CSE 598: Deep Learning Project Report

Rashmeet Kaur Nayyar (ASU ID: 1215139744)

1 Introduction

In this project, I built hierarchical autonomous agents inspired by [1] in PyTorch for two openAI gym environments with continuous state and action spaces: (a) Mountain Car, and (b) Inverted Pendulum that efficiently learn multiple levels of policies in parallel and accelerate reinforcement learning.

2 Problem

A long standing problem in Reinforcement Learning (RL) is its sample inefficiency for computing a policy for complex problems. Several approaches have tried to effectively compute a policy by hierarchically organizing prior knowledge. Many Hierarchical Reinforcement Learning (HRL) algorithms accelerate learning by decomposing the problems into sub-problems and learn policies at multiple levels of abstraction. The idea is that the high-level only needs to learn sequence of sub-tasks and a low-level needs to learn policies for each of these sub-tasks. Leveraging abstract high-level policies significantly boost the speed of computation of a low-level concrete policy. However, the biggest drawback is that most approaches do not learn the policies for sub-tasks in parallel and learning policies parallely is highly unstable.

For example, if a robot has to deliver coffees to all the customers at a restaurant, then the high-level policy containing sub-goals would tell the robot to first collect and then deliver each coffee to the right customer consecutively. This enables the low-level to compute a sub-policy for each sub-goal. However, computing these sub-policies in parallel using a stable algorithm can significantly speed up the computation of the final low-level policy. The challenge is to learn policies at higher levels even though the transition and reward functions at the low-level are changing. Overcoming this challenge will lead to jointly learning multiple levels of policies for faster decision-making of the original tasks.

3 Datasets

The problem does not require datasets explicitly as the agents learn from experience in a simulation. Each action of the agent is simulated in the simulator and the next state is received as an output. I chose to use two OpenAI gym robotic simulators [2] with continuous state and action spaces: Mountain Car and Inverted Pendulum. Each of these simulations were built using the Mujoco physics engine [3]. Both of the environments are described in detail below:

(a) **Mountain Car:**
The continuous Mountain Car environment can be found at [4]. Fig.1 shows the environment of continuous Mountain Car. A car is on a one-dimensional track, positioned between two "mountains" in the valley. The goal is to drive up the mountain on the right and reach the yellow flag; however, the car's engine is not strong enough to scale the mountain in a single pass. Therefore, the only way to succeed is to drive back and forth to build up momentum. Here, the reward is greater if you spend less energy to reach the goal.

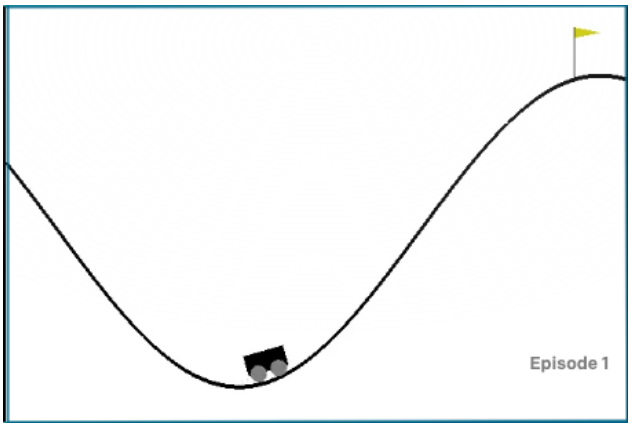

Figure 1: Continuous Mountain Car

The state space for this environment contains the car positions and the velocities. The position values span from -1.2 to 0.6 and the possible values for velocities lie between -0.07 to 0.07. A positive velocity means that the agent is accelerating to the right and a zero velocity means that the agent ceases to accelerate. The action space consists of the power coefficient that the car can use and the possible values lie between -1.0 to 1.0. The actual driving force is calculated by multiplying this power coefficient by power of 0.0015. A reward of 100 is awarded if the agent reaches the flag (at the position 0.45) on top of the mountain.

(b) **Inverted Pendulum:**
The continuous Inverted Pendulum environment can be found at [5]. Fig.2 shows the environment of continuous Inverted Pendulum. The inverted pendulum swingup problem is a classic problem in the control literature. In this version of the problem, the pendulum starts in a random position, and the goal is to swing it up so it stays upright. The pendulum is expected to rise to its maximum height as shown by the blue circle.

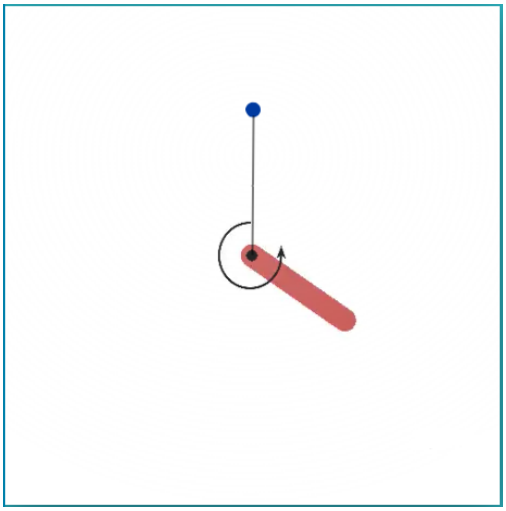

Figure 2: Continuous Inverted Pendulum

The state space of this environment consists of the angle θ, that lie between $-\pi$ and π, and the angular velocity $\dot{\theta}$, that lie between -8.0 and 8.0. The primitive action space consists of torque values between -2.0 and 2.0. The goal state is the state where the angle is 0 as well as the angular velocity is 0.

4 Methodology

The goal is to solve a Universal Markov Decision Process (UMDP) i.e. an MDP augmented with goals. A Hierarchical Actor Critic (HAC) algorithm solves a UMDP by breaking it down into a hierarchy of UMDPs and learning policy of each level in parallel.

4.1 Hierarchical Actor Critic (HAC)

Hierarchical Actor Critic is a Hierarchical Reinforcement Learning framework introduced by [6] [7] that trains each level of the hierarchy independently of the lower levels by assuming at each level that the lower level policies are already optimal. It is the first approach to

successfully learn 3-level hierarchies in parallel for tasks with continuous state and action spaces. Generally, the policy at any level depends on the current policies below that level. Training each level by simulating a transition function that assumes optimal lower level policies helps in overcoming the instability issues that arise due to non-stationary low-level transition functions. Learning is possible because the transition function for each level stabilizes when the lower level policies converge to optimal policies.
The two major components of HAC are: (a) Hierarchical architecture consisting of nested goal-conditioned policies, and (b) Methods for learning joint policies given sparse rewards.

4.2 Hierarchy of nested policies

The highest level in the hierarchy takes as input the current state and the original goal state for the task. It outputs a sub-goal state for the lower level to achieve. The lower level takes as input the current state and this sub-goal and proposes another sub-goal for the level below it to achieve. This process continues until the last level where the bottom-most level inputs the current state and sub-goal it obtains from the level above and outputs a primitive action. If a level runs out of a certain number of attempts or achieves its sub-goal, then execution at that level ends and the level above proposes another sub-goal.

4.3 Hindsight Action Transitions

Hindsight Action Transitions help the agent to learn policies at a level using a transition function that simulates the optimal lower level policy. This helps the agent to learn multiple policies parallely as the high level focuses on high-level decisions and the low levels focus on learning short sequences of actions to achieve each of those sub-tasks. When a level proposes a sub-goal state that the lower level does not achieve within the limits of a certain number of attempts, a hindsight action transition is created. The sub-goal state the agent instead ends up in in hindsight is used by the sub-goal level as the action component in the transition in-place of the originally proposed sub-goal.

4.4 Hindsight Goal Transitions

Hindsight Goal Transitions is a data augmentation technique that helps each level learn a goal-conditioned policy in tasks containing sparse rewards. It extends the idea of Hindsight Experience Replay (HER) to the hierarchical setting. After the agent executes a primitive action, the low level creates a transition using primitive actions and another copy of the same transition with missing reward and goal component. At the end of each episode, the original

goal state of each incomplete transition is replaced with a state that was actually achieved during the episode. This guarantees that at least one of HER transitions will contain sparse reward.

HER transitions with such sparse rewards help the Universal Value Function Approximator (UVFA) learn about regions of state, goal, and action spaces that should have high Q-values and extrapolates to learn other relevant areas of the space for achieving the goals.

4.5 Sub-goal Testing Transitions

Sub-goal Testing Transitions enables the agent to understand whether a sub-goal state can be achieved by the current low level and avoids ignoring Qvalues of infeasible sub-goals. It plays the opposite role of Hindsight Action Transitions. Proposing sub-goal actions that are distant and thus can not be achieved within the limit of horizon or number of attempts allowed by the lower level can cause the agent to act absurdly. Ideally, sub-goals feasible to achieve for the lower level should be proposed and this preference can be taken into account using sub-goal testing transitions. To overcome this issue, with a certain sub-goal testing rate, the lower level follows the policy exactly instead of exploring and if it does not achieve the sub-goal within a certain number of attempts, then a penalty is provided. A discount of 0 is used to avoid issues due to changing transition function.

5 Baseline and Evaluation Metric

As a baseline, I implemented Deep Deterministic Policy Gradient (DDPG) [8] with Hindsight Experience Replay [7] (1 level or flat). Owing to the time constraints, I could not implement the additional network HIRO: Hierarchical Reinforcement learning with Off-policy correction [9] (2 levels) for comparison with performance of my implementation. However, as suggested in my proposal, I was going to implement HIRO only if time permits. Hence, I evaluated my Hierarchical Actor Critic Agent (HAC) agents using Deep Deterministic Policy Gradient with Hindsight Experience Replay (DDPG-HER) as a baseline for comparison.

The results of my implementation of baseline using a flat policy are shown in the Fig.3 and Fig.4 (shown as 1 level in the legend) along with the results of my approach for both 2-level and 3-level hierarchies for easier comparison. I used success rate as the metric to evaluate the approach. Success rate is defined as the number of successful episodes upon the total number of episodes used for training. The training episodes are shown on the x-axis and success rate is shown on the y-axis of the plots.

6 Experiments and Results

I evaluated my implementation of the Hierarchical Actor Critic architecture on two continuous simulated robotic environments: Mountain Car, that had not been evaluated in the original paper, and Pendulum. I used the environments from Open AI Gym [2] developed using the MuJoCo physics engine [3]. In each task, I evaluated the performance of the agent using hierarchical policies of 2 levels and 3 levels as well as a flat policy of 1 layer. Agents using the flat policy of 1 layer used only DDPG and HER.

6.1 States, actions, rewards, and maximum horizon used

The state space for both the environments contain possible joint positions and joint velocities for the agent, whereas, the action space include joint torques. Thus, the dimensionality of the state space was 2 and dimensionality of action space was 1. All the tasks had sparse rewards. In particular, if an agent took an action suggested by the actor network and reached the sub-goal, then a reward of 0.0 was provided to the agent and if the action did not help the agent to reach the sub-goal, then a reward of -1.0 was provided. If the sub-goal networks proposed a sub-goal that the lower level could not reach, then a larger penalty of - maximum horizon was provided to the agent. I used a maximum horizon of 20 for both the tasks. This helps bound the number of attempts the agent takes for achieving any sub-goal before another sub-goal is proposed.

6.2 Goals, sub-goal threshold, and sub-goal testing rate used

For the Mountain Car task, I used the goal position as 0.46 and the goal velocity as 0.04. For the Inverted Pendulum task, I used the goal angle as 0 and goal angular velocity as 0. A sub-goal threshold parameter helped me decide whether the goal is achieved by checking if the agent is within a certain distance from the actual goal state. Using such a threshold is important for continuous domains. I used a sub-goal threshold of 0.01 position and 0.02 velocity for the Mountain Car task, and 10 degrees and 0.05 angular velocity as the threshold for Pendulum task. I also used a sub-goal testing rate of 0.3 which allowed the agent to decide whether a sub-goal should be tested to determine whether a sub-goal proposed by a certain level can be achieved by the lower level of the hierarchy.

6.3 Hardware Architecture and Training Configuration used

All of my experiments were executed on 5.0 GHz Intel i9 CPUs with 64 GB RAM running Ubuntu 18.04. I trained the HAC agent network for 1000 episodes and tested it for 10

episodes. The network was trained using 100 iterations and a batch size of 100. I also saved the model at intervals of 50 to avoid losing my work in case the experiment ended abruptly due to any reasons. I used a discount of 0.95 and a learning rate of 0.001. All of the hyper-parameters were tried and tested before finalizing the optimized ones, however, due to time constraints the set of parameter values tested was not exhaustive. The averages of results of 10 runs were taken to evaluate the stability of the learned architecture and obtain reliable results. I experimented with 2 level and 3 level hierarchical agents.

6.4 Results

Fig.3 shows the plot for average success rate of my implementation of Mountain Car agent with increasing number of training episodes. The lines show the success rates for baseline agent (flat policy or level 1) (shown in red) as well as the success rates for 2-level and 3-level hierarchical agents (shown in blue and green respectively). The agents began with no prior experience and learned from scratch. The agents explored random policies in the beginning before actually learning which is why they behave erratically during the first few episodes.

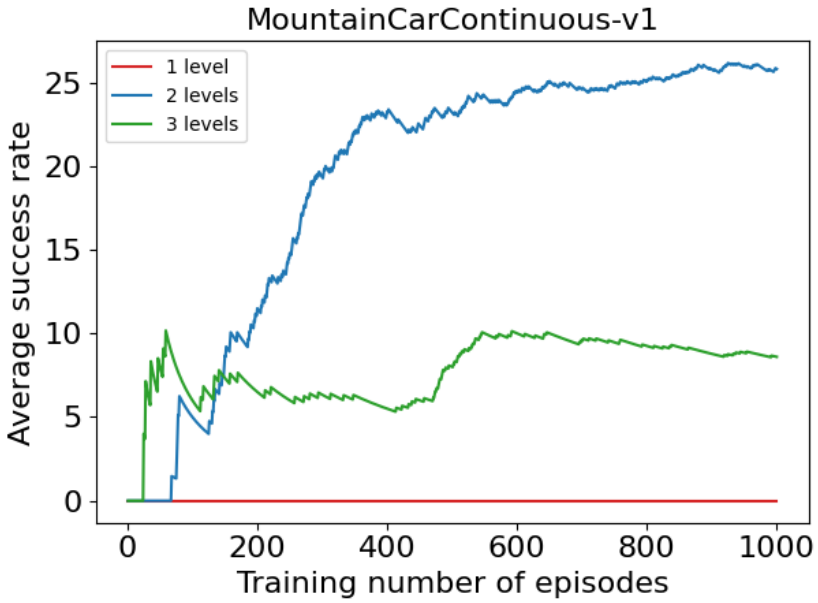

Figure 3: Average Success Rate vs number of training episodes for the baseline (1 level), 2 levels of hierarchy, and 3 levels of hierarchy in Mountain Car domain.

The results for both the 2-level as well as 3-level hierarchical agents in Fig.3 support the claim that multiple levels of policies in a hierarchy can indeed be learned simultaneously and that a Hierarchical Actor Critic (HAC) agent performs better than an agent that uses flat

policies. Both HAC agents learn robust policies significantly faster compared to the agent with no sub-goal layer using a flat policy, which consistently struggled to learn and could not solve the task at all even once during the training period of 1000 episodes.

Fig.4 shows the plot for average success rate for all three levels of agents for the Pendulum task with increasing number of training episodes. The 2-level agent (shown in blue) performs significantly faster compared to the 1-level baseline agent (shown in red). The success rate continues to increase speedily after 200 episodes of training and converges around 700 number of episodes. The 1-level agent falls far behind and its learning stagnates after approximately 400 episodes until 1000 episodes at only a 10% success rate. The 3-level agent (shown in green), on the other hand, did not perform as expected until 800 episodes of training after which it showed potential of learning at a fast rate until 1000 number of episodes. The 2-level agent reached a success rate of 50% within 600 episodes.

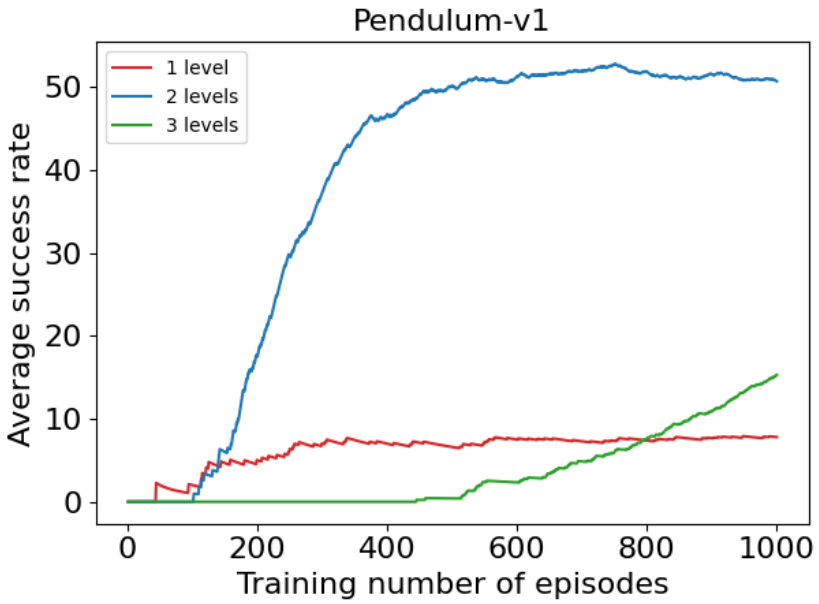

Figure 4: Average Success Rate vs number of training episodes for the baseline (1 level), 2 levels of hierarchy, and 3 levels of hierarchy in Pendulum domain.

The 2-level hierarchical agents significantly outperformed the 1-level agents as expected. However, it was unexpected that the 3-level agent did not perform better than the 2-level hierarchical agent for both the tasks as claimed in the original paper. I conjecture that there may be a bug in my implementation of the algorithm that prevented the agent with more number of levels in the hierarchy to learn faster. However, after immense effort I was not able to find any flaw in my implementation and could not make the 3-level hierarchical agent to perform better than the 2-level agent. It is definitely something I will continue to keep

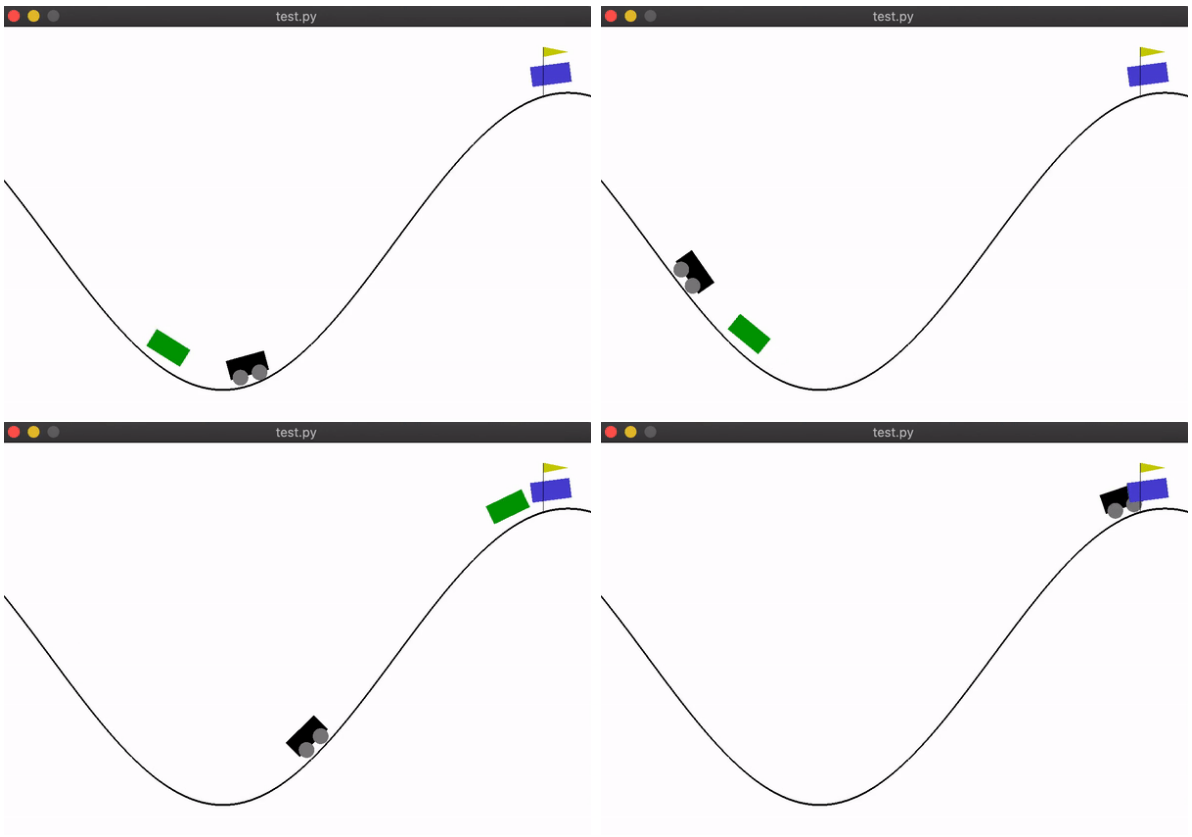

Figure 5: Intermediate shots of 2-level trained agent performing Mountain Car task

debugging and fix in case I find any flaw. It is clear however from all my results that the hierarchical agents, including the 3-level agent for the Mountain Car task, perform much better than the flat policy agents who do not make use of hierarchy at all.

Fig .5 shows the intermediate shots for the 2-level hierarchical agent solving Mountain Car task during testing. Fig .6 shows the intermediate shots for the 2-level hierarchical agent solving the Pendulum task during testing. For clear visualization, I also rendered the sub-goals (shown in green) and the end goal (shown in blue) for each task. In the Mountain car domain, for instance, the goal is shown by a blue rectangle and the green rectangle represents the sub-goals learned by the hierarchical agent. The car constantly attempts to achieve the proposed sub-goals instead of randomly exploring the state space. This helps the agent to solve the original problem and reach the top of the hill much faster compared to a flat agent. In the Inverted Pendulum task, proposed sub-goals are shown as green circles and the agent constantly tries to reach the sub-goals to eventually solve the original problem and reach the end goal of swinging itself to an inverted position (shown as blue circle).

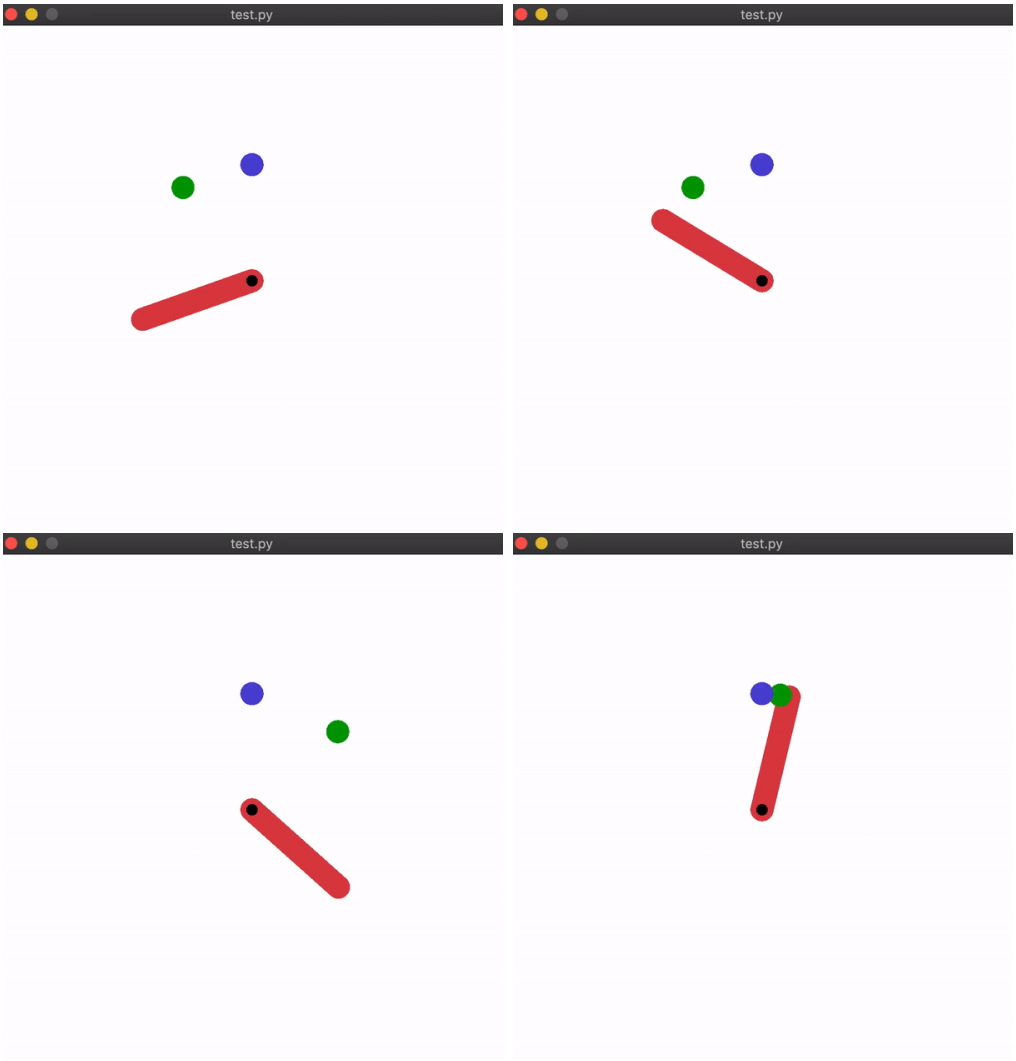

Figure 6: Intermediate shots of 2-level trained agent performing Mountain Car task

It can be clearly seen that the agent learns how to break the high-level problem into sub-problems. These sub-tasks are easier to solve than the original task and policies for each of these tasks are learned in parallel. This led to significant speedup in the learning of the reinforcement learning agents.

7 Conclusion

Hierarchical agents indeed have the potential to break down tasks into smaller problems that require decision making of smaller horizons and hence are easier to solve. These sub-tasks can be solved in parallel to achieve faster decision making. Hierarchical Actor Critic is the first step in training agents acting in continuous state action spaces in a stable manner.

My implementation of the architecture supports that the multiple levels of policies can be learned in parallel using hindsight action and goal transitions and significantly outperform flat RL agents. In the future, I will test agents with a larger hierarchy more rigorously and experiment with more complex continuous environments.

References

[1] A. Levy, G. Konidaris, R. Platt, and K. Saenko, "Learning multi-level hierarchies with hindsight," *arXiv preprint arXiv:1712.00948*, 2017.

[2] *Gym*, `https://gym.openai.com/envs/#classic_control`.

[3] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2012, pp. 5026–5033.

[4] *Moutain car*, `https://gym.openai.com/envs/MountainCarContinuous-v0/`.

[5] *Inverted pendulum*, `https://gym.openai.com/envs/InvertedPendulum-v2/`.

[6] A. Levy, R. Platt, and K. Saenko, "Hierarchical actor-critic," *arXiv preprint arXiv:1712.00948*, vol. 12, 2017.

[7] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight experience replay," *arXiv preprint arXiv:1707.01495*, 2017.

[8] 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," *arXiv preprint arXiv:1509.02971*, 2015.

[9] O. Nachum, S. Gu, H. Lee, and S. Levine, "Data-efficient hierarchical reinforcement learning," *arXiv preprint arXiv:1805.08296*, 2018.