Composing Task-Agnostic Policies with Deep Reinforcement Learning

Anonymous authors
Paper under double-blind review

Abstract

The composition of elementary behaviors to solve challenging transfer learning problems is one of the key elements in building intelligent machines. To date, there has been plenty of work on learning task-specific policies or skills but almost no focus on composing necessary, task-agnostic skills to find a solution to new problems. In this paper, we propose a novel deep reinforcement learning-based skill transfer and composition method that takes the agent's primitive policies to solve unseen tasks. We evaluate our method in difficult cases where training policy through standard reinforcement learning (RL) or even hierarchical RL is either not feasible or exhibits high sample complexity. We show that our method not only transfers skills to new problem settings but also solves the challenging environments requiring both task planning and motion control with high data efficiency.

1 Introduction

Compositionality is the integration of primitive functions into new complex functions that can further be composed into even more complex functions to solve novel problems (Kaelbling & Lozano-Pérez, 2017). Evidence from neuroscience and behavioral biology research shows that humans and animals have the innate ability to transfer their basic skills to new domains and compose them hierarchically into complex behaviors (Rizzolatti et al., 2001). In robotics, the primary focus is on acquiring new behaviors rather than composing and re-using the already acquired skills to solve novel, unseen tasks (Lake et al., 2017).

In this paper, we propose a novel policy ensemble composition method[1] that takes the basic, task-agnostic robot policies, transfers them to new complex problems, and efficiently learns a composite model through standard- or hierarchical-RL (Lillicrap et al., 2015; Schulman et al., 2015; 2017; Haarnoja et al., 2018b; Dayan & Hinton, 1993; Vezhnevets et al., 2017; Florensa et al., 2017; Nachum et al., 2018). Our model has an encoder-decoder architecture. The encoder is a bidirectional recurrent neural network that embeds the given skill set into latent states. The decoder is a feed-forward neural network that takes the given task information and latent encodings of the skills to output the mixture weights for skill set composition. We show that our composition framework can combine the given skills both concurrently (*and* -operation) and sequentially (*or* -operation) as per the need of the given task. We evaluate our method in challenging scenarios including problems with sparse rewards and benchmark it against the state-of-the-art standard- and hierarchical- RL methods. Our results show that the proposed composition framework is able to solve extremely hard RL-problems where standard- and hierarchical-RL methods are sample inefficient and either fail or yield unsatisfactory results.

2 Related Work

In the past, robotics research has been primarily focused on acquiring new skills such as Dynamic Movement Primitives (DMPs) (Schaal et al., 2005) or standard reinforcement learning policies. A lot of research in DMPs revolves around learning compact, parameterized, and modular representations of robot skills (Schaal et al., 2005; Ijspeert et al., 2013; Paraschos et al., 2013; Matsubara et al., 2011). However, there have been quite a few approaches that address the challenge of composing

[1]Supplementary material and videos are available at https://sites.google.com/view/compositional-rl

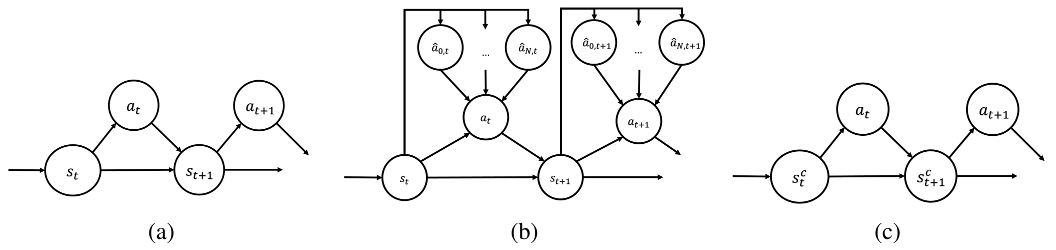


Figure 1: Composition as Markov Decision Process. Fig. (a) represents the graphical model for a simple MDP. Fig. (b) is the augmented graphical model that integrates composition of sub-level policies. Fig. (c) is the new MDP with the augmented state-space.

DMPs in an efficient, scalable manner. To date, DMPs are usually combined through human-defined heuristics, imitation learning or planning (Konidaris et al., 2012; Muelling et al., 2010; Arie et al., 2012; Veeraraghavan & Veloso, 2008; Zoliner et al., 2005). Likewise, RL (Sutton & Barto, 2018) research is also centralized around learning new policies (Lillicrap et al., 2015; Schulman et al., 2015; 2017; Haarnoja et al., 2018b) for complex decision-making tasks by maximizing human-defined rewards or intrinsic motivations (Silver et al., 2016; Qureshi et al., 2017; 2018; Levine et al., 2016).

To the best of the authors' knowledge, there hardly exists approaches that simultaneously combine and transfer past skills into new skills for solving new complicated problems. For instance, Todorov (2009), Haarnoja et al. (2018a) and Sahni et al. (2017) require humans to decompose high-level tasks into intermediate objectives for which either Q-functions or policies are obtained via learning. The high-level task is then solved by merely maximizing the average intermediate Q-functions or combining intermediate policies through temporal-logic. Note that these approaches do not combine task-agnostic skills thus lack generalizability and the ability to transfer skills to the new domains.

Recent advancements lead to Hierarchical RL (HRL) that automatically decomposes the complex tasks into subtasks and sequentially solves them by optimizing the given objective function (Dayan & Hinton, 1993; Vezhnevets et al., 2017; Nachum et al., 2018). In a similar vein, the options framework (Sutton et al., 1999; Precup, 2000) is proposed that solves the given task through temporal abstraction. Recent methods such as option-critic algorithm (Bacon et al., 2017) simultaneously learns a set of sub-level policies (options), their termination functions, and a high-level policy over options to solve the given problem. Despite being an exciting step, the option-critic algorithm is hard to train and requires regularization (Vezhnevets et al., 2016; Harb et al., 2018), or else it ends up discovering options for every time step or a single option for the entire task. In practice, HRL methods tend to exhibit high sample complexity and therefore, require a huge number of interactions with the real environment. Furthermore, the sub-level options or objectives obtained via HRL are inherently task-specific and therefore cannot be transferred to new domains.

3 Background

We consider a standard RL formulation (Fig. 1 (a)) based on Markov Decision Process (MDP) defined by a tuple $\{\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}\}$, where $\mathcal{S}$ and $\mathcal{A}$ represent the state and action space, $\mathcal{P}$ is the set of transition probabilities, and $\mathcal{R}$ denotes the reward function. At time $t \geq 0$, the agent observes a state $s_t \in \mathcal{S}$ and performs an action $a_t \in \mathcal{A}$. The agent's action a_t transitions the environment state from $s_t \in \mathcal{S}$ to $s_{t+1} \in \mathcal{S}$ with respect to the transition probability $\mathcal{P}(s_{t+1}|s_t, a_t)$ and leads to a reward $r_t \in \mathcal{R}$.

For compositionality, we extend the standard RL framework by assuming that the agent has access to the finite set of primitive policies $\Pi = \{\pi_i\}_{i=0}^N$ that could correspond to agent's skills, controller, or motor-primitives. Our composition model is agnostic to the structure of primitive policy functions, but for the sake of this work, we assume that each of the sub-policies $\{\pi_i\}_{i=0}^N$ solves the MDP defined by a tuple $\{\hat{\mathcal{S}}, \mathcal{A}, \hat{\mathcal{P}}, \hat{\mathcal{R}}^i\}$. Therefore, $\hat{\mathcal{S}}$, $\hat{\mathcal{P}}$ and $\hat{\mathcal{R}}^i$ are the state-space, transition probabilities and rewards of the primitive policy π_i, respectively. Each of the primitive

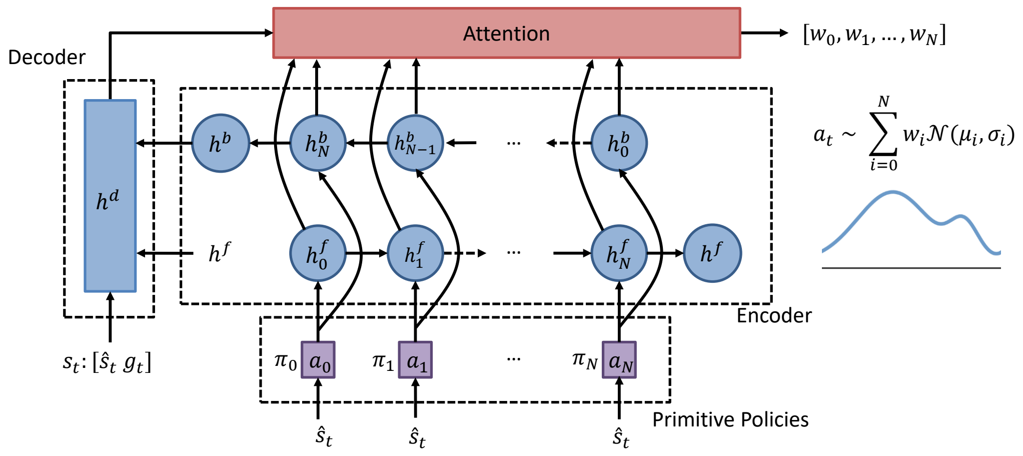


Figure 2: Policy ensemble composition model that takes the state information s_t and a set of primitive policies' output $\{\hat{a}_i\}_{i=0}^{N}$ to compute a composite action a_t.

policies $\pi_i : \hat{S} \times \mathcal{A} \to [0,1], \forall i \in [0,1,\cdots,N]$, takes a state $\hat{s} \in \hat{\mathcal{S}}$ and outputs a distribution over the agent's action space $\mathcal{A}$. We define our composition model as a composite policy $\pi_\theta^c : \mathcal{S} \times \mathcal{A}^{N+1} \times \mathcal{A} \to [0,1]$, parameterize by θ, that outputs a distribution over the action space conditioned on the environment's current state $s \in \mathcal{S}$ and the primitive policies $\{\hat{a}_i \in \mathcal{A}\}_{i=0}^{N} \sim \Pi$. The state space of the composite model is $\mathcal{S} = [\hat{\mathcal{S}}, \mathcal{G}]$. The space $\mathcal{G}$ could include any task specific information such as target locations. Hence, in our framework, the state inputs to the primitive policies Π and composite policy π_θ^c need not to be the same.

In remainder of this section, we show that our composition model solves an MDP problem. To avoid clutter, we assume that both primitive policy ensemble and composite policy have the same state space $\mathcal{S}$, i.e., $\mathcal{G} = \emptyset$. The composition model samples an action from a distribution parameterized by the actions of sub-level policies and the state $s \in \mathcal{S}$ of the environment. We can augment the naive graphical model in Fig. 1 (a) to incorporate the outputs of sub-policies to determine the composite actions, as shown in Fig. 1 (b). It can be seen that by defining a new state space $\mathcal{S}^c$ as $\mathcal{A}^{N+1} \times \mathcal{S}$, where $\mathcal{A}^{N+1} : \{\mathcal{A}^i\}_{i=0}^{N}$ are the outputs of sub-level policies, we can construct a new MDP, as shown in Fig. 1 (c), to represent our composite model. This new MDP is defined as $\{\mathcal{S}^c, \mathcal{A}, \mathcal{P}^c, \mathcal{R}\}$ where $\mathcal{S}^c = \mathcal{A}^N \times \mathcal{S}$ is the new composite state-space, $\mathcal{A}$ is the action-space, $\mathcal{P}^c : \mathcal{S}^c \times \mathcal{S}^c \times \mathcal{A} \to [0,1]$ is the transition probability function, and $\mathcal{R}$ is the reward function for the given task.

4 Policy Ensemble Composition

In this section, we present our policy ensemble composition framework, shown in Fig. 2. Our composition model consists of i) the encoder network that takes the outputs of primitive policies and embeds them into latent spaces; ii) the decoder network that takes current state s_t of the environment and the latent embeddings from the encoder network to parameterize the attention network; iii) the attention network that outputs the probability distribution over the primitive low-level policies representing their mixture weights. The remainder of the section explains the individual models of our composition framework and the overall training procedure.

4.1 Encoder Network

Our encoder is a bidirectional recurrent neural network (BRNN) that consists of Long Short-Term Memory units (Hochreiter & Schmidhuber, 1997). The encoder takes the outputs of the policy ensemble $\{\hat{a}_i\}_{i=0}^{N}$ and transform them into latent states of forward and backward RNN, denoted as $\{h_i^f\}_{i=0}^{N+1}$ and $\{h_i^b\}_{i=0}^{N+1}$, respectively, where $h_i^f, h_i^b \in \mathbb{R}^d; \forall i \in [0,1,\cdots,N+1]$. The $N+1$ states of forward and backward RNN corresponds to their last hidden states denoted as h^f and h^b, respectively, in Fig. 2.

4.2 Decoder Network

Our decoder is a simple feed-forward neural network that takes the last hidden states of the forward and backward encoder network, i.e., $\{h^f, h^b\}$, and the current state of the environment s to map them into a latent space $h \in \mathbb{R}^d$. The state input to the decoder network is defined as $s : [\hat{s}, g]$, where $\hat{s} \in \hat{\mathcal{S}}$ is the state input to the low-level policy ensemble and $g \in \mathcal{G}$ could be any additional information related to the given task, e.g., goal position of the target to be reached by the agent.

4.3 Attention Network

The composition weights (see Fig. 2) $\{w_i \in [0, 1]\}_{i=0}^N$ are determined by the attention network as follows:

$$q_i = W^T \cdot \tanh(W_f \cdot h_i^f + W_b \cdot h_i^b + W_d \cdot h); \forall i \in [0, N] \tag{1}$$

where $W_f, W_b, W_d \in \mathbb{R}^{d\times d}$ and $W \in \mathbb{R}^d$. The weights $\{w_i\}_{i=0}^N$ for the composite policy are computed using gumbel-softmax denoted as $\mathrm{softmax(q/T)}$, where T is the temperature term (Jang et al., 2016).

4.4 Composite policy

Given the primitive policy ensemble $\Pi = \{\pi_i\}_{i=0}^N$, the composite action is the weighted sum of all primitive policies outputs, i.e., $\pi_\theta^c = \sum_i^N w_i \pi_i$. Since, we consider the primitive policies to be Gaussian distributions, the output of each primitive policy is parameterized by mean μ and variance σ, i.e., $\{\hat{a}_i \sim \mathcal{N}(\mu_i, \sigma_i)\}_{i=0}^N \leftarrow \{\pi_i\}_{i=0}^N$. Hence, the composite policy can be represented as $\pi_\theta^c = \sum_i^N w_i \mathcal{N}(\mu_i, \sigma_i)$, where $\mathcal{N}(\cdot)$ denotes Gaussian distribution, and $\sum_i w_i = 1$. Given the mixture weights, other types of primitive policies, such as DMPs (Schaal et al., 2005), can also be composed together by the weighted combination of their normalized outputs.

4.5 Composite model training objective

The general objective of RL methods is to maximize the cumulative expected reward, i.e., $J(\pi_\theta^c) = \mathbb{E}_{\pi_\theta^c}[\sum_{t=0}^\infty \gamma^t r_t]$, where $\gamma : (0, 1]$ is a discount factor. We consider the policy gradient methods to update the parameters θ of our composite model, i.e., $\theta \leftarrow \theta + \eta \bigtriangledown_\theta J(\pi_\theta^c)$, where η is the learning rate. We show that our composite policy can be trained through standard RL and HRL methods, described as follow.

4.5.1 Standard Reinforcement learning

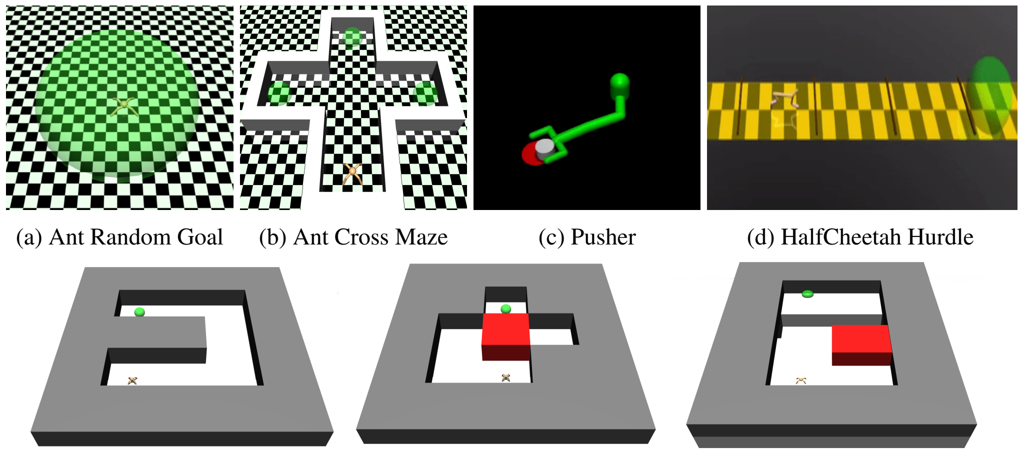
In standard RL, the policy gradients are determined by either on-policy or off-policy updates (Lillicrap et al., 2015; Schulman et al., 2015; 2017; Haarnoja et al., 2018b) and any of them could be used to train our composite model. However, in this paper, we consider off-policy soft-actor critic (SAC) method (Haarnoja et al., 2018b) for the training of our policy function. SAC maximizes the expected entropy $\mathcal{H}(\cdot)$ in addition to the expected reward, i.e.,

$$J(\pi_\theta^c) = \sum_{t=0}^{T} \mathbb{E}_{\pi_\theta^c}[r(s_t, a_t) + \lambda \mathcal{H}(\pi_\theta^c(\cdot|s_t))] \tag{2}$$

where λ is a hyperparameter. We use SAC as it motivates exploration and has been shown to capture the underlying multiple modes of an optimal behavior. Since there is no direct method to estimate a low-variance gradient of Eq (2), we use off-policy value function-based optimization algorithm (for details refer to Appendix A.1 of supplementary material).

4.5.2 Hierarchical Reinforcement Learning

In HRL, there are currently two streams - task decomposition through sub-goals (Nachum et al., 2018) and option framework (Bacon et al., 2017) that learns temporal abstractions. In the options framework, the options can be composite policies that are acquired with their termination functions. In task decomposition methods that generate sub-goal through high-level policy, the low-level policy can be replaced with our composite policy. In our work, we use the latter approach (Nachum et al., 2018), known as HIRO algorithm, to train our policy function.



(a) Ant Random Goal (b) Ant Cross Maze (c) Pusher (d) HalfCheetah Hurdle

(e) Ant Maze (f) Ant Push (g) Ant Fall

Figure 3: Benchmark control and manipulation tasks requiring an agent to reach or move the object to the given targets (shown in red for pusher and green for rest).

Like, standard HIRO, we use two level policy structure. At each time step t, the high-level policy $\pi^{hi}_{\theta'}$, with parameters θ', observes a state s_t and takes an action by generating a goal $g_t \in \mathcal{S}$ in the state-space $\mathcal{S}$ for the composite low-level policy $\pi^{c:low}_{\theta}$ to achieve. The $\pi^{c:low}_{\theta}$ takes the state s_t, the goal g_t, and the primitive actions $\{\hat{a}_i\}_0^N$ to predict a composite action a_t through which an agent interacts with the environment. The high-level policy is trained to maximize the expected task rewards given by the environment whereas the composite low-level policy is trained to maximize the expected intrinsic reward defined as the negative of distance between current and goal states, i.e., $\|s_t + g_t - s_{t+1}\|^2$. To conform with HIRO settings, we perform off-policy correction of the high-level policy experiences and we train both high- and low-level policies via TD3 algorithm (Fujimoto et al., 2018) (for details refer to Appendix A.2 of supplementary material).

5 Experiments and Results

We evaluate and compare our method against standard RL, and HRL approaches in challenging environments (shown in Fig. 3) that requires complex task planning and motion control. The implementation details of all presented methods and environment settings are provided in Appendix B of supplementary material. We also do an ablative study in which we take away different components of our composite model to highlight their importance. Furthermore, we depict attention weights of our model in a navigation task to highlight its ability of concurrent and sequential composition.

We consider the following seven environments for our analysis: (1) **Pusher:** A simple manipulator has to push an object to a given target location. (2) **Ant Random Goal:** In this environment, a quadruped-Ant is trained to reach the randomly sampled goal location in the confined circular region. (3) **Ant Cross Maze:** The cross-maze contains three target locations. The task for a quadruped Ant is to reach any of the three given targets by navigating through a 3D maze without collision. (4) **HalfCheetah Hurdle:** In this problem, the task for a halfcheetah is to run and jump over the three barriers to reach the given target location. (5) **Ant Maze:** A $\supset$-shaped maze poses a challenging navigation task for a quadruped-Ant. In this task, the agent is given random targets all along the maze to reach while training. However, during the evaluation, we test the agent for reaching the farthest end of the maze. (6) **Ant Push:** A challenging environment that requires both task and motion planning. The environment contains a movable block, and the goal region is located behind that block. The task for an agent is to reach the target by first moving to the left of the maze so that it can move up and right to push the block out of the way for reaching the target. (7) **Ant Fall:** A navigation task where the target is located across the rift in front of the agent's initial position. There also happen to be a moveable block, so the agent has to move to the right, push the block forward, fill the gap, walk across, and move to the left to reach the target location. (8) **Multi-goal Point Mass:** In this scenario, the task is to navigate a point-mass to one of the four goals located diagonally to agent initial position.

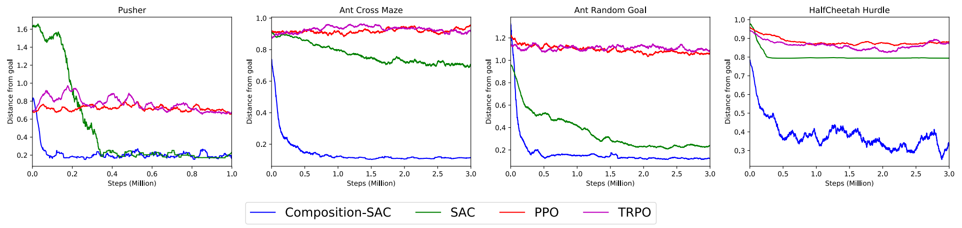


Figure 4: Comparison results of our method against several standard RL methods averaged over ten trials in a set of difficult tasks. The vertical and horizontal axis represents the distance of the agent/object from the target and environment steps in millions, respectively. Note that our composition framework learns to solve the task with high samples efficiency, whereas other benchmark methods either fail or perform poorly.

In all tasks, we also acquire primitive skills of the agent for our composite policy. For Ant, we use four basic policies for moving left, right, up, and down. The pusher uses two primitive policies that are to push an object to the left and down. In HalfCheetah hurdle environment, the low-level policies include jumping and running forward. Finally fot the point-mass robot, the composition model takes four policies for moving in the up, down, left and right directions. Furthermore, in all environments, except pusher, the primitive policies were agnostic of high-level tasks (such as target locations) that were therefore provided separately to our composite model via decoder network. This highlights the ability of our model to transfer basic robot skills to novel problems.

Methods	Environments			
	Ant Random Goal	Ant Cross Maze	Pusher	HalfCheetah Hurdle
SAC	0.21 ± 0.08	0.78 ± 0.06	0.17 ± 0.02	0.79 ± 0.01
TRPO	1.09 ± 0.15	0.85 ± 0.15	0.64 ± 0.09	0.87 ± 0.05
PPO	1.06 ± 0.11	0.95 ± 0.07	0.71 ± 0.06	0.88 ± 0.04
Our Method	$\mathbf{0.11 \pm 0.05}$	$\mathbf{0.11 \pm 0.02}$	$\mathbf{0.14 \pm 0.02}$	$\mathbf{0.27 \pm 0.22}$

Table 1: Performance comparison of our model against SAC (Haarnoja et al., 2018b), TRPO (Schulman et al., 2015), and PPO (Schulman et al., 2017) on benchmark control tasks in terms of distance (lower the better) of an agent from the given target. The mean final distances with standard deviations over ten trials are reported. We also normalize the reported values by the agent initial distance from the goal so values close to 1 or higher show failure. It can be seen that our method (shown in bold) accomplishes the tasks by reaching goals whereas other methods fail except for SAC in simple Pusher and Ant Random Goal environments.

5.1 Comparative study

In our comparative studies, we divide our test environments into two groups. The first group includes Pusher, Random Goal Ant, Ant Cross Maze, and HalfCheetah-Hurdle environments, whereas the second group comprises the remaining environments that require task and motion planning under weak reward signals.

In the first group of settings, we compare our composite model trained with SAC (Haarnoja et al., 2018b) against the standard Gaussian policies obtained using SAC (Haarnoja et al., 2018b), PPO (Schulman et al., 2017), and TRPO (Schulman et al., 2015). We exclude HRL methods in these cases for two reasons. First, the environment rewards sufficiently represent the underlying task, whereas HRL approaches are applicable in cases that have a weak reward signal or require task and motion planning. Second, HRL methods usually need a large number of training steps generally much more than tradition RL methods. Table 1 presents the mean and standard deviation of the agent's final distance from the given targets after the end of an evaluation rollout over the ten trials. Fig. 4 shows the mean learning performance over all trials during the three million training steps. In these set of problems, TRPO and PPO entirely fail to reach the goal, and SAC performs reasonably well but only in simple Ant Random Goal and Pusher environments as it fails in other cases. Our composite policy obtained using SAC successfully solves all tasks and exhibit high data-efficiency by learning in merely a few thousand training steps.

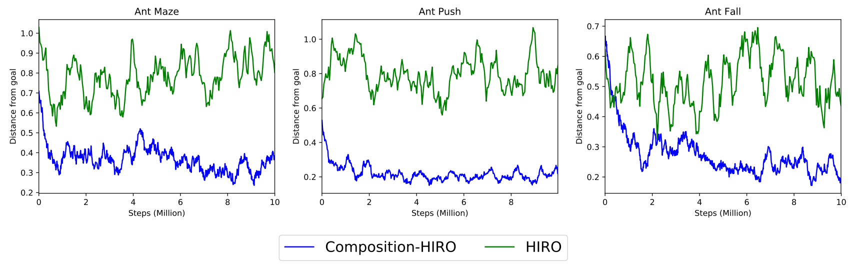


Figure 5: Performance comparison of our composition model trained with HIRO against standard HIRO in three challenging environments with a standard Ant of 150 units torque limit. We report mean and standard error, over ten trials, of agent final distances from the given given goals, normalized by their initial distance, over 10 million steps.

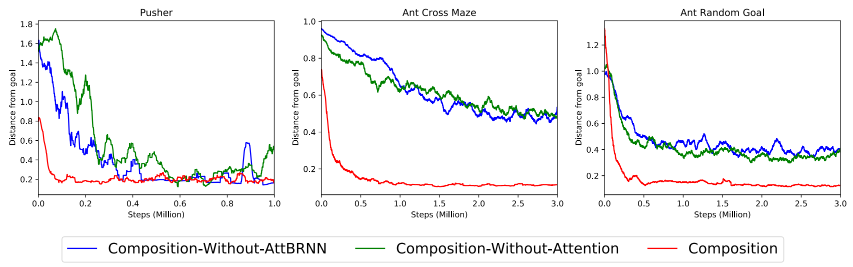


Figure 6: Ablative Study: Performance comparison, averaged over ten trials, of our composite model against its ablated variations that lack attention model or both attention and bidirectional-RNN (AttBRNN) in three different environments.

In our second group of environments, we use distance-based rewards that are weak signals as greedily following them does not lead to solving the problem. Furthermore, in these environments, policies trained with standard RL, including our composite policy, failed to solve the problem even after 20 million training steps. Therefore, we trained our composite policy with HIRO (Nachum et al., 2018) and compared its performance against standard HIRO formulation (Nachum et al., 2018). We also tried to include option-critic framework (Bacon et al., 2017), but we were unable to get any considerable results with their online implementation despite several attempts with the parameter tuning. One of the reasons option-critic fails is because it relies purely on task rewards to learn, which makes them inapplicable for cases with weak reward signals (Nachum et al., 2018). Furthermore, unlike HIRO that used a modified Ant with 30 units joint torque limit, we use Mujoco standard Ant that has a torque limit of 150 units and makes the learning even harder as the Ant is now more prone to instability.

Fig. 5 shows the learning performance, averaged over ten trials, during 10 million steps. In these problems, the composite policy with HIRO outperforms standard HIRO (Nachum et al., 2018) by a significant margin that certifies the utility of solving RL tasks using composition by leveraging basic pre-acquired skills. HIRO performs poorly with standard Ant as it imposes a harder control problem since the agent should also learn to balance the Ant to prevent it from flipping over due to high torques. We were able to replicate the results of HIRO (Nachum et al., 2018) on their modified Ant (Torque Limit 30) and also, our composition model gave comparably better results on modified Ant than standard Ant. However, we use a standard-Ant to conform among all Ant environments presented in this paper. In the Ant Fall environment, composition model struggles to perform well which we believe is because the low-level policies were trained in a 2D planner space rather than a 3D space with an elevation that slightly changes the underlying state-space.

5.2 Ablative study

We remove attention-network, and both attention-network and BRNN (AttBRNN) from our composition model to highlight their importance in the proposed architecture in solving complex problems. We train all models with SAC (Haarnoja et al., 2018b). The first model is our composite policy without attention in which the decoder network takes the state information and last hidden states of the encoder (BRNN) to directly output actions rather than mixture weights. The second model is without attention network and BRNN; it is a feed-forward neural network that takes the state information and the primitive actions and predicts the action to interact with the environment. Fig. 6 shows the mean performance comparison, over ten trials, of our composite model against its ablated versions on a Ant Random Goal, Cross Maze Ant, and Pusher environment. We exclude remaining test environments in this study as ablated models completely failed to perform or show any progress. Note that the better performance of our method compared to ablated versions highlight the merits of our architecture design. Intuitively, BRNN allows the dynamic composition of a skill set of variable lengths and the attention network bypasses the complex transformation of action embeddings (model-without-attention) or actions and state-information (AttBRNN model) directly to action space.

5.3 Depiction of attention weights

In order to further assess the merit of utilizing an attention network, we apply our model to a simple 2D multi-goal point-mass environment as shown in Fig. 7. The point-mass is initialized around the origin (with variance $\sigma^2 = 0.1$) and must randomly choose one of four goals to reach. For this experiment we use dense rewards with both a positional and actuation cost. Primitive policies of *up* $(+y)$, *down* $(-y)$, *left* $(-x)$, and *right* $(+x)$ were trained and composed to reach goals, represented here as red dots, in the "diagonal" directions where a combination of two or more primitive policies are required to reach each goal.

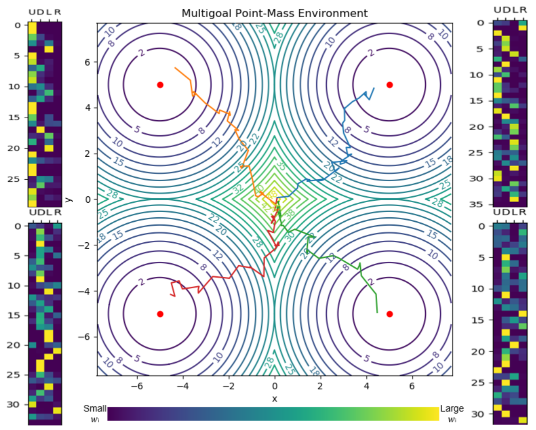


Figure 7: Each path corresponds to its adjacent attention weight mapping. The weighting "strength" of each primitive policy is depicted for each step (i.e. up (U), down (D), left (L), and right (R)). Each path begins at the origin and ends when the point-mass is within one unit of a goal. The plot contours represent the position cost.

The four mappings in the figure give us insight into how the attention network is utilizing the given primitives to achieve the desired task. At each step in a given path, the weights $\{w_i\}_{i=0}^N$ for each low-level policy are assigned and composed together to move the point-mass in the desired direction. We see here that even with some noise and short-term error, the attention weights are strongest for primitive policies that move the point-mass to its chosen goal. We also see that multiple policies are activated at once to achieve more direct movements toward the goal, as opposed to "stair-stepping" where only one primitive is activated at a time. Both of these observations point to the concurrent and sequential nature of this composition model.

6 Conclusions and Future work

We present a novel policy ensemble composition method that combines a set of independent and task-agnostic primitive policies through reinforcement learning to solve the given tasks. We show that our method can transfer the given skills to novel problems and can compose them both sequentially (*or* -operation) and concurrently (*and* -operation) to find a solution for the task in hand. Our experiments highlight that composition is vital for solving problems requiring complex motion skills and decision-making where standard reinforcement learning and hierarchical reinforcement learning methods either fail or need a massive number of interactive experiences to achieve the desired results.

In our future work, we plan to extend our method to automatically acquire the missing skills in the given skillset that are necessary to solve the specific problems. We also aim towards a system that learns the hierarchies of composition models by combining primitive policies into complex policies that would further be composed together for a combinatorial outburst in the agent's skillset.

REFERENCES

Hiroaki Arie, Takafumi Arakaki, Shigeki Sugano, and Jun Tani. Imitating others by composition of primitive actions: A neuro-dynamic model. *Robotics and Autonomous Systems*, 60(5):729–741, 2012.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *Advances in neural information processing systems*, pp. 271–278, 1993.

Carlos Florensa, Yan Duan, and Pieter Abbeel. Stochastic neural networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1704.03012*, 2017.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

Tuomas Haarnoja, Vitchyr Pong, Aurick Zhou, Murtaza Dalal, Pieter Abbeel, and Sergey Levine. Composable deep reinforcement learning for robotic manipulation. *arXiv preprint arXiv:1803.06773*, 2018a.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018b.

Jean Harb, Pierre-Luc Bacon, Martin Klissarov, and Doina Precup. When waiting is not an option: Learning options with a deliberation cost. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2): 328–373, 2013.

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

Leslie Pack Kaelbling and Tomás Lozano-Pérez. Learning composable models of parameterized skills. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pp. 886–893. IEEE, 2017.

George Konidaris, Scott Kuindersma, Roderic Grupen, and Andrew Barto. Robot learning from demonstration by constructing skill trees. *The International Journal of Robotics Research*, 31(3): 360–375, 2012.

Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40, 2017.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 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.

Takamitsu Matsubara, Sang-Ho Hyon, and Jun Morimoto. Learning parametric dynamic movement primitives from multiple demonstrations. *Neural networks*, 24(5):493–500, 2011.

Katharina Muelling, Jens Kober, and Jan Peters. Learning table tennis with a mixture of motor primitives. In *Humanoid Robots (Humanoids), 2010 10th IEEE-RAS International Conference on*, pp. 411–416. IEEE, 2010.

Ofir Nachum, Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. *arXiv preprint arXiv:1805.08296*, 2018.

Alexandros Paraschos, Christian Daniel, Jan R Peters, and Gerhard Neumann. Probabilistic movement primitives. In *Advances in neural information processing systems*, pp. 2616–2624, 2013.

Doina Precup. *Temporal abstraction in reinforcement learning*. University of Massachusetts Amherst, 2000.

Ahmed Hussain Qureshi, Yutaka Nakamura, Yuichiro Yoshikawa, and Hiroshi Ishiguro. Show, attend and interact: Perceivable human-robot social interaction through neural attention q-network. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1639–1645. IEEE, 2017.

Ahmed Hussain Qureshi, Yutaka Nakamura, Yuichiro Yoshikawa, and Hiroshi Ishiguro. Intrinsically motivated reinforcement learning for human–robot interaction in the real-world. *Neural Networks*, 2018.

Giacomo Rizzolatti, Leonardo Fogassi, and Vittorio Gallese. Neurophysiological mechanisms underlying the understanding and imitation of action. *Nature reviews neuroscience*, 2(9):661, 2001.

Himanshu Sahni, Saurabh Kumar, Farhan Tejani, and Charles Isbell. Learning to compose skills. *arXiv preprint arXiv:1711.11289*, 2017.

Stefan Schaal, Jan Peters, Jun Nakanishi, and Auke Ijspeert. Learning movement primitives. In *Robotics research. the eleventh international symposium*, pp. 561–572. Springer, 2005.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pp. 1889–1897, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

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, 2016.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

Minija Tamosiunaite, Bojan Nemec, Aleš Ude, and Florentin Wörgötter. Learning to pour with a robot arm combining goal and shape learning for dynamic movement primitives. *Robotics and Autonomous Systems*, 59(11):910–922, 2011.

Emanuel Todorov. Compositionality of optimal control laws. In *Advances in Neural Information Processing Systems*, pp. 1856–1864, 2009.

Harini Veeraraghavan and Manuela Veloso. Teaching sequential tasks with repetition through demonstration. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 3*, pp. 1357–1360. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

Alexander Vezhnevets, Volodymyr Mnih, Simon Osindero, Alex Graves, Oriol Vinyals, John Agapiou, et al. Strategic attentive writer for learning macro-actions. In *Advances in neural information processing systems*, pp. 3486–3494, 2016.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1703.01161*, 2017.

R Zoliner, Michael Pardowitz, Steffen Knoop, and Rüdiger Dillmann. Towards cognitive robots: Building hierarchical task representations of manipulations from human demonstration. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pp. 1535–1540. IEEE, 2005.

Algorithm 1: Composition model training using SAC

```
Initialize parameter vectors $\phi$, $\phi'$, $\theta$, $\xi$
Input: Primitive policies $\Pi = \{\pi_i\}_{i=0}^N$
for each iteration do
    for each environment step do
        Compute primitive policies state $\hat{s}_t \leftarrow s_t \backslash g_t$
        Sample primitive actions $\{\hat{a}_{i,t}\}_{i=0}^N \sim \Pi(\hat{s}_t)$
        Sample composite action $a_t \sim \pi_\theta^c(a_t|s_t, \{\hat{a}_{i,t}\}_{i=0}^N)$
        Sample next state $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
        $\mathcal{M} \leftarrow \mathcal{M} \cup \{(s_t, a_t, \{\hat{a}_{i,t}\}_{i=0}^N, r_t, s_{t+1})\}$
    for each gradient step do
        Update value function $\phi \leftarrow \phi - \eta \bigtriangledown_\phi J_V(\phi)$
        Update Q-function $\xi \leftarrow \xi - \eta \bigtriangledown_\xi J_Q(\xi)$
        Update policy $\theta \leftarrow \theta - \eta \bigtriangledown_\theta J_{\pi^c}(\theta)$
        $\phi' \leftarrow \tau\phi + (1-\tau)\phi'$
```

A Composite Model Training Algorithms

A.1 Training with Soft Actor-Critic

In this section, we briefly describe the procedure to train our composition model using SAC (Haarnoja et al., 2018b). Although any RL method can be used to optimize our model, we use SAC as it is reported to perform better than other training methods. Our composite policy is a tractable function $\pi_\theta^c(a_t|s_t, \{\pi_i\}_{i=0}^N)$ parameterized by θ. The composite policy update through SAC requires the approximation of Q- and value-functions. The parametrized value- and Q-function are denoted as $V_\phi(s_t)$ with parameters ϕ, and $Q_\xi(s_t, a_t)$ with parameters ξ, respectively. Since, SAC algorithm build on the soft-policy iteration, the soft value-function $V_\phi(s_t)$ and soft Q-function $Q_\xi(s_t, a_t)$ are learned by minimizing the squared residual error $J_V(\phi)$ and squared Bellman error $J_Q(\xi)$, respectively, i.e.,

$$J_V(\phi) = \mathbb{E}_{s_t \sim \mathcal{M}}[\frac{1}{2}(V_\phi(s_t) - \hat{V}(s_t))^2] \tag{3}$$

$$J_Q(\xi) = \mathbb{E}_{(s_t,a_t) \sim \mathcal{M}}[\frac{1}{2}(Q_\xi(s_t, a_t) - \hat{Q}(s_t, a_t))^2] \tag{4}$$

where $\mathcal{M}$ is a replay buffer, $\hat{V}(s_t) = \mathbb{E}_{a_t \sim \pi_\theta^c}[Q_\xi(s_t, a_t) - \log \pi_\theta^c(a_t|s_t)]$ and $\hat{Q}$ is the Bellman target computed as follows:

$$\hat{Q}(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim p}[V_{\phi'}(s_t + 1)] \tag{5}$$

The function $V_{\phi'}(s_t)$ is the target value function with parameters ϕ'. The parameters ϕ' are the moving average of the parameters ϕ computed as $\tau\phi + (1-\tau)\phi'$, where τ is the smoothing coefficient. Finally the policy parameters are updated by minimizing the following expected KL-divergence.

$$J_{\pi^c}(\theta) = \mathbb{E}_{s_t \sim \mathcal{M}}\left[D_{KL}\left(\pi_\theta^c(\cdot|s_t) || \frac{\exp(Q_\xi(s_t, \cdot))}{Z_\xi(s_t)}\right)\right] \tag{6}$$

where Z_ξ is a partition function that normalizes the distribution. Since, just-like SAC, our Q-function is differentiable, the above cost function can be determined through a simple reparametization trick, see Haarnoja et al. (2018b) for details. Like SAC, we also maintain two Q-functions that are trained independently, and we use the minimum of two Q-functions to compute Eqn. 3 and Eqn. 6. This way of using two Q-function has been shown to alleviate the positive biasness problem in the policy improvement step. The overall training procedure is summarized in Algorithm 1.

A.2 Training with HIRO

In this section, we outline the algorithm to train composite policy through HIRO that employs the two level policy structure. The high-level policy generates the sub-goals for the low-level composite

Algorithm 2: Composition model training using HIRO

```
Initialize parameter vectors $\phi$, $\phi'$, $\theta$, $\xi$
Input: Primitive policies $\Pi = \{\pi_i\}_{i=0}^N$
for each iteration do
    for each environment step do
        Compute primitive policies state $\hat{s}_t \leftarrow s_t \backslash g_t$
        Sample primitive actions $\{\hat{a}_{i,t}\}_{i=0}^N \sim \Pi(\hat{s}_t)$
        Sample high-level action $g_t \sim \pi^{hi}(\hat{s}_t)$
        Sample composite action $a_t \sim \pi_\theta^c(a_t|s_t, g_t, \{\hat{a}_{i,t}\}_{i=0}^N)$
        Sample next state $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
        $\mathcal{M} \leftarrow \mathcal{M} \cup \{(s_t, g_t, a_t, \{\hat{a}_{i,t}\}_{i=0}^N, r_t, s_{t+1})\}$
    for each gradient step do
        Sample mini-batch with c-step transitions
          $\{(g^k, s_{t:t+c}^k, a_{t:t+c-1}^k, \{\hat{a}_{j,t:t+c-1}^k\}_{j=0}^N, r_{t:t+c-1}^k)\}_{k=1}^B \sim \mathcal{M}$
        Compute rewards for low-level policy $\{r_i^{lo}\}_{i=0}^B \leftarrow \{r^{lo}(s_i, g_i, s_{i+1})\}_{i=0}^B$
        Update $\pi^{c:lo}$ w.r.t $Q^{lo}$ using $\{(s_k, g_k, a_k, \{\hat{a}_{i,k}\}_{i=0}^N, r_k^{lo}, s_{k+1})\}_{k=0}^{B-1}$ (Nachum et al.,
          2018)
        Update $\pi^{hi}$ w.r.t $Q^{hi}$ using $\{(s_k, \hat{g}_k, \sum_{i=k}^{c-1} r_k, s_{k+c})\}_{k=0}^{B-1}$ (Nachum et al., 2018)
```

policy to achieve. The low-level composite policy also have access to the primitive policy actions. Like HIRO, we use TD3 algorithm (Fujimoto et al., 2018) to train both high-level and low-level policies with their corresponding Q-functions, Q^{hi} and Q^{lo}, respectively. The low-level policy $\pi_\theta^{c:low}$, with parameters θ, is trained to maximize the Q-values from the low-level Q-function Q^{lo} for the given state-goal pairs. The Q-function (Q^{lo}) parameters ξ are optimized by minimizing temporal-difference error for the given transitions, i.e.,

$$J_Q^{lo}(\xi) = \left(r^{lo}(s_t, g_t, s_{t+1}) + \gamma Q_\xi^{lo}(s_{t+1}, g_{t+1}, \pi_\theta^{c:lo}(s_{t+1}, g_{t+1}, \{\hat{a}\}_0^N)) - Q_\xi^{lo}(s_t, g_t, a_t)\right)^2 \quad (7)$$

where $r^{lo}(s_t, g_t, s_{t+1}) = -\|s_t + g_t - s_{t+1}\|$ and $g_{t+1} \sim \pi_{\theta'}^{hi}(s_{t+1})$.

The high-level policy $\pi_{\theta'}^{hi}$, with parameters θ', is trained to maximize the values of Q^{hi}. The Q-function (Q^{hi}) parameters ξ' are trained through minimizing the following loss for the given transitions.

$$J_Q^{hi}(\xi') = \Big(\sum_{t=0}^{c-1} R_t(s_t, a_t, s_{t+1}) + \gamma Q_{\xi'}^{hi}(s_{t+c}, \pi_{\theta'}^{hi}(s_{t+c})) - Q_\xi^{hi}(s_t, \hat{g}_t)\Big)^2 \quad (8)$$

During training, the continuous adaptation of low-level policy poses a non-stationery problem for the high-level policy. To mitigate the changing behavior of low-level policy, Nachum et al. (2018) introduced off-policy correction of the high-level actions. During correction, the high-level policy action g is usually re-labeled with $\hat{g}$ that would induce the same low-level policy behavior as was previously induced by the original high-level action g (for details, refer to (Nachum et al., 2018)). Algorithm 2 presents the procedure to train our composite policy with HIRO.

B Implementation details

B.1 Environment Details

In this section, we present the environment details including reward functions, primitive policies, and state space information. The reward functions are presented in the Table 3 together with the overall reward scaling values.

B.1.1 Ant environments

In these environments, we use 8 DOF four-legged Ant with 150 units torque limit. The primitive policies of moving left, right, down and up were shared across all these tasks. In these environments, the information g in the state $s : [\hat{s}, g]$ corresponds to the target location. Let us introduce the notation to defined reward function. Let r_{xy}, g_{xy}, u, and f_c denote xy-position of the robot's torso, xy-position of the goal, joint torques, and contact-cost, respectively. The scaling factors are defined as λ. The reward function for the following environments is defined as with reward scaling of 5 units:

$$-\lambda_g||r_{xy} - g_{xy}||^2 + \lambda_v v_{xy} + \lambda_s I(\text{IsAlive}) - \lambda_{ct}||u||^2 - \lambda_c f_c \tag{9}$$

Ant Random Goal: In this environment, the ant has to navigate to any randomly sampled target within the confined circular region of radius 5 units. The goal radius is defined to be 0.25 units. The reward function coefficients λ_g, λ_v, λ_s, λ_{ct}, and λ_c are 0.3, 0.0, 0.05, 0.01, and 0.001, respectively.

Ant Cross Maze: In this environment, the ant has to navigate through the 3D maze to reach any of the target sampled from the three targets. The goal radius is defined to be 1.0 units. The reward function parameters are same as for the random-goal ant environment.

For the remaining environment (Ant Maze, Ant Push and Ant Fall), we use the following reward function with no reward scaling:

$$\lambda_g||r_{xyz} - g_{xyz}||^2 - \lambda_{ct}||u||^2 - \lambda_c f_c \tag{10}$$

where coefficients λ_g, λ_{ct}, and λ_c are set to be 1.0, 0.05, and 0.5×10^{-4}.

Ant Maze: In this environment, we place the Ant in a $\supset$-shaped maze for a navigation task between given start and goal configurations. The goal radius is defined to be 5 units. During training, the goal is uniformly sampled from $[-4, 20] \times [-4, 20]$ space, and the Ant initial location is always fixed at $(0, 0)$. During testing, the agent is evaluated to reach the farthest end of the maze located at $(0, 19)$ within L2 distance of 5.

Ant Push: In this environment, the Ant is initially located at $(0, 0)$ coordinate, the moveable block is at $(0, 8)$, and the goal is at $(0, 19)$. The agent is trained to reach randomly sampled targets whereas during testing, we evaluate the agent to reach the goal at $(0, 19)$ within L2 distance of 5.

Ant Fall: In this environment, the Ant has to navigate in a 3D maze. The initial agent location is $(0, 0)$, and a movable block is at $(8, 8)$ at the same elevation as Ant. Their is a rift in the region $[-4, 12] \times [12, 20]$. To reach the target on the other side of the rift, the Ant must push the block down into the rift, and then step on it to get to the goal position.

Parameters	SAC	HIRO	TRPO	PPO
Learning rate (η)	3×10^{-4}	1×10^{-4}	-	-
Discount factor (γ)	0.99	0.99	0.99	0.99
Nonlinearity in feedforward networks	ReLU	ReLU	ReLU	ReLU
Minibatch samples size	256	128	-	-
Replay buffer size	10^6	2×10^5	-	-
Batch-size	-	-	1000	1000
Target parameters smoothing coefficient (τ)	0.005	0.005	-	-
Target parameters update interval	1	2	-	-
Gradient steps	1	1	0.01	0.01
Gumbel-softmax temperature (T)	0.5	0.5	-	-

Table 2: Hyperparameters

B.1.2 PUSHER

In pusher environment, a simple manipulator has to move an object to the target location. The primitive policies were to push the object to the bottom and left. In this environment, the state information for both primitive policies and the composite policy include the goal location. Therefore, $\mathcal{G}$, in this case, is null. The reward function is given as:

$$-\lambda_g||o_{xy} - g_{xy}||^2 - \lambda_o||r_{xy} - o_{xy}||^2 - \lambda_{ct}||u||^2 \tag{11}$$

where o_{xy}, g_{xy}, r_{xy}, and u are xy-position of object, xy-position of goal, xy-position of arm, and joint-torques. The coefficients λ_g, λ_o, and λ_{ct} are 1.0, 0.1, and 0.1, respectively.

Model Architectures		Hidden units
Composition-HIRO	High-level Policy: Three layer feed forward network	300
	Encoder Network: Bidirectional RNN with LSTMs	128
	Decoder Network (Single layer feed forward network)	128
	Attention Network: $W_f, W_b, W_d \in \mathbb{R}^{d \times d}; W \in \mathbb{R}^d$	128
Composition-SAC	Encoder Network: Bidirectional RNN with LSTMs	128
	Decoder Network (Single layer feed forward network)	128
	Attention Network: $W_f, W_b, W_d \in \mathbb{R}^{d \times d}; W \in \mathbb{R}^d$	128
HIRO	High-level Policy: Three layer feed forward network	300
	Low-level Policy: Three layer feed forward network	300
Standard RL policy	Two layer feed forward network	256

Table 3: Network Architectures

B.1.3 HALFCHEETAH-HURDLE

In halfcheetah-hurdle environment, a 2D cheetah has to jump over the three hurdles to reach the target. In this environment, the information g in the state $s : [\hat{s}, g]$ corresponds to the x-position of the next nearest hurdle in front of the agent as well as the distance from that hurdle. The reward function is defined as:

$$-\lambda_g ||r_{xy} - g_{xy}||^2 - \lambda_{hc} hc(\cdot) + \lambda_{rg} I(\text{goal}) + \lambda_z |v_z| + \lambda_v v_x - \lambda_c cc(\cdot) \quad (12)$$

where r_{xy}, g_{xy}, v_z, and v_x are xy-position of robot torso, xy-position of goal, velocity along z-axis, and velocity along x-axis, respectively. The function $hc(\cdot)$ returns a count indicating the number of hurdles in front of the robot. The indicator function $I(\text{goal})$ returns 1 if the agent has reached the target otherwise 0. The function $cc(\cdot)$ is a collision checker which returns 1 if the agent collides with the hurdle otherwise 0. The reward function coefficients λ_g, λ_{hc}, λ_{rg}, λ_z, λ_v, and λ_c are 0.1, 1.0, 1000, 0.3, 1.0 and 2, respectively.

B.2 HYPERPARAMETERS AND NETWORK ARCHITECTURES

Table 2 summarizes the hyperparameters used to train policies with SAC (Haarnoja et al., 2018b), TRPO (Schulman et al., 2015), PPO (Schulman et al., 2017), and HIRO (Nachum et al., 2018).

Table 3 summarizes the network architectures. The standard RL policy structure correspond to simple SAC, TRPO and PPO policies. The right most column shows the hidden units per layer.