PLEX: PLanner and EXecutor for Embodied Learning in Navigation

Anonymous authors
Paper under double-blind review

Abstract

We present a method for policy learning to navigate indoor environments. We adopt a hierarchical policy approach, where two agents are trained to work in cohesion with one another to perform a complex navigation task. A Planner agent operates at a higher level and proposes sub-goals for an Executor agent. The Executor reports an embedding summary back to the Planner as additional side information at the end of its series of operations for the Planner's next sub-goal proposal. The end goal is generated by the environment and exposed to the Planner which then decides which set of sub-goals to propose to the Executor. We show that this Planner-Executor setup drastically increases the sample efficiency of our method over traditional single agent approaches, effectively mitigating the difficulty accompanying long series of actions with a sparse reward signal. On the challenging Habitat environment (Savva et al., 2019) which requires navigating various realistic indoor environments, we demonstrate that our approach offers a significant improvement over prior work for navigation.

1 Introduction

The ability to model and understand the world at a high-level is crucial for performing complex tasks in real world environments. Part of this high-level understanding involves the ability to divide and plan out tasks that are complicated and have long time horizons into more manageable subtasks. For example, when navigating to a new location, we typically break the task down into a set of manageable directions (*i.e.* drive along a certain road until a familiar landmark before taking a turn). Imbuing machines with this ability of creating abstractions for long and complex tasks is an active area of research known as hierarchical learning (Sutton et al., 1998; 1999).

Research for navigation has recently seen a rejuvenation due to the advent of learning-based approaches (Gupta et al., 2017; Parisotto & Salakhutdinov, 2017; Zhang et al., 2017; Henriques & Vedaldi, 2018). Embodied learning-based approaches have shown some appealing properties over classical approaches such as being able to operate in complex environments with limited sensor data (Savva et al., 2019; Mishkin et al., 2019). However, there is a need for the ability to plan across long time horizons with sparse reward signals. This in effect, causes limitations such as the inability to overcome small obstacles when navigating towards a given goal and the requirement of invoking the environment a large number of times for any meaningful learning to occur (Le et al., 2018). Works which have combined hierarchical reinforcement learning with imitation learning have shown promising results (Das et al., 2018b; Le et al., 2018), by leveraging expert trajectories with policy sketches (Andreas et al., 2017), which are less expensive to obtain; however these sketches still require annotation of the environment.

In this work, we study such hierarchical control for the task of indoor navigation, whereby an embodied agent is randomly spawned within a novel and complex environment and must learn to navigate this environment through interaction (Das et al., 2018a). We address this challenging learning problem through a hierarchical policy approach, where two agents are cooperatively trained together. Each agent performs a different role, where one agent acts as a Planner, learning how to propose good sub-goals to an Executor agent, which acts at the low level to achieve these sub-goals (Fig. 1). In contrast to existing hierarchical policy learning approaches, communication between our two agents is two-way, where the Executor provides the Planner with a summary of its series of actions and recent observations. This aids the Planner in deciding the next sub-goal with additional side

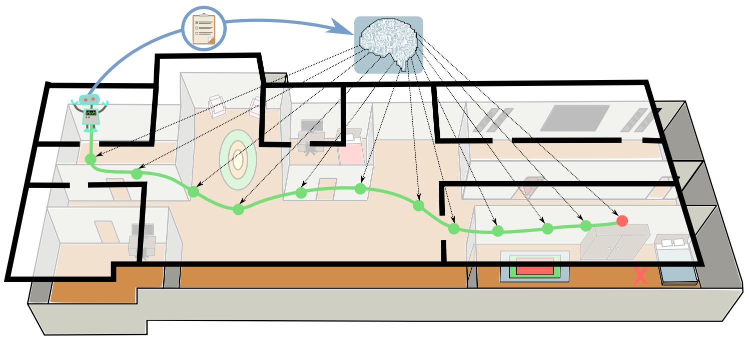


Figure 1: Our PLEX framework adopts a hierarchical policy approach, where a Planner proposes sub-goals for an Executor to act upon within an environment. The Planner receives an egocentric, top-down view with the target location and an embedding summary provided by the Executor. The Executor receives visual sensory data (*i.e.* colour and depth) as its input and a sub-goal provided by the Planner. Our method reduces the need for long-term planning and addresses the known sample inefficiency problem accompanying memory models within deep reinforcement learning approaches.

information provided by the Executor. To this end, we propose PLEX, a planning and executing learning framework which offers the following contributions:

- A hierarchical reinforcement learning approach where two agents specialise on different tasks but are jointly trained by sharing information
- We demonstrate both theoretically and empirically that our method benefits from significantly improved sample efficiency as the time horizon is distributed between the Planner and Executor
- By extension, our approach mitigates problems prevalent in long-horizon planning, especially those adopting LSTM (Hochreiter & Schmidhuber, 1997) planning approaches

2 Related Work

Hierarchical Reinforcement Learning The application of hierarchical reinforcement learning (Sutton et al., 1998; 1999; Bakker et al., 2004) in real world settings has allowed deep reinforcement learning approaches to scale with the increasingly higher sample requirements that are required by complex real world environments (Das et al., 2018b;a). Early works considered an options-based approach, where there was an underlying assumption of the existence of some useful set of options which are fully defined beforehand; this allowed learning to occur at a higher level in terms of those set of options (Sutton et al., 1998; 1999). In Kulkarni et al. (2016), a hierarchical approach was adopted without the use of imitation learning. However, the work was limited to exploring non-realistic environments and information flow was uni-directional from master to controller. Andreas et al. (2017) proposed sub-goals which have semantic meaning; although this demanded a high-level of supervision where rich annotations on a given environment were required. The work of Le et al. (2018) showed that using a hierarchical learning approach can lead to a reduction in the cost of exploration on problems with sparse rewards under the limited context of game environments. Das et al. (2018b) explored the embodied question answering task (Das et al., 2018a), leveraging heuristics embedded in the environment for providing expert trajectories used in the imitation learning initialisation stage. This method is heavily supervised and assumes that full semantic information of the environment is known. Additionally, the approach of Das et al. (2018b) is limited to a specific set of sub-goals, for example: Exit-room, Find-object, *etc.*, imposing limitations on the expressiveness of the master policy. By contrast, our method allows the Planner to propose sub-goals which directly relate to the given environment (*i.e.* a continuous point-goal vector) and does not rely on external annotations for supervision.

Embodied Agent Learning There has been a recent surge of interest towards embodied agent learning (Gupta et al., 2017) where an agent is personified and spawned within an environment and set to complete a certain task. Such tasks may include question answering (Das et al., 2018a), point goal navigation (Anderson et al., 2018), roaming (Fang et al., 2019) and scene exploration or coverage (Chen et al., 2019; Fang et al., 2019). This problem is purposely set up in a way which allows for relatively easy transfer from simulation environment to a physical robotic platform operating in a real world environment. Various simulation environments have been designed around this (Savva et al., 2019; Chang et al., 2017; Xia et al., 2018; Brodeur et al., 2017; Kolve et al., 2017), which aim to provide a realistic indoor simulation for training embodied agents on the aforementioned tasks. These environments allow an agent to invoke the environment numerous times, and in effect, scale the sample hungry nature of the problem to an extent where agents perform at a reasonable level (Savva et al., 2019; Das et al., 2018a). Savva et al. (2019) trained an agent using the Proximal Policy Optimisation algorithm (Schulman et al., 2017), augmenting it with a memory unit (Chung et al., 2014). Although this provided a strong initial baseline, achieving this required invoking an environment for millions of steps. The recent work of Fang et al. (2019) attempted to address this by proposing a memory component which makes use of the transformer network (Vaswani et al., 2017). Although this approach mitigated a key weakness in LSTM memory units (Hochreiter & Schmidhuber, 1997), it incurs with it a linear growth of memory usage which is addressed via memory compression and hence increases the computational complexity of the model. The aforementioned issues are not present in our framework since the Executor's rollout is strictly shorter as we will soon demonstrate.

3 METHOD

3.1 EXPLORING EFFICIENTLY

Previously, we have discussed an inefficiency in non-hierarchical reinforcement learning frameworks; this inefficiency stems from the nature of the problem setup. Given an agent that is normally initialised without any priors and explores an environment randomly, intuitively we can see that if the distance between the goal state and an agent's starting state is large, reaching the defined goal will require several exploration rollouts. In other words, the probability of an agent reaching its goal decreases with the distance of this goal and conversely increases as the distance to the goal shrinks. More formally, we can define the relationship between the rollout length of the agent and the exploration probability *w.r.t* to that rollout length:

Proposition 1. *The rollout evolution of a randomly initialised agent is simplified to be a sum of Bernoulli random variables $X_i \in [-1, 1]$ with the initial probability $p = \frac{1}{2}$. The sum of these i.i.d variables is $S_N = \sum_{i=1}^{N} X_i$. As we increase the rollout length N, the agent's probability of exploring decreases exponentially and is given by the following bound:*

$$\mathcal{P}(S_N \geq \alpha N) \leq e^{-N\beta} \tag{1}$$

where $\alpha > \frac{1}{2}$. For proof of Proposition 1, please refer to Appendix A.2.

Intuitively and as shown theoretically, reducing the rollout length N has a positive impact on the required exploration (for a more general form of this proposition, we refer the interested reader to Gardes & Prat (2000)). One way to exploit this insight is through a hierarchical approach which allows setting sub-goals that are closer in state space to the agent's current state.

3.2 PLANNING AND EXECUTING

Given this insight, we now provide details of our hierarchical approach which consists of two main components: a Planner and an Executor policy. Each of these components is treated as an independent agent with respective policies $\pi_{\theta_{PL}}$ and $\pi_{\theta_{EX}}$. A high-level overview of our framework is shown in Fig. 2. The environment is denoted as $\mathcal{P}$, from which states s are sampled. A reward function $\mathcal{R}$ provides the reward conditioned on a state s, an action a and a goal. In general, a goal denoted by g indicates the end goal for the current episode, with g_P denoting the sub-goals given by the Planner. N_{EX} and N_{PL} denote the respective rollout lengths of the Executor and Planner.

Executor Building upon previous RL frameworks, an Executor is optimised using standard policy learning techniques (Schulman et al., 2017). A key difference between existing non-hierarchical

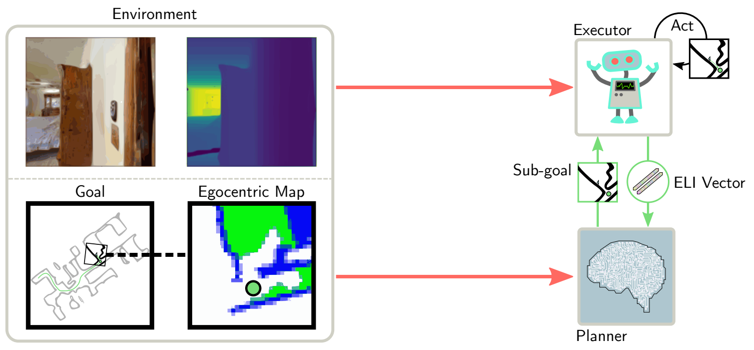


Figure 2: A system overview of PLEX which consists of two main components: a Planner and Executor agent. The Executor's task is to perform a series of actions such that it will traverse the environment towards a target location given by a point-goal vector (a sub-goal that is provided by the Planner). The Planner's task is to generate sub-goals that help navigate the entire system to an end goal (which only the Planner can see). An environment's state is comprised of an RGB-D observation, an egocentric map and a point-goal vector measurement pointing towards a target location as defined by Anderson et al. (2018). The Executor is provided with the RGB-D observation, a sub-goal generated by the Planner and returns an Executor Latent Information (ELI) vector that summarises its rollout. The Planner sees an egocentric map with a roughly $4m$ view horizon and the ELI vector. The Planner and Executor agents are trained simultaneously and using the sub-goal by the Planner and ELI vector by the Executor, we create a two-way communication channel, enabling our framework to regulate the flow of information between our two agents.

approaches and our approach is the Executor End State or sub-goal g_P, which is not provided by the environment but rather by a Planner policy. Consequently, the rewards provided to the Executor are sampled using the sub-goal (*i.e.*: $r_i \sim \mathcal{R}(r_i|s_{i-1}, a_i, g_P)$). In line with this, the Executor policy also provides an Executor Latent Information (ELI) vector which summarises its rollout experience back to the Planner policy to provide feedback for the Planner towards planning the next sub-goal. In effect, this creates a cooperative feedback loop between the Executor and Planner. A detailed outline of the Executor routine is provided by Algorithm 1.

Planner The Planner does not invoke the environment directly, but it modifies the goal and rewards observed by the Executor when the Executor interacts with the environment. This modification is achieved through the generation of planned sub-goals given to the Executor by the Planner. As such, given a state s and an ELI vector provided by the Executor, the Planner produces a sub-goal. Using this sub-goal, the Executor routine (Algorithm 1) is invoked and a reward is returned to the Planner which provides it with feedback on the generated sub-goal. Upon optimising the Planner, we note that the Executor should not be directly affected from the Planner optimisation process (*i.e.* we terminate backpropagated gradients from the Planner policy before they reach the Executor policy). In other words, the ELI vector provided by the Executor does not affect the Executor directly, but implicitly through improving the Planner policy's generated sub-goals since providing direct feedback to the Executor through the ELI vector will result in a redundant Planner. Algorithm 2 fully outlines our approach and details the Planner routine.

4 Experiments

We focus on the indoor PointGoal navigation task as defined in Anderson et al. (2018). In this setup, a point-goal vector providing the distance and angle towards the target point in the environment is assumed. If the indoor environment would be an empty space, this task would be trivial. However, in a more realistic, real world setting, the agent is spawned in a room within the environment and the target may be in a different room. In such a case, the agent needs to effectively navigate around obstacles and plan its future actions in the presence of other rooms that may or may not lead to the target point.

4.1 Sensory inputs and model outputs

The environment provides RGB-D sensory information and inaccurate access to the location and orientation (as could be obtained using a fusion of GPS and IMU sensors (Caron et al., 2006)). From these sensors, the environment can provide a coarse egocentric top-down map; we construct our egocentric map by adapting a similar approach in Chen et al. (2019)[1] (Section A.1 details the method of Chen et al. (2019) for constructing the egocentric map). The top-down map has a resolution of $0.5m$ per pixel which is a reasonable assumption given the provided sensors. In our hierarchical problem formulation, the Planner is provided with the egocentric, top-down map of size 32×32 pixels, the point-goal vector (translated to ego-map point-goal) $(distance, direction)$ and the Executor Latent Information (ELI) provided by the Executor (a vector of 128 values). The Planner emits a binary category distribution for *continue* or *stop* operations and two continuous sub-goal distributions: the distance $\mathcal{N}(\mu_\rho, \sigma_\rho)$ and direction $\mathcal{N}(\mu_\theta, \sigma_\theta)$. The Executor is provided with RGB-D sensory information of 256×256 resolution along with the sub-goal computed by the Planner and emits a four category distribution with the actions described in Section 4.3, and emits the ELI vector which is returned to the Planner when the Executor stops or performs the maximum number of actions, N_{EX} (note that for the Executor, the stop action does not stop the entire episode but simply returns control back to the Planner).

4.2 Model architecture and parameter values

Our Planner has a perception model for extracting embeddings from the egocentric map. This perception model is a CNN composed of 2 convolutional layers with filter size 4×4 and 3×3 with a stride of 2 and 1 respectively with ReLU (Nair & Hinton, 2010) activation. The output of these convolution layers is flattened and concatenated with the ELI vector and transferred to the last fully-connected linear layer which outputs a 512 vector with ReLU activation. This output is concatenated with the 2-valued point-goal vector.

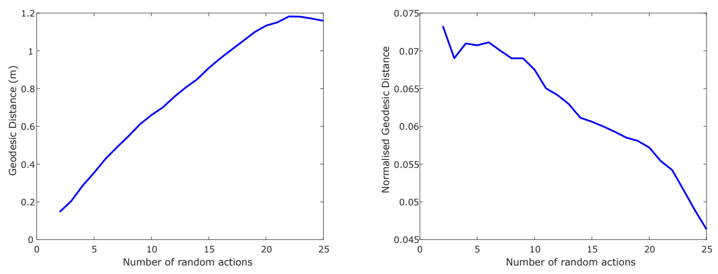
The Executor's perception model consists of 3 convolution layers with filter sizes of 8×8, 4×4 and 3×3 with strides of 4, 2 and 1 respectively and ReLU activations. This is followed by a fully-connected linear layer which outputs an embedding of 512 or 128, which depends on the model variant (LSTM (Hochreiter & Schmidhuber, 1997) or SMT (Fang et al., 2019) respectively). We concatenate the 2-valued sub-goal vector to the output embedding. For the LSTM memory model, we employ the GRU formulation (Chung et al., 2014) with hidden state of size 512. For the SMT memory model, we use an 8 multi-head attention mechanism (we do not employ the furthest point sampling technique used in (Fang et al., 2019) as it did not affect the performance of our model). The output of either the GRU component or the SMT is the action embedding (a vector of 128 values corresponding to the output state vector of the LSTM) and also functions as the ELI vector.

Both the Planner and the Executor policies are trained using the Proximal Policy Optimisation algorithm from Schulman et al. (2017), following the original settings found in the paper except for parameters specific to Algorithms 1 and 2: $N_{EX} = 10$, $\gamma_{EX} = 0.9$, $N_{PL} = 128$, $\gamma_{PL} = 0.95$. Note that the values of N_{EX} and γ_{EX} are chosen such that the Executor will focus on short-term rewards; this in turn will implicitly encourage the Planner to generate subsequent sub-goals which are nearby (further discussed in Section 4.4).

4.3 Environment Setup

Habitat (Savva et al., 2019) is a modular environment API and can be used with realistic indoor environments (Xia et al., 2018; Chang et al., 2017) coupled with tasks such as PointGoal navigation (Anderson et al., 2018) and Embodied Question Answering (Das et al., 2018a). For our experiments, we use the Gibson dataset (Xia et al., 2018) and perform the PointGoal navigation task (Anderson et al., 2018). We use the same train-test split as provided in Savva et al. (2019) which consists of 72 train scenes with $4.9M$ PointGoal navigation tasks and 16 unseen test scenes comprised of $1k$ PointGoal navigation tasks. The environment accepts 4 actions: [Forward, Turn left, Turn right, Stop], where invoking the stop action ends an episode with an episode running to a maximum of 500 steps otherwise. The reward structure is similar to Savva et al. (2019) with a success providing a reward of 10; otherwise, the agent's reward is the change in geodesic distance

[1]Code adapted from: `https://github.com/taochenshh/exp4nav`



(a) Geodesic Distance (b) Normalised Geodesic Distance

Figure 3: **(a)** The geodesic distance travelled by a random exploration policy with increasing number of random actions. **(b)** The geodesic distance normalised by the number of random steps taken. Both figures are averaged from 500 scenes.

between the agent and the target with an additional slack of $\lambda = -0.01$. In addition, we penalise for obstacle collision with a cost $c = -0.01$. This additional collision penalty is a desirable property for embodied agents (Chen et al., 2019).

4.4 Executor Rollout Horizon and Exploration Efficiency

In Section 3, we discussed the negative impact a long rollout horizon has on the exploration efficiency. The context was in a setup which allowed two actions, and did not take into account the complexities of realistic environments (such as obstacles and more actions). Hence, to choose an appropriate rollout length for the Executor (given the chosen environment and sensory input), we perform a short experiment which analyses a random exploration policy within the proposed setup. We randomly choose 500 starting locations from the training scenes, with a different initial starting point for recurring scenes. The agent can take 3 actions: [Forward, Turn Left, Turn Right], where the "Stop" action is excluded so that the rollout length can be inspected without being shortened.

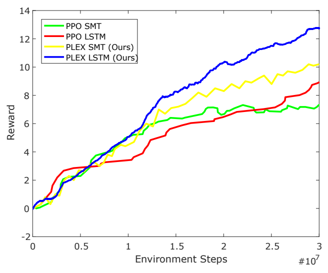
As we increase the number of random actions the random policy takes ($N = [2, 25]$), we inspect two values: the geodesic distance between the starting position and the end position of the policy (Fig. 3a), and the geodesic distance normalised with the number of random actions taken by the policy (Fig. 3b). The geodesic distance normalised by policy actions indicates the average distance travelled per number of actions and is an indication of exploration efficiency. As such, for initial exploration, we would generally want this value to be high.

Fig. 3b observes that as the rollout horizon of the random policy increases, the exploration efficiency begins to rapidly decline. Whilst this decline is not exponential as suggested by Proposition 1, it provides us with further motivation not to set N_{EX} too high. Further, we need to consider a large enough N_{EX} to enable the Executor explore its environment. Given Figs. 3a and 3b and given a egocentric map with a resolution of $0.5m$ per pixel, $N_{EX} = 10$ appears to maintain a good balance between the two criteria we wish to impose on the Executor: a high probability of exploring more than a unit distance in the egocentric map, along with high exploration efficiency.

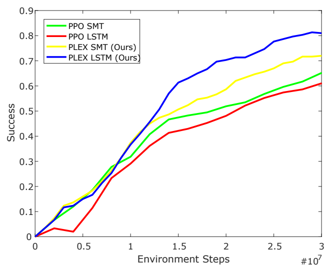
4.5 Experimental Results

Baselines We compare our approach to two baseline methods PPO LSTM and PPO SMT. The former refers to the method found in Savva et al. (2019), and the latter is a recent approach found in Fang et al. (2019). For comparison, we employ two variants of our PLEX framework: PLEX LSTM and PLEX SMT. This comparison allows a fair comparison regarding the benefits toward improved sample efficiency that our framework offers for both compared baselines.

Evaluation Metrics For evaluating the performance on the PointGoal task, we examine the reward each method obtains and the success ratio of each method. A success for an episode is defined when the agent provides a stop action and the geodesic shortest path distance between the agent and its end goal is less than $0.2m$. We do not use the Shortest Path Length (SPL) as a evaluation metric,



(a) Reward Plots



(b) Success Plots

Figure 4: **(a)** The observed reward obtained by each model over $30M$ environment steps. **(b)** The observed success rate of each model. In this case, a success is defined as the agent issuing a stop action when the geodesic shortest path distance between the agent and its goal is less than $0.2m$. Note that across both reward and success rate curves, the baseline methods exhibit a slow down in learning at around $15M$ environment steps whilst our method remains relatively robust and is able to maintain a steady learning rate.

as defined by Anderson et al. (2018) since maximising on this metric as an objective can result in agents which exhibit undesirable behaviours (such as approaching too closely and colliding with obstacles). Instead, we observe the reward which is a unitless measure and penalises undesirable behaviours such as obstacle collision and moving in the wrong direction. This metric, along with success rates, provides good indicators for assessing performance on this task (Fang et al., 2019).

Quantitative Results In Figs. 4a and 4b, we show the respective reward and success ratios of each method. Similarly, in Table 1, we outline the mean and variance of the final rewards and success ratios for all the methods after $30M$ environment steps. Each experiment was repeated 10 times across all methods. Note that the LSTM variant of our method (PLEX LSTM) achieves a success rate of 0.81 which is only slightly higher than the PPO LSTM result reported in Savva et al. (2019). The key difference is the significant lower number of environment steps (approximately half) that our method required for achieving a similar result.

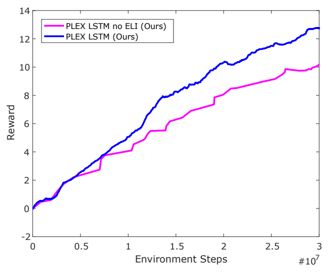
4.6 The Effect of Executor Latent Information

The key idea behind the ELI vector is to provide additional information to the Planner that may not be observed in the egocentric map. Hence, the ELI vector is a mechanism that can be used by the Executor to relay a compressed representation of its previous observations to the Planner; assisting it towards generating sub-goals. Since the ELI vector is an integration of previous observations outputted by the Executor's LSTM component, we refer to it as a summary of the Executor's rollout.

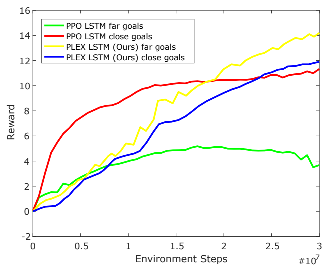
We adapt a hierarchical framework by Kulkarni et al. (2016), where conveniently, we can modify the Q-learning approach to a policy-based approach. In this context, our Executor replaces the

Performance Metrics				
	PPO SMT (Fang et al., 2019)	PPO LSTM (Savva et al., 2019)	PLEX SMT (Ours)	PLEX LSTM (Ours)
Avg. Reward	7.17±0.54	8.85±0.72	10.20±0.61	**12.76±0.58**
Avg. Success	0.65±0.03	0.61±0.02	0.72±0.02	**0.81±0.01**

Table 1: Avg. Reward and Avg. Success across the baseline models and our method applied to each method. Note that whilst the performance difference between the two baselines is small, both of our models outperform the baselines by a significant margin. Specifically, our PLEX LSTM variant shows a significant improvement over both baselines, achieving a higher result than the baseline PPO LSTM model (Savva et al., 2019) trained on more than double the number of environments steps.



(a) Effect of ELI vector



(b) Close vs. Far Goals

Figure 5: **(a)** The impact towards obtained rewards with and without the ELI vector being provided to the Planner on the LSTM variant of our PLEX framework. **(b)** The performance of our method compared with PPO LSTM (Savva et al., 2019) on close and far subsets of the test set. In this case, close goals were selected when mean geodesic shortest path distance was less than $5.76m$ and were classified as far goals otherwise.

"Controller" and our Planner serves as the "Meta Controller". This allows us to assess the effect of not providing the Planner with an ELI vector. Empirically, these results are shown in Fig. 5a. We can see that when the ELI vector is not provided, the communication channel from the Executor to the Planner is removed and we observe a decline in performance. This observation aligns well with our framework and demonstrates the value of enabling communication from the Executor back to the Planner.

4.7 Learning Across Long Time Horizons

Across both reward and success ratio curves, the two baseline methods gradually exhibit a slow down behaviour starting at around $15M$ timesteps. In contrast, our PLEX framework appears relatively robust, demonstrating a more consistent learning pace. This highlights a key difference between our hierarchical method compared to the baseline approaches. Both baselines and our method are able to successfully learn simpler tasks, where the goal is relatively close to the agent's spawn location. However, for tasks where the goal is further away, the need for a more structured approach is necessary. Our empirical results illustrate a potential drawback of LSTM-like units (and by extension, GRU), where they may suffer when learning long time horizon tasks.

We divide the test set into two subsets containing close and far goals using a dividing criteria which uses the mean geodesic shortest path between the agent's spawn location and its end goal. In this case, close goals were selected when mean distance was less than $5.76m$ and were classified as far goals otherwise. The results for this experiment are shown in Fig. 5b where the PPO LSTM is unable to navigate to the far goals in the allocated number of environment steps. In contrast, our method is far more successful in this task and is able to achieve a higher score on reaching far goals due to the extra reward provided for navigating to further away targets. Although the PPO LSTM baseline does eventually learn to navigate to further targets, it requires a much larger number of training iterations and environment invocations.

5 Conclusion

In this work, we present a hierarchical reinforcement learning approach for solving PointGoal navigation tasks. Our proposed approach uses a cooperative learning strategy in which two agents, an Executor and a Planner are jointly learned to solve this task. This is enabled through a two-way communication channel established between the two agents through the use of an Executor Latent Information vector provided by the Executor and sub-goals generated by the Planner. We motivate the use of this hierarchical approach both theoretically, as well as through empirical experiments which demonstrate a significant improvement in sampling efficiency of our approach, allowing our structured approach to perform significantly better on increasingly harder tasks when compared to baseline approaches.

REFERENCES

Peter Anderson, Angel Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, et al. On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*, 2018.

Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 166–175. JMLR. org, 2017.

Bram Bakker, Jürgen Schmidhuber, et al. Hierarchical reinforcement learning based on subgoal discovery and subpolicy specialization. In *Proc. of the 8-th Conf. on Intelligent Autonomous Systems*, pp. 438–445, 2004.

Simon Brodeur, Ethan Perez, Ankesh Anand, Florian Golemo, Luca Celotti, Florian Strub, Jean Rouat, Hugo Larochelle, and Aaron Courville. Home: A household multimodal environment. *arXiv preprint arXiv:1711.11017*, 2017.

Francois Caron, Emmanuel Duflos, Denis Pomorski, and Philippe Vanheeghe. Gps/imu data fusion using multisensor kalman filtering: introduction of contextual aspects. *Information fusion*, 7(2): 221–230, 2006.

Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niebner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. In *2017 International Conference on 3D Vision (3DV)*, pp. 667–676. IEEE, 2017.

Tao Chen, Saurabh Gupta, and Abhinav Gupta. Learning exploration policies for navigation. *arXiv preprint arXiv:1903.01959*, 2019.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Abhishek Das, Samyak Datta, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. Embodied question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 2054–2063, 2018a.

Abhishek Das, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. Neural modular control for embodied question answering. In *Conference on Robot Learning*, pp. 53–62, 2018b.

Kuan Fang, Alexander Toshev, Li Fei-Fei, and Silvio Savarese. Scene memory transformer for embodied agents in long-horizon tasks. *arXiv preprint arXiv:1903.03878*, 2019.

François Gardes and Georges Prat. *Price Expectations in Goods and Financial Markets*. Edward Elgar Publishing, 2000.

Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2616–2625, 2017.

Joao F Henriques and Andrea Vedaldi. Mapnet: An allocentric spatial memory for mapping environments. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8476–8484, 2018.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Eric Kolve, Roozbeh Mottaghi, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474*, 2017.

Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*, pp. 3675–3683, 2016.

Hoang M Le, Nan Jiang, Alekh Agarwal, Miroslav Dudík, Yisong Yue, and Hal Daumé III. Hierarchical imitation and reinforcement learning. *arXiv preprint arXiv:1803.00590*, 2018.

Dmytro Mishkin, Alexey Dosovitskiy, and Vladlen Koltun. Benchmarking classic and learned navigation in complex 3d environments. *arXiv preprint arXiv:1901.10915*, 2019.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

Emilio Parisotto and Ruslan Salakhutdinov. Neural map: Structured memory for deep reinforcement learning. *arXiv preprint arXiv:1702.08360*, 2017.

Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. Habitat: A platform for embodied ai research. *arXiv preprint arXiv:1904.01201*, 2019.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Richard S Sutton, Doina Precup, and Satinder P Singh. Intra-option learning about temporally abstract actions. In *ICML*, volume 98, pp. 556–564, 1998.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Fei Xia, Amir R. Zamir, Zhi-Yang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: real-world perception for embodied agents. In *Computer Vision and Pattern Recognition (CVPR), 2018 IEEE Conference on*. IEEE, 2018.

Jingwei Zhang, Lei Tai, Joschka Boedecker, Wolfram Burgard, and Ming Liu. Neural slam: Learning to explore with external memory. *arXiv preprint arXiv:1706.09520*, 2017.

A Appendix

A.1 Egocentric Top-Down Map construction

Given an embodied agent trajectory of length N with corresponding depth images $\{D_n\}_{n=1}^{N} \in \mathbb{R}^{H\times W}$ and relative camera poses $\{T_n\}_{n=1}^{N} \in \mathbb{SE}(3)$, the set of $3D$ points $P_n \in \mathbb{R}^{HW\times 3}$ for frame n are extracted as follow:

$$P_n[i,j,k] = T_n D[i,j] K^{-1} \left[i,j,1\right]^{\top} \tag{2}$$

where the camera intrinsic matrix is $K \in \mathbb{R}^{3\times 3}$. We then define the global set of all points at the end of the trajectory as: $\mathcal{M} \in \mathbb{R}^{HWN\times 3}$. The set $\mathcal{M}$ starts empty and is iteratively updated by the following rule:

$$\mathcal{M}_n = \mathcal{M}_{n-1} \cup P_n \tag{3}$$

As the set of all points $\mathcal{M}$ is built, the egocentric map can be derived by projecting all points in $\mathcal{M}$ to a plane (we lose the "height" dimension and points projected to the same slot are resolved by taking the highest point), cropping and aligning a box using the vector of the agent's facing direction which results in an egocentric map.

A.2 Proof of Proposition 1

Proof. The moment generating function of S_N is given by:

$$\mathcal{M}_{S_N}(\theta) = [\mathcal{M}_X(\theta)]^N = [\sum_{x=0}^{\infty} \mathcal{P}(X=x)e^{\theta x}]^N = [\frac{1+e^{\theta}}{2}]^N \tag{4}$$

The probability of the Bernoulli sum being greater then αN with $\alpha > \frac{1}{2}$ can be bounded using the Chernoff bound:

$$\mathcal{P}(S_N \geq \alpha N) \leq \sup_{\theta>0}\{e^{-\theta\alpha N}[\frac{1+e^{\theta}}{2}]\} \tag{5}$$

Analytically solving this bound, we obtain $\theta^* = \log\frac{\alpha}{1-\alpha}$. We are specifically interested in the case where $\alpha > \frac{1}{2}$, and as a result $\theta^* > 0$. Substituting θ^* into Eq. 5 and defining $\beta \equiv \log\frac{1}{1-\alpha} + \alpha\log\frac{\alpha}{1-\alpha}$ concludes the proof. $\square$

A.3 Planner and Executor Algorithms

Algorithm 1 A pseudo algorithm of Executor Policy Gradient; Actor-Critic variant

```
$N_{EX} \leftarrow$ initialise Executor rollout length
$\gamma_{EX} \leftarrow$ initialise Executor discount factor
$g_P \leftarrow$ initialise Planner goal
$r_P = 0$ initialise Planner reward
$buffers[s_b, a_b, r_b] \leftarrow$ initialise state, action, reward buffers
for $i = 1$ to $N_{EX}$ do
    $a_i, e_{\theta_{EX}} \sim \pi_{\theta_{EX}}(a_i, e_{\theta_{EX}}|s_{i-1}, g_P)$ Sample policy for action and latent information
    $s_i, t_i \sim \mathcal{P}(s_i, t_i|s_{i-1}, a_i)$ Sample environment for state and terminal
    $r_i \sim \mathcal{R}(r_i|s_{i-1}, a_i, g_P)$ Sample reward given Planner goal
    $accumulate\ r_P \leftarrow r \sim \mathcal{R}(r|s_{i-1}, a_i, g)$ Sample reward given end goal
    $update\ buffers \leftarrow s_{i-1}, a_i, r_i$
end for
$\pi_{\theta_{EX}} \leftarrow update(\pi_{\theta_{EX}}, buffers, \gamma_{EX})$ (as in PPO (Schulman et al., 2017))
return $(s_i, r_P, t_i, e_{\theta_{EX}})$
```

Algorithm 2 A pseudo algorithm of our PLEX framework

```
$\pi_{\theta_{PL}}, \pi_{\theta_{EX}} \leftarrow$ initialise Planner and Executor parameterised by $\theta_{PL}, \theta_{EX}$ respectively
$max\_iters \leftarrow$ initialise number of training iterations
$N_{PL} \leftarrow$ initialise Planner rollout length
$\gamma_{PL} \leftarrow$ initialise Planner discount factor
$s_{i-1} \sim \mathcal{P}(s_{i-1})$ Reset environment get initial state
$e^{-}_{\theta_{EX}} \leftarrow$ initialise Executor Latent Information (ELI) vector with zeros
**for** $iter = 1$ to $max\_iters$ **do**
    $buffers[s_b, a_b, r_b, t_b, e_{\theta b}] \leftarrow$ initialise state, action, reward, terminal and ELI vector buffers
    **for** $i = 1$ to $N_{PL}$ **do**
        $g_i \sim \pi_{\theta_{PL}}(g_i | s_{i-1}, e^{-}_{\theta_{EX}})$ Sample Planner policy for sub-goal
        $(s_i, r_i, t_i, e^{+}_{\theta_{EX}}) \leftarrow \boldsymbol{Algorithm 1}(g_i)$
        $update\ buffers \leftarrow (s_{i-1}, g_i, r_i, t_i, e^{-}_{\theta_{EX}})$
        $e^{-}_{\theta_{EX}} \leftarrow e^{+}_{\theta_{EX}}$
    **end for**
    $\pi_{\theta_{PL}} \leftarrow update(\pi_{\theta_{PL}}, buffers, \gamma_{PL})$ (as in PPO (Schulman et al., 2017))
**end for**
**return** $(\pi_{\theta_{PL}}, \pi_{\theta_{EX}})$
```