The Generalist vs. Expert Dilemma in Reinforcement Learning

Amirhossein Sohrabbeig[1] **Kiana Aghakasiri**[2] **Amirhossein Rajabpour**[2] **Mohammad Tavakoli**[2]

Abstract

Multi-task learning has become a pivotal strategy in reinforcement learning (RL) research, aiming to equip agents with the capability to generalize their knowledge and skills across diverse tasks. However, the delicate balance between training a generalist agent capable of handling diverse tasks and focusing on a specialist agent dedicated to a single task poses a fundamental challenge, commonly known as the "Generalist vs. Expert Dilemma" in reinforcement learning. While multi-task learning offers advantages in terms of leveraging shared patterns across tasks and enhancing robustness, single-task learning remains vital in scenarios where specialization is crucial. In this project, we want to explore this dilemma and aim to address the challenge of determining when to employ single-task learning and when to employ multi-task learning. Our research investigates the impact of task similarity, the number of environments, total budgets, and various algorithms. Code can be accessed here.

1. Introduction

Reinforcement Learning (RL) holds promise for numerous applications such as self-driving cars, algorithm management, and robotics. However, for RL to truly be effective in real-world settings, it needs to be resilient to ever-changing environments and should possess the ability to adjust and familiarize itself with untried yet analogous settings during its operation.

Therefore, the focus in reinforcement learning is gradually shifting towards ensuring generalization, where an RL agent can perform well in its training environment and unfamiliar or slightly altered environments. Generalization in RL is about building models that can adapt and transfer their knowledge to new, unseen scenarios. This is crucial for real-world applications where the agent encounters numerous situations not present during training.

One promising approach to achieve this desired generalization is through multi-task learning (MTL) (Vithayathil Varghese & Mahmoud, 2020). MTL is a paradigm where a single model is trained on multiple related tasks simultaneously. The underlying principle is that by exposing the model to diverse tasks, it can leverage shared patterns and structures among them, leading to a more robust and generalized understanding of the environment.

In the context of reinforcement learning, multi-task learning works by training an agent on several tasks or environments concurrently. The agent learns to optimize multiple reward functions, possibly from different domains, at the same time. This process forces the agent to identify and extract commonalities across tasks, allowing it to generalize its knowledge and be more resilient to changes in its environment. For example, an RL agent trained for both walking and running might develop a more comprehensive understanding of locomotion compared to one trained solely for walking.

The concept of generalization across different tasks can also be explained using human's everyday life. To learn how to cook, one might benefit from learning about heat management, timing, chemical reactions of different ingredients, and nutritional science. Similarly, aspiring to become a proficient computer scientist isn't just about understanding algorithms and coding; it also involves grasping mathematical concepts, logical reasoning, and even effective communication skills, all of which play crucial roles in tackling complex computing problems and innovating in the field.

However, due to the limitations involved, such as limited time to interact with the environment and the possibility of unrelated tasks negatively affecting learning other tasks, also known as the negative knowledge transfer (Vithayathil Varghese & Mahmoud, 2020), multi-task learning is not always the best option.

[*]Equal contribution [1]Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB T6G 2W3, Canada [2]Department of Computing Science, University of Alberta, Edmonton, T6G 2E8, Canada. Correspondence to: Amirhossein Sohrabbeig <sohrabbe@ualberta.ca>, Kiana Aghakasiri <kaghakas@ualberta.ca>, Amirhossein Rajabpour <arajabpo@ualberta.ca>, Mohammad Tavakoli <tavakol5@ualberta.ca>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

In this project, we aim to investigate the delicate balance between training a generalist agent on multiple tasks versus focusing on a specialist agent dedicated to a single task. A complete elucidation of this balance is beyond the scope of this course; thus, we will base our project on certain assumptions and simplifications, leaving a broader resolution for subsequent research.

2. Background

The Generalist vs. Expert Dilemma in reinforcement learning refers to a fundamental challenge in the development of intelligent systems. In reinforcement learning, an agent makes sequential decisions and interacts with the environment to maximize a reward signal. In this context, one of the dilemmas is whether our agent should be an expert and master at one specific task or should be a generalist and have good knowledge of various tasks. Does being trained on various tasks help the model to perform better for the specific main task or not?
Single-task and Multi-task agents, each have pros and cons. The training time, diversity, and amount of data required increase for a single-task agent. Furthermore, its performance continues to improve even at the frontier of data, compute, and model scale (Kaplan et al., 2020; Hoffmann et al., 2022; Reed et al., 2022). Also, it is empirically observed that an agent trained on many variations (a generalist) tends to learn faster at the beginning, yet its performance plateaus at a less optimal level for a long time (Jia et al., 2022). In contrast, an agent trained only on a few variations (a specialist) can often achieve high returns under a limited computational budget (Jia et al., 2022).

2.1. Problem Formulation

In the following, we will define the notations used in this project.
A general Markov Decision Process (MDP) is defined as a tuple $M = (S, A, P, R, \gamma)$, where:

- S and A are the state space and action space, respectively.
- $P(s' \mid s, a) \in [0, 1]$ is the state transition probability. The probabilities given by p completely characterize the environment's dynamics.
- $R(s, a)$ is the reward function.
- $\gamma \in [0, 1)$ is the discount factor. It discounts future rewards to give more weight to immediate rewards.

The transition probability function $p(s', r \mid s, a)$ is defined as:

$$p(s', r \mid s, a) = \mathbb{P}\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\} \tag{1}$$

for all $s' \in \mathcal{S}$, $s \in \mathcal{S}$, $r \in \mathcal{R}$, and $a \in \mathcal{A}(s)$. The function p defines the dynamics of the Markov Decision Process (MDP) (Sutton & Barto, 2018).

In reinforcement learning, we aim to train a policy $\pi(a \mid s)$ that maximizes the total amount of reward. It is based on the assumption that "goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward)"(Sutton & Barto, 2018). The formula for cumulative reward (return) in reinforcement learning is defined as:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \tag{2}$$

2.2. Related Work

IMPALA (Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures). The most related work to this project is by Espeholt et al. (2018), in which they came up with IMPALA, Importance Weighted Actor-Learner Architecture. In this paper, they used this distributed agent to learn different tasks asynchronously. In the single-task version of this agent, they used actors on the same task and then updated the agent for the multi-task version in which they assigned a fixed number of actors for each task. Furthermore, it combines the algorithm with a novel off-policy correction method, which enables the agent to overcome the problem of policy-lag. Another huge merit of this algorithm is that it is more robust to hyperparameter values and the architecture of the network compared to similar approaches, making it easier to benefit from the deeper neural networks. As depicted in Figure 1, on the left, we have the single-agent version of IMPALA which uses all the actors for training the target task which is not our goal in this work. On the right, the multi-agent version of it is shown, which assigns a specific number of actors for each task, and policy parameters are distributed between these different learners.

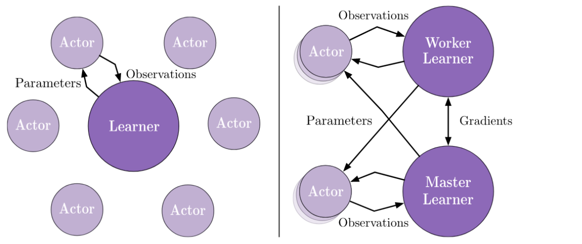


Figure 1. Left: Single Learner. Right: Multiple Learners (Espeholt et al., 2018)

We are going to use IMPALA for our multi-task agent as it has high GPU efficiency and is capable of scaling up while maintaining training stability. We describe this agent in more detail in our experimental design section.

Proximal Policy Optimization (PPO). PPO is another related work to our project (Schulman et al., 2017) which is a reinforcement learning algorithm developed by OpenAI in 2017. PPO is a policy gradient method that attempts to keep successive policy updates close together to avoid training instability. It employs an objective function to compute an update at each step that minimizes the cost function while ensuring the deviation from the previous policy is relatively small. PPO balances ease of implementation, sample complexity, and ease of tuning, making it a popular choice for continuous control tasks.

Asynchronous PPO (APPO). APPO, a variant of PPO, allows multiple workers to collect experiences independently and update the policy asynchronously (Schulman et al., 2017). This design leads to increased training speed as it can process multiple tasks simultaneously, leading to higher throughput. However, the asynchronous nature of APPO can also lead to stale experiences where some workers are using an outdated policy to generate experiences, potentially causing the learning process to stagnate.

3. Research Questions

We aim to investigate the following research question: "With a set interaction budget for the environment, is it beneficial to distribute this budget across multiple related tasks or to concentrate solely on one task and devote the entire budget to it? In other words, is it better to have a generalized agent that has been trained on multiple tasks or have an expert agent that is trained only on the target task?"

We divide the following into four simpler questions: "Given a fixed budget of interaction with the environment, is it better to do single-task learning or multi-task learning?". In fact, each of these questions tries to answer the impact of one of the varying factors on the answer to the main question. The varying factors are the similarity or distance of environments, the number of environments, the total budget that we have, and the algorithms used for training.

4. Methodology

In our exploration of the specified research question, our initial approach involves establishing foundational assumptions and implementing certain simplifications. As the study progresses, we can systematically relax these initial conditions, aiming to derive a more universal solution to the question at hand.

4.1. Assumptions and Simplifications

All experiments in our study were conducted within the four MiniGrid environments (Chevalier-Boisvert et al., 2023). We use MiniGrid 8×8 environments for all of our experiments. With the exception of section 5.2 where a range of environments was employed and also section 5.1 where we generate two random environments, all other experiments were conducted using a set of four fixed environments. These environments can be seen in Figure 2.

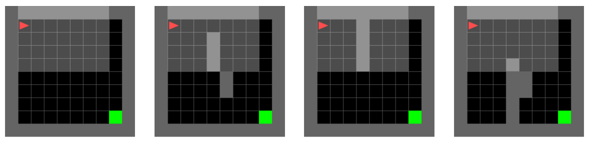


Figure 2. MiniGrid Environments (environments 1 to 4 are shown from left to right)

In all the experiments, the agent starts at the top left of the map, and our goal is at the bottom right of the map. Also, we have set the maximum number of steps that an agent can take within a single episode to 400. Once this limit is reached, the episode will terminate even if the agent has not achieved its goal. Moreover, we use 1M time steps for all the experiments with the exception of section 5.3. To facilitate analysis and visualization of the outcomes, we employ timesteps and training iterations on the X-axis, with mean reward represented on the Y-axis. Training iterations typically represent a complete cycle of data collection and learning.

4.2. Baselines

Because we are comparing single-task and multi-task approaches in different experiment setups and with different algorithms, we use a single-task approach of a certain map as its baseline and compare the results of the multi-task approach for that specific map with it.

4.3. Experiments

The experiments are divided into the following four sections:

4.3.1. Similarity of Environments

In this section, we explore the effect of task similarity, or distance, in our conclusions. First, we should have a distance metric to compare the similarity of environments in different settings. We propose a metric and mathematically prove that it is actually a distance metric.

Our proposed distance metric can be expressed as the minimum number of edits we must make to transform one environment to another, i.e., transform the first 8×8 map to the second one. This metric comprises two sub-metrics:

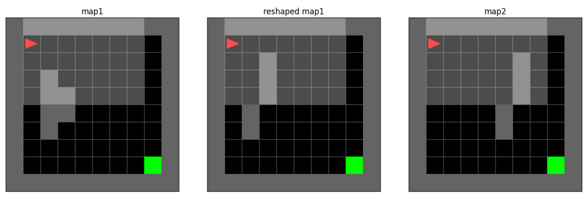
1. The number of edits required to *reshape* the shape of the internal walls of the first map to the shape of the internal walls in the second map. An edit is adding one block of wall or removing one block of wall.

2. The number of moves required to *transfer* the reshape of map1 to the location of map2. Moving the reshaped map one step in each direction counts as one edit. This second part is simply the Manhattan distance between two shapes.

The proposed distance metric can be written as follows:

$$d(x, y) = d_{reshape}(x, x') + d_{transfer}(x', y), \quad (3)$$

where x is map1, x' is the reshape of map1, and y is map2. Figure 3 shows the process of reshaping and transferring map1 to reach map2, for an example of two maps. In this example, the reshape cost is 5, the transfer cost is 3, and therefore, the distance between the two maps is 8.



(a) map1 (b) reshaped map1 (c) map2

Figure 3. The process of calculation of distance between two maps. First, the wall block of map1 gets reshaped and then transferred to where the wall block of map2 is located.

Proof of this metric is a distance metric can be found in Appendix A.

Each experiment starts with generating two random maps. We first create one map using the provided random seed and keep it unchanged. Then, we create the second map as a copy of the first one and repeatedly make minor adjustments to it until we achieve the desired distance from the first map. later, we train two models: one using a multi-tasking approach on both environments and the other trained solely on the target environment, i.e., the first of the two environments. The mean discounted reward achieved during the training process, as well as the last training step, will be stored for further analysis. This procedure will be repeated for different distances between two environments, ranging from one to twenty, and results will be averaged over ten runs, using ten different random seeds.

The mean discounted reward on the last training step indicates whether or not the agent successfully learned a good policy, while the mean discounted reward in the entire training process roughly explains how quickly our model learned a good policy. Overall, we conducted 210 experiments.

4.3.2. Number of Environments

In this section, we aim to examine the impact of varying the number of environments on the efficacy of multi-task learning. The aim of conducting this experiment is to assess the impact of generalization on both the convergence speed and the acquired reward. Our investigation seeks to determine the efficacy of generalization across multiple environments, identifying the optimal number of environments that maximizes reward acquisition and accelerates convergence for the agent. Additionally, we aim to explore whether the difficulty level of environments influences the generalization effect. To achieve this objective, we run the IMPALA algorithm across a spectrum of five distinct numbers of environments, ranging from two to six. Our working hypothesis posits that, irrespective of the specific number of environments employed in each experiment, the difficulty level of the environments escalates (in terms of both obstacle density and the complexity of paths leading to the goal). For instance, in the case of three environments, the third environment is posited to be the most challenging, while the first environment is presumed to be the least demanding.

4.3.3. Total Budget

In this section, we train the IMPALA algorithm in identical environments (given in Figure 2) under three distinct budget constraints: 500k, 1m, and 5m interactions. We also conduct another experiment on the environments shown in appendix C Figure 11. Each experiment was run on 5 constant seeds in this section.

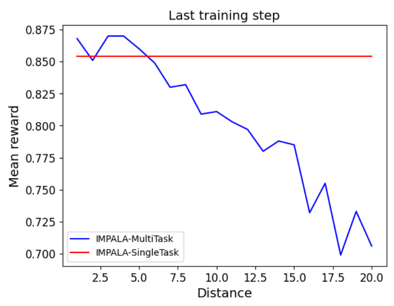
4.3.4. Algorithms

For this experiment, we ran these three algorithms (IMPALA, PPO, APPO) with 10 different constant seeds for them. These were run on our 4 default environments (Figure 2) on multi-task mode, and on the 4th map for the single-task mode.

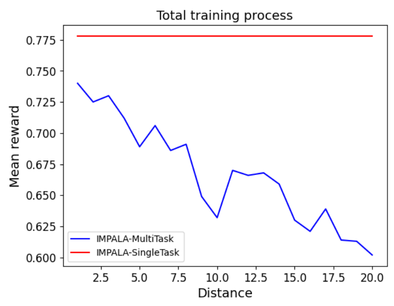
4.4. Evaluation

We assess the agent's average scores at specific intervals during the training process. The performance of the algorithms is evaluated based on the mean rewards obtained over each iteration (training step).

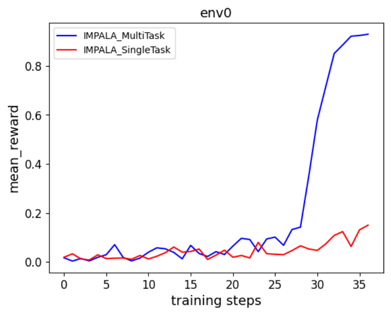
The evaluation is conducted over 10 different seeds, which are constant for all sections of the experiments (due to computation resource constraints, section 5.2 and 5.3 were run over 5 seeds). We consider an average of these different runs to ensure consistency and reliability of the results. This methodology allows us to conduct a comprehensive and reliable comparison of multi-task and single-task learning approaches across different environments and algorithms.



(a) mean reward in last training step



(b) mean reward in total training process



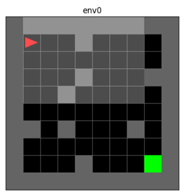
(c) a special case

Figure 4. Changes of mean reward achieved in the last training step (a), in the total training process (b), and a special case (c), where the multi-task model performs better.

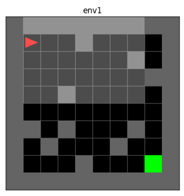
5. Empirical Results

5.1. Similarity of Environments

As shown in Figure 4 (a) and (b), the average performance of multi-task learning is close to that of single-task learning. The gap grows by increasing the distance between the two environments, which follows our initial expectation. However, this does not explain all the cases. Although the performance of the multi-task model clearly shows a downward trend, there are some variations here. We noticed that in some cases the multi-task model outperforms the single-task model. Figure 5 shows an example of this phenomenon. Figure 4 (c) compares the mean discounted reward achieved by both models in the provided example.



(a) environment 1



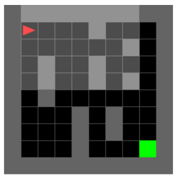
(b) environment 2

Figure 5. A special case where the performance of the multi-task model outperforms the single-task model

We can explain this case by the difficulty of each environment for the agent to learn. It appears that the agent has a difficult job to escape the cage that it has initially been captivated in. It rarely happens that it can get itself free and explore the rest of the environment. In this situation, multi-task learning becomes helpful, where the second environment is similar to the first one, but it is less difficult in the early stages. The policy learned in the second environment can help the other agent working on the first environment to find its way more easily toward the goal. Here, we didn't define the difficulty of a map precisely; instead, we relied on our intuitive sense. Defining a metric to measure the difficulty of a microgrid environment can be kept for future work.

5.2. Number of Environments

Our assumption on all these six below environments is that the agent starts at the top left of the map and our goal is at the bottom right of the map. Also, it is worth mentioning that the outcome figures are the average results of the 5 runs with specified seeds to compare more precisely. Besides the four default environments shown in Figure 2, we add two enviroments, which can be shown in Figure 6, and we consider them the fifth and sixth environments.


(a) environment 5

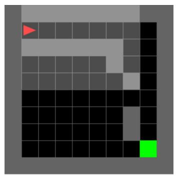
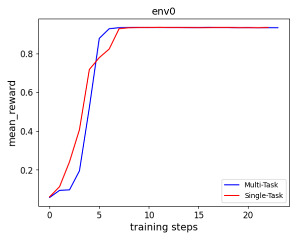

(b) environment 6

Figure 6. Environments maps

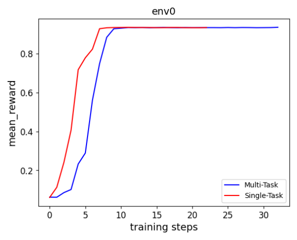
The experimental results are as follows: In initial experiments on the first two environments, multi-task learning showed faster convergence (around five training steps) compared to single-task learning (around seven steps). Subsequent experiments on the first three environments demonstrated a decline in multi-task learning performance (around eight steps), with the third environment's complexity affecting the agent's policy learning. Expanding to the first four environments, multi-task learning led to convergence at around eleven steps, echoing previous observations of performance decline due to environmental complexities. Further experiments on the first five environments showed a slight improvement in multi-task learning (around eight steps), attributed to the influence of the fifth environment.

Lastly, experiments on all six environments revealed multi-task learning convergence at around eleven steps, with performance worsening due to the complexity of the sixth environment.

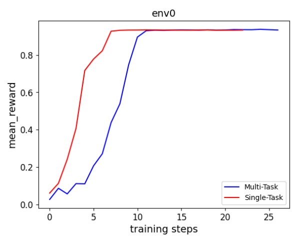
Another finding is that, in the experiment involving four environments, the performance of the single-task agent significantly outperforms the multi-task agent in the fourth environment. Similarly, in the experiment with six environments, the performance of the single-task agent excels notably in the sixth environment. The cause lies in the fact that other environments make the agent acquire a suboptimal policy. The results can be seen in Figure 7.



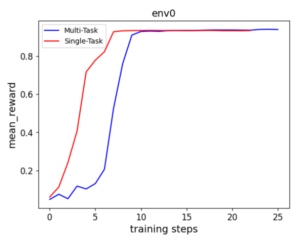
(a) 2 envs, Map 0



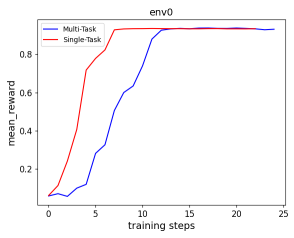
(b) 3 envs, Map 0



(c) 4 envs, Map 0



(d) 5 envs, Map 0



(e) 6 envs, Map 0

Figure 7. Environments maps

5.2.1. Analysis

Based on the outcomes, we can infer that if the environments are dissimilar and widely spaced, the performance of the multi-task learning agent tends to worsen on average as the number of environments increases. Additionally, if convergence occurs, it happens at a slower pace. This is attributed to the challenge posed by dissimilar environments, making it harder for the agent to learn an optimal policy that works effectively across all or most environments. Conversely, in cases where the environments are similar, increasing their number may lead to improved performance and faster convergence of the multi-task learning agent. This is because the agent can leverage policies learned from one environment and apply them to others. The results also suggest that in dissimilar environments, the single-task learning agent outperforms the multi-task learning agent, achieving higher rewards and faster convergence. However, in similar environments, the multi-task agent may exhibit superior performance, as seen in the first experiment involving two environments. Another noteworthy observation is the influence of environmental difficulty on generalization. For instance, the experiment involving five challenging environments reveals that the agent struggles to converge even in the supposedly easiest environment. This suggests that a diverse range of environmental difficulties, particularly when environments are dissimilar and distant, yields unfavorable results. The suboptimal performance stems from the agent's inability to learn a universally applicable policy, leading to the adoption of a subpar strategy that diminishes overall performance.

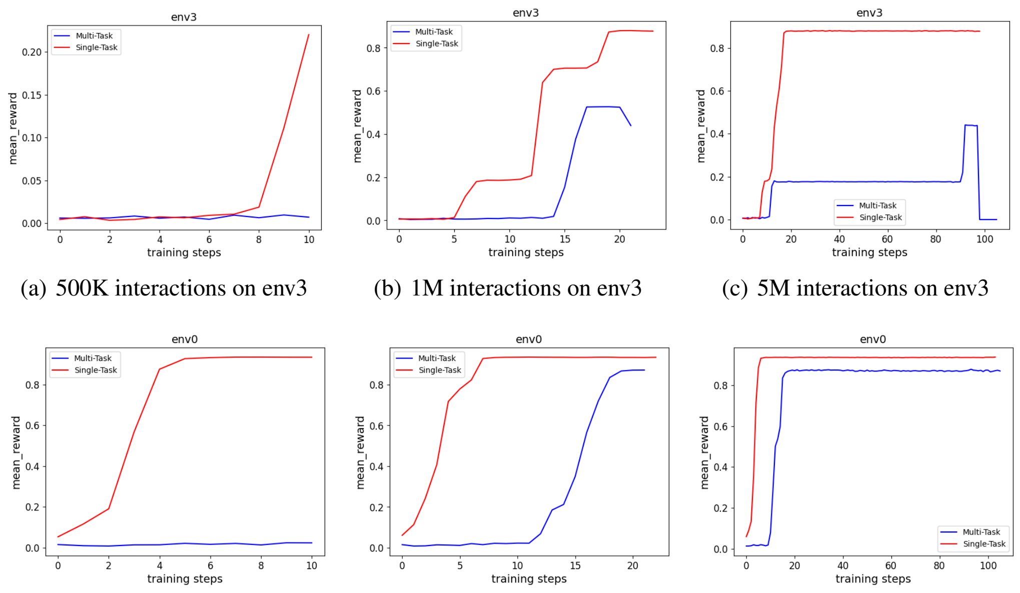
5.3. Total Budget

In this section, we focus on the critical aspect of budget allocation within a controlled experimental framework. The primary goal here is to investigate the impact of varying budgets on the outcomes of the experiment while keeping other influential variables such as the environments and choice of algorithm constant. Firstly, we hypothesize that in environments characterized by lower complexity, the single-task approach will learn faster than the multi-task approach. Conversely, in more complex environments, multi-task learning is expected to exhibit quicker adaptation by leveraging diverse experiences. Furthermore, we hypothesize that with less number of interactions, single-task shows better performance. Most importantly, we predict that after a constant number of interactions, both single-task and multi-task approaches will eventually converge to comparable performance levels, independent of the starting differences in learning rates.

5.3.1. Analysis

This section provides a detailed examination of the empirical outcomes and explores different budgets' influence on single-task and multi-task approaches. We can see the results of experiments in Figure 8.

Across all interaction budgets in the less complex environment (env0), the single-task approach consistently outperforms the multi-task approach. The simplicity of env0, devoid of obstacles, allows the single-task approach to take advantage of an accelerated learning process. Single-task



(a) 500K interactions on env3 (b) 1M interactions on env3 (c) 5M interactions on env3

(d) 500K interactions on env0 (e) 1M interactions on env0 (f) 5M interactions on env0

Figure 8. Multi-task and single-task comparison among different interaction budgets on env0 and env3

learning focuses entirely on mastering the complications of a single environment. This deeper focus proves advantageous in lower complexity environments, leading to accelerated learning and superior performance compared to the multi-task approach, which may struggle with unnecessary diversions from its primary tasks.

Over a consistent number of interactions, single-task, and multi-task approaches do not converge to identical performance levels in env0 and env3. This outcome contradicts our initial expectation of complete convergence. Variations in initial conditions can lead to divergent learning trajectories, particularly in complex environments where small differences may have a significant effect. Additionally, it depends on the tasks' complexity and relations themselves. In this specific experiment, In the case of env0, where no obstacles obstruct the path from start to goal, the multi-task approach, having learned detours from navigating environments with obstacles, may unintentionally deviate from optimal paths. This behavior could result in suboptimal performance, as the algorithm might not easily recognize and exploit the most direct path in an obstacle-free environment. Moreover, when comparing the impact of learned detours in more complex maps, such as env2 and env3, distinct differences emerge. The spatial distribution of obstacles in env2 and env3, with obstacles spanning from top to middle in one row and middle to bottom in another row, may pose specific constraints on the multi-task approach. The learned detours, adapted from navigating these obstacle configurations, might not generalize optimally to both maps. Thus, we see inconsistency and a decrease in multi-task performance over time in env3.

We employed five different seeds for our experiment, and among these seeds, only one successfully learned, while others remained stagnant with consistently zero rewards after approximately 20 iterations (the figure is given in the appendix B Figure 10). Towards the end of the plot, the seed that exhibited learning capabilities concluded its training earlier than the others. Consequently, this significant discrepancy in learning rates among the different seeds manifests as a sharp decrease in multi-task performance in the later iterations (Figure 8(c)).

Additionally, we have tested different maps to ensure if single-task always outperforms multi-task. In the environments given in the appendix Figure 11, multi-task performs better on the more complex environment and achieves higher rewards in 1M interactions. For the simpler map of 0, we can see convergence through multi-task and single-task performance as stated in the hypothesis. (appendix C Figure 12).

5.4. Algorithms

In this section, we aim to see whether there is a huge gap between the performances of the IMPALA algorithm, which is the main algorithm we are working on, and two other state-of-the-art distributed algorithms namely, PPO and APPO, and analyze the results.

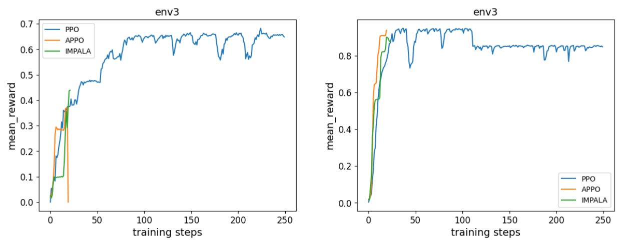


Figure 9. Comparison of different algorithms in env3 averaged on 10 runs. Left: Multi-task learning, Right: Single-task learning

5.4.1. PERFORMANCE

As shown in Figure 9, the PPO algorithm performs better in multitasking with respect to average rewards. This can be due to maintaining stability during training by using trust regions. This ensures that updates to the policy are not too large, which can be particularly beneficial in multi-task learning where the policy needs to generalize across different tasks without destabilizing. Another reason for its superior performance in obtaining higher average rewards in multi-task mode is that PPO uses synchronous updates, meaning that it waits for all workers to collect data before performing an update. This can lead to more efficient learning as the updates are based on a larger, more consistent batch of experiences.

On the other hand, PPO performs similarly with respect to average rewards to our two other algorithms on a single task on env3. In single-task learning, the advantages of PPO's on-policy learning and stability might be less pronounced compared to multi-task learning. APPO and IMPALA are both based on the actor-critic architecture and use experience replay for off-policy learning. In a single-task setting, the differences between on-policy (PPO) and off-policy (APPO, IMPALA) methods may not be as impactful, leading to similar performance.

Furthermore, we can see that PPO is very noisy compared to other algorithms here. It is mainly because of two reasons. Firstly, PPO is an on-policy algorithm, which means it learns from data collected from the current policy. This can lead to higher variance because the policy is constantly changing, and in multi-task learning, the data becomes obsolete quickly. In contrast, IMPALA and APPO can learn from off-policy data, which can be more stable as they can utilize a larger batch of experiences that may include data from multiple versions of the policy. Furthermore, policy gradient methods, including PPO, use a stochastic policy, meaning there is a certain level of randomness involved in choosing the next action. This randomness can cause significant variability in the rewards obtained during training. Secondly, APPO and IMPALA are designed to handle asynchronous data collection, which can lead to more stable learning as they are less affected by the variance in the incoming data stream. PPO typically relies on synchronous data collection, which can be more susceptible to the variance introduced by the environment or the stochasticity of the policy.

Another important observation from the Figure 9 is that the APPO's performance suddenly decreased sharply in multi-task mode. From other runs of the program, we see that APPO sometimes gets stuck and does not learn anything at all even by increasing the number of interactions to fivefold, significantly affecting the average. Our guesses for this phenomenon are that in APPO, stale experiences can occur when some workers use outdated policies to generate experiences, potentially causing the learning process to stagnate. Additionally, an imbalance in the adversarial loop, where one part of the loop progresses too far ahead of the other, can lead to sudden performance drops.

5.4.2. SPEED ANALYSIS

Lastly, about the speed of the algorithms, we see that PPO takes 10 times more time than APPO and IMPALA algorithms[1]. This lack of speed of PPO is because of its synchronous updates that bring many advantages to it but for its training speed becomes a demerit. Both IMPALA and APPO collect data asynchronously across multiple workers. This means that while one worker is collecting data, another can process it, and another can update the model. This parallelism leads to more efficient use of resources and faster overall training times. In the case of multi-task learning algorithms like IMPALA and APPO, multiple tasks are being solved in parallel during each iteration (Espeholt et al., 2018; Schulman et al., 2017). This leads to more interactions in each iteration and, thus, fewer iterations overall.

6. Conclusion

This study investigated whether it is more effective to train an RL agent as a generalist across multiple tasks or as an expert focused on a single task. The study finds that the efficiency of multi-task versus single-task learning is highly context-dependent. While single-task learning often excels, multi-task learning can surpass it under specific conditions, such as in similar but less challenging secondary environments or complex settings with constrained interaction budgets. The study also revealed that increasing the number of environments can either enhance or degrade multi-task learning effectiveness, depending on the similarity and complexity of these environments. Furthermore, the study presented a comprehensive comparison of different RL algorithms in both multi-task and single-task settings,

[1]Running 1 million interactions with 8 CPU cores on 4 maps took about 4 minutes on APPO and IMPALA but about 40 minutes on PPO.

uncovering notable variations in their performance.

7. Contributions

Amirhossein Sohrabbeig answered how the similarity or distance of two environments being used for multi-tasking affects performance. Kiana Aghakasiri explored how changing interaction budgets affects the performance of multi-tasking and single-tasking. Amirhossein Rajabpour's contribution to the project lies in the empirical evaluation of the IMPALA algorithm against two other state-of-the-art distributed RL algorithms, PPO and APPO. He has conducted a comparative analysis focusing on these algorithms in multi-task and single-task environments to determine performance gaps. Mohammad Tavakoli explored the effect of changing the number of environments on multi-task learning performance and compared the results with those of a single-task learning agent.

References

Chevalier-Boisvert, M., Dai, B., Towers, M., de Lazcano, R., Willems, L., Lahlou, S., Pal, S., Castro, P. S., and Terry, J. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. *CoRR*, abs/2306.13831, 2023.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pp. 1407–1416. PMLR, 2018.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., de Las Casas, D., Hendricks, L. A., Welbl, J., Clark, A., et al. Training compute-optimal large language models. *Preprint arXiv*, 2203.15556, 2022.

Jia, Z., Li, X., Ling, Z., Liu, S., Wu, Y., and Su, H. Improving policy optimization with generalist-specialist learning. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162, Baltimore, Maryland, USA, 2022. PMLR. Copyright 2022 by the author(s).

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *Preprint arXiv*, 2001.08361, 2020.

Reed, S., Zolna, K., Parisotto, E., Colmenarejo, S. G., Novikov, A., Barth-Maron, G., Gimenez, M., Sulsky, Y., Kay, J., Springenberg, J. T., et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. The MIT Press, 2018. ISBN 978-0262039246.

Vithayathil Varghese, N. and Mahmoud, Q. H. A survey of multi-task deep reinforcement learning. *Electronics*, 9(9): 1363, 2020.

A. Proof of distance metric

To prove that the metric introduced in section 4.3.1 is a distance metric, we have to show that it satisfies the following axioms:

1. The distance from a point to itself is zero:
$$d(x, x) = 0 \tag{4}$$

2. (Positivity) The distance between two distinct points is always positive:
$$if\ x \neq y,\ then\ d(x, y) > 0 \tag{5}$$

3. (Symmetry) The distance from x to y is always the same as the distance from y to x:
$$d(x, y) = d(y, x) \tag{6}$$

4. The triangle inequality holds:
$$d(x, z) \leq d(x, y) + d(y, z) \tag{7}$$

Here, we just have to prove these axioms only for the first part of our proposed metric, i.e., the reshape distance, because we already know the Manhattan distance is a distance metric and, therefore, satisfies those axioms.

The proofs for the first and second axioms are almost trivial:

1. The number of wall blocks to be added or removed from a map to reshape it into itself is zero; therefore, $d(x, x) = 0$.

2. Two different shapes are different at least in one wall block, so we need to add or remove at least one wall block; therefore, $d(x, y)$ must be greater than 0.

3. To prove the symmetry and triangle axioms, we first need to define the reshaping procedure in the following algorithm:

Algorithm 1 Calculate Minimum Edits to Transform Map1 into Map2

```
Input: binary maps $map1$, $map2$
Output: minimum number of edits $minEdits$
Convert $map1$ and $map2$ to arrays
Initialize $length \leftarrow \text{len}(map1)$, $width \leftarrow \text{len}(map1[0])$
Create zeroed array $paddedMap$ of size $(3 \times length - 2) \times (3 \times width - 2)$
Copy $paddedMap$ to $paddedMap1$ and $paddedMap2$
Place $map1$ and $map2$ in the center of $paddedMap1$ and $paddedMap2$ respectively
Initialize $minEdits \leftarrow \infty$
for $i = 0$ to $\text{len}(paddedMap2) - length$ do
    for $j = 0$ to $\text{len}(paddedMap2[0]) - width$ do
        Calculate $edits$ for the current overlap
        if $edits < minEdits$ then
            $minEdits \leftarrow edits$
        end if
    end for
end for
return $minEdits$
```

Intuitively, this algorithm convolves map1 with the zero-padded version of map2 to find out the maximum intersection these two maps have. Then, the distance, i.e., the minimum edits required, is calculated as the total number of internal wall blocks in two maps minus two times the number of maximum intersections, i.e., the symmetric set difference.

Therefore, no matter whether we calculate the distance between map1 and map2 or vice versa, the maximum intersection and the symmetric intersection are the same, which means it satisfies the Symmetry axiom.

4. Proving the triangle inequality is a little bit less straightforward. In fact, we have to prove that the symmetric difference satisfies the triangle inequality:

   Let A, B, and C be sets.

$$|A \triangle C| \leq |A \triangle B| + |B \triangle C| \tag{8}$$

   First, recall the definition of the symmetric difference:

$$\begin{aligned} A \triangle B &= (A \setminus B) \cup (B \setminus A) \\ B \triangle C &= (B \setminus C) \cup (C \setminus B) \\ A \triangle C &= (A \setminus C) \cup (C \setminus A) \end{aligned} \tag{9}$$

   Now consider the set $(A \triangle B) \cup (B \triangle C)$:

$$(A \triangle B) \cup (B \triangle C) = ((A \setminus B) \cup (B \setminus A)) \cup ((B \setminus C) \cup (C \setminus B)) \tag{10}$$

   This union includes all elements that are in A but not C and all elements that are in C but not A. Thus, it contains $A \triangle C$. Therefore:

$$A \triangle C \subseteq (A \triangle B) \cup (B \triangle C) \tag{11}$$

   Taking cardinalities, we have:

$$|A \triangle C| \leq |(A \triangle B) \cup (B \triangle C)| \tag{12}$$

   By the principle of inclusion-exclusion for cardinalities, we get:

$$|(A \triangle B) \cup (B \triangle C)| \leq |A \triangle B| + |B \triangle C| \tag{13}$$

   Combining the above inequalities, we conclude:

$$|A \triangle C| \leq |A \triangle B| + |B \triangle C| \tag{14}$$

B. Different runs

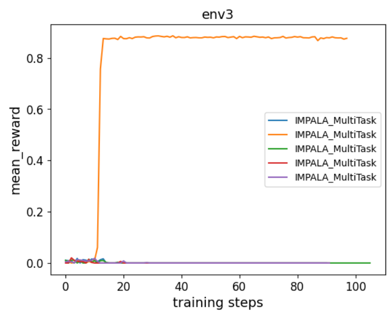


Figure 10. Impala multi-task on different seeds

C. Other experiments for section 5.3

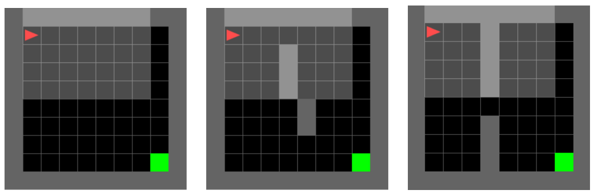
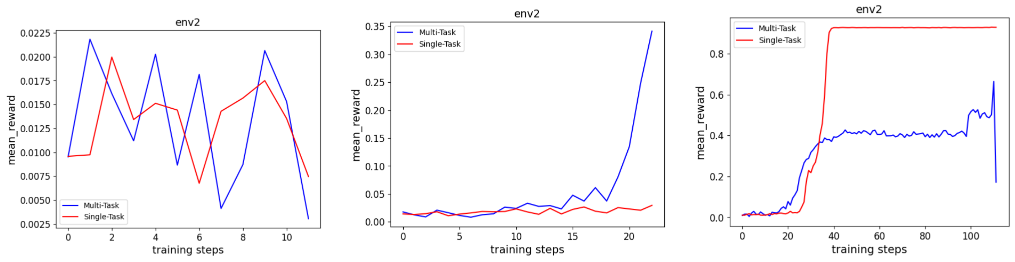

(a) environment 1 (b) environment 2 (c) environment 3

Figure 11. Environments maps


(a) 500K interactions on env2 (b) 1M interactions on env2 (c) 5M interactions on env2

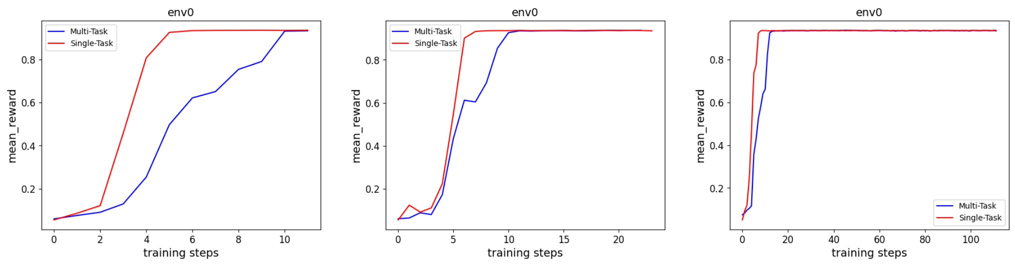

(d) 500K interactions on env0 (e) 1M interactions on env0 (f) 5M interactions on env0

Figure 12. Multi-task and single-task comparison among different interaction budgets on env0 and env2 for 3 different environments given in Figure 11. We can see convergence to the same reward in the simple environment of env0 as the maps didn't conflict and the obstacles in env1 and env2 did not stop the agent from reaching the optimal path. For the more complex environment of env2, simple-task needs more than 1M interactions to find the optimal path. Thus, multi-task performs better than single-task for less amount of budget such as 1M although the rewards are not optimum and at most, about 0.35. So we cannot state that multi-task performed notably for this case but we can claim that with a limited budget of 1M which was not enough for a single-task to learn the complex environment, multi-task outperformed, and regarding the environments, multi-task can be the better option.