STOCHASTIC NEURAL NETWORKS FOR HIERARCHICAL REINFORCEMENT LEARNING

Carlos Florensa[†], **Yan Duan**[†‡], **Pieter Abbeel**[†‡]
[†] UC Berkeley, Department of Electrical Engineering and Computer Science
[‡] OpenAI
florensa@berkeley.edu, {rocky,pieter}@openai.com

ABSTRACT

Deep reinforcement learning has achieved many impressive results in recent years. However, many of the deep RL algorithms still employ naive exploration strategies, and they have been shown to perform poorly in tasks with sparse rewards, and/or with long horizons. To tackle these challenges, there are two common approaches. The first approach is to design a hierarchy over the actions, which would require domain-specific knowledge and careful hand-engineering. A different line of work utilizes domain-agnostic intrinsic rewards to guide exploration, which has been shown to be effective in tasks with sparse rewards. However, it is unclear how the knowledge of solving a task can be utilized for other tasks, leading to a high sample complexity overall for the entire collection of tasks. In this paper, we propose a general framework for learning useful skills in a pre-training environment, which can then be utilized in downstream tasks by training a high-level policy over these skills. To learn these skills, we use stochastic neural networks (SNNs) combined with a proxy reward, the design of which requires very minimal domain knowledge about the downstream tasks. Our experiments show that this combination is effective in learning a wide span of interpretable skills in a sample-efficient way, and, when used on downstream tasks, can significantly boost the learning performance uniformly across all these tasks.

1 INTRODUCTION

In recent years, deep reinforcement learning has achieved many impressive results, including playing Atari games using raw pixel inputs (Guo et al., 2014; Mnih et al., 2015; Schulman et al., 2015), mastering the game of Go (Silver et al., 2016), and acquiring advanced manipulation and locomotion skills from raw sensory inputs (Schulman et al., 2015; Lillicrap et al., 2015; Watter et al., 2015; Schulman et al., 2016; Heess et al., 2015; Levine et al., 2016). 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 (Duan et al., 2016; Houthooft et al., 2016; Bellemare et al., 2016). Tasks with sparse rewards are common in realistic scenarios. For example, in navigation tasks, the agent is not rewarded until it finds the target. The challenge is further complicated by long horizons, where naive exploration strategies can lead to exponentially large sample complexity (Osband et al., 2014).

To tackle these challenges, there are two natural strategies. The first strategy is to design a hierarchy over the actions (Parr & Russell, 1998; Sutton et al., 1999; Dietterich, 2000). 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. Another line of work uses intrinsic rewards to guide exploration (Schmidhuber, 1991; 2010; Houthooft et al., 2016; Bellemare et al., 2016). 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 paper, we propose a general framework for training policies in a collection of tasks with sparse rewards. Our framework first learns a span of skills in a pre-training environment, where it

is employed with nothing but a proxy reward signal, whose design only require very minimal domain knowledge about the downstream tasks. This proxy reward can be understood as some kind 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) (Neal, 1990; 1992; Tang & Salakhutdinov, 2013), 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 find that our hierarchical policy-learning framework can learn a wide range of skills, which are clearly interpretable as the latent code is varied. Furthermore, we show that training high-level policies on top of the learned skills can lead to great performance on a set of challenging tasks with long horizons and sparse rewards.

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 (Parr & Russell, 1998; Sutton et al., 1999; Dietterich, 2000). 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 (Chentanez et al., 2004; Vigorito & Barto, 2010). A popular approach there is to use statistics about state transitions to identify bottleneck states (Stolle & Precup, 2002; Mannor et al., 2004; Şimşek et al., 2005). It is however not clear how these techniques can be applied to settings with high-dimensional continuous spaces. More recently, Mnih et al. (2016) propose a DRAW-like (Gregor et al., 2015) recurrent neural network architecture that can learn temporally extended macro actions. However, the learning needs to be guided by external rewards as supervisory signals, and hence the algorithm cannot be straightforwardly applied in sparse reward settings.

There has also been work on learning skills in tasks with continuous actions (Schaal et al., 2005; Konidaris et al., 2011; Daniel et al., 2013; Ranchod et al., 2015). These methods extract useful skills from good-performing trajectories in the same or related tasks, and hence require either demonstrations to be provided, or first training on similar tasks with non-sparse reward signals.

Another line of work on skill discovery particularly relevant to our approach is the HiREPS algorithm by Daniel et al. (2012), where instead of having a MI bonus like in this paper, 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 (Bacon & Precup, 2016) can learn interpretable skills, but whether they can be reuse across complex tasks is still an open question. Recently, Heess et al. (2016) 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 proxy rewards 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.

3 Preliminaries

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 $s_{t+1} \sim \mathcal{P}(s_{t+1}|s_t, a_t)$.

4 PROBLEM STATEMENT

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 interacts 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 can imagine that a robot 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* (Konidaris & Barto, 2007).

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 a sequential setting, 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 (Taylor & Stone, 2009; Wilson et al., 2007; Lazaric & Ghavamzadeh, 2010; Devin et al., 2016). However, this requires that the earlier tasks are relatively easy to solve through good reward shaping, 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.

5 METHODOLOGY

In this section, we describe our formulation to solve a collection of tasks, exploiting the structural assumption articulated above. In Sec. 5.1, we describe the pre-training environment, where we use proxy rewards that will be useful for learning a span of skills. In Sec. 5.2, 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. 5.3, we describe an information-theoretic regularizer that further improves the span of skills learned by SNNs. In Sec. 5.4, 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. 5.5, we describe the policy optimization for both phases of training.

5.1 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 that will be useful for 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.

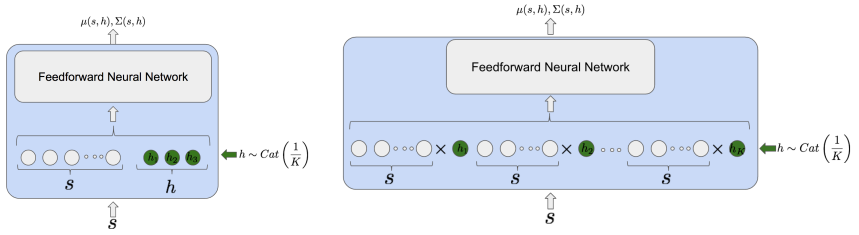
Rather than specifying goals in the pre-training environment corresponding to the desired skills, which requires precise specification about what the skills should entail, we use a proxy reward as the only reward signal to guide skill learning. The design of the proxy reward should encourage the existence of locally optimal solutions, which will correspond to different skills the agent should learn. 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.

5.2 STOCHASTIC NEURAL NETWORKS FOR SKILL LEARNING

Having constructed the pre-training environment, a straightforward way to learn a span of skills is to train different policies, each with a uni-modal action distribution (e.g. a multivariate Gaussian) under different random initializations, and hope that they will converge to different local optima, hence learning different skills. However, the sample complexity of this approach is at least directly proportional to the number of skills we would like to learn. This can become quite expensive when the agent learns from high-dimensional sensory signals like images, and needs to rebuild a good feature representation from scratch. In addition, we do not have any control over whether the different policies actually learn different skills. We address these two issues separately in the current and the next subsection.

To tackle the first issue, 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) (Smolensky, 1986; Hinton, 2002), Deep Belief Networks (DBNs) (Hinton et al., 2006), and Sigmoid Belief Networks (SBNs) (Neal, 1990; Tang & Salakhutdinov, 2013). They have rich representation power and can in fact approximate any well-behaved probability distributions (Le Roux & Bengio, 2008; Cho et al., 2013). 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 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 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 a hyperparameter that upper bounds the number of skills that we would like to learn.



(a) Concatenation (b) Bilinear integration

Figure 1: Different architectures for the integration of the latent variables in a FNN

The simplest joint embedding, as shown in Figure 1(a), is to concatenate the observations and the latent variables directly.[1] 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 (Fukui et al., 2016; Wu et al., 2016). Inspired by this work, we study using a simple bilinear integration, by forming the outer product between the observation and the latent variable (Fig. 1(b)). Note that the concatenation integration effectively corresponds to changing the bias term of the first hidden layer depending on the latent code h 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.

Compared to training separate policies, training a single SNN allows for flexible weight-sharing schemes among different policies. To obtain temporally extended and consistent behavior associated to 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. Finally, we can easily further encourage

[1] 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.

the diversity of skills learned by the SNN by introducing an information theoretic regularizer, as detailed in the next section. After training, each of the latent code in the categorical distribution will correspond to a different, interpretable skill, which can then be used for the downstream tasks.

5.3 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 GANs (Chen et al., 2016).

We design this objective specifically for mobile robots, although in principle it can be also generalized to other contexts. Concretely, we add an additional reward bonus, proportional to the mutual information (MI) between the latent variable and the state it is currently in. We only measure the MI with respect to a relevant subset in the state. For a mobile robot, we choose this to be the coordinates of its center of mass. We further partition the (x, y) coordinate space into discrete grids, and map the continuous-valued coordinate into the closest grid. This way, calculation of empirical MI only requires maintaining visitation counts of the latent variables in each grid.

Mathematically, let X be a random variable denoting the grid in which the agent is currently situated, and let Z be a random variable for the latent variable. Our additional reward bonus is $\alpha_H I(Z; X) = \alpha_H(H(Z) - H(Z|X))$, where H denotes the entropy function. In our case, $H(Z)$ is constant since the distribution of the latent variable is fixed, and hence maximizing MI is equivalent to minimizing the conditional entropy. Therefore, another interpretation of this bonus is that given where the robot is, it should be easy to infer what skill the robot is currently performing.

5.4 Learning High-level Policies

Given a span of skills learned from 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. Assuming that the categorical latent variable of the pretrained SNN has K classes, and hence there are K different skills, we train a high-level policy (Manager Neural Network) that operates by selecting among these different skills. For a given task $M \in \mathcal{M}$, and given a known 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 out of K possible choices, as depicted in Fig. 2. The high-level policy operates at a slower time scale than the SNN policy, only switching its categorical output every $\mathcal{T}$ time-steps. We call $\mathcal{T}$ the switch time and it is a hyperparameter to be determined based on the downstream task (see Appendix C.3 for a sensitivity analysis). Note that the same structure can be used to switch between K full policies independently pre-trained; this will be one of our baselines (called Multi-policy) further detailed in the experiment section.

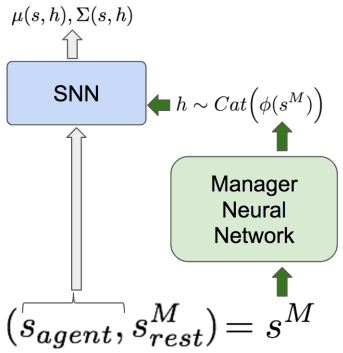


Figure 2: Hierarchical SNN architecture to solve downstream tasks

We freeze the weights of the SNN during this phase of training. In principle, they can also be jointly optimized to adapt the skills to the task at hand. 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 Jang et al. (2016) or Maddison et al. (2016). 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.

5.5 Policy Optimization

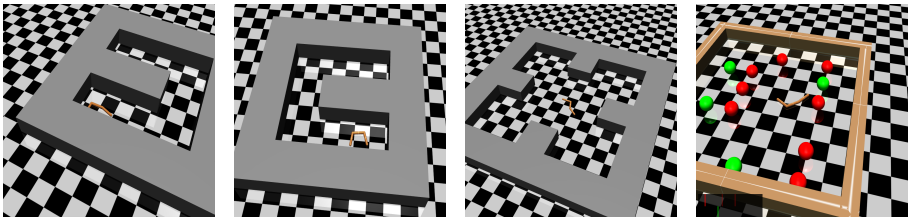
For both the pre-training phase and the training of the high-level policies, we use Trust Region Policy Optimization (TRPO) as the policy optimization algorithm (Schulman et al., 2015). We

choose TRPO due to its excellent empirical performance and because it does not require excessive hyperparameter tuning. For the pre-training phase, the training of policies is complicated by the fact that due to the presence of 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 expressive latent variables. This also complicates the computation of KL divergence, which is needed to control the step size of the policy update. In our implementation, we hold the latent variable as fixed in the optimization phase, and compute $\pi(a|s, z)$, where z is the latent variable, without marginalizing over z. The rest of the TRPO procedure can then be straightforwardly applied.

6 EXPERIMENTS

We have applied our framework to the two hierarchical tasks described in the benchmark by Duan et al. (2016): Locomotion + Maze and Locomotion + Food Collection (Gather). The observation space of these tasks naturally decompose into S_{agent} being the robot and S^M_{rest} 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. We report results with more complex robots in Appendix C-D.

To increase the variety of downstream tasks, we have constructed 4 different mazes. Maze 0 is the same described in the benchmark (Duan et al., 2016) and Maze 1 is its reflexion, where the robot has to go backwards-right-right instead of forward-left-left. These correspond to Figs. 3(a)-3(b), 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. 3(c), 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. 3(d), 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. The sensor readings for the Mazes and the Gather tasks are further described in the benchmark of continuous control problems (Duan et al., 2016), where it was shown that algorithms that employ naive exploration strategies could not solve them. More advanced intrinsically motivated explorations (Houthooft et al., 2016) do achieve some progress, and we report our stronger results with the exact same setting in Appendix B. Our hyperparameters for the neural network architectures and algorithms are detailed in the Appendix A.



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

Figure 3: Illustration of the sparse reward tasks

7 RESULTS

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 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?

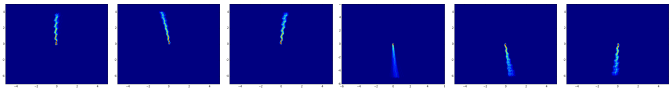
Appart from the explanations below, videos of the achieved results are available[2].

[2] https://goo.gl/5wp4VP

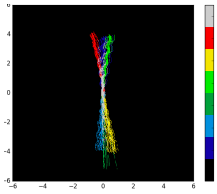
7.1 SKILL LEARNING IN PRETRAIN

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. 4(a) 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. 4(b) 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 ways of advancing, hence granting a potentially useful skills to solve downstream tasks when embedded in our described Multi-policy hierarchical architecture.

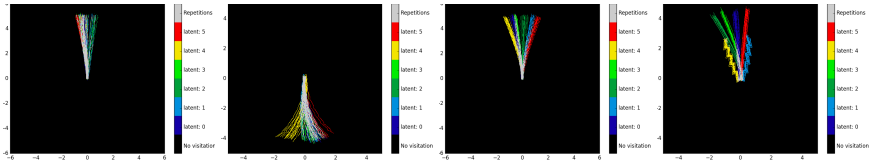
Next we show that, using SNNs, we can learn a similar or even larger span of skills without training several independent policies. The number of samples used to train a SNN is the same as a single feed-forward policy from Fig. 4(a), therefore this method of learning skills effectively reduces the sample complexity by a factor equal to the numbers of skills being learned and with an adequate architecture and MI bonus, we can even generate a richer span. In the two rows of Figs. 4(c)-4(d) we present the visitation plots of SNNs policies obtained for different design choices. Now the colors indicate the latent that was sampled at the beginning of the rollout. We observe that each latent 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 latents (concatenation in first row and bilinear in the second) and the MI bonus (increasing coeficient α_H towards the right). Simple concatenation of latents with the observations rarely yields distinctive behaviors for each latent. 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 latents. 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 latents.



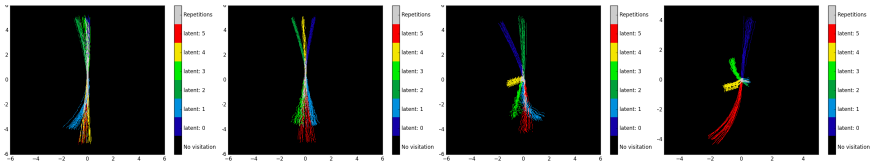
(a) Independently trained policies in the pre-train MDP with the proxy reward of the CoM speed norm



(b) Superposed policy visitations from (a)



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

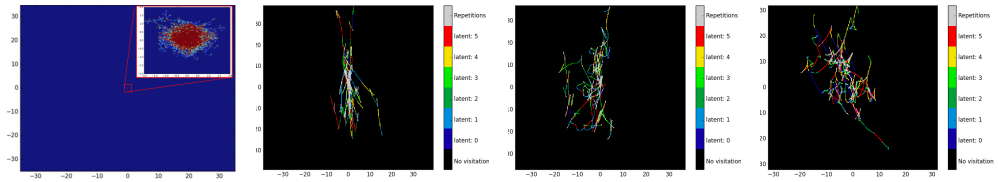


(d) SNN *with* bilinear integration and increasing $\alpha_H = 0, 0.001, 0.01, 0.1$

Figure 4: Span of skills learn by different methods and architectures

7.2 HIERARCHICAL USE OF SKILLS

The hierarchical architectures we propose have a direct impact on the areas covered by random exploration. We will examine these with visitation plots of a single rollout of one million steps. The exploration obtained with Gaussian noise at the output with $\mu = 0$ and $\Sigma = I$ is reported in Fig. 5(a). This noise is relatively large as the swimmer robot has actions clipped to $[-1, 1]$, but it still does not yield good exploration. On the other hand, using our hierarchical structures with pretrained policies yields a drastic increase in exploration, as shown in Fig. 5(b)-5(d). All these plots show a possible rollout under a randomly initialized Manager Network, hence outputting uniformly distributed one-hot vectors every $\mathcal{T} = 500$ steps, also for one million steps. In Fig. 5(b) the one-hot output of the Manager Network is used to select one of the six policies which visitation is shown in Fig. 4(a). This hierarchical architecture is our strong baseline (it has used 6 times more samples for pretrain because it trained every policy independently) and will be referred to as Multi-policy. Our architecture using the SNN described in Sec. 5.4 uses the one-hot vector instead of the latent. The same one million steps visitation, for pre-trained bilinear SNN (Bil-SNN) with $\alpha_H = 0$ and 0.01 can be seen in Figs. 5(c) and 5(d) respectively. The exploration given by the Multi-policy hierarchy concentrates quite heavily in upward and downword motion, as is expected form the individual policies composing it (refer back to Fig. 4(a)). On the other hand the exploration obtained with SNNs yields a wider coverage of the space as the underlying policy usually has additional behaviors. Note how most of the changes of latent yield a corresponding change in direction. Our simple pre-train setup and the interpretability of the obtained skills are key advantages with respect to other similar hierarchical structures like seen in Heess et al. (2016).



(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 $\alpha_H = 0.01$

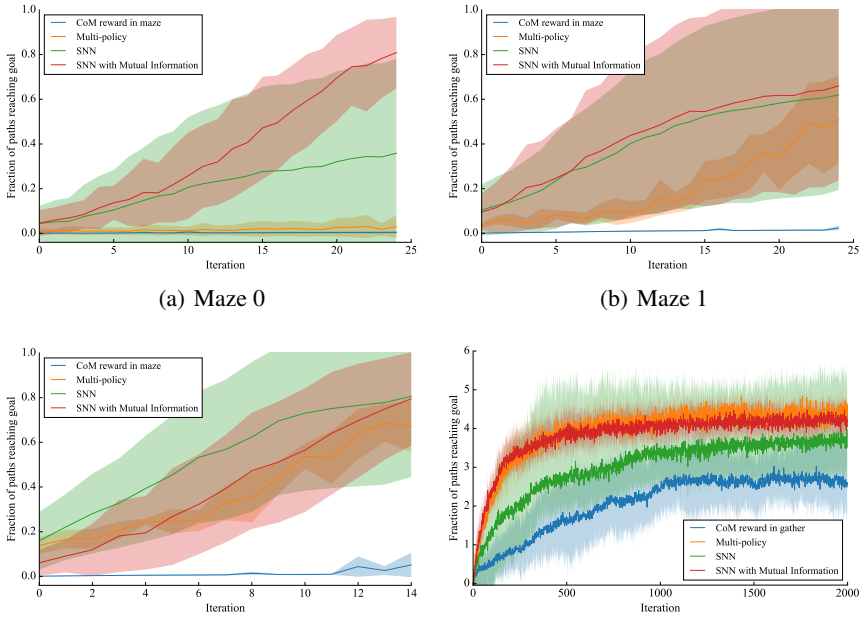
Figure 5: Visitation plots for different randomly initialized architectures (one rollout of 1M steps). All axis are in scale [-30,30] and we added a zoom for the Gaussian noise to scale [-2,2]

7.3 MAZES AND GATHER TASKS

In Fig. 6 we evaluate the different hierarchical architectures proposed. Due to the sparsity of these tasks, none can be properly solved by standard reinforcement algorithms (Duan et al., 2016). 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. 6(a)-6(c). 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 boosts 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. 6(d), 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 B.

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 C.4 for more details.



(a) Maze 0

(b) Maze 1

(c) Aggregated results for Mazes 2 and 3

(d) Gather task

Figure 6: Faster learning of the hierarchical architectures in the sparse downstream MDPs

8 Discussion and future work

We propose a framework for learning a diverse set of skills using stochastic neural networks with minimum supervision, and utilize these skills in a hierarchical architecture to solve challenging tasks with sparse rewards. Our 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, we demonstrate how drastically it can boost the exploration of an agent in a new environment and we demonstrate its relevance for solving complex tasks as mazes or gathering.

One limitations of our current approach is the switching between skills for unstable agents, as reported in the Appendix D.2 for the "Ant" robot. There are multiple future directions to make our framework more robust to such challenging robots, like learning a transition policy or integrating switching in the pretrain task. Other limitations of our current approach are having fixed sub-policies and fixed switch time $\mathcal{T}$ during the training of downstream tasks. The first issue can be alleviated by introducing end-to-end training, for example using the new straight-through gradient estimators for Stochastic Computations Graphs with discrete latents (Jang et al., 2016; Maddison et al., 2016). The second issue is not critical for static tasks like the ones used here, as studied in Appendix C.3. But in case of becoming a bottleneck for more complex dynamic tasks, a termination policy could be learned by the Manager, similar to the option framework. Finally, we only used feedforward architectures and hence the decision of what skill to use next only depends on the observation at the moment of switching, not using any sensory information gathered while the previous skill was active. This limitation could be eliminated by introducing a recurrent architecture at the Manager level.

ACKNOWLEDGMENTS

This work was supported in part by DARPA, the Berkeley Vision and Learning Center (BVLC), the Berkeley Artificial Intelligence Research (BAIR) laboratory, and Berkeley Deep Drive (BDD). This research was funded in part by ONR through a PECASE award. Carlos Florensa was also supported by the Ph.D. Fellowship of La Caixa - Spain. Yan Duan was also supported by a Berkeley AI Research lab Fellowship and a Huawei Fellowship.

REFERENCES

Pierre-Luc Bacon and Doina Precup. The option-critic architecture. *arXiv preprint arXiv:1609.05140v2*, 2016.

Marc G Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *Advances in Neural Information Processing Systems*, 2016.

Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *Advances in Neural Information Processing Systems*, 2016.

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

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

Christian Daniel, Gerhard Neumann, and Jan Peters. Hierarchical relative entropy policy search. In *AISTATS*, pp. 273–281, 2012.

Christian Daniel, Gerhard Neumann, and Jan Peters. Autonomous reinforcement learning with hierarchical reps. In *International Joint Conference on Neural Networks*, pp. 1–8. IEEE, 2013.

Coline Devin, Abhishek Gupta, Trevor Darrell, Pieter Abbeel, and Sergey Levine. Learning modular neural network policies for multi-task and multi-robot transfer. *arXiv preprint arXiv:1609.07088*, 2016.

Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. *International Conference on Machine Learning*, 2016.

Akira Fukui, Dong Huk Park, Daylen Yang, Anna Rohrbach, Trevor Darrell, and Marcus Rohrbach. Multimodal compact bilinear pooling for visual question answering and visual grounding. *arXiv preprint arXiv:1606.01847*, 2016.

Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015.

Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard L Lewis, and Xiaoshi Wang. Deep learning for real-time atari game play using offline monte-carlo tree search planning. In *Advances in Neural Information Processing Systems*, pp. 3338–3346, 2014.

Nicolas Heess, Gregory Wayne, David Silver, Tim Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, pp. 2944–2952, 2015.

Nicolas Heess, Greg Wayne, Yuval Tassa, Timothy Lillicrap, Martin Riedmiller, and David Silver. Learning and transfer of modulated locomotor controllers. *arXiv preprint arXiv:1610.05182*, 2016.

Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, 2002.

Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Variational information maximizing exploration. *Advances in Neural Information Processing Systems*, 2016.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

George Konidaris and Andrew G Barto. Building portable options: Skill transfer in reinforcement learning. In *IJCAI*, volume 7, pp. 895–900, 2007.

George Konidaris, Scott Kuindersma, Roderic A Grupen, and Andrew G Barto. Autonomous skill acquisition on a mobile manipulator. In *AAAI*, 2011.

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

Nicolas Le Roux and Yoshua Bengio. Representational power of restricted boltzmann machines and deep belief networks. *Neural Computation*, 20(6):1631–1649, 2008.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

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

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

Shie Mannor, Ishai Menache, Amit Hoze, and Uri Klein. Dynamic abstraction in reinforcement learning via clustering. In *21st International Conference on Machine Learning*, pp. 71. ACM, 2004.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Volodymyr Mnih, John Agapiou, Simon Osindero, Alex Graves, Oriol Vinyals, Koray Kavukcuoglu, et al. Strategic attentive writer for learning macro-actions. *arXiv preprint arXiv:1606.04695*, 2016.

Radford M Neal. Learning stochastic feedforward networks. *Department of Computer Science, University of Toronto*, 1990.

Radford M Neal. Connectionist learning of belief networks. *Artificial intelligence*, 56(1):71–113, 1992.

Ian Osband, Benjamin Van Roy, and Zheng Wen. Generalization and exploration via randomized value functions. *arXiv preprint arXiv:1402.0635*, 2014.

Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. *Advances in Neural Information Processing Systems*, pp. 1043–1049, 1998.

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*, pp. 471–477. IEEE, 2015.

Stefan Schaal, Jan Peters, Jun Nakanishi, and Auke Ijspeert. Learning movement primitives. In *Robotics Research. The Eleventh International Symposium*, pp. 561–572, 2005.

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

Jürgen Schmidhuber. Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE Transactions on Autonomous Mental Development*, 2(3):230–247, 2010.

J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz. Trust region policy optimization. In *International Conference on Learning Representations*, pp. 1889–1897, 2015.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations*, 2016.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

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*, pp. 816–823. ACM, 2005.

Paul Smolensky. Information processing in dynamical systems: Foundations of harmony theory. Technical report, DTIC Document, 1986.

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

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1):181–211, 1999.

Yichuan Tang and Ruslan R Salakhutdinov. Learning stochastic feedforward neural networks. In *Advances in Neural Information Processing Systems 26*, pp. 530–538, 2013.

Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10:1633–1685, 2009.

Christopher M Vigorito and Andrew G Barto. Intrinsically motivated hierarchical skill learning in structured environments. *IEEE Transactions on Autonomous Mental Development*, 2(2):132–143, 2010.

Manuel Watter, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in Neural Information Processing Systems*, pp. 2746–2754, 2015.

Aaron Wilson, Alan Fern, Soumya Ray, and Prasad Tadepalli. Multi-task reinforcement learning: a hierarchical bayesian approach. In *Proceedings of the 24th international conference on Machine learning*, pp. 1015–1022. ACM, 2007.

Yuhuai Wu, Saizheng Zhang, Ying Zhang, Yoshua Bengio, and Ruslan Salakhutdinov. On multiplicative integration with recurrent neural networks. *arXiv preprint arXiv:1606.06630*, 2016.

A 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 number of latents 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 (Duan et al., 2016): 50,000 and 500 respectively. For the downstream tasks, see Tab. 1.

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

Table 1: Parameters of algorithms for downstream tasks

B 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. (2016): maximum path length of 500 and batch-size of 50k. Our SNN hierarchical approach outperforms state-of-the-art intrinsic motivation results like VIME (Houthooft et al., 2016).

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.

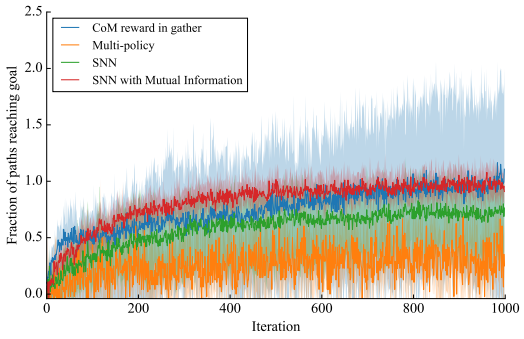


Figure 7: Results for Gather environment in the benchmark settings

C SNAKE

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. 8. 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.

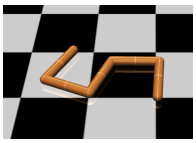


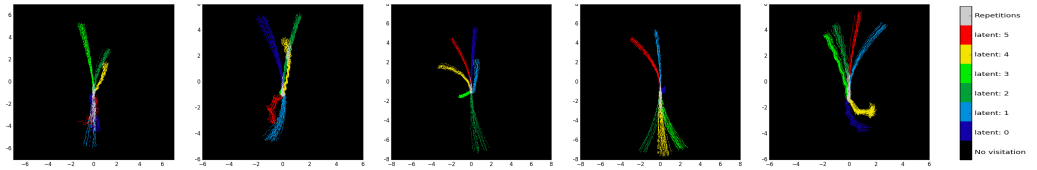
Figure 8: Snake

C.1 SKILL LEARNING IN PRETRAIN

The Snake robot can learn a large span of skills as consistently as Swimmer. We report in Figs. 9(a)-9(e) the visitation plots obtained for five different random seeds in our SNN pretraining algorithm from Secs. 5.1-5.3, with Mutual Information bonus of 0.05. The impact of the different spans of skills on the later hierarchical training is discussed in Sec. C.4.

C.2 HIERARCHICAL USE OF SKILLS IN MAZE AND GATHER

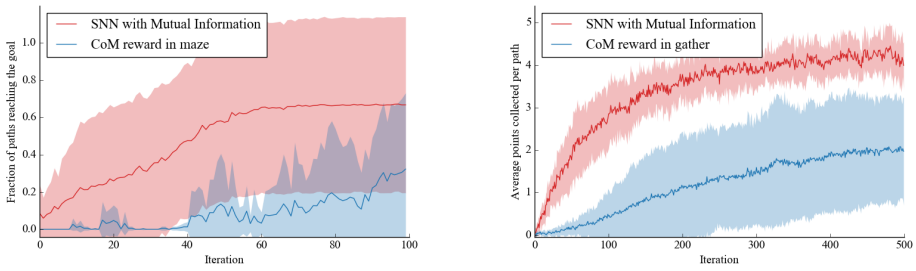
Here we evaluate the performance of our hierarchical architecture to solve with Snake the sparse tasks Maze 0 and Gather from Sec. 6. Given the larger size of the robot, we have also increased



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

Figure 9: Span of skills learned using different random seeds to pretraining a SNN with MI $= 0.05$

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. C.3 for an analysis on $\mathcal{T}$). Despite the tasks being harder, our approach still achieves good results, and Figs. 10(a)-10(b), 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 10: Faster learning of our hierarchical architecture in sparse downstream MDPs with Snake

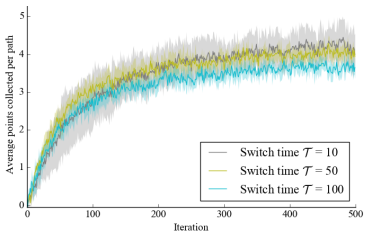
C.3 Analysis of the switch time $\mathcal{T}$

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. 11(a)-11(b). 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. 11(a)). 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. 11(b)). It is still surprising the mild changes produced by such large variations in the switch time $\mathcal{T}$ for the Gather task.

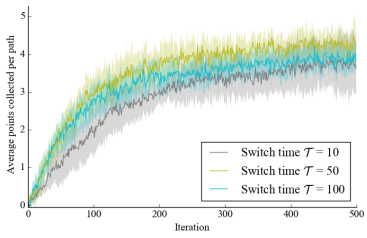
For the Maze 0 task, Figs. 11(c)-11(d) show that the difference between different $\mathcal{T}$ is important but not critical. In particular, radical increases of $\mathcal{T}$ have little effect on the performance in this task because the robot will simply keep bumping against a wall until it switches latent. 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. C.4.

C.4 Impact of the pre-trained SNN on the hierarchy

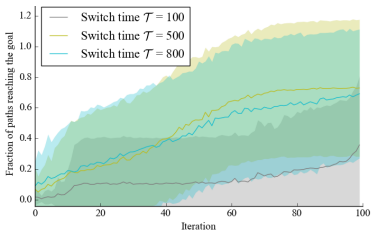
In the previous section, the low variance of the learning curves for Gather indicates that all pretrained SNN preform equally good in this task. For Maze, the performance depends strongly on the pretrained SNN. For example we observe that the one obtained with the random seed 50 has a visitation with a weak backward span (Fig. 9(e)), which happens to be critical to solve the Maze 0



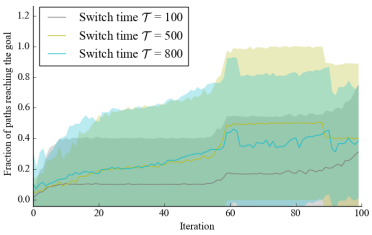
(a) Gather task with size 10



(b) Gather task with size 15



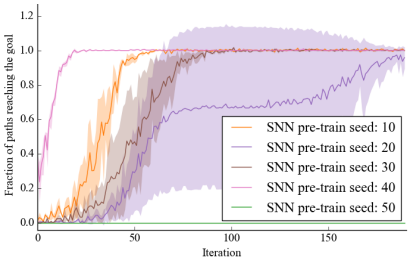
(c) Maze 0 task with size 7



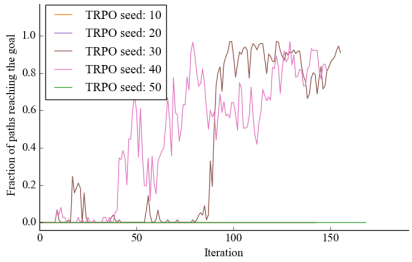
(d) Maze 0 task with size 9

Figure 11: Mild effect of switch time $\mathcal{T}$ on different sizes of Gather and Maze 0

(as can be seen in the videos[2]). This explains the lack of learning in Fig. 12(a) 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. 12(b). 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 12: Variation in performance among different pretrained SNNs and different TRPO seeds

D ANT

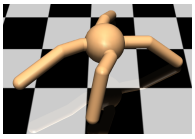


Figure 13: Ant

In this section we report the progress and failure modes of our approach with more challenging and unstable robots, like Ant (described in Duan et al. (2016)). This agent has a 27-dimensional space and an 8-dimensional action space. It is unstable in the sense that it might fall over and cannot recover. We show that despite being able to learn well differentiated skills with SNNs, switching between skills is a hard task because of the instability.

D.1 SILL LEARNING IN PRETRAIN

Our SNN pretraining step is still able to yield large span of skills covering most directions, as long as the MI bonus is well tuned, as observed in Fig. 14. Videos comparing all the different gaits can be observed in the attached videos[2].

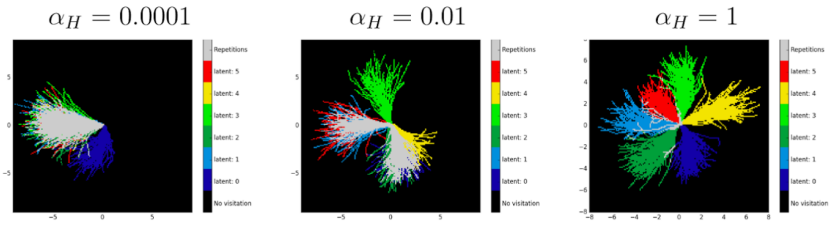


Figure 14: Pretrain visitation for Ant with different MI bonus

D.2 Failure modes for unstable robots

When switching to a new skill, the robot finds itself in a region of the state-space that might be unknown to the new skill. This increases the instability of the robot, and for example in the case of Ant this might leads to falling over and terminating the rollout. We see this in Fig. 15, where the 5 depicted rollouts terminate because of the ant falling over. For more details on failure cases with Ant and the possible solutions, please refer to the technical report: `https://goo.gl/JjmPqM`

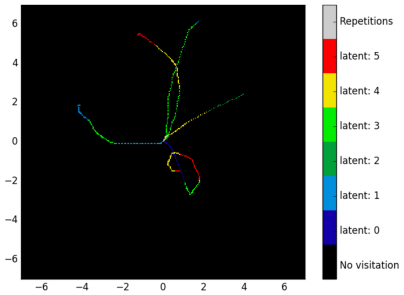


Figure 15: Failure mode of Ant: here 5 rollouts terminate in less than 6 skill switches.