Data-Efficient Hierarchical Reinforcement Learning

CS780: Deep Reinforcement Learning

Group No. - 1
Ayush Kumar (200246)
Somya Gupta (211049)
Yashwant Mahajan (201156)

Supervisor: Dr. Ashutosh Modi

Problem Addressed

The paper focuses on applying hierarchical reinforcement learning (HRL) to real-world scenarios. The key issues discussed are:

1. Task-specific Design and On-policy Training: The majority of current HRL methods require careful task-specific design and on-policy training. This limits their applicability in real-world scenarios, where adaptability and generality are crucial.

2. Generality and Efficiency: The goal is to develop HRL algorithms that are both general and efficient. Generality implies that these algorithms do not rely on onerous additional assumptions beyond standard RL algorithms, making them applicable to a wide range of tasks. Efficiency, in this context, refers to the ability to work with modest numbers of interaction samples, which is essential for real-world problems such as robotic control.

Problem Addressed

3. Supervised Learning of Lower-level Controllers: A scheme is proposed where lower-level controllers are supervised with goals that are learned and proposed automatically by the higher-level controllers. This contributes to the generality of the approach.

4. Off-policy Experience for Training: To enhance efficiency, the paper suggests using off-policy experience for both higher- and lower-level training. This introduces a challenge due to changes in lower-level behaviors affecting the action space for the higher-level policy, and an off-policy correction is introduced to address this challenge.

Motivation

The motivation lies in overcoming limitations of current hierarchical reinforcement learning (HRL) methods, which often require task-specific design and on-policy training.

The goal is to develop a general and efficient HRL algorithm, exemplified by the introduced HIRO agent.

HIRO (Hierarchical Off-policy Reinforcement Learning): The proposed HRL agent, named HIRO, is introduced as a result of the developed approach. HIRO is designed to be generally applicable and highly sample-efficient, with the ability to learn complex behaviors for simulated robots using substantially fewer environment interactions than on-policy algorithms.

Use cases

Robotics:

HIRO can be used to train simulated robots to perform complex tasks, such as navigation, manipulation, and locomotion. Its sample efficiency makes it suitable for training robots in simulation before transferring learned behaviors to real-world robots.

Games:

HIRO can be applied to learn hierarchical strategies in games, where the agent needs to perform a sequence of actions to achieve a high-level goal. This can be useful for developing AI agents for complex strategy games or multi-agent environments.

Autonomous Vehicles:

HIRO can be used to train autonomous vehicles to perform complex driving tasks, such as lane changing, merging, and navigating through intersections. The hierarchical nature of the algorithm allows the agent to learn high-level driving policies and low-level control strategies.

Use cases

Resource Management:

HIRO can be applied to optimize resource allocation in complex systems, such as energy management in smart grids or inventory management in supply chains. The hierarchical structure of the algorithm can model the decision-making process at different levels of abstraction.

Industrial Automation:

HIRO can be used to optimize production processes in manufacturing environments, where the agent needs to learn complex sequences of actions to maximize efficiency and minimize downtime.

Introduction

What is HRL?

Hierarchical reinforcement learning (HRL) decomposes a Reinforcement Learning problem into a hierarchy of subproblems or subtasks such that higher-level parent-tasks invoke lower-level child tasks as if they were primitive actions.

What do we do?

We introduce a method to train a multi-level HRL agent that stands out from previous methods by being both generally applicable and data-efficient

Introduction (Contd.)

How we do what we do?

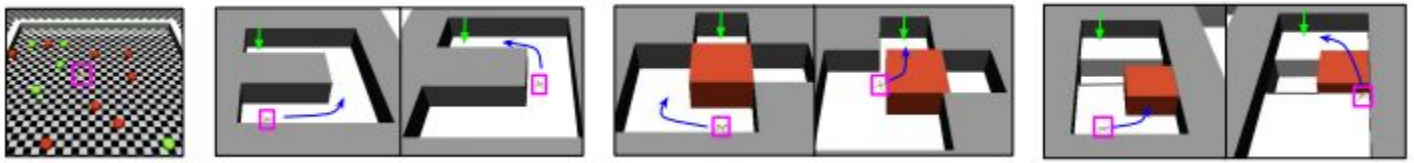
Our method achieves generality by training the lower-level policy to reach goal states learned and instructed by the higher-levels. In contrast to prior work that operates in this goal-setting model, we use states as goals directly, which allows for simple and fast training of the lower layer. Moreover, by using off-policy training with 2 our novel off-policy correction, our method is extremely sample-efficient.

Testing

We evaluate our method on several difficult environments. These environments require the ability to perform exploratory navigation as well as complex sequences of interaction with objects in the environment

Introduction (Contd.)

Testing



While these tasks are unsolvable by existing non-HRL methods, we find that our HRL setup can learn successful policies. When compared to other published HRL methods, we also observe the superiority of our method, in terms of both final performance and speed of learning.

Methodology

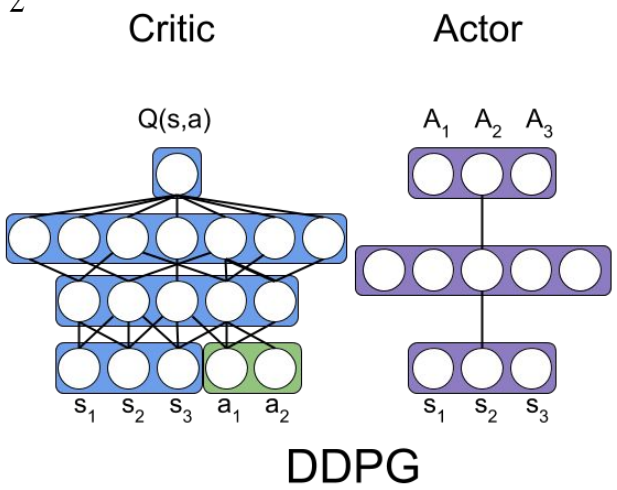
Off-Policy Temporal Difference Learning

In our HRL method, we utilize the TD3 learning algorithm, a variant of the popular DDPG algorithm for continuous control.

$$E(s_t, a_t, s_{t+1}) = (Q_\theta(s_t, a_t) - R_t - \gamma Q_\theta(s_{t+1}, \mu_\phi(s_{t+1})))^2$$

Although DDPG trains a deterministic policy μφ, its behavior policy, which is used to collect experience during training is augmented with Gaussian noise (fixed σ)

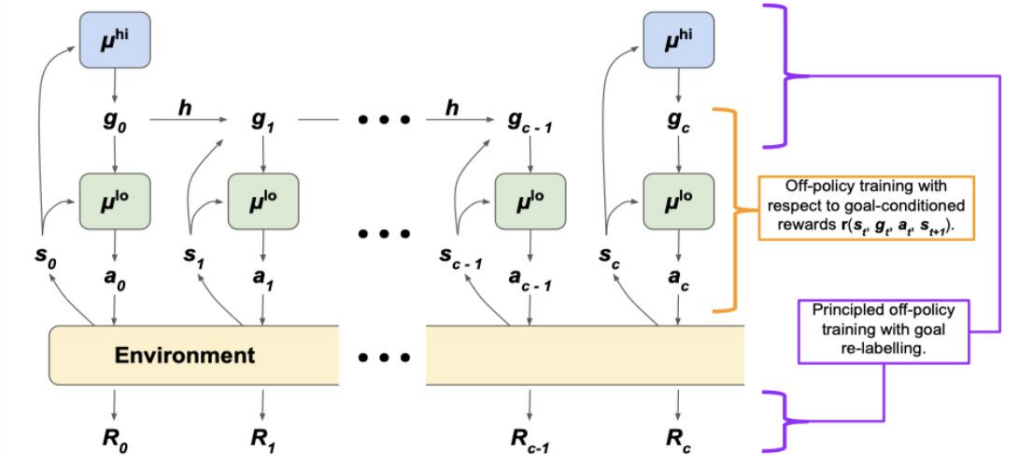
$$a_t \sim \mu_\phi(s_t)$$



Hierarchy of Two Policies

We extend the standard RL setup to a hierarchical two-layer structure.

- The higher-level policy operates at a coarser layer of abstraction and sets goals to the lower-level policy.
- The higher-level controller provides the lower-level with an intrinsic reward
- The lower-level and higher level policy will store the experience for off-policy training.
- The higher-level policy collects the environment rewards R every c time steps



A High-Level Action (or goal):
Sampling from $g_t \sim \mu^{hi}$
Or
fixed goal transition function $g_t = h(s_{t-1}, g_{t-1}, s_t)$

Low-Level Action: $a_t \sim \mu^{lo}(s_t, g_t)$

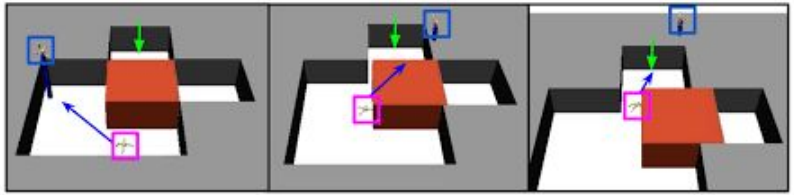
Parameterized Rewards

Our higher-level policy produces goals gt indicating desired relative changes in state observations. $s_{t+c} \sim s_t + g_t$

To maintain the same absolute position of the goal regardless of state change, the goal transition model h is defined as: $h(s_t, g_t, s_{t+1}) = s_t + g_t - s_{t+1}$

We define the intrinsic reward as a parameterized reward function based on the distance between the current observation and the goal observation:

$$r(s_t, g_t, a_t, s_{t+1}) = -||s_t + g_t - s_{t+1}||^2$$



The lower-level policy may be trained using standard methods by simply incorporating g_t as an additional input into the value and policy models. Eg-Bellman Error will be now written as

$$(Q_\theta^{lo}(s_t, g_t, a_t) - r(s_t, g_t, a_t, s_{t+1}) - \gamma Q_\theta^{lo}(s_{t+1}, g_{t+1}, \mu_\phi^{lo}(s_{t+1}, g_{t+1})))^2$$

Off-Policy Corrections for Higher-Level Training

We would like to take the higher-level transition tuples $(s_{t:t+c-1}, g_{t:t+c-1}, a_{t:t+c-1}, R_{t:t+c-1}, s_{t+c})$

convert them to state-action-reward $(s_t, g_t, \sum_t R_{t:t+c-1}, s_{t+c})$

Introducing a correction that translates old transitions into ones that agree with the current lower-level controller:

Re-labeling the high-level transition with a different high-level action $\tilde{g}_t$ chosen to maximize the probability $\mu^{\text{lo}}(a_{t:t+c-1} | s_{t:t+c-1}, \tilde{g}_{t:t+c-1})$

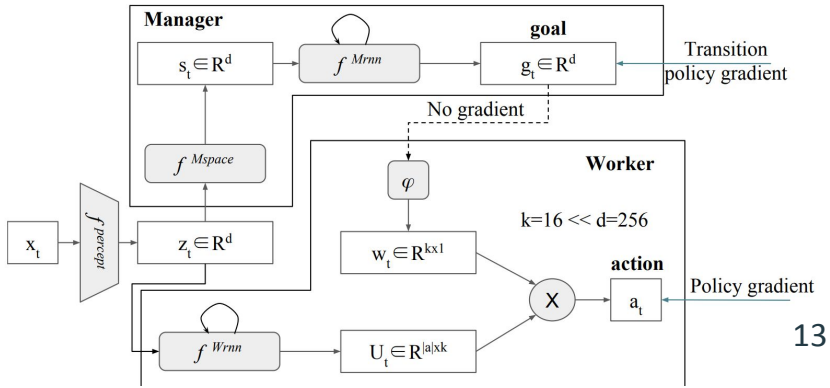
Since the behavior policy is stochastic, the log probability may be computed as:

$$\log \mu^{lo}(a_{t:t+c-1} | s_{t:t+c-1}, \tilde{g}_{t:t+c-1}) \propto -\frac{1}{2} \sum_{i=t}^{t+c-1} \| a_i - \mu^{lo}(s_i, \tilde{g}_i) \|_2^2 + \text{const.}$$

To approximately maximize this quantity in practice, we compute this log probability for a number of goals $\tilde{g}_t$,and choose the maximal goal to re-label the experience

Related Works

FeUdal Networks for Hierarchical Reinforcement Learning: Framework consists of two main components: a Manager module and a Worker module. The Manager operates at a lower temporal resolution and defines high-level goals, which are then executed by the Worker. The Worker generates basic actions continuously within the environment. This decoupled architecture of our framework, called FuN, offers several advantages. It not only enables effective credit assignment over long timescales but also promotes the emergence of sub-policies aligned with various goals defined by the Manager.



Environments Used

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. The state includes the positions, velocities, and orientations of the ant's body parts, and the action space consists of continuous actions representing joint torques.

Reacher-v2: Represents a 2D robotic arm with two joints. The goal is to control the robotic arm to reach a specific target position. The state includes the positions and velocities of the arm's joints, and the action space consists of continuous actions representing joint torques.

HalfCheetah-v2: 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. The state includes the positions, velocities, and orientations of the half-cheetah's body parts, and the action space consists of continuous actions representing joint torques.

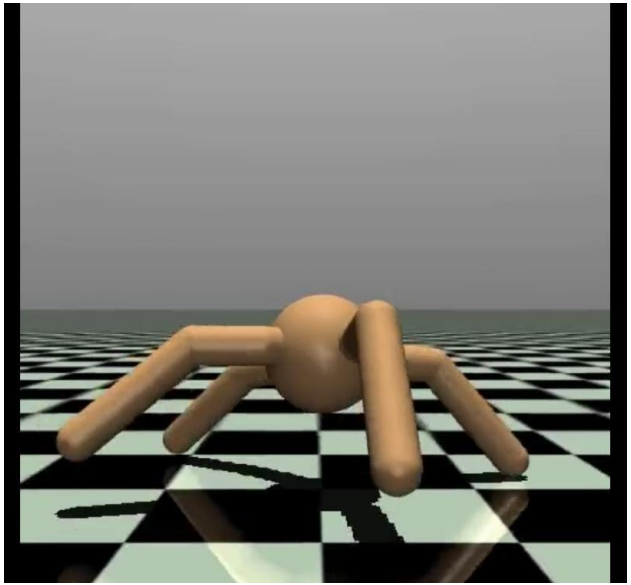
Environments Used

InvertedPendulum-v2: Represents a simple inverted pendulum system where the goal is to balance the pendulum upright by applying appropriate forces. The state includes the angle of the pendulum and its angular velocity, and the action space consists of a single continuous action representing the force to apply to the pendulum.

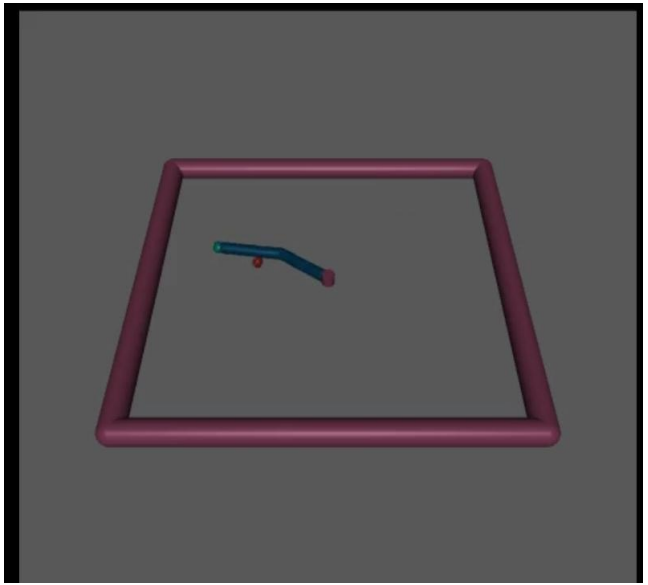
InvertedDoublePendulum-v2: Similar to the InvertedPendulum-v2 but with an additional pendulum attached to the end of the first pendulum. This makes the system more complex and challenging to control.

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. The state includes the positions, velocities, and orientations of the walker's body parts, and the action space consists of continuous actions representing joint torques.

Environments Used

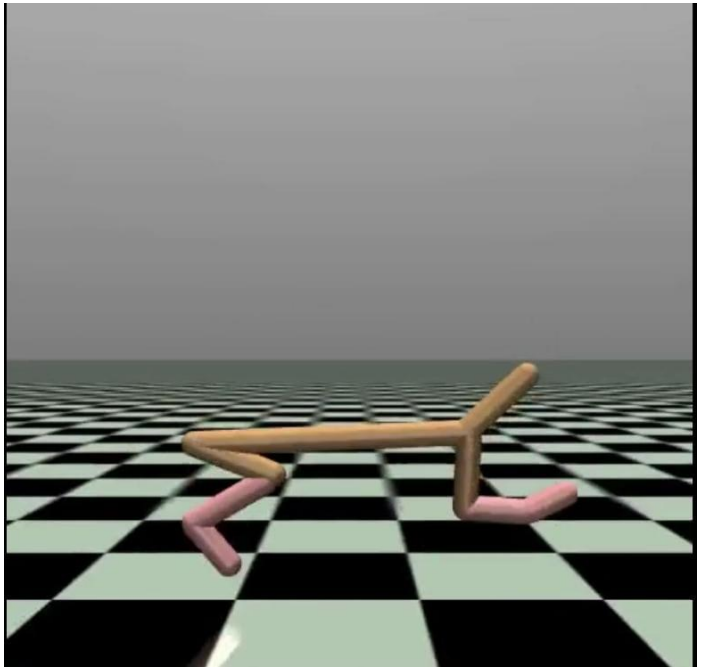


Ant-v2

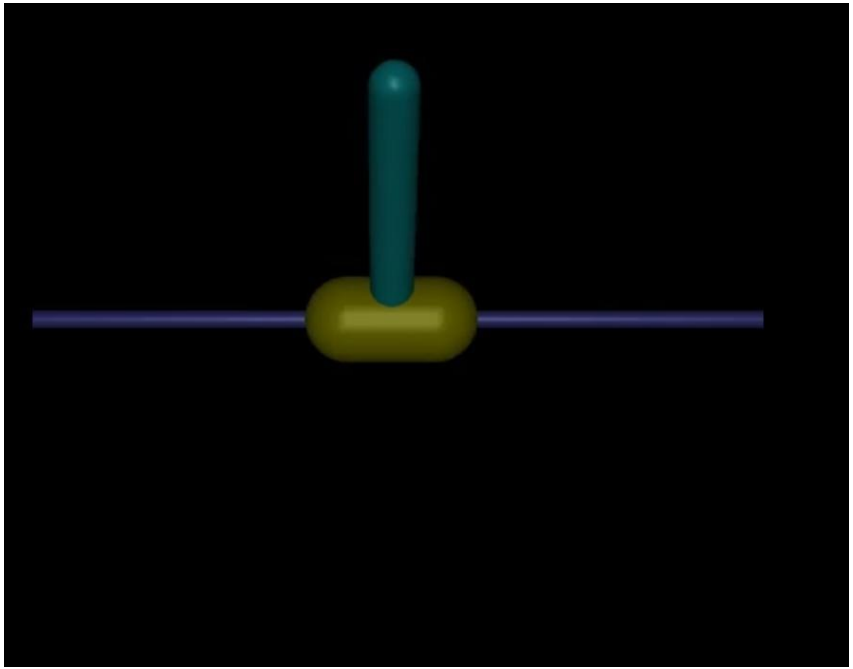


Reacher-v2

Environments Used

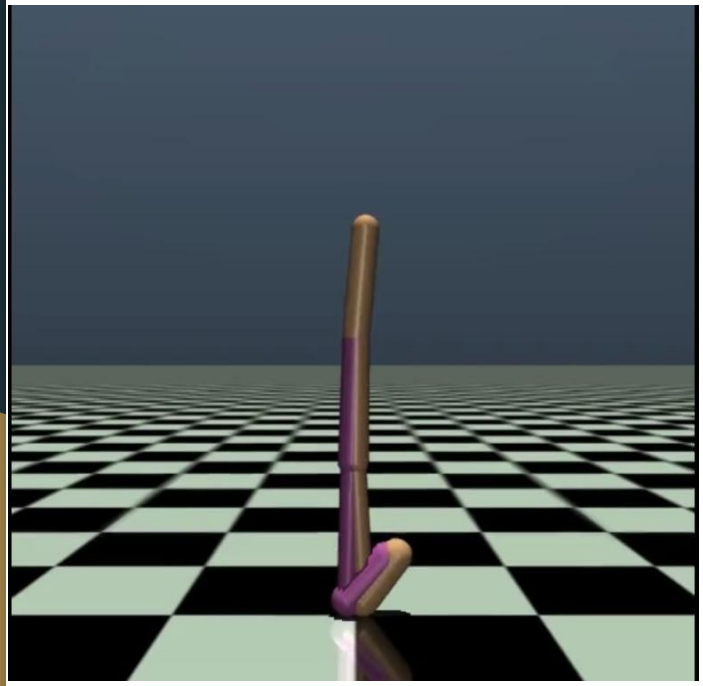


HalfCheetah-v2

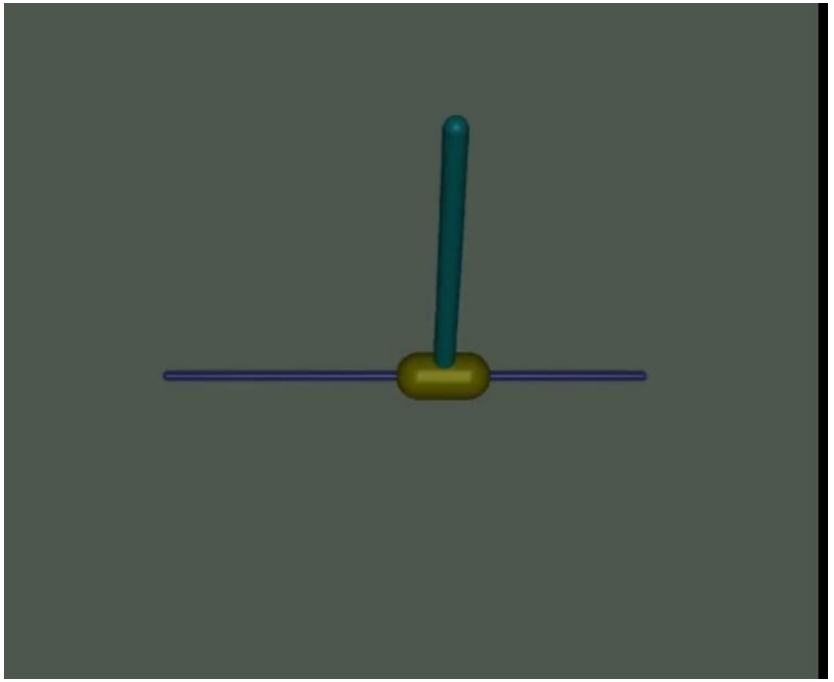


InvertedPendulum-v2

Environments Used

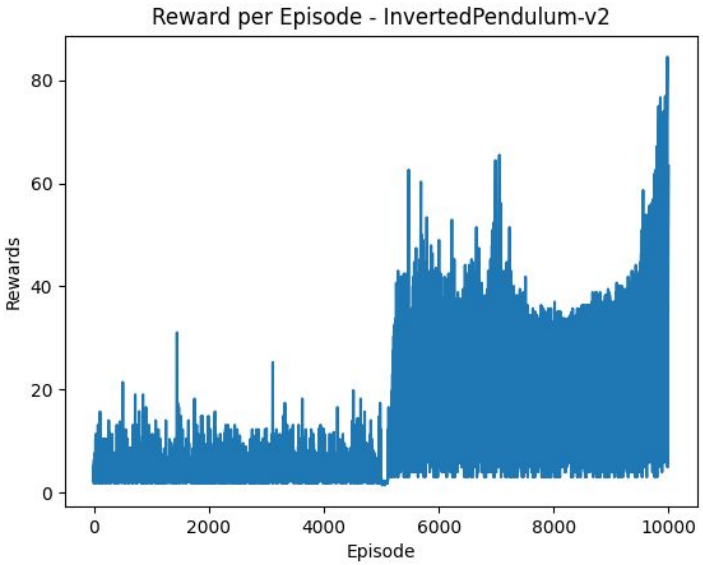


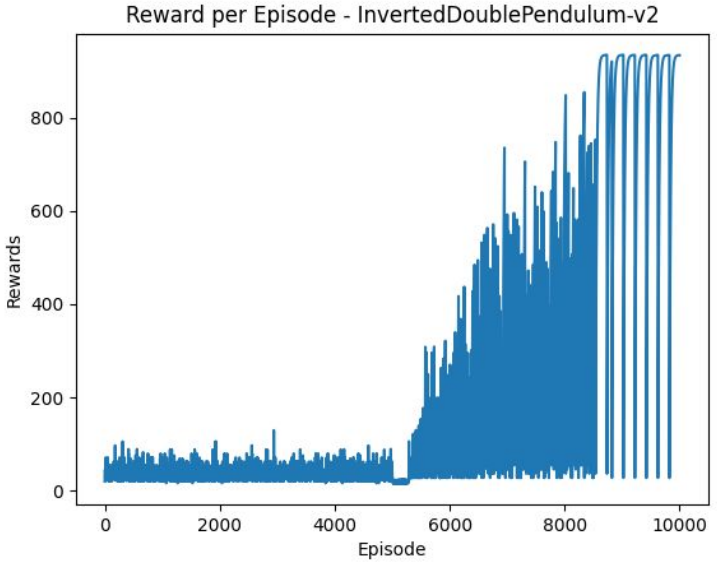
Walker2d-v2



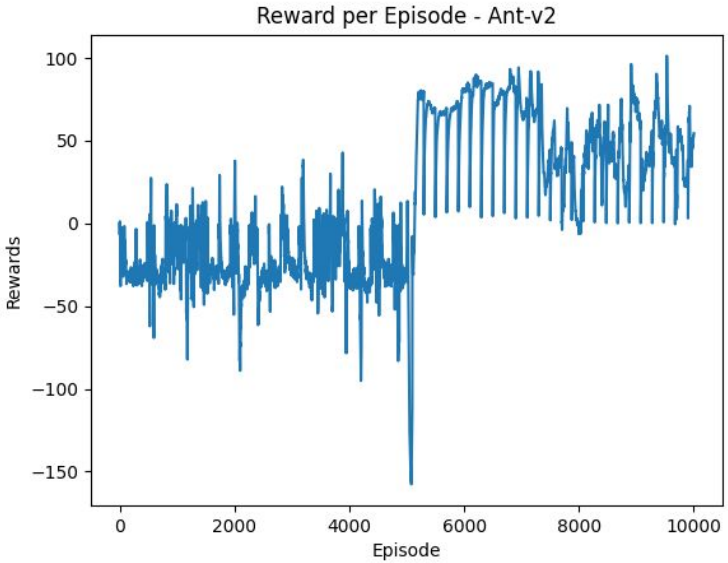
InvertedDoublePendulum-v2

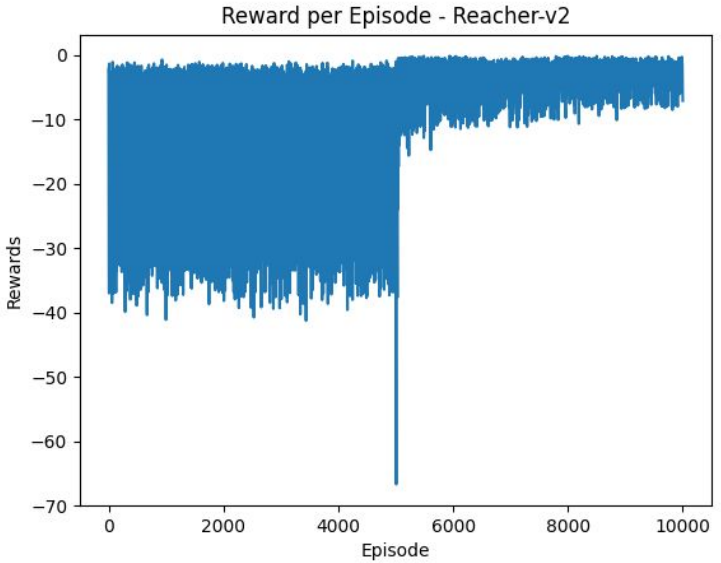
Results


Reward per Episode - InvertedPendulum-v2
Rewards
Episode
0
20
40
60
80
0
2000
4000
6000
8000
10000

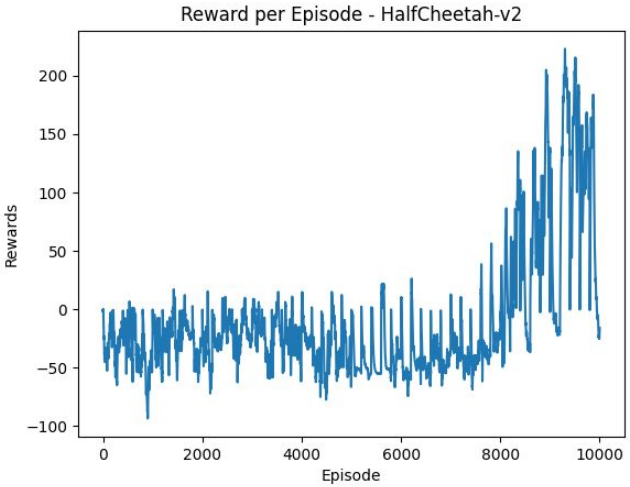

Reward per Episode - InvertedDoublePendulum-v2
Rewards
Episode
0
200
400
600
800
0
2000
4000
6000
8000
10000

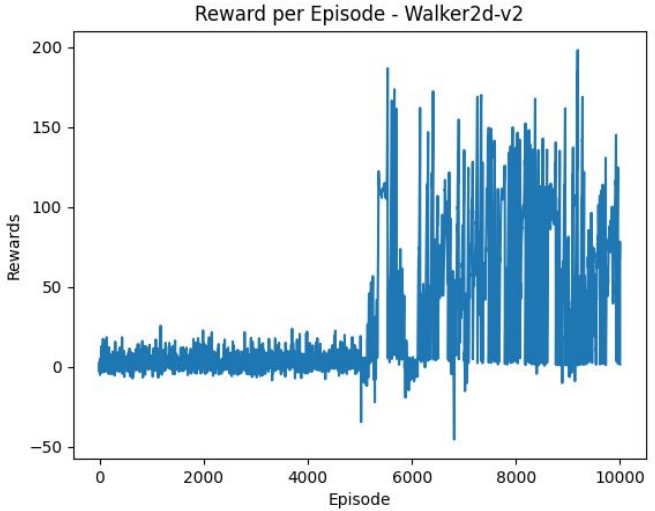
Results


Reward per Episode - Ant-v2
Rewards
Episode
100
50
0
−50
−100
−150
0
2000
4000
6000
8000
10000


Reward per Episode - Reacher-v2
Rewards
Episode
0
−10
−20
−30
−40
−50
−60
−70
0
2000
4000
6000
8000
10000

Results


Reward per Episode - HalfCheetah-v2
Rewards
Episode
200
150
100
50
0
−50
−100
0
2000
4000
6000
8000
10000


Reward per Episode - Walker2d-v2
Rewards
Episode
200
150
100
50
0
−50
0
2000
4000
6000
8000
10000

Error Analysis

- In many of the environments where our algorithm was applied, we observed sudden spikes in rewards. This could be attributed to the relatively low number of training steps used in our experiments and also because of the complex nature of the environments.

- Reward functions used could have been better for these environments

- Walk2d-v2 and Ant-v2 environments were not giving results upto mark because environment was pretty complex.

Environments Used

Experiments are conducted on a set of challenging environments that require a combination of locomotion and object manipulation.

Ant Gather A simulated ant must navigate to gather apples while avoiding bombs, which are randomly placed in the environment at the beginning of each episode. The ant receives a reward of 1 for each apple and a reward of −1 for each bomb.

Ant Push In this task we introduce a movable block which the agent can interact with. A greedy agent would move forward, unknowingly pushing the movable block until it blocks its path to the target. To successfully reach the target, the ant must first move to the left around the block and then push the block right, clearing the path towards the target location.

Environments Used (Cont.)

Ant Maze In this environment an ant must navigate to various locations in a ‘⊃’-shaped corridor. Increase the default size of the maze so that the corridor is of width 8. In evaluation, we assess the success rate of the policy when attempting to reach the end of the maze

Ant Fall This task extends the navigation to three dimensions. The ant is placed on a raised platform, with the target location directly in front of it but separated by a chasm which it cannot traverse by itself. Luckily, a movable block is provided on its right. To successfully reach the target, the ant must first walk to the right, push the block into the chasm, and then safely cross.

Results

	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

- Performance of the best policy obtained in 10M steps of training, averaged over 10 randomly seeded trials with standard error
- Comparisons are among variants of FuN SNN4HRL and VIME

Future Work

- We're going to experiment with combining techniques inspired by state-of-the-art papers in the field.
- Review more recent and SOTA approaches
- We can also try different approaches on the deep learning end.

Improvements

- There is a possibility to enhance off-policy correction by better sampling methods rather than random sampling.
- Another area of improvement is the utilization of neural networks, such as convolutional neural networks (CNNs) or attention mechanisms, within the algorithm.
- We can work on using better intrinsic reward functions in the algorithm.

Contributions

Yashwant Mahajan	Ayush Kumar	Somya Gupta
33.33%	33.33%	33.33%
Literature Review Worked on Baseline Model Researched on New Method to be used in current model	Literature Review Worked on Baseline Model Researched on New Method to be used in current model	Literature Review Worked on Baseline Model Researched on New Method to be used in current model

Citations

- Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In Advances in Neural Information Processing Systems, pages 5048–5058, 2017.
- Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In AAAI, pages 1726–1734, 2017.
- Gabriel Barth-Maron, Matthew W Hoffman, David Budden, Will Dabney, Dan Horgan, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. Distributed distributional deterministic policy gradients. arXiv preprint arXiv:1804.08617, 2018.
- Andrew G Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. Discrete Event Dynamic Systems, 13(4):341–379, 2003.
- Nuttapong Chentanez, Andrew G Barto, and Satinder P Singh. Intrinsically motivated reinforcement learning. In Advances in neural information processing systems, pages 1281–1288, 2005.
- Christian Daniel, Gerhard Neumann, and Jan Peters. Hierarchical relative entropy policy search. In Artificial Intelligence and Statistics, pages 273–281, 2012.
- Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In Advances in neural information processing systems, pages 271–278, 1993.

Thank You