HIERARCHICAL SELF-IMITATION LEARNING IN SINGLE-AGENT SPARSE-REWARD ENVIRONMENTS

By

Neeloy Chakraborty

Senior Thesis in Computer Engineering

University of Illinois at Urbana-Champaign

Advisor: Professor Katherine Driggs-Campbell

May 2021

Abstract

Reinforcement learning problems with sparse and delayed rewards are challenging to solve because the algorithms explore environments to gain experience from high performing rollouts. Classical methods of encouraging exploration during training such as ϵ-greedy and noise-based exploration are not adequate on their own to explore large state spaces (Fortunato et al., 2018). Self-imitation learning (SIL) has been shown to allow an agent to learn to mimic high performing, long-horizon trajectories, but SIL is heavily reliant on exploration to find such trajectories (Oh et al., 2018). On the other hand, hierarchical learning (HL) may be unstable during training but incorporates noise and failures that effectively explore the environment and may learn tasks with higher sample efficiency (Levy et al., 2019).

This thesis presents a single agent reinforcement learning algorithm that combines the effects of SIL and HL – Generative Adversarial Self Imitation Learning + Hierarchical Actor-Critic (GASIL+HAC). GASIL+HAC represents the policy as multiple trainable levels of Deep Deterministic Policy Gradient (DDPG) optimizers from Lillicrap et al., (2016), where the purpose of the higher-level policies is to set waypoints to guide the lower-level policies to receive the highest cumulative return. The highest-level policy of the hierarchy is trained with GASIL on the sparse environment reward to set goals that imitate past well-performing trajectories, while the lower levels are trained on an artificial reward signal to set intermediate goals and achieve the desired high-level path.

We perform experiments in OpenAI's Multi-Agent Particle Environment in sparse and delayed reward stochastic scenarios to identify benefits and hinderances of GASIL+HAC compared to DDPG, GASIL, and HAC in sample efficiency, generalizability, exploration, and goal reachability. Through these experiments, we find that GASIL+HAC has the potential to increase sample efficiency in stochastic tasks and increase the number of explored states during training. However, there is an inherent increase in instability of training hierarchical methods and SIL-based methods are still highly dependent on exploration to find high-return trajectories. Further experiments over several more seeds must be run to come to a complete conclusion on the effectiveness of the proposed algorithm.

Subject Keywords: reinforcement learning; sparse/delayed rewards; self-imitation learning; hierarchical learning

Acknowledgments

This work would not have been possible without the immense support of several outstandingly intelligent, caring, and funny individuals who made the thesis writing process a joyful experience. I would like to first and foremost thank my research advisor, Professor Katherine Driggs-Campbell, for discussing this project with me every week and providing me with invaluable professional and academic advice. My graduate student mentor, Shuijing Liu, truly took me under her wing from an early stage and I cannot express how incredibly thankful I am for her support and ideas throughout this whole process. Additionally, this work was a collaboration with an Illinois alumnus, Kshitij Gupta, with whom I have spent countless hours discussing novel ideas to move past the hundreds, if not thousands, of problems faced during this project. I look forward to continuing this project with him soon. Other graduate students of the Human-Centered Autonomy Lab genuinely made me feel like a part of the Lab family, and I am truly excited for future collaborations. Last, but of course not least, I would like to thank my parents, Savitri and Subhasish Chakraborty, who have always suggested I should explore every facet of learning available to me.

Contents

1. Introduction

This section introduces the motivation behind this work and contributions made through this project.

1.1. Motivation

Reinforcement learning has been applied to countless challenging sequential decision-making problems where an agent in an environment receives an observation, takes an action in the environment, and receives a quantitative reward stating how well that decision performed at that point in time. These problems are usually formulated such that the overall goal of the agent is to maximize the cumulative reward received over an entire episode within the environment. When these rewards are given to the agent at sparse intervals rather than every time step, it is difficult to train the agent to correlate a current observation with an optimal action. This problem amounts to that of exploration – if the agent never discovers a state-action pair that produces a concrete reward, then it cannot learn how to act optimally. One method shown to solve such long-horizon tasks well is self-imitation learning.

Self-imitation learning trains an agent to mimic past well-performing trajectories in the environment [1]. Specifically, the sparse rewards given to the agent are replaced by a dense reward depending on how closely the agent chose an action that resembles the past trajectory. This learning method performs well in long-horizon tasks and even may generalize in stochastic environments but is heavily dependent on an exploration method to find good trajectories.

Hierarchical learning may provide a means by which the agent can perform further exploration in the environment [2]. A multi-level hierarchy may be trained in tandem such that the highest-level sets goals in the environment every few timesteps that lower levels act to attempt to achieve. In early stages of training, the lower levels have not learned to act optimally to achieve those goals set, so the agent explores randomly in the environment, building up this experience. Hierarchical learning has the promise of speeding up training time depending on number of levels used and increasing exploration. However, hierarchical learning fails for long horizon tasks if not using several levels, and instability in training increases as the number of levels increases.

Hierarchical learning combined with self-imitation learning may have the unique ability to increase exploration and decrease training time in long horizon, stochastic tasks with sparse or delayed rewards. This provided motivation to develop the algorithm introduced in this thesis – GASIL+HAC. GASIL+HAC trains a hierarchy of low-level, goal-conditioned policies using HL, while a top-level goal-setter is trained on SIL. With this formulation, we would expect to see the sample efficiency and exploration benefits of

HL combined with the ability for SIL to learn to imitate long-horizon trajectories. GASIL+HAC, was applied to two sparse reward scenarios against three baselines: the vanilla, flat optimizer – DDPG; DDPG with a form of self-imitation learning – GASIL; and a form of hierarchical learning – HAC. The experiment simulations were developed in OpenAI's Multi-Agent Particle Environment to compare the methods' sample efficiency, generalizability, and exploration capabilities [3].

1.2. Contributions

This work (1) presents challenging scenarios in OpenAI's Multi-Agent Particle Environment with sparse or delayed rewards; (2) proposes a novel algorithm and hierarchical architecture to specialize in training within sparse reward scenarios; (3) presents initial results on the effectiveness of the method in comparison to baselines in three experiments. This thesis is organized as follows: chapter 2 discusses related works; chapter 3 provides the formulation of the reinforcement learning problem statement, and the background behind the three algorithms GASIL+HAC is built upon; chapter 4 discusses the developed algorithm and architecture; chapter 5 discusses the experimental setup and results of experiments; and chapter 6 provides tentative conclusions based on results and appropriate future work.

2. Related Works

This section reviews past literature related to the work presented in this thesis in the topics of exploration, imitation learning, and hierarchical learning.

2.1. Exploration Methods

In reinforcement learning, a good exploration policy is the key to finding trajectories with high return. If the agent never discovers well performing trajectories, it cannot begin to even learn to act optimally in its environment. On that same note, it is also important to balance exploring an environment with evaluating how well the current policy performs and learning from its current experience. Classic methods of exploration like ϵ-greedy deal with the exploration and exploitation trade-off problem by assuming that the policy will have found high return trajectories early in training so the policy should perform fewer exploration rounds later in training. However, those classical methods alone are insufficient to search the state-space of a large environment [4] – [5]. Pathak et al., (2017) attempts to introduce an intrinsic curiosity reward signal to a single-agent environment with sparse rewards to help the agent explore the environment and learn skills. The proposed Intrinsic Curiosity module (ICM) consists of inverse & forward dynamics models and the intrinsic reward is the prediction error in predicting the next state feature given a current state feature [6]. While an agent can cover several scenes in a large environment, this work is hindered by the noisy TV problem, where the agent will get stuck when its observation is highly stochastic. Random Network Distillation replaces the curiosity reward with the prediction error between a fixed network and a learning network given an observation so an agent will not face the noisy TV problem, but the method has very low sample efficiency [7]. Anderson et al., (2018) introduces the Dreaming Variational Autoencoder, which generates artificial experience for an agent to be trained on based on seen observations. The method shows promising results in smaller environments, but the generator cannot produce accurate predictions of unvisited areas [8]. DIAYN uses meta reinforcement learning to enable an agent to learn a diverse set of skills conditioned on a learnt latent space of the observation space, and it would be interesting to use this procedure to bootstrap the learning process of hierarchical or imitation learning [9].

2.2. Self-Imitation Learning

Imitation learning has been shown to enable agents to learn to mimic long-horizon trajectories. Relay Policy Learning applies imitation learning to a hierarchical policy in long-horizon robotics tasks, but the method relies on a set of given expert demonstrations in the environment [10]. The original Self-Imitation Learning algorithm paved the way for a new subset of reinforcement learning algorithms to be built from, as an agent now learns to imitate its own past well-performing trajectories [1]. SIL methods are reliant on a good exploration policy to have relevant demonstrations to learn from. Go-Explore uses SIL in a two-stage learning process consisting of “explore until solved,” and “robustification” [11]. In this method, an agent learns to reach a state in the most efficient manner possible and continues to explore from that point. However, the method requires an environment that can be reset to any specific state and is deterministic. DTSIL from Guo et al., (2021) entices the agent to learn from a diverse set of trajectories from its past rollouts by storing trajectories with high uncertainty (or low number of visits) in a trajectory buffer. Again, this method’s performance is contingent on a good exploration policy and does not work well in stochastic environments [12]. More recently developed, SQIL is discussed to be less brittle during training and would an interesting approach to try in this work [13]. GASIL [14], the method of SIL used in this thesis is discussed later in chapter 3.3.

2.3. Hierarchical Learning

Nachum et al., (2019) provides an interesting perspective on the successes shown by some hierarchical methods and attributes their improved performance to increased exploration. This effect implicitly arises due to effective management of exploration versus exploitation during training [2]. However, if multiple levels’ policies could be stably trained in parallel without having to pretrain any networks, then the agent would enjoy the benefits of higher sample efficiency in addition to exploration. One such sample efficient method, and the hierarchical method utilized in this thesis, is known as HAC [15], and is discussed further in chapter 3.4.

3. Background

This section introduces the problem formulation that reinforcement learning optimizers attempt to solve in this paper and the three fundamental building block algorithms that GASIL+HAC is built upon: DDPG, GASIL, and HAC. Several of the formulations in this section are based on writing from OpenAI's Spinning Up project [16].

3.1. Problem Formulation

We formalize the single agent reinforcement learning optimization problems for the environments presented in this paper as Markov decision problems (MDPs). An MDP is represented by the following tuple in Eq. (1).

$$< S, A, P, R, \gamma, S_0 > \tag{1}$$

In Eq. (1), S is the set of all reachable states by an agent in an environment and A is the set of all actions the agent can take to go to any next state s'. P is the state transition probability function that transitions the agent to a next state given a current state and action: $P(s'|s, a)$. The agent begins in a given environment at state s_0 within a set of start states S_0. At every timestep, the agent receives a state observation s from the environment, takes an action a to transition to a next state s', and receives a one-step reward of r given by the environment reward function R. $\gamma \in [0,1]$ is a tunable discount factor that weights the importance of long-horizon rewards. The return is then the cumulative sum of discounted rewards over one complete trajectory, as written out in Eq. (2).

$$R_T = \sum_{t=0}^{\mathrm{T}} \gamma^t \, r_t \tag{2}$$

An agent takes actions in its environment every timestep according to its policy π, which may be parameterized by some function or neural network θ, as seen in Eq. (3).

$$a_t = \pi_\theta(s_t) \tag{3}$$

The value function when an agent starts in a state s and follows a policy π is the expected return the agent will receive, written out in Eq. (4).

$$V^\pi(s) = \mathrm{E}[R_T | s_0 = s] \tag{4}$$

Similarly, the action-value function is the expected return of the agent if the agent were to begin at state s, take action a, and follow policy π, as seen in Eq. (5).

$$Q^{\pi}(s,a) = \mathrm{E}[R_T | s_0 = s, a_0 = a] \tag{5}$$

The goal of reinforcement learning is to learn an optimal policy π^* that receives the highest return over all possible parameterizable policies π. If the agent follows the optimal policy, then Eq. (6) and Eq. (7) can be derived by taking the maximum over the entire expectation.

$$V^*(s) = \max_{\pi} \mathrm{E}[R_T | s_0 = s] \tag{6}$$

$$Q^*(s,a) = \max_{\pi} \mathrm{E}[R_T | s_0 = s, a_0 = a] \tag{7}$$

The optimal policy π^* always takes the optimal action every timestep, so if Q^* is known, then the optimal action a^* in Eq. (8) can be found by taking the argmax over all actions for the current state.

$$a^*(s) = \mathrm{argmax}_a Q^*(s,a) \tag{8}$$

3.2. Deep Deterministic Policy Gradient

There are several methods in existence that attempt to solve for the optimal policy π^*. The optimizer used in this paper is known as Deep Deterministic Policy Gradient (DDPG) [17]. DDPG attempts to learn to approximate both $Q^*(s,a)$ and a^* but only works in environments with a continuous action space. At every timestep, the agent observes state s, takes action a according to a parameterized policy θ with added noise to induce exploration in the environment, and receives a reward r, next state s', and done signal d. The tuple $< s, a, r, s', d >$ is stored into a replay buffer [18]. Then, we sample a random batch of transitions from the replay buffer. This sample is first used to update the network ϕ used to approximate $Q^*(s,a)$ according to the Bellman optimality equation for $Q^*(s,a)$, which is stated in Eq. (9).

$$Q^*(s,a) = \mathrm{E}[r(s,a) + \gamma \max_{a'} Q^*(s',a')] \tag{9}$$

Because ϕ is an approximation of $Q^*(s,a)$, to calculate the loss $L(\phi)$ for ϕ in Eq. (10), we can take the mean square error between the estimated action-value output of ϕ for the current state-action pair, and the estimated $Q^*(s,a)$ value according to the Bellman equation from Eq. (9):

$$L(\phi) = \mathrm{E}\left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d)\max_{a'} Q_\phi(s',a')\right)\right)^2\right] \tag{10}$$

After the Q-function update, we can update the policy θ directly using gradient ascent because a^* is the argmax of $Q^*(s,a)$ for a given state s. This update is also the reason for why DDPG only works in continuous action spaces: it is difficult to differentiate with respect to discrete actions.

3.3. Generative Adversarial Self-Imitation Learning

As stated earlier, when the environment gives sparse or delayed rewards, it becomes much more difficult for the optimizer to learn an optimal policy. Generative Adversarial Self-Imitation Learning (GASIL) is a form of learning that attempts to assist the optimizer to update the policy to mimic past well performing trajectories [14]. In addition to a normal replay buffer that stores uncorrelated one-step transitions, a new expert trajectory buffer is added. At the end of every episode, the current complete trajectory is pushed to the expert buffer according to how much return that trajectory received. Specifically, trajectories with higher return should be kept in the expert buffer while those with low return should be thrown out.

During every update, expert trajectory samples are taken from the expert buffer and normal uncorrelated transitions are taken from the normal replay buffer. A discriminator in the form of a neural network is trained on a binary cross-entropy loss to solve a classical classification problem where expert transitions should receive an output reward of one, whereas non-expert transitions should receive a reward of zero. Then, to update the Q-function and policy approximators, we pass the normal sample of transitions through the discriminator to assign new rewards to each transition. These newly labeled transitions are used to update the approximators via the DDPG algorithm described in the previous section. The discriminator, policy, and critic experience pushing, sampling, and updating is described in Figure 1.

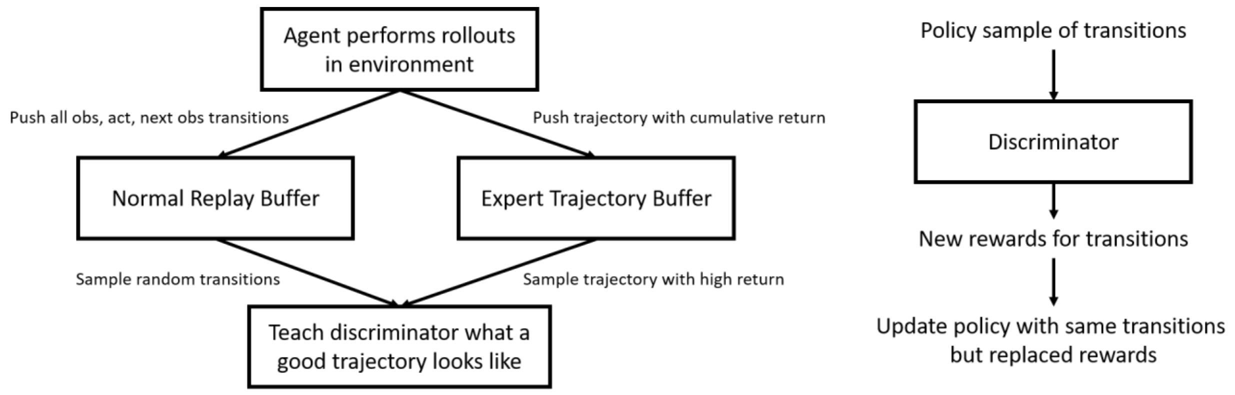

Agent performs rollouts in environment
Push all obs, act, next obs transitions
Push trajectory with cumulative return
Normal Replay Buffer
Expert Trajectory Buffer
Sample random transitions
Sample trajectory with high return
Teach discriminator what a good trajectory looks like
Policy sample of transitions
Discriminator
New rewards for transitions
Update policy with same transitions but replaced rewards

Figure 1. Overall experience storage and training process of discriminator and policy in GASIL.

Over time, the expert replay buffer should hold better performing trajectories, the discriminator should be able to differentiate between poor and well-performing trajectories, and the policy itself should learn to mimic those well-performing trajectories. GASIL assists in converting a sparse reward problem to a dense one because of the discriminator output, and it has been shown that GASIL can train well in long-horizon, stochastic environments [14]. Though, once again, if the agent never discovers any well-performing trajectories to fill its expert buffer with, then the agent will never learn to act well. Therefore, GASIL is heavily dependent on a strong method of exploration.

3.4. Hierarchical Actor-Critic

Hierarchical Actor-Critic (HAC) is a hierarchical reinforcement learning algorithm that improves sample efficiency in training [15]. Sample efficiency is a metric in reinforcement learning that measures how quickly an algorithm can learn to solve a problem versus how many episode rollouts (or sampled experience) it took to get there. Algorithms with higher sample efficiency can learn to solve a problem quicker than others. In contrast to a flat algorithm like DDPG or GASIL which learns a single policy conditioned on an observation which directly outputs a low-level action, HAC learns a multi-level hierarchy of policies – each of whom are responsible for a smaller subtask within the entire trajectory. This multi-level method is claimed to increase sample efficiency.

Specifically, HAC is a k-level algorithm that trains k DDPG policies conditioned on a user-defined goal in the observation space of the agent. The goal of HAC is not to receive the highest return from the environment, but to learn to reach that user-defined goal within the observation space of the agent. The top-level policy takes the current observation of the agent and the user-defined goal, and outputs a new subgoal within the observation space of the agent. Similarly, mid-level policies take the goal output by higher levels and the current observation of the agent to generate a goal for their lower levels. The lowest level policy takes the goal output by the level just above it and the current observation of the agent to generate a low-level action in the actual action space of the agent. An example of a 3-level HAC hierarchy is shown in Fig. 2.

The lower-level policies have at most H attempts to reach the goals set by higher-level agents. If a lower-level policy reaches the goal given by a higher-level within some threshold in under H attempts, then the higher-level prematurely changes its goal output. Otherwise, the higher-level changes its goal output on the final attempt. By constraining lower levels to have at most H attempts to reach the goal output by the higher level, the policies learn to solve smaller sub-tasks which promotes sample efficiency.

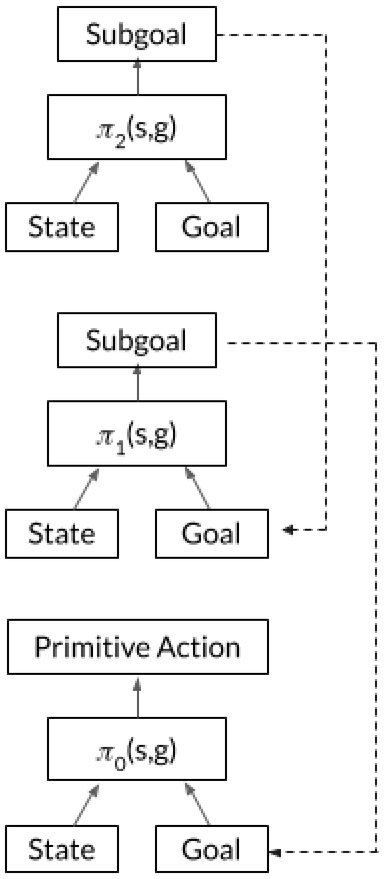


Figure 2. An example of a 3-level architecture for HAC. Top level policy π_2 takes in handcrafted user goal and outputs goal in same space as observation space. Bottom level policy π_0 takes in goal output from π_1 and outputs a primitive action [15].

However, training a multi-level hierarchy of several policies conditioned on one another in parallel is highly unstable. This issue is known to arise due to non-stationary functions. A higher-level agent may give the same goal to an agent given the same original state, but because the lower-level policy is learning concurrently, it may take different trajectories to reach the same goal over time. Similarly, there are multiple methods to reach the same end state from a current state – each of which may have different returns. The Q-function update is dependent on a stationary set of samples from the same distribution, and it will be difficult to converge to an optimal policy if the samples are not stationary.

Every policy is trained from experience pulled from their own replay buffer. Each transition pushed to a replay buffer is a tuple consisting of the agent's observation, the level's action, reward received, next observation, goal input, discount factor γ, and a done signal: $< s, a, r, s', g, \gamma, d >$.

HAC deals with the non-stationary problem by introducing three types of transitions pushed to the replay buffers of each of the policies: Hindsight Action Transitions (HATs), Hindsight Goal Transitions (HGTs), and Subgoal Testing Transitions (STTs). HATs are pushed for non-base level policies and assume that lower-level policies are already acting optimally. When the non-base level is about to change its goal output, the level exchanges its output goal action for the actual state reached by the lower level. The reward is -1 if the goal input is not reached and 0 otherwise. HATs for the base level do not replace the actions outputted by the lowest level, but the rewards are still conditioned on whether the goal was reached. While HATs help in solving the non-stationary problem, they also make rewards sparse. HGTs attempt to provide more dense rewards by replacing the goal input to a level over a complete goal duration with the actual final state reached by the agent at the end of the goal duration. Every HGT's reward is -1 except for the final transition which becomes 0. HGTs are like the Hindsight Experience Replay of Andrychowicz et al., (2017). They were developed to stabilize the training of goal-conditioned policies [19]. Finally, SGTs only occur every goal duration with some random probability and penalizes a higher level for setting an unreachable goal. If a goal duration is randomly selected for Subgoal Testing and the lower level does not reach the goal given by the higher level within the goal duration, then the higher level is penalized with a reward of -H.

In summary, HAC introduces a more sample efficient method to train long horizon tasks with multiple policies in parallel and uses three transition types to deal with the non-stationary problem introduced by hierarchical learning.

4. Methodology

This section formally introduces the GASIL+HAC architecture.

4.1. Introduction

GASIL+HAC aims to combine the effects of GASIL and HAC together into one algorithm. While GASIL can convert sparse problems into dense ones via a discriminator and mimic well-performing, long-duration trajectories, it is highly dependent on a strong exploration policy to gather good experience. On the other hand, HAC increases sample efficiency the greater number of levels used, but it also increases the number of network parameters and increases instability. The noise induced by failures within HAC can assist the exploration process of GASIL and splitting up the task to several sublevels can increase the sample efficiency of the algorithms. The goal of GASIL+HAC, as explained in the coming sections, is to train a top-level policy that can delegate subgoals in a manner that imitates past well-performing trajectories explored by the agent during training, while the lower-level policies split the goals into smaller tasks to achieve the goal set by the highest level in the most efficient way possible.

4.2. Experience Replay Buffers

Similar to HAC, every level in GASIL+HAC has its own normal replay buffer where one-step transitions in the form $< s, a, r, s', g, \gamma, d >$ are stored. The top-level normal replay buffer only stores HATs and SGTs; lowest-level buffer stores only HATs and HGTs; and mid-level replay buffers store all three HAC transition types. In addition to the normal replay buffer, a priority expert replay buffer stores the top N performing trajectories of the agent during training for the top-level to learn to mimic. These trajectories are a list of tuples of the form $< s, s' >$ where s is the original observation of the agent and s' is the actual next state of the agent after H^{K-1} timesteps in the environment. These trajectories are pushed to the priority expert replay buffer according to the total return of that trajectory. The expert buffer is explained further below in the self-imitation section.

4.3. Top Level Architecture and Transitions

Whereas the top-level architecture of HAC was trained on an artificial reward signal conditioned on a user-defined goal in the observation space of the agent and the current observation itself, GASIL+HAC trains the top-level policy and critic on the environment reward. This means that GASIL+HAC can be extended to any environment with a reward signal directly, rather than having to handcraft a goal condition for every new environment. The HAT and SGT formulations that exist in the original HAC implementation had to be extended to the GASIL+HAC algorithm to ensure the nonstationary problem was still addressed. Specifically, when the lower level has completed H attempts at reaching the current goal output of the top-level or the lower level reaches the goal before H attempts, the top-level is signaled to change its goal in the next timestep. At this moment, an HAT is pushed to the top-level's buffer with the goal action replaced by the actual next state reached by the lower-level. Similarly, if Subgoal Testing was on, an SGT is pushed to penalize the top-level if the lower level did not reach the goal set. However, because the top-level is no longer conditioned on a goal input, HGTs are no longer pushed to the top-level normal replay buffer.

4.4. Discriminator and Self-Imitation Learning Process

The goal of the top-level policy is to learn to produce goals that imitate past well-performing trajectories. This goal is encouraged by way that GASIL is conditioned on actual well-performing trajectories from the agent. The naïve implementation to store expert trajectories would be to append a $< s, s' >$ HAT tuple to an episode trajectory array every time the top-level is about to change its goal output. This choice would lead the expert replay buffer to potentially hold trajectories of different lengths if the top-level prematurely changes its goal output when its lower level reaches the goal before the goal duration is complete. And thus, it becomes difficult to update a policy from trajectories of varying lengths. To fix this problem, we decided to append a transition to the episode trajectory every H^{K-1} timesteps, which would be the normal timestep when the top-level would change its goal output if its lower level would not prematurely reach the given goal. This choice ensures all trajectories in the expert replay buffer have the same length.

Like in GASIL, a discriminator is trained to differentiate well-performing trajectories from poor ones by solving a classification problem between trajectories sampled from the expert buffer and those sampled from the top-level's normal replay buffer. When it is time to update the top-level critic and policy, a sample is taken from the normal replay buffer of the top-level, its rewards are replaced with the output of the discriminator, and the parameters are updated according to the DDPG update.

4.5. Lower Levels in GASIL+HAC

The non-top levels of GASIL+HAC are trained in the same way as vanilla HAC. The lowest level only pushes HATs and HGTs to its replay buffer while middle levels push all three transition types to their buffers. Again, these non-top levels are trained on an artificial reward signal defined by whether the lower level reached the goal set by its higher level within the given time constraint and goal threshold.

4.6. Complete Architecture

Given a K-level hierarchy to be trained in GASIL+HAC, we can define an upper-bound on how many networks will be required to train with the algorithm. Every level of the architecture will have an action-value approximator and a policy approximator. In addition, because the top level is trained on GASIL, there will be one discriminator network required. Overall, for a K-level hierarchy, there will be a maximum of $2k + 1$ networks required to train the architecture. Parameter sharing of critic networks can assist in decreasing the number of networks, since only one critic is required to approximate the value of the actions taken by all policy networks. Figure 3 shows an arbitrary K-level hierarchy of policies in GASIL+HAC and can be compared against the hierarchy developed in HAC shown in Fig. 2. Figure 4 summarizes the steps of GASIL+HAC and the networks and buffers utilized at each step.

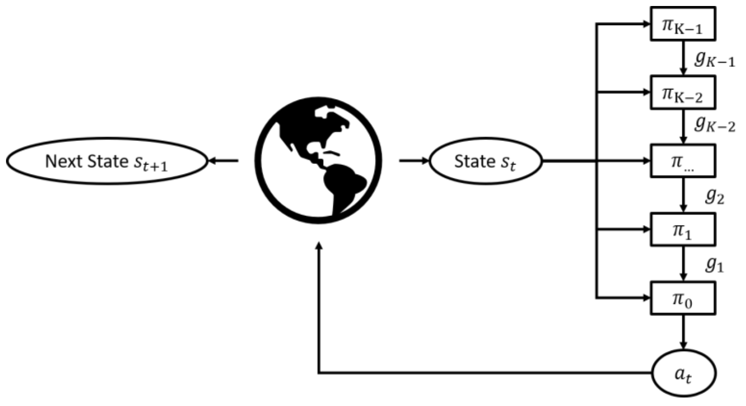


Figure 3. An arbitrary K-level hierarchy architecture in GASIL+HAC. Notice how top-level policy π_{K-1} only takes the current state of the agent as an input to generate a goal. All other levels follow a similar architecture to HAC.

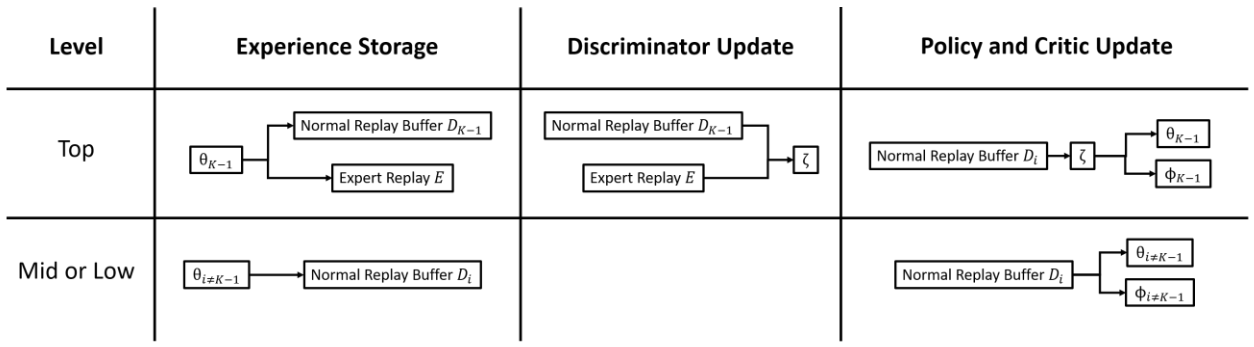


Figure 4. A summary of the experience storage, discriminator update, and policy/critic update processes in GASIL+HAC.

4.7. Algorithm

The complete algorithm for GASIL+HAC can be found in Appendix A in Fig. 24.

5. Results

This section demonstrates the two simulated scenarios that GASIL+HAC was trained on and results of training against DDPG, GASIL, and HAC. We also discuss the role of HAC in exploration by visualizing trajectories in the environment.

5.1. Simulation Environments

DDPG, GASIL, HAC, and GASIL+HAC were evaluated in two sparse reward scenarios titled, the *Simple* scenario and the *Simple Order* scenario, respectively. These scenarios were developed in OpenAI's Multi-Agent Particle Environment [3]. In the *Simple* scenario, shown in Fig. 5, our grey agent receives an observation consisting of its relative x- and y-position from a red landmark at every timestep. If the grey agent's position comes within 0.5 distance to the red landmark, it receives a reward of 1.0. Otherwise, the agent receives a reward of 0.0. In both the *Simple* and *Simple Order* scenarios, the agent takes a velocity action in the environment to accelerate itself in a particular direction, and the scenarios complete after a fixed number of timesteps. The *Simple Order* scenario is made to be an even more sparse and long-duration problem to solve. Given a set number of colored landmarks, the agent receives a reward of 1.0 every time it reaches a landmark in the correct order. For example, in Fig. 6, the agent receives a maximum return of 3.0 over one complete episode by passing over the left green, bottom black, and right red landmarks in that order (1.0 reward is given every time the agent passes over the next correct ordered landmark). If the agent passes over a different landmark from the next landmark in the golden solution, then a reward of 0.0 will always be returned for the rest of the episode. The observation space of the agent is a list of the relative positions of the agent to all landmarks.

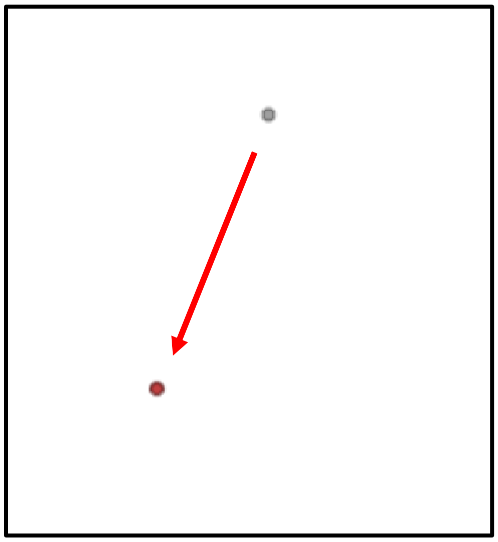


Figure 5. The *Simple* Scenario with a grey agent and red landmark.

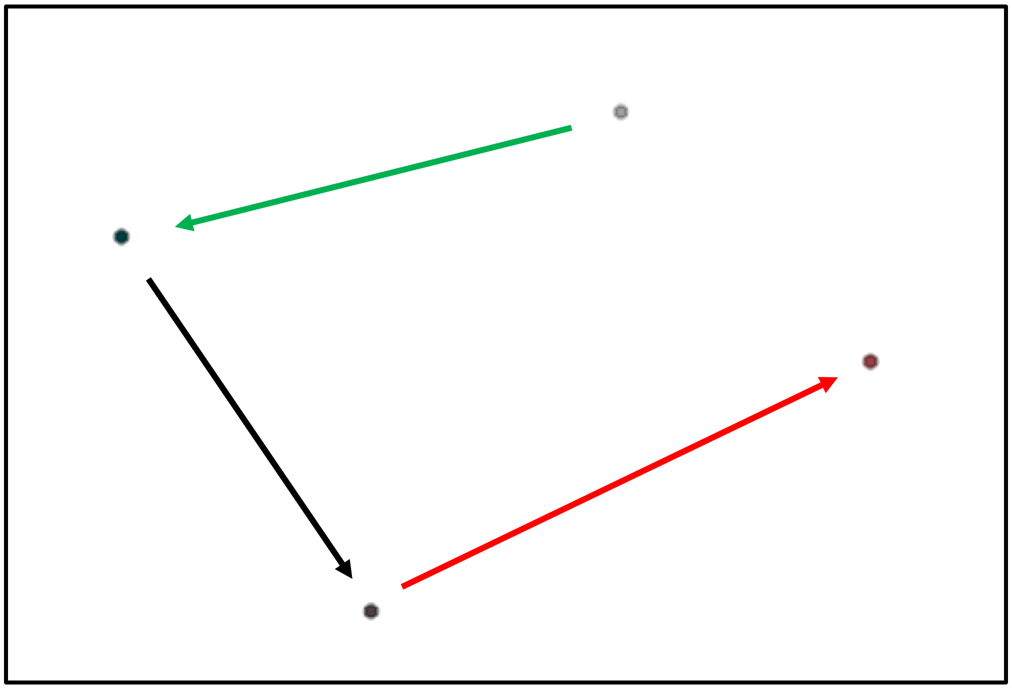


Figure 6. The *Simple Order* Scenario with a grey agent and three landmarks. The agent must go to the left green landmark, then the bottom black landmark, and finally the right red landmark to achieve the greatest amount of return.

5.2. Experiment Setup

Three primary experiments were run over the *Simple* and *Simple Order* environments to evaluate the generalization, sample efficiency, and exploration of methods. First, in an experiment referred to as the *Simple-General-Experiment*, we evaluated the generalization and sample efficiency of all four algorithms in the *Simple* scenario. The agent was set to respawn randomly within a (-5, 5) range of x- y-positions while the landmark is always at the (0, 0) position. An algorithm capable of generalizing with high sample efficiency will quickly learn to act optimally from various random initial locations.

Second, the *Simple-Fixed-Experiment* aimed to evaluate the exploration procedure of GASIL, HAC, and GASIL+HAC with in the *Simple* scenario with fixed agent and landmark spawn locations.

Finally, the *Simple-Order-Fixed-Experiment* performed a similar evaluation to the *Simple-Fixed-Experiment* setup, but in the more complex *Simple Order* scenario. Here, the agent and all three landmarks spawned at fixed locations.

5.3. Training

All hyperparameters used to train each algorithm in each experiment are outlined in Appendix B.

5.4. Simple-General-Experiment

DDPG, GASIL, HAC, and GASIL+HAC were trained on four environment seeds (random number generators), and the averaged returns over the number of episodes are shown in Fig. 7. Using only relative position data as its observation input, we can see that HAC and GASIL+HAC outperform DDPG and GASIL in sample efficiency and they achieve higher returns. Figure 8 showcases two low-level and goal-setting trajectories taken by GASIL+HAC in the *Simple* scenario, in which we can see goals are set to pull the agent closer to the landmark where the sparse reward exists.

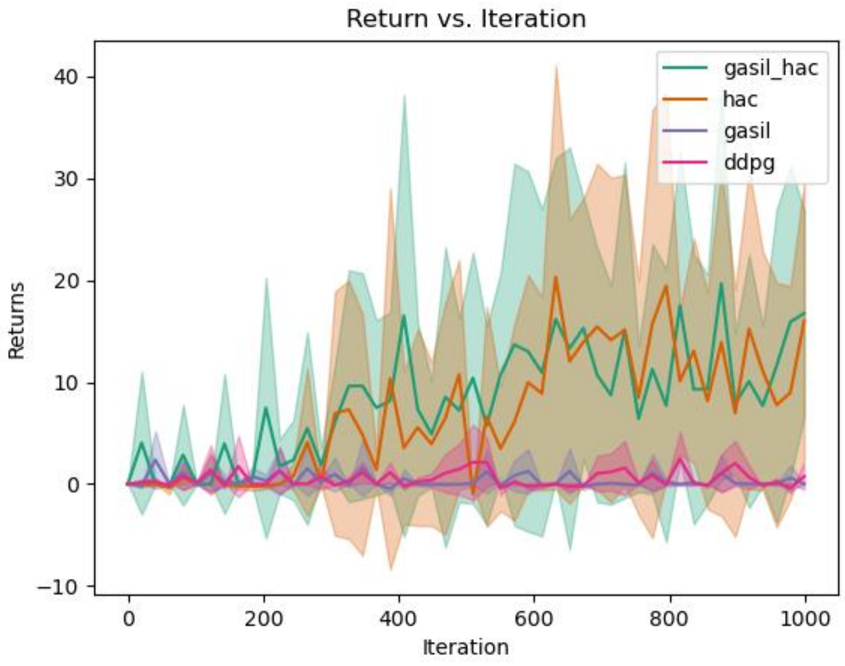


Figure 7. Returns over episodes for all four algorithms across four averaged seeds. Observation only includes relative position to landmark.

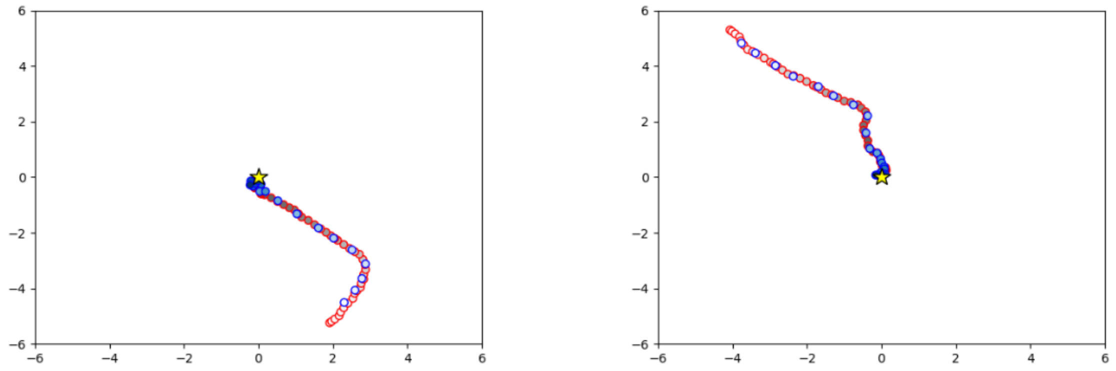


Figure 8. Sample trajectories of well-performing GASIL+HAC policy. Blue points are goals set by top-level and red points make up actual trajectory taken.

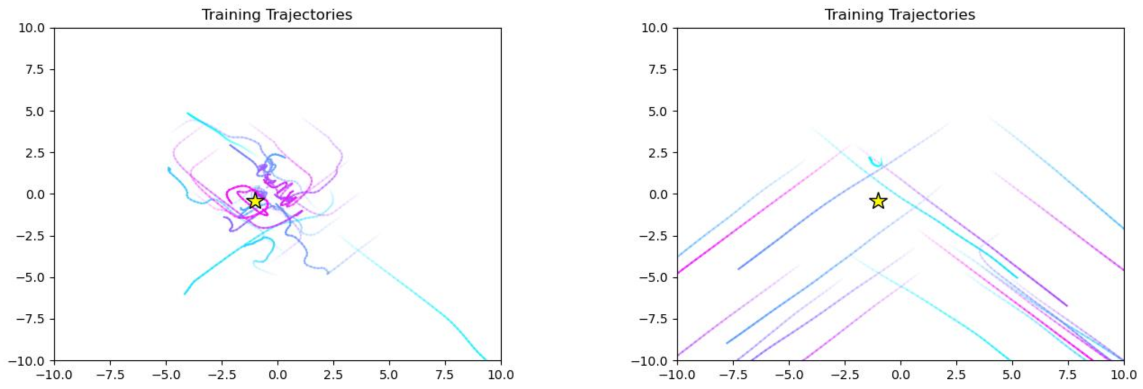


Figure 9. Trajectories taken by one seed of GASIL+HAC (left) and GASIL (right) agent during training period. Turquoise trajectories are early in training while pink trajectories are most recent.

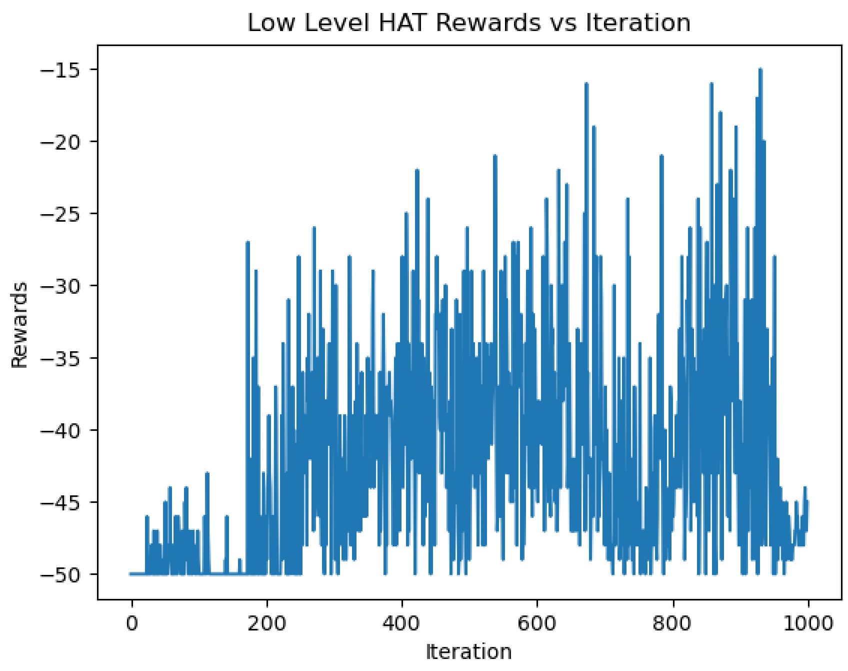


Figure 10. Artificial HAT returns of low-level policy per episode.

Figure 9 shows trajectories taken by the GASIL+HAC agent during training where turquoise trajectories were early in the training process and pink trajectories are near the end. We can see from this trajectory graph that the agent learns to generalize the process of reaching the (0, 0) position from nearly any initial location on that seed.

While the environment returns graph can be used to visualize the performance of an overall algorithm, the learning of the lower level(s) in a hierarchy can be monitored by keeping track of the number of penalized HATs given to a lower level per episode. As shown in Fig. 10, as the low-level policy learns to take actions to meet the goals given to it by its higher level, fewer penalizing HATs are given.

Over the four seeds used in this setup of the *Simple* scenario, we can see GASIL+HAC performs very similarly to HAC alone. However, DDPG and GASIL do not learn anything comparable to the other algorithms. Specifically, Fig. 9 shows that GASIL has primarily learned to always move down. To address whether the performance issue of GASIL alone was an error in implementation, an ablation study was performed using the original implementation of the *Simple* scenario from OpenAI. In addition to the relative position of the agent to the landmark, OpenAI's implementation of the *Simple* scenario also appended the current velocity state of the agent which may provide more useful information to the agent policy. The retrained policies are once again compared in Fig. 11. There, we can see that GASIL+HAC and HAC are still more sample efficient than DDPG and GASIL alone, but GASIL is able to reach about the same performance as the hierarchical policies by the end of training.

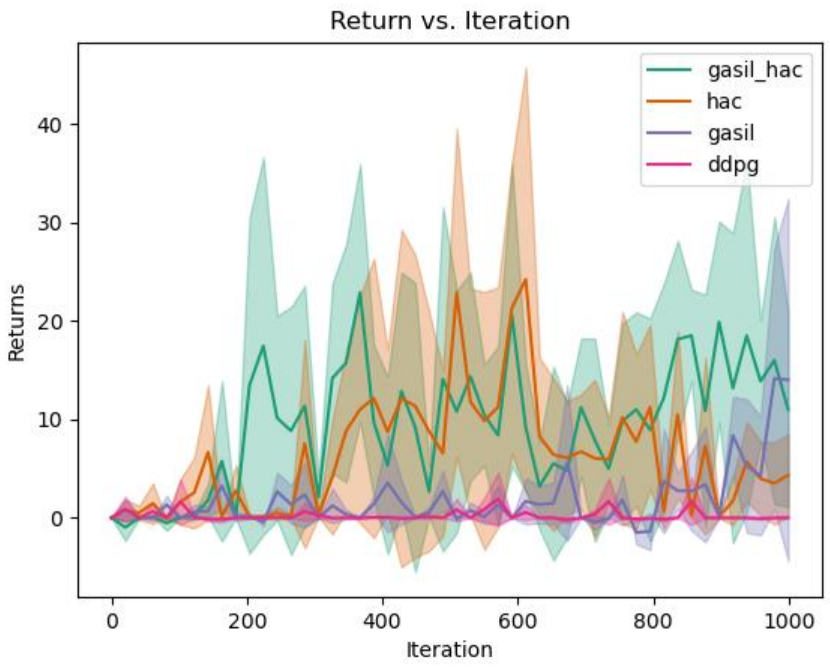


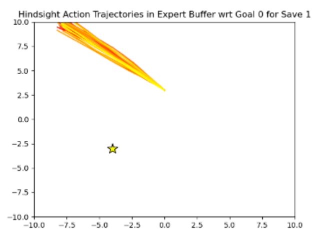
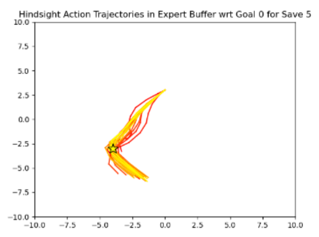
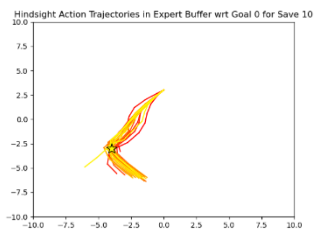
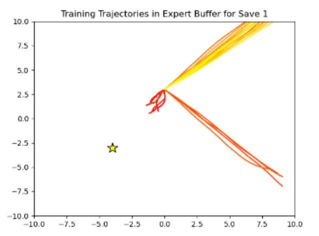
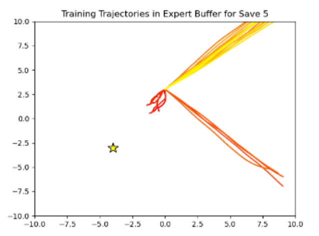
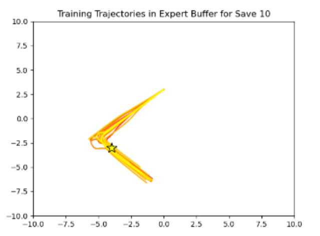
Figure 11. Returns over episodes for all four algorithms across four averaged seeds. Observation includes relative position to landmark and agent's velocity.

5.5. Simple-Fixed-Experiment

The *Simple-Fixed-Experiment* was developed to evaluate the exploration techniques of GASIL, HAC, and GASIL+HAC in the *Simple* scenario. Recall that GASIL and GASIL+HAC learn to imitate past well-performing trajectories that are stored in an expert trajectory replay buffer. A policy that explored its environment well during training will store trajectories with higher returns over time. We can therefore plot the trajectories stored in the expert replay buffer of GASIL and GASIL+HAC for a seed of training over several iterations and compare the exploration performed.

Table 1 plots the trajectories in the expert replay buffers of GASIL and GASIL+HAC at three iterations during training on one seed in the *Simple* fixed scenario. It shows that both algorithms are able to discover high returning trajectories that pass through the landmark for the seeds graphed. Figure 12 shows the same information in a quantitative view, where we can see that GASIL+HAC finds high return trajectories around iteration 400 while GASIL does the same at around iteration 600 for that seed.

Table 1. Trajectories stored in the expert replay buffers of GASIL and GASIL+HAC for one seed during the training of the *Simple-Fixed-Experiment*.

Algorithm	Iteration 100	Iteration 500	Iteration 1,000
GASIL + HAC			
GASIL			

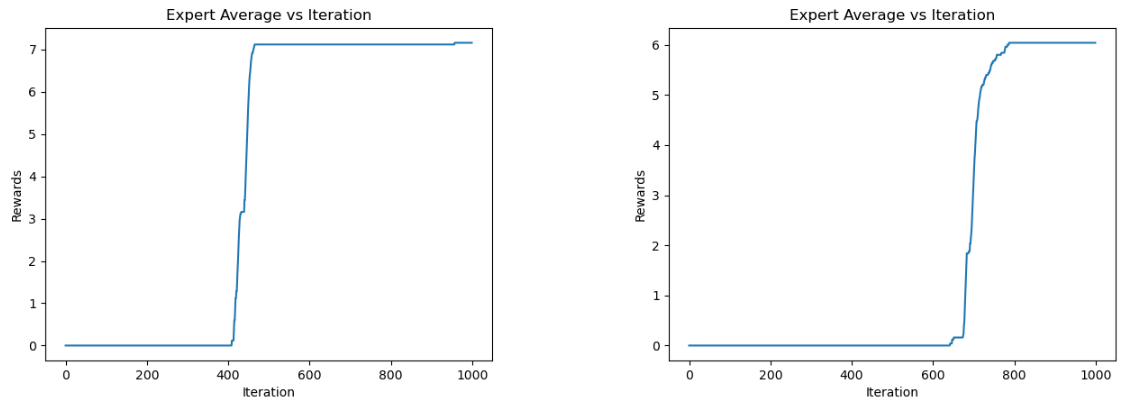


Figure 12. Average return of trajectories in expert replay buffer of GASIL+HAC (left) and GASIL (right) for one seed.

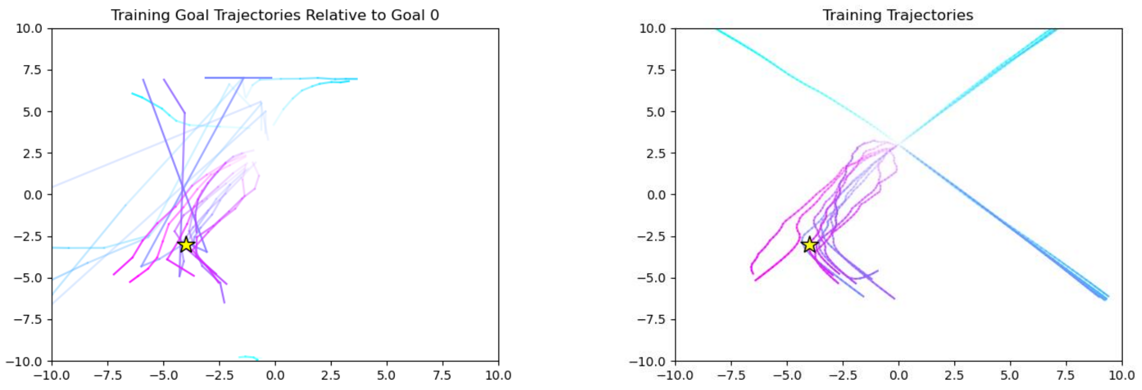


Figure 13. GASIL+HAC goal setting trajectories (left) and low-level trajectories (right) in *Simple* fixed scenario for one seed. Turquoise trajectories happened earlier in training while pink trajectories occurred later.

Another method to evaluate the exploration of algorithms is to plot a trajectory taken by the agent every few episodes during the entire training period. An agent that filled its replay buffers with a diverse set of experiences would have a set of training trajectories that span a wide range of its state space. Figure 13 shows the trajectories taken by a GASIL+HAC agent over one seed of training in the *Simple* fixed scenario. The top-level goal setting agent slowly learns to output goals trajectories that lead to the landmark while the low-level used its initial poor performance to explore its environment. Similarly, Fig. 14 shows that, once a GASIL agent finds well-performing trajectories, it can converge to a similar policy to GASIL+HAC.

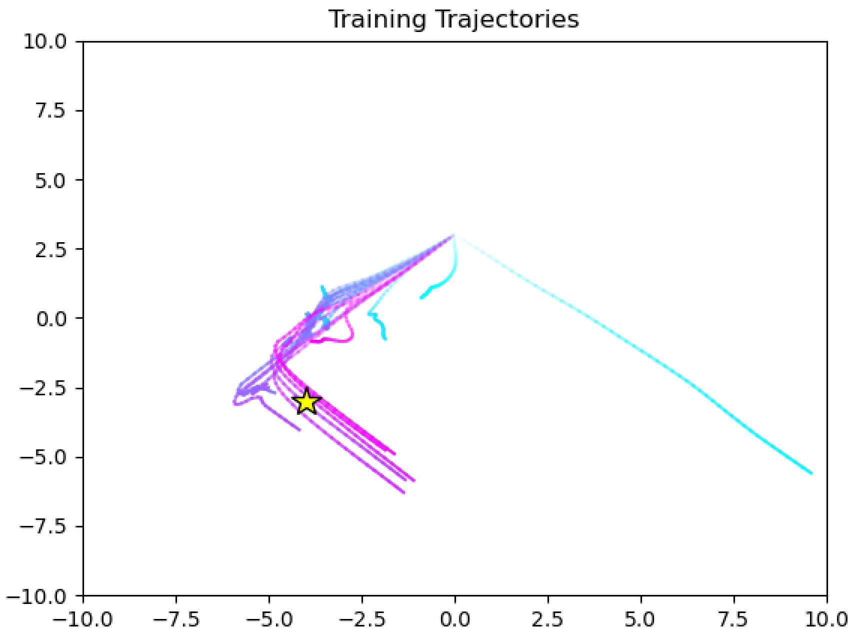


Figure 14. GASIL trajectories during training in one seed of *Simple* fixed scenario.

In contrast, we can see an example of poor exploration occurring in HAC within Fig. 15 where the agent never passes over the landmark, so it never receives an environment reward. In this case, the agent converges to a poor policy that decides to move diagonally to the bottom right until the episode ends.

For completeness, we provide the full comparison of algorithms' performance averaged over four seeds in Fig. 16. We can see that all three algorithms reach some reward by end of training, but this experiment may be run over more seeds to be smoothed further and reach more conclusive results.

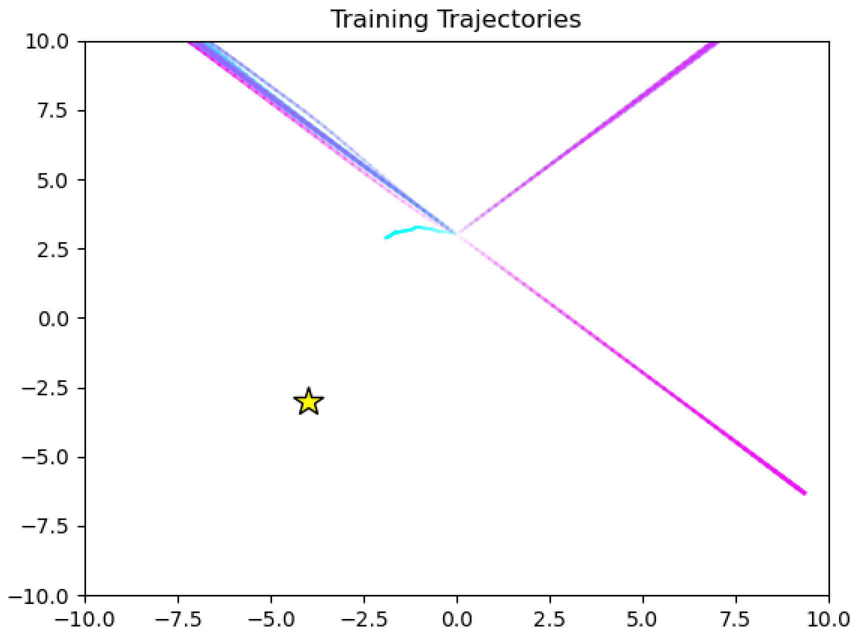


Figure 15. Training trajectories for one seed for HAC in *Simple* fixed scenario. Showcases a poor exploration outcome.

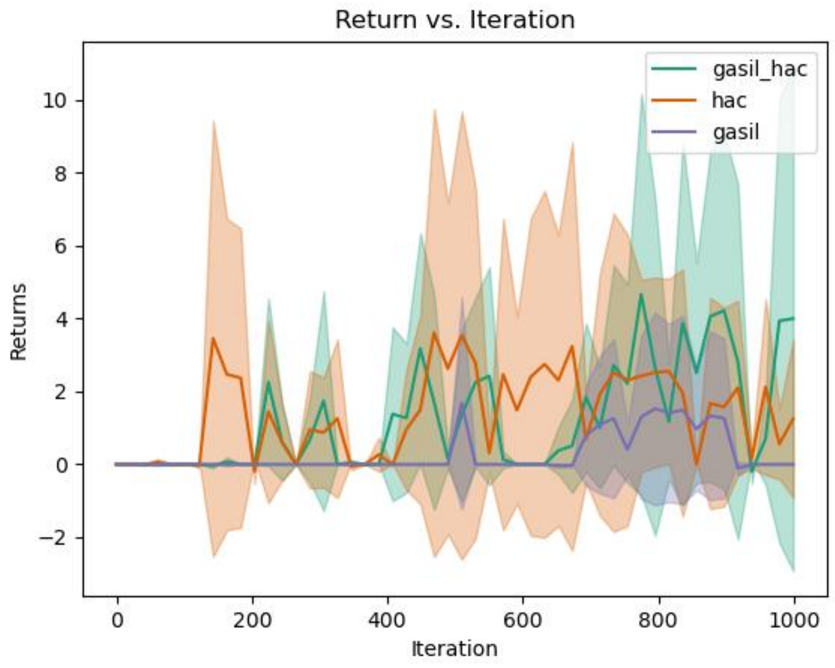


Figure 16. Returns over all training iterations averaged over four seeds for GASIL+HAC, HAC, and GASIL.

5.6. Simple-Order-Fixed-Experiment

Like the *Simple-Fixed-Experiment*, the *Simple-Order-Fixed-Experiment* attempts to visualize the exploration methods of policies, but in a longer-duration sparse reward task. Recall from Fig. 6 that an agent must pass through each landmark in its observation in a very specific order to receive the greatest amount of return, which inherently makes the problem sparser than the *Sparse* scenario. In turn, the observation space of the agent has also become more complex. For an experiment with three landmarks, we encode the relative position to each landmark into a tuple of length six. For example, an observation tuple of $< 1, 2, 0, 0, -3, 3 >$ encodes that the agent is 1 unit to the right and 2 units above the first landmark; at the same position as the second landmark; and 3 units to the left and 3 units above the final landmark.

Before delving into results of this experiment, it is important to discuss a direct consequence of the hierarchical goal setting approach with regards to setting reachable goals. Recall that all non-base level policies in the hierarchical methods discussed output a tuple in the same space as the observation space. In the case of the *Simple-Order* scenario, the goal setters will output a similar tuple of length six encoding a desired relative to position to all three landmarks. During training of these hierarchical policies, it is possible for the goal setters to output a goal that is physically impossible to be reached by lower levels. For example, if all three landmarks are at three different absolute positions in the environment, but the goal given is $< 0, 0, 0, 0, 0, 0 >$, then the agent cannot physically reach that state because it cannot be at all three landmarks at the same time. This problem of reachability needs to be addressed in the future to ensure lower levels do not receive misleading artificial rewards during training.

We show the results of one interesting seed for GASIL+HAC in Figs. 17 – 20. Specifically, we can see from Fig. 17 that the policy was able to find at least one expert trajectory during exploration that passes through two landmarks in the correct order to receive a return of 2.0. However, Figs. 17 and 18 also show that the hierarchy did not learn to imitate those well-performing trajectories, and instead converged on a policy that moves to the top left of the start point, which receives a return of 0.0 normally. On the front of reachability, in Fig. 19, we plot the final trajectories of the goal output by the top level with respect to each landmark that the subgoal is encoded for on one seed. In this case, we can see that GASIL+HAC has learned to set reachable goals because the goal trajectories with respect to each landmark are very similar. Furthermore, Figs. 19 and 20 show that the low-level was able to learn to follow the goals set by the top-level goal setter as the low-level returns increased over time.

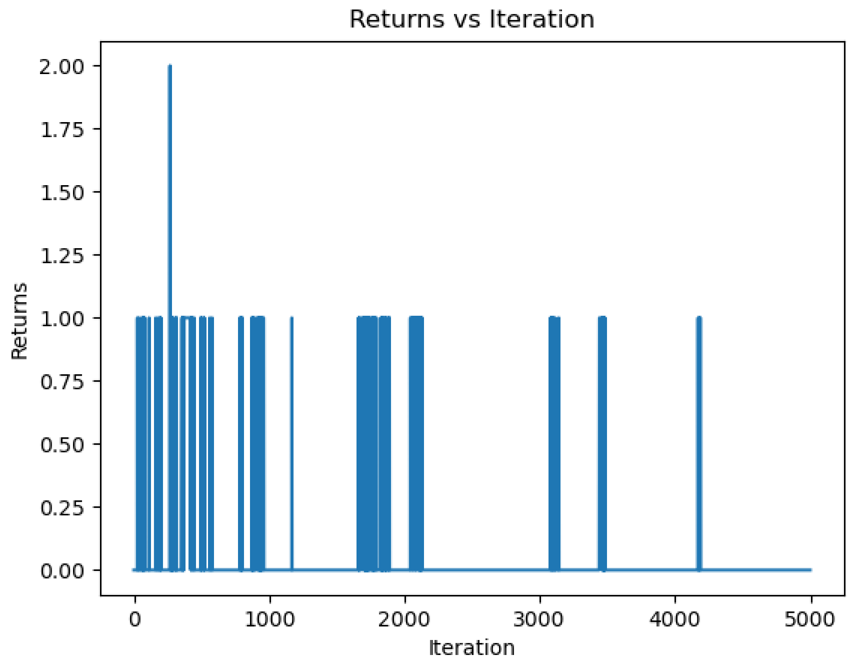


Figure 17. Returns over iterations for one seed of GASIL+HAC in *Simple-Order-Fixed-Experiment*

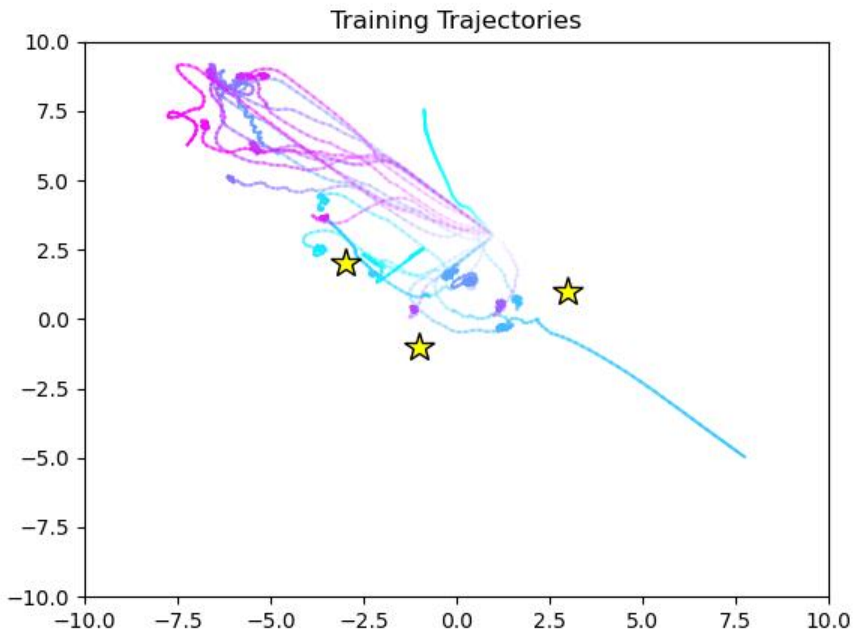


Figure 18. All trajectories taken by GASIL+HAC for one seed during training in *Simple-Order-Fixed-Experiment*

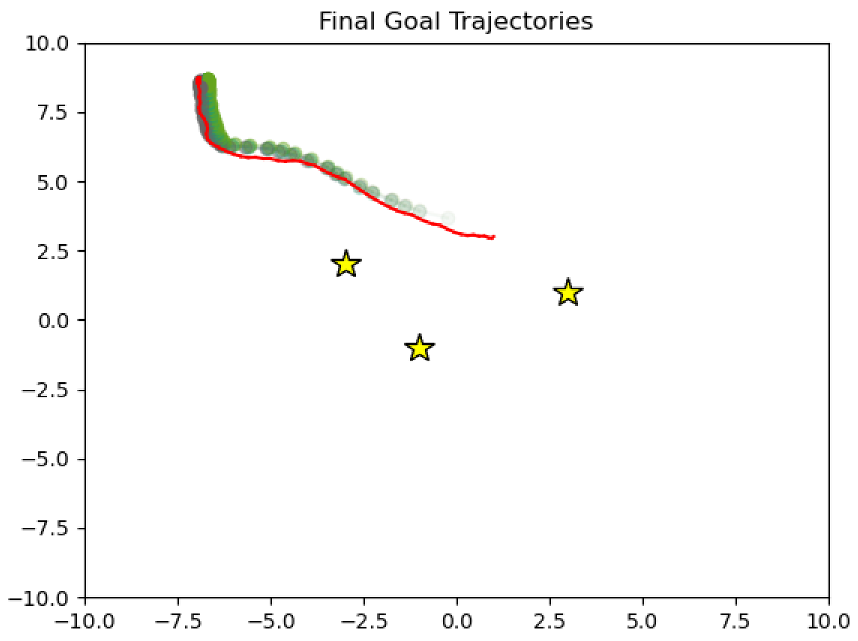


Figure 19. Goal-output and low-level trajectories for final episode on one seed of training. High level goal outputs relative to each goal are plotted with circle markers in grey, green, and blue. The high-level goal setter has learned to produce reachable goals because the goal trajectories with respect to each goal overlap. The low-level policy in red has learned to reach those goals.

On the other hand, Figs. 21 – 23 are the resulting data from a seed where GASIL+HAC performed poorly in exploration and the top-level did not learn to give reachable goals. Figure 21 shows that, in this seed, the agent never passed over the first landmark, so it never received a return of greater than 0.0 for any trajectory. Furthermore, Fig. 22 hints to the idea that the low-level never learned to follow the goals of the top-level as its learning curve never consistently increases. Fig. 23 supports this belief as, when we plot the relative goals given by the top-level with respect to each landmark, there is no significant overlap in goal trajectories – the goals the top-level learned to give were unreachable. Similarly, the low-level just learned to move diagonally to the bottom left without reaching goals consistently.

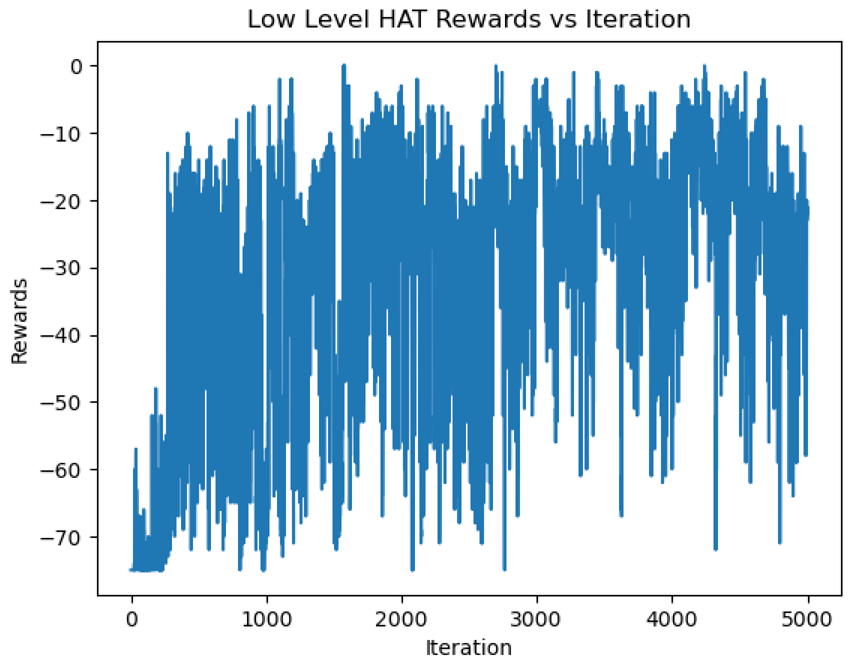


Figure 20. Low-level returns over iterations for one seed in *Simple-Order-Fixed-Experiment.*

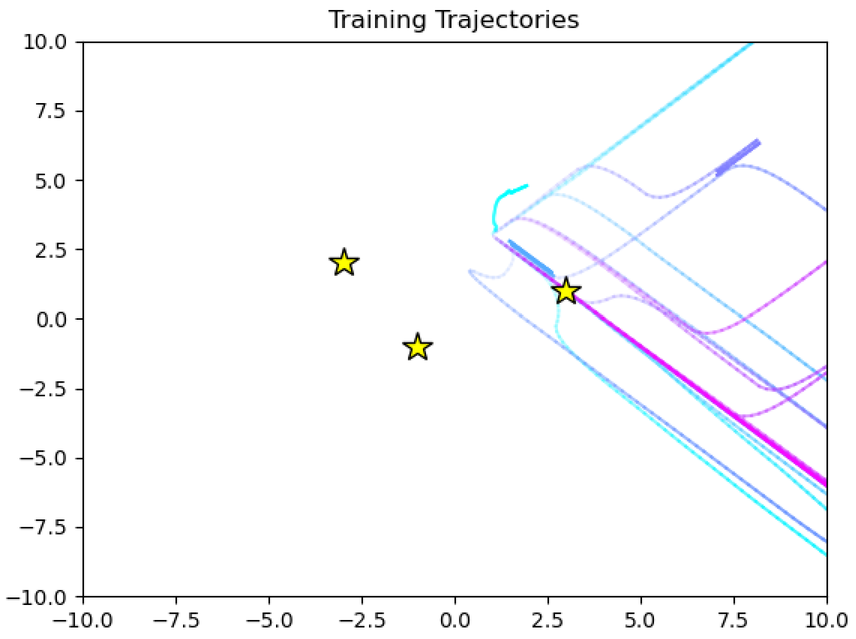


Figure 21. All trajectories taken by GASIL+HAC for one seed during training in this experiment with poor exploration.

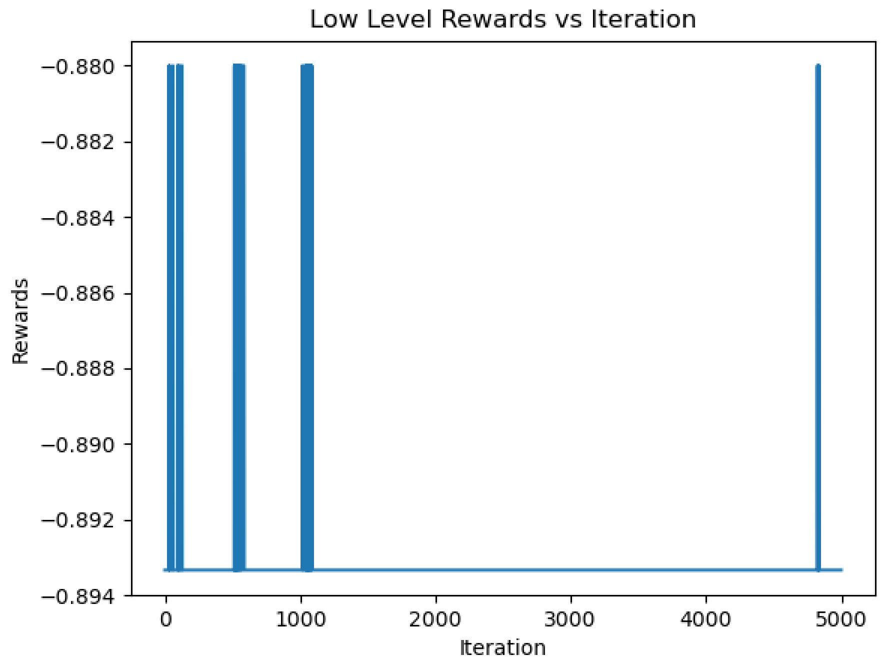


Figure 22. Low-level returns over iterations for one seed in this experiment where non-reachable goals are given.

Overall, the results of the two seeds presented for GASIL+HAC showcase two extreme ends of the spectrum of effectiveness of the algorithm in more complex environment. Specifically, we can see GASIL+HAC is still reliant on a good exploration policy to find expert trajectories with high return. Even when good expert trajectories are found, there is a chance that multiple levels of the hierarchy collapse during training and stop imitating the good expert trajectories. Finally, in complex environments with multiple relative landmark points, it is important to deal with the reachable goal problem as the lower levels may not learn if they cannot ever physically reach the goals set by higher levels.

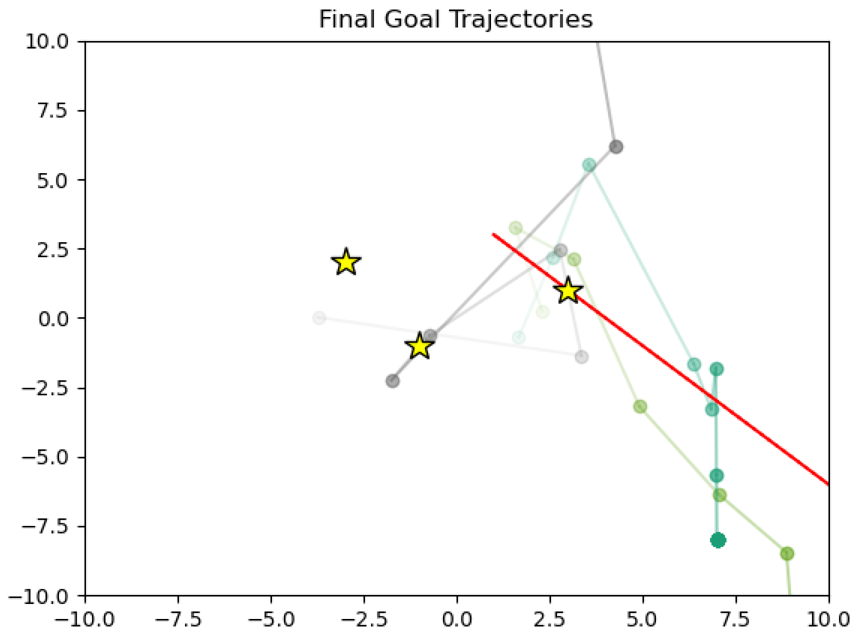


Figure 23. Goal-output and low-level trajectories for final episode on one seed of training with non-reachable goals given. High level goal outputs relative to each goal are plotted with circle markers in grey, green, and blue. The high-level goal setter has learned to produce non-reachable goals because the goal trajectories with respect to each goal do not overlap. The low-level policy in red has converged to move diagonally to the right without receiving any rewards.

6. Conclusion

This thesis reports on experiments run primarily to evaluate the sample efficiency, generalizability, exploration effectiveness, and goal reachability of the proposed GASIL+HAC algorithm in comparison to baselines of DDPG, GASIL, and HAC. We find that HAC and GASIL+HAC have a similar sample efficiency that is greater than that of DDPG and GASIL alone in the *Simple-General-Experiment*. By the end of training, GASIL+HAC and HAC were able to generalize across four seeds in the stochastic environment whereas DDPG and GASIL faced difficulty in finding an optimal policy. An ablation was run in the same scenario but with velocity information incorporated into the observation space of the agent, through which we found that GASIL was still able to match the same training performance of HAC and GASIL+HAC by the end of training (with lower sample efficiency.)

Both the *Simple-Fixed-Experiment* and the *Simple-Order-Fixed-Experiment* showcase the importance of a good exploration method for GASIL and GASIL+HAC to learn well. Seeds where self-imitating policies found well-performing trajectories assisted the policies to learn to act more optimally in the environment as they learned to mimic those trajectories over time. However, there were instances where, even though the expert buffer became filled with high-return trajectories, because a level in the hierarchy of GASIL+HAC or HAC collapsed, the policy did not learn to match those trajectories.

Finally, we investigated the ability of hierarchical methods to output reachable goals in environments with more complex observation spaces. In certain seeds, GASIL+HAC was able to learn to generate reachable goals to lower levels but did not end up imitating the trajectories in its expert buffer with high return. On the other hand, some seeds were found to converge to a very poor low-level policy early on in training. As a result, the policy did not explore enough to find any landmarks and the high-level goal setter did not learn to give physically reachable goals in the environment.

Overall, we can see that GASIL+HAC has the potential to increase sample efficiency of solving sparse reward scenarios in comparison to non-hierarchical methods. Furthermore, hierarchical methods are seen to increase the number of explored states in an environment on certain seeds due to early training process of the low-level. However, HAC and GASIL+HAC inherently have greater instability in training due to the nonstationary problem. In addition, if the method converges to a non-optimal policy early on, then it will be unable to explore and find trajectories with high return.

This collection of benefits and drawbacks leads us to several potential areas of future study. Firstly, several more experiments must be run with a greater number of seeds and hyperparameter tuning to come to a complete conclusion on the relative performance of all algorithms on the presented scenarios. Some methods may be explored to deal with the issue of setting reachable goals. This includes incorporating a pretraining process where the higher-level is trained to learn to give reachable goals via a different discriminator or just modifying the environment observation itself to always be reachable. Additionally, we would like to explore methods to improve the exploration of the policies themselves. Finally, these hierarchical approaches may be extended to collaborative multi-agent reinforcement learning where a goal-setter may learn to designate goals to several agents to receive the greatest cumulative return over time.

References

[1] J. Oh, Y. Guo, S. Singh and H. Lee, "Self-Imitation Learning," *PMLR,* 2018.

[2] O. Nachum, H. Tang, X. Lu, S. Gu, H. Lee and S. Levine, "Why Does Hierarchy (Sometimes) Work So Well in Reinforcement Learning?," *NeurIPS DeepRL Workshop,* 2019.

[3] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel and I. Mordatch, "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments," *NIPS,* 2017.

[4] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell and S. Legg, "Noisy Networks for Exploration," *ICLR,* 2018.

[5] M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel and M. Andrychowicz, "Parameter Space Noise for Exploration," *ICLR,* 2018.

[6] D. Pathak, P. Agrawal, A. A. Efros and T. Darrell, "Curiosity-driven Exploration by Self-supervised Prediction," *ICML,* 2017.

[7] Y. Burda, H. Edwards, A. Storkey and O. Klimov, "Exploration by Random Network Distillation," *ICLR,* 2019.

[8] P.-A. Andersen, M. Goodwin and O.-C. Granmo, "The Dreaming Variational Autoencoder for Reinforcement Learning Environments," *AI,* 2018.

[9] B. Eysenbach, A. Gupta, J. Ibarz and S. Levine, "Diversity is All You Need: Learning Skills without a Reward Function," *arXiv:1802.06070,* 2018.

[10] A. Gupta, V. Kumar, C. Lynch, S. Levine and K. Hausman, "Relay Policy Learning: Solving Long-Horizon Tasks via Imitation and Reinforcement Learning," *CoRL,* 2019.

[11] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley and J. Clune, "Go-Explore: A New Approach for Hard-Exploration Problems," *arXiv:1901.10995,* 2021.

[12] Y. Guo, J. Choi, M. Moczulski, S. Feng, S. Bengio, M. Norouzi and H. Lee, "Memory Based Trajectory-conditioned Policies for Learning from Sparse Rewards," *arXiv:1907.10247,* 2021.

[13] S. Reddy, A. D. Dragan and S. Levine, "SQIL: Imitation Learning via Reinforcement Learning with Sparse Rewards," *ICLR,* 2020.

[14] Y. Guo, J. Oh, S. Singh and H. Lee, "Generative Adversarial Self-Imitation Learning," *arXiv:1812.00950,* 2018.

[15] A. Levy, G. Konidaris, R. Platt and K. Saenko, "Learning Multi-Level Hierarchies with Hindsight," *ICLR,* 2019.

[16] J. Achiam, "Spinning Up in Deep Reinforcement Learning," 2018.

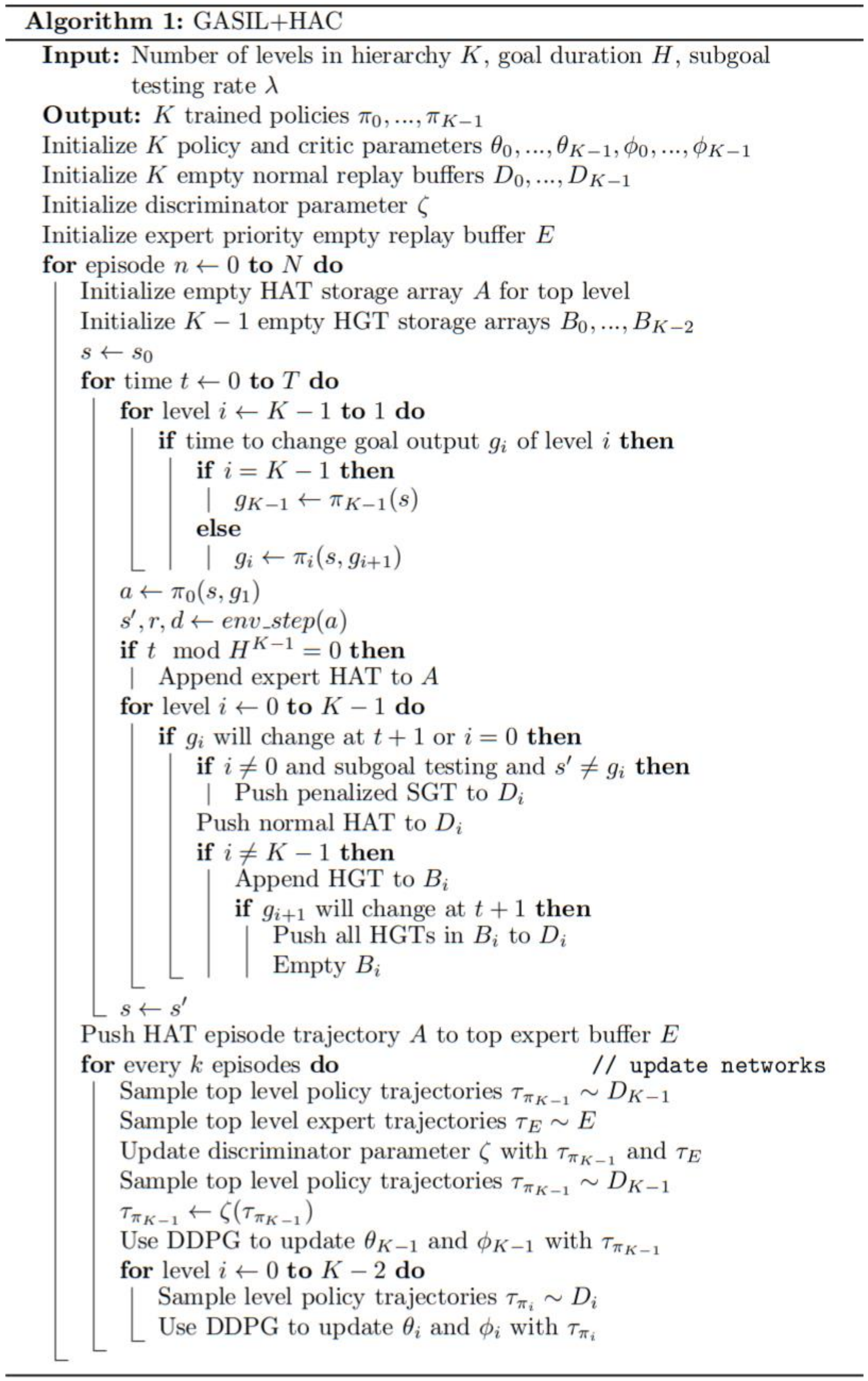
[17] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver and D. Wierstra, "Continuous Control with Deep Reinforcement Learning," *ICLR,* 2016.

[18] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *NIPS,* 2013.

[19] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel and W. Zaremba, "Hindsight Experience Replay," *NIPS,* 2017.

[20] L. Weng, "Exploration Strategies in Deep Reinforcement Learning," *lilianweng.github.io/lil-log,* 2020.

Appendix A GASIL+HAC Algorithm



```
Algorithm 1: GASIL+HAC
Input: Number of levels in hierarchy $K$, goal duration $H$, subgoal
       testing rate $\lambda$
Output: $K$ trained policies $\pi_0, ..., \pi_{K-1}$
Initialize $K$ policy and critic parameters $\theta_0, ..., \theta_{K-1}, \phi_0, ..., \phi_{K-1}$
Initialize $K$ empty normal replay buffers $D_0, ..., D_{K-1}$
Initialize discriminator parameter $\zeta$
Initialize expert priority empty replay buffer $E$
for episode $n \leftarrow 0$ to $N$ do
    Initialize empty HAT storage array $A$ for top level
    Initialize $K-1$ empty HGT storage arrays $B_0, ..., B_{K-2}$
    $s \leftarrow s_0$
    for time $t \leftarrow 0$ to $T$ do
        for level $i \leftarrow K-1$ to 1 do
            if time to change goal output $g_i$ of level $i$ then
                if $i = K-1$ then
                    $g_{K-1} \leftarrow \pi_{K-1}(s)$
                else
                    $g_i \leftarrow \pi_i(s, g_{i+1})$
        $a \leftarrow \pi_0(s, g_1)$
        $s', r, d \leftarrow env\_step(a)$
        if $t \mod H^{K-1} = 0$ then
            Append expert HAT to $A$
        for level $i \leftarrow 0$ to $K-1$ do
            if $g_i$ will change at $t+1$ or $i = 0$ then
                if $i \neq 0$ and subgoal testing and $s' \neq g_i$ then
                    Push penalized SGT to $D_i$
                Push normal HAT to $D_i$
                if $i \neq K-1$ then
                    Append HGT to $B_i$
                    if $g_{i+1}$ will change at $t+1$ then
                        Push all HGTs in $B_i$ to $D_i$
                        Empty $B_i$
        $s \leftarrow s'$
    Push HAT episode trajectory $A$ to top expert buffer $E$
    for every $k$ episodes do                          // update networks
        Sample top level policy trajectories $\tau_{\pi_{K-1}} \sim D_{K-1}$
        Sample top level expert trajectories $\tau_E \sim E$
        Update discriminator parameter $\zeta$ with $\tau_{\pi_{K-1}}$ and $\tau_E$
        Sample top level policy trajectories $\tau_{\pi_{K-1}} \sim D_{K-1}$
        $\tau_{\pi_{K-1}} \leftarrow \zeta(\tau_{\pi_{K-1}})$
        Use DDPG to update $\theta_{K-1}$ and $\phi_{K-1}$ with $\tau_{\pi_{K-1}}$
        for level $i \leftarrow 0$ to $K-2$ do
            Sample level policy trajectories $\tau_{\pi_i} \sim D_i$
            Use DDPG to update $\theta_i$ and $\phi_i$ with $\tau_{\pi_i}$
```

Figure 24. GASIL+HAC Algorithm

Appendix B Networks and Hyperparameters Used

Table 2. Parameters tuned in experiments run in this paper and their definitions.

Parameter	Definition
# Seeds	# of different runs for experiment
Buffer Length	Max # of transitions in normal buffers
Expert Buffer Length	Max # of trajectories in expert buffer
# Episodes	# of episodes trained on
Episode Length	Time at which each episode times out and resets
Episodes per Update	Networks updated every this many episodes
Batch Size	Sample size for policy/critic updates
Discriminator Batch Size	Sample size for discriminator updates
# Discriminator Updates	# minibatches every discriminator update
# Actor/Critic Updates	# minibatches every policy/critic update
Learning Rate	Learning rate of policy/critic networks
Discriminator Learning Rate	Learning rate of discriminator network
τ	Target network update rate
K	# levels in hierarchy
H	Max goal duration of goal-setters
Goal Threshold	Threshold between state and goal that is rewarded
γ	Discount Factor
λ	Subgoal testing rate
State Bounds Tuple	Goal-setting levels' outputs bounded by max/min of tuple

All networks used to parameterize functions were built as three-layer Multi-Layer Perceptron Networks with a hidden layer size of 64 and intermediate ReLU nonlinearities. Vanilla DDPG and GASIL policy nets had a final activation of tanh while critic nets had no activation at the output. The low-level actor of both hierarchical methods had a similar structure, but we also followed the methodology of the original HAC paper and bounded low critic nets to output between $(-H, 0)$. High-level actor outputs were scaled to output between state bounds of the scenario observation space. Discriminator nets had a similar architecture but used intermediate tanh nonlinearities and was unconstrained at its output. The definitions of hyperparameters used during training are described in Table 2, and the specific values of those hyperparameters used for each algorithm are listed in Table 3 through Table 6.

Table 3. Parameters that GASIL+HAC were run with in the three experiments presented in this paper.

Parameter	**Simple General Env**	**Simple Fixed Env**	**Simple Order Fixed Env**
# Seeds	4	4	4
Buffer Length	5e5	5e5	5e5
Expert Buffer Length	25	25	20
# Episodes	1,000	1,000	5,000
Episode Length	50	50	75
Episodes per Update	1	1	1
Batch Size	100	100	100
Discriminator Batch Size	256	256	256
# Discriminator Updates	10	10	10
# Actor/Critic Updates	100	100	100
Learning Rate	1e-3	1e-3	1e-3
Discriminator Learning Rate	3e-4	3e-4	3e-4
τ	1e-3	1e-3	1e-3
K	2	2	2
H	5	5	5
Goal Threshold	0.1	0.1	0.5
γ	0.95	0.95	0.95
λ	0.3	0.3	0.3
State Bounds Tuple	$< -5, 5 >$	$< -10, 10 >$	$< -10, 10 >$

Table 4. Parameters that HAC were run with in *Simple* scenario experiments presented in this paper.

Parameter	**Simple General Env**	**Simple Fixed Env**
# Seeds	4	4
Buffer Length	5e5	5e5
# Episodes	1,000	1,000
Episode Length	50	50
Episodes per Update	1	1
Batch Size	100	100
# Actor/Critic Updates	100	100
Learning Rate	1e-3	1e-3
τ	1e-3	1e-3
K	2	2
H	5	5
Goal Threshold	0.1	0.1
γ	0.95	0.95
λ	0.3	0.3
State Bounds Tuple	$< -5, 5 >$	$< -10, 10 >$

Table 5. Parameters that GASIL were run with in *Simple* scenario experiments presented in this paper.

Parameter	Simple General Env	Simple Fixed Env
# Seeds	4	4
Buffer Length	5e5	5e5
Expert Buffer Length	25	25
# Episodes	1,000	1,000
Episode Length	50	50
Episodes per Update	1	1
Batch Size	100	100
Discriminator Batch Size	256	256
# Discriminator Updates	10	10
# Actor/Critic Updates	100	100
Learning Rate	1e-3	1e-3
Discriminator Learning Rate	3e-4	3e-4
τ	1e-3	1e-3
γ	0.95	0.95

Table 6. Parameters that DDPG were run with in *Simple-General-Experiment*.

Parameter	Simple General Env
# Seeds	4
Buffer Length	5e5
# Episodes	1,000
Episode Length	50
Episodes per Update	1
Batch Size	100
# Actor/Critic Updates	100
Learning Rate	1e-3
τ	1e-3
γ	0.95