Project 1: Data-Efficient Hierarchical Reinforcement Learning

Somya Gupta[1], **Yashwant Mahajan**[2], **Ayush Kumar**[3]
[1]211049, [2]201156, [3]200246
[1]CHM, [2]EE, [3]EE
somyavg21@iitk.ac.in, yashwantm20@iitk.ac.in, ayushk20@iitk.ac.in

Abstract

Hierarchical Reinforcement Learning (HRL) simplifies complex tasks by organizing them into a hierarchy of subtasks. We propose a two-level hierarchy where the higher-level policy assigns subgoals to lower-level controllers, and the lower-level policy executes actions to achieve these subgoals.Since, existing HRL algorithms often struggle with efficiency and generality,we try to improve on them by introducing intrinsic parameterized rewards and adopt off-policy methods. However, using off-policy methods presents a challenge:changes in lower-level behaviors affect the action space for the higher-level policy. We address this with an off-policy correction mechanism. Our resulting HRL agent, HIRO, demonstrates superior performance and sample efficiency compared to existing methods.

1 Introduction

Hierarchical reinforcement learning (HRL) involves training multiple layers of policies to make decisions at different levels of abstraction, allowing for longer-term planning and exploration in complex environments. However, practical implementation of HRL poses some challenges. Key issues include inducing distinct behavior in lower-level policies, defining high-level policy actions, and efficiently training multiple policies. Previous approaches have shown promise but often lack generality and rely on expensive on-policy training, limiting their scalability and applicability.For generality, we propose to take advantage of the state observation provided by the environment to the agent. We let the high-level actions be goal states and reward the lower-level policy for performing actions which yield it an observation close to matching the desired goal. In this way, our HRL setup does not require a manual or multi-task design and is fully general.There exists previous work on this idea but they represented goals and rewarded matching observations within a learned embedding space,we on the other hand use the state observations in their raw form. This significantly simplifies the learning.While these goal-proposing methods are very general, they require training with on-policy RL algorithms, which are generally less efficient than off-policy methods.But off policy training poses a challenge,since the lower-level policy is changing underneath the higher-level policy, a sample observed for a certain high-level action in the past may not yield the same low-level behavior in the future,this amounts to a non-stationary problem for the higher-level policy. We remedy this issue by introducing an off-policy correction, which relabels an experience in the past with a high-level action chosen to maximize the probability of the past lower-level actions. In this way, we are able to use past experience for training the higher-level policy, taking advantage of off policy RL methods.In summary, we present a method to train a multi-level HRL agent that distinguishes itself from previous methods by its general applicability and data efficiency. Compared to other published HRL methods, our approach demonstrates superior final performance and learning speed.

2 Problem Defintion

We adopt the standard continuous control RL setting, in which an agent interacts with an environment over periods of time according to a behavior policy μ. At each time step t, the environment produces a state observation $s_t \in R^{d_s}$.The agent then samples an action $a_t \sim \mu(s_t), \quad a_t \in R^{d_a}$ and applies the action to the environment. The environment then yields a reward R_t sampled from an unknown reward function $R(s_t, a_t)$ and either terminates the episode at state s_T or transitions to a new state s_{t+1} sampled from an unknown transition function $f(s_t, a_t)$. The agent's goal is to maximize the expected future discounted reward $E_{s_0:T}, a_{0:T-1}, R_{0:T-1}[\sum_{i=0}^{T-1} \gamma^i R_i]$,where $0 \leq \gamma < 1$ is a userspecified discount factor. A

well-performing RL algorithm will learn a good behavior policy μ from(ideally a small number of) interactions with the environment.

2.1 Off-Policy Temporal Difference Learning

Temporal difference learning is a powerful paradigm in RL, in which a policy may be learned efficiently from state-action-reward transition tuples $(s_t, a_t, R_t, s_t + 1)$ collected from interactions with the environment. In our HRL method, we utilize the TD3 learning algorithm, a variant of the popular DDPG algorithm.DDPG is a reinforcement learning algorithm for continuous action spaces. It combines actor-critic architecture with experience replay and target networks for stability. The actor network learns a deterministic policy, while the critic network estimates the action-value function. It uses noise for exploration and updates networks periodically. $E(s_t, a_t, s_{t+1}) = (Q_\theta(s_t, a_t) - R_t - \gamma Q_\theta(s_{t+1}, \mu_\phi(s_{t+1})))^2$ its behavior policy is augmented with Gaussian noise.Therefore, actions are collected as $a_t \sim \mathcal{N}(\mu_\phi(s_t), \sigma)$ for fixed standard deviation , which we will shorten as $a_t \sim \mu_\phi(s_t)$. We will take advantage of the fact that the behavior policy is stochastic for the off-policy correction in our HRL method. TD3 makes several modifications to DDPG's learning algorithm to yield a more robust and stable procedure. Its main modification is using an ensemble over Q-value models and adding noise to the policy when computing the target value

3 Related Work

3.1 HIERARCHICAL REINFORCEMENT LEARNING BY DISCOVERING INTRINSIC OPTIONS

HIDIO encourages lower-level option learning that is independent of the task at hand, requiring few assumptions or little knowledge about the task structure. These options are learned through an intrinsic entropy minimization objective conditioned on the option sub-trajectories. The learned options are diverse and task-agnostic. We assume little prior knowledge about the task structure, except that it can be learned through a hierarchy of two levels. The higher-level policy (the scheduler), is trained to maximize environment reward, while the lower-level policy (the worker) is trained in a self-supervised manner to efficiently discover options that are utilized by to accomplish tasks.

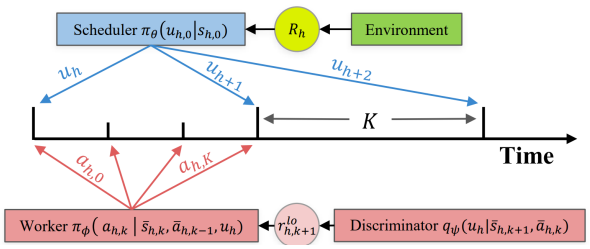


Figure 1: HIERARCHICAL REINFORCEMENT LEARNING BY DISCOVERING INTRINSIC OPTIONS

3.2 FeUdal Networks for Hierarchical Reinforcement Learning

Framework employs a Manager module and a Worker module. The Manager operates at a lower temporal resolution and sets abstract goals which are conveyed to and enacted by the Worker. The Worker generates primitive actions at every tick of the environment. The decoupled structure of FuN conveys several benefits – in addition to facilitating very long timescale credit assignment it also encourages the emergence of sub-policies associated with different goals set by the Manager. These properties allow FuN to dramatically outperform a strong baseline agent on tasks that involve long-term credit assignment or memorisation

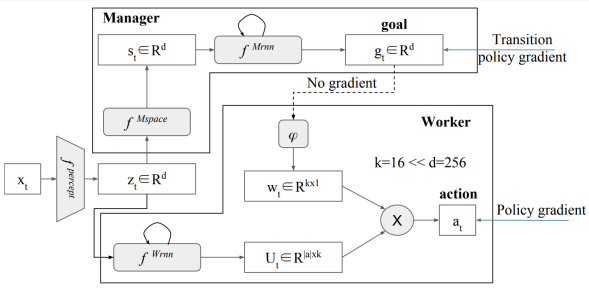


Figure 2: FeUdal Networks

3.3 HIERARCHICAL DECISION TRANSFORMER

We employ two decision transformer models in the form of a high-level mechanism and a low-level controller. The high-level mechanism guides the low-level controller through the task by selecting sub-goal states, based on the history of sub-goals and states, for the low-level controller to try to reach. The low-level controller is conditioned on the history of past states, sub-goals and actions to select the appropriate action. By reaching each sub-goal, the controller gets closer to completing the task. With this architecture, the model doesn't

require a user to specify the initial value of desired rewards for a given task and becomes more robust to sparse rewards and tasks with long episodes

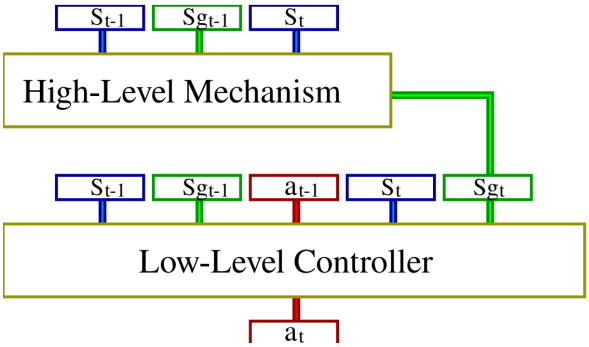
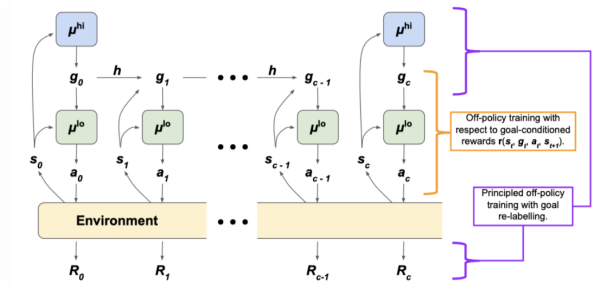


Figure 3: HIERARCHICAL DECISION TRANSFORMER

4 Methodology/Algorithm Description

4.1 Hierarchy of Two Policies



We introduce a hierarchical RL setup with two layers: a lower-level policy μ_{lo} and a higher-level policy μ_{hi}. The higher-level policy operates at a coarser abstraction layer, setting goals for the lower-level policy, corresponding directly to states the lower-level policy aims to reach. At each time step t, the environment provides an observation state s_t. The higher-level policy observes the state and produces a high-level action (or goal) $g_t \in R^{d_s}$, either by sampling from its policy $g_t \sim \mu_{hi}$ when $t \equiv 0c$, or using a fixed goal transition function $g_t = h(s_{t-1}, g_{t-1}, s_t)$. This temporal abstraction ensures high-level decisions are made only every c steps. The lower-level policy μ_{lo} observes the state s_t and goal g_t, producing a low-level atomic action $a_t \sim \mu_{lo}(s_t, g_t)$. The environment yields a reward R_t sampled from an unknown reward function $R(s_t, a_t)$ and transitions to a new state s_{t+1} sampled from an unknown transition function $f(s_t, a_t)$. The higher-level controller provides the lower-level with an intrinsic reward $r_t = r(s_t, g_t, a_t, s_{t+1})$ using a parameterized reward function r. The lower-level policy stores the experience $(s_t, g_t, a_t, r_t, s_{t+1}, h(s_t, g_t, s_{t+1}))$ for off-policy training. The higher-level policy collects environment rewards R_t and, every c time steps, stores the higher-level transition $(s_{t:t+c-1}, g_{t:t+c-1}, a_{t:t+c-1}, R_{t:t+c-1}, s_{t+c})$ for off-policy training.

4.2 Parameterized Rewards

Our higher-level policy generates goals g_t representing desired changes in state observations at step t, aiming for the lower-level agent to produce observations close to $s_t + g_t$. To maintain the goal's absolute position regardless of state changes, we define the goal transition model h as:
$h(s_t, g_t, s_{t+1}) = s_t + g_t - s_{t+1}$.
The intrinsic reward is parameterized based on the distance between the current observation and the goal observation:
$r(s_t, g_t, a_t, s_{t+1}) = -||s_t + g_t - s_{t+1}||^2$.
This rewards the lower-level policy for actions leading to observations close to $s_t + g_t$. We use all positional observations as the representation for g_t, without distinguishing between root position or joints, ensuring a generic and widely applicable choice of goal space. The lower-level policy can be trained using standard methods by incorporating g_t as an additional input.The policy μ_{lo}^{ϕ} is trained to maximize $Q_{lo}^{\theta}(s_t, g_t, \mu_{lo}^{\phi}(s_t, g_t))$ for all sampled state-goal tuples (s_t, g_t). This method, using parameterized rewards, has been studied previously and is chosen for its generality. Unlike prior approaches, our method directly uses the state observation as the goal, allowing the lower-level policy to receive reward signals immediately. This produces substantially better results in our experiments.

4.3 Off-Policy Corrections for Higher-Level Training

Previous two-level HRL designs often need on-policy training due to non-stationarity caused by lower-level policy changes. Off-policy algorithms, more sample-efficient, are favored. We tackle off-policy training for the higher-level policy to enhance real-world applicability.We would like to take the higher-level transition tuples $(s_{t:t+c-1}, g_{t:t+c-1}, a_{t:t+c-1}, R_{t:t+c-1}, s_{t+c})$which are collected by the higher-level policy and convert them to state-action-reward transitions $s\left(s_t, g_t, \sum_{i=t}^{t+c-1} R_i, s_{t+c}\right)$.We observe that past high-level goals g_t can be adjusted to better match

observed actions given the current policy μ_{lo}. By relabeling high-level actions to $\tilde{g}_t$, maximizing μ_{lo} probability, we address off-policy issues in transitions $(s_t, g_t, PR_{t:t+c-1}, s_{t+c})$.To maximize this quantity practically, we compute the log probability for several candidate goals $\tilde{g}t$ and choose the one with the highest value for relabeling. We sample eight goals randomly from a Gaussian distribution centered at $st + c - s_t$, including the original goal g_t and a goal representing the difference $s_{t+c} - s_t$ for a total of 10 candidates. This diverse set helps approximate the argmax of the log probability,$\log \mu_{lo}(a_{t:t+c-1}|s_{t:t+c-1}, \tilde{g}_{t:t+c-1})$. while considering problem knowledge.

5 Model Results

The paper used 4 environments namely Ant Gather, Ant Maze, Ant Push and Ant Fall. Performance of the best policy obtained in 10M steps of training, averaged over 10 randomly seeded trials with standard error. Comparisons are to variants of FuN , SNN4HRL , and VIME . Even after extensive hyper-parameter searches, we were unable to achieve competitive performance from the baselines on any of our tasks.

	Ant Gather	Ant Maze	Ant Push	Ant Fall
HIRO	**3.02±1.49**	**0.99±0.01**	**0.92±0.04**	**0.66±0.07**
FuN representation	0.03 ± 0.01	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
FuN transition PG	0.41 ± 0.06	0.0 ± 0.0	0.56 ± 0.39	0.01 ± 0.02
FuN cos similarity	0.85 ± 1.17	0.16 ± 0.33	0.06 ± 0.17	0.07 ± 0.22
FuN	0.01 ± 0.01	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
SNN4HRL	1.92 ± 0.52	0.0 ± 0.0	0.02 ± 0.01	0.0 ± 0.0
VIME	1.42 ± 0.90	0.0 ± 0.0	0.02 ± 0.02	0.0 ± 0.0

Figure 4: Result Comparisons

6 Environments Used

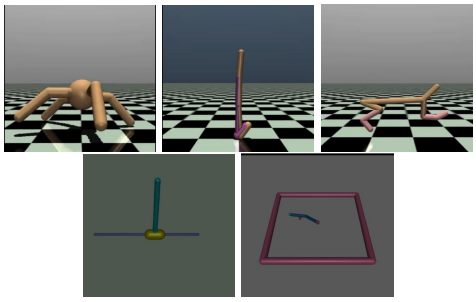


Figure 5: Enviroment used

- InvertedPendulum-v2: Represents a simple inverted pendulum system where the goal is to balance the pendulum upright by applying appropriate forces.
- Walker2d-v2: Represents a 2D bipedal walker robot with multiple joints that can move in various directions. The goal is to make the walker walk or run forward while maintaining balance.
- Ant-v2: Represents a 3D ant-like robot with multiple legs that can move in various directions. The goal is to make the ant walk or run forward while maintaining balance.
- Reacher-v2: Represents a 2D robotic arm with two joints. The goal is to control the robotic arm to reach a specific target position.
- HalfCheetah-v4: Represents a 2D half-cheetah robot with multiple joints that can move in various directions. The goal is to make the half-cheetah run forward while maintaining balance.

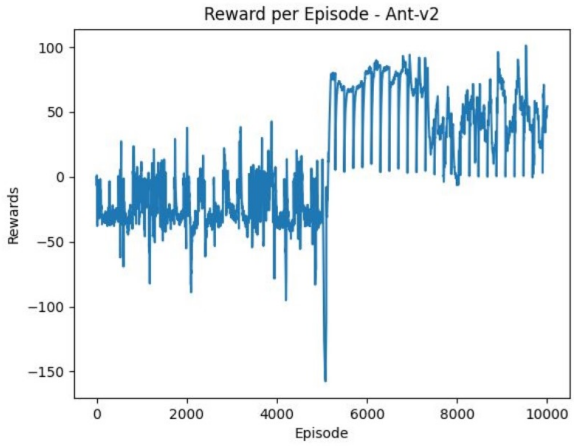
7 Experiments

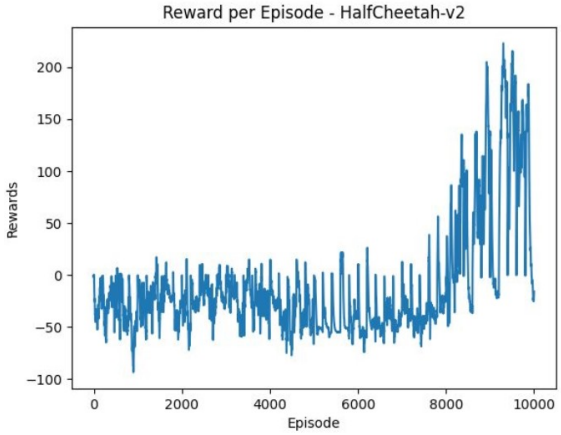
Our study employs five distinct environments: InvertedPendulum, Walk2D, Ant, HalfCheetah, and Reacher. We assessed the performance of the best policy after 5,000 training steps. Overall, the model demonstrated commendable performance across most environments. For each environment, we generated plots depicting total rewards versus episodes. To optimize results, adjustments were made to the upper policy rewards, aligning them with the objectives of each environment.

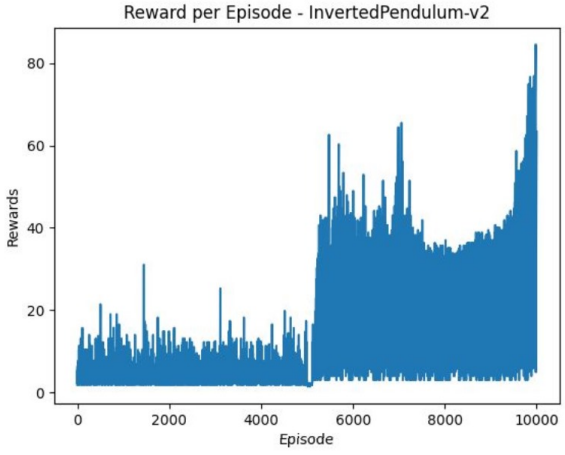
8 GitHub

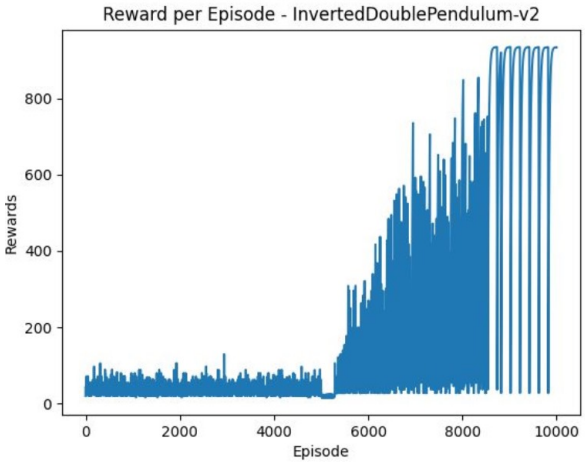
Please visit my GitHub repository: HIRO

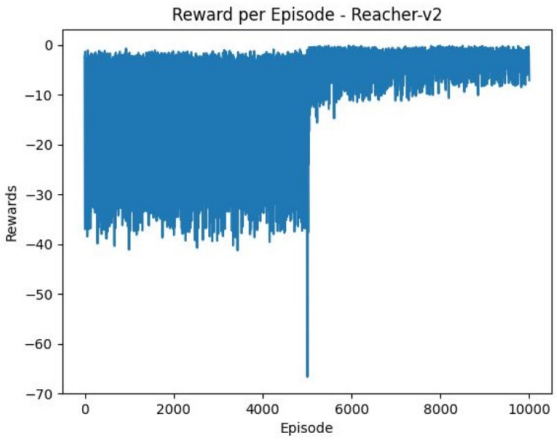
9 Plots

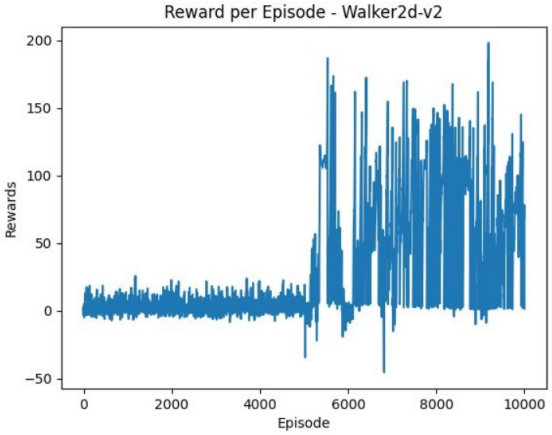












10 Error Analysis

Our experiments revealed some limitations that could be addressed in future iterations. First, we observed sudden spikes in rewards across various environments. This might be due to the limited number of training steps used and the inherent complexity of the environments themselves. Second, the reward functions used might not have been optimally designed for these environments, potentially hindering efficient policy learning. Finally, envi-

ronments like Walk2d-v2 and Ant-v2 presented particular challenges due to their complexity, resulting in less than ideal performance. These observations highlight the importance of using sufficient training steps, carefully crafting reward functions for specific environments, and potentially adapting the algorithm for even greater effectiveness in handling complex tasks.

11 Possible Improvements

- Enhanced Off-Policy Correction: The current implementation utilizes random sampling for off-policy correction. This can be improved by incorporating more sophisticated sampling techniques. One potential avenue is to leverage importance sampling, which prioritizes samples that are more informative for the learning process.

- Latent Variable Goal Estimation: The current method relies on directly using states as goals. We can explore using Variational Autoencoders (VAEs) to estimate a latent variable representation (g) of the goal state derived from the original state. This latent representation could potentially capture more abstract and relevant goal information, leading to better policy learning.

- Neural Network Integration: The current framework does not explicitly utilize neural networks. Future work could explore incorporating neural network architectures like Convolutional Neural Networks (CNNs) to better extract features from the state space, particularly in complex environments with spatial dependencies. Additionally, attention mechanisms could be used to focus the agent's attention on crucial aspects of the state relevant to achieving the high-level goal.

12 Individual Contribution

Member	Tasks
Somya Gupta	Literature Review Worked on Baseline Model Researched on improvements in current model
Yashwant Mahajan	Literature Review Worked on Baseline Model Researched on improvements in current model
Ayush Kumar	Literature Review Worked on Baseline Model Researched on improvements in current model

13 Conclusion

We have read a method for training a two-layer hierarchical policy .The experiments show that the method outperforms prior HRL algorithms and can solve exceedingly complex tasks that combine locomotion and rudimentary object interaction.

References

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

Pierre-Luc Bacon, Jean Harb, and Doina Precup. 2017. The option-critic architecture.

Gabriel Barth-Maron, Matthew W Hoffman, David Budden, Will Dabney, Dan Horgan, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. 2018. Distributed distributional deterministic policy gradients.

Andrew G Barto and Sridhar Mahadevan. 2003. Recent advances in hierarchical reinforcement learning.

Nuttapong Chentanez, Andrew G Barto, and Satinder P Singh. 2005. Intrinsically motivated reinforcement learning.

Christian Daniel, Gerhard Neumann, and Jan Peters. 2012. Hierarchical relative entropy policy search.

Peter Dayan and Geoffrey E Hinton. 1993. Feudal reinforcement learning.

(Andrychowicz et al., 2017) (Bacon et al., 2017) (Barth-Maron et al., 2018) (Barto and Mahadevan, 2003) (Chentanez et al., 2005) (Daniel et al., 2012) (Dayan and Hinton, 1993)