Hierarchical Actor-Critic Exploration with Synchronized, Adversarial, & Knowledge-Based Actions

Ayush Jain

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2022-6
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2022/EECS-2022-6.html

April 17, 2022

Copyright © 2022, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Acknowledgement

To Theophile Cabannes, Professor Alex Bayen, and Professor Pieter Abbeel, for years of guidance and mentorship without which I would have never made it this far. Their depth of experience and willingness to share it has made every daunting challenge more approachable.

To my mother, father, and sister, for showing me the kind of unequivocal love and support that only family can give.

To many friends, from home and at Berkeley, who never flinched whether I needed help or just a laugh and for whom I can only ever hope to be as good a friend.

I am unbelievably fortunate. Thank you all.

HAC-E-SAK: Hierarchical Actor-Critic Exploration with Synchronized, Adversarial, & Knowledge-Based Actions

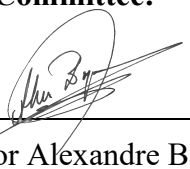
by Ayush Jain

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

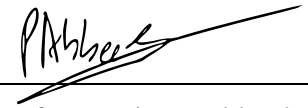


Professor Alexandre Bayen
Research Advisor

4/16/22

(Date)

* * * * * * *



Professor Pieter Abbeel
Second Reader

4/16/22

(Date)

HAC-E-SAK: Hierarchical Actor-Critic Exploration with Synchronized, Adversarial, & Knowledge-Based Actions

Ayush Jain
UC Berkeley
ayush.jain@berkeley.edu

Abstract

This work leans into the efficiency and robustness advantages of a hierarchical learning structure by introducing HAC-E-SAK, which expands beyond the Hierarchical Actor-Critic framework with an Exploration paradigm driven by Synchronized, Adversarial, and Knowledge-based actions. While Hierarchical Actor-Critic (HAC) emphasizes a strictly-defined hierarchical organization for rapid learning through parallelized training of multilevel subtask transition functions, it does not extend this principle to the exploration phase of training, an oversight addressed by this work's approach. Further, HAC's exploration strategy consists of simple ϵ-greedy based perturbations to deterministic actions generated from the DDPG algorithm. The novel approach presented in this work substitutes this with an alternate adversarial strategy relying on knowledge of prior agent experiences, motivating guided environment discovery for tasks with continuous state and action spaces. HAC-E-SAK extends the aforementioned hierarchical organization used by leading methods in subtask learning for the parallel purpose of structured exploration, allowing for explicit synchronization between levels. Experiments across a number of sparse-reward scenarios in Flow and OpenAI Gym demonstrate HAC-E-SAK's consistent outperformance over other tested procedures in terms of both sample efficiency and task success rates.

1 Introduction

Hierarchical Reinforcement Learning (HRL) methods provide a methodology by which to train agents to solve multi-dimensional decision tasks in a sample efficient manner, particularly as compared to agents trained through non-HRL procedures (1). By construction (2), HRL agents decompose goal tasks into sequential subtasks that are considerably more approachable and require shorter decision sequences to solve, enabling faster learning times in scenarios with sparse or delayed rewards. Additionally, this structure allows HRL agents to play a leading role in improving convergence for Federated Learning procedures (3), which is key to the development of privacy-preserving learning. This work identifies and addresses key weaknesses and unaddressed fallibility among even leading methods, as detailed in the following sections.

As is typical of reinforcement learning setups, this work aims to solve a Markov Decision Process (MDP). The standard MDP structure is augmented here to be goal-conditioned, following the convention noted in (4). Said convention introduces this notion of goal-conditioning by way of a Universal MDP (UMDP), or an MDP with transitions each including a goal among a set of goals $\mathcal{G}$ (each a state or set of states) that an agent will be told to learn.

A UMDP is then denoted as 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 $T\left(s, a, s'\right)$ is the

Research Project submitted in partial satisfaction of the requirements for a Master of Science, Plan II.

probability of transitioning to state s' when action a is taken in state s; R is the reward function; γ is a discount rate $\in [0, 1)$. Notations derived from (4). Each of $\mathcal{S}, \mathcal{G}, \mathcal{A}$ is a connected space.

At the start of every episode in a UMDP, a goal $g \in \mathcal{G}$ is selected for the entirety of the episode. Extending the solution formulation for a standard MDP to the goal-conditioned case, the solution to a UMDP is then a corresponding control policy $\pi : \mathcal{S}, \mathcal{G} \rightarrow \mathcal{A}$ that maximizes the value function $v_\pi(s, g) = \mathbb{E}_\pi \left[\sum_{n=0}^{\infty} \gamma^n R_{t+n+1} \mid s_t = s, g_t = g \right]$ for an initial state s and goal g.

The main contribution of this work is to considerably improve the convergence sample efficiency and overall learning speed beyond those of currently leading hierarchical subtask-decomposing learning methods with the introduction of HAC-E-SAK under the UMDP framework. This work does so through a novel hierarchical exploration strategy, and assesses the viability of this method through utilization in concert with a variety of noise and policy-based exploration techniques, including a new exploration procedure inspired by the principles of adversarial learning and self-play autocurricula generation. The results detailed in this work confirm the validity of this approach on a variety of environments ranging from mixed-autonomy traffic to physical control environments, and highlight particularly its benefits in scenarios with sparse reward and complex tasks.

2 Related Work

2.1 Hierarchical Actor-Critic

A number of HRL techniques have been devised for continuous control domains, and one such present leading proposal is the Hierarchical Actor-Critic (HAC) (5) framework. HAC is designed to substantially accelerate the training process by directing agents to learn an organized level hierarchy of policies in a joint manner. The HAC framework consists of two key aspects: a carefully-designed k-level hierarchical architecture and a strategy by which to parallelize the training of all policies in the hierarchical structure in an environment with relatively sparse rewards. In particular, HAC leverages a nested, goal-conditioned policy approach, in which all but the lowest-level policy treat the environment's state space as their action space and learn to generate subtasks.

More specifically, the highest level policy is provided the current and environment goal state as inputs, and generates a subgoal state to provide to the next level policy, which takes it and the current state to generate yet another subgoal state. This recursive structure continues down to the lowest level, at which point the current state and most basic subgoal are used to generate a primitive environment action. Each level's policy is given some number H of attempts to achieve the goal provided to it before being provided another from the policy one level above it.

Compared to other HRL methods (6; 7; 8; 9) with similar architecture that incrementally train an agent to go from start state to some environment goal state, HAC enables parallel learning by utilizing the notion of hindsight to decouple adjacent policy levels, thus allowing for simultaneous training. Hindsight is expressed in two ways. The first is *hindsight action*: when a higher-level policy poses a subtask state to a lower-level policy and the lower-level policy is unable to achieve it within the allocated steps, the transition received by the higher-level policy includes not the posed subtask but the eventual state achieved (as if it had been the action all along). The second is *hindsight goal*, a hierarchical extension of Hindsight Experience Replay (10) where the goal of an experience is replaced post-facto by the actual final state achieved by the experience. While HIRO (11), a prior HRL method, values subgoal actions with respect to a transition function derived from the current lower-level policy hierarchy, HAC values subgoals based on the optimal lower-level policy hierarchy, with optimality informed by hindsight as described. As a result, with HIRO, a higher-level policy can only learn meaningfully once the policy at the subsequent lower-level has converged, all the way down.

HAC does not extend its structured hierarchical principle to its exploration mechanism, and instead simply opts to randomly augment the action generated by the policy at any individual level with Gaussian white noise, independent of whether or not such an augmentation is chosen in parallel steps at the other levels. The proposed approach addresses this for higher-level policies, and also further leverages the presented experience replay-based hindsight framework to better inform agent exploration.

2.2 Asymmetric Self-Play

An alternate training methodology considered and studied at length in pursuit of a robust, generalizable, and efficient learning baseline was Asymmetric Self-Play (ASP) ([12]; [13]). ASP is designed to leverage self-play strategies for a single agent as a means to learn about a new and unknown environment. Specifically, the singular agent in question is compartmentalized into two "minds" or sub-agents, Alice and Bob, with a kind of adversarial relationship. Both Alice and Bob are designed such that they have the same architecture at the most basic level, but function based on distinct parameters and policy functions (input: state observations, output: distribution over actions).

More generally, Alice sets a goal, and Bob then attempts that goal. Alice takes her turn in the ASP procedure, attempting to trick or confuse Bob by traveling to a challenging state that she thinks Bob will struggle with. Bob subsequently is instructed to mimic Alice's behavior. If the environment is resettable, Bob is to reproduce Alice's behavior to the best of his ability. If the environment is instead reversible, Bob begins in Alice's end state and tries to work his way back to her initial state (reversibility OR resettability are thus prerequisites for an ASP-compatible environment).

The authors' definition of reward functions reflects Alice's aim to set a goal state that will be difficult for Bob to achieve. Expressed as the difference between her time to complete a task and Bob's time to mimic it in the ways specified above, Alice is rewarded based on how much harder a task is for Bob than it is for her. Thus, as Alice must be able to accomplish the goal for this to be a meaningful reward, the goal state is guaranteed to exist, even if Bob is not able to reach it. Similarly, as Bob tries to copy Alice, he aims to complete her specified task in as little time as possible.

While ASP as posed provides a concrete framework by which to automatically generate a curriculum to complete a task, its simple architecture quickly begins to struggle in the presence of just a few challenges, such as increased reward sparsity or task complexity. Experimentation revealed that ASP was unsuccessful in solving environments with few reward signals, even more so for complex decision tasks with the potential to be broken down into smaller tasks. However, it retained its promise and reliability in more simplistic scenarios, which this work extends to address the challenge of motivating exploration through an adversarial approach.

3 Approach

The proposed approach, explained in this section, devises a novel method by which to incorporate the salient underlying principles of the highlighted related works into a faster, more sample-efficient, and broadly generalizable learning procedure. Having motivated this objective in the previous sections, the following sections detail the key facets of HAC-E-SAK's implementation.

3.1 Training Speedup and Parallelization Methods

One of the major strengths of the HAC methodology is that it is very sample efficient ([5]). Learning via the saved replay buffer eliminates the need for constant sampling from the environment on-policy.

However, being able to simulate many samples in a computationally efficient manner is still extremely important for large-scale, complex tasks. Recently, NVIDIA's Isaac Gym has emerged as a contact physics-based RL simulation framework that leverages the parallel structure of a GPU to dramatically speed up common RL tasks ([14]). Specifically, Isaac Gym enables high-performance physics on the GPU instead of the CPU, relaxing the need for large computer clusters to achieve high quality results. Another key advantage of Isaac Gym is in keeping the entire state of the simulation in GPU memory, eliminating costly memory copies between GPU and CPU that are a typical performance bottleneck.

While the benefits of a framework like Isaac Gym are certainly impressive, they can only be realized through a concerted effort to adapt RL algorithms and frameworks to the highly-parallelized setting. One of the most obvious ways in which Isaac Gym uses the GPU to great effect is by creating several thousand adjacent environments in the simulator at the same time, enabling physics updates to occur in batched form. However, to achieve maximum throughput, the RL algorithm being deployed must also be able to produce batched action results given the current state vectors from the environments. This is commonly not a problem for conventional deep RL approaches, but the recursive nature of the HAC algorithm as presented in ([5]) makes this requirement challenging.

To support this need, this work reworks the recursive implementation of HAC into an iterative, parallelizable formulation (**Algorithms 1 and 2**) that makes the best use of Isaac Gym's capabilities. When considered independent of the algorithmic exploration contributions discussed in subsequent sections, this work's formulation of IterHAC is logically identical to the original work's. A distributed model of computation is used, in which a single HAC Coordinator maintains ownership of the k policies and replay buffers that are shared among the n HAC Mini Agents, where n is the number of simultaneous environments supported by the GPU using Isaac Gym. This sets the scene for the sections that follow, in which this work's novel exploration strategies are introduced.

Though measurements were limited to a single workstation and have not been rigorously validated, a multiple-magnitude speedup in wall clock time per episode was observed as a result of this parallelization. This section is included here to demonstrate the feasibility and correctness of an iterative HRL procedure, which allows for direct parallelization whereas recursive methods do not. Future work might thoroughly quantify the benefits of such a strategy.

Algorithm 1 Iterative, parallelized implementation of the HAC algorithm (IterHAC)

Require: A single coordinator, which maintains k policies, exploration policies, and replay buffers
Require: n mini-agents, each maintaining a length-k array of elapsed steps through hierarchies and an experience buffer, with access to the shared policies, exploration policies, and buffers

procedure AGENTRECORDTRANSITION($state, action, next\_state, done$)
  $i \leftarrow 0$
  while $i < k$ and ($i = 0$ or $steps[i-1] >= H$) **do**
    $reached\_goal \leftarrow$ whether or not $next\_state$ matches $goal_i$
    if $i > 0$ **then**
      Record HAC Hindsight Action into $buffer_i$
    end if
    if $done$ or $reached_goal$ or $steps[i] >= H$ **then**
      Load $experience_i$ and record as HAC Hindsight Goal into $buffer_i$
    end if
    if $i > 0$ and level $i-1$ in $SUBGOAL$ mode and not $next\_state$ matches $goal_{i-1}$ **then**
      Add HAC Subgoal Penalty to $buffer_i$
    end if
    $experience_i \leftarrow (transition)$
    $i \leftarrow i + 1$
  end while
end procedure
procedure HACCOORDINATOR(s_0, g_0, k, n, ep_len)
  $i \leftarrow 0$
  $states \leftarrow s_0$
  $goal \leftarrow g_0$
  while $i < ep\_len$ **do**
    $actions \leftarrow$ result of $AgentGetAction(states, goal)$ for all n agents
    $next\_states, external\_rewards, dones \leftarrow$ step environment with $actions$
    $AgentRecordTransition(states, actions, next\_states, dones)$ for all n agents
    Update all k exploration and exploitation policies
    $states \leftarrow next\_states$
    $i \leftarrow i + 1$
  end while
end procedure

3.2 Hierarchically Synchronized Exploration

While HRL methods emphasize a hierarchical structure for the purpose of decomposing tasks into more approachable subtasks, and leading methods such as HAC and HIRO provide for manager-worker relationships between policy layers for learning said subtasks, these relationships are not extended to exploration mechanisms at each level. More specifically, while HAC/HIRO have designed an architecture in which a higher-level manager policy controls the goal subtask for a lower-level worker and chooses when to evaluate the performance of all workers at levels below the manager, this

control and rigor is not afforded to exploration. Existing exploration is handled entirely independently at each level, with any given level perturbing the output action of the policy at that level on its own accord, using an ϵ-greedy strategy. This is amended with **Algorithm 2**, creating a hierarchical structure explicitly for the purpose of guided exploration that allows for synchronization between policy levels. This new structure provides substantial benefit, as highlighted in later sections.

Algorithm 2 Synchronized Exploration Procedure (Iterative)

procedure AGENTGETACTION($state$, $goal$)
  $i \leftarrow 0$
  while $steps[i] >= H$ **do** ▷ Find first level in hierarchy with steps remaining
    $steps[i] \leftarrow 0$
    if $i = k$ **then** ▷ Exit if the highest level has been reached
      break
    else
      $i \leftarrow i + 1$
    end if
  end while
  while $i > 0$ **do** ▷ Proceed downwards from level that has steps remaining
    $subgoal_{i-1} \leftarrow policy_i(state, subgoal_i)$ ▷ Populate the lower-level's subgoal
    if level i in $SUBGOAL$ mode or with λ probability **then**
      set level $i - 1$ to $SUBGOAL$ mode
    end if
    if level i in $EXPLORE$ mode or with γ probability **then**
      set level $i - 1$ to $EXPLORE$ mode
      $subgoal_{i-1} \leftarrow exploration\_policy_i(state, subgoal_{i-1})$
      ▷ Adjust subgoal based on exploration strategy
    end if
    $steps[i] \leftarrow steps[i] + 1$
    $i \leftarrow i - 1$
  end while
  $action \leftarrow policy_0(state, subgoal_0)$ ▷ Get low-level environment action to return
  $exploration\_action \leftarrow exploration\_policy_0(state, subgoal_0)$
  ▷ Adjust action based on exploration strategy
  $steps[0] \leftarrow steps[0] + 1$
  return $exploration\_action$
end procedure

3.3 Exploration Mechanisms

Having constructed this hierarchically-synchronized exploration paradigm, a number of exploratory action selection techniques are surveyed for incorporate into this work's procedure, including a novel adversarially-trained surprise policy model motivated by the theoretical underpinnings of ASP (12).

3.3.1 Gaussian and Ornstein-Uhlenbeck Noise Exploration

The architecture used for the k-level policy hierarchy as proposed in HAC is the DDPG algorithm (15), which is designed to train a deterministic policy in an off-policy manner. As a result of its determinism, should the agent try to explore on-policy, it would likely not attempt a wide enough variety of actions to find meaningful learning signals early. To that end, in an effort to improve exploration for the DDPG policy structure, this work adds noise to their actions at training time. In (15), the authors suggest the use of time-correlated Ornstein-Uhlenbeck (OU) noise, but due to its implementation simplicity, this work also includes uncorrelated, zero-mean Gaussian noise.

A simple signal example illustrates well the different exploration patterns this work hopes to codify into learning strategy with each, as shown in Figure 1. With Gaussian noise, this work generates a decorrelated random signal with zero mean, the effect of which is largely averaged over time due its oscillatory behavior. This provides for a consistent but not dominating exploratory effect, which allows for measured discovery by the agent throughout the learning process. With OU noise, due its

intrinsic correlation to previously generated noise, it will tend to stay in the same direction for longer durations as opposed to quickly canceling itself out, consequently allowing it to increase velocity and unfreeze the position. The velocity and position are consistently pushed in the same direction, which naturally demonstrates a tendency towards exploration. In a following section, this work assesses how these behaviors play with the proposed synchronized exploration procedure.

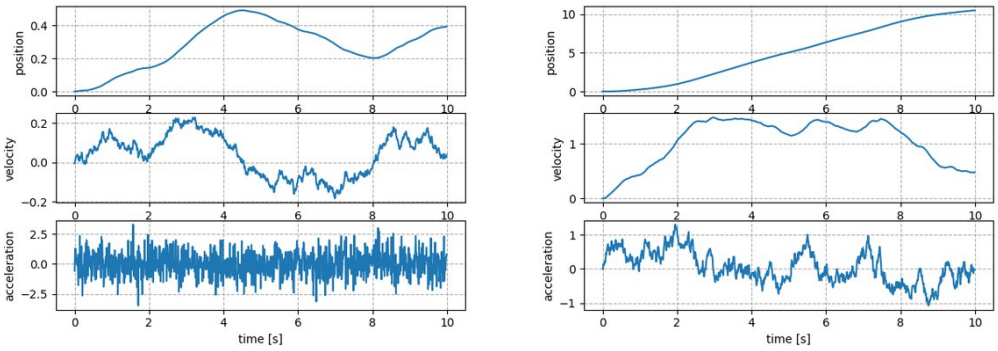


Figure 1: Applying Gaussian vs. OU-Process Noise

3.3.2 Knowledge-based Adversarial Surprise Exploration

The notion of *surprise* as a guiding force for exploratory agent behavior is not a new one, and has been verified across a number of environments ranging from Atari games to continuous control tasks. As noted in (16), simpler approaches such as the Gaussian control noise schema mentioned have merit, but there are many tasks where these methods are insufficient to make any learning progress. Naturally, one might expect the kind of tasks best suited for HRL methods involving nontrivial action sequences may be among those, but leading works in the area such as HAC and HIRO restricted their exploration paradigms to only the simpler methods.

While (16) pursues a strategy built around forming intrinsic rewards that approximate the KL-divergence of the true transition probabilities, this work instead presents a new alternate approach that leverages the principle of autocurricula generation by way of adversarial learning. In the proposed schema, as detailed in **Algorithm 3**, agent Lewis generates not increasingly challenging and complex tasks for agent Clark to complete, but instead generates perturbed actions aimed at surprising Clark. Lewis does so by utilizing the hindsight experience replay buffer as constructed in HAC for the purpose of transforming an action with a suprise-maximizing objective.

Clark receives reward inverse that of Lewis just as Bob received reward inverse of Alice in ASP, but, unlike ASP, does not attempt not to mimic or undo some tasks that Lewis has provided for him. Instead, Clark seeks to minimize how surprised he is by Lewis's generated action perturbations. Clark does so by computing the loss between the next state reached and his prediction for the next state. This approach is found here to provide a highly desirable mechanism for exploration, as Lewis succeeds in surprising Clark early on and gradually becomes predictable to him, aligning well with this work's desires for increased determinism as the agent gets better at the goal task itself over time.

4 Experiments

As shown in Figure 2, the proposed approach was trained and validated on on the above tasks and environments, which the authors believe to be representative of the current baseline for algorithm comparison and can attest to robustness and generalizability. Rates of success, the measures of which are defined below, are averaged over periods of 100 episodes.

The Ring Road and Highway environments are simulated using the Flow (17) computational framework for mixed autonomy traffic control, in which a subset of vehicles (represented in red) are autonomous and are tasked with reducing oscillations in vehicle speeds, known as stop-and-go traffic. The simulation is warmed up for 1000 steps to allow for regular and persistent stop-and-go waves

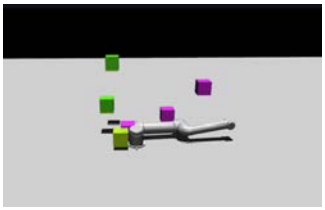
Algorithm 3 Knowledge-based Adversarial Surprise Exploration (Training)

Require: Replay buffers of experienced transitions maintained by the singular HAC coordinator
Note: In $exploration\_policy$, action generated by querying Lewis's policy $\pi_L(state, action)$

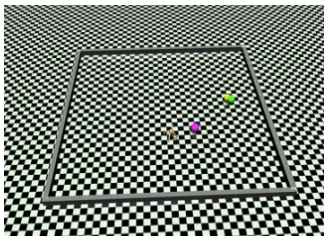
procedure AK($buffer_i$)
    Randomly initialize ϕ_L and ϕ_C for adversarial policies
    $S, A, N, \_ \leftarrow buffer_i$ ▷ Fetch state, action, and next state batches from level's buffer
    $A_e \leftarrow \pi^L(S, A)$ ▷ Generate exploratory actions from Lewis's policy
    $N_p \leftarrow \pi^C(S, A_e)$ ▷ Generate predictions for next states from Clark's policy
    $R_L \leftarrow MSE(N_p, N)$ ▷ Compute Lewis's reward
    $R_C \leftarrow -R_L$ ▷ Set Clark's reward
    $\phi_L \leftarrow update(\pi^L, R_L)$ ▷ Update Lewis's policy weights
    $\phi_C \leftarrow update(\pi^C, R_C)$ ▷ Update Clark's policy weights
end procedure

(a) FLOW Highway

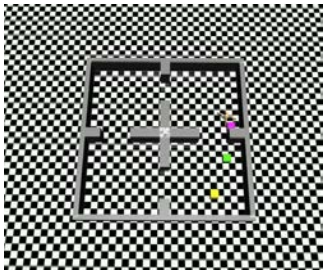
(b) FLOW Ring Road



(c) MuJoCo UR5 Reacher



(d) MuJoCo Ant Reacher



(e) MuJoCo Ant Four Rooms

Figure 2: Training Environments. Two mixed-autonomy traffic control tasks (renderings from (18)) and three MuJoCo control tasks.

to form during resets, and the environment horizons are set to 1000 steps. Following directly the configurations in (18), the simulation step size for Ring Road is set to 0.2 sec/step, while that for Highway is set to 0.4 sec/step. There are 22 vehicles in the network, with one autonomous vehicle among them controlled by a policy learned through an RL procedure such as the method considered in this work. Success is defined by reaching a reward threshold, also as specified in (18).

The UR5 Reacher, Ant Reacher, and Ant Four Rooms environments are simulated using the MuJoCo physics engine for model-based control (19) with time horizons of 600 steps, 500 steps, and 700 steps respectively. Success in each environment is defined by arriving within an L_2 distance of 0.5 at the final step of a given episode.

4.1 Results

In pursuit of a rigorous comparative assessment of the proposed method, experiments were performed with a comprehensive set of baselines and ablations. The first of these and this work's namesake, HAC-E-SAK, incorporates both synchronization and knowledge-based adversarial surprise. Two variations of the method are also included (denoted as HAC-E-SNN and HAC-E-SON), substituting out knowledge-based adversarial surprise for Gaussian noise and Ornstein-Uhlenbeck noise respectively. Pure HAC baselines are also included (modified only to align with the previously detailed IterHAC execution structure), one with 2 hierarchy levels and one with 3.

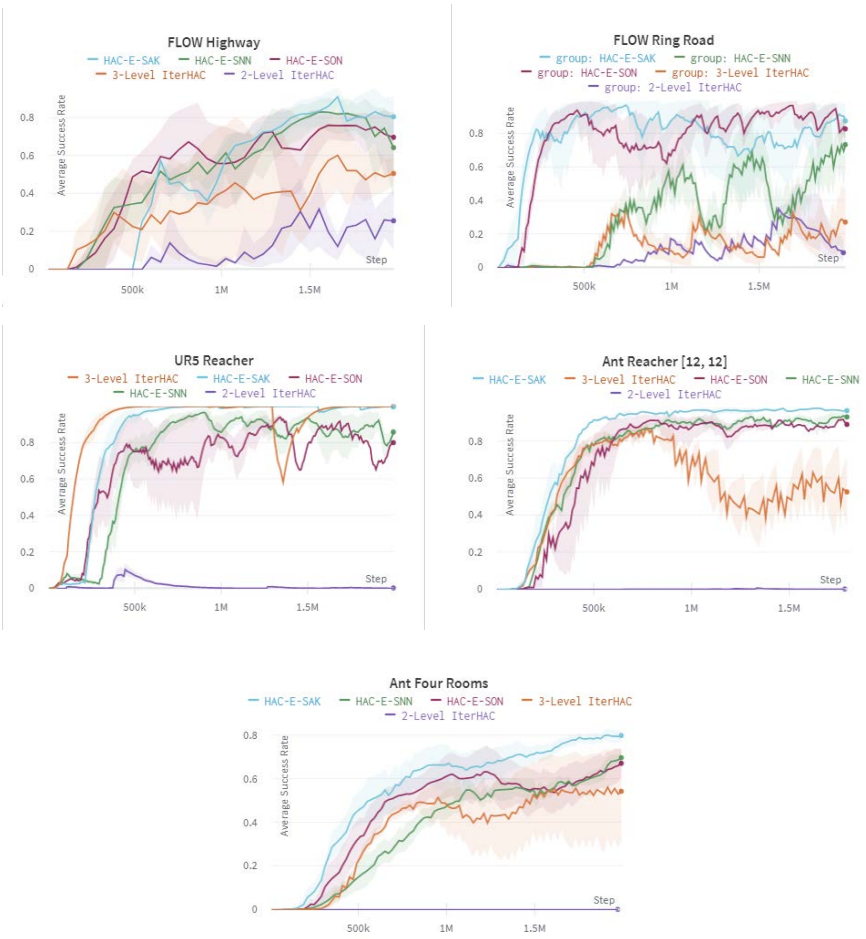


Figure 3: Training Results. All results reported over 10 random seeds, with error bars showing 1 standard deviation.

As shown across each environment in Figure 3, the single best combination of parameters involved all of the key proposed contributions, the synchronized exploration and the knowledge-based adversarial surprise. It is a combination marked by broadly superior stability, accuracy, and sample efficiency.

This combination of methods to both encourage all layers of the hierarchy to explore together, while also maximizing the surprise captured by each exploratory action, led to a quicker convergence to the optimal score and a reduction in the number of steps needed to settle into a stable evaluation pattern. Importantly, the variance measured during training on the MuJoCo environments around the performance curves is markedly less as well, which indicates that this exploration's guided approach is likely minimizing the degree of 'flailing' commonly observed during the initial episodes of Deep RL training. Similar patterns are also present in the mixed-autonomy Flow environments, but to a comparatively lesser degree.

4.2 Ablative Analysis

Through a series of experiments, this work shows that each of the preferred additions introduced are reflected in performance as notable improvements over the baseline and other options considered.

First, considered here is the benefit of synchronized exploration at all levels of the agent, versus the alternative of the traditional, independent exploration described in the original HAC work. When using Gaussian/Normal noise as in HAC-E-SNN, a distinct improvement is seen in convergence caused by synchronization; the first string of successes is achieved at a rate substantially higher as compared to the unmodified exploration scheme. This effect is most visible in the highly complex

model-based MuJoCo physical control tasks, where standard 2-Level HAC is largely unsuccessful. An even more pronounced effect is seen when comparing synchronized to independent exploration using the surprise-based selection as in HAC-E-SAK. The experiments with synchronization using OU noise as in HAC-E-SON produced similar but at times more muted effects. The authors of this work attribute this to the notion of 'overexploration' - the baseline of OU noise is already extremely conducive for exploration, and the multiplying factor of synchronization may at times prevent the agent from exploiting enough of the high-value states it rapidly uncovers during this frenzy. Perhaps most notable in these results is the improvement in performance of the synchronized exploration methods with 2-level hierarchies over pure HAC with a 3-level hierarchy. This comparison offers up the clearest indication that augmenting simpler architectures with hierarchically synchronized exploration allows for the successful completion of tasks at and above rates achieved by unaugmented architectures with markedly higher complexity.

Next, considered are the three action modifiers: Normal, OU, and Knowledge-based Adversarial Surprise. Figure 3 shows all three action modifiers with synchronized exploration across agent levels. The surprise-and-sync setup outperforms all others.

The authors of this work propose that the proposed surprise-based method is effective because its ASP-inspired model essentially shapes the previously-undirected noise in the specific axes of maximum uncertainty. The authors also submit that the synchronization across levels in the hierarchy is essential to accomplish broad 'sweeps' of the state and action space. Though experiments were conducted with relatively shallow hierarchies, in larger trees one would imagine that the value of intelligent exploration will become even more necessary.

5 Conclusion

The combination of the proposed synchronization structure with the proposed adversarial knowledge-based exploration learning strategy outperforms all other presented procedures in each of the presented set of diverse environments, validating the viability of HAC-E-SAK as a robust method for hierarchical learning.

5.1 Future Work

A limitation of HAC-E-SAK's goal-oriented hierarchy is that the highest level requires an explicit and specific goal state for the overall environment. This maintains the pleasantly recursive nature of the hierarchical subgoals, but is an extremely limiting characteristic in open-ended environments. For example, in a task like ant locomotion across a rough terrain, the only relevant parts of the state for goal determination are the torso coordinates; the legs could be in any arbitrary state while still satisfying the authors' notion of success. HAC-E-SAK aims to match a specific goal state and actually disregards the external reward function entirely, so it cannot generalize to a broader class of goal states. Future work could explore adding an additional model to encode this class into a more specific subgoal level to kickstart the hierarchy.

The authors believe that more complex navigation and mixed-autonomy tasks will be excellent candidates for future work exploring HRL. The classical separation between a low-level controller and a high-level planner for navigation in an environment lines up well with the hierarchical structure of methods like HAC. As the scope of real-world robotics and control tasks becomes increasingly broad, it is reasonable to anticipate that HRL will become more and more widely considered and applied.

References

[1] S. Pateria, B. Subagdja, A.-h. Tan, and C. Quek, "Hierarchical reinforcement learning: A comprehensive survey," *ACM Comput. Surv.*, vol. 54, no. 5, jun 2021. [Online]. Available: https://doi.org/10.1145/3453160

[2] R. Parr and S. Russell, "Reinforcement learning with hierarchies of machines," in *Proceedings of the 1997 Conference on Advances in Neural Information Processing Systems 10*, ser. NIPS '97. Cambridge, MA, USA: MIT Press, 1998, p. 1043–1049.

[3] L. Liu, J. Zhang, S. Song, and K. Letaief, "Client-edge-cloud hierarchical federated learning," 06 2020, pp. 1–6.

[4] Z. Huang, F. Liu, and H. Su, "Mapping state space using landmarks for universal goal reaching," in *NeurIPS*, 2019.

[5] A. Levy, G. D. Konidaris, R. W. Platt, and K. Saenko, "Learning multi-level hierarchies with hindsight," in *ICLR*, 2019.

[6] J. Schmidhuber, "Learning to generate subgoals for action sequences," *IJCNN-91-Seattle International Joint Conference on Neural Networks*, vol. ii, pp. 453 vol.2–, 1991.

[7] G. Konidaris and A. Barto, "Skill discovery in continuous reinforcement learning domains using skill chaining." vol. Vol. 22, 01 2009, pp. 1015–1023.

[8] P.-L. Bacon, J. Harb, and D. Precup, "The option-critic architecture," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, ser. AAAI'17. AAAI Press, 2017, p. 1726–1734.

[9] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu, "Feudal networks for hierarchical reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17. JMLR.org, 2017, p. 3540–3549.

[10] M. Andrychowicz, D. Crow, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *NIPS*, 2017.

[11] O. Nachum, S. S. Gu, H. Lee, and S. Levine, "Data-efficient hierarchical reinforcement learning," in *NeurIPS*, 2018.

[12] S. Sukhbaatar, I. Kostrikov, A. D. Szlam, and R. Fergus, "Intrinsic motivation and automatic curricula via asymmetric self-play," *ArXiv*, vol. abs/1703.05407, 2018.

[13] OpenAI, M. Plappert, R. Sampedro, T. Xu, I. Akkaya, V. Kosaraju, P. Welinder, R. D'Sa, A. Petron, H. P. de Oliveira Pinto, A. Paino, H. Noh, L. Weng, Q. Yuan, C. Chu, and W. Zaremba, "Asymmetric self-play for automatic goal discovery in robotic manipulation," *ArXiv*, vol. abs/2101.04882, 2021.

[14] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State, "Isaac gym: High performance GPU based physics simulation for robot learning," in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. [Online]. Available: https://openreview.net/forum?id=fgFBtYgJQX_

[15] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. M. O. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2016.

[16] J. Achiam and S. S. Sastry, "Surprise-based intrinsic motivation for deep reinforcement learning," *ArXiv*, vol. abs/1703.01732, 2017.

[17] C. Wu, A. R. Kreidieh, K. Parvate, E. Vinitsky, and A. M. Bayen, "Flow: A modular learning framework for mixed autonomy traffic," *IEEE Transactions on Robotics*, vol. 38, no. 2, pp. 1270–1286, apr 2022. [Online]. Available: https://doi.org/10.1109%2Ftro.2021.3087314

[18] A. R. Kreidieh, G. Berseth, B. Trabucco, S. Parajuli, S. Levine, and A. M. Bayen, "Inter-level cooperation in hierarchical reinforcement learning," 2019. [Online]. Available: https://arxiv.org/abs/1912.02368

[19] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," 10 2012, pp. 5026–5033.