LEARNING FUNCTIONALLY DECOMPOSED HIERARCHIES FOR CONTINUOUS NAVIGATION TASKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Solving long-horizon sequential decision making tasks in environments with sparse rewards is a longstanding problem in reinforcement learning (RL) research. Hierarchical Reinforcement Learning (HRL) has held the promise to enhance the capabilities of RL agents via operation on different levels of temporal abstraction. Despite the success of recent works in dealing with inherent nonstationarity and sample complexity, it remains difficult to generalize to unseen environments and to transfer different layers of the policy to other agents. In this paper, we propose a novel HRL architecture, *Hierarchical Decompositional Reinforcement Learning* (HiDe), which allows decomposition of the hierarchical layers into independent subtasks, yet allows for joint training of all layers in end-to-end manner. The main insight is to combine a control policy on a lower level with an image-based planning policy on a higher level. We evaluate our method on various complex continuous control tasks for navigation, demonstrating that generalization across environments and transfer of higher level policies can be achieved. See videos `https://sites.google.com/view/hide-rl`.

1 INTRODUCTION

Reinforcement learning (RL) has been succesfully applied to sequential-decision making tasks, such as learning how to play video games in Atari (Mnih et al., 2013), mastering the game of Go (Silver et al., 2017) or continuous control in robotics (Lillicrap et al., 2015; Levine et al., 2015; Schulman et al., 2017). However, despite the success of RL agents in learning control policies for myopic tasks, such as reaching a nearby target, they lack the ability to effectively reason over extended horizons. In this paper, we consider continuous control tasks that require planning over long horizons in navigation environments with sparse rewards. The task becomes particularly challenging with sparse and delayed rewards since an agent needs to infer which actions caused the reward in a domain where most samples give no signal at all. Common techniques to mitigate the issue of sparse rewards include learning from demonstrations (Schaal, 1999; Peng et al., 2018) or using enhanced exploration strategies (Bellemare et al., 2016; Pathak et al., 2017; Andrychowicz et al., 2017).

Hierarchical Reinforcement Learning (HRL) has been proposed in part to solve such tasks. Typically, a sequential decision making task is split into several simpler subtasks of different temporal and functional abstraction levels (Sutton et al., 1999; Andre & Russell, 2002). Although the hierarchies would ideally be learned in parallel, most methods resort to curriculum learning (Frans et al., 2017; Florensa et al., 2017; Bacon et al., 2016; Vezhnevets et al., 2017). Recent goal-conditioned hierarchical architectures have successfully trained policies jointly via off-policy learning (Levy et al., 2019; Nachum et al., 2018; 2019). However, these methods often do not generalize to unseen environments as we show in Section 5.1. We argue that this is due to a lack of true separation of planning and low-level control across the hierarchy. In this paper, we consider two main problems, namely functional decomposition of HRL architectures in navigation-based domains and generalization of RL agents to unseen environments (figure 1).

To address these issues, we propose a novel multi-level HRL architecture that enables both *functional decomposition* and *temporal abstraction*. We introduce a 3-level hierarchy that decouples the major roles in a complex navigation task, namely planning and low-level control. The benefit of a modular design is twofold. First, layers have access to only task-relevant information for a predefined task, which significantly improves the generalization ability of the overall policy. Hence, this enables policies learned on a single task to solve randomly configured environments. Second,

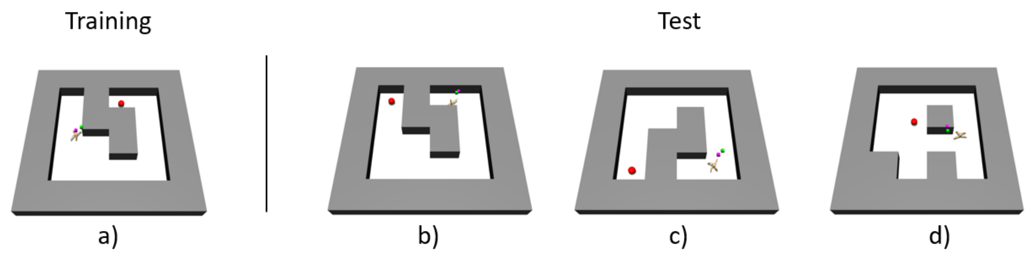


Figure 1: Navigation environments. The red sphere indicates the goal an agent needs to reach, with the starting point at the opposite end of the maze. The agent is trained on environment a). To test generalization, we use the environments with b) reversed starting and goal positions, c) mirrored maze with reversed starting and goal positions and d) randomly generated mazes.

the planning and control layers are modular and thus allow for composition of cross-agent architectures. We empirically show that the planning layer of the hierarchy can be transferred successfully to new agents. During training we provide global environment information only to the planning layer, whereas the full internal state of the agent is only accessible by the control layer. The actions of the top and middle layers are in the form of displacement in space. Similarly, the goals of the middle and lowest layers are relative to the current position. This prevents the policies from overfitting to the global position in an environment and hence ensures generalization to new environments.

In our framework (see figure 2), the planner (i.e., the highest level policy π_2) learns to find a trajectory leading the agent to the goal. Specifically, we learn a value map of the environment by means of a value propagation network (Nardelli et al., 2019). To prevent the policy from issuing too ambitious subgoals, an attention network estimates the range of the lower level policy π_0 (i.e., the agent). This attention mask also ensures that the planning considers the agent performance. The action of π_2 is the position which maximizes the masked value map, which serves as goal input to the policy π_1. The middle layer implements an interface between the upper planner and lower control layer, which refines the coarser subgoals into shorter and reachable targets for the agent. The middle layer is crucial in functionally decoupling the abstract task of planning (π_2) from agent specific continuous control. The lowest layer learns a control policy π_0 to steer the agent to intermediate goals. While the policies are functionally decoupled, they are trained together and must learn to cooperate.

In this work, we focus on solving long-horizon tasks with sparse rewards in complex continuous navigation domains. We first show in a maze environment that generalization causes challenges for state-of-the-art approaches. We then demonstrate that training with the same environment configuration (i.e., fixed start and goal positions) can generalize to randomly configured environments. Lastly, we show the benefits of functional decomposition via transfer of individual layers between different agents. In particular, we train our method with a simple 2DoF ball agent in a maze environment to learn the planning layer which is later used to steer a more complex agent. The results indicate that the proposed decomposition of policy layers is effective and can generalize to unseen environments. In summary our main contributions include:

- A novel multi-layer HRL architecture that allows functional decomposition and temporal abstraction for navigation tasks.
- This architecture enables generalization beyond training conditions and environments.
- Functional decomposition that allows transfer of individual layers across different agents.

2 Related Work

Learning hierarchical policies has seen lasting interest (Sutton et al., 1999; Schmidhuber, 1991; Dietterich, 1999; Parr & Russell, 1998; McGovern & Barto, 2001; Dayan & Hinton, 2000), but many approaches are limited to discrete domains, induce priors or require knowledge of the environment.

More recent works (Bacon et al., 2016; Vezhnevets et al., 2017; Nachum et al., 2018; Levy et al., 2019) have demonstrated HRL architectures in continuous domains. Vezhnevets et al. (2017) introduce FeUdal Networks (FUN), which was inspired by feudal reinforcement learning (Dayan &

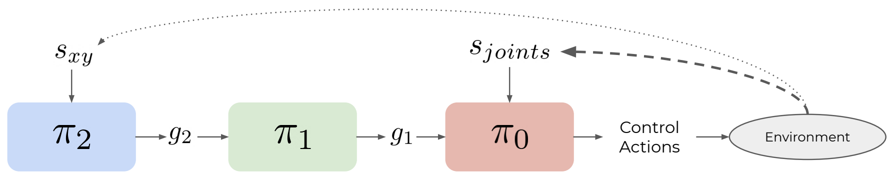


Figure 2: Our 3-layer HRL architecture. The planning layer π_2 receives a birds eye view of the environment and the agent's position s_{xy} and sets an intermediate target position g_2. The interface layer π_2 splits this subgoal into reachable targets g_1. A goal-conditioned control policy π_0 learns the required motor skills to reach the target g_1 given the agent's joint information s_{joints}.

Hinton, 2000). In FUN, a hierarchic decomposition is achieved via a learned state representation in latent space. While being able to operate in continuous state space, the approach is limited to discrete action spaces. Nachum et al. (2018) present HIRO, an off-policy HRL method with two levels of hierarchy. The non-stationary signal of the upper policy is mitigated via off-policy corrections, while the lower control policy benefits from densely shaped rewards. (Nachum et al., 2019) extends HIRO by learning a representation space from environment images, replacing the state and subgoal space with neural representations. Levy et al. (2019) introduce Hierarchical Actor-Critic (HAC), an approach that can jointly learn multiple policies in parallel. The policies are trained in sparse reward environments via different hindsight techniques. Both HAC and HIRO consist of a set of nested policies where the goal of a policy is provided by the top layer. In this setting the goal and state space of the lower policy is identical to the action space of the upper policy. This necessitates sharing of the state space across layers. This shared state space in turn makes global information available to all layers, and thus makes the above approaches prone to overfitting to the training environment. Overcoming this limitation, we introduce a modular design to decouple the functionality of individual layers. This allows us to define different state, action and goal spaces for each layer. Global information about the environment is only available to the planning layer, while lower levels only receive information that is specific to the respective layer.

In model-based reinforcement learning much attention has been given to learning of a dynamics model of the environment and subsequent planning (Sutton, 1990; Sutton et al., 2012; Wang et al., 2019). Eysenbach et al. (2019) propose a planning method that performs a graph search over the replay buffer. However, they require to spawn the agent at different locations in the environment and let it learn a distance function in order to build the search graph. Unlike model-based RL, we do not learn state transitions explicitly. Instead, we learn a spatial value map from collected rewards.

Recently, differentiable planning modules that can be trained via model-free reinforcement learning have been proposed (Tamar et al., 2016; Oh et al., 2017; Nardelli et al., 2019; Srinivas et al., 2018). Tamar et al. (2016) establish a connection between convolutional neural networks and Value Iteration (Bertsekas, 2000). They propose *Value Iteration Networks* (VIN), an approach where model-free RL policies are additionally conditioned on a fully differentiable planning module. MVProp (Nardelli et al., 2019) extends this work by making it more parameter-efficient and generalizable. The planning layer in our approach is based on MVProp, however contrary to prior work we do not rely on a fixed neighborhood mask to sequentially provide actions in its vicinity in order to reach a goal. Instead we propose to learn an attention mask which is used to generate intermediate goals for the underlying layers. Gupta et al. (2017) learn a map of indoor spaces and planning on it using a multi-scale VIN. In their setting, the policy is learned from expert actions using supervised learning. Moreover, the robot operates on discrete set of high level macro actions.

3 BACKGROUND

3.1 GOAL-CONDITIONED REINFORCEMENT LEARNING

We model a Markov Decision Process (MDP) augmented with a set of goals $\mathcal{G}$. We define the MDP as a tuple $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{G}, \mathcal{R}, \mathcal{T}, \rho_0, \gamma\}$, where $\mathcal{S}$ and $\mathcal{A}$ are set of states and actions, respectively, $\mathcal{R}_t = r(s_t, a_t, g_t)$ a reward function, γ a discount factor $\in [0, 1]$, $\mathcal{T} = p(s_{t+1}|s_t, a_t)$ the transition dynamics of the environment and $\rho_0 = p(s_1)$ the initial state distribution, with $s_t \in \mathcal{S}$ and $a_t \in \mathcal{A}$.

Each episode is initialized with a goal $g \in \mathcal{G}$ and an initial state is sampled from ρ_0. We aim to find a policy $\pi : \mathcal{S} \times \mathcal{G} \to \mathcal{A}$, which maximizes the expected return.

We train our policies by using an actor-critic framework where the goal augmented action-value function is defined as:

$$Q(s, g, a) = \mathbb{E}_{a_t \sim \pi, s_{t+1} \sim \mathcal{T}} \left[\sum_{i=t}^{T} \gamma^{i-t} r(s_t, g_t, a_t) | s_t = s, g_t = g, a_t = a \right] \tag{1}$$

The Q-function (critic) and the policy π (actor) are approximated by using neural networks with parameters θ^Q and θ^π. The objective for θ^Q minimizes the loss:

$$\begin{aligned} L(\theta^Q) &= \mathbb{E}_{\mathcal{M}} \left[\left(Q(s_t, g_t, a_t; \theta^Q) - y_t \right)^2 \right] \quad \text{, where} \\ y_t &= r(s_t, g_t, a_t) + \gamma Q(s_{t+1}, g_{t+1}, a_{t+1}; \theta^Q) \quad . \end{aligned} \tag{2}$$

The policy parameters θ^π are trained to maximize the Q-value:

$$L(\theta^\pi) = \mathbb{E}_\pi \left[Q(s_t, g_t, a_t; \theta^Q) | s_t, g_t, a_t = \pi(s_t, g_t; \theta^\pi) \right] \tag{3}$$

To address the issue of sparse rewards, we utilize Hindsight Experience Replay (HER) (Andrychowicz et al., 2017), a technique to improve sample-efficiency in training goal-conditioned environments. The insight is that the desired goals of transitions stored in the replay buffer can be relabeled as states that were achieved in hindsight. Such data augmentation allows learning from failed episodes, which may generalize enough to solve the intended goal.

3.2 Hindsight Techniques

In HAC, Levy et al. (2019) apply two hindsight techniques to address the challenges introduced by the non-stationary nature of hierarchical policies and the environments with sparse rewards. In order to train a policy π_i, optimal behavior of the lower-level policy is simulated by *hindsight action transitions*. More specifically, the action a_i is replaced with a state s_{i-1} that is actually achieved by the lower-level policy π_{i-1}. Identically to HER, *hindsight goal transitions* replace the subgoal g_{i-1} with an achieved state s_{i-1}, which consequently assigns a reward to the lower-level policy π_{i-1} for achieving the virtual subgoal. Additionally, a third technique called *subgoal testing* is proposed. The incentive of subgoal testing is to help a higher-level policy understand the current capability of a lower-level policy and to learn Q-values for subgoal actions that are out of reach. We find both techniques effective and apply them to our model during training.

3.3 Value Propagation Networks

Tamar et al. (2016) propose differentiable value iteration networks (VIN) for path planning and navigation problems. Nardelli et al. (2019) propose value propagation networks (MVProp) with better sample efficiency and generalization behavior. MVProp creates reward- and propagation maps covering the environment. The reward map highlights the goal location and the propagation map determines the propagation factor of values through a particular location. The reward map is an image $\bar{r}_{i,j}$ of the same size as the environment image I, where $\bar{r}_{i,j} = 0$ if the pixel (i, j) overlaps with the goal position and -1 otherwise. The value map V is calculated by unrolling max-pooling operations in a neighborhood N for k steps as follows:

$$\begin{aligned} v_{i,j}^{(0)} &= \bar{r}_{i,j} \\ v_{i,j}^{(k)} &= \max \left(v_{i,j}^{(k-1)}, \max_{(i',j') \in N(i,j)} (\bar{r}_{i,j} + p_{i,j}(v_{i',j'}^{(k-1)} - \bar{r}_{i,j}) \right) \end{aligned} \tag{4}$$

The action (i.e., the target position) is selected to be the pixels (i', j') maximizing the value in a predefined $3x3$ neighborhood $N(i_0, j_0)$ of the agent's current position (i_0, j_0):

$$\pi(s, (i_0, j_0)) = \underset{i',j' \in N(i_0,j_0)}{\text{argmax}} \; v_{i',j'}^{(k)} \tag{5}$$

Note that the window $N(i_0, j_0)$ is determined by the discrete, pixel-wise actions.

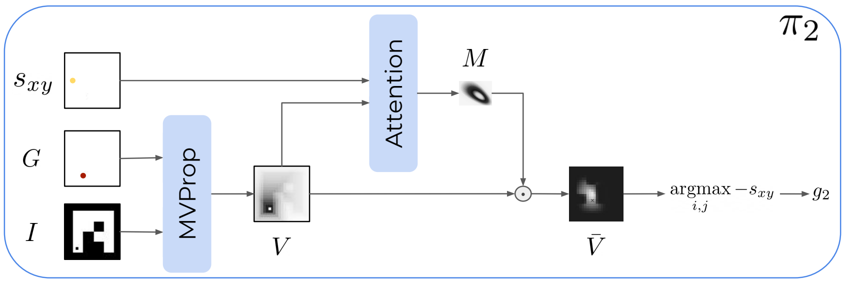


Figure 3: Planner layer $\pi_2(s_{xy}, G, I)$. Given the top-view environment image I and goal G on the map, the maximum value propagation network (MVProp) calculates a value map V. By using the agent's current position s_{xy}, we estimate an attention mask M restricting the global value map V to a local and reachable subgoal map $\bar{V}$. The policy π_2 selects the coordinates with maximum value and assigns the lower policy π_1 with a sugboal that is relative to the agent's current position.

4 Hierarchical Decompositional Reinforcement Learning

We introduce a novel hierarchical architecture, HiDe, allowing for an explicit functional *decomposition* across layers. Similar to HAC (Levy et al., 2019), our method achieves temporal abstractions via nested policies. Moreover, our architecture enables functional decomposition explicitly. This is achieved by nesting i) an abstract planning layer, followed ii) by a local planer to iii) guide a control component. Crucially, only the top layer receives global information and is responsible for planning a trajectory towards a goal. The lowest layer learns a control policy for agent locomotion. The middle layer converts the planning layer's input into subgoals for the control layer. Achieving functional decoupling across layers crucially depends on reducing the state in each layer to the information that is relevant to its specific task. This design significantly improves generalization (see Section 5).

4.1 Planning Layer

The highest layer of a hierarchical architecture is expected to learn high-level actions over a longer horizon, which define a coarse trajectory in navigation-based tasks. In the related work (Levy et al., 2019; Nachum et al., 2018; 2019), the planning layer, learning an *implicit* value function, shares the same architecture as lower layers. Since the task is learned for a specific environment, limits to generalization are inherent to this design choice. In contrast, we introduce a planning specific layer consisting of several components to learn the map and to find a feasible path to the goal.

The planning layer is illustrated in figure 3. We utilize a value propagation network (MVProp) (Nardelli et al., 2019) to learn an *explicit* value map which projects the collected rewards onto the environment image. Given a top-down image of the environment, a convolutional network determines the per pixel flow probability $p_{i,j}$. For example, the probability value of a pixel corresponding to a wall should be 0 and that for free passages 1 respectively.

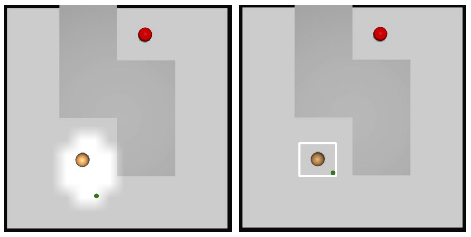
Nardelli et al. (2019) use a predefined 3×3 neighborhood of the agent's current position and pass the location of the maximum value in this neighbourhood as goal position to the agent (equation 5). We augment a MVProp network with an attention model which learns to define the neighborhood dynamically and adaptively. Given the value map V and the agent's current position s_{xy}, we estimate how far the agent can go, modeled by a 2D Gaussian. More specifically, we predict a full covariance matrix Σ with the agent's global position s_{xy} as mean. We later build a 2D mask M of the same size as the environment image I by using the likelihood function:

$$m_{i,j} = \mathcal{N}((i,j)|s_{xy}, \Sigma) \tag{6}$$

Intuitively, the mask defines the density for the agent's success rate. Our planner policy selects an action (i.e., subgoal) that maximizes the masked value map as follows:

$$\begin{aligned} \bar{V} &= M \cdot V \\ \pi_2(s_{xy}, G, I) &= \underset{i,j}{\text{argmax}}\ \bar{v}_{i,j} \\ g_2 &= \pi_2(s_{xy}) - s_{xy} \quad , \end{aligned} \tag{7}$$

Figure 4: A visual comparison of *(left)* our dynamic attention window with a *(right)* fixed neighborhood. The green dot corresponds to the selected subgoal in this case. Notice how our window is shaped so that it avoids the wall and induces a further subgoal.



where $\bar{v}_{i,j}$ corresponds to the value at pixel (i, j) on the masked value map $\bar{V}$. Note that the subgoal selected by the planning layer g_2 is relative to the agent's current position s_{xy}, which improves generalization performance of our model.

The benefits of having an attention model are twofold. First, the planning layer considers the agent dynamics in assigning subgoals which may lead to fine- or coarse-grained subgoals depending on the underlying agent's performance. Second, the Gaussian window allows us to define a dynamic set of actions for the planner policy π_2, which is essential to find a trajectory of subgoals on the map. While the action space includes all pixels of the value map V, it is limited to the subset of only reachable pixels by the Gaussian mask M. Qualitatively we find this leads to better obstacle avoidance behaviour such as the corners and walls shown in figure 4.

Since our planner layer operates in a discrete action space (i.e., pixels), the resolution of the projected maze image defines the minimum amount of displacement for the agent, affecting maneuverability. This could be tackled by using a soft-argmax (Chapelle & Wu, 2010) to select the subgoal pixel, allowing to choose real-valued actions and providing in-variance to image resolution. In our experiments we see no difference in terms of the final performance. However, since the former setting allows for the use of DQN (Mnih et al., 2013) instead of DDPG (Silver et al., 2014), we prefer the discrete action space for simplicity and faster convergence.

Both the MVProp (equation 4) and Gaussian likelihood (equation 6) operations are differentiable. Hence, MVProp and the attention model parameters are trained by minimizing the standard mean squared Bellman error objective as defined in equation 2.

4.2 Interface Layer

The middle layer in our hierarchy interfaces the high-level planning with low-level control by introducing an additional level of temporal abstraction. The planner's longer-term goals are further split into a number of shorter-term targets. Such refinement policy provides the lower-level control layer with reachable targets, which in return yields easier rewards and hence accelerated learning.

The interface layer policy is the only layer that is not directly interacting with the environment. More specifically, the policy π_1 only receives the subgoal g_2 from the upper layer π_2 and chooses an action (i.e. subgoal g_1) for the lower-level locomotion layer π_0. Note that all the goal, state and action spaces of the policy π_1 are in 2D space. Contrary to Levy et al. (2019), we use subgoals that are relative to the agent's position s_{xy}. This helps to learn symmetrical gaits and helps to generalize.

4.3 Control Layer

The lowest layer learns a goal-conditioned control policy. Due to our explicit functional decomposition, it is the only layer with access to the agent's internal state s_{joints} including joint positions and velocities. Whereas the higher layers only have access to the agent's position. In a navigation task, the agent has to learn locomotion to reach the goal position. Similar to HAC, we use *hindsight goal transition* techniques so that the control policy receives rewards even in failure cases.

All policies in our hierarchy are jointly-trained. We use the DDPG algorithm (Lillicrap et al., 2015) with the goal-augmented actor-critic framework (equation 2-3) for the control and interface layers, and DQN (Mnih et al., 2013) for the planning layer (see section 4.1).

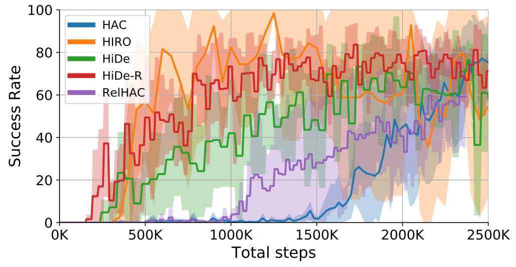


Figure 5: Success rates in forward maze wrt the number of environment steps. All algorithms eventually learn the task. HIRO converges the fastest because it benefits from dense rewards.

Experiment 1	Forward	Backward	Flipped
HAC	82 ± 16	0 ± 0	0 ± 0
HIRO	$\mathbf{99 \pm 1}$	0 ± 0	0 ± 0
RelHAC	66 ± 27	4 ± 8	0 ± 0
HiDe-R	89 ± 3	$\mathbf{61 \pm 14}$	$\mathbf{90 \pm 3}$
HiDe	85 ± 6	55 ± 20	69 ± 40

Table 1: Success rates of achieving a goal with an Ant agent in a simple maze environment. All algorithms except for HiDe have been trained with randomly sampled goal positions.

5 EXPERIMENTS

We evaluate our method on a series of simulated continuous control tasks in navigation-based environments[1]. All environments are simulated in the MuJoCo physics engine (Todorov et al., 2012). Experiments and implementation details are provided in the Appendix B. First, in section 5.1, we compare to various baseline methods in the maze introduced by Nachum et al. (2018). In section 5.2, we move to a new maze with more complex design in order to show our model's generalization capabilities. Finally in section 5.3, we demonstrate that our approach indeed leads to functional decomposition by composing *new* agents via combining the planning layer of one agent with the locomotion layer of another. We introduce the following task configurations:

Maze Forward: the training environment in all experiments. The task is to reach a goal from a fixed pre-determined start position.
Maze Backward: the training maze layout with swapped start and goal positions.
Maze Flipped: a mirrored version of the training environment.
Maze Random: a randomly generated layout with random start and goal positions.

In our experiments, we always train in the Maze Forward environment. We test the agents on the above tasks with fixed starting and goal position. For more details about the environments, we refer to Appendix A. We intend to answer the following two questions: 1) Can our method generalize to unseen test environments? 2) Is it possible to transfer the planning layer policies between agents?

5.1 SIMPLE MAZE NAVIGATION

We compare our method to state-of-the-art approaches including HIRO (Nachum et al., 2019), HAC (Levy et al., 2019) and a modified version of HAC called RelHAC in a simple environment (Maze Forward), shown in figure 6. For a fair comparison, we made a number of modifications to HAC and HIRO implementations. For HAC, we introduced target networks and used the hindsight experience replay technique with the *future* strategy (Andrychowicz et al., 2017). In our experiments we observed that oscillations around the goal kept HIRO agents from finishing the task, which was solved via doubling the distance-threshold of success.

We train a modified HAC model, dubbed RelHAC, to asses our planning layer. RelHAC has the same lowest and middle layers as HiDe but the same top layer as HAC. HiDe is trained with fixed start and goal positions whereas the baseline models are trained by using fixed start and random goal positions. HiDe-R represents HiDe under the same conditions as the baseline methods.

All models learned this task successfully as shown in figure 5 and table 1 (Forward column). HIRO demonstrates slightly better convergence and final performance, which can be attributed to the fact that HIRO is the only approach that is trained with dense rewards. RelHAC performs worse than HAC due to the pruned state space of each layer and due to the lack of an effective planner.

Table 1 summarizes the models' generalization ability to the unseen Maze Backward and Maze Flipped environments (see figure 6). While both HIRO and HAC manage to solve the training

[1]Videos available at `https://sites.google.com/view/hide-rl`

Experiment 2	Ant	Ball
Forward	81 ± 8	96 ± 7
Random	90 ± 2	96 ± 1
Backward	58 ± 10	100 ± 0
Flipped	56 ± 10	99 ± 2

Table 2: Success rates in the complex maze. We train an ant and a ball on Forward maze and test on unseen Random, Backward, and Flipped environments.

Experiment 3	Ant $\rightarrow$ Ball	Ball $\rightarrow$ Ant
Forward	100 ± 0	66 ± 14
Random	97 ± 1	86 ± 5
Backward	90 ± 22	53 ± 9
Flipped	81 ± 44	59 ± 27

Table 3: Success rates of achieving a goal in complex maze environment with transferred agents. We train with a different agent on Forward maze.

environment (Maze Forward) with success rates of 99% and 82%, they suffer from overfiting to the training environment, indicated by the 0% success rates in the unseen test environments. Contrarily, our method is able to achieve 54% and 69% success rates in this generalization task. As expected, training our model with random goal positions (i.e., HiDe-R) yields a more robust model outperforming vanilla HiDe. In subsequent experiments, we only evaluate our method, since the preliminary experiments have shown that the baseline methods do not generalize well.

5.2 Complex Maze Navigation

In this experiment, we train an ant and a ball agent (see Appendix A.1) in the Maze Forward task with a more complex environment layout (cf. figure 1), while we keep both the start and goal positions intact. We then evaluate this model in 4 different tasks (see section 5).

Table 2 reports success rates of both agents in this complex task. Our model successfully transfers its navigation skills to unseen environments. The performance for the Maze Backward and Maze Flipped tasks is similar to the results shown in section 5.1 despite the increased difficulty. Since the randomly generated mazes are typically easier, our model shows similar or better performance.

5.3 Transfer of Policies

To demonstrate that the layers in our architecture indeed learn separate sub-tasks we transfer individual layers across different agents. We take the models from section 5.2 and attach their planning layers to another agent. For the locomotion layer we trained agents without a planning layer. Since we still require the interface layer, we use the control and interface layers of RelHAC.

We carry out two sets of experiments. First, we transfer the ant model's planning layer to the simpler 2 DoF ball agent. As indicated in Table 3, the performance of the ball with the ant's planning layer matches the results in Table 2. The ball agent's success rate increases for random (from 96% to 100%) and forward (96% to 97%) maze tasks whereas it decreases slightly in the backward (from 100% to 90%) and flipped (from 99% to 88%) configurations.

Second, we attach the ball agent's planning layer to the more complex ant agent. Our new compositional agent performs marginally better or worse in flipped, random and backward maze tasks. Please note that this experiment is an example of a case where the environment is first learned with a fast and easy-to-train agent (i.e., ball) and then utilized by a more complex agent.

6 Conclusion

In this paper, we introduce a novel HRL architecture that can solve complex navigation tasks in 3D-based maze environments. The architecture consists of a planning layer which learns an explicit value map and is connected with a subgoal refinement layer and a low-level control layer. The framework can be trained end-to-end. While training with a fixed starting and goal position, our method is able to generalize to previously unseen settings and environments. Furthermore, we demonstrate that transfer of planners between different agents can be achieved, enabling us to move a planner trained with a simplistic agent in a complex environment to a more complex agent. In future work, we want to consider integration of a more general planner that is not restricted to navigation-based environments.

References

David Andre and Stuart Russell. State abstraction for programmable reinforcement learning agents. 08 2002.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight Experience Replay. In *Advances in Neural Information Processing Systems*, pp. 5048–5058, 2017.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. abs/1609.05140, 2016.

Marc G. Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Rémi Munos. Unifying count-based exploration and intrinsic motivation. abs/1606.01868, 2016.

Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 2nd edition, 2000. ISBN 1886529094.

Olivier Chapelle and Mingrui Wu. Gradient descent optimization of smoothed information retrieval metrics. *Information retrieval*, 13(3):216–235, 2010.

Peter Dayan and Geoffrey Hinton. Feudal reinforcement learning. *Advances in Neural Information Processing Systems*, 5, 09 2000. doi: 10.1002/0471214426.pas0303.

Thomas G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. cs.LG/9905014, 1999.

Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking Deep Reinforcement Learning for Continuous Control. In *International Conference on Machine Learning*, pp. 1329–1338, 2016.

Benjamin Eysenbach, Ruslan Salakhutdinov, and Sergey Levine. Search on the Replay Buffer: Bridging Planning and Reinforcement Learning. *arXiv preprint arXiv:1906.05253*, 2019.

Carlos Florensa, Yan Duan, and Pieter Abbeel. Stochastic neural networks for hierarchical reinforcement learning. abs/1704.03012, 2017.

Kevin Frans, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman. Meta learning shared hierarchies. abs/1710.09767, 2017.

Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2616–2625, 2017.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. abs/1504.00702, 2015.

Andrew Levy, Robert Platt, and Kate Saenko. Learning Multi-Level Hierarchies with Hindsight. In *International Conference on Learning Representations*, 2019.

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.

Amy McGovern and Andrew G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pp. 361–368, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-778-1.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. abs/1312.5602, 2013.

Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient Hierarchical Reinforcement Learning. In *Advances in Neural Information Processing Systems*, pp. 3303–3313, 2018.

Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Near-Optimal Representation Learning for Hierarchical Reinforcement Learning. In *International Conference on Learning Representations*, 2019.

Nantas Nardelli, Gabriel Synnaeve, Zeming Lin, Pushmeet Kohli, Philip H. S. Torr, and Nicolas Usunier. Value Propagation Networks. In *International Conference on Learning Representations*, 2019.

Junhyuk Oh, Satinder Singh, and Honglak Lee. Value prediction network. abs/1707.03497, 2017.

Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. In *Proceedings of the 1997 Conference on Advances in Neural Information Processing Systems 10*, NIPS '97, pp. 1043–1049, Cambridge, MA, USA, 1998. MIT Press. ISBN 0-262-10076-2.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. 2017.

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. abs/1705.05363, 2017.

Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. abs/1804.02717, 2018.

Stefan Schaal. Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences*, 3: 233–242, 1999.

Jürgen Schmidhuber. Learning to generate subgoals for action sequences. *IJCNN-91-Seattle International Joint Conference on Neural Networks*, ii:453 vol.2–, 1991.

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

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic Policy Gradient Algorithms. In *International Conference on Machine Learning*, pp. 387–395, 2014.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–, October 2017.

Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks. abs/1804.00645, 2018.

Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *ML*, 1990.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 112(1-2):181–211, August 1999. ISSN 0004-3702. doi: 10.1016/S0004-3702(99)00052-1.

Richard S. Sutton, Csaba Szepesvári, Alborz Geramifard, and Michael Bowling. Dyna-style planning with linear function approximation and prioritized sweeping. abs/1206.3285, 2012.

Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value Iteration Networks. In *Advances in Neural Information Processing Systems*, pp. 2154–2162, 2016.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. DeepMind Control Suite. Technical report, January 2018.

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

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

Tingwu Wang, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. Benchmarking model-based reinforcement learning. abs/1907.02057, 2019.

A Environment details

We build on the Mujoco (Todorov et al., 2012) environments used in Nachum et al. (2018). All environments use $dt = 0.02$. Each episode in experiment 1 is terminated after 500 steps and after 800 steps in the rest of the experiments or after the goal in reached. All rewards are sparse as in Levy et al. (2019), i.e., 0 for reaching the goal and -1 otherwise. We consider goal reached if $|s - g|_{\max} < 1$. Since HIRO sets the goals in the far distance to encourage the lower layer to move to the goal faster, it can't stay exactly at the target position. Moreover, they do not terminate the episode after the goal is reached. Thus for HIRO, we consider a goal reached if $|s - g|_2 < 2.5$.

A.1 Agents

Our Ant agent is equivalent to the one in Levy et al. (2019). In other words, the Ant from Rllab (Duan et al., 2016) with gear power of 16 instead of 150 and 10 frame skip instead of 5. Our Ball agent is the PointMass agent from DM Control Suite (Tassa et al., 2018). We made the change the joints so that the ball rolls instead of sliding. Furthermore, we resize the motor gear and the ball itself to match the maze size.

A.2 Mazes

All mazes are modelled by immovable blocks of size $4 \times 4 \times 4$. Nachum et al. (2018) uses blocks of $8 \times 8 \times 8$. The environment shapes are clearly depicted in figure 1. For the randomly generated maze, we sample each block with probability being empty $p = 0.8$,start and goal positions are also sampled randomly at uniform. Mazes where start and goal positions are adjacent or where goal is not reachable are discarded. For the evaluation, we generated 500 of such environments and reused them (one per episode) for all experiments.

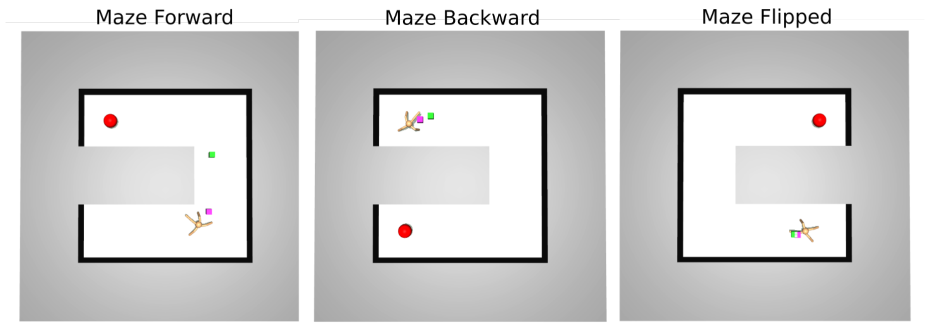


Figure 6: Environments used for testing in Section 5.1. The red sphere indicates the goal of the task.

B Implementation Details

Our PyTorch (Paszke et al., 2017) implementation will be available at the project website. [2]

B.1 Baseline experiments

For both HIRO and HAC we used the original authors implementation[3][4]. In HIRO, we set the goal success radius for evaluation as described above. We ran the hiro_xy variant, which uses only position coordinates for subgoal instead of all joint positions, to have a fair comparison with our method. To improve the performance of HAC in experiment one, we modified their Hindsight Experience Replay (Andrychowicz et al., 2017) implementation so that they use Future strategy. More importantly, we also added target networks to both the actor and critic to improve the performance.

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

[3] HIRO:`https://github.com/tensorflow/models/tree/master/research/efficient-hrl`

[4] HAC:`https://github.com/andrew-j-levy/Hierarchical-Actor-Critc-HAC-`

B.2 Network Structure

B.2.1 Planning layer

Input images for the planning layer were binnarized in the following way: each pixel corresponds to one block (0 if it was a wall or 1 if it was a corridor). In our planning layer, we process the input image of size 32x32 (20x20 for experiment 1) via two convolutional layers with 3×3 kernels. Both layers have only 1 input and output channel and are padded so that the output size is the same as the input size. We propagate the value through the value map as in Nardelli et al. (2019) $K = 35$ times using a 3×3 max pooling layer. Finally, the value map and agent position image (a black image with a dot at agent position) is processed by 3 convolutions with 32 output channels and 3×3 filter window interleaved by 2×2 max pool with ReLU activation functions and zero padding. The final result is flatten and processed by two fully connected layers with 64 neurons each producing three outputs: σ_1, σ_2, ρ with $\mathrm{softplus}$, $\mathrm{softplus}$ and $\tanh$ activation functions respectively. The final covariance matrix Σ is given by

$$\Sigma = \begin{pmatrix} \sigma_1^2 & \rho\,\sigma_1\sigma_2 \\ \rho\,\sigma_1\sigma_2 & \sigma_2^2, \end{pmatrix}$$

so that the matrix is always symmetric and positive definite. For numerical reasons, we multiply by binnarized kernel mask instead of the actual Gaussian densities. We set values higher than mean to 1 and others to zeros. In practice, we use this line:

```
kernel = t.where(kernel >= kernel.mean(dim=[1,2], keepdim=True),
                 t.ones_like(kernel), t.zeros_like(kernel))
```

B.2.2 Middle and Locomotion layer

We use the same network architecture for the middle and lower layer as proposed by Levy et al. (2019), i.e. we use 3 times fully connected layer with ReLU activation function. The locomation layer is activated with $\tanh$, which is then scaled to the action range.

B.2.3 Training parameters

- Discount $\gamma = 0.98$ for all agents.
- Adam optimizer. Learning rate 0.001 for all actors and critics.
- Soft updates using moving average; $\tau = 0.05$ for all controllers.
- Replay buffer size was designed to store 500 episodes, similarly as in Levy et al. (2019)
- We performed 40 updates after each epoch on each layer, after the replay buffer contained at least 256 transitions.
- Batch size 1024.
- No gradient clipping
- Rewards 0 and -1 without any normalization.
- Subgoal testing (Levy et al., 2019) only for the middle layer.
- Observations also were not normalized.
- 2 HER transitions per transition using the FUTURE strategy (Andrychowicz et al., 2017).
- Exploration noise: 0.05, 0.01 and 0.1 for the planning, middle and locomotion layer respectively.

Experiment	Forward	Backward	Flipped
HAC 1	96.4	00.0	00.0
HAC 2	82.0	00.0	00.0
HAC 3	85.6	00.4	00.0
HAC 4	92.8	00.0	00.0
HAC 5	55.6	00.0	00.0
HIRO 1	100	00.0	00.0
HIRO 2	99.8	00.0	00.0
HIRO 3	99.0	00.0	00.0
HIRO 4	99.6	00.0	00.0
HIRO 5	97.2	00.0	00.0
RelHAC 1	77.6	18.8	00.0
RelHAC 2	86.4	00.0	00.0
RelHAC 3	77.8	00.0	00.0
RelHAC 4	19.6	00.0	00.0
RelHAC 5	71.0	00.0	00.0
HiDe-R 1	87.6	72.2	90.6
HiDe-R 2	87.8	70.4	89.6
HiDe-R 3	86.4	38.2	89.8
HiDe-R 4	90.6	69.4	94.0
HiDe-R 5	94.4	56.2	87.0
HiDe 1	80.6	21.2	00.0
HiDe 2	94.6	71.8	96.8
HiDe 3	81.6	53.4	90.2
HiDe 4	79.4	61.4	91.2
HiDe 5	87.0	66.0	66.6

Table 4: Results for experiment 1 for individual seeds.

	Ant 1	Ant 2	Ant 3	Ant 4	Ant 5	Ball 1	Ball 2	Ball 3	Ball 4	Ball 5
Forward	82.0	80	92.2	70.6	82.0	99.6	99.0	100	83.2	100
Random	85.8	88.2	89.2	91.4	92.0	97.0	94.2	97.6	97.2	96.0
Backward	61.8	61.4	56.0	42.0	60.4	100	100	100	100	100
Flipped	87.4	68.2	64.8	64.0	85.4	100	96.4	100	100	100

Table 5: Results for experiment 2 for individual seeds.

	A→B 1	A→B 2	A→B 3	A→B 4	A→B 5
Forward	100	100	100	100	100
Random	96.2	96.6	96.8	97.8	96.6
Backward	90.2	99.8	99.8	99.2	100
Flipped	100	100	100	100	100

Table 6: Results for Ant to Ball transfer for individual seeds.

	B→A 1	B→A 2	B→A 3	B→A 4	B→A 5
Forward	45.8	81.2	75.8	64.8	64.4
Random	81.6	85	94.4	83.4	88.0
Backward	49.4	45.4	68.6	48.0	54.6
Flipped	32.0	83.8	79.6	26.8	70.4

Table 7: Results for Ball to Ant transfer for individual seeds.