to our baseline policies, such as, Residual policies, in terms of sample efficiency and task completion? And, is the proposed algorithm capable of performing peg insertion on a real robot?

Comparison Methods

All the different baselines were initialized about 75 cm away from the goal. They were all implemented on our real robotics system. As such we compare our proposed method to the following:

- **MB-Perfect.** This method consists of a scripted policy under perfect state estimation.
- **MB-Rand-Perfect.** This method uses the same policy as MB-Perfect where we injected random actions, which we sample from a normal distribution with 0 mean and a standard deviation defined by the perception uncertainty from DOPE (which is around 2.5cm to 3 cm).
- **MB-DOPE.** This method is similar to MB-Perfect, but instead uses the pose estimator prediction to servo to the hole and accomplish insertion.
- **MB-Rand-Dope.** This method uses the same policy as MB-Dope where we injected random actions, which is sampled in the same way as MB-Rand-Perfect.
- **SAC.** This uses just the policy learned from the RL algorithm, Soft-Actor Critic (SAC), to accomplish the task.
- **Residual.** This method is based off recent residual-learning techniques that combine model-based and reinforcement learning methods [143, 66].

Results

The results comparing the different methods is shown in Table 4.1, this table presents the success rate for insertion as well as the average number of steps needed for completion (a step is equivalent to 50 milliseconds of following the same robot command, as our policy is running at 20 Hz), and the percentage that the end-effector ends up in the $\mathcal{S}_u$ and $\hat{\mathcal{S}}_u$ regions over 30 trials. We also present training iteration performance (task success and steps to completion) for the different methods in Figure 4.9.

MB-Perfect is able to insert 100% of the time, as it has perfect knowledge of the state, and can be seen as an oracle. We can see that taking random actions with MB-Rand-Perfect does not degrade excessively the full performance achieved by MB-Perfect. However, when we used DOPE as the perception system, which has around 2.5 to 3.5 cm of noise and error, the performance of MB-DOPE and MB-Rand-DOPE drops drastically. MB-Rand-DOPE performs 26.6% better than MB-DOPE, as the random actions can help offset the perception error.

In our setup SAC did not achieve any insertion. This is due to the low number of samples that SAC trained on, since most success stories of RL in the real world require several orders of magnitude more data [91]. The Residual method also did not achieve any insertions. The Residual method often would apply large actions far away from the hole opening, and end up sliding off the box and getting stuck pushing against the side of the box. In comparison, GUAPO only turns on the reinforcement learning policy once it is already nearby the region of interest, and hence does not suffer from this. However, Residual was able to reach $\hat{\mathcal{S}}_u$ 100% of the time after 120 training episodes, while SAC never did.

In comparison, as seen in Fig. 4.9, after around 8 training iterations, GUAPO is also able to start inserting into the hole (which is about 12 minute real-world training time). As the policy trains, the average number of steps it takes to insert the peg also decreases. After 120 training episodes (and 90 minutes of training), GUAPO is able to achieve 93% insertion rate.

4.8 Conclusions and Future Work

Hindsight relabeling can be used to learn useful behaviors without any reward supervision for goal-conditioned tasks, but they are inefficient when the state-space is large or includes exploration bottlenecks. In this work we show how only a few demonstrations can be leveraged to improve the convergence speed of these methods. We introduce a novel algorithm, *goalGAIL*, that converges faster than HER and to a better final performance than a naive goal-conditioned GAIL. We also study the effect of doing expert relabeling as a type of data augmentation on the provided demonstrations, and demonstrate it improves the performance of our *goalGAIL* as well as goal-conditioned Behavioral Cloning. We emphasize that our *goalGAIL* method only needs state demonstrations, without using expert actions like other Behavioral Cloning methods. Finally, we show that *goalGAIL* is robust to sub-optimalities in the expert behavior.

All the above factors make our *goalGAIL* algorithm very suited for real-world robotics. This is a very exciting future work. In the same line, we also want to test the performance of these methods in vision-based tasks. Our preliminary experiments show that Behavioral Cloning fails completely in the low data regime in which we operate (less than 20 demonstrations).

We also introduce a novel algorithm, Guided Uncertainty Aware Policy Optimization (GUAPO), that combines the generalization capabilities of model-based methods and the adaptability of learning-based methods. It allows to loosely define the task to perform, by solely providing a coarse model of the objects, and a rough description of the area where some operation needs to be performed. The model-based system leverage this high-level information and accessible state estimation systems to create a funnel around the area of interest. We use the uncertainty estimate provided by the perception system to automatically switch between the model-based policy, and a learning-based policy that can learn from an easy-to-define sparse reward, overcoming the model and estimation errors of the model-based part. We show learning in the real world of a peg insertion task.

Chapter 5

Conclusions

Learning through interaction with weak supervision is one of the hallmarks of Intelligence. This dissertation is our contribution towards understanding what types of supervision scale well enough to enable general agents to tackle diverse ranges of tasks. To this end, we presented two methods on Hierarchical Reinforcement Learning, allowing to learn only from a sparse reward signal in long-horizon tasks. The re-usability of the skills allows to reduce the sample complexity on many related tasks. Then we explicitly considered the objective of performing well on a distribution of tasks, which lead to proposing two innovative curriculum learning algorithms, enabling robotic agents to learn some of the most complex manipulation tasks seen in this body of literature. Finally, we considered the cost of using other sources of supervision, like partial experts. We proposed an algorithm that leverages and outperform sub-optimal demonstrations without the need of any additional reward signal, and another algorithm that leverages and outperform an expert that can only be queried in a fraction of the state-space.

We hope this dissertation inspires other researchers to evaluate the cost of providing the supervision required by their algorithms under the three axis proposed here: how much domain knowledge is required to obtain that supervision form, what is the total volume of interaction under such supervision needed to learn a task, and how does this amount increases for every new task that the agent has to learn. This considerations will only get more important as the field of autonomous agents starts to move away from simulation and games to tackle more impactful real-world problems.

Bibliography

[1] Jacob Andreas, Dan Klein, and Sergey Levine. "Modular Multitask Reinforcement Learning with Policy Sketches". In: *International Conference in Machine Learning* (2017). URL: `http://github.com/`.

[2] Marcin Andrychowicz et al. "Hindsight Experience Replay". In: *Advances in Neural Information Processing Systems* (2017). ISSN: 10495258. DOI: `10.1016/j.surfcoat.2018.06.018`. URL: `http://arxiv.org/abs/1707.01495`.

[3] Marcin Andrychowicz et al. "Learning Dexterous In-Hand Manipulation". In: (), pp. 1–27.

[4] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein GAN". In: *Stat*. 2017.

[5] Minoru Asada et al. "Purposive Behavior Acquisition for a Real Robot by Vision-Based Reinforcement Learning". en. In: *Machine Learning* (1996).

[6] Pierre-Luc Bacon, Jean Harb, and Doina Precup. "The Option-Critic Architecture". In: *AAAI* (2017), pp. 1726–1734. URL: `http://arxiv.org/abs/1609.05140`.

[7] Pierre-Luc Bacon and Doina Precup. "The option-critic architecture". In: *arXiv preprint arXiv:1609.05140v2* (2016).

[8] J Andrew Bagnell et al. "Policy search by dynamic programming". In: *Advances in Neural Information Processing Systems* 16 (2003), p. 79.

[9] Adrien Baranes and Pierre-Yves Oudeyer. "Active Learning of Inverse Models with Intrinsically Motivated Goal Exploration in Robots". In: *Robotics and Autonomous Systems* 61.1 (2013).

[10] Adrien Baranes and Pierre-Yves Oudeyer. "Active learning of inverse models with intrinsically motivated goal exploration in robots". In: *Robotics and Autonomous Systems* 61.1 (2013), pp. 49–73.

[11] Marc G Bellemare et al. "Unifying Count-Based Exploration and Intrinsic Motivation". In: *Advances in Neural Information Processing Systems* (2016).

[12] Samy Bengio et al. "Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks". In: *Advances in Neural Information Processing Systems*. 2015.

[13] Yoshua Bengio et al. "Curriculum learning". In: *International Conference on Machine Learning*. ACM. 2009, pp. 41–48.

[14] Lionel Blondé and Alexandros Kalousis. “Sample-Efficient Imitation Learning via Generative Adversarial Nets”. In: *AISTATS* (2019). URL: `https://youtu.be/-nCsqUJnRKU.`.

[15] Mariusz Bojarski et al. “End to End Learning for Self-Driving Cars”. In: (2016). URL: `http://arxiv.org/abs/1604.07316`.

[16] Robert R Burridge, Alfred A Rizzi, and Daniel E Koditschek. “Sequential composition of dynamically dexterous robot behaviors”. In: *The International Journal of Robotics Research* (1999).

[17] Yevgen Chebotar et al. “Closing the sim-to-real loop: Adapting simulation randomization with real world experience”. In: *arXiv preprint arXiv:1810.05687* (2018).

[18] Xi Chen et al. “Infogan: Interpretable representation learning by information maximizing generative adversarial nets”. In: *Advances in Neural Information Processing Systems* (2016).

[19] Ching-An Cheng et al. “RMPflow : A Computational Graph for Automatic Motion Policy Generation”. In: *preprint arxiv:1811.07049* (2019). URL: `https://arxiv.org/pdf/1811.07049.pdf`.

[20] Nuttapong Chentanez, Andrew G Barto, and Satinder P Singh. “Intrinsically motivated reinforcement learning”. In: *Advances in Neural Information Processing Systems*. 2004, pp. 1281–1288.

[21] Kyung Hyun Cho, Tapani Raiko, and Alexander Ilin. “Gaussian-bernoulli deep boltzmann machine”. In: *International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2013, pp. 1–7.

[22] Christian Daniel, Gerhard Neumann, and Jan Peters. “Autonomous reinforcement learning with hierarchical REPS”. In: *International Joint Conference on Neural Networks*. IEEE. 2013, pp. 1–8.

[23] Christian Daniel, Gerhard Neumann, and Jan Peters. “Hierarchical Relative Entropy Policy Search.” In: *AISTATS*. 2012, pp. 273–281.

[24] Christian Daniel et al. “Probabilistic inference for determining options in reinforcement learning”. In: *Machine Learning* 104.104 (2016). DOI: `10.1007/s10994-016-5580-x`.

[25] Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. “A survey on policy search for robotics”. In: *Foundations and Trends in Robotics* 2.1–2 (2013), pp. 1–142.

[26] Marc Peter Deisenroth et al. “Multi-task policy search for robotics”. In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE. 2014, pp. 3876–3881.

[27] Coline Devin et al. “Learning Modular Neural Network Policies for Multi-Task and Multi-Robot Transfer”. In: *International Conference on Robotics and Automation*. 2017.

[28] Thomas G Dietterich. “Hierarchical reinforcement learning with the MAXQ value function decomposition”. In: *Journal of Artificial Intelligence Research* 13 (2000), pp. 227–303.

[29] Yiming Ding* et al. "Goal-conditioned Imitation Learning". In: *Advances in Neural Information Processing Systems* (2019).

[30] Yan Duan et al. "Benchmarking Deep Reinforcement Learning for Continuous Control". In: *International Conference on Machine Learning* (2016).

[31] Benjamin Eysenbach et al. "Diversity is All You Need: Learning Skills without a Reward Function". In: *International Conference in Learning Representations* (2019). URL: http://arxiv.org/abs/1802.06070.

[32] Alexander Fabisch and Jan Hendrik Metzen. "Active contextual policy search." In: *Journal of Machine Learning Research* 15.1 (2014), pp. 3371–3399.

[33] Chelsea Finn, Sergey Levine, and Pieter Abbeel. "Guided Cost Learning: Deep Inverse Optimal Control via Policy Optimization". In: *Internation Conference in Machine Learning* (Mar. 2016). URL: http://arxiv.org/abs/1603.00448.

[34] Jaime F. Fisac et al. "A General Safety Framework for Learning-Based Control in Uncertain Robotic Systems". In: *preprint arxiv:1705.01292* (May 2017). URL: http://arxiv.org/abs/1705.01292.

[35] Carlos Florensa, Yan Duan, and Pieter Abbeel. "Stochastic Neural Networks for Hierarchical Reinforcement Learning". In: *International Conference in Learning Representations* (2017), pp. 1–17.

[36] Carlos Florensa et al. "Reverse Curriculum Generation for Reinforcement Learning". In: *Conference on Robot Learning* (2017).

[37] Carlos Florensa et al. "Reverse Curriculum Generation for Reinforcement Learning". In: *Conference on Robot Learning (CoRL)* (2017).

[38] Carlos Florensa et al. "Self-supervised Learning of Image Embedding for Continuous Control". In: *Workshop on Inference to Control at NeurIPS*. 2018. URL: http://arxiv.org/abs/1901.00943.

[39] Carlos Florensa* et al. "Automatic Goal Generation for Reinforcement Learning Agents". In: *International Conference in Machine Learning* (2018).

[40] Kevin Frans et al. "Meta Learning Shared Hierarchies". In: *International Conference in Learning Representations* (2018), pp. 1–11. ISSN: 14639076. DOI: 10.1039/b203755f. URL: http://arxiv.org/abs/1710.09767.

[41] Justin Fu, Katie Luo, and Sergey Levine. "Learning Robust Rewards with Adversarial Inverse Reinforcement Learning". In: *International Conference in Learning Representations* (Oct. 2018). URL: http://arxiv.org/abs/1710.11248.

[42] Akira Fukui et al. "Multimodal Compact Bilinear Pooling for Visual Question Answering and Visual Grounding". In: *EMNLP*. 2016.

[43] Shameek Ganguly and Oussama Khatib. "Experimental studies of contact space model for multi-surface collisions in articulated rigid-body systems". In: *International Symposium on Experimental Robotics*. Springer. 2018.

[44] Yang Gao et al. "Reinforcement Learning from Imperfect Demonstrations". In: *Internation Conference in Machine Learning* (2018).

[45] Mohammad Ghavamzadeh and Sridhar Mahadevan. "Hierarchical Policy Gradient Algorithms". In: *International Conference in Machine Learning* (2003). URL: `http://chercheurs.lille.inria.fr/~ghavamza/my_website/Publications_files/icml03.pdf`.

[46] Ian J Goodfellow et al. "Generative Adversarial Nets". In: (), pp. 1–9.

[47] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in Neural Information Processing Systems*. 2014, pp. 2672–2680.

[48] Alex Graves et al. "Automated Curriculum Learning for Neural Networks". In: (2017). URL: `https://arxiv.org/pdf/1704.03003.pdf%20http://arxiv.org/abs/1704.03003`.

[49] Tuomas Haarnoja et al. "Composable Deep Reinforcement Learning for Robotic Manipulation". In: *Proceedings - IEEE International Conference on Robotics and Automation*. 1. 2018, pp. 6244–6251. ISBN: 9781538630815. DOI: `10.1109/ICRA.2018.8460756`.

[50] Tuomas Haarnoja et al. "Latent Space Policies for Hierarchical Reinforcement Learning". In: *Internation Conference in Machine Learning* (2018). URL: `http://arxiv.org/abs/1804.02808`.

[51] Tuomas Haarnoja et al. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *Internation Conference in Machine Learning* (2018), pp. 1–15.

[52] Jean Harb et al. "When Waiting is not an Option : Learning Options with a Deliberation Cost". In: *AAAI* (Sept. 2017). URL: `http://arxiv.org/abs/1709.04571`.

[53] Karol Hausman et al. "Learning an Embedding Space for Transferable Robot Skills". In: *International Conference in Learning Representations* (2018), pp. 1–16.

[54] Nicolas Heess et al. "Emergence of Locomotion Behaviours in Rich Environments". In: (July 2017). URL: `http://arxiv.org/abs/1707.02286`.

[55] Nicolas Heess et al. "Learning and Transfer of Modulated Locomotor Controllers". In: *arXiv preprint arXiv:1610.05182* (2016).

[56] Nicolas Heess et al. "Learning and Transfer of Modulated Locomotor Controllers". In: (2016). URL: `https://arxiv.org/abs/1610.05182`.

[57] S. Hinterstoisser et al. "Model based training, detection and pose estimation of texture-less 3D objects in heavily cluttered scenes". In: *ACCV*. 2012.

[58] Geoffrey E Hinton. "Training products of experts by minimizing contrastive divergence". In: *Neural Computation* 14.8 (2002), pp. 1771–1800.

[59] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. "A fast learning algorithm for deep belief nets". In: *Neural Computation* 18.7 (2006), pp. 1527–1554.

[60] Jonathan Ho and Stefano Ermon. “Generative Adversarial Imitation Learning”. In: *Advances in Neural Information Processing Systems* (2016). URL: `http://arxiv.org/abs/1606.03476`.

[61] T. Hodaň et al. “T-LESS: An RGB-D Dataset for 6D Pose Estimation of Texture-less Objects”. In: *WACV*. 2017.

[62] Rein Houthooft et al. “Variational Information Maximizing Exploration”. In: *Advances in Neural Information Processing Systems* (2016).

[63] Yinlin Hu et al. “Segmentation-driven 6D object pose estimation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 3385–3394.

[64] Eric Jang, Shixiang Gu, and Ben Poole. “Categorical Reparameterization with Gumbel-Softmax”. In: *International Conference on Learning Representations*. 2017.

[65] Lu Jiang et al. “Self-Paced Curriculum Learning.” In: *AAAI*. Vol. 2. 2015, p. 6.

[66] Tobias Johannink et al. “Residual Reinforcement Learning for Robot Control”. In: *preprint arxiv:1812.03201* (2019), pp. 1–9. DOI: `arXiv:1812.03201v1`.

[67] Lars Johannsmeier, Malkin Gerchow, and Sami Haddadin. *A Framework for Robot Manipulation: Skill Formalism, Meta Learning and Adaptive Control*. Tech. rep. Technische Universitat Munchen, 2019. URL: `https://arxiv.org/pdf/1805.08576.pdf`.

[68] Leslie P. Kaelbling. “Learning to Achieve Goals”. In: *International Joint Conference on Artificial Intelligence (IJCAI)* (1993), pp. 1094–1098.

[69] Sham Kakade. “A Natural Policy Gradient”. In: *Advances in Neural Information Processing Systems* (2002).

[70] Sham Kakade and John Langford. “Approximately Optimal Approximate Reinforcement Learning”. In: *International Conference in Machine Learning* (2002). URL: `https://people.eecs.berkeley.edu/~pabbeel/cs287-fa09/readings/KakadeLangford-icml2002.pdf`.

[71] Mrinal Kalakrishnan et al. “Learning locomotion over rough terrain using terrain templates”. In: *International Conference on Intelligent Robots and Systems*. IEEE, 2009, pp. 167–172. ISBN: 978-1-4244-3803-7. DOI: `10.1109/IROS.2009.5354701`. URL: `http://ieeexplore.ieee.org/document/5354701/`.

[72] Andrej Karpathy and Michiel Van De Panne. “Curriculum learning for motor skills”. In: *Canadian Conference on Artificial Intelligence*. 2012, pp. 325–330.

[73] Michael Kearns, Yishay Mansour, and Andrew Y. Ng. “A Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes”. In: *Machine Learning* 49.2/3 (2002), pp. 193–208. ISSN: 08856125. DOI: `10.1023/A:1017932429737`. URL: `http://link.springer.com/10.1023/A:1017932429737`.

[74] Chung Hee Kim and Jungwon Seo. “Shallow-Depth Insertion: Peg in Shallow Hole Through Robotic In-Hand Manipulation”. In: *IEEE Robotics and Automation Letters* 4.2 (Apr. 2019), pp. 383–390. ISSN: 2377-3766. DOI: `10.1109/LRA.2018.2890449`. URL: `https://ieeexplore.ieee.org/document/8598749/`.

[75] George Konidaris and Andrew G Barto. “Building Portable Options: Skill Transfer in Reinforcement Learning.” In: *IJCAI*. Vol. 7. 2007, pp. 895–900.

[76] George Konidaris et al. “Autonomous Skill Acquisition on a Mobile Manipulator.” In: *AAAI*. 2011.

[77] Ilya Kostrikov et al. “DISCRIMINATOR-ACTOR-CRITIC: ADDRESSING SAMPLE INEFFICIENCY AND REWARD BIAS IN ADVERSARIAL IMITATION LEARNING”. In: *International Conference in Learning Representations* (2019).

[78] J.J. Kuffner and S.M. LaValle. “RRT-connect: An efficient approach to single-query path planning”. In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*. Vol. 2. IEEE, pp. 995–1001. ISBN: 0-7803-5886-4. DOI: `10.1109/ROBOT.2000.844730`. URL: `http://ieeexplore.ieee.org/document/844730/`.

[79] Tejas D Kulkarni et al. “Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation”. In: *Advances in Neural Information Processing Systems* (2016), pp. 1–13.

[80] M. P. Kumar, Benjamin Packer, and Daphne Koller. “Self-Paced Learning for Latent Variable Models”. In: *Advances in Neural Information Processing Systems*. 2010, pp. 1189–1197.

[81] Andras Gabor Kupcsik et al. “Data-efficient generalization of robot skills with contextual policy search”. In: *AAAI Conference on Artificial Intelligence*. 2013, pp. 1401–1407.

[82] S. M. LaValle. *Planning Algorithms*. Available at http://planning.cs.uiuc.edu/. Cambridge, U.K.: Cambridge University Press, 2006.

[83] Alessandro Lazaric and Mohammad Ghavamzadeh. “Bayesian multi-task reinforcement learning”. In: *27th International Conference on Machine Learning*. Omnipress. 2010, pp. 599–606.

[84] Nicolas Le Roux and Yoshua Bengio. “Representational power of restricted Boltzmann machines and deep belief networks”. In: *Neural Computation* 20.6 (2008), pp. 1631–1649.

[85] Hoang M Le et al. “Hierarchical Imitation and Reinforcement Learning”. In: *International Conference in Machine Learning* (2018).

[86] Michelle A Lee et al. “Making Sense of Vision and Touch: Self-Supervised Learning of Multimodal Representations for Contact-Rich Tasks”. In: *International Conference on Robotics and Automation* (2018).

[87] Michelle A Lee* et al. “Guided Uncertainty-Aware Policy Optimization: Combining Learning and Model-Based Strategies for Sample-Efficient Policy Learning”. In: *International Conference on Robotics and Automation* (2020).

[88] V. Lepetit, F. Moreno-Noguer, and P. Fua. “EP*n*P: An Accurate O(*n*) Solution to the P*n*P Problem”. In: *International Journal Computer Vision* 81.2 (2009).

[89] Sergey Levine and Chelsea Finn. “End-to-End Training of Deep Visuomotor Policies”. In: *Journal of Machine Learning Research* 17 (2016), pp. 1–40.

[90] Sergey Levine et al. “End-to-end training of deep visuomotor policies”. In: *Journal of Machine Learning Research* 17.39 (2016), pp. 1–40.

[91] Sergey Levine et al. “Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection”. In: *The International Journal of Robotics Research* 37.4-5 (2018), pp. 421–436.

[92] Andrew Levy, Robert Platt, and Kate Saenko. “Hierarchical Actor-Critic”. In: *arXiv:1712.00948* (Dec. 2017). URL: `http://arxiv.org/abs/1712.00948`.

[93] Andrew Levy, Robert Platt, and Kate Saenko. “Hierarchical Reinforcement Learning with Hindsight”. In: *International Conference on Learning Representations* (May 2019). URL: `http://arxiv.org/abs/1805.08180`.

[94] Alexander C. Li* et al. “Sub-policy Adaptation for Hierarchical Reinforcement Learning”. In: *International Conference on Learning Representations* (2020).

[95] Timothy P. Lillicrap et al. “Continuous control with deep reinforcement learning”. In: *arXiv preprint arXiv:1509.02971* (2015), pp. 1–14. URL: `http://arxiv.org/abs/1509.02971`.

[96] Xingyu Lin et al. “Adaptive Variance for Changing Sparse-Reward Environments”. In: *International Conference on Robotics and Automation* (2019). arXiv: `1903.06309`. URL: `http://arxiv.org/abs/1903.06309`.

[97] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. “The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables”. In: *International Conference on Learning Representations*. 2017.

[98] Shie Mannor et al. “Dynamic abstraction in reinforcement learning via clustering”. In: *21st International Conference on Machine Learning*. ACM. 2004, p. 71.

[99] Simon Manschitz et al. “Learning movement primitive attractor goals and sequential skills from kinesthetic demonstrations”. In: *Robotics and Autonomous Systems* 74 (2015), pp. 97–107. DOI: `10.1016/J.ROBOT.2015.07.005`.

[100] Xudong Mao et al. “On the Effectiveness of Least Squares Generative Adversarial Networks”. In: *arXiv preprint arXiv:1712.06391* (2017).

[101] Josh Merel et al. “Hierarchical visuomotor control of humanoids”. In: *International Conference in Learning Representations* (2019). URL: `http://arxiv.org/abs/1811.09656`.

[102] Volodymyr Mnih et al. “Strategic Attentive Writer for Learning Macro-Actions”. In: *Advances in Neural Information Processing Systems*. 2016.

[103] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (2015), pp. 529–533.

[104] Ofir Nachum et al. “Data-Efficient Hierarchical Reinforcement Learning”. In: *Advances in Neural Information Processing Systems* (2018).

[105] Ashvin Nair et al. “Overcoming Exploration in Reinforcement Learning with Demonstrations”. In: *International Conference on Robotics and Automation* (2018). ISSN: 0969-2290. DOI: 10.1080/09692290.2013.809781. URL: http://arxiv.org/abs/1709.10089.

[106] Ashvin Nair et al. “Visual Reinforcement Learning with Imagined Goals”. In: *Adavances in Neural Information Processing Systems* (2018).

[107] Radford M Neal. “Connectionist learning of belief networks”. In: *Artificial intelligence* 56.1 (1992), pp. 71–113.

[108] Radford M Neal. “Learning stochastic feedforward networks”. In: *Department of Computer Science, University of Toronto* (1990).

[109] Andrew Y Ng, Daishi Harada, and Stuart Russell. “Policy invariance under reward transformations: Theory and application to reward shaping”. In: *International Conference on Machine Learning*. Vol. 99. 1999, pp. 278–287.

[110] Ian Osband, Benjamin Van Roy, and Zheng Wen. “Generalization and Exploration via Randomized Value Functions”. In: *arXiv preprint arXiv:1402.0635* (2014).

[111] Ronald Parr and Stuart Russell. “Reinforcement learning with hierarchies of machines”. In: *Advances in Neural Information Processing Systems* (1998), pp. 1043–1049.

[112] Sida Peng et al. “PVNet: Pixel-wise Voting Network for 6DoF Pose Estimation”. In: *CVPR*. 2019.

[113] Xue Bin Peng et al. “DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills”. In: *Transactions on Graphics (Proc. ACM SIGGRAPH)* 37.4 (2018). DOI: 10.1145/3197517.3201311. URL: http://arxiv.org/abs/1804.02717%0Ahttp://dx.doi.org/10.1145/3197517.3201311.

[114] Xue Bin Peng et al. “MCP: Learning Composable Hierarchical Control with Multiplicative Compositional Policies”. In: (May 2019). URL: http://arxiv.org/abs/1905.09808.

[115] Jan Peters and Stefan Schaal. “Natural Actor-Critic”. In: *Neurocomputing* 71.7-9 (2008), pp. 1180–1190. ISSN: 09252312. DOI: 10.1016/j.neucom.2007.11.026.

[116] Dean A Pomerleau. "ALVINN: an autonomous land vehicle in a neural network". In: *Advances in Neural Information Processing Systems* (1989), pp. 305–313. URL: `https://papers.nips.cc/paper/95-alvinn-an-autonomous-land-vehicle-in-a-neural-network.pdf%20http://dl.acm.org/citation.cfm?id=89851.89891`.

[117] Ivaylo Popov et al. "Data-efficient Deep Reinforcement Learning for Dexterous Manipulation". In: (). URL: `https://arxiv.org/pdf/1704.03073.pdf`.

[118] Doina Precup. *Temporal abstraction in reinforcement learning*. Jan. 2000. URL: `https://scholarworks.umass.edu/dissertations/AAI9978540`.

[119] Aravind Rajeswaran et al. "Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations". In: *Robotics: Science and Systems* (2018).

[120] Aravind Rajeswaran et al. "Towards Generalization and Simplicity in Continuous Control". In: (2017). URL: `https://arxiv.org/pdf/1703.02660.pdf%20http://arxiv.org/abs/1703.02660`.

[121] Pravesh Ranchod, Benjamin Rosman, and George Konidaris. "Nonparametric Bayesian Reward Segmentation for Skill Discovery Using Inverse Reinforcement Learning". In: (2015). ISSN: 21530866. DOI: `10.1109/IROS.2015.7353414`.

[122] Pravesh Ranchod, Benjamin Rosman, and George Konidaris. "Nonparametric Bayesian reward segmentation for skill discovery using inverse reinforcement learning". In: *International Conference on Intelligent Robots and Systems*. IEEE. 2015, pp. 471–477.

[123] Nathan D. Ratliff et al. "Riemannian Motion Policies". In: *arXiv preprint arXiv:1801.02854* (Jan. 2018). URL: `http://arxiv.org/abs/1801.02854`.

[124] Nathan Ratliff, Marc Toussaint, and Stefan Schaal. "Understanding the Geometry of Workspace Obstacles in Motion Optimization". In: *International Conference on Robotics and Automation*. 2015.

[125] Martin Riedmiller et al. "Learning by Playing – Solving Sparse Reward Tasks from Scratch". In: *Internation Conference in Machine Learning* (2018).

[126] Stéphane Ross, Geoffrey J Gordon, and J Andrew Bagnell. "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning". In: *International Conference on Artificial Intelligence and Statistics* (2011).

[127] Fumihiro Sasaki, Tetsuya Yohira, and Atsuo Kawaguchi. "Sample Efficient Imitation Learning for Continuous Control". In: *International Conference in Learning Representationsa* (2019), pp. 1–15.

[128] Stefan Schaal et al. "Learning movement primitives". In: *Robotics Research. The Eleventh International Symposium*. 2005, pp. 561–572.

[129] Tom Schaul et al. "Universal value function approximators". In: *International Conference on Machine Learning*. 2015, pp. 1312–1320.

[130] Jürgen Schmidhuber. "Curious model-building control systems". In: *International Joint Conference on Neural Networks*. IEEE. 1991, pp. 1458–1463.

[131] Jürgen Schmidhuber. "Formal theory of creativity, fun, and intrinsic motivation (1990–2010)". In: *IEEE Transactions on Autonomous Mental Development* 2.3 (2010), pp. 230–247.

[132] Jürgen Schmidhuber. "POWER PLAY : Training an Increasingly General Problem Solver by Continually Searching for the Simplest Still Unsolvable Problem". In: *Frontiers in Psychology* 4.313 (2013). URL: https://arxiv.org/pdf/1112.5309.pdf%20http://journal.frontiersin.org/article/10.3389/fpsyg.2013.00313.

[133] Tanner Schmidt, Richard Newcombe, and Dieter Fox. *DART: Dense Articulated Real-Time Tracking*. Tech. rep. University of Washington, 2015. URL: https://www.cc.gatech.edu/~afb/classes/CS7495-Fall2014/readings/dart.pdf.

[134] Yannick Schroecker, Mel Vecerik, and Jonathan Scholz. "Generative predecessor models for sample-efficient imitation learning". In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=SkeVsiAcYm.

[135] John Schulman et al. "Finding Locally Optimal, Collision-Free Trajectories with Sequential Convex Optimization". In: *RSS*. June 2013. DOI: 10.15607/RSS.2013.IX.031.

[136] John Schulman et al. "High-dimensional continuous control using generalized advantage estimation". In: *International Conference on Learning Representations*. 2016.

[137] John Schulman et al. "Proximal Policy Optimization Algorithms". In: (2017). URL: https://openai-public.s3-us-west-2.amazonaws.com/blog/2017-07/ppo/ppo-arxiv.pdf.

[138] John Schulman et al. "Trust Region Policy Optimization". In: *International Conference in Machine Learning* (2015).

[139] Arjun Sharma et al. "Directed-Info GAIL: Learning Hierarchical Policies from Unsegmented Demonstrations using Directed Information". In: *International Conference in Learning Representations* (2018). URL: http://arxiv.org/abs/1810.01266.

[140] Sahil Sharma and Balaraman Ravindran. "Online Multi-Task Learning Using Biased Sampling". In: (2017).

[141] Tianmin Shu, Caiming Xiong, and Richard Socher. "Hierarchical and interpretable skill acquisition in multi-task reinforcement Learning". In: *International Conference in Learning Representations* 3 (2018), pp. 1–13. DOI: 10.1109/MWC.2016.7553036.

[142] David Silver et al. "Mastering the game of Go without human knowledge". In: *Nature* 550.7676 (Oct. 2017), pp. 354–359. ISSN: 14764687. DOI: 10.1038/nature24270. URL: http://arxiv.org/abs/1610.00633.

[143] Tom Silver et al. "Residual Policy Learning". In: *preprint arXiv:1812.06298* (2018).

[144] Özgür Şimşek, Alicia P Wolfe, and Andrew G Barto. "Identifying useful subgoals in reinforcement learning by local graph partitioning". In: *Proceedings of the 22nd international conference on Machine learning*. ACM. 2005, pp. 816–823.

[145] Matthew J. A. Smith, Herke van Hoof, and Joelle Pineau. *An inference-based policy gradient method for learning options*. Feb. 2018. URL: `https://openreview.net/forum?id=rJIgf7bAZ`.

[146] Paul Smolensky. *Information processing in dynamical systems: Foundations of harmony theory*. Tech. rep. DTIC Document, 1986.

[147] Sungryull Sohn, Junhyuk Oh, and Honglak Lee. "Multitask Reinforcement Learning for Zero-shot Generalization with Subtask Dependencies". In: *Advances in Neural Information Processing Systems* (2018).

[148] Rupesh Kumar Srivastava et al. "Continually adding self-invented problems to the repertoire: First experiments with POWERPLAY". In: *2012 IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL)*. IEEE, Nov. 2012, pp. 1–6. ISBN: 978-1-4673-4965-9. DOI: `10.1109/DevLrn.2012.6400843`. URL: `http://ieeexplore.ieee.org/document/6400843/`.

[149] Bradly C. Stadie, Pieter Abbeel, and Ilya Sutskever. "Third-Person Imitation Learning". In: *International Conference in Learning Representations* (Mar. 2017). URL: `http://arxiv.org/abs/1703.01703`.

[150] Martin Stolle and Doina Precup. "Learning options in reinforcement learning". In: *International Symposium on Abstraction, Reformulation, and Approximation*. 2002, pp. 212–223.

[151] Kaushik Subramanian, Charles L Isbell Jr., and Andrea L Thomaz. "Exploration from Demonstration for Interactive Reinforcement Learning". In: *Proceedings of the International Conference on Autonomous Agents & Multiagent Systems*. 2016.

[152] Sainbayar Sukhbaatar et al. "Intrinsic Motivation and Automatic Curricula via Asymmetric Self-Play". In: 0 ().

[153] Wen Sun, J. Andrew Bagnell, and Byron Boots. "Truncated Horizon Policy Search: Combining Reinforcement Learning & Imitation Learning". In: (2018), pp. 1–14. ISSN: 0004-6361. DOI: `10.1051/0004-6361/201527329`. URL: `http://arxiv.org/abs/1805.11240`.

[154] Martin Sundermeyer et al. "Implicit 3D Orientation Learning for 6D Object Detection from RGB Images". In: *ECCV*. 2018.

[155] Richard S Sutton and Andrew G Barto. "Reinforcement Learning : An Introduction". In: (1998). ISSN: 18726240. DOI: `10.1016/j.brainres.2010.09.091`.

[156] Richard S Sutton, Doina Precup, and Satinder Singh. "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning". In: *Artificial intelligence* 112.1 (1999), pp. 181–211.

[157] Richard S Sutton, Doina Precup, and Satinder Singh. "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning". In: *Artificial Intelligence* 112 (1999), pp. 181–211. URL: `http://www-anw.cs.umass.edu/~barto/courses/cs687/Sutton-Precup-Singh-AIJ99.pdf`.

[158] Aviv Tamar, Sergey Levine, and Pieter Abbeel. "Value Iteration Networks". In: *Advances in Neural Information Processing Systems*.

[159] Yichuan Tang and Ruslan Salakhutdinov. "Learning Stochastic Feedforward Neural Networks". In: *Advances in Neural Information Processing Systems* 2 (2013), pp. 530–538. DOI: `10.1.1.63.1777`.

[160] Yichuan Tang and Ruslan R Salakhutdinov. "Learning Stochastic Feedforward Neural Networks". In: *Advances in Neural Information Processing Systems 26*. 2013, pp. 530–538.

[161] Matthew E Taylor and Peter Stone. "Transfer learning for reinforcement learning domains: A survey". In: *Journal of Machine Learning Research* 10 (2009), pp. 1633–1685.

[162] Russ Tedrake et al. "LQR-trees: Feedback motion planning via sums-of-squares verification". In: *The International Journal of Robotics Research* 29.8 (2010), pp. 1038–1052.

[163] Bugra Tekin, Sudipta N. Sinha, and Pascal Fua. "Real-Time Seamless Single Shot 6D Object Pose Prediction". In: *CVPR*. 2018.

[164] Garrett Thomas et al. "Learning Robotic Assembly from CAD". In: *International Conference on Robotics and Automation* (2018).

[165] Josh Tobin et al. "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World". In: (Mar. 2017). URL: `http://arxiv.org/abs/1703.06907`.

[166] Emanuel Todorov, Tom Erez, and Yuval Tassa. "MuJoCo : A physics engine for model-based control". In: (2012), pp. 5026–5033.

[167] Emanuel Todorov and Weiwei Li. "A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems". In: *American Control Conference, 2005. Proceedings of the 2005*. IEEE. 2005, pp. 300–306.

[168] Jonathan Tremblay et al. "Deep object pose estimation for semantic robotic grasping of household objects". In: *arXiv preprint arXiv:1809.10790* (2018).

[169] George Tucker et al. "The Mirage of Action-Dependent Baselines in Reinforcement Learning". In: *Internation Conference in Machine Learning* (2018). URL: `http://arxiv.org/abs/1802.10031`.

[170] Karl Van Wyk et al. "Comparative Peg-in-Hole Testing of a Force-Based Manipulation Controlled Robotic Hand". In: *IEEE Transactions on Robotics* 34.2 (2018), pp. 542–549. ISSN: 15523098. DOI: `10.1109/TRO.2018.2791591`.

[171] Mel Vecerik et al. "Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards". In: (2017), pp. 1–11.

[172] Alexander Sasha Vezhnevets et al. "Feudal Networks for Hierarchical Reinforcement Learning". In: *International Conference in Machine Learning* (2017). URL: `https://arxiv.org/pdf/1703.01161.pdf`.

[173] Alexander Vezhnevets et al. "Strategic Attentive Writer for Learning Macro-Actions". In: *Advances in Neural Information Processing Systems* (2016).

[174] Christopher M Vigorito and Andrew G Barto. "Intrinsically motivated hierarchical skill learning in structured environments". In: *IEEE Transactions on Autonomous Mental Development* 2.2 (2010), pp. 132–143.

[175] Oriol Vinyals et al. *AlphaStar: Mastering the Real-Time Strategy Game StarCraft II*. Tech. rep. 2019.

[176] Théophane Weber et al. "Credit Assignment Techniques in Stochastic Computation Graphs". In: (Jan. 2019). URL: `http://arxiv.org/abs/1901.01761`.

[177] Ronald J Williams. "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning". In: *Machine Learning* 8.3-4 (1992), pp. 229–256.

[178] Aaron Wilson et al. "Multi-task reinforcement learning: a hierarchical Bayesian approach". In: *Proceedings of the 24th international conference on Machine learning*. ACM. 2007, pp. 1015–1022.

[179] Yuhuai Wu et al. "On multiplicative integration with recurrent neural networks". In: *Advances in Neural Information Processing Systems*. 2016.

[180] M. Wuthrich et al. "Probabilistic Object Tracking Using a Range Camera". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, Nov. 2013, pp. 3195–3202.

[181] Y. Xiang et al. "PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes". In: *RSS*. 2018.

[182] Sergey Zakharov, Ivan Shugurov, and Slobodan Ilic. "DPOD: Dense 6D pose object detector in RGB images". In: *arXiv preprint arXiv:1902.11020* (2019).

[183] Wojciech Zaremba and Ilya Sutskever. "Learning to execute". In: *arXiv preprint arXiv:1410.4615* (2014).

[184] Stephan Zheng, Yisong Yue, and Jennifer Hobbs. "Generating Long-term Trajectories Using Deep Hierarchical Networks". In: *Advances in Neural Information Processing Systems 29*. Ed. by D D Lee et al. 2016.

[185] Henry Zhu et al. "Dexterous Manipulation with Deep Reinforcement Learning: Efficient, General, and Low-Cost". In: (Oct. 2018). URL: `http://arxiv.org/abs/1810.06045`.

[186] Henry Zhu et al. "Dexterous manipulation with deep reinforcement learning: Efficient, general, and low-cost". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 3651–3657.

[187] Yuke Zhu et al. “Reinforcement and Imitation Learning for Diverse Visuomotor Skills”. In: *Robotics: Science and Systems* (2018). URL: `http://arxiv.org/abs/1802.09564`.

[188] Brian D Ziebart et al. “Maximum Entropy Inverse Reinforcement Learning”. In: (2008), pp. 1433–1438.

Appendix A

Appendix

A.1 Stochastic Neural Networks for Hierarchical Reinforcement Learning Appendix

Hyperparameters

All policies are trained with TRPO with step size 0.01 and discount 0.99. All neural networks (each of the Multi-policy ones, the SNN and the Manager Network) have 2 layers of 32 hidden units. For the SNN training, the mesh density used to grid the (x, y) space and give the MI bonus is 10 divisions/unit. The number of skills trained (ie dimension of latent variable in the SNN or number of independently trained policies in the Mulit-policy setup) is 6. The batch size and the maximum path length for the pre-train task are also the ones used in the benchmark [30]: 50,000 and 500 respectively. For the downstream tasks, see Tab. A.1.

Parameter	Mazes	Food Gather
Batch size	1M	100k
Maximum path length	10k	5k
Switch time $\mathcal{T}$	500	10

Table A.1: Parameters of algorithms for downstream tasks, measured in number of time-steps of the low level control

Results for Gather with Benchmark settings

To fairly compare our methods to previous work on the downstream Gather environment, we report here our results in the exact settings used in Duan et al. [30]: maximum path length of 500 and batch-size of 50k. Our SNN hierarchical approach outperforms state-of-the-art intrinsic motivation results like VIME [62].

The baseline of having a Center of Mass speed intrinsic reward in the task happens to be stronger than expected. This is due to two factors. First, the task offers a not-so-sparse reward (green and red balls may lie quite close to the robot), which can be easily reached by an agent intrinsically motivated to move. Second, the limit of 500 steps makes the original task much simpler in the sense that the agent only needs to learn how to reach the nearest green ball, as it won't have time to reach anything else. Therefore there is no actual need of skills to re-orient or properly navigate the environment, making the hierarchy useless. This is why when increasing the time-horizon to 5,000 steps, the hierarchy shines much more, as can also be seen in the videos.

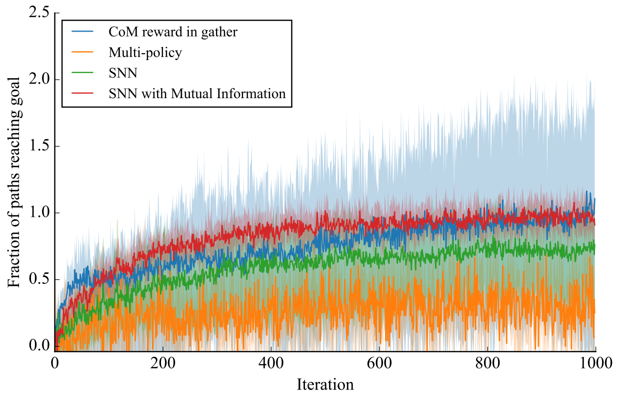


Figure A.1: Results of SNN4HRL for the Gather environment in the benchmark settings

Additional Experiments with "Snake"

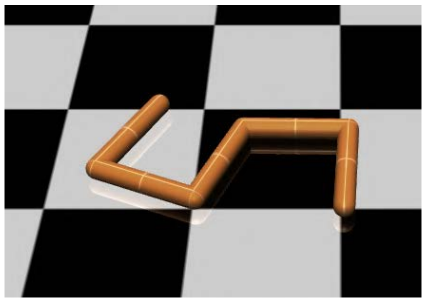
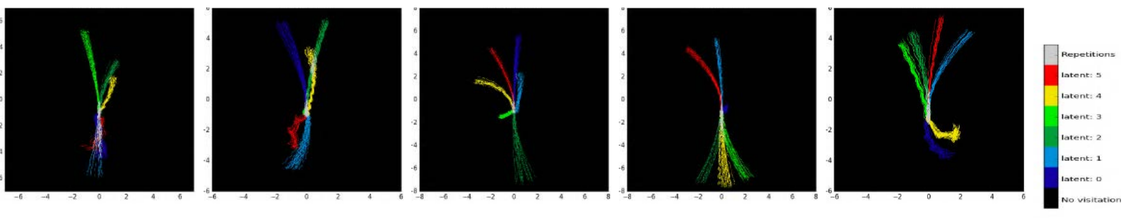


Figure A.2: Snake Environment

In this section we study how our method scales to a more complex robot. We repeat most experiments from Section 7 with the "Snake" agent, a 5-link robot depicted in Fig. A.2. Compared to Swimmer, the action dimension doubles and the state-space increases by 50%. We also perform a further analysis on the relevance of the switch time $\mathcal{T}$ and end up by reporting the performance variation among different pretrained SNNs on the hierarchical tasks.

Skill learning in pretrain

The Snake robot can learn a large span of skills as consistently as Swimmer. We report in Figs. A.3a-A.3e the visitation plots obtained for five different random seeds in our SNN pretraining algorithm from Secs. 2.4-2.4, with Mutual Information bonus of 0.05. The impact of the different spans of skills on the later hierarchical training is discussed in Sec. A.1.

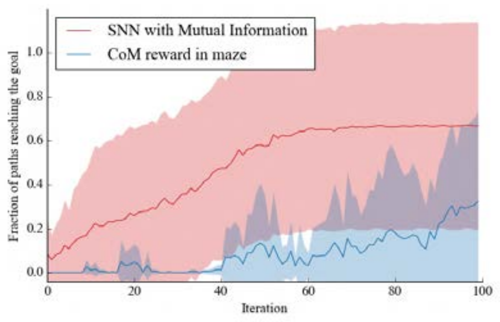


(a) seed 10 (b) seed 20 (c) seed 30 (d) seed 40 (e) seed 50

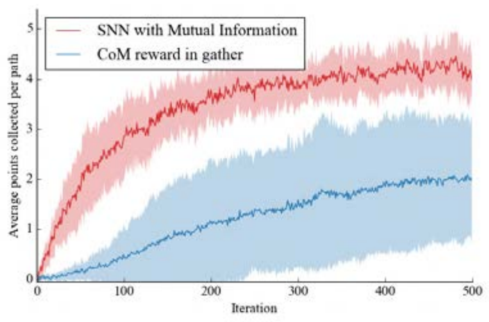
Figure A.3: Span of skills learned using different random seeds to pretraining a SNN with MI $= 0.05$

Hierarchical use of skills in Maze and Gather

Here we evaluate the performance of our hierarchical architecture to solve with Snake the sparse tasks Maze 0 and Gather from Sec. 2.5. Given the larger size of the robot, we have also increased the size of the Maze and the Gather task, from 2 to 7 and from 6 to 10 respectively. The maximum path length is also increased in the same proportion, but not the batch size nor the switch time $\mathcal{T}$ (see Sec. A.1 for an analysis on $\mathcal{T}$). Despite the tasks being harder, our approach still achieves good results, and Figs. A.4a-A.4b, clearly shows how it outperforms the baseline of having the intrinsic motivation of the Center of Mass in the hierarchical task.



(a) Maze 0 with Snake



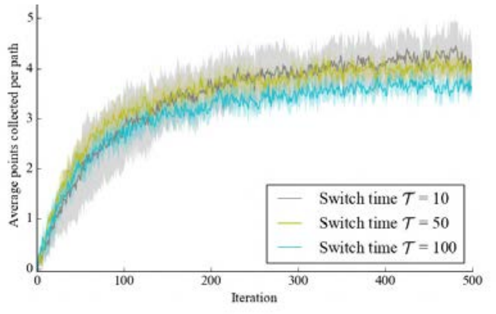
(b) Gather task with Snake

Figure A.4: Faster learning of SNN4HRL in sparse downstream MDPs with Snake

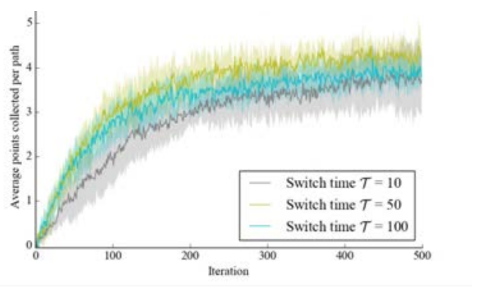
Analysis of the switch time $\mathcal{T}$

The switch time $\mathcal{T}$ does not critically affect the performance for these static tasks, where the robot is not required to react fast and precisely to changes in the environment. The performance of very different $\mathcal{T}$ is reported for the Gather task of sizes 10 and 15 in Figs. A.5a-A.5b. A smaller environment implies having red/green balls closer to each other, and hence more precise navigation is expected to achieve higher scores. In our framework, lower switch time $\mathcal{T}$ allows faster changes between skills, therefore explaining the slightly better performance of $\mathcal{T} = 10$ or 50 over 100 for the Gather task of size 10 (Fig. A.5a). On the other hand, larger environments mean more sparcity as the same number of balls is uniformly initialized in a wider space. In such case, longer commitment to every skill is expected to improve the exploration range. This explains the slower learning of $\mathcal{T} = 10$ in the Gather task of size 15 (Fig. A.5b). It is still surprising the mild changes produced by such large variations in the switch time $\mathcal{T}$ for the Gather task.

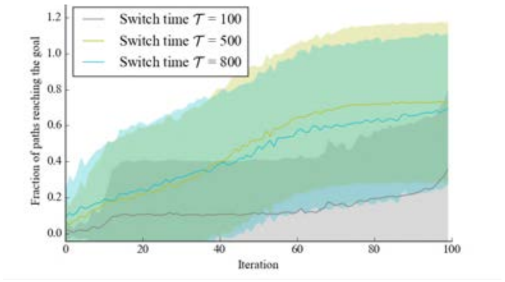
For the Maze 0 task, Figs. A.5c-A.5d show that the difference between different $\mathcal{T}$ is important but not critical. In particular, radical increases of $\mathcal{T}$ have little effect on the performance in this task because the robot will simply keep bumping against a wall until it switches latent code. This behavior is observed in the videos attached with the paper[2]. The large variances observed are due to the performance of the different pretrained SNN. This is further studied in Sec. A.1.



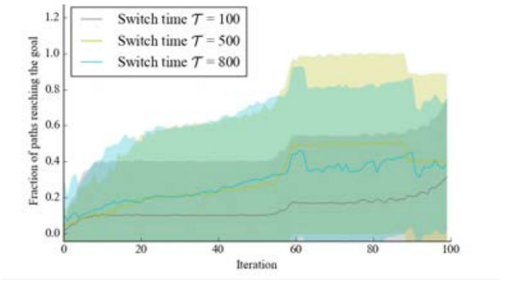
(a) Gather task with size 10



(b) Gather task with size 15



(c) Maze 0 task with size 7

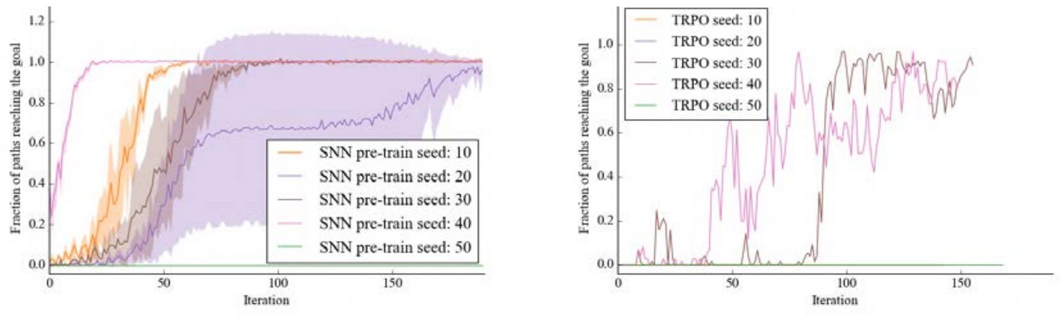


(d) Maze 0 task with size 9

Figure A.5: Mild effect of switch time $\mathcal{T}$ on different sizes of Gather and Maze 0

Impact of the pre-trained SNN on the hierarchy

In the previous section, the low variance of the learning curves for Gather indicates that all pretrained SNN preform equally good in this task. For Maze, the performance depends strongly on the pretrained SNN. For example we observe that the one obtained with the random seed 50 has a visitation with a weak backward span (Fig. A.3e), which happens to be critical to solve the Maze 0 (as can be seen in the videos[2]). This explains the lack of learning in Fig. A.6a for this particular pretrained SNN. Note that the large variability between different random seeds is even worse in the baseline of having the CoM reward in the Maze, as seen in Fig. A.6b. 3 out of 5 seeds never reach the goal and the ones that does have an unstable learning due to the long time-horizon.



(a) Snake on Maze 0 of size 7 and switch-time $\mathcal{T} = 500$

(b) Snake on Maze 0 of size 7 trained with TRPO and CoM reward

Figure A.6: Variation in performance among different pretrained SNNs and different TRPO seeds

A.2 Hierarchical Proximal Policy Optimization Appendix

Hyperparameters and Architectures Details

The Block environments used a horizon of 1000 and a batch size of 50,000, while Gather used a batch size of 100,000. Ant Gather has a horizon of 5000, while Snake Gather has a horizon of 8000 due to its larger size. For all experiments, both PPO and HiPPO used learning rate 3×10^{-3}, clipping parameter $\epsilon = 0.1$, 10 gradient updates per iteration, and discount $\gamma = 0.999$. The learning rate, clipping parameter, and number of gradient updates come from the OpenAI Baselines implementation.

HiPPO used $n = 6$ sub-policies. HiPPO uses a manager network with 2 hidden layers of 32 units, and a skill network with 2 hidden layers of 64 units. In order to have roughly the same number of parameters for each algorithm, flat PPO uses a network with 2 hidden layers with 256 and 64 units respectively. For HiPPO with randomized period, we resample $p \sim \text{Uniform}\{5, 15\}$ every time the manager network outputs a latent, and provide the number of timesteps until the next latent selection as an input into both the manager and skill networks. The single baselines and skill-dependent baselines used a MLP with 2 hidden layers of 32 units to fit the value function. The skill-dependent baseline receives, in addition to the full observation, the active latent code and the time remaining until the next skill sampling. All runs used five random seeds.

Robot Agent Description

Hopper is a 3-link robot with a 14-dimensional observation space and a 3-dimensional action space. Half-Cheetah has a 20-dimensional observation space and a 6-dimensional action space. We evaluate both of these agents on a sparse block hopping task. In addition to observing their own joint angles and positions, they observe the height and length of the next wall, the x-position of the next wall, and the distance to the wall from the agent. We also provide the same wall observations for the previous wall, which the agent can still interact with.

Snake is a 5-link robot with a 17-dimensional observation space and a 4-dimensional action space. Ant is a quadrupedal robot with a 27-dimensional observation space and a 8-dimensional action space. Both Ant and Snake can move and rotate in all directions, and Ant faces the added challenge of avoiding falling over irrecoverably. In the Gather environment, agents also receive 2 sets of 10-dimensional lidar observations, whcih correspond to separate apple and bomb observations. The observation displays the distance to the nearest apple or bomb in each $36°$ bin, respectively. All environments are simulated with the physics engine MuJoCo [166].

Proofs

Lemma 1. If the skills are sufficiently differentiated, then the latent variable can be treated as part of the observation to compute the gradient of the trajectory probability. Concretely, if $\pi_{\theta_h}(z|s)$ and $\pi_{\theta_l}(a|s,z)$ are Lipschitz in their parameters, and $0 < \pi_{\theta_l}(a_t|s_t, z_j) < \epsilon \ \forall j \neq kp$, then

$$\nabla_\theta \log P(\tau) = \sum_{k=0}^{H/p} \nabla_\theta \log \pi_{\theta_h}(z_{kp}|s_{kp}) + \sum_{t=1}^{p} \nabla_\theta \log \pi_{\theta_l}(a_t|s_t, z_{kp}) + \mathcal{O}(nH\epsilon^{p-1}) \tag{A.1}$$

Proof. From the point of view of the MDP, a trajectory is a sequence $\tau = (s_0, a_0, s_1, a_1, \dots, a_{H-1}, s_H)$. Let's assume we use the hierarchical policy introduced above, with a higher-level policy modeled as a parameterized discrete distribution with n possible outcomes $\pi_{\theta_h}(z|s) = Categorical_{\theta_h}(n)$. We can expand $P(\tau)$ into the product of policy and environment dynamics terms, with z_j denoting the jth possible value out of the n choices,

$$P(\tau) = \Bigg(\prod_{k=0}^{H/p} \Big[\sum_{j=1}^{n} \pi_{\theta_h}(z_j|s_{kp}) \prod_{t=kp}^{(k+1)p-1} \pi_{\theta_l}(a_t|s_t, z_j)\Big]\Bigg) \Big[P(s_0) \prod_{t=1}^{H} P(s_{t+1}|s_t, a_t)\Big]$$

Taking the gradient of $\log P(\tau)$ with respect to the policy parameters $\theta = [\theta_h, \theta_l]$, the dynamics terms disappear, leaving:

$$\begin{aligned}\nabla_\theta \log P(\tau) &= \sum_{k=0}^{H/p} \nabla_\theta \log \Big(\sum_{j=1}^{n} \pi_{\theta_l}(z_j|s_{kp}) \prod_{t=kp}^{(k+1)p-1} \pi_{s,\theta}(a_t|s_t, z_j)\Big) \\ &= \sum_{k=0}^{H/p} \frac{\sum_{j=1}^{n} \nabla_\theta \Big(\pi_{\theta_h}(z_j|s_{kp}) \prod_{t=kp}^{(k+1)p-1} \pi_{\theta_l}(a_t|s_t, z_j)\Big)}{\sum_{j=1}^{n} \pi_{\theta_h}(z_j|s_{kp}) \prod_{t=kp}^{(k+1)p-1} \pi_{\theta_l}(a_t|s_t, z_j)}\end{aligned}$$

The sum over possible values of z prevents the logarithm from splitting the product over the p-step sub-trajectories. This term is problematic, as this product quickly approaches 0 as p increases, and suffers from considerable numerical instabilities. Instead, we want to approximate this sum of products by a single one of the terms, which can then be decomposed into a sum of logs. For this we study each of the terms in the sum: the gradient of a sub-trajectory probability under a specific latent $\nabla_\theta\Big(\pi_{\theta_h}(z_j|s_{kp})\prod_{t=kp}^{(k+1)p-1}\pi_{\theta_l}(a_t|s_t,z_j)\Big)$. Now we can use the assumption that the skills are easy to distinguish, $0<\pi_{\theta_l}(a_t|s_t,z_j)<\epsilon\;\forall j\neq kp$. Therefore, the probability of the sub-trajectory under a latent different than the one that was originally sampled $z_j\neq z_{kp}$, is upper bounded by ϵ^p. Taking the gradient, applying the product rule, and the Lipschitz continuity of the policies, we obtain that for all $z_j\neq z_{kp}$,

$$\begin{aligned}
\nabla_\theta\Big(\pi_{\theta_h}(z_j|s_{kp})\prod_{t=kp}^{(k+1)p-1}\pi_{\theta_l}(a_t|s_t,z_j)\Big) &= \nabla_\theta\pi_{\theta_h}(z_j|s_{kp})\prod_{t=kp}^{(k+1)p-1}\pi_{\theta_l}(a_t|s_t,z_j)+\\
&\qquad\sum_{t=kp}^{(k+1)p-1}\pi_{\theta_h}(z_j|s_{kp})\big(\nabla_\theta\pi_{\theta_l}(a_t|s_t,z_j)\big)\prod_{\substack{t=kp\\t'\neq t}}^{(k+1)p-1}\pi_{\theta_l}(a_{t'}|s_{t'},z_j)\\
&= \mathcal{O}(p\epsilon^{p-1})
\end{aligned}$$

Thus, we can across the board replace the summation over latents by the single term corresponding to the latent that was sampled at that time.

$$\begin{aligned}
\nabla_\theta\log P(\tau) &= \sum_{k=0}^{H/p}\frac{1}{\pi_{\theta_h}(z_{kp}|s_{kp})\prod_{t=kp}^{(k+1)p-1}\pi_{\theta_l}(a_t|s_t,z_{kp})}\nabla_\theta\Big(P(z_{kp}|s_{kp})\prod_{t=kp}^{(k+1)p-1}\pi_{\theta_l}(a_t|s_t,z_{kp})\Big)\\
&\qquad+\frac{nH}{p}\mathcal{O}(p\epsilon^{p-1})\\
&= \sum_{k=0}^{H/p}\nabla_\theta\log\Big(\pi_{\theta_h}(z_{kp}|s_{kp})\prod_{t=kp}^{(k+1)p-1}\pi_{\theta_l}(a_t|s_t,z_{kp})\Big)+\mathcal{O}(nH\epsilon^{p-1})\\
&= \mathbb{E}_\tau\Big[\Big(\sum_{k=0}^{H/p}\nabla_\theta\log\pi_{\theta_h}(z_{kp}|s_{kp})+\sum_{t=1}^{H}\nabla_\theta\log\pi_{\theta_l}(a_t|s_t,z_{kp})\Big)\Big]+\mathcal{O}(nH\epsilon^{p-1})
\end{aligned}$$

Interestingly, this is exactly $\nabla_\theta P(s_0,z_0,a_0,s_1,\dots)$. In other words, it's the gradient of the probability of that trajectory, where the trajectory now includes the variables z as if they were observed.

□

Lemma 2. For any functions $b_h:\mathcal{S}\to\mathbb{R}$ and $b_l:\mathcal{S}\times\mathcal{Z}\to\mathbb{R}$ we have:

$$\mathbb{E}_\tau[\sum_{k=0}^{H/p}\nabla_\theta\log P(z_{kp}|s_{kp})b(s_{kp})]=0$$

$$\mathbb{E}_\tau[\sum_{t=0}^{H} \nabla_\theta \log \pi_{\theta_l}(a_t|s_t, z_{kp})b(s_t, z_{kp})] = 0$$

Proof. We can use the tower property as well as the fact that the interior expression only depends on s_{kp} and z_{kp}:

$$\begin{aligned}\mathbb{E}_\tau[\sum_{k=0}^{H/p} \nabla_\theta \log P(z_{kp}|s_{kp})b(s_{kp})] &= \sum_{k=0}^{H/p} \mathbb{E}_{s_{kp},z_{kp}}[\mathbb{E}_{\tau \setminus s_{kp},z_{kp}}[\nabla_\theta \log P(z_{kp}|s_{kp})b(s_{kp})]] \\ &= \sum_{k=0}^{H/p} \mathbb{E}_{s_{kp},z_{kp}}[\nabla_\theta \log P(z_{kp}|s_{kp})b(s_{kp})]\end{aligned}$$

Then, we can write out the definition of the expectation and undo the gradient-log trick to prove that the baseline is unbiased.

$$\begin{aligned}\mathbb{E}_\tau \sum_{k=0}^{H/p} \nabla_\theta \log \pi_{\theta_h}(z_{kp}|s_{kp})b(s_{kp}) &= \sum_{k=0}^{H/p} \int_{(s_{kp},z_{kp})} P(s_{kp}, z_{kp}) \nabla_\theta \log \pi_{\theta_h}(z_{kp}|s_{kp})b(s_{kp})dz_{kp}ds_{kp} \\ &= \sum_{k=0}^{H/p} \int_{s_{kp}} P(s_{kp})b(s_{kp}) \int_{z_{kp}} \pi_{\theta_h}(z_{kp}|s_{kp})\nabla_\theta \log \pi_{\theta_h}(z_{kp}|s_{kp})dz_{kp}ds_{kp} \\ &= \sum_{k=0}^{H/p} \int_{s_{kp}} P(s_{kp})b(s_{kp}) \int_{z_{kp}} \pi_{\theta_h}(z_{kp}|s_{kp}) \frac{\nabla_\theta \pi_{\theta_h}(z_{kp}|s_{kp})}{\pi_{\theta_h}(z_{kp}|s_{kp})} dz_{kp}ds_{kp} \\ &= \sum_{k=0}^{H/p} \int_{s_{kp}} P(s_{kp})b(s_{kp})\nabla_\theta \int_{z_{kp}} \pi_{\theta_h}(z_{kp}|s_{kp})dz_{kp}ds_{kp} \\ &= \sum_{k=0}^{H/p} \int_{s_{kp}} P(s_{kp})b(s_{kp})\nabla_\theta 1 ds_{kp} \\ &= 0\end{aligned}$$

□

Subtracting a state- and subpolicy- dependent baseline from the second term is also unbiased, i.e.

$$\mathbb{E}_\tau[\sum_{t=0}^{H} \nabla_\theta \log \pi_{s,\theta}(a_t|s_t, z_{kp})b(s_t, z_{kp})] = 0$$

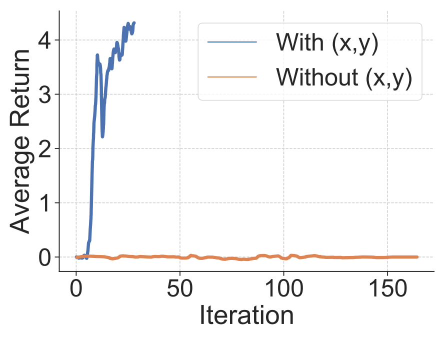


Figure A.7: HIRO performance on Ant Gather with and without access to the ground truth (x, y), which it needs to communicate useful goals.

We'll follow the same strategy to prove the second equality: apply the tower property, express the expectation as an integral, and undo the gradient-log trick.

$$\mathbb{E}_\tau[\sum_{t=0}^{H} \nabla_\theta \log \pi_{\theta_l}(a_t|s_t, z_{kp}) b(s_t, z_{kp})]$$

$$= \sum_{t=0}^{H} \mathbb{E}_{s_t,a_t,z_{kp}}[\mathbb{E}_{\tau \setminus s_t,a_t,z_{kp}}[\nabla_\theta \log \pi_{\theta_m}(a_t|s_t, z_{kp}) b(s_t, z_{kp})]]$$

$$= \sum_{t=0}^{H} \mathbb{E}_{s_t,a_t,z_{kp}}[\nabla_\theta \log \pi_{\theta_l}(a_t|s_t, z_{kp}) b(s_{kp}, z_{kp})]$$

$$= \sum_{t=0}^{H} \int_{(s_t,z_{kp})} P(s_t, z_{kp}) b(s_t, z_{kp}) \int_{a_t} \pi_{\theta_l}(a_t|s_t, z_{kp}) \nabla_\theta \log \pi_{\theta_l}(a_t|s_t, z_{kp}) da_t dz_{kp} ds_t$$

$$= \sum_{t=0}^{H} \int_{(s_t,z_{kp})} P(s_t, z_{kp}) b(s_t, z_{kp}) \nabla_\theta 1 dz_{kp} ds_t$$

$$= 0$$

HIRO sensitivity to observation-space

In this section we provide a more detailed explanation of why HIRO [104] performs poorly under our environments. As explained in our related work section, HIRO belongs to the general category of algorithms that train goal-reaching policies as lower levels of the hierarchy [172, 92]. These methods rely on having a goal-space that is meaningful for the task at hand. For example, in navigation tasks they require having access to the (x, y) position of the agent such that deltas in that space can be given as meaningful goals to move in the environment. Unfortunately, in many cases the only readily available information (if there's no GPS signal or other positioning system installed) are raw sensory inputs, like cameras or the LIDAR sensors we mimic in our environments.

In such cases, our method still performs well because it doesn't rely on the goal-reaching extra supervision that is leveraged (and detrimental in this case) in HIRO and similar methods. In Figure A.7, we show that knowing the ground truth location is critical for its success. We have reproduced the HIRO results in Fig. A.7 using the published codebase, so we are convinced that our results showcase a failure mode of HIRO.

Hyperparameter Sensitivity Plots

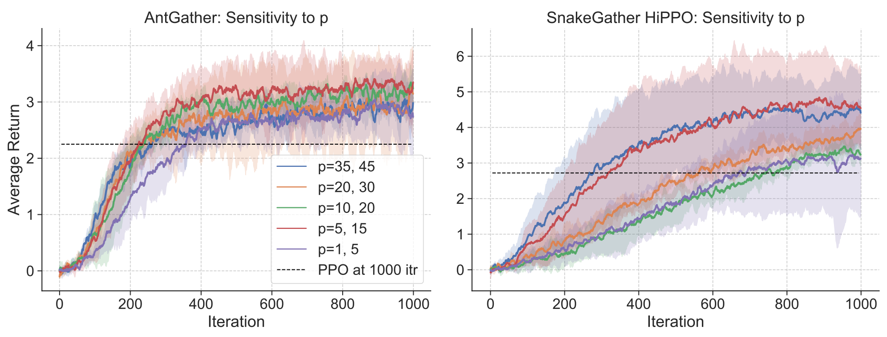


Figure A.8: Sensitivity of HiPPO to variation in the time-commitment.

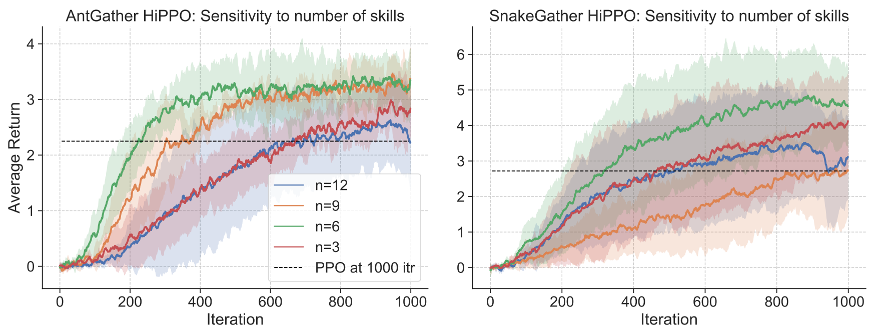


Figure A.9: Sensitivity of HiPPO to variation in the number of skills.

A.3 Automatic Curriculum Generation for RL Agents Appendix

Implementation details

Replay buffer

In addition to training our policy on the goals that were generated in the current iteration, we also save a list (“regularized replay buffer") of goals that were generated during previous iterations (`update_replay`). These goals are also used to train our policy, so that our policy does not forget how to achieve goals that it has previously learned. When we generate goals for our policy to train on, we sample two thirds of the goals from the Goal GAN and we sample the one third of the goals uniformly from the replay buffer. To prevent the replay buffer from concentrating in a small portion of goal space, we only insert new goals that are further away than ϵ from the goals already in the buffer, where we chose the goal-space metric and ϵ to be the same as the ones introduced in Section 3.3.

Goal GAN Initialization

In order to begin our training procedure, we need to initialize our goal generator to produce an initial set of goals (`initialize_GAN`). If we initialize the goal generator randomly (or if we initialize it to sample uniformly from the goal space), it is likely that, for most (or all) of the sampled goals, our initial policy would receives no reward due to the sparsity of the reward function. Thus we might have that all of our initial goals g have $\bar{R}^g(\pi_0) < R_{\min}$, leading to very slow training.

To avoid this problem, we initialize our goal generator to output a set of goals that our initial policy is likely to be able to achieve with $\bar{R}^g(\pi_i) \geq R_{\min}$. To accomplish this, we run our initial policy $\pi_0(a_t \mid s_t, g)$ with goals sampled uniformly from the goal space. We then observe the set of states S^v that are visited by our initial policy. These are states that can be easily achieved with the initial policy, π_0, so the goals corresponding to such states will likely be contained within S_0^I. We then train the goal generator to produce goals that match the state-visitation distribution $p_v(g)$, defined as the uniform distribution over the set $f(S^v)$. We can achieve this through traditional GAN training, with $p_{\text{d}ata}(g) = p_v(g)$. This initialization of the generator allows us to bootstrap the Goal GAN training process, and our policy is able to quickly improve its performance.

Experimental details

Ant specifications

The ant is a quadruped with 8 actuated joints, 2 for each leg. The environment is implemented in Mujoco [166]. Besides the coordinates of the center of mass, the joint angles and joint velocities are also included in the observation of the agent. The high degrees of freedom make navigation a quite complex task requiring motor coordination. More details can be found in Duan et al. [30], and the only difference is that in our goal-oriented version of the Ant we append the observation with the goal, the vector from the CoM to the goal and the distance to the goal. For the Free Ant experiments the objective is to reach any point in the square $[-5m, 5m]^2$ on command. The maximum time-steps given to reach the current goal are 500.

Ant Maze Environment

The agent is constrained to move within the maze environment, which has dimensions of 6m x 6m. The full state-space has an area of size 10 m x 10 m, within which the maze is centered. To compute the coverage objective, goals are sampled from within the maze according to a uniform grid on the maze interior. The maximum time-steps given to reach the current goal are 500.

Point-mass specifications

For the N-dim point mass of Section 3.5, in each episode (rollout) the point-mass has 400 timesteps to reach the goal, where each timestep is 0.02 seconds. The agent can accelerate in up to a rate of 5 m/s^2 in each dimension ($N = 2$ for the maze). The observations of the agent are $2N$ dimensional, including position and velocity of the point-mass.

Goal GAN design and training

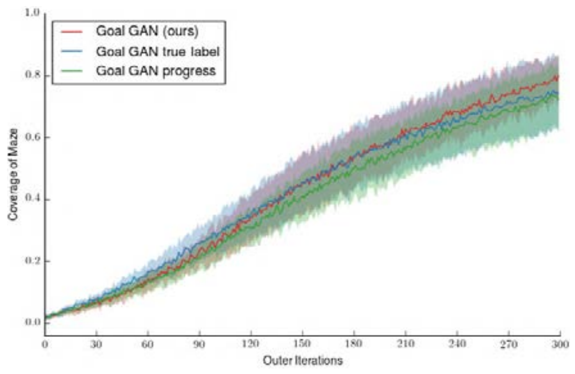
After the generator generates goals, we add noise to each dimension of the goal sampled from a normal distribution with zero mean and unit variance. At each step of the algorithm, we train the policy for 5 iterations, each of which consists of 100 episodes. After 5 policy iterations, we then train the GAN for 200 iterations, each of which consists of 1 iteration of training the discriminator and 1 iteration of training the generator. The generator receives as input 4 dimensional noise sampled from the standard normal distribution. The goal generator consists of two hidden layers with 128 nodes, and the goal discriminator consists of two hidden layers with 256 nodes, with relu nonlinearities.

Policy and optimization

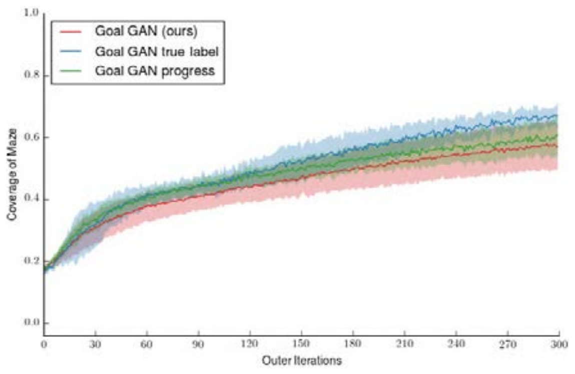
The policy is defined by a neural network which receives as input the goal appended to the agent observations described above. The inputs are sent to two hidden layers of size 32 with tanh nonlinearities. The final hidden layer is followed by a linear N-dimensional output, corresponding to accelerations in the N dimensions. For policy optimization, we use a discount factor of 0.998 and a GAE lambda of 0.995. The policy is trained with TRPO with Generalized Advantage Estimation implemented in rllab [138, 30, 136]. Every "update_policy" consists of 5 iterations of this algorithm.

Study of GoalGAN goals

To label a given goal (Section 3.4), we could empirically estimate the expected return for this goal $\bar{R}^g(\pi_i)$ by performing rollouts of our current policy π_i. The label for this goal is then set to $y_g = \mathbb{1}\Big\{R_{\min} \leq \bar{R}^g(\pi_i) \leq R_{\max}\Big\}$. Nevertheless, having to execute additional rollouts just for labeling is not sample efficient. Therefore, we instead use the rollouts that were used for the most recent policy update. This is an approximation as the rollouts where performed under π_{i-1}, but as we show in Figs. A.10a-A.10b, this small “delay" does not affect learning significantly. Indeed, using the true label (estimated with three new rollouts from π_i) yields the *Goal GAN true label* curves that are only slightly better than what our method does. Furthermore, no matter what labeling technique is used, the success rate of most goals is computed as an average of at most four attempts. Therefore, the statement $R_{\min} \leq \bar{R}^g(\pi_i)$ will be unchanged for any value of $R_{\min} \in (0, 0.25)$. Same for $\bar{R}^g(\pi_i) \leq R_{\max}$ and $R_{\max} \in (0.75, 1)$. This implies that the labels estimates (and hence our automatic curriculum generation algorithm) is almost invariant for any value of the hypermparameters $R_{\min}$ and $R_{\max}$ in these ranges.



(a) Free Ant - Variants



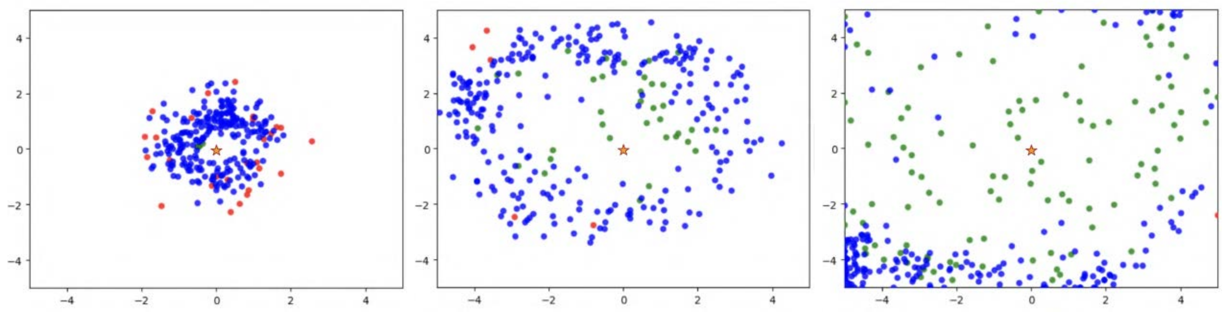
(b) Maze Ant - Variants

Figure A.10: Learning curves comparing the training efficiency of our method and different variants. All plots are an average over 10 random seeds.

In the same plots we also study another criteria to choose the goals to train on that has been previously used in the literature: learning progress [9, 48]. Given that we work in a continuous goal-space, estimating the learning progress of a single goal requires estimating the performance of the policy on that goal before the policy update and after the policy update (potentially being able to replace one of these estimations with the rollouts from the policy optimization, but not both). Therefore the method does require more samples, but we deemed interesting to compare how well the metrics allow to automatically build a curriculum. We see in the Figs. A.10a-A.10b that the two metrics yield a very similar learning, at least in the case of Ant navigation tasks with sparse rewards.

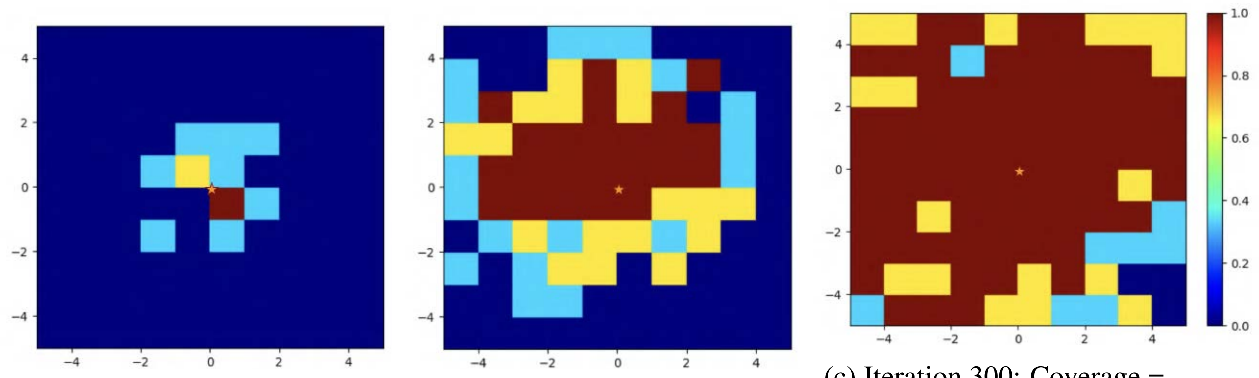
Goal Generation for Free Ant

Similar to the experiments in Figures 3.3 and 3.4, here we show the goals that were generated for the Free Ant experiment in which a robotic quadruped must learn to move to all points in free space. Figures A.11 and A.12 show the results. As shown, our method produces a growing circle around the origin; as the policy learns to move the ant to nearby points, the generator learns to generate goals at increasingly distant positions.



(a) Iteration 10 (b) Iteration 100 (c) Iterartion 300

Figure A.11: Goals that our algorithm trains on (200 sampled from the Goal GAN, 100 from the replay). “High rewards" (green) are goals with $\bar{R}^g(\pi_i) \geq R_{\max}$; $GOID_i$ (blue) have appropriate difficulty for the current policy $R_{\min} \leq \bar{R}^g(\pi_i) \leq R_{\max}$. The red ones have $R_{\min} \geq \bar{R}^g(\pi_i)$

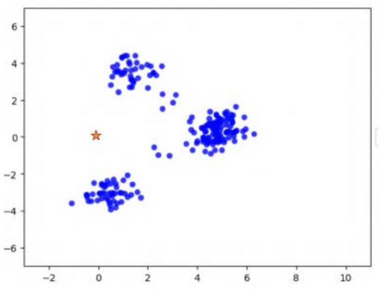


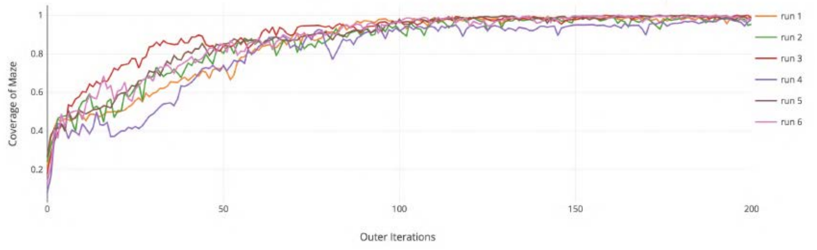
(a) Iteration 10: Coverage = 0.037 (b) Iteration 100: Coverage = 0.4 (c) Iteration 300: Coverage = 0.86

Figure A.12: Visualization of the policy performance for different parts of the state space (same policy training as in Fig. A.11). For illustration purposes, the feasible state-space is divided into a grid, and a goal location is selected from the center of each grid cell. Each grid cell is colored according to the expected return achieved on this goal: Red indicates 100% success; blue indicates 0% success.

Learning for Multi-path point-mass

To clearly observe that our GoalGAN approach is capable of fitting multimodal distributions, we have plotted in Fig. A.13a only the samples coming from the GoalGAN (i.e. no samples from the replay buffer). Also, in this environment there are several ways of reaching every part of the maze. This is not a problem for our algorithm, as can be seen in the full learning curves in Fig.A.13b, where we see that all runs of the algorithm reliably reaches full coverage of the multi-path maze.





(a) Iteration 10 Goal GAN samples (Fig. 3.5b without replay samples)

(b) Learning curves of our algorithm on Multi-path Point-mass Maze, consistently achieving full coverage

Figure A.13: Study of goalGAN in the Multi-Path Maze environment

Comparisons with other methods

Asymmetric self-play [152]

Although not specifically designed for the problem presented in this paper, it is straight forward to apply the method proposed by Sukhbaatar et al. [152] to our problem. An interesting study of its limitations in a similar setting can be found in [37].

SAGG-RIAC [9]

In our implementation of this method, we use TRPO as the “Low-Level Goal-Directed Exploration with Evolving Context". We therefore implement the method as batch: at every iteration, we sample N_{new} new goals $\{y_i\}_{i=0 \dots N_{new}}$, then we collect rollouts of t_{max} steps trying to reach them, and perform the optimization of the parameters using all the collected data. The detailed algorithm is given in the following pseudo-code.

UpdateRegions(R, y_f, Γ_{y_f}) is exactly the Algorithm 2 described in the original paper, and **Self-generate** is the "Active Goal Self-Generation (high-level)" also described in the paper (Section 2.4.4 and Algorithm 1), but it’s repeated N_{new} times to produce a batch of N_{new} goals jointly. As for the competence Γ_{y_g}, we use the same formula as in their section 2.4.1 (use highest competence if reached close enough to the goal) and $C(y_g, y_f)$ is computed with their equation (7). The `collect_rollout` function resets the state $s_0 = s_{reset}$ and then applies actions following the goal-conditioned policy $\pi_\theta(\cdot, y_g)$ until it reaches the goal or the maximum number of steps t_{max} has been taken. The final state, transformed in goal space, y_f is returned.

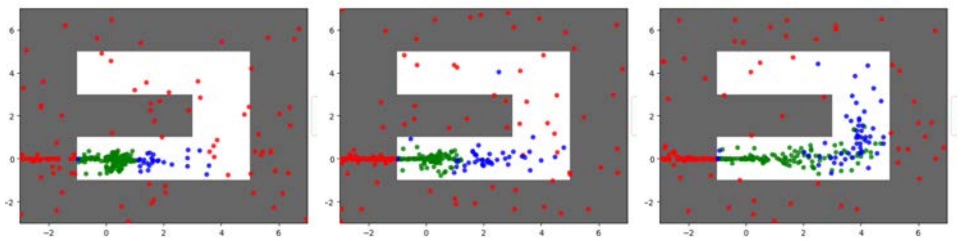
As hyperparameters, we have used the recommended ones in the paper, when available: $p_1 = 0.7$, $p_2 = 0.2$, $p_3 = 0.1$. For the rest, the best performance in an hyperparameter sweep yields: $\zeta = 100$, $g_{max} = 100$. The noise for mode(3) is chosen to be Gaussian with variance 0.1, the same as the tolerance threshold ϵ_{max} and the competence threshold ϵ_C.

As other details, in our tasks there are no constraints to penalize for, so $\rho = \emptyset$. Also, there are no sub-goals. The reset value r is 1 as we reset to s_{start} after every reaching attempt. The number

Algorithm 9: Generative Goal with Sagg-RIAC

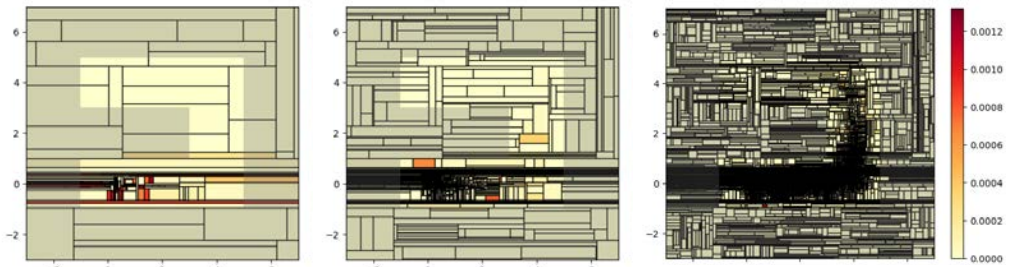
```
Hyper-parameters : window size $\zeta$, tolerance threshold $\epsilon_{max}$, competence threshold $\epsilon_C$,
                   maximum time horizon $t_{max}$, number of new goals $N_{new}$, maximum
                   number of goals $g_{max}$, mode proportions $(p_1, p_2, p_3)$
Input            : Policy $\pi_{\theta_0}(s_{start}, y_g)$, goal bounds $B_Y$, reset position $s_{rest}$
Output           : Policy $\pi_{\theta_N}(s_{start}, y_g)$
R $\leftarrow \{(R_0, \Gamma_{R_0})\}$ where $R_0 = Region(B_Y)$, $\Gamma_{R_0} = 0$;
for $i \leftarrow 1$ to $N$ do
    $goals \leftarrow$ Self-generate $N_{new}$ goals: $\{y_j\}_{j=0 \ldots N_{new}}$;
    $paths = [\,]$;
    while $number\_steps\_in(paths) < batch\_size$ do
        Reset $s_0 \leftarrow s_{rest}$;
        $y_g \leftarrow Uniform(goals)$;
        $y_f, \Gamma_{y_g}, path \leftarrow$ collect_rollout$(\pi_{\theta_i}(\cdot, y_g), s_{reset})$;
        $paths$.append$(path)$;
        UpdateRegions(R, $y_f, 0$) ;
        UpdateRegions(R, $y_g, \Gamma_{y_g}$);
    end
    $\pi_{\theta_{i+1}} \leftarrow$ train $\pi_{\theta_i}$ with TRPO on collected $paths$;
end
```

of explorative movements $q \in \mathbb{N}$ has a less clear equivalence as we use a policy gradient update with a stochastic policy π_θ instead of a SSA-type algorithm.



(a) Iteration 2 (b) Iteration 20 (c) Iterartion 300

Figure A.14: Goals sampled by SAGG-RIAC (same policy training as in Fig. A.15). “High rewards" (in green) are goals with $\bar{R}^g(\pi_i) \geq R_{\max}$; $GOID_i$ (in blue) are those with the appropriate level of difficulty for the current policy ($R_{\min} \leq \bar{R}^g(\pi_i) \leq R_{\max}$). The red ones have $R_{\min} \geq \bar{R}^g(\pi_i)$



(a) Iteration 2: Num. of Regions = 54 (b) Iteration 100: Num. of Regions = 1442 (c) Iteration 300: Num. of Regions = 15420

Figure A.15: Visualization of the regions generated by the SAGG-RIAC algorithm

A.4 Reverse Curriculum for Reinforcement Learning Appendix

Experiment Implementation Details

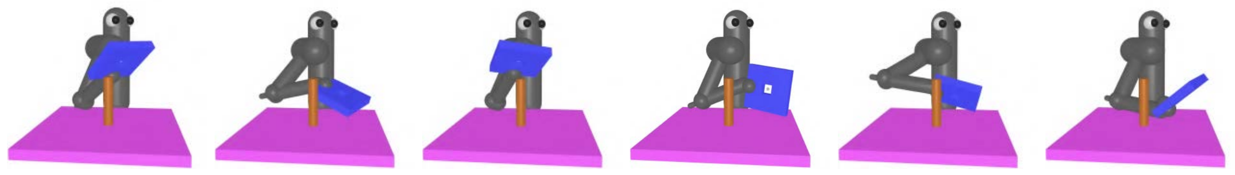
Hyperparameters

Here we describe the hyperparemeters used for our method. Each iteration, we generate new start states (as described in Section 3.6 and Procedure 6), which we append to the seed states until we have a total of $M = 10000$ start states. We then subsample these down to $N_{new} = 200$ new start states. These are appended with $N_{old} = 100$ sampled old start states (as described in Section 3.6 and Procedure 5), and these states are used to initialize our agent when we train our policy. The “Brownian motion" rollouts have a horizon of $T_B = 50$ timesteps, and the actions taken are random, sampled from a standard normal distribution (e.g. a 0-mean Gaussian with a covariance $\Sigma = I$).

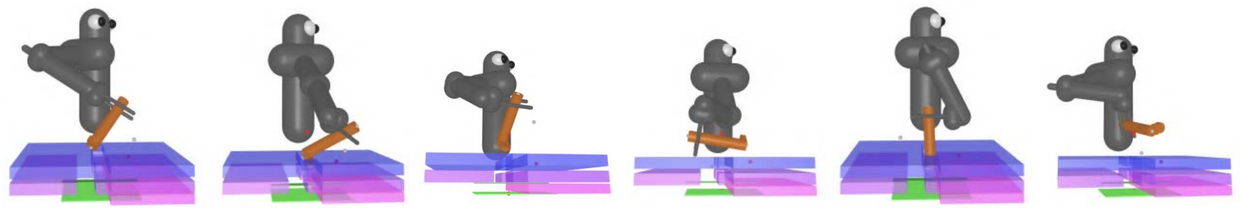
For our method as well as the baselines, we train a $(64, 64)$ multi-layer perceptron (MLP) Gaussian policy with TRPO [138], implemented with rllab [30]. We use a TRPO step-size of 0.01 and a $(32, 32)$ MLP baseline. For all tasks, we train with a batch size of 50,000 timesteps. All experiments use a maximum horizon of $T = 500$ time steps except for the Ant maze experiments that use a maximum horizon of $T = 2000$. The episode ends as soon as the agent reaches a goal state. We define the goal set S^g to be a ball around the goal state, in which the ball has a radius of $0.03m$ for the ring and key tasks, $0.3m$ for the point-mass maze task and $0.5m$ for the ant-maze task. In our definition of S_i^0, we use $R_{\min} = 0.1$ and $R_{\max} = 0.9$. We use a discount factor $\gamma = 0.998$ for the optimization, in order to encourage the policy to reach the goal as fast as possible.

Performance metric

The aim of our tasks is to reach a specified goal region S^g from all start states $s_0 \in S^0$ that are feasible and within a certain distance of that goal region. Therefore, to evaluate the progress on $\eta_{\rho_0}(\pi_i)$ we need to collect trajectories starting at states uniformly sampled from S^0. For the point-mass maze navigation task this is straight forward as the designer can give a concrete description of the feasible (x, y) space, so we can uniformly sample from it. Nevertheless, it is not trivial to uniformly sample from all feasible start states for the robotics tasks. In particular, the state space is in joint angles and angular velocities of the 7 DOF arm, but the physical constraints of these contact-rich environments are given by the geometries of the task. Therefore, uniformly sampling from the angular bounds mostly yields infeasible states, with some part of the arm or the end-effector intersecting with other objects in the scene. In order to approximate uniformly sampling from S^0, we make use of our assumptions (Section 3.6). We simply run our `SampleNearby` procedure initialized with $starts = [s^g]$ with a very large M and long time horizons T_B. This large aggregated state data-set is saved and samples from it are used as proxy to S^0 to evaluate the performance of our algorithm. Figures A.16a and A.16b show six sampled start states from the data sets used to evaluate the ring task and the key task. These data sets are available at the project website[1] for future reproducibility and benchmarking.



(a) Uniformly sampled start states for ring task. There are 39,530 states in the data-set, of which 5,660 have the ring with its hole already in the peg



(b) Uniformly sampled start states for key task. There are 544,575 states in the data-set, of which 120,784 have the key somewhere inside the key-hole

Figure A.16: Samples from the test distribution for the manipulation tasks

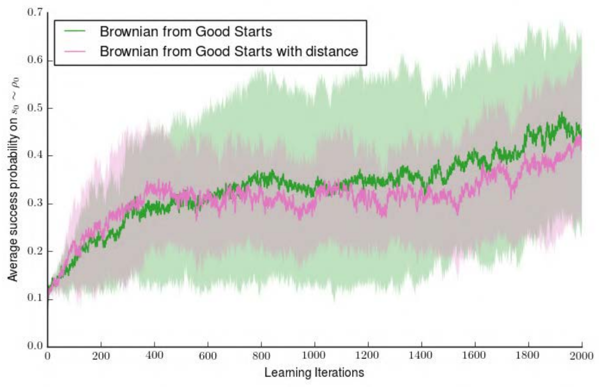
Given the quasi-static nature of the tasks considered, we generate only initial joint positions, and we set all initial velocities to zero. Generating initial velocities is a fairly simple extension of our approach that we leave for future work.

[1]Videos, data sets and code available at: bit.ly/reversecurriculum

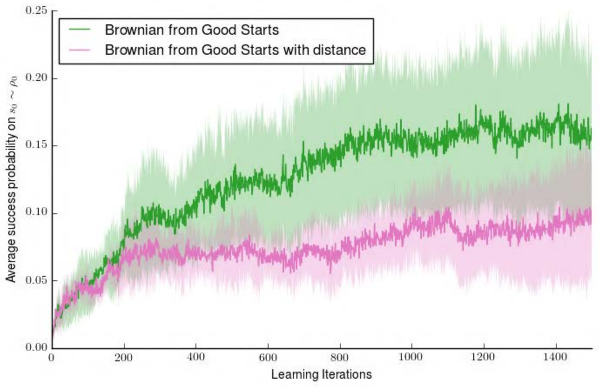
Other methods

Distance reward shaping

Although our method is able to train policies with sparse rewards, the policy optimization steps `train_pol` can use any kind of reward shaping available. To an extent, we already do that by using a discount factor γ, which motivates the policies to reach the goal as soon as possible. Similar reward modulations could be included to take into account energy penalties or reward shaping from prior knowledge. For example, in the robotics tasks considered in this paper, the goal is defined in terms of a reference state, and hence it seems natural to try to use the distance to this state as an additional penalty to guide learning. However, we have found that this modification does not actually improve training. For the start states near to the goal, the policy can learn to reach the goal simply from the indicator reward introduced in Section . For the states that are further away, the distance to the goal is actually not a useful metric to guide the policy; hence, the distance reward actually guides the policy updates towards a suboptimal local optimum, leading to poor performance. In Fig. A.17 we see that the ring task is not much affected by the additional reward, whereas the key task suffers considerably if this reward is added.



(a) Ring on Peg task



(b) Key insertion task

Figure A.17: Learning curves for the robotics manipulation tasks

Failure cases of Uniform Sampling for maze navigation

In the case of the maze navigation task, we observe that applying TRPO directly on the original MDP incurs a very high variance across learning curves. We have observed that some policies only learned how to perform well from a certain side of the goal. The reason for this is that our learning algorithm (TRPO) is a batch on-policy method; therefore, at the beginning of learning, uniformly sampling from the state-space might give a batch with very few trajectories that reach the goal and hence it is likely that the successful trajectories all come from one side of the goal. In this case, the algorithm will update the policy to go in the same direction from everywhere, wrongly extrapolating from these very few successful trajectories it received. This is less likely to happen if the trajectories for the batch are collected with a different start state distribution that concentrates more uniformly around the goal, as the better learning progress of the other curves show.

Failure cases of Asymmetric Self-play

In Section 3.7, we compare the performance of our method to the asymmetric self-play approach of Sukhbaatar et al. [152]. Although such an approach learns faster than the uniform sampling baseline, it gets stuck in a local optimum and fails to learn to reach the goal from more than 40% of start-states in the point-mass maze task.

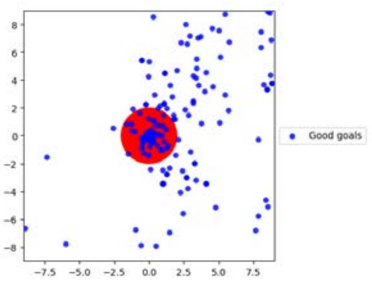
As explained above, part of the reason that this method gets stuck in a local optimum is that "Alice" (the policy that is proposing start-states) is represented with a unimodal Gaussian distribution, which is a common representation for policies in continuous action spaces. Thus Alice's policy tends to converge to moving in a single direction. In the original paper, this problem is somewhat mitigated by using a discrete action space, in which a multi-modal distribution for Alice can be maintained. However, even in such a case, the authors of the original paper also observed that Alice tends to converge to a local optimum [152].

A further difficulty for Alice is that her reward function can be sparse, which can be inherently difficult to optimize. Alice's reward is defined as $r_A = \max(0, t_B - t_A)$, where t_A is the time that it takes Alice to reach a given start state from the goal (at which point Alice executes the "stop" action), and t_B is the time that it takes Bob to return to the goal from the start state. Based on this reward, the optimal policy for Alice is to find the nearest state for which Bob does not know how to return to the goal; this will lead to a large value for t_B with a small value for t_A. In theory, this should lead to an automatic curriculum of start-states for Bob.

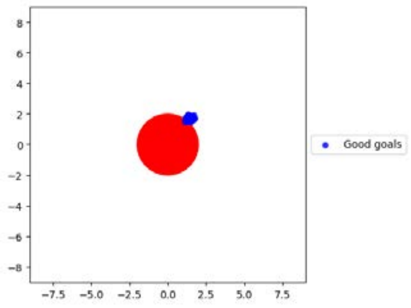
However, in practice, we find that sometimes, Bob's policy might improve faster than Alice's. In such a case, Bob will have learned how to return to the goal from many start states much faster than Alice can reach those start states from the goal. In such cases, we would have that $t_B < t_A$, and hence $r_A = 0$. Thus, Alice's rewards are sparse (many actions that Alice takes result in 0 reward) and hence it will be difficult for Alice's policy to improve, leading to a locally optimal policy for Alice. For these reasons, we have observed Alice's policy often getting "stuck," in which Alice is unable to find new start-states to propose for Bob that Bob does not already know how to reach the goal from.

We have implemented a simple environment that illustrates these issues. In this environment, we use a synthetic "Bob" that can reach the goal from any state within a radius r_B from the goal. For states within r_B, Bob can reach the goal in a time proportional to the distance between the state and the goal; in other words, for such states $s_0 \in \{s : |s - s^g| < r_B, s \in S^0\}$, we have that $t_B = |s_0 - s^g|/v_B$, where $|s_0 - s^g|$ is the distance between state s_0 and the goal s^g, and v_B is Bob's speed. For states further than r_B from the goal, Bob does not know how to reach the goal, and thus t_B for such states takes the maximum possible value.

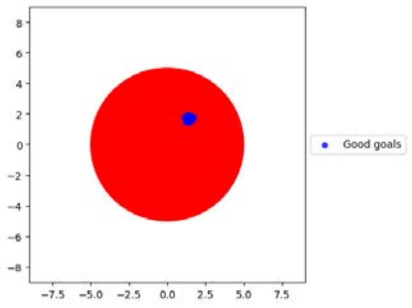
This setup is illustrated in Figure A.18. The region shown in red designates the area within r_B from the goal, e.g. the set of states from which Bob knows how to reach the goal. On the first iteration, Alice has a random policy (Figure A.18a). After 10 iterations of training, Alice has converged to a policy that reaches the location just outside of the set of states from which Bob knows how to reach the goal (Figure A.18b). From these states, Alice receives a maximum reward, because t_B is very large while t_A is low. Note that we also observe the unimodal nature of Alice's policy; Alice has converged to a policy which proposes just one small set of states among all possible states for which she would receive a similar reward.



(a) Iteration 1



(b) Iteration 10



(c) Iteration 32

Figure A.18: Simple environment to illustrate asymmetric self-play [152]. The red areas indicate the states from which Bob knows how to reach the goal. The blue points are the start-states proposed by Alice at each iteration (i.e. the states from which Alice performed the stop action)

.

At this point we synthetically increase r_B, corresponding to the situation in which Bob learns how to reach the goal from a larger set of states. However, Alice's policy has already converged to reaching a small set of states which were optimal for Bob's previous policy. From these states Alice now receives a reward of 0, as described above: Bob can return from these states quickly to the goal, so we have that $t_B < t_A$ and $r_A = 0$. Thus, Alice does not receive any reward signal and is not able to improve her policy. Hence, Alice's policy remains stuck at this point and she is not able to find new states to propose to Bob (Figure A.18c).

In this simple case, one could attempt to perform various hacks to try to fix the situation, e.g. by artificially increasing Alice's variance, or by resetting Alice to a random policy. However, note that, in a real example, Bob is learning an increasingly complex policy, and so Alice would need to learn an equally complex policy to find a set of states that Bob cannot succeed from; hence, these simple fixes would not suffice to overcome this problem. Fundamentally, the asymmetric nature of the self-play between Alice and Bob creates a situation in which Alice has a difficult time learning and often gets stuck in a local optimum from which she is unable to improve.