COMPOSITIONAL LEARNING

Aryaman Singh Samyal
Department of Aerospace Engineering
The University of Texas at Austin
aryamansinghsamyal@utexas.edu

Venkata Sai Santosh Siripurapu
Department of Electrical and Computer Engineering
The University of Texas at Austin
sai.santosh@utexas.edu

1 Introduction

In this project, we explore the concept of compositional reinforcement learning. For our project, we will be using a bi-level optimization framework [1] to solve navigation and path-planning tasks, while ensuring that the agent achieves the goal with all the constraints and task specifications being met. The framework consists of a high-level controller that analyzes the collection of the low-level sub-systems, and devises interfaces between them. This allows the framework to split the task into multiple sub-tasks and assign the appropriate low-level subsystem for each sub-task. The collection of sub-task models for a particular environment acts as a library of trained policies for their respective sub-task. The high-level model plans a meta-policy to dictate the sequence of sub-systems to execute to attain the target state.

Our environment consists of a labyrinth environment constructed in the Unity Game Engine consisting of multiple rooms having either two or a single point for entry and exit. The goal for the agent would be to navigate from the initialization state till the target location. Each room would be modelled as a sub-task, and therefore the frameworks low-level library would consist of a multiple policies to navigate each of the rooms. For obtaining these policies, we are planning to train the agent for each room with both reinforcement learning and imitation learning and do a comparative study between both.

In our project we use this framework to check its performance and efficiency for two use-cases.

- Reinforcement Learning
- Imitation Learning

The video for the project can be accessed by this Link

2 Previous Work

[2] proposes a framework for verifiable and compositional reinforcement learning (CRL) in which a collection of RL sub-systems, each of which learns to accomplish a separate sub-task, are composed to achieve an overall task. It uses a parametric Markov decision process (pMDP [7, 8] to plan and analyze the composition of the subsystems and of the collection of sub-systems. The authors define interfaces between the sub-systems that enables automatic decomposition of task specifications into individual sub-task specifications. [3] review several approaches to temporal abstraction and hierarchical organization that machine learning researchers have recently developed. Intrinsically motivated agents can explore new behavior for their own sake rather than to directly solve external goals. Such intrinsic behaviors could eventually help the agent solve tasks posed by the environment.[4] presents hierarchical-DQN (h-DQN), a framework to integrate hierarchical action-value functions with goal-driven intrinsically motivated deep reinforcement learning. A top-level q-value function learns a policy over intrinsic goals, while a lower-level function learns a policy over atomic actions to satisfy the given goals. [5] presents a Hierarchical reinforcement learning (HRL) agent, HIRO, that is general, in the sense that it does not make onerous additional assumptions beyond standard RL algorithms, and efficient, in the sense that it can be used with modest numbers of interaction samples, like a few million. A novel HRL framework is proposed by [9], that attempts to tackle the inherent instability that arises by utilizing learning for multiple hierarchical levels of policy. It also reasons about the benefits linked with using system that decomposes a problem into sub-tasks and attempts to solve it explicitly, for example, decomposition and refinement of task specifications and selection of different algorithms for efficient solution of the sub-system and reduction of the state-action space [10]. [11] and [12]

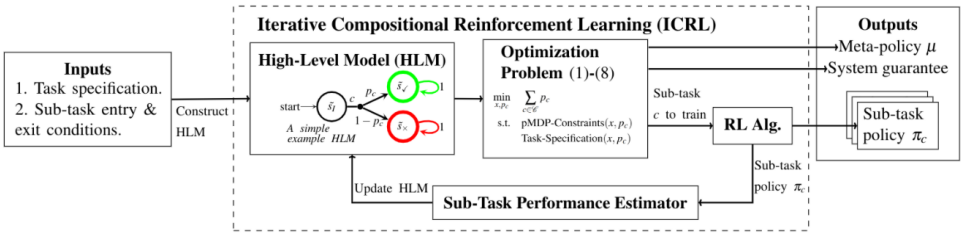

Figure 1: The compositional framework. Here 'c' indices point to the subtasks. 's' are the various subgoals.

formulate a Linear Temporal Logic-based approach to solve Markov Decision Processes and further use it to devise reward functions that results in RL algorithms that are more robust and are able to tackle complex problems. A similar method for symbolic learning by incorporating RL learners for an agent to reach a "sub-goal" when a plan isn't available was put forward by [13]. [14] proposes decomposition of RL tasks using reward machines, and explicit learning (using Q-Learning for Reward Machines – QRM) for separate policies for the various components of the task. [15] synthesizes a framework for simplifying cooperative multi-agent RL setting by splitting team-goals into individual sub-goals.

3 Current work by the authors

The framework consists of a high-level meta policy, represented as a parametric Markov decision process (pMDP) which is used to plan and analyze sub-systems. The problem is set up to find an optimal set of parameters in the pMDP to automatically decompose task specifications into sub-task specifications. In this way, the framework enables automatic decomposition of task specifications, e.g., reaching a target set of states with a probability of at least 0.95. This allows for the independent training and testing of the sub-systems; if they each learn a policy satisfying the appropriate sub-task specification, then their composition is guaranteed to satisfy the overall task specification. If some of the sub-task specifications cannot be satisfied by the corresponding learned policies, sampling-based estimates of their behavior are used to update the high-level model, and train subsystems to achieve the refined subtask specifications. The pMDP assigns probability values to the execution of different subsystems, given the current environment state. In other words, the collection of sub-systems defines the set of actions for the pMDP. From a given state, the pMDP is used to select a subsystem to execute. The sub-system's policy is then followed until it either reaches an exit condition. If the exit condition is reached, the metapolicy selects the next sub-system to execute, and the process repeats. If not, the execution of the meta-policy stops.

4 Motivation

In the original paper, the authors train a collection of low-level RL sub-systems using proximal policy optimization algorithms to navigate different portions/rooms of the labyrinth environment. We try to replace these RL algorithms with Imitation Learning (IL), Behavioral Cloning (BC), Generative Adversarial Imitation Learning (GAIL), Advantage Actor-Critic (A2C) and Soft-Actor Critic (SAC) methods and show a comparative study between them. In this section, we will discuss the motivation behind the use of these algorithms to communicate with the low-level environment.

4.1 Imitation Learning

In an Imitation Learning (IL) setting, the model learns by imitating a teacher. Therefore, the training is done by demonstrating the model about the correct actions. For example, in this case, the blue ball can be trained by showing how the trainer behaves while moving through a particular room in the labyrinth environment and the model can then be expected to do the same on its own later. Since the model tries to imitate the teacher's behavior, it is called IL. Unlike IL, a Reinforcement Learning (RL) model learns on its own by a combination of exploration and exploitation. The model trains on its own without the external help of a teacher by maximizing a scalar reward. Now that we've discussed the major differences between IL and RL, let's discuss some of the key benefits of using IL over RL.

- During an IL training episode, the trainer can use actions that are in the action space of the model i.e., the action space of the model and that of the trainer has complete or a significant overlap or the action space is really small, which is the case with this particular task. Whereas, while training an RL model, there is still

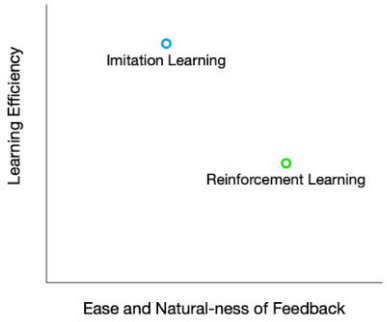

Figure 2: Imitation Learning vs Reinforcement Learning

a need to provide the rules of the reward mechanism, which in some scenarios (certainly this one) can be extremely challenging.

- The number of training episodes required with IL is fewer than that required with RL because of the 'dense' nature of feedback in Imitation Learning. RL, however, requires more training episodes, the reason being that the learning efficiency of the model is reduced due to the sparse nature of scalar rewards. In this case, the agent receives a reward only when it reaches the green box. Fig. 2 compares the learning efficiency and ease of feedback between the two methods.

4.1.1 Behavioral Cloning

The simplest form of imitation learning is behavioral cloning (BC), which focuses on learning the expert's policy using supervised learning. Given the teacher's demonstrations, we divide these lessons into state-action pairs and treat them as examples and apply supervised learning. Its main advantages are its low complexity and efficiency. Since in this case, each subtask is relatively simpler and independent and thus, doesn't require extreme long-term planning, the expert's trajectories can cover the state space. However, this approach may not yield very good performance since the learning process is only based on a set of samples provided by the expert and this is what we intend to examine.

4.1.2 Generative Adversarial Imitation Learning

Generative Adversarial Imitation Learning (GAIL) is a model-free imitation learning algorithm. GAIL can learn from a very small number of expert demonstration trajectories with the goal to train generators that have similar behaviors to the teachers. In addition, the discriminators serve as the reward functions which judge whether the generator's behavior looks like the teacher's. GAIL has obtained a substantial performance boost over existing methods in imitating complex behaviors in large, high-dimensional environments.

4.2 Soft-Actor Critic

Soft-Actor Critic (SAC) seeks to also maximize the lifetime rewards as well as the entropy of the policy. High entropy encourages exploration, ensuring that the policy does not collapse into repeatedly selecting a particular action, especially important in cases where the room is shuffled.

4.3 Advantage Actor-Critic

Advantage Actor-Critic or A2C is a off-policy method that utilizes advantage estimates to calculate the value proposition for each action state pair. It is the synchronous, deterministic version of A3C [16] which is more cost-effective and faster. It uses multiple workers to avoid the use of a replay buffer.

4.4 Twin Delayed DDPG

Twin Delayed DDPG or TD3 is anoff-policy algorithm that can only be utilized for continuous action-spaces (hence, is useful for us). It is sensitive and brittle to hyper-parameters and tuning because it learns two Q-functions and uses smaller of the two Q-values to form targets in the loss function. The policy is updated less frequently than the

Q-function. Also, TD3 adds noise to the target action in order to use Q-function error for smoothing out Q along with changes in action. This also promotes exploration.

5 Methodology

We use the Unity Game-Engine too devise a labyrinth environment. Our environment is a modified version of the labyrinth used in [1]. This is because when experimenting with the original environment, we observed that the low-level controllers found it difficult to take the shorter path and navigate through the "lava-rooms" (Subtask: $c_0 \rightarrow c_4 \rightarrow c_5 \rightarrow c_9$). Therefore, in hopes of alleviating this issue, we created the new environment given by Figure 3.

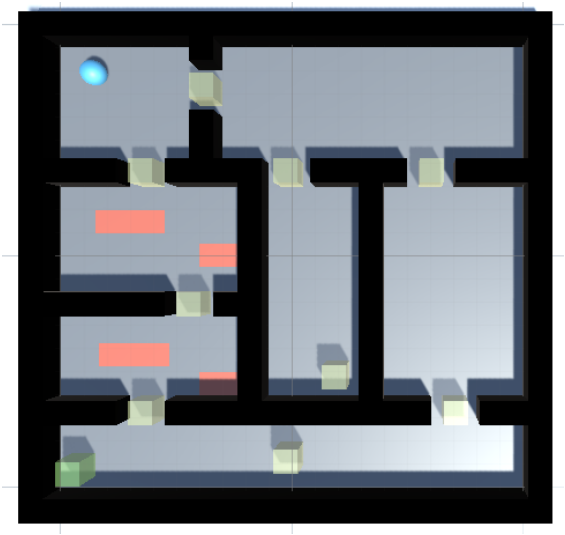

Figure 3: Unity Labyrinth

The framework was written in Python in the Windows Subsystem for Linux 2, as several dependencies such as Stable-Baselines3 were only available for the Linux platform. Furthermore, as the Unity Game-Engine is only stable on Windows, we could had to find a way to interface a linux-based distro and Windows, hence, the decision WSL2 over a native linux OS.

The operating environment and the agent was created using the Unity-MLAgents framework. Moreover, to interface Unity with WSL2 python scripts (Unity natively operates on C#), we build a custom communication channel using the `SideChannel` abstract class available in Unity. The data flow is Figure 4

We performed six sets of experiments on the labyrinth environment. The first three involved exploring the use of reinforcement learning based low-level controllers. For this we utilized PPO, SAC, A2C, TD3, four of the most popular algorithms for continuous domains. The last two involved using imitation learning to check its performance in the labyrinth and to see if they are viable methods to be utilized as the low-level controllers for the compositional framework.

RL-based low-level controllers: We used PPO, SAC, TD3 and A2C as low-level controllers and checked whether they converged to a solution on the labyrinth. For hyperparameter tuning, we took reference from the Stable-Baselines3 zoo repository which contains saved hyperparamters for various continuous domains.

PPO is an on-policy algorithm that is simple to implement. For PPO there are two primary variants:

1. PPO-Penalty: In this the KL divergence is penalized.
2. PPO-Clip: The objective function is clipped

For our experiments we used the PPO-Clip variant of the method.

We trained three off-policy methods on the labyrinth environment; TD3, A2C and SAC. TD3 can only be trained on continuous state-space and uses noise to balance exploration and exploitation. For our case we used basic gaussian noise. A2C is the synchronous version of the A3C [16] algorithm. It is quite similar to PPO, with the only difference

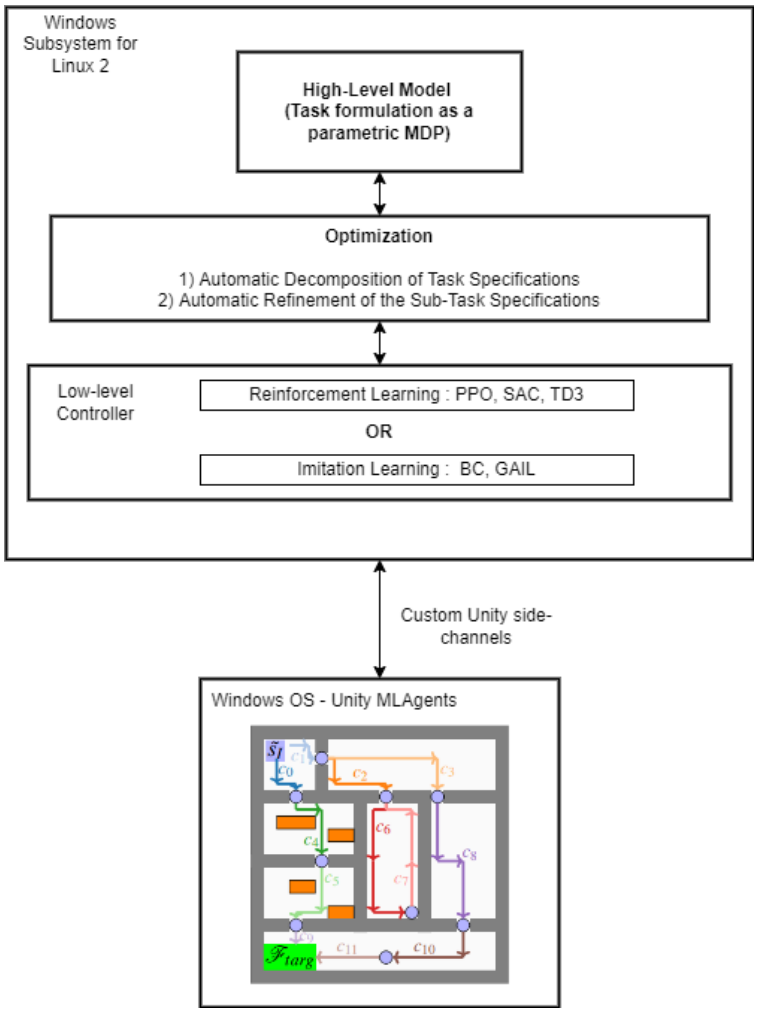

Figure 4: Unity G.E. and compositional framework integration

being the loss function. The calculation of the policy loss is where the main difference between the two methods lie. SAC is more sample-efficient than PPO and therefore takes significantly less time to convergence. Given the large state-space of our continuous environment, SAC proved to be exceptionally effective in finding the solution.

Imitation learning-based low-level controllers: Given that RL agents were finding it difficult to cross through the "lava-room", we decided to train and check the performance of GAIL and BC on our environment to see if they are viable to be used as a replacement for RL-based low-level controllers in the compositional framework.

For GAIL we used around 40 expert demonstrations that were recorded in Unity through the lava subtask c_4 and evaluated its performance. We also implemented GAIL on the non-lava subtask c_3, with around the same number of expert demonstrations. For Behavioral Cloning (BC) we used 50 demonstrations for both the lava route and the non-lava route (Subtask: $c_1 \rightarrow c_3 \rightarrow c_8 \rightarrow c_10 \rightarrow c_11$).

6 Results and Discussion

Graphical result by using PPO as the low-level controller is given by 6. PPO was fairly effective at finding the optimal solution. It converged in 700000 training steps and the solution involved going through the non-lava route.

SAC was even better than PPO at converging to a solution. This was expected as SAC is known to be a more sample-efficient method. It converged in about 550000 training steps. The resultant plots are given by Figure 7.

Figure 5: Subsystem indexing and corresponding color

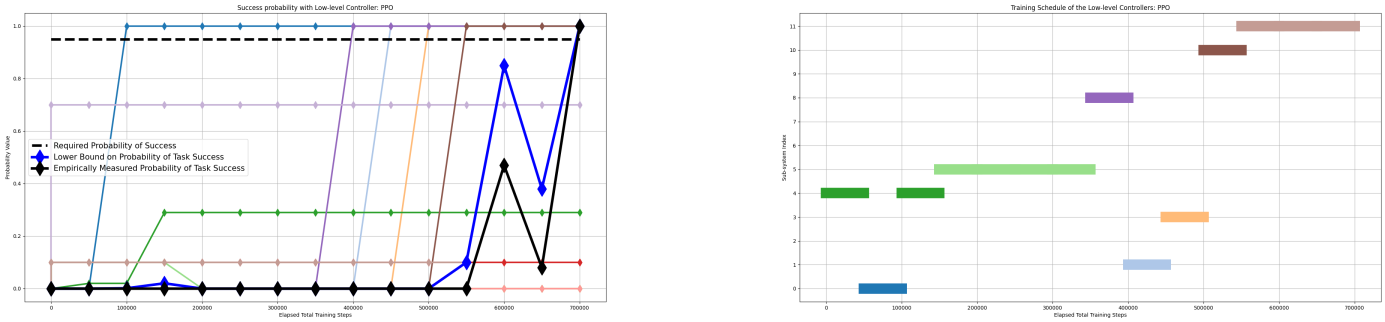

Figure 6: PPO training result. Each subtask is represented by a different color, matching those used in Figure 5

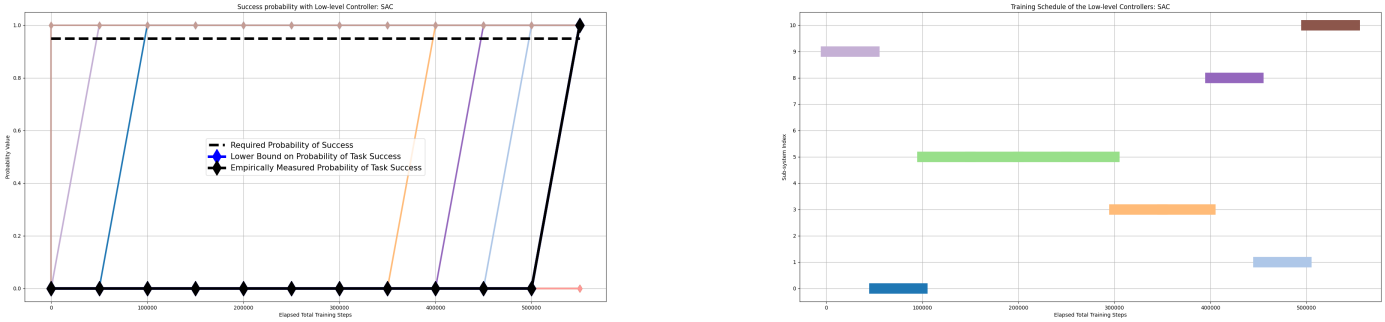

Figure 7: SAC training result

We also trained A2C and TD3, however we could not obtain satisfactory results with either. Given the high sensitivity of A2C to its hyperparameters which can lead to instability, we believe that A2C only requires more tuning to work. One advantage that we saw in A2C was that, it has a higher probability success rate in clearing the lava rooms than others. As for TD3, the agent seemed to have trouble exploring, so we believe that experimenting with different instantiation of noise might fix it as that would promote more exploration (For eg. using Ornstein Uhlenbeck action noise). The plots for A2C and TD3 are given by Figure 8 and Figure 9. As seen through through the graphs, the cummulative reward and episode length had stagnated without the controller converging onto a solution.

As for imitation learning, the behavioral cloning agent was able to navigate through the non-lava route with a low probabilty of success, the GAIL agent was unable to solve even a single room and seemed to be oscillating over the expert trajectories instead of finishing the subtask. Upon further investigation we found out that this was due to the **"Survivor/Reward bias"** issue prevalent in imitation learning method. This problem entails that because the reward is calculated intrinsically by the discriminator, the agent essentially tries to stay alive and imitate the expert in order to increase its cumulative reward. This can be offset by enabling a high extrinsic reward upon achieving a subtask. However to implement this we require more time as it would involve a complete overhaul of the framework.

As for the low success probability of the BC agent, the reasoning behind this issue is that, as BC directly tries to copy the agent, the number of expert samples need to be very high with lots of variation. This is because in case the agent finds itself in a state which was not recorded in the expert demonstrations, the episode will probably end in a failure as the agent would not know what to do in that particular state.

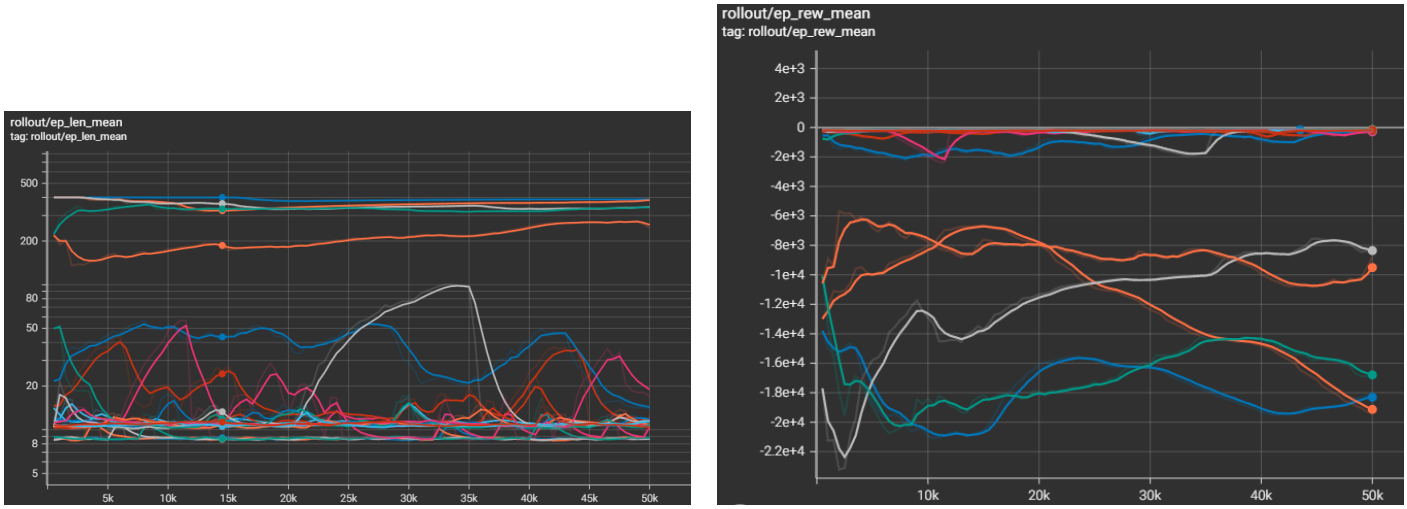

Figure 8: A2C plots. As we can see the cumulative reward of the individual controllers stagnated without any improvement in episode length and non-convergence to a solution

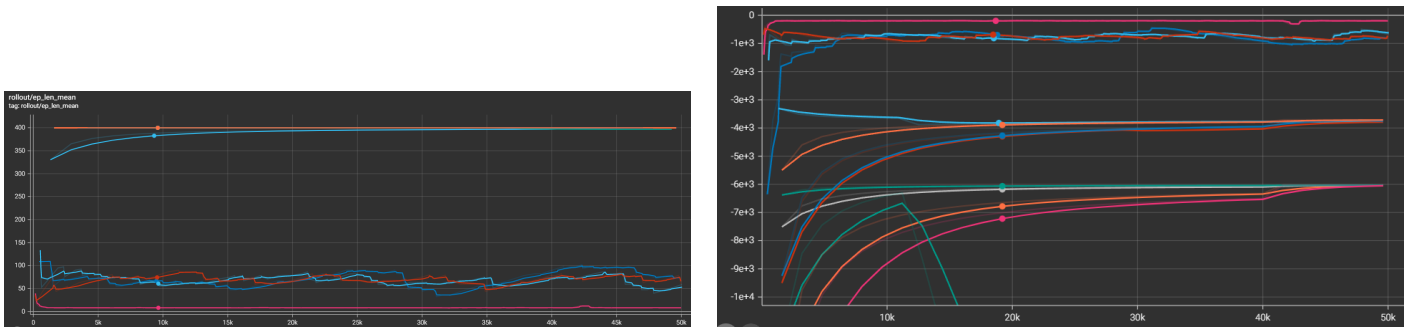

Figure 9: TD3 plots. As we can see the cumulative reward of the individual controllers stagnated without any improvement in episode length and non-convergence to a solution

7 Conclusion and Future Work

In this project we explore compositional reinforcement learning in a continuous navigation task. Compositional RL enables us decompose complex tasks in simpler subsystems which allows achieving verifiable task specifications. Our compositional framework consists of a High-level model (HLM) that formulates the problem as a parametric MDP (pMDP). Further, the HLM solves the pMDP with the parameters being the individual subsystems. The HLM then produces a meta-policy that join subsystems to achieve the over high-level objective with a verifiable success probability. We test this framework with various different reinforcement learning algorithm as well as imitation learning methods. In the future, we like to improve upon the RL-based low-level controller integration of the TD3 and A2C algorithm. Moreover, we would like to implement extrinsic reward signals so that imitation learning algorithm do encounter the "survivor bias" and are able to achieve the given subtasks.

References

[1] Neary, Cyrus, Christos Verginis, Murat Cubuktepe, and Ufuk Topcu. "Verifiable and Compositional Reinforcement Learning Systems." arXiv preprint arXiv:2106.05864 (2021).

[2] Andrew G Barto and Sridhar Mahadevan. “Recent advances in hierarchical reinforcement learning”. In: Discrete event dynamic systems 13.1 (2003), pp. 41–77.

[3] Tejas D Kulkarni et al. “Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and IntrinsicMotivation”. In: 30th Conference on Neural Information Processing Systems (NIPS 2016). 2016.

[4] O Nachum et al. “Data-Efficient Hierarchical Reinforcement Learning”. In: 32nd Conference on Neural Information Processing Systems (NeurIPS 2018). Curran Associates, Inc. 2019, pp. 3303–3313.

[5] Richard S Sutton, Doina Precup, and Satinder Singh. “Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning”. In: Artificial intelligence 112.1-2 (1999), pp. 181–211.

[6] Alexander Sasha Vezhnevets et al. “Feudal networks for hierarchical reinforcement learning”. In: International Conference on Machine Learning. PMLR. 2017, pp. 3540–3549.

[7] Murat Cubuktepe et al. “Synthesis in pMDPs: A Tale of 1001 Parameters”. In: International Symposium on Automated Technology for Verification and Analysis. Springer. 2018, pp. 160–176.

[8] Sebastian Junges. “Parameter Synthesis in Markov Models”. PhD thesis.

[9] Levy, A.; Konidaris, G. D.; Platt, R. W.; and Saenko, K. 2019. Learning Multi-Level Hierarchies with Hindsight. In International Conference on Learning Representations.

[10] ateria, S.; Subagdja, B.; Tan, A.-h.; and Quek, C. 2021. Hierarchical Reinforcement Learning: A Comprehensive Survey. ACM Computing Surveys (CSUR), 54(5): 1–35.

[11] Camacho, A.; Chen, O.; Sanner, S.; and McIlraith, S. A. 2017. Non-markovian rewards expressed in LTL: guiding search via reward shaping. 10th Annual Symposium on Combinatorial Search.

[12] Littman, M. L.; Topcu, U.; Fu, J.; Isbell, C.; Wen, M.; and MacGlashan, J. 2017. Environment-Independent Task Specifications via GLTL. arXiv:1704.04341.

[13] Sarathy, V.; Kasenberg, D.; Goel, S.; Sinapov, J.; and Scheutz, M. 2021. SPOTTER: Extending Symbolic Planning Operators through Targeted Reinforcement Learning. In Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS ’21, 1118–1126.

[14] Toro Icarte, R.; Klassen, T.; Valenzano, R.; and McIlraith, S.2018. Using reward machines for high-level task specification and decomposition in reinforcement learning. International Conference on Machine Learning, 2107–2116.

[15] Neary, C.; Xu, Z.; Wu, B.; and Topcu, U. 2021. Reward Machines for Cooperative Multi-Agent Reinforcement Learning. In Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS ’21, 934–942

[16] Mnih, Volodymyr, et al. "Asynchronous methods for deep reinforcement learning." International conference on machine learning. PMLR, 2016.