GARGAML: ADDRESSING MULTITASK AND MULTIGOAL TRANSFER WITH GOAL-CONDITIONED META REINFORCEMENT LEARNING AND REPLAY

Ahmed Akakzia, Mohamed Chetouani, Olivier Sigaud

Sorbonne Université, CNRS UMR 7222
Institut des Systèmes Intelligents et de Robotique, F-75005 Paris, France
ahmed.akakzia@isir.upmc.fr

ABSTRACT

When tasks and goals are not known in advance, an agent may use either multitask learning or meta reinforcement learning to learn how to transfer knowledge from what it learned before. Recently, goal-conditioned policies and hindsight experience replay have become standard tools to address transfer between goals in the multitask learning setting. In this paper, we show that these tools can also be imported into the meta reinforcement learning when one wants to address transfer between tasks and between goals at the same time. More importantly, we investigate whether the meta reinforcement learning approach brings any benefit with respect to multitask learning in this specific context. Our study based on a basic meta reinforcement learning framework reveals that showcasing such benefits is not straightforward, calling for further comparisons between more advanced frameworks and richer sets of tasks.

1 INTRODUCTION

In an open-ended learning context (Doncieux et al., 2018), an agent faces a continuous stream of unforeseen tasks along its lifetime and must learn how to accomplish them. For doing so, two closely related reinforcement learning (RL) frameworks are available: multitask learning (MTL) and meta reinforcement learning (MRL).

Multitask learning was first defined in Caruana (1997). The general idea was to train a unique parametric policy to solve a finite set of tasks so that it would finally perform well on all these tasks. Various MTL frameworks have been defined since then (Yang & Hospedales, 2014). For instance, an MTL agent may learn several policies sharing only a subset of their parameters so that *transfer learning* can occur between these policies (Taylor & Stone, 2009). Multitask learning is the matter of an increasing research effort since the advent of powerful RL methods (Florensa et al., 2018; Veeriah et al., 2018; Ghosh et al., 2018). But this effort came with a drift from the multitask to the multigoal context. Actually, the distinction between tasks and goals is not always clear. In this paper we will rely on an intuitive notion illustrated in Figure 1a of task as some abstract activity such as *push blocks* or *stack blocks*, and we define a goal as some concrete state of the world that an agent may want to achieve given its current task, such as *pick block "A" and place it inside this specific area*. If we stick to these definition, a lot of work pretending to transfer between tasks actually transfer between goals. Anyways, for multigoal learning, goal-conditioned policies (GCPs) have emerged as a satisfactory framework to represent a set of policies to address various goals, as it naturally provides some generalization property between these goals. Besides, the use of Hindsight Experience Replay (HER) (Andrychowicz et al., 2017) has been shown to significantly speed up the process of learning to reach several goals when the reward is sparse. In this context, works trying to learn several tasks and several goals at the same time are just emerging (see e.g. Table 1 in Colas et al. (2019)).

Meta reinforcement learning is a broader framework. Generally speaking, it consists in using inductive bias obtained from learning a set of policies, so that new policies for addressing similar unknown tasks can be learned in only a few gradient steps. In this latter process called *fine tuning*,

policy parameters are tuned specifically to each new task (Rakelly et al., 2019). The MRL framework encompasses several different perspectives (Weng, 2019). One consists in learning a recurrent neural network whose dynamics update the weights of a policy so as to mimic a reinforcement learning process (Duan et al., 2016; Wang et al., 2016). This approach is classified as *context-based* meta-learning in Rakelly et al. (2019) as the internal variables of the recurrent network can be seen as latent context variables. Another perspective is efficient parameter initialization (Finn et al., 2017), classified as *gradient-based* meta-learning. Here, we focus on a family of algorithms encompassing the second perspective which started with Model-Agnostic Meta-Learning (MAML) (Finn et al., 2017). A crucial feature of this family of algorithms is that they introduce a specific mechanism to favor transfer between tasks, by looking for initial policy parameters that are close enough to the manifold of optimal policy parameters over all tasks.

Actually, though in principle MRL is more meant to address several tasks than several goals in the same tasks, in practice empirical studies often address the same benchmarks which are multigoal rather than multitask Finn et al. (2017); Rothfuss et al. (2018); Rakelly et al. (2019). Only a few papers truly transfer knowledge from one task to another (Zhao et al., 2017; Colas et al., 2019; Fournier et al., 2019), but at least the latter two do not classify themselves as performing MRL.

The common practice in MTL consists in sampling from all target tasks, which limits its applicability to the open-ended learning context, where some target tasks may not be known in advance. Besides, transfer between tasks in MTL, or rather between goals, usually relies on the generalization capability of the underlying function approximator, without any specific mechanism to improve it, resulting in potential brittleness when this approximator is not appropriate. Nevertheless, when applied to the multigoal setting using GCPs and HER, this approach has been shown to provide efficient transfer capabilities.

By contrast, in MRL, the test tasks are left apart during training. To ensure good transfer to these unseen tasks, the approach in MAML consists in iteratively refining the initial policy parameters so that a new task can be learned through only a few gradient steps. In principle, this latter transfer-oriented mechanism makes MRL more appropriate for open-ended learning, where the agent cannot train in advance on future unknown tasks.

The goal of this paper is to show that the above multigoal and multitask learning mechanisms can be combined into a single learning framework addressing open-ended learning when each task can be instantiated with multiple goals. By doing so, we also clarify the relationship between these mechanisms and show that they are more complementary than competing.

The main contributions of the paper are the following:

- We design a new MAML-like MRL algorithm named GARGAML[1]. GARGAML is based on the Soft Actor Critic (SAC) algorithm (Haarnoja et al., 2018a), a sample efficient off-policy RL algorithm, it incorporates GCPs and HER to deal with multiple goals, and uses the specific transfer-oriented mechanism of MAML to deal with multiple tasks. We show how each of these components contributes to the better performance of the global system in a series of benchmarks.
- We show that, when applied to a single task with multiple goals, the specific transfer-oriented mechanism of MAML does not bring any benefit with respect to just using GCPs and HER, even when clearly separating training goals from test goals.
- By contrast, we show that the same mechanism plays a crucial role for transfer between training tasks and test tasks, opening the way to efficient multigoal open-ended learning.

2 Related work

Deep RL is a successful framework where an agent can learn to solve a task or reach a goal by maximizing an external reward signal. Recently, the field has expanded towards agents simultaneously addressing a set of such tasks (e.g. Zhao et al. (2017)) or goals (e.g. Florensa et al. (2018); Veeriah et al. (2018); Ghosh et al. (2018)), but only few works address multiple tasks and multiple goals at the same time (Colas et al., 2019; Fournier et al., 2019). Actually, the latter two demonstrate some

[1] for "Goal-Aware Replay in Generalization-Apt Meta-Learning"

transfer between tasks instantiated with multiple goals as we are doing here, but they do so in the context of intrinsically motivated agents learning their own curriculum, which is not our focus.

The recognition that generalization should be addressed by more clearly distinguishing a set of training tasks and test tasks is recent in reinforcement learning. This point is central to Zhao et al. (2019), which also provides a comparison between MTL and MRL in the context of a few dedicated benchmarks. Our work goes beyond this by combining some MTL and some MRL mechanisms rather than just comparing them.

Our main contribution in this paper is an MRL algorithm based on MAML which incorporates SAC, GCPs and HER, resulting in much better performance. At the moment, the MRL framework is the focus of intensive research effort. Starting from MAML, the REPTILE algorithm (Nichol et al., 2018) replaced a second order gradient calculation by a faster first order one, showing performance improvement in most cases. Leveraging the exploration strategy of meta-learning with a multi-actor perspective, the MAESN algorithm (Gupta et al., 2018) provided again improved performance and stability. More recently and closer to our approach, the PEARL algorithm (Rakelly et al., 2019) showed impressive increase of sample and time efficiency by relying on SAC instead of REINFORCE and TRPO. But PEARL uses latent context variables resulting in the capability to consider richer relationships between tasks, and can be classified as context-based rather than gradient-based meta-learning. Closer to the original MAML algorithm, the MAML ++ algorithm (Antoniou et al., 2018) also improves stability, sample efficiency and performance by fixing a few issues in the original algorithm. However, the experimental study is focused on supervised meta-learning rather than on MRL, whereas our focus here is on RL.

To summarize, to our knowledge, no MRL algorithm addresses multiple goals and multiple tasks at the same time, neither GCPs nor HER have been incorporated into an MRL algorithm yet and SAC has not been used at the heart of a typical gradient-based MRL algorithm, even if it has already been used in the more context-based PEARL algorithm.

3 BACKGROUND

Our work compares and combines several existing mechanisms and algorithms which are briefly described in the following sections. For more details, we refer the reader to the original papers.

3.1 SOFT ACTOR CRITIC

The Soft Actor Critic (SAC) algorithm (Haarnoja et al., 2018a;b) is a state-of-the-art deep reinforcement learning algorithm which builds on many ideas from the DDPG and TD3 deterministic actor-critic algorithms, but uses a stochastic actor together with an entropy regularization mechanism. By doing so, it benefits from the off-policy critic update mechanism of DDPG and TD3, making it possible to use a replay buffer and improve sample efficiency, but it also avoids the instability of these algorithms by letting the entropy regularized stochastic actor ensure efficient exploration.

3.2 GOAL-CONDITIONED POLICIES

An early version of goal-conditioned policies (GCPs) before the emergence of deep RL was proposed in Kaelbling (1993) to deal with dynamically changing goals. The idea of learning policies conditioned on goals with RL gained a lot of popularity more recently based on Universal Value Function Approximators (UVFAs) (Schaul et al., 2015). This approach is now ubiquitous in MTL, where several works learn them with goal-parametrized reward functions (Florensa et al., 2018; Veeriah et al., 2018).

The notion of GCP itself is formalized in Ghosh et al. (2018). It corresponds to a policy where the action is conditioned both on the state of the agent and on a goal. When GCPs are represented with a neural network, they can generalize not only over states but also over goals, leading to a form of transfer which does not require a specific mechanism.

3.3 HINDSIGHT EXPERIENCE REPLAY

The Hindsight Experience Replay algorithm (Andrychowicz et al., 2017) proposes a simple yet effective mechanism to address an environment with several goals when the reward function for reaching goals is sparse. The general idea is that, whenever an agent is aiming for a goal but reaches a different state, its trajectory can be used to learn how to reach that particular state as a fictive goal. Implementing HER is straightforward in the context of RL algorithms using a replay buffer and using GCPs as a multigoal policy representation. Given that SAC uses a replay buffer, we combine it with GCPs and HER as our MTL baseline in the empirical studies below, and we call the resulting system SAC +GCPs +HER.

3.4 MODEL AGNOSTIC META LEARNING

The purpose of the MAML algorithm is to make profit of training tasks to provide a good initialization of policy parameters so that these policies can be specialized to unseen test tasks after a subsequent fine tuning stage in very few gradient steps, aka few "shots". The initialization stage starts with some policy parameters θ and comes with two loops. For each sampled training task τ_i, the inner loop first collects data from running the current policy on τ_i and separately improves the corresponding parameters θ_i from θ using few steps (typically one) of the REINFORCE vanilla policy gradient algorithm (Williams, 1992). Then, during the outer loop, these improved policies are trained again using Trust Region Policy Optimization (TRPO) (Schulman et al., 2015), and a sum of the resulting gradients is finally applied to θ to get the initial parameters of the next iteration. These meta-gradient updates contain a gradient of a gradient which can either be computed using a Hessian-vector product or approximated to first-order gradient (see Weng (2019) for a clear presentation). In practice, the obtained parameters θ are shown to be a good starting point to learn policies for new unseen tasks in few shots.

Importantly, a simple analysis shows that, when the first-order gradient approximation variant is used, the specific transfer-oriented mechanism of MAML is not much different from performing a simple averaging over the gradient obtained for each task, which is what MTL would do (see Appendix E for details).

4 METHODS

In this section, we present our new GARGAML algorithm, explaining how we combined various existing components into a single architecture. We first start by clarifying the notions of tasks and goals incorporated in this paper. We then give a high level explanation of how we incorporate SAC into an MTL workflow. After that, we present GARGAML as an MRL multigoal algorithm.

4.1 MARKOV DECISION PROCESSES, TASKS AND GOALS

The gap that exists between MDPs, tasks and goals is very tight. Actually, and as we mentioned earlier in this report, the literature usually opts for the terms multitask and multi-goal interchangeably and without any specific disentanglement. We propose here to confront the formal definitions in order to get a better view on the distinctions and links existing between these terms. We think this is crucial for the understanding of the rest of the work.

On the one hand, an MDP $\mathcal{M}$ is defined as a set of different components $\{\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma\}$. In other words, an MDP is a Markov Chain (in the sense that it satisfies the Markov Property), augmented by a reward signal and a control mechanism. On the other hand, a task $\mathcal{T}$ is formally defined with reference to an MDP as follows (Finn, 2018):

$$\mathcal{T}_{\mathcal{M}} = \mathcal{T} = \{p(s_0), p(s_{t+1}|s_t, a_t), \mathcal{L}, H\}$$

where s_t is a state in $\mathcal{S}$, a_t an action in $\mathcal{A}$, $\mathcal{L}$ a loss function and H the time horizon. The state and action spaces and the probability distribution of transitions are the components of the MDP $\mathcal{M}$ to which is referenced the task $\mathcal{T}_{\mathcal{M}}$. Hence, for a single fixed MDP, we can construct multiple tasks only by changing the initial state probability distribution or the form of the loss function without

modifying the nature of the reward signal. Moving on, a goal g is an element in $\mathcal{G}$ such as $\mathcal{G} \subset \mathcal{S}$. In other words, a goal g is a reachable state s or a partially-reachable state $\tilde{s}$.

Usually, reaching a goal terminates the episode. Thereof, the goal is a signal on which depends the rewards received by the agent. If this signal is internal, meaning that the MDP is defined as $\{\mathcal{S}*, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma\}$ where $\mathcal{S}* = \mathcal{S} \times \mathcal{G}$, we are in the context of GCPs. If the goal signal is external from the MDP (we are not using GCPs), each goal exclusively define a task. Hence, in this case, multi-goal and multitask mean the same thing.

In Appendix G, we prove that in the context of GCPs and under some constraints, a multi-goal problem corresponds to one single task.

4.2 USING SAC TO PERFORM SAMPLE EFFICIENT MRL

The MAML algorithm uses two nested loops based on REINFORCE and TRPO. These algorithms are stable, but being on-policy they lack sample efficiency and they would benefit from a more efficient exploration strategy.

By contrast, the SAC algorithm is not only sample efficient as it uses a replay buffer, but also more robust due to its efficient entropy-based exploration strategy. However, SAC does not incorporate a mechanism for transferring knowledge between tasks or goals.

As outlined in Section 3, we first designed a SAC +GCPs +HER algorithm to deal with multiple goals. We now present the GARGAML algorithm which uses SAC +GCPs +HER in its inner and outer loops and combines it with the transfer-oriented mechanism of MAML to deal with multiple tasks and perform efficient transfer from training tasks to test tasks in a few shots. However, although MAML is on-policy and uses actual trajectories to perform updates, GARGAML uses two different replay buffers: one before the inner update and one after the inner update to be used in the meta-optimization update of the outer loop.

4.3 USING SAC IN A MTL APPROACH: SAC +HER MTL

In this section, we present the MTL algorithm we use in Section 6.2 to compare our MAML-based algorithm's initialization with a classic MTL-based one. We consider exactly the same definition as SAC algorithm. Nevertheless, we randomly sample a new task from the training-set tasks at each epoch. If we denote by $|\mathcal{T}^{train}|$ the number of training tasks, each task is then chosen with probability $\frac{1}{|\mathcal{T}^{train}|}$. We share the same replay buffer among all these training tasks. The latter is motivated by the fact that in the context of sparse reward signals where we use GCPs, all the training tasks receive similar rewards as a function of the achieved goal and the desired one.

4.4 GOAL-AWARE REPLAY IN GENERALIZATION-APT META-LEARNING

In this section, we give the key components of our new Goal-Aware Replay in Generalization-Apt Meta-Learning algorithm (GARGAML) which is based on a multi-goal perspective and dedicated for meta-learning some useful initialization for fast adaptation within a few gradient steps when encountering previously inexperienced tasks. The foundations of GARGAML makes it an interesting candidate that provides a benchmark of comparison between the gradient-based meta-learning and goal conditioned MTL.

GARGAML gives a possibility to use MAML off-policy, thus dodging the sample inefficiency issue and providing more flexibility to actually test transfer learning between tasks and not only goals. By definition in Section 3.4, the data used by MAML in reinforcement learning corresponds to whole trajectories. In fact, the training agent performs K roll-outs before and after the inner update for each task. These roll-outs are stored as a batch of history to be used either for optimizing the inner loss or the outer loss. Here, we propose to use replay buffers and sample transitions instead of using whole trajectories. Based on the fact that the algorithm is permutation invariant (Finn, 2018), we propose a fast mechanism to perform nested optimizations aiming at finding optimal sets of parameters in the manifold between task-specific optimal parameters set.

Given that MAML deals in an iterative scheme with each of the meta-training tasks, it is important that the transitions sampled from the buffer correspond to the actual task at hand. Assuming that each task corresponds to a different Markov Decision Process, using one single replay buffer is not efficient: imagine that you meta-train a robot to push blocks on a table, meaning that you are looking for an initialization of the parameters of the models such that few inner gradient updates are sufficient for optimal performance, then using transitions from a block stacking task would be absurd. Another way of seeing this: at meta-testing time, you would have one single unknown task that you want to solve. You would sample transitions for this unique task, however, during training, you used transitions from multiple tasks. This may lead to an actual underfitting problem growing as the tasks differ from one another.

Hence, we use a multitask replay buffer: a dictionary of replay buffers for each task. When collecting trajectories for the task $\mathcal{T}_i$, we would store them in the replay buffer $\mathcal{R}_i$.

Moving on, GARGAML is based on the SAC algorithm and detailed in Haarnoja et al. (2018a) and Haarnoja et al. (2018a). As an actor-critic algorithm, SAC optimizes a loss function for the actor $\mathcal{L}_{actor}$ and a loss function for the critic $\mathcal{L}_{critic}$. However we remind that there are constraints on the loss function in meta-learning that we briefly highlighted in Section 3.4 and that are more detailed in Finn (2018). We show in Appendix F that the actor loss function $\mathcal{L}_{actor}$ is not adequate for gradient-based meta-reinforcement learning. Thereof, we only perform the meta-optimization step using gradients of the critic loss function $\nabla_\theta \mathcal{L}_{critic}$, meaning that we transfer the actor parameters without meta-optimization step whereas the critic parameters receive this meta-optimization step. Algorithm 1 shows more details about how GARGAML works.

Algorithm 1 GARGAML for Reinforcement Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: Step size parameters for each network
  randomly initialize ϕ, θ_1, θ_2
  target Q networks: $\bar{\theta_1} = \theta_1$ and $\bar{\theta_2} = \theta_2$
  initialize buffers $\mathcal{R} = \{\mathcal{R}_1, \mathcal{R}_2, ...\}$ and $\mathcal{R}_{meta} = \{\mathcal{R}_{meta1}, \mathcal{R}_{meta2}, ...\}$
  while not done
    Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
    for all $\mathcal{T}_i$
      for all environment steps
        $a_t \sim \pi_\phi(a_t|s_t)$
        $s_t \sim p(s_{t+1}|s_t, a_t)$
        $\mathcal{R}_i \leftarrow \mathcal{R}_i \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$
      Use HER to append $\mathcal{R}_i$
      for all gradient steps
        update parameters $\phi^i, \theta_1^i, \theta_2^i$ using $\mathcal{R}_i$ and $\mathcal{T}_i$
      reset environment
      for all environment steps
        $a_t \sim \pi_\phi(a_t|s_t)$
        $s_t \sim p(s_{t+1}|s_t, a_t)$
        $\mathcal{R}_{metai} \leftarrow \mathcal{R}_{metai} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$
    Sample batch of meta tasks $\mathcal{T}_i^{meta} \sim p(\mathcal{T})$
    Meta-optimization update θ_1 and θ_2 using $\mathcal{R}_{meta}$ and $\mathcal{T}_i^{meta}$ (Only the critic)

5 EXPERIMENTAL SETUP

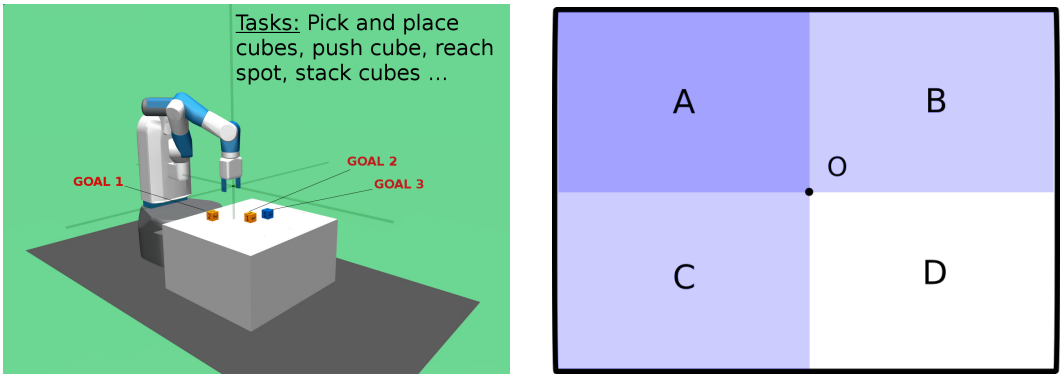
The GARGAML algorithm combines multigoal RL using SAC +GCPs +HER and MRL to learn new multigoal tasks in few shots. However, when including GCPs in an MRL framework, it might be the case that the underlying policy representation is rich enough so that a single policy parameter vector θ can solve all tasks without any subsequent fine tuning.

Our first experiment is designed to stress that this perspective, which considers MTL and MRL as two competing mechanisms to address multigoal RL problems, is inadequate.

We show that, when addressing a single task with multiple goals, the transfer-oriented mechanism of MAML that we included in GARGAML does not bring any performance improvement with respect to SAC +GCPs +HER.

So, is the transfer-oriented mechanism of GARGAML useless? To answer this question, our second experiment compares two initialization stages with multigoal training tasks, one with SAC +GCPs +HER and one with GARGAML. We show that the transfer-oriented mechanism of GARGAML is crucial for subsequent fine tuning on unseen tasks, thus for open-ended learning.

We perform the above experiments using 4 environments of increasing dimension and complexity, based on the FETCH robotics environment Plappert et al. (2018), namely FETCHREACH-v1, FETCHPUSH-v1, FETCHSLIDE-v1 and FETCHPICKANDPLACE-v1. After describing them, we present our evaluation protocol.



(a) (b)

Figure 1: (a) The FETCH environment, with various tasks and goals. (b) Schema of the multigoal evaluation protocol in the first experiment.

5.1 FETCH ENVIRONMENTS

All the FETCH environments we used are based on a 7-DoF simulated robotics arm with a two-fingered parallel gripper (Plappert et al., 2018), see Figure 1a. Actions are 4-dimensional: 3 dimensions specify the desired gripper movement in Cartesian coordinates and the last dimension controls whether the gripper is open or closed. In all or environments, rewards are sparse and binary: the agent obtains a reward of 1 if the goal is reached and 0 otherwise. The agent action frequency is $f = 25$ Hz. In order to provide the agent with a GCP representation, we append the 3 values corresponding to the goal position to the state representation.

5.2 PROTOCOL IN THE FIRST EXPERIMENT

We first compare GARGAML and SAC +GCPs +HER in their ability to transfer between multiple goals in the context of a single task.

The 2D plane is decomposed into 4 zones as depicted in Figure 1b: zones A, B and C form the training zone ABC, whereas zone D is the testing zone. Goals are sampled uniformly in the 3D space generated by these zones depending on whether this is a training or a test trial.

The goal comes with a sparse reward function which is $+1$ if the Euclidean distance between the agent and the goal is less or equal than the predefined threshold. The starting position of the agent is in O at the center of the 3D plane and it must reach the rewarded area at each episode within a horizon of H timesteps.

For all experiments in the FETCHREACH-v1, FETCHPUSH-v1 and FETCHPICKANDPLACE-v1 environments, training uses 50 epochs, and 100 in FETCHSLIDE-v1. Each epoch contains 2 cycles of 50 episodes of length 50 each, and we use 5 seeds for each training process. Further details about the configuration and parameters are given in Appendix A.
Evaluation is performed as follows: at the end of each epoch, the obtained parameters are tested on goals drawn from the testing zone, where the evaluation metric is the success rate.

5.3 PROTOCOL IN THE SECOND EXPERIMENT

In this section, we consider transfer between tasks rather than between goals.

We consider the standard FETCHREACH-v1, FETCHPUSH-v1, FETCHPICKANDPLACE-v1 and FETCHSLIDE-v1FETCH robotics environments. Among these environments, we consider two sets of tasks: the training tasks set contains FETCHREACH-v1, FETCHPUSH-v1 and FETCHPICKANDPLACE-v1, whereas the test tasks set just contains FETCHSLIDE-v1. As mentioned is Section 4.4, we share the same replay buffer for the training tasks in the SAC +HERMTL algorithm. For the proposed GARGAML algorithm, we opt for task-specific replay buffer since we are interested in updating task-specific copies of the model parameters for each of the training tasks separately in the inner loop. Furthermore, task-specific replay buffers allow the evaluation of the fine-tuning for each task during the meta-optimization step.

6 EXPERIMENTAL RESULTS

As stated in Section 5, our experimental work is designed to answer the following questions:

- Does the specific transfer-oriented mechanism of GARGAML facilitate learning a single policy to reach multiple goals in a single task?
- Does it facilitate learning new tasks with multiple goals when using fine tuning after finding a good initialization from training tasks?

The results of the corresponding experiments are described below.

6.1 ONE TASK, MULTIPLE GOALS

In this section, we train the GARGAML agent and the SAC +GCPs +HER agent to reach multiple goals in the same task for two different tasks.

Results are presented in Figure 2. One can see that both in FETCHREACH-v1 (Figure 2a) and FETCHPUSH-v1 (Figure 2b), both agents perform at the same level and reach a success rate of 100%.

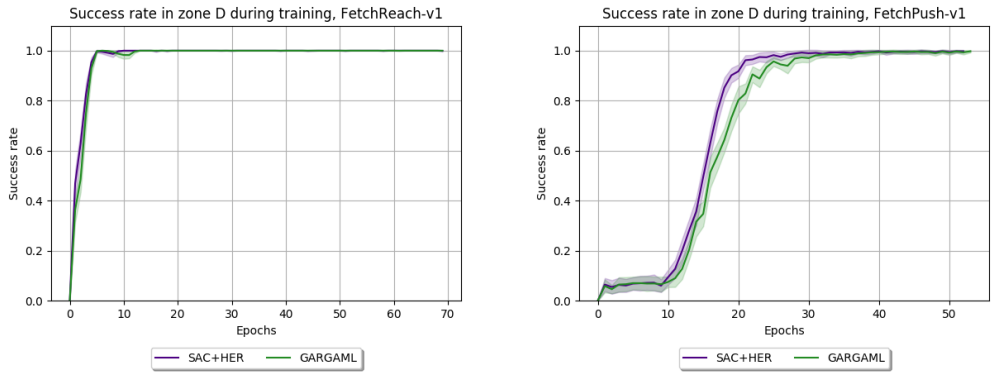
From the fact that they reach a 100% success rate, one can conclude that fine tuning is not necessary in this context, because using GCPs makes it possible to represent all goals with a unique set of parameters.

From the fact that GARGAML and SAC +GCPs +HER perform very similarly, one can conclude that in this context, the transfer-oriented mechanism of MAML does not bring any benefit in terms of generalization capability. This is not much surprising given that this mechanism is not much different from gradient averaging as performed in standard MTL. The open question is now: does this mechanism make a significant difference when transferring from some tasks to other unseen tasks and using fine tuning to address these new tasks? We investigate this question in the next section.

6.2 UNSEEN TASKS WITH MULTIPLE GOALS

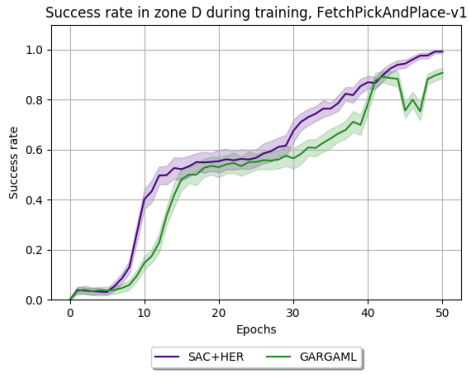
In this section, we compare:

- Fine tuning starting from the initial policy parameters learned with GARGAML, versus learning a policy on the testing held-out task from scratch.
- GARGAML's capability to accelerate the acquisition of the new held-out testing task versus that of SAC +HER MTL trained not only on one task but on the whole training tasks.



(a)

(b)



(c)

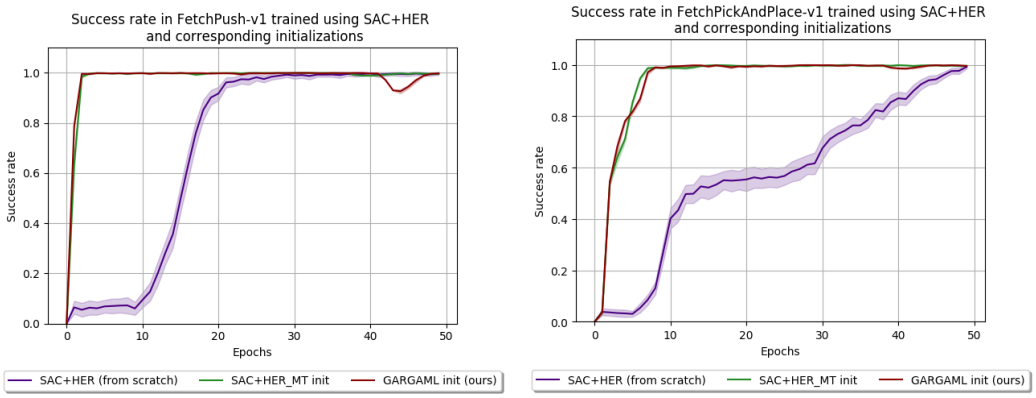
Figure 2: Success rate progress in testing zone D during training (500 epochs) in (a) FETCHREACH-v1 (b) FETCHPUSH-v1 (c) FETCHPICKANDPLACE-v1

Results are presented in Figure 3. One can see that an initialization learned using the training tasks accelerates the fine tuning for each of these training tasks. This shows that the tasks used for training share a certain structure and that we managed to find optimal parameters capable of performing quite well on all these tasks at the same time. In other words, the sets of optimal parameters of the training tasks intersect and the learned parameters lay somewhere in this intersection. Otherwise, we could have had a catastrophic forgetting phenomenon. Thus, it would have been impossible to find optimal parameters able to perform optimally on all training tasks at the same time.

Furthermore, Figure 3a shows that SAC +HER MTL manages to converge quicker that GARGAML. This is not much surprising, as SAC +HER MTL is designed to perform well on the training tasks set whereas GARGAML is designed to provide a quick fine tuning when facing new held-out tasks via the meta-optimization step.

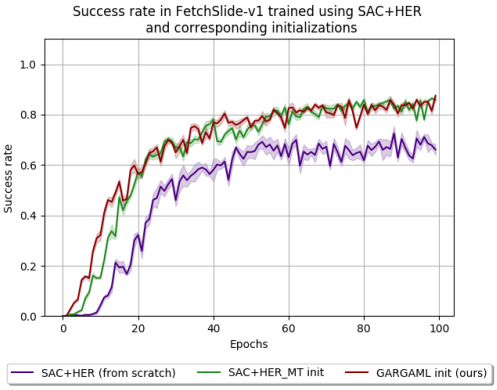
Finally, Figure 3c shows that SAC +HER MTL provides a better initialization than GARGAML as fine tuning from the former is quicker than from the latter. This means that:

- either the meta-optimization step is not needed to efficiently transfer learning between tasks when working with GCPs;
- or the FETCH robotics environments are so similar that an additional meta-optimization step with a large meta-learning rate is detrimental to the transfer of skills.



(a)

(b)



(c)

Figure 3: Success rate progress using SAC +HER algorithm with different initializations on previously seen training tasks (a) FETCHPUSH-v1 (b) FETCHPICKANDPLACE-v1 and on the held-out testing task (c) FETCHSLIDE-v1

7 CONCLUSION AND FUTURE WORK

The first experiment shows that using MRL is useless when dealing with multiple goals of a single task in the context of GCPs. The second experiment shows that our proposed GARGAML algorithm is faster when adapting to new previously unseen multigoal tasks. This provides us more insight about the role of GCPs:

- First, let's consider the single task, multiple goals case. Without GCPs, the agent must explore when encountering new goals. The only thing that differs is the reward function, which is function of the goal. If the goal is appended to the state space using GCPs, the reward function depends only on the state (implicitly on the goal, but here it is included in the state). So using GCPs makes our model Goal-Agnostic.
- FETCHREACH-v1, FETCHPUSH-v1 and FETCHPICKANDPLACE-v1 share the same state space, action space and probability distribution. Only the reward function differs, depending on the red spot position. When using GCPs, the reward function becomes Goal-Agnostic, thus the model itself becomes Task-Agnostic for the three tasks above. This explains why we can find a policy that performs well on all these tasks at the same time. However, FETCHSLIDE-v1 does not share the same probability distribution with the other tasks as the object's friction changes. So the model needs to adapt to this change. We showed that GARGAML is slightly faster that the used MTL approach. In fact, the meta-optimization step incorporated in GARGAML makes it more sensitive to the changes in the

task as the parameters found by the algorithm lie in the manifold between the set of FETCH tasks. This is why it slightly outperforms the concurrent MTL approach.

From the above thoughts, and even though we managed to show that GARGAML is slightly better than the MTL approach when adapting to the new unseen multigoal task, it is clear that we cannot conclude outright that MRL always outperforms MTL in the context of multigoal, multitask transfer: using a richer of test tasks is necessary, and further experiments with additional MRL mechanisms such as the ones used in PEARL may provide different results. Nevertheless, in this report we have provided the first empirical comparison between MTL and MRL when using GCPs from both sides, and our results have shown that there might be an advantage in using MRL to transfer the learning between multigoal tasks.

REFERENCES

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight Experience Replay. *arXiv preprint arXiv:1707.01495*, 2017.

Antreas Antoniou, Harrison Edwards, and Amos Storkey. How to train your MAML. *arXiv preprint arXiv:1810.09502*, 2018.

Richard Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997.

Cédric Colas, Pierre-Yves Oudeyer, Olivier Sigaud, Pierre Fournier, and Mohamed Chetouani. CURIOUS: Intrinsically motivated multi-task, multi-goal reinforcement learning. In *International Conference on Machine Learning (ICML)*, pp. 1331–1340, 2019.

Stephane Doncieux, David Filliat, Natalia Díaz-Rodríguez, Timothy Hospedales, Richard Duro, Alexandre Coninx, Diederik M. Roijers, Benoît Girard, Nicolas Perrin, and Olivier Sigaud. Open-ended learning: a conceptual framework based on representational redescription. *Frontiers in Robotics and AI*, 12, 2018. doi: 10.3389/fnbot.2018.00059.

Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. RL^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

Chelsea Finn. *Learning to Learn with Gradients*. PhD thesis, UC Berkeley, 2018.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1126–1135. JMLR. org, 2017.

Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In *International Conference on Machine Learning*, pp. 1514–1523, 2018.

Pierre Fournier, Olivier Sigaud, Mohamed Chetouani, and Cédric Colas. CLIC: Curriculum learning and imitation for feature control in non-rewarding environments. *arXiv preprint arXiv:1901.09720*, 2019.

Dibya Ghosh, Abhishek Gupta, and Sergey Levine. Learning actionable representations with goal-conditioned policies. *arXiv preprint arXiv:1811.07819*, 2018.

Abhishek Gupta, Russell Mendonca, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Meta-reinforcement learning of structured exploration strategies. In *Advances in Neural Information Processing Systems*, pp. 5302–5311, 2018.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018a.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018b.

Leslie Pack Kaelbling. Learning to achieve goals. In *IJCAI*, pp. 1094–1099, 1993.

Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, et al. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*, 2018.

Aravind Rajeswaran, Chelsea Finn, Sham Kakade, and Sergey Levine. Meta-learning with implicit gradients. *arXiv preprint arXiv:1909.04630*, 2019.

Kate Rakelly, Aurick Zhou, Deirdre Quillen, Chelsea Finn, and Sergey Levine. Efficient off-policy meta-reinforcement learning via probabilistic context variables. *arXiv preprint arXiv:1903.08254*, 2019.

Jonas Rothfuss, Dennis Lee, Ignasi Clavera, Tamim Asfour, and Pieter Abbeel. Promp: Proximal meta-policy search. *arXiv preprint arXiv:1810.06784*, 2018.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International Conference on Machine Learning*, pp. 1312–1320, 2015.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR, abs/1502.05477*, 2015.

Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685, 2009.

Vivek Veeriah, Junhyuk Oh, and Satinder Singh. Many-goals reinforcement learning. *arXiv preprint arXiv:1806.09605*, 2018.

Jane X. Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z. Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

Lilian Weng. Meta reinforcement learning. *lilianweng.github.io/lil-log*, 2019. URL `http://lilianweng.github.io/lil-log/2019/06/23/meta-reinforcement-learning.html`.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Yongxin Yang and Timothy M. Hospedales. A unified perspective on multi-domain and multi-task learning. *arXiv preprint arXiv:1412.7489*, 2014.

Chenyang Zhao, Timothy Hospedales, Freek Stulp, and Olivier Sigaud. Tensor-based knowledge transfer across skill categories for robot control. In *Proceedings IJCAI*, 2017.

Chenyang Zhao, Olivier Sigaud, Freek Stulp, and Timothy M Hospedales. Investigating generalisation in continuous deep reinforcement learning. *arXiv preprint arXiv:1902.07015*, 2019.

A HYPER-PARAMETERS

In this appendix, we present the hyper-parameters that we used in our experiments for each agent during training and fine-tuning.

B PARAMETERS USED WITH SAC +HER

Parameter	Value
Actor network learning rate	0.001
Critic network learning rate	0.001
Entropy learning rate	0.001
Discount factor for critic updates γ	0.98
Soft target update parameter α	0.001
Maximum size of the replay buffer	1000000
Size of minibatch for minibatch-SGD	256
Random seed	123
Number of epochs	200
Maximum number of episodes per epoch	50
Number of gradient updates	40
Maximum episode length	50
HER	true / false
k her	4

C PARAMETERS USED WITH GARGAML

Parameter	Value
Actor network learning rate	0.001
Critic network learning rate	0.001
Entropy learning rate	0.001
Actor network meta-learning rate	0.005
Critic network meta-learning rate	0.005
Discount factor for critic updates γ	0.98
Soft target update parameter α	0.001
Maximum size of the replay buffer	1000000
Size of minibatch for minibatch-SGD	256
Size of minibatch for meta-optimization	128
Random seed	123
Number of epochs	200
Number of inner gradient updates	10
Maximum number of episodes per epoch	50
Maximum episode length	50
HER	true / false
k her	4

D PARAMETERS USED WITH SAC +HER MTL

Parameter	Value
Actor network learning rate	0.001
Critic network learning rate	0.001
Entropy learning rate	0.001
Discount factor for critic updates γ	0.98
Soft target update parameter α	0.001
Maximum size of the replay buffer	1000000
Size of minibatch for minibatch-SGD	256
Random seed	123
Number of epochs	200
Number of gradient updates	20
Maximum number of episodes per epoch	50
Maximum episode length	50
HER	true / false
k her	4

E GRADIENTS IN MAML AND MTL

In this section we compare the computation of gradients in MAML and the MTL algorithm we used. At first glance, the differences are the following:

- MAML uses a meta-optimization step, making it necessary to distinguish gradient computation in the inner and outer update.
- MAML requires performing new rollouts after each update in order to obtain the validation trajectory set.
- In MAML, the meta-parameters only change at the end of each epoch. This means that back-propagation needs to be performed through the outer and the inner update. Hence, the gradients are always computed with respect to the initial parameters. By contrast, with MTL, the model parameters change at each update.

In the more detailed presentation given below, we are interested in the expression of the gradient at the end of a single epoch. Notations are provided bit by bit as we show the computational details.

E.1 MAML GRADIENT

We initialize the model parameters θ_0 randomly. We sample a batch of N tasks and we perform k inner update for each task:

$$\begin{aligned}
&\theta_0^i = \theta_0 \\
&\theta_1^i = \theta_0^i - \alpha \nabla_\theta \mathcal{L}(\theta_0^i, \mathcal{D}_i^{tr}) \\
&\theta_2^i = \theta_1^i - \alpha \nabla_\theta \mathcal{L}(\theta_1^i, \mathcal{D}_i^{tr}) \\
&\quad ... \\
&\theta_k^i = \theta_{k-1}^i - \alpha \nabla_\theta \mathcal{L}(\theta_{k-1}^i, \mathcal{D}_i^{tr})
\end{aligned}$$

Where $\mathcal{D}_i^{tr}$ corresponds to the trajectories obtained in the task i using the parameters before the considered update. At the end of the last inner update of the last task in the batch, we obtain a sequence of parameters $\{\theta_k^1, \theta_k^2, \theta_k^3, ..., \theta_k^N\}$ for each of the N tasks. These parameters are used to generate new trajectories $\mathcal{D}_i^{val}$ for each task i.

In the outer loop, MAML uses the newly sampled trajectories to update the meta-objective:

$$\theta \leftarrow \theta - \beta g_{MAML},$$

where g_{MAML} is computed in (Wang et al., 2016). Here we take back the same computations and add summation across all copies of task-specific parameters:

$$
\begin{aligned}
g_{MAML} &= \nabla_\theta \sum_{i=1}^{N} \mathcal{L}(\theta_k^i, \mathcal{D}_i^{val}) \\
&= \sum_{i=1}^{N} \nabla_\theta \mathcal{L}(\theta_k^i, \mathcal{D}_i^{val}) \\
&= \sum_{i=1}^{N} \nabla_{\theta_k^i} \mathcal{L}(\theta_k^i, \mathcal{D}_i^{val}).(\nabla_{\theta_{k-1}^i} \theta_k^i)...(\nabla_{\theta_0^i} \theta_1^i).(\nabla_\theta \theta_0^i) \\
&= \sum_{i=1}^{N} \nabla_{\theta_k^i} \mathcal{L}(\theta_k^i, \mathcal{D}_i^{val}).\prod_{j=1}^{k} \nabla_{\theta_{j-1}^i} \theta_j^i \\
&= \sum_{i=1}^{N} \nabla_{\theta_k^i} \mathcal{L}(\theta_k^i, \mathcal{D}_i^{val}).\prod_{j=1}^{k} \nabla_{\theta_{j-1}^i} (\theta_{j-1}^i - \alpha \nabla_\theta \mathcal{L}(\theta_{j-1}^i, \mathcal{D}_i^{tr})) \\
&= \sum_{i=1}^{N} \nabla_{\theta_k^i} \mathcal{L}(\theta_k^i, \mathcal{D}_i^{val}).\prod_{j=1}^{k} (I - \alpha \nabla_{\theta_{j-1}^i} (\nabla_\theta \mathcal{L}(\theta_{j-1}^i, \mathcal{D}_i^{tr}))
\end{aligned}
$$

E.2 MULTITASK LEARNING ALGORITHM GRADIENT

In MTL, there is no distinction between an inner and an outer loop. The model parameters θ_0 are initialized randomly. In contrast to MAML, the gradients do not refer to the initial parameters but to the last updated one. We start by sampling a batch of N tasks. We perform k updates for each task sequentially:

$$
\begin{aligned}
\theta_0^1 &= \theta_0 \\
\theta_1^1 &= \theta_0^1 - \alpha \nabla_{\theta_0^1} \mathcal{L}(\theta_0^1, \mathcal{D}) \\
\theta_2^1 &= \theta_1^1 - \alpha \nabla_{\theta_1^1} \mathcal{L}(\theta_1^1, \mathcal{D}) \\
&... \\
\theta_k^1 &= \theta_{k-1}^1 - \alpha \nabla_{\theta_{k-1}^1} \mathcal{L}(\theta_{k-1}^1, \mathcal{D}) \\
\theta_0^2 &= \theta_k^1 - \alpha \nabla_{\theta_k^1} \mathcal{L}(\theta_k^1, \mathcal{D}) \\
&... \\
&... \\
\theta_{k-1}^N &= \theta_{k-2}^N - \alpha \nabla_{\theta_{k-2}^N} \mathcal{L}(\theta_{k-2}^N, \mathcal{D}),
\end{aligned}
$$

where $\mathcal{D}$ is a shared set of trajectories between all tasks. Note that sharing the same buffer for different tasks comes with some constraints: tasks must share a common MDP structure, onlmy the reward is task-specific. This is the case of the FETCH robotics environments.

To get a mathematical expression of the MTL gradient at the end of an epoch, which we note g_{MTL}, we use the recursive relation found above. For simplicity of notations, we note the sequence $(\theta_0^1, ..., \theta_k^1, \theta_0^2, ...\theta_{k-1}^2, ..., \theta_0^N, ..., \theta_{k-1}^N)$ = $(\psi_0, ..., \psi_{Nk})$:

$$
\begin{aligned}
\psi_{Nk} &= \psi_{Nk-1} - \alpha \nabla_{\psi_{Nk-1}} \mathcal{L}(\psi_{Nk-1}, \mathcal{D}) \\
&= \psi_{Nk-2} - \alpha \nabla_{\psi_{Nk-2}} \mathcal{L}(\psi_{Nk-2}, \mathcal{D}) - \alpha \nabla_{\psi_{Nk-1}} \mathcal{L}(\psi_{Nk-1}, \mathcal{D}) \\
&\dots \\
&= \psi_0 - \alpha \sum_{i=0}^{Nk-1} \nabla_{\psi_i} \mathcal{L}(\psi_i, \mathcal{D}) \\
&= \psi_0 - \alpha g_{MTL}.
\end{aligned}
$$

Since $\psi_i = \theta_{i\%k}^{i \div k+1}$, the overall update after on epoch is:

$$
\theta \leftarrow \theta - \alpha g_{MTL}.
$$

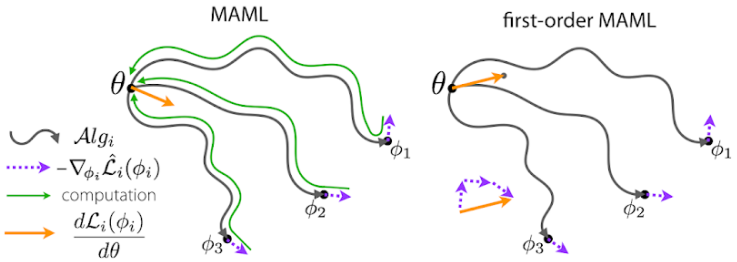


Figure 4: Diagram showing the path taken during the optimization step. $\mathcal{A}lg_i$ refers to the inner updates taken during task labeled i (image taken from Rajeswaran et al. (2019)).

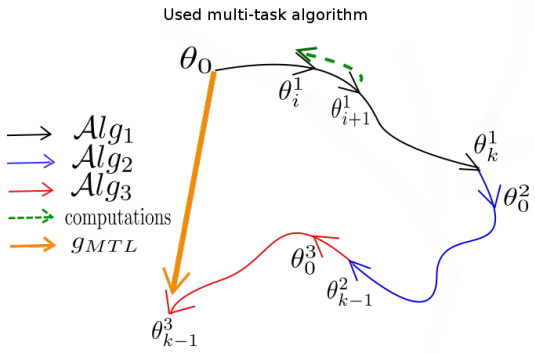


Figure 5: Diagram showing the path taken by the used multi-task algorithm. Here the computation is done at each step contrarily to what is shown in Figure 4, where computation is performed in the meta-update step. $\mathcal{A}lg_i$ refers to the updates taken during task labeled i.

Even though optimization-based MRL and MTL may seem very related, the gradients they compute are a lot different. The main differences are:

- During one single epoch, MTL tends to update the model parameters during each gradient step, leading to more overall updates than optimization-based MRL. The latter tends to create copies of the meta-parameters, update each copy according to each task and then back-propagate through the inner optimization step to compute the gradients.
- Back-propagating through the inner optimization-step leads to second-order gradients derivative computations. The considered loss functions are differentiable almost everywhere. Computing the second-order derivative may be tricky. In fact, back-propagating through the inner-updates may lead to unstable gradients.

$$\boxed{\begin{aligned} g_{MAML} &= \sum_{i=1}^{N} \nabla_{\theta_k^i} \mathcal{L}(\theta_k^i, \mathcal{D}_i^{val}).\prod_{j=1}^{k}(I - \alpha \nabla_{\theta_{j-1}^i}(\nabla_\theta \mathcal{L}(\theta_{j-1}^i, \mathcal{D}_i^{tr})) \\ g_{MTL} &= \sum_{i=0}^{Nk-1} \nabla_{\psi_i} \mathcal{L}(\psi_i, \mathcal{D}) \end{aligned}} \tag{1}$$

F SOFT-ACTOR-CRITIC LOSS FUNCTIONS SPECIFICATIONS

In this appendix, we show show that the critic loss function in SAC satisfies the specifications given in Finn (2018) whereas the actor loss function does not.

The loss functions of the critic and the actor of the SAC algorithm can be written as follows:

$$\begin{aligned} \mathcal{L}_{critic}(y, y_{true}) &= \frac{1}{2}||y - y_{true}||^2 \\ \mathcal{L}_{actor}(y, y_{true}) &= \log y - y_{true} \end{aligned}$$

where y_{true} corresponds in each case to the correspondent value taken as label in the SAC algorithm (Haarnoja et al., 2018b). The gradient with reference to the output are given as follows:

$$\begin{aligned} \nabla_y \mathcal{L}_{critic}(y, 0) &= -y \\ \nabla_y \mathcal{L}_{actor}(y, 0) &= \frac{1}{y}, \end{aligned}$$

We clearly see that the gradient of critic loss function is linear with reference to the output of the last layer when the true value is taken equal to 0 and that the matrix $A = -I$ is invertible. Nevertheless, this not the case for the actor loss function.

G MULTITASK PROBLEM WITH GOAL CONDITIONED POLICIES AND SPARSE REWARDS

In this appendix, we propose to show that multigoal learning within one single task doesn't correspond to multitask learning when using GCPs. In other words, GCPs make multigoal learning not suitable for multitask framework. This explains why MRL doesn't bring too much when compared to MTL in the context of GCPs. Note that here we use the definition of task given in Finn (2018).

Theorem 1 Let $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_N\}$ be a set of reaching tasks i.e tasks that involve reaching particular goals and where:

$$\begin{aligned} \mathcal{T}_i &= \{p_i(s_0), p_i(s_{t+1}|s_t, a_t), \mathcal{L}_i\} \\ &= \{p(s_0), p(s_{t+1}|s_t, a_t), \mathcal{L}_i\} \end{aligned}$$

If the following constraints are satisfied:

- The Markov Decision Processes corresponding to each task i share the same state space $\mathcal{S}$, action space $\mathcal{A}$ and discount factor γ.
- $\mathcal{L}_i = -\mathbb{E}_{s_t, a_t}[\sum_{t=0}^{H} \gamma^t r_t^i(s_t, a_t)|s_0]$
- $\mathcal{S} = \mathcal{O} \times \mathcal{G}$ where $\mathcal{O}$ the set of observations and $\mathcal{G}$ the set of goals such that $\mathcal{G} \subset \mathcal{O}$

Then $\mathcal{T}_i = \mathcal{T}_j \; \forall (i, j) \in [1, ..., N]^2$

Demonstration For each task i, we denote g^i the goal position to be reached.

$\forall t < H$, $s_t = (o_t, g^i) = (s_t^o, s_t^g) = (s_t^o, s^g)$ where $o_t \in \mathcal{O}$ and $g^i \in \mathcal{G}$.

Let $r_t^i : \mathcal{O} \times \mathcal{G} \times \mathcal{A} \to \mathbb{R}$ denote the reward function specific the task i that takes a state and an action taken and outputs the reward received from the environment. Then we have:

$$r_t^i(s_t, a_t) = \mathbb{E}[f(s_{t+1})|s_t, a_t] \qquad \forall s_t \in \mathcal{S}, a_t, \in \mathcal{A}$$

where $f : \mathcal{O} \times \mathcal{G} \to \mathbb{R}$ denotes any function that evaluates how close the new position reached from s_t when taking action a_t is to the goal (example euclidean distance, binary 0/1, binary -1/0 ...)

$$\begin{aligned}
r_t^i(s_t, a_t) &= \mathbb{E}[f(s_{t+1})|s_t, a_t] \\
&= \int_{x \in \mathcal{S}} f(x) p(x|s_t, a_t) dx \\
&= \int_{o \in \mathcal{O}} \int_{g \in \mathcal{G}} f((o, g)) p((o, g)|s_t, a_t) do dg \qquad \text{(variable change)} \\
&= \int_{o \in \mathcal{O}} f((o, g^i)) p((o, g^i)|s_t, a_t) do \qquad \text{(the goal is fixed)} \\
&= \int_{o \in \mathcal{O}} f((o, s^g)) p((o, s^g)|s_t, a_t) do
\end{aligned}$$

Since the reward function does not depend on the task i anymore, then:

$$r_t^i(s_t, a_t) = r_t^j(s_t, a_t) \qquad \forall i, j \in [1, ..., N]$$

As a consequence, $\mathcal{L}_i = \mathcal{L}_j \ \forall i, j \in [1, ..., N]$, which leads to the wanted result.

Hence, when using GCPs and single task multigoal learning, MRL is actually receiving one task during training and needs to fine-tune on that same task, even though we decompose the space into training and testing zone.