HIERARCHICAL REINFORCEMENT LEARNING WITH HINDSIGHT

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce a new Hierarchical Reinforcement Learning (HRL) framework that can accelerate learning in tasks involving long time horizons and sparse rewards. Our approach improves sample efficiency by enabling agents to simultaneously learn a hierarchy of short policies that operate at different time scales. Our method can also learn policy hierarchies with an arbitrary number of levels. Indeed, our framework is the first HRL approach to show results in which a 3-level agent outperforms both 2-level and 1-level agents in tasks with continuous state and action spaces. We demonstrate experimentally in both grid world and simulated robotics domains that our approach can significantly boost sample efficiency. A video illustrating our results is available at `https://www.youtube.com/watch?v=i04QF7Yi50Y`.

1 INTRODUCTION

Reinforcement Learning (RL) algorithms often struggle in continuous domains that involve long time horizons and sparse rewards. A major reason for this outcome is that many RL agents learn using 1-Step Temporal Difference (TD) methods(Sutton & Barto (1998)), which is limited to performing credit assignment one time step at a time. Hierarchical Reinforcement Learning, in which agents learn multiple policies that act at different time scales, can potentially help with accelerating credit assignment. Consider a 2-level agent in which the higher level proposes subgoal states for the lower level to achieve. Assuming each subgoal from the higher level can consist of at most 4 primitive actions from the lower level, 1 time step at the higher level equates to a maximum of 4 time steps at the lower level. If this agent was given a task to achieve some goal state that requires at least 16 primitive actions, the 2-level agent could learn a policy that achieves the goal using as few as 4 iterations of 1-Step TD. On the other hand, a flat agent may need to perform 16 iterations of 1-step TD.

Yet HRL agents with multiple levels are difficult to train due to the issue of non-stationary transition functions. When an agent uses a hierarchy of policies, as soon as one of the policies within the hierarchy changes, the state transition function for all levels above the level that changed now have a different transition function. A different transition function often means that a level needs to learn a new policy for the new RL problem. The end result is that agents are often forced to learn the policies of the hierarchy bottom-up instead of in parallel, which may entirely diminish the benefits that HRL can provide.

We introduce a new HRL framework that enables agents to learn the policies within the hierarchy in parallel and in an end-to-end manner. Our framework contains two components: (i) an MDP transformation operation that enables agents to learn the policies within the hierarchy in parallel and (ii) Hindsight Experience Replay (HER)(Andrychowicz et al. (2017)), which helps each level learn an effective policy when the reward function is sparse. In addition, the policy hierarchies that our framework learns can support an arbitrary number of policy levels. Indeed, to the best of our knowledge, our HRL framework is the first to show results in which 3-level agent outperforms both 2-level and 1-level agents in tasks that involve continuous state and action spaces.

There are other automated hierarchical RL approaches that can work in tasks with continuous state and action spaces (Nachum et al. (2018), Bacon et al. (2016), Vezhnevets et al. (2017), Konidaris & Barto (2009), Schmidhuber (1991)). Relative to these other HRL approaches, our method has

at least 1 of the following 2 advantages: (i) the ability to learn the policies within the hierarchy in parallel and (ii) the ability to learn more than 2 policies.

We evaluated our approach on both grid world tasks and more complex simulated robotics environments. For each task, we evaluated agents with 1, 2, and 3 levels of hierarchy. In all tasks, agents using 2 and 3 levels of hierarchy significantly outperformed agents that learned a flat policy. Further, in many of the challenging tasks, agents using 3 levels of hierarchy significantly outperformed agents using 2 levels of hierarchy. In addition, we compared our approach against another HRL algorithm HIRO(Nachum et al. (2018)) on two simulated robotics tasks. Our approach outperformed HIRO on both tasks.

2 BACKGROUND

The particular sequential decision making problem we are interested in solving is a Markov Decision Process (MDP) augmented with a set of goals $\mathcal{G}$ we would like an agent to learn. In this paper, a goal is just a state or a set of states. We define an MDP augmented with a set of goals as a *Universal MDP* (UMDP). A UMDP is a tuple $\mathcal{U} = (\mathcal{S}, \mathcal{G}, \mathcal{A}, T, R, \gamma)$, in which $\mathcal{S}$ is the set of states; $\mathcal{G}$ is the set of goals; $\mathcal{A}$ is the set of actions; T is the transition probability function in which $T(s, a, s')$ is the probability of transitioning to state s' when action a is taken from state s; R is the reward function; γ is the discount rate. At the beginning of each episode in a UMDP, a fixed goal $g \in \mathcal{G}$ is assigned that defines the reward function for the duration of that episode. The solution to a UMDP is a control policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that maximizes the the value function $v_\pi(s, g) = \mathbb{E}_\pi[\sum_{n=0}^{\infty} \gamma^n R_{t+n+1} | s_t = s, g_t = g]$ for an initial state s and goal g.

In order to implement hierarchical agents in tasks with continuous state and actions spaces, we will use two techniques from the RL literature: (i) the Universal Value Function Approximator (UVFA)(Schaul et al. (2015)) and (ii) Hindsight Experience Replay (Andrychowicz et al. (2017)). The UVFA will be used to estimate the action-value function of a policy π, $q_\pi(s, g, a) = \mathbb{E}_\pi[\sum_{n=0}^{\infty} \gamma^n R_{t+n+1} | s_t = s, g_t = g, a_t = a]$ in tasks that involve continuous state and action spaces. In our experiments, the UVFAs used will be in the form of feed-forward neural networks. UVFAs are important for learning goal-conditioned policies because they can potentially generalize Q-values from certain regions of *(state, goal, action)* tuple space to other regions of the tuple space. Yet UVFAs are less helpful in difficult tasks that use sparse reward functions. In these tasks when the sparse reward is rarely achieved, the UVFA will not have large regions of the *(state, goal, action)* tuple space with relatively high Q-values that it can transfer to other regions. For this reason, we also use Hindsight Experience Replay (Andrychowicz et al. (2017)) in our approach. HER is a data augmentation technique that creates copies of the traditional *[state, action, reward, next state, goal]* that are created in traditional off-policy RL. The original goal element is replaced with a state that was actually achieved during the episode, which guarantees that at least one of the HER transitions will contain the sparse reward. These HER transitions in turn help the UVFA learn about regions of the *(state, goal, action)* tuple space that should have relatively high Q-values, which the UVFA can then potentially transfer to the other areas of the tuple space.

3 APPROACH

We introduce a framework that can learn the policies contained within a larger hierarchical policy in parallel. The framework contains two components. The first component is a UMDP transformation operation that will make it possible to learn the policies within the hierarchy simultaneously. The second element is Hindsight Experience Replay(Andrychowicz et al. (2017)), which will make it easier for each policy in the hierarchy to learn effective goal-conditioned policies in challenging, sparse-reward tasks. Before discussing each of these components, we will first provide more detail on the hierarchical policy we seek, and why it is difficult to learn the individual policies within the hierarchy in parallel.

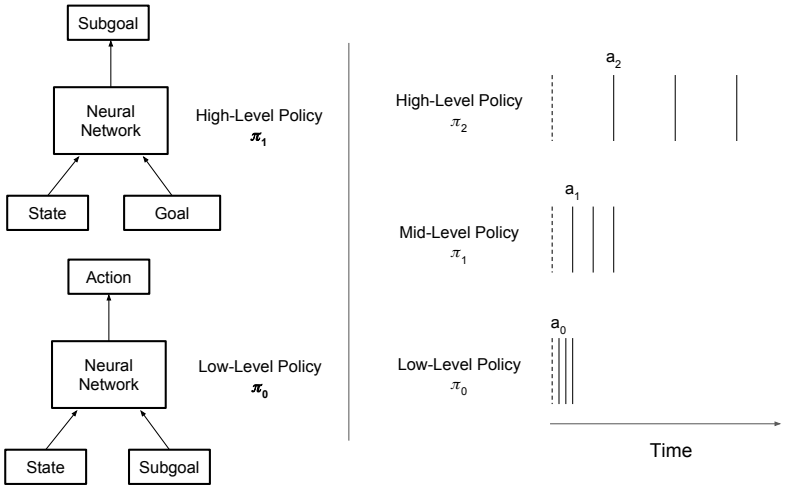

Figure 1: (Left) Policy architecture of an agent with 2 levels of hierarchy that operates in a continuous domain. (Right) Maximum time scales for each policy within a 3-level hierarchy when H=4. The time scale grows exponentially at each successive level.

3.1 Desired Hierarchical Policy

We seek to learn a hierarchical policy Π_{k-1} consisting of k individual policies $\pi_0, \ldots, \pi_{k-1}$, in which k is a hyperparameter chosen by the user. We require that the hierarchical policy Π_{k-1} have the following properties.

Each policy will be a deterministic mapping of states and goals to actions: $\pi_i : \mathcal{S}_i, \mathcal{G}_i \rightarrow \mathcal{A}_i$. The action space for all policies except the bottom-most level will be a desired subgoal state for the next level to achieve: $A_{i:i>0} = S$. The actions output by the base policy π_0 will be the set of primitive actions that are available to the agent. Figure 1(Left) shows the architecture of a k = 2 level policy hierarchy.

In addition, we require that Π_{k-1} be nested. The subgoal state output by policy π_i within the hierarchy should require at most H actions from the policy one level below π_{i-1}, in which H, or the maximum horizon of a subgoal action, is another parameter provided by the user. The nested structure is critical because it results in policies that operate at time scales that increase exponentially with the level of the policy. Policies that act with exponentially increasing time scales are important because they can exponentially reduce the length of the sequence of actions that each policy needs to learn. Consider the k = 3 level hierarchical policy in Figure 1(Right), in which the the maximum horizon of a subgoal H = 4 actions. In this example, the time scale of the π_0, π_1, and π_2 are 1, 4, and 16 primitive actions, respectively. If a task required 64 primitive actions, each of the policies would only need to learn sequences of 4 actions each.

Combining the action space for each policy and the desired nested structure of the policy hierarchy, the transition function for level $i : i > 0$, $T_{i|\Pi_{i-1}}(state, action)$, will work as follows. The subgoal action selected a_i by level i is assigned to be the goal of level $i-1$: $g_{i-1} = a_i$. π_{i-1} then has at most H attempts to achieve g_{i-1}. When either π_{i-1} runs out of H attempts or a goal $g_n, n \geq i-1$, is achieved, the transition function terminates and the agents current state is returned. The state transition for function is labeled as $T_{i|\Pi_{i-1}}$ because it depends on the full policy hierarchy below level i, Π_{i-1}, to determine the next state. The full state transition function for level $i : i > 0$ is provided in Algorithm 2, which is listed in the Appendix due to space constraints. The base transition function T_0 is assumed to be provided by the task.

3.2 Challenges with Learning Policies in Parallel

Without further changes, it is not possible to train these policies in parallel for two reasons. The first reason is the issue of non-stationary state transition functions. As soon as an update is made

to the policy at level i, the state transition function for all levels $n > i$, $T_{n|\Pi_{n-1}}$, will automatically change. This means that any previous updates made to level i's Q-function $Q_{i|\Pi_{i-1}}$ may be obsolete and any replay transitions that are stored may be invalid as they could violate the expectation of the Q-function, which assumes a stationary transition function.

The second key reason is that each level would not be able to explore. If level i proposed a subgoal a_i for level $i-1$ to achieve, and level $i-1$ acted with a noisy policy when trying to achieve a_i, this would result in some transition *[state, action, reward, next state, goal]* for level i. However, this transition could not be used to update level i's Q-function $Q_{i|\Pi_{i-1}}$ because this transition was created with a different transition function than $T_{i|\Pi_{i-1}}$ as Π_{i-1} was not followed. The use of this transition would thus violate the expectation of the Q-function, $Q_{i|\Pi_{i-1}}$, which requires a stationary transition function. Yet, each level in the hierarchy needs to be able to explore. Due to the deterministic nature each policy π_i, an off-policy strategy, in which level i uses a behavioral policy π_{i_b} to explore the environment while optimizing a different target policy π_i, would be required.

As a consequence of not being able to learn these policies in parallel, an agent would be forced to learn one policy at a time, bottom-up. Only when the level $i-1$ policy π_{i-1} has converged to a near-optimal policy, can the level i be trained. This is clearly unsatisfying because (i) it would be difficult to know when a policy π_i has converged when it is not known beforehand which *(initial state, goal)* combinations π_i will need to learn to achieve and (ii) learning the policies within the hierarchy in parallel could be much faster.

3.3 Learning Policies Simultaneously via Hindsight Actions

In order for an agent to learn the policies within the hierarchy in parallel, there are two obstacles that need to be overcome: (i) the non-stationary state transition functions and (ii) the inability to explore. We next derive a potential solution to the non-stationarity issue, which will also happen to solve the exploration problem.

The non-stationary state transition function issue will continue to hinder the training of level i's policy π_i until the lower level policy hierarchy Π_{i-1} has stabilized. But Π_{i-1} will not stabilize until each $\pi \in \Pi_{i-1}$ has converged to its optimal policy π^*. Thus, in order to learn all policies in parallel, each policy needs to be trained with respect to its optimal lower level policy hierarchy Π^*_{i-1}. In other words, in order to learn the policies simultaneously, each level i needs to learn the policy π_i that maximizes the action-value function function $q_{\pi_i|\Pi^*_{i-1}}$.

The next challenge lies in how to learn the level i's estimated Q-function $Q_{\pi_i|\Pi^*_{i-1}}$ when level i does not have access to the optimal policy hierarchy below level i, Π^*_{i-1}. Without Π^*_{i-1}, we do not know the transition function, $T_{i|\Pi^*_{i-1}}(s,a)$. Without the transition function, we would not not be able to judge whether a transition *[state, action, reward, next state, goal* violates the transition function and thus the expectation within $Q_{\pi_i|\Pi^*_{i-1}}$. But one aspect of the transition function $T_{i|\Pi^*_{i-1}}(s,a)$ is known. If the subgoal action a_i is achievable within H actions by the policy below π_{i-1}, then when a_i is proposed in current state s_i, $T_{i|\Pi^*_{i-1}}(s_i,a_i) = a_i$ (i.e., the agent will successfully move to state a_i). Further, a level can discover subgoal states that require at most H actions, by simply having the level below follow some exploratory behavior policy, π_{i-1_b}, for at most H actions and then note the state that was achieved in hindsight s_{hind}.

The strategy for training the level i function $Q_{\pi_i|\Pi^*_{i-1}}$ is now almost complete. Level i can propose some subgoal state a_i in its current state s_i for level $i-1$ to achieve. Level $i-1$ can then use some behavior policy for at most H actions to try to achieve its goal $g_{i-1} = a_i$ and then note the state s_{hind} that was achieved in hindsight. Level i can then create a transition that replays its previous action as if it had proposed the state achieved in hindsight s_{hind} as its original subgoal: *[state = s, action = s_{hind}, reward = TBD, next state = s_{hind},goal = g_i]*. The last component that needs to be completed is the reward value. The major requirement for the reward is that it must be independent of the path that level $i-1$ followed before landing in state s_{hind} because that path is likely different than had the optimal policy hierarchy Π^*_{i-1} been followed exactly. Instead the reward should depend on the state achieved in hindsight s_{hind} and level i's goal g_i. We will use the reward function $R_i(s',g)$ that grants a reward of 0 if the next state maps to the goal and -1 otherwise: $R_i(s'_i, g_i) = 0, \forall s'_i : s'_i \in g_i$ and $R_i(s'_i, g_i) = -1, \forall s'_i : s'_i \notin g_i$. In addition, we will also set the discount rate $\gamma_i = 0$, when the hindsight action also achieves the goal. The full transition used to train $Q_{\pi_i|\Pi^*_{i-1}}$ is now: *[state = s,*

action = s_{hind}, *reward* = $\in \{-1, 0\}, nextstate = s_{hind}$, *goal* = g_i*]*. This transition is valid because if level $i-1$ had followed Π^*_{i-1}, the same transition would have been created.

This strategy of replaying actions with the action executed in hindsight can be scaled to all k levels, enabling all levels to learn in parallel. All that is required is some k-level hierarchical behavior policy Π_{k_b}. In our experiments, we use a noisy version of the hierarchical policy Π_k that has been learned so far for the hierarchical behavior policy. Hindsight actions transitions are then created for level i as soon as H actions have been executed by level $i-1$ or earlier if a goal $g_n, n \geq i-1$ is achieved. In addition, not how learning via hindsight actions has also overcome the exploration obstacle. Because we are training the level i policy with respect to an optimal lower-level policy hierarchy Π^*_{i-1} instead of the current lower-level policy hierarchy Π_{i-1}, exploration noise can be added to any level.

3.4 Flaws of Learning via Hindsight Actions

Learning via hindsight actions does enable all policies $\pi_i \in \Pi_k$ to be trained in parallel, but the strategy does have some serious shortcomings. The most significant flaw is that the strategy only enables a level to learn about a restricted set of subgoal states. A level i will only execute in hindsight subgoal actions that can be achieved with at most H actions from level $i-1$. As a result, level i will only learn Q-values for subgoal actions that are relatively close to its current state and will ignore the Q-values for all subgoal actions that require more than H actions. This is problematic because the action space for all subgoal levels should be the full state space in order for the framework to be end-to-end. If the action space is the full state space and the Q-function is ignoring large regions of the action space, significant problems will occur if the learned Q-function assigns higher Q-values to distant subgoals than to feasible subgoals that can be achieved with at most H actions from the level below. π_i may adjust its policy to output these distant subgoals that have relatively high Q-values. Yet the lower level policy hierarchy Π_{i-1} has not been trained to achieve distant subgoals so the agent may act erratically as a result.

A second shortcoming is that level i may be able to learn goal-conditioned policies more quickly if it took into account the capabilities of the current lower level policy hierarchy Π_{i-1}. Consider a $k = 2$ level agent attempting to learn a task that requires 16 primitive actions when $H = 8$. When learning from hindsight actions, the path with the highest Q-values would be the two-action sequence that first moves the agent halfway between the initial state and goal, which requires 8 primitive actions from π_0. The second subgoal action in this optimal path would then move the agent from the halfway state to the goal, which also requires 8 primitive actions. However, it may be the case that when this optimal path has been found at level 1, the level 0 policy has not fully learned the sequences of 8 actions to achieve each of these subgoals. As a result, the current hierarchical policy Π_1 may not result in a sequence of actions that brings the agent the initial state to the goal state. On the other hand, the level 0 policy may have already learned the 4 consecutive subgoal tasks, each of which contains 4 primitive actions, that lead from the initial state to the goal. If the Q-function at level 1 also penalized subgoals that cannot be achieved with the current iteration of π_0, then acting greedily with respect to this Q-function may produce the path of 4 subgoals that can actually lead the agent from the initial state to the goal.

In order to overcome both of these flaws but still enable agents to learn policies in parallel, we will make the following changes to our approach. First, we will have all levels $i > 0$ at times execute a process we will refer to as subgoal testing. After level i proposes a subgoal a_i, a certain fraction of the time λ, the lower level behavior policy hierarchy, Π_{i-1_b}, used to achieve that goal must be the current lower level policy hierarchy Π_{i-1}. That is, instead of a level being able to explore when trying to achieve its goal, the current lower level policy hierarchy must be followed exactly. In our experiments, we set $\lambda = 0.2$. Second, when subgoal testing is executed, a different reward function is used to evaluate an action. This other reward function will have a third term. $R_i(a, s', g)$ will now issue a $penalty$ reward if the next state s'_i does not map to the original subgoal action a_i (i.e., subgoal action a_i was not achieved by level $i-1$ in H actions). The updated reward function is now $R_i(a_i, s'_i, g_i) = penalty, \forall s'_i : s'_i \notin a_i$, $R_i(a_i, s'_i, g_i) = -1, \forall s'_i : (s'_i \in a_i) \wedge (s'_i \notin g_i)$, and $R_i(a_i, s'_i, g_i) = 0, \forall s'_i : (s'_i \in a_i) \wedge (s'_i \in g_i)$. In our experiments, we set the $penalty$ reward to $-H$. In addition, in the situation in which a subgoal is tested and missed, we set the discount rate γ_i=0. This will ensure that the Bellman target resulting from the transition describing the missed

subgoal will be a fixed low value and will not depend on the next state term, which will vary as π_{i-1} changes.

Subgoal testing overcomes the issue of a level ignoring distant subgoals when updating the Q-function. Whenever level i proposes and then tests a subgoal a_i that requires more than H actions from the level below, the following transition is guaranteed to be created: *[state = s_i,action = a_i,reward = $-H$, next state = ignore, goal=g_i]*. The next state is ignored because the discount rate γ_i = 0 when a subgoal is tested and missed. This transition in turn will incentivize the Q-function to assign a low Q-value to the unfeasible subgoal.

At the same time, integrating subgoal testing still enables agents to learn the policies within the hierarchy in parallel. Whereas before the Q-function was trying to estimate the action-value function that assumed an optimal lower level policy hierarchy, $q_{\pi_i|\Pi^*_{i-1}}(s,a) = \mathbb{E}_{\pi_i|\Pi^*_{i-1}}[R_{i_{t+1}} + \gamma_i V_i(s')|s_{i_t} = s, a_{i_t} = a]$. Now the Q-function is trying to estimate the action-value function that is a weighted combination of the action-value function using the current lower level policy hierarchy and the action-value function using the optimal lower level policy hierarchy: $q_{\pi_i}(s,a) = \lambda \cdot q_{\pi_i|\Pi_{i-1}}(s,a) + (1-\lambda) \cdot q_{\pi_i|\Pi^*_{i-1}}(s,a)$. In other words, the Q-value of a subgoal action will now take into account whether the subgoal can actually be achieved by the current lower level policy hierarchy because the only difference between the reward functions is the penalty term. This updated Q-function still enables policies to be learned in parallel because subgoal actions a that had higher Q-values than other subgoal actions b under the former Q-function using Π^*_{i-1} should still have higher Q-values under the weighted Q-function, except potentially in the case in which the current lower level policy hierarchy can achieve subgoal b but not a. This overcomes the second shortcoming mentioned previously as the weighted Q-function incentivizes the policy to find paths to the goal that can actually be executed given the current lower level policy hierarchy Π_{i-1}.

3.5 UMDP TRANSFORMATION

We now formally define our strategy to learn policies in parallel in terms of the Universal MDP (UMDP) introduced earlier. Given the UMDP provided by the task $\mathcal{U}_{original} = (\mathcal{S}, \mathcal{G}, \mathcal{A}, T, R, \gamma)$, we convert $\mathcal{U}_{original}$ into a hierarchy of k UMDPS $\mathcal{U}_i = (\mathcal{S}_i, \mathcal{G}_i, \mathcal{A}_i, T_i, R_i, \gamma_i)$. These UMDP tuples are defined as follows:

$\underline{\mathcal{U}_0}$: This is the lowest level of the hierarchy. It has the same state set, action set, and state transition function as $\mathcal{U}_{original}$: $\mathcal{S}_0 = \mathcal{S}, \mathcal{A}_0 = \mathcal{A}$, and $T_0 = T$. The goal states for which $\mathcal{U}_0$ will be responsible for learning will be dictated by the UMDP one level higher, $\mathcal{U}_1$. However, given that every state is potentially a goal, the goal space is defined to be the state space: $\mathcal{G}_0 = \mathcal{S}$. The framework is flexible as to the reward function used at level 0, but we will use the shortest path reward function, in which a reward of 0 is granted if the next state, s', maps to the goal and -1 otherwise: $R_0(s'_0, g_0) = 0, \forall s'_0 : s'_0 \in g_0$ and $R_0(s'_0, g_0) = -1, \forall s'_0 : s'_0 \notin g_0$. In addition, the discount rate is set to 0 if the goal has been achieved, but remains γ otherwise: $\gamma_0(s'_0, g_0) = 0, \forall s'_0 : s'_0 \in g_0, \gamma_0(s'_0, g_0) = \gamma, \forall s'_0 : s'_0 \notin g_0$. The objective in $\mathcal{U}_0$ is to find the policy $\pi_0 : \mathcal{S}_0, \mathcal{G}_0 \rightarrow \mathcal{A}_0$ that maximizes the value function $v_{\pi_0}(s,g) = \mathbb{E}_{\pi_0}[\sum_{n=0}^{\infty} \gamma_0^n R_{0_{t+n+1}} | s_{0_t} = s, g_{0_t} = g]$ for the initial state and goal combinations that the policy π_1 from $\mathcal{U}_1$ requires.

$\underline{\mathcal{U}_i, 1 \leq i < k}$: These UMDPs represent the remainder of the hierarchy. The state space is the same as the state space in $U_{original}$: $\mathcal{S}_i = \mathcal{S}$. For all levels except for the top level, the set of goals is dictated by the level above. These goals can potentially be any state: $G_{i:1 \leq i < k-1} = \mathcal{S}$. For the top level, the goal set is provided by the task: $\mathcal{G}_{k-1} = \mathcal{G}$. The action space is the state space as these policies will output subgoal states for the next level to achieve: $\mathcal{A}_i = \mathcal{S}$. The state transition function consists of two transition functions: $T_i(s,a)$ = {(i) $T_{i|\Pi_{i-1}}(s,a)$, (ii) $T_{i|\Pi^*_{i-1}}(s,a)$}. The first transition function will be used for subgoal testing and will follow the current lower level policy hierarchy exactly. The second transition function will be used to generate hindsight transitions. Note that the hindsight action a passed to the second transition function must be generated by some other hierarchical behavior policy Π_{i_b}. The reward function is $R_i(a, s', g) = penalty, \forall s' : s' \notin a$, $R_i(a, s', g) = -1, \forall s' : (s' \in a) \wedge (s' \notin g)$, and $R_i(a, s', g) = 0, \forall s' : (s' \in a) \wedge (s' \in g)$. The penalty reward is only issued during subgoal testing. γ_i is set to 0 if a subgoal is tested and missed or if an action achieves the goal, but is otherwise γ from $\mathcal{U}_{original}$: $\gamma_i(a, s', g) = 0, \forall s' : (s' \notin a) \vee (s' \in g)$. The objective in each U_i is to learn a policy $\pi_i : \mathcal{S}_i, \mathcal{G}_i \rightarrow \mathcal{A}_i$ that maximizes the weighted value function $v_{\pi_i}(s,g) = \lambda \cdot v_{\pi_i|\Pi_{i-1}}(s,g) + (1-\lambda) \cdot v_{\pi_i|\Pi^*_{i-1}}(s,g)$.

3.6 INTEGRATING HINDSIGHT EXPERIENCE REPLAY

The UMDP transformation operation does enable agents to learn to solve a set of UMDPs in parallel, but each UMDP remains difficult due to the sparsity of the reward. The hindsight actions help each level discover feasible subgoal actions, but these subgoal actions do not necessarily help the level achieve its goal. As currently constructed, each level would need to its behavior policy to randomly achieve the goal in order to receive the sparse reward, but this can be unlikely in difficult tasks.

To make it easier for each level to learn a goal-conditioned policy in the sparse reward environments, we integrated Hindsight Experience Replay (HER)(Andrychowicz et al. (2017)) into every level of the hierarchy. This is implemented as follows. When a level i is given a goal from level $i+1$, level i will create a hindsight action transition for each action executed. Assuming that H hindsight actions were executed, these transitions will be of the form *[state =* s_{i_t}*, action =* s_{hind_t}*, reward =* r_{i_t}*, next state =* s_{i_t}*, goal =* g_{i_t}*]*, in which $t \in \{0, \ldots, H-1\}$. To incorporate HER, one or more copies of these transitions will be created and the reward and goal components will be erased. For each set of copies, a new goal will be selected from the set of *next state* transition elements and inserted into the set of copied transitions as the goal component. The reward $R_i(a_i, s_i', g_i)$ element will also be updated to reflect the new goal.

HER should significantly help each level learn an effective goal-conditioned policy because it guarantees that after every sequence of actions, at least one transition will be created that contains the sparse reward (in our case a reward of 0). These transitions containing the sparse reward will in turn incentivize the UVFA critic function to assign relatively high Q-values to the *(state, action, goal)* tuples described by these transitions. The UVFA can then potentially transfer these high Q-values to the actions that could help level i solve its tasks.

Algorithm 1 Hierarchical Actor-Critic (HAC)

for M episodes **do** ▷ Train for M episodes
  $s \leftarrow S_{init}$, $g \leftarrow G_{k-1}$ ▷ Sample initial state and task goal
  $train-level(k-1, s, g)$ ▷ Begin training
  Update all actor and critic networks
end for

function TRAIN-LEVEL($i :: level, s :: state, g :: goal$)
  $s_i \leftarrow s$, $g_i \leftarrow g$ ▷ Set current state and goal for level i
  for H attempts or until g_n, $i \leq n < k$ achieved **do**
    $a_i \leftarrow \pi_i(s_i, g_i)$ + $noise$ (if not subgoal testing) ▷ Sample (noisy) action from policy
    if $i > 0$ **then**
      Determine whether to test subgoal a_i
      $s_i^{'} \leftarrow train-level(i-1, s_i, a_i)$ ▷ Train level $i-1$ using subgoal a_i
    else
      Execute primitive action a_0 and observe next state $s_0^{'}$
    end if
    ▷ Create replay transitions
    if $i > 0$ and a_i missed **then**
      if a_i was tested **then** ▷ Penalize subgoal a_i
        $Replay\_Buffer_i \leftarrow [s = s_i, a = a_i, r = Penalty, s^{'} = s_i^{'}, g = g_i]$
      end if
      $a_i \leftarrow s_i^{'}$ ▷ Replace original action with action executed in hindsight
    end if
    ▷ Evaluate executed action on current goal and hindsight goals
    $Replay\_Buffer_i \leftarrow [s = s_i, a = a_i, r = \{-1, 0\}, s^{'} = s_i^{'}, g = g_i]$
    $HER\_Storage_i \leftarrow [s = s_i, a = a_i, r = TBD, s^{'} = s_i^{'}, g = TBD]$
    $s_i \leftarrow s_i^{'}$
  end for
  $Replay\_Buffer_i \leftarrow$ Perform HER using $HER\_Storage_i$ transitions
  return $s_i^{'}$ ▷ Output current state
end function

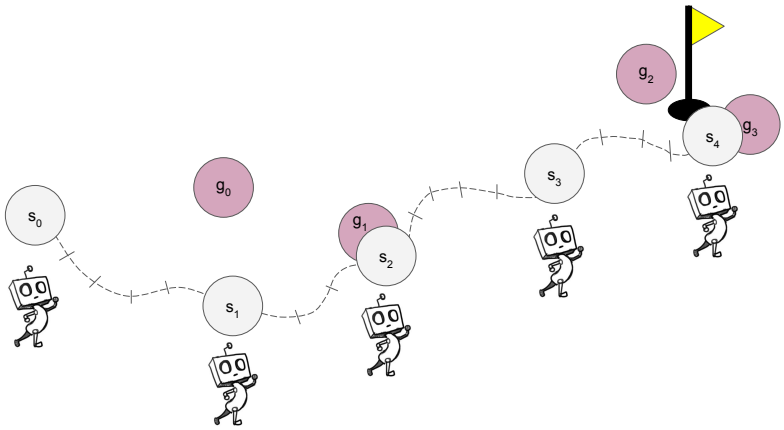

Figure 2: Figure shows an example episode trajectory for a simple toy example. The tic marks along the trajectory show the next states for the robot after each atomic action is executed. The gray spheres show the subgoal states reached in hindsight after at most T actions. The pink spheres show the original subgoal actions.

3.7 Hierarchical Actor-Critic Algorithm

Algorithm 1 shows the procedure used to train policy hierarchies in continuous domains, which we refer to as *Hierarchical Actor-Critic (HAC)*.

3.8 Example

This section walks through a simple toy example showing how our frameworks helps agents learn a set of policies in parallel by replacing both actions and goals. The agent in this task is a robot and its goal is to reach the yellow flag. The agent has two levels in its hierarchy and a maximum subgoal horizon parameter $H = 5$ actions. Thus, the agent's goal is to learn both a low and high-level policy that focus on action sequences that contain no more than 5 actions.

Figure 2 shows an example episode trajectory. Please note that all subscript numbers in the figure refer to the iteration number of the high-level policy. The episode begins with the robot in state s_0 when the high-level policy π_1 proposes its first subgoal state g_0. The robot decides that during this iteration, subgoal g_0 will not be tested. As a result, the low-level policy, π_0, is free to add exploration noise to the primitive actions it proposes. Level 0 then has at most $H = 5$ primitive actions (i.e., joint torques in this case) to achieve g_0. After each action, a transition will be created showing how the action from π_0 performed given the goal state it was provided. For instance, the first transition in this example may look like $[s = s_0, a = Joint\ Torques, r = -1, s' = first\ tick\ mark, g = g_0]$. In addition, a separate transition will be created and temporarily stored in order to be used for level 0's Hindsight Experience Replay. This transition will be of the form $[s = s_0, a = Joint\ Torques, r = TBD, s' = first\ tick\ mark, g = TBD]$. Per the example in Figure 2, the robot fails to achieve state g_0 after 5 attempts and lands in state s_1 instead. After this sequence of primitive actions, HER is performed for layer 0. The state s_1 can replace the TBD goal component in the extra hindsight transitions that were created and the reward would then be replaced with its appropriate value.

The completion of level 0's for loop represents a single action by level 1's policy π_1. Because its original subgoal g_0 was not achieved, π_1 will then record the hindsight state s_1 as the actual action that was taken. The transition $[s = s_0, a = s_1, r = -1, s' = s_1, g = Yellow\ Flag]$ will be added to level 1's replay buffer. The additional transition $[s = s_0, a = s_1, r = TBD, s' = s_1, g = TBD]$ will be stored until HER is performed for level 1.

Skipping ahead to when the robot is in state s_2, the high-level policy proposes subgoal g_2 and then decides to test the subgoal. The low-level policy then has $H = 5$ attempts without exploration noise to try to achieve g_2. In this scenario, the robot fails to achieve g_2. Indeed, state g_2 is too far away from s_2 to ever be achieved in 5 attempts. Because the subgoal was tested and the robot failed to achieve it, a penalty transition is created and passed to the replay buffer for level 1. Assuming the penalty is $-H = -5$, the transition would be of the form $[s = s_2, a = g_2, r = -5, s' = s_3, g = Yellow$

$Flag$]. The robot will also create the regular and hindsight transitions showing that state s_3 is a possible subgoal state from s_2. The regular transition is again of the form $[s = s_2, a = s_3, r = -1, s' = s_3, g = Yellow\ Flag]$. The episode then finishes with the agent achieving its goal. After the episode, HER is performed for level 1 and then all actor-critic networks are updated using an off-policy RL algorithm, such as DDPG (Lillicrap et al. (2015)).

From this single episode, both levels have learned important information. Using HER, level 0 has learned the joint torques that can achieve a variety of states. Similarly, by replacing the original proposed subgoals with the subgoal actions achieved in hindsight, level 1 has learned a sequence of subgoal states $(s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4)$ that fits its time scale and can lead the robot to a certain area in the goal space. The sequences of actions that each level is provided is short and thus can potentially be learned more quickly than a flat policy trying to learn a long sequence of atomic actions. Moreover, through subgoal testing, level 1 has also learned about subgoals that may be too ambitious. Using its UVFA, each level should be able to use this new information to better generalize to different goal states.

4 EXPERIMENTS

4.1 ENVIRONMENTS

We evaluated our framework in a variety of discrete and continuous tasks. The discrete tasks consisted of grid world environments. The continuous tasks consisted of the following simulated robotics environments developed in Mujoco (Todorov et al. (2012)): (i) Inverted Pendulum, (ii) UR5 Reacher, (iii) Cartpole, and (iv) Pick-and-Place. A video showing our experiments can is available at `https://www.youtube.com/watch?v=i04QF7Yi50Y`.

4.2 RESULTS

For each task we compared the performance of agents using policy hierarchies with 1 (i.e., flat), 2, and 3 levels. The flat agents used Q-learning with HER in the discrete tasks and DDPG with HER in the continuous tasks.

In all tasks, our approach significantly outperformed the flat agent. Figure 4 shows the performance graphs for each agent in each task. Each chart plots the average episode success rate for a given level of training for the 3 level agent (Red), 2 level agent (Blue), and 1 level agent (Green). For the discrete tasks, inverted pendulum, and UR Reacher task, the level of training is expressed in terms of the actual number of training episodes that have taken place. For the remaining tasks, the level of training is expressed in terms of test periods, in which each test period is separated by around 250 training episodes. For the discrete tasks, the charts averages data from 50 trial runs. The continuous tasks use data from 5-10 trial runs. Each chart also plots the areas that are 1 standard deviation from the mean success rate.

Our empirical results also support our claim that additional layers of hierarchy can improve sample efficiency because they can shorten the action sequence length that each policy needs to learn. In all of the discrete tasks and in the more challenging continuous tasks, such as UR5 Reacher and Cartpole, the agent using 3 levels outperformed the agent 2 levels.

4.2.1 BASELINE COMPARISON

We compared our approach HAC against another HRL technique, HIRO (Nachum et al. (2018)) because HIRO has shown it can outperform the other leading HRL techniques that can work in continuous state and action spaces: FeUdal Networks(Vezhnevets et al. (2017)) and Option-Critic. HIRO enables agents to a 2-level hierarchical policy. Like our approach, these policies are goal-conditioned and can be trained off-policy. In addition to being limited to 2 levels, the other major differences are that (i) HIRO does not use HER at either of the 2 levels and (ii) HIRO uses a different approach for handling the non-stationary transition functions. Instead of replacing the original proposed action with the hindsight action as in our approach, HIRO inserts the action from a set of actions that would most likely cause the sequence of *(state, action)* tuples that originally occurred at level 0, when the level 0 policy was trying to achieve the original subgoal. The set of potential replacement actions include the original proposed subgoal action, the hindsight action, and 8 other

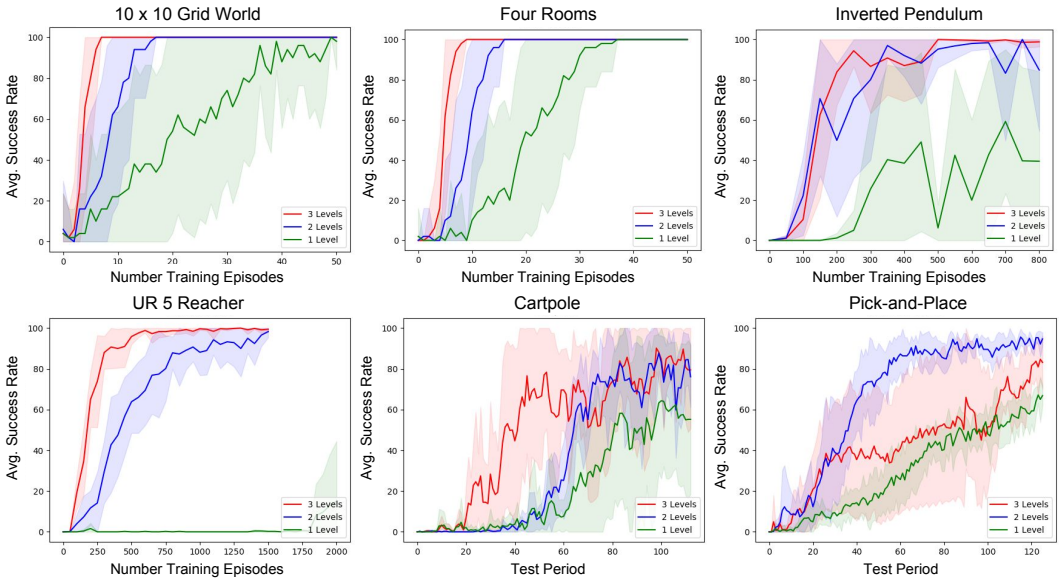

Figure 3: Figure presents results of our experiments. For each task, the average success rate is shown over time for a 3-level agent (Red), a 2-level agent (Blue), and a 1-level (Green). The error bars represent 1 standard deviation.

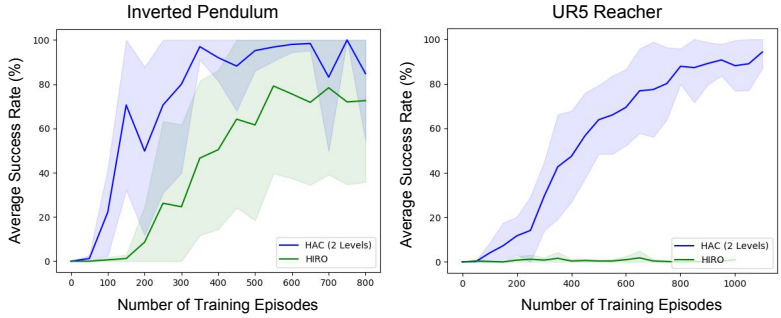

Figure 4: Figure compares the performance of HAC (2 Levels) and HIRO. The charts show the average success rate over 5 trials for each algorithm. The error bars represent 1 standard deviation.

actions sampled from a Gaussian around the hindsight action. We believe that although this is a clever way to enable the agent to learn using off-policy methods, it does not enable an agent to learn the policies within a hierarchy in parallel. When HIRO makes updates to the level 1 policy, these are still in respect to the current level 0 policy. When the level 0 policy inevitably changes, prior updates to the level 1 policy will become obsolete.

We compared the 2-level version of HAC to HIRO on the easier Inverted Pendulum task and the more difficult UR5 Reacher task. In both experiments, HAC outperformed HIRO. The out-performance was substantial in the UR5 Reacher task, as HIRO was unable to maintain a success rate ¿ 0%, while the 2-level version of HAC could achieve around a 90% average success rate in around 900 episodes. Figure 4 shows the results of the comparison. We attribute HAC's better performance to the use of HER, which can accelerate learning in sparse reward tasks, and our approach for learning policies in parallel.

5 Related Work

There have been a wide range of hierarchical reinforcement learning approaches that have been implemented. Most of these approaches either only work in discrete domains, require low-level controllers, or require a model of the environment [*Suttonet al.* (1999), *Dietterich* (1998), *McGovern&Barto* (2001), *Kulkarniet al.* (2016),

$Menacheet\ al.$ (2002), $Simseket\ al.$ (2005), $Bakker\&Schmidhuber$ (2004), $Wiering\&Schmidhuber$ (1997)]

There are a few other automated hierarchical RL techniques that can work in continuous domains. Konidaris & Barto (2009) proposed Skill-Chaining, a method that iteratively chains options from the end goal state to the start state. A key advantage that our framework has relative to Skill-Chaining is that our approach learns the subgoal options needed to reach a more distant goal state in parallel rather than incrementally, which can lead to improved sample efficiency. For instance, in the toy robot example provided above, agents using Skill-Chaining may use the experience to only learn an option to move from state s_3 to s_4. On the other hand, our approach can immediately begin to learn all of the low-level subgoal policies (i.e., $s_0 \rightarrow s_1, s_1 \rightarrow s_2, s_2 \rightarrow s_3, s_3 \rightarrow s_4$) in parallel. A key advantage of HAC relative to Option-Critic architecture (Bacon et al. (2016)) and FeUdal Networks (FUN) (Vezhnevets et al. (2017)) is that these methods are limited to hierarchies with only two levels, whereas our framework can support an arbitrary number of levels. This enables agents using our approach to divide the original task into a parallel set of even shorter action sequences.

6 Conclusion

We propose a new framework for effectively scaling reinforcement learning to long time horizon tasks involving sparse rewards. Instead of learning a single lengthy policy, our approach learns a hierarchy of limited policies that operate at different time scales. The policies are trained in parallel and end-to-end using a combination of a particular UMDP transformation and Hindsight Experience Replay. Our results show that our approach can significantly improve sample efficiency in both discrete and continuous domains.

References

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *NIPS*, 2017.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. *CoRR*, abs/1609.05140, 2016. URL http://arxiv.org/abs/1609.05140.

Bram Bakker and Jürgen Schmidhuber. Hierarchical reinforcement learning with subpolicies specializing for learned subgoals. In *Neural Networks and Computational Intelligence*, pp. 125–130. IASTED/ACTA Press, 2004.

Thomas G. Dietterich. The maxq method for hierarchical reinforcement learning. In *In Proceedings of the Fifteenth International Conference on Machine Learning*, pp. 118–126. Morgan Kaufmann, 1998.

George Dimitri Konidaris and Andrew G. Barto. Skill chaining : Skill discovery in continuous domains. 2009.

Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 29*, pp. 3675–3683. Curran Associates, Inc., 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. *CoRR*, abs/1509.02971, 2015. URL http://arxiv.org/abs/1509.02971.

Amy McGovern and Andrew G Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In *ICML*, volume 1, pp. 361–368, 2001.

Ishai Menache, Shie Mannor, and Nahum Shimkin. Q-cut - dynamic discovery of sub-goals in reinforcement learning. In *ECML*, 2002.

Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. *CoRR*, abs/1805.08296, 2018.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In Francis Bach and David Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1312–1320, Lille, France, 07–09 Jul 2015. PMLR. URL http://proceedings.mlr.press/v37/schaul15.html.

Jürgen Schmidhuber. Learning to generate sub-goals for action sequences. *Artificial Neural Networks*, pp. 967–972, 1991.

Özgür Simsek, Alicia P. Wolfe, and Andrew G. Barto. Identifying useful subgoals in reinforcement learning by local graph partitioning. In *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005*, pp. 816–823, 2005. doi: 10.1145/1102351.1102454. URL http://doi.acm.org/10.1145/1102351.1102454.

R. S. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: a framework for temporal abstraction in reinforcement learning. *Artificial Intelligence Journal*, 112:181–211, 1999.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning : An Introduction*. MIT Press, 1998.

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. *CoRR*, abs/1703.01161, 2017. URL http://arxiv.org/abs/1703.01161.

Marco Wiering and Jürgen Schmidhuber. Hq-learning. *Adaptive Behaviour*, 6(2):219–246, 1997.

7 APPENDIX

7.0.1 UMDP STATE TRANSITION FUNCTION

Algorithm 2 UMDP $\mathcal{U}_i$ Transition Function: $T_{i|\Pi_{i-1}}(state, action)$

Input: state s, action a,
Output: next state

return $Execute - H - Actions(s, a, i-1, H)$ ▷ Execute $\leq H$ actions using policy π_{i-1}

function EXECUTE-H-ACTIONS($s :: state, a :: action, i :: level, itr :: iteration$)
  s' = $T_i(s, \pi_i(s, a))$ ▷ Execute 1 action using policy π_i
  itr -= 1 ▷ Decrement iteration counter
  if $itr == 0$ or $s' \in g, \forall g \in \{g_i, ..., g_{k-1}\}$ **then**
    return s' ▷ Return next state if out of iterations or goal achieved
  else
    return $Execute - H - Actions(s', a, i, itr)$ ▷ Execute another action from π_i
  end if
end function

7.1 SAMPLE EPISODE SEQUENCES

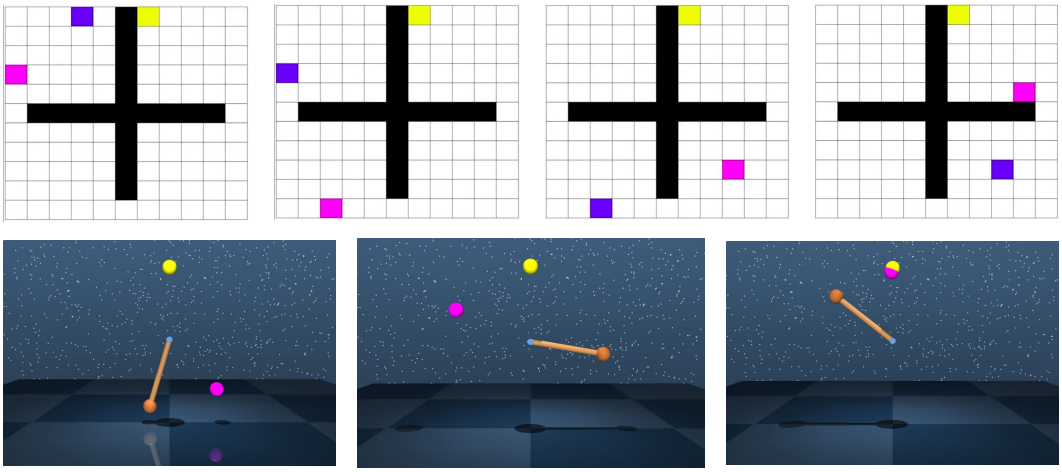

Figure 5: Episode sequences from the four rooms (Top) and inverted pendulum tasks (Bottom). In the four rooms task, the agent is the blue square, the goal is the yellow square, and the learned subgoal is the purple square. In the inverted pendulum task, the goal is the yellow sphere and the subgoal is the purple sphere.