Semi Unsupervised Meta Reinforcement Learning

March 1, 2021

A thesis presented for the master of
Artificial Intelligence

Student: Spyridon Krantas
`s.krantas@students.uu.nl`

First Supervisor: Thijs van Ommen
`m.vanommen1@uu.nl`

Second Supervisor: Georg Krempl
`g.m.krempl@uu.nl`

Contents

1 Introduction

In the past twenty years, Artificial Intelligence (AI) has become an essential part of our lives. We are witnessing the birth of Society 5.0 [18, 63], in which people have to keep up with technological advances. Vehicles, education, games, security and many more have already been merged with the cyberspace for either scientific or practical reasons, resulting in a new era of production and communication. Although machines progressively become more intelligent, humans must ensure that the symbiosis of the two species will be advantageous for both. The field of Machine Learning is gaining popularity in every aspect of society and has thus become a hot topic for scientists.

Machine learning focuses on creating programs that can use or create data to automatically learn and improve. The main purpose of these programs is to execute tasks without the help of human engineering; that is the algorithm created by humans must be designed in such a way that can deal with problems by itself or even discover new approaches when it comes to finding solutions. There are currently three main approaches for machines to learn: *Supervised Learning*, *Unsupervised Learning* and *Reinforcement Learning.*

Supervised Learning uses labeled datasets to infer a function/algorithm that can predict and classify given unlabeled data. Unsupervised learning, on the other hand, is a method in which the data provided to the algorithms is unlabeled. Its goal is to detect patterns in the given data without human supervision and let the program label them by itself. Reinforcement learning is concerned with how programs, which we usually call *agents*, can act and behave in a simulated environment. The role of the agents is to maximize a stated reward, which refers to finding an optimal way to reach their objective.

The baseline of this research will be the latter: the framework of Reinforcement Learning. We will use certain techniques to extend it in order to make it more flexible to new and unknown situations. We believe that by developing strong reinforcement learning algorithms, we could gradually apply them not only to simulated environments, but also to real-life problem solving.

1.1 Overview

Reinforcement Learning (RL) has been gaining popularity over the past years. From its inception in the early 1980's until today, the field of RL has been growing incrementally, transforming this area of Machine Learning into a powerful tool. With the combination of RL and non-linear function approximators, scientists have had huge successes in a variety of tasks, such as games [74], continuous control [47], locomotion skills [26], navigating in complex environments [89], and even robotics [37].

Basic RL is modeled as a Markov Decision Process (MDP) with four elements: a set of states, a set of actions, a probability matrix and a reward matrix. By mimicking human behaviour, agents in RL receive extrinsic (the difference with intrinsic rewards will be specified later on) "rewards" or "punishments" for each action(s) they make, indicating the way they should learn. One could describe this process as a form of "trial and error", the most popular way of problem solving among humans; and perhaps this is the optimal way to make machines learn, at least for now. But is it mandatory to use extrinsic rewards and punishments in order to make someone or something learn? Is basic RL or Deep RL (RL with Neural Networks) enough to solve problems that mankind is struggling with?

In order to enhance the framework of RL, we need to identify the main problems or weaknesses that we are dealing with when trying to construct an RL algorithm. While humans can master new tasks quite easily by using their prior knowledge, RL agents find it quite difficult and time-consuming. Massive action or state environments (curse of dimensionality), sub-optimal sampling efficiency and inadequate knowledge representation are some of the main issues that researchers are trying to solve, in order to make RL more generally applicable. We will mostly focus on improving the sample efficiency of our data and alongside provide our agent the knowledge it needs to be able to adapt to tasks it has never come across before.

Training an agent to learn and master specific tasks is becoming increasingly feasible. With different kinds of machine learning, the world is moving towards a brighter future with a thirst for new ideas and innovations. However, in order to move one step further in the field of AI, we should probably stop giving the machines specific instructions about everything, otherwise they won't become more intelligent than just finding patterns from raw or modified data. They have to gain the ability to learn by themselves or to be more precise; they have to learn how to learn. This is known as lifelong learning or, officially, Meta-Learning (ML) [65]. This is going to be covered in more detail in section 2.

ML is an old idea that nowadays is gaining momentum. As it is right now, it has limited potential, but in the near future it can be transformed into a powerful tool. The main goal of ML is to apply machine learning algorithms not on current data, but on the metadata collected while running a simulation—specifically for RL. By doing that, not only do we show the agent how to deal with a certain situation or solve a problem, but we also make sure the agent is able to adapt to new environments more easily. This is also one of the most promising approaches we can take in order to help the process of Transfer Learning (TL), i.e. the problem of storing knowledge and using it in different and unknown, but related, tasks.

1.2 Philosophical view

In the last couple of years, people start to realize that AI has the potential to affect not only our lives, but in general the evolution of both the human species and the world as a whole. According to cosmology, the age of the universe is estimated to be 13.8 billion years old [2]. Since the Big Bang, galaxies were created, planets were formed and a variety of species were born. If we attempted to find one common element between the entities of this world, one could say that everything needs change or, to put it better, the need to evolve. A characteristic case in point for this speculation is the human species, as we have much information regarding the changes that have happened during the last few thousand years; but we can also spot it in the largest possible perspective, as we are able to notice that the universe is also changing and, more specifically, expanding [57][48]. Unfortunately, in both cases, the goal of these changes is unknown to us, therefore we cannot specify the deeper reasons behind these actions.

Human evolution since the Big Bang has led to the appearance of *AMH*, which stands for *anatomically modern humans* and is our current form. However, studies have shown that we as a species diverged from other mammals many million years before this and have been evolving since. Our present characteristics are a result of the development of certain skills like communication or tool use, but also from interbreeding with other hominins [32]. In fact, Darwinism claims that all the biological organisms gain the ability to compete, survive and reproduce through inherited variations [33]. This simply means that diversity is not a sufficient condition for a species to evolve, since these organisms must also compete for the survival of the fittest. Quoting Charles Darwin:

> "It is not the strongest of the species that survive, nor the most intelligent, but the one most responsive to change"

At the moment, the human race is considered to be the most dominant species on the planet in terms of overall power and control. Nevertheless, we shouldn't consider ourselves as the "finale" of the human evolution or being the perfect species, nor believe that we don't have any room for improvement. The idea of a new species [49], created by the hands of humans and possibly ready to take over, is maybe closer than expected. Although this may sound scary and unrealistic, it may as well be inevitable; there may come a moment when we will have to make a choice, either benefit from it or be against it.

Looking at the bright side, the author's philosophical point of view is that humans could learn a lot from the machines. If we want to evolve further, we need to act fast and steady, but always with extreme caution. According to one of the technological leaders in our era, Elon Musk, in a few years robots will be able to do everything better than us; this means that our physical and intellectual capabilities will be surpassed by those of computers. Hence we need to build a trustful bond that will hopefully lead to a harmonic coexistence. By securing this affiliation, humans can be taught by a species which will have a better understanding of this world.

This thesis aims at getting one step closer to Artificial General Intelligence (AGI) [21], a concept where a machine can learn and understand equally or even better than a human. Creating an artificial type of mind can lead to successful discoveries, valuable innovations, and help us answer some of the most difficult and unanswerable questions that would probably take hundreds of years for us to answer.

1.3 Goal

The primary goal of this thesis is the creation of an algorithm or an overall process to automate an end-to-end learning process by combining ML and RL techniques for fast and efficient adaptation to new and unknown tasks. In addition, we will try to achieve that by using an unsupervised learning approach, because we believe that it could give more freedom and better generalization to our agents. Our evaluation metric will be the average return of our algorithm during the ML process when provided with both pretrained tasks and random tasks.

Without diverting to another subject, what we mean by giving more freedom is that we want our agent to meta-learn in a way that it will have as few constraints as possible. The reason behind our thinking is that humans may know a lot of ways of solving a given problem, but these ways are certainly not optimal. By letting an agent act in a way that it finds most efficient, could open a path for new methods and techniques.

ML is an exciting idea that could help accelerate general learning and holds a big promise for the evolution of the machines. AI field has to escape from its narrow objectives [34, 27, 31] and move towards a wider road; ML could be one of these starting points. If agents are able to build their own learning algorithms and experiences when they are dealing with a task and use it later on unknown ones, they will get one step closer to human intelligence. This brings us to our main research question: is the combination of these two strong enough to give an agent the opportunity to freely lifelong learn? If this does work, how effective can it be and to what extent?

1.4 Outline

This thesis will be divided in 4 sections (excluding the introduction). Each section and subsection will build up for the next so we can create a comprehensible hierarchy. We will start from the basics of each framework, discuss some of the existing methods, present our algorithm and the results, and finally conclude.

Section 2 will be dedicated to literature review. We will give an extended introduction on RL and its main characteristics. We will also going to talk about ML, where we will present some of the popular approaches, introduce a new framework called *Unsupervised Reinforcement Learning (URL)* and briefly discuss about *Multi-Task Learning*. Finally, we will show the related work that has been so far and point out the papers that we are interested on, along with their advantages and disadvantages.

In section 3, we will present the algorithm. We will show the theory behind our method and how we managed to obtain specific parts of prior work and merge them into one algorithm/process. We will focus on the details of this merge and why this approach could lead to better results. Finally, we will show the structure of the algorithm, mention some of the benefits, but also some of its limitations.

Section 4 will be the section in which we will present all our results. There will be plots showing the performance of the algorithm, either compared to a variety of parameters or to a different procedure. Furthermore, we will discuss about the significance of these results and what they mean for future related efforts like ours.

Lastly, section 5 will feature the conclusion of our research and a small recap. We will talk about the future work that could be done in order to make the algorithm stronger and further applicable.

2 Literature Review

Machine learning is a field that keeps growing incrementally. Data is considered to be the new currency and every mechanism that surrounds it will be a fundamental tool for the next generation's technology. There are already many scientists that have shifted their interest to the field of AI, making it one of the primary choices for research. Many books and papers have already been published in the last twenty years and the number will only be increasing due to the fact that we are still in the early steps of a new advanced technological era.

We want our research to make a small contribution to this leap of technology. As our base, we will use the framework of RL, around which we will develop our ideas. This section will be used for literature review and to provide the readers with a solid theoretical perspective on the forthcoming practical application. More specific, in subsections 2.1 and 2.2 we will present the frameworks of RL and ML, respectively. In subsection 2.3 we will talk about *Unsupervised Reinforcement Learning*, a new domain that has been developed in the last few years, and how it will help us perform RL without extrinsic rewards, while in subsection 2.4 we will give a brief explanation on *Multi-Task Learning*. Finally, in subsection 2.5 we will discuss about the related work and introduce the main papers that we used as building blocks for this research.

2.1 Reinforcement Learning

Reinforcement Learning is an area of machine learning inspired by behaviorist psychology; we will try to give an introduction of RL based on the famous book of Richard S. Sutton [80] and the slides of one of the best AI researchers, David Silver [73]. The goal of an agent is to select the best possible actions and maximize its cumulative reward. A *reward* is a scalar feedback signal which indicates how well an agent is doing at a specific time step t. The *return* is the cumulative reward until the end of the simulation (when we reach a terminal state), and *value* is the true (unknown) quantity of the future reward, e.g. the expected return given a state. These can be defined as:

Return: $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$

Value (or state value function): $v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$

In this subsection we will talk about the basic RL, so we hypothesize that all our environments are modelled as MDPs. An MDP is a Markov reward process with decisions. As mentioned above, it consists of five elements or more formally a tuple of $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$, where:

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a state transition probability matrix,

$$P^a_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$$

- $\mathcal{R}$ is a reward function,

$$R^a_s = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$$

- γ is a discount factor, $\gamma \in [0, 1]$

where a is the action that our agent executes.

The role of γ is to measure how far in the future we want or need our algorithm to look. By placing a discount factor, we avoid infinite returns and predictions that can not be represented. If we give γ a value that is close to zero, then we are waiting for results that concern the near future, leading to "myopic" evaluation, while with a γ close to one, we get results concerning the distant future. Usually, we prefer the second case for far-sighted evaluation (financial rewards may differ because of the interest of immediate rewards).

An RL agent may include one or more of the following components: a) *Policy*: the agent's behaviour function, which is basically a map from state to action, b) *Value function*: how good is each state and/or action, c) *Model*: the agent's representation of the environment. In order for the agent to achieve its goal, it needs to interact with the current environment, discover a good policy and keep optimizing it. In every state, the agent must make a choice of what action it should take; either following a policy or not. This is where one of the biggest dilemmas of RL starts. It would make sense that the agent always choose the action that can return the optimal reward. But because we are dealing with a sequence of actions, it is not guaranteed that the reward of the first action would be the global optimal reward at the end of this path. For instance, if you follow a suboptimal action at the early run, you could have a greater cumulative reward at the end—a greater return. But where does the dilemma lie?

When an agent interacts with the environment, its main goal is to figure out the optimal path that could lead to maximizing the cumulative reward, by exploring a variety of paths. In order to do that, it needs to sometimes take a different route than the one it has already exploited. But while the agent is doing that, it automatically keeps losing rewards along the way to its destination. This means that the agent is choosing to give up rewards it already knows about in exchange for data/information that may later result in trajectories with a better cumulative reward. In a nutshell, the goal of an agent is to find the best possible path-policy for achieving the task's goal, without losing too much reward by exploring. This dilemma is called the *exploration - exploitation dilemma.* When focusing on exploration, we can find more information about the environment and discover new states and new paths, while with exploitation we exploit the information we have already discovered in order to maximize the return. Despite the fact that in small problems exploration doesn't seem so important, for large and more complex environments, we need it as much as we need exploitation. This is the reason why there needs to be a balance between these two processes.

We have already discussed that an agent must find a really good behaviour/policy to achieve its goal, and now is the time to define it. A policy π is a distribution over actions given the states:

$$\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$$

This is the likelihood of every action when an agent is in a particular state. An algorithm can be categorized according to the update of the policy. We have *On-Policy* and *Off-Policy* algorithms; the difference is that in On-Policy RL, each time the policy is changed, we need to generate new samples and then continue with our algorithm. On the other side, in Off-Policy RL, we are able to improve the current policy without generating new samples from that policy. We will see examples of both types of algorithms in the following subsections.

So far, we have defined the value function $v_\pi(s)$ which is the expected return when following a policy π. We will now introduce a new notion called *action-value function*, which is the expected return of taking an action when following a policy π. It is defined as:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$

Our goal is to maximize this formula over all policies to find our optimal policy. We can symbolize the optimality with a star ($*$). The optimal state-value function and the optimal action-value function are defined respectively as follows:

$$v_*(s) = \max_\pi v_\pi(s)$$

$$q_*(s, a) = \max_\pi q_\pi(s, a)$$

There is always a deterministic optimal policy for any MDP; thus if we know $q_*(s, a)$, we immediately have the optimal policy.

Until now, value functions could be seen as lookup matrices. Although this is enough for small MDPs, for larger MDPs we need to be more flexible because of the amount of states and actions. As mentioned in the introduction, scientists did solve this problem by using *Value Function Approximation.* This refers to the state value function or the action value function that can now be represented by a parameterized function instead of a matrix.

The most famous algorithm for value function approximation is *Stochastic Gradient Descent (SGD).* It uses gradient descent, a method that finds a local minimum of a function and aims at discovering a parameter vector that minimises the loss between a true value function and its approximation. So, SGD belongs to the family of optimization algorithms that uses gradient descent; but it's not the only one. As we have repeatedly indicated above, the goal of RL is to find the best possible strategy that optimizes our rewards. This is the reason why *Policy Gradient Methods* were born.

Policy gradient algorithms search for a local maximum in any policy objective function $J(\theta)$ by ascending the gradient of the policy. Formally the policy gradient is:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q^{\pi_\theta}(s,a)]$$

where $\nabla_\theta J(\theta)$ is the gradient of $J(\theta)$ and $\pi_\theta(a,s)$ is the policy that is modeled with a parameterized function with respect to θ. There are many famous policy gradient algorithms, like Actor-Critic [39], TRPO [67], PPO [68], SAC [25] that can be used in our research.

As a final note on RL, all the concepts we have already introduced, use the reward function for their functionality, but we haven't mentioned that the type of rewards can either be intrinsic or extrinsic [62]. Intrinsic rewards are problem-independent and are used to motivate the agent to do something because it is inherently enjoyable, while extrinsic rewards (the ones we have referred to so far) are used for rewarding the agent regarding its interaction with the environment and are more problem-specific [7]. We will later show that researchers have discovered ways to train an agent without using any extrinsic rewards.

2.2 Meta Learning

Although ML doesn't yet have a clear definition, we can think of it as the idea of "learning how to learn" [65]. For any problem that we need to solve, we usually create an algorithm that could handle this task as efficiently as possible. Accordingly, in machine learning tasks, learning algorithms should make good use of the current input and return the optimal solution through a repetition of learning episodes. However, when dealing with slightly different tasks, they fail to adapt. ML is a process in which we update our learning algorithm to make it more flexible for future tasks, like a non-stop optimization of an existing algorithm, or even better "transform" the algorithm in order for it to learn to solve these tasks entirely by itself.

Let's try to formalize the notion of ML [30]. In standard supervised learning, the objective is to use a given dataset $\mathcal{D} = (x_1, y_1), ..., (x_N, y_N)$ and train a model $y = f_\theta(x)$ for our predictions by using the formula:

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \, \mathcal{L}(\mathcal{D}; \theta, \omega)$$

where $\mathcal{L}$ is the loss function (measures the difference between the true labels and our predictions) and ω a pre-specified condition that indicates the dependence of the solution on assumptions about how the algorithm can learn (e.g. what optimizer we will choose for θ). In contrast, ML assumes that ω is not fixed; because ω can have a big impact on performance measures, like prediction accuracy or data efficiency, ML tries to improve these measures by learning ω itself.

For ML to be successful, we can infer that we need a number of tasks; ML can work with one, but the performance will be poor. In research, it is common to train and test a ML algorithm over a distribution of tasks, a process often referred to as *Multi-Task Learning.* We will discuss more about it in subsection 2.4.

2.2.1 Approaches

Despite the fact that there are a lot of methods for ML to be applied, we still don't have an effective classification system for these methods. So far, there are considered to be three main approaches: *Optimization based, Metric based* and *Model based* methods [84].

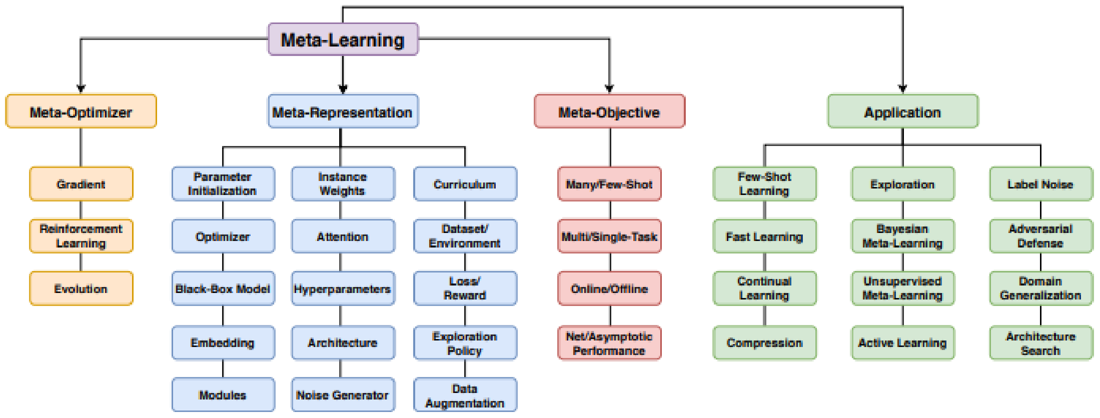


Figure 1: Meta-Learning Breakdown (Reprinted from [30])

Optimization based methods [60, 54] focus on dealing with the given task as an optimization problem and solving it using ω. For example, MAML [14], which is the most famous algorithm in this approach, uses ω as an initialization of the model parameters (for more information on MAML refer to subsection 2.5.1). *Metric based methods* [38, 82, 78, 75] is a simple idea in which we are trying to learn a metric function over objects. We measure the similarities between two data samples, generate a weight with the help of a kernel function and use that weight to predict probabilities over a set of known labels. As stated by its name, *Model based methods* [64, 53] use a model designed specifically for learning, by updating its parameters either with the help of another model or with its own structure (e.g. architecture of Neural Network). Although this method is considered to be robust, it faces difficulties when having to generalize in out-of-distribution tasks.

The authors of [30] introduce a new way of looking at the current approaches of ML. Instead of focusing on classifying the methods based on technical details, they divide these methods on the question each of them is supposed to answer. The approaches are once again three: *Meta-representation* which answers the question of "what" will be the representation of our ω, *Meta-optimizer* which tells us "how" can we optimize ω and finally *Meta-objective* which answers "why" are we following the current process of ML. Figure 1 breaks down the main approaches and shows where they can be applied.

2.2.2 A closer look at Meta Reinforcement Learning

In order to train an RL agent, we need to create data by interacting with the environment. This is how the agent observes, learns and can later master a new task. However, RL and even more meta-RL deals with a number of difficulties; several algorithms have been created for the purpose of solving all these problems that RL is facing. For example, a popular problem in Meta-RL is the optimization problem. Optimizing an agent's policy in Meta-RL is not so easy and often leads to confusing results. The issue is that when an agent is interacting with the environments that contain sparse rewards, the sample efficiency and the overall performance of the algorithm could instantly drop. Most of the meta-RL algorithms are created in order to optimize the exploration of the agent along with the sample efficiency of these algorithms.

When we want to use RL in order to solve a problem, before we even start creating our algorithms and expand our ideas, we need to make up our minds on what path we will follow, On-Policy or Off-Policy RL. This dilemma is also true with Meta-RL. While Off-Policy methods are considered to be more sample efficient in RL, it is really difficult to apply them in Meta-RL [59] [12]. On the contrary, On-Policy methods can be handled more easily in the framework of Meta-RL, but with the disadvantage of losing sample efficiency. In our work, we will examine algorithms from both perspectives.

In general, RL is struggling with exploration. As has been mentioned before, the dilemma of exploration-exploitation is an issue in RL, as well as in Meta-RL. Thus, researchers shifted the problem of solving a specific task to discovering better ways of exploring the state space [24, 76]. By improving exploration, we also improve sample efficiency, resulting in less time-consuming solutions. A new trend has appeared in the last few years that is becoming more and more popular; the notion of *Intrinsic Motivation*, which is tightly connected to the intrinsic rewards we already talked about in subsection 2.1. There will be more details in the next subsection.

2.3 Unsupervised Reinforcement Learning

The combination of RL with Unsupervised Learning leads to a new framework called *Unsupervised Reinforcement Learning (URL)*, a term that has not been officially defined. URL is the process of combining RL with an objective function that currently stems from the field of *Information Theory* [42], in order to train an RL agent without defining any extrinsic rewards. This leads to a training process with the single purpose of freeing ourselves from the continuous supervision of the agent.

Information theory studies the transmission of information between communication systems. Its quantitative measure is *entropy*, a notion developed by Claude Shannon [70], which measures the level of information (uncertainty) of a random process. On top of entropy, *Mutual Information (MI)* was created. MI quantifies the amount of information that a random variable could tell us about another random variable. The formulas of both the quantities can be seen below:

$$\textit{Entropy: } H(X) = \textstyle\sum_{i=1}^{n} P(x_i) \log_b P(x_i)$$

$$\textit{Mutual Information: } I(X;Y) = D_{KL}(P_{(X,Y)} || P_x \otimes P_y)$$

where KL is the Kullback–Leibler divergence [41, 40]. With the help of these entities, we are able to develop maximum entropy algorithms.

The principle of maximum entropy claims that in a variety of probability distributions, the one that represents the current state of knowledge the best, is the one with the largest entropy [58]. The idea behind this principle is that by taking the largest entropy, we are making less assumptions about the current state of knowledge and ensure a non-biased analysis. Combined with the RL framework, powerful techniques for better sampling and exploration were created [90, 25].

Recently, an improved version of this approach "manufactured" URL. As we know, extrinsic rewards are essential for the RL process and creating them by hand can sometimes be troublesome. A way to surpass this obstacle is to use MI in order to define intrinsic rewards, which could motivate the agent to explore the state space or discover a variety of options/skills [22, 52, 1, 11, 72].

The idea of *Intrinsic Motivation* originated in developmental psychology [56, 55, 66]. Similar to a person who is curious about the world and eager to learn and explore everything, an RL agent could act likewise and develop behaviours and skills that could benefit its existence without human supervision [77, 45, 5, 7, 6].

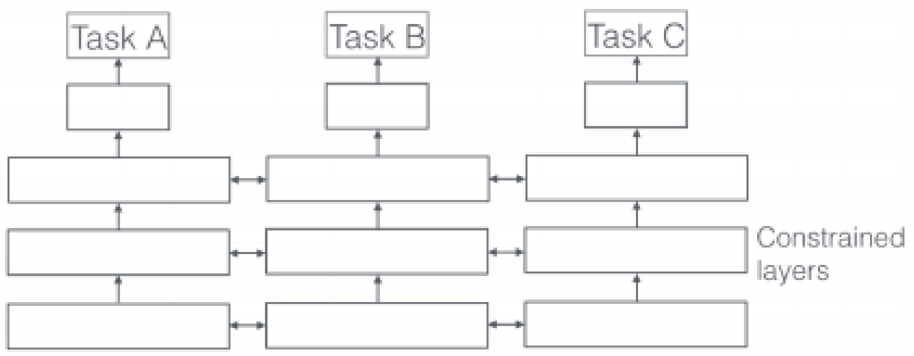


Figure 2: Soft parameter sharing for multi-task learning in deep neural networks (Reprinted from [61])

Using this new approach of URL along with ML, a new framework was created, namely *Unsupervised Meta-Reinforcement Learning*[23]. This will be the main theoretical basis of our thesis.

2.4 Multi-Task Learning

Performing two tasks or more at the same time (multi-tasking) is easy and natural for people, but not so easy for machines. In most cases, programs are created to solve a single task while programmers are waiting for a single output-solution at the end of the run. In the machine learning field, *Multi-Task Learning (MTL)* refers to things that the agent has to learn simultaneously, but not act on. It is an essential domain that can be used to learn tasks faster than learning them independently but also as a solution to problems with insufficient data [13]. For example, if we do not have much information about how to solve a task, we could use data from related tasks and get closer to the solution. One can say that the concept behind MTL is similar to TL. Their main difference is that in TL we are using a number of tasks to improve the performance in a single target-task, while in MTL we are trying to optimize the performance in all tasks [88]. Before we use this connection to develop our algorithm (section 3), we need to expand our knowledge on the current methods of MTL. There are many algorithms with different approaches on how to multi-task, however we will only introduce the two main methods that are being used in combination with neural networks [61].

2.4.1 Soft parameter sharing

In the soft parameter sharing method, each task we are trying to solve has its own model and parameters. The distance between the parameters of the models is regularized and the layers are constrained in order for the parameters to become as similar as possible and represent all the tasks. The architecture can be seen in (Figure 2).

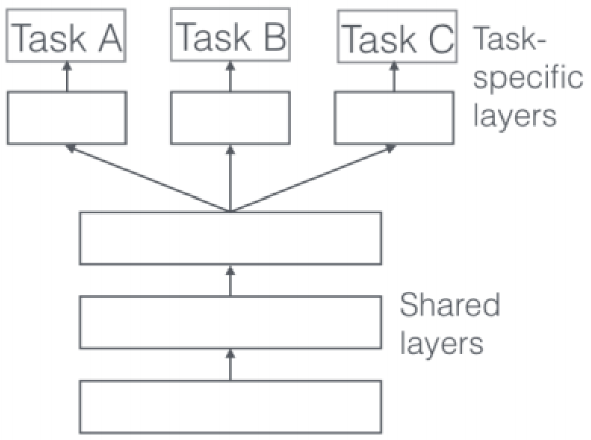


Figure 3: Hard parameter sharing for multi-task learning in deep neural networks (Reprinted from [61])

2.4.2 Hard parameter sharing

Hard parameter sharing is a technique of sharing the hidden layers of the neural network between all tasks, but keeping the task-specific output layers. This means that we are trying to find some common characteristics between our problems that can lead us to a solid base for solving any given related future task. By creating this base (Neural Network), we are gathering different data from various tasks which gives us the opportunity to reduce the risk of overfitting. The architecture can be seen in (Figure 3).

2.5 Related Work

As we mentioned above, the framework of unsupervised RL is "under development", without clear definitions or solid methodologies. On the other hand, the field of ML is on a continuous rise, transforming this framework into a future promising trend.

Although the idea of ML is old, it was only recently that researchers paid more attention to it. The method of meta learning with gradient descent was introduced in 2001 [29, 87], where the researchers used recurrent neural networks and obtained great results; two years later, [69] focused explicitly on applying ML to the RL framework in order to fine-tune its meta-parameters. With the breakthrough of *Deep Learning*, the authors of [83] showed the way for deep meta-reinforcement learning. More recent approaches in ML focus on designing optimization algorithms [4, 46, 44], creating neural networks with various different architectures [64, 50, 79], using one-shot learning [9, 16] or optimizing exploration [86, 76, 20].

In this subsection, we will briefly introduce specific algorithms that we either used in our research or got inspired by. We will also describe the intuition behind their functionalities and, finally, explain their technical details.

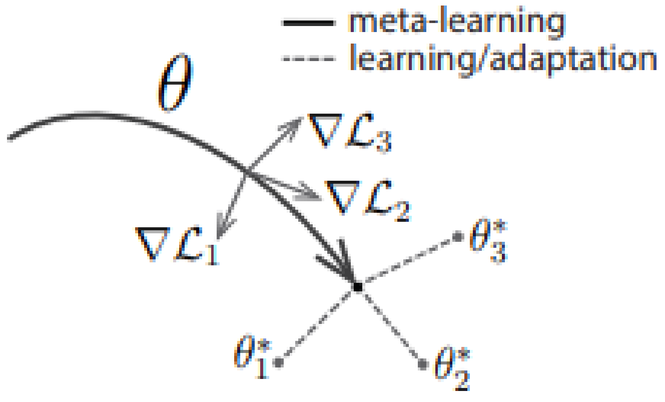


Figure 4: MAML (Reprinted from [14])

2.5.1 Model Agnostic Meta Learning - MAML

MAML [14] is a model-agnostic ML algorithm which is based on learning the parameters of any model that uses gradient descent. Its goal is simple: fast adaptation to new tasks with a small number of gradient steps. The structure of MAML was mentioned in subsection 2.2 and here it will be discussed in more detail.

We have a distribution over related tasks $p_{(T)}$ and a model f parameterized with a random variable θ. We sample a number of tasks from the distribution—using 3 tasks for simplicity—and start training the model. So, we have our tasks $T = (T_1, T_2, T_3)$, our initial θ that we want to optimize later and we are ready to compute the loss function $\mathcal{L}_{T_i}(f_\theta)$, use gradient descent to minimize it and repeat this process until we find the optimal set of parameters. We can see how our θ can be updated below:

$$\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{T_i}(f_\theta)$$

where α could be fixed as a hyperparameter or meta-learned, and $\nabla_\theta \mathcal{L}_{T_i}(f_\theta)$ is the gradient of task T_i. By repeating the above steps for all three of our sampled tasks and randomizing every time our initial θ, the output will be new optimal parameters $\theta^* = (\theta_1^*, \theta_2^*, \theta_3^*)$. Before we continue to sample the next batch of tasks from the distribution, we must perform the *meta-update*; meaning that we are looking for a θ that is optimal and common for all three tasks, so we can start with a better θ in the new batch instead of randomizing it (Figure 4). In this way, the new tasks would need less data and less gradient steps resulting in faster learning and adaptation.

This new optimal common θ can be found with the help of SGD, such that it can be updated with this formula:

$$\theta^* = \theta - \beta \nabla_\theta \sum_{T_i \sim p(T)} \mathcal{L}_{T_i}(f_{\theta_i'})$$

where β is the meta-step size and $\nabla_\theta \sum_{T_i \sim p(T)} \mathcal{L}_{T_i}(f_{\theta_i'})$ the gradient for each new task T_i.

Now, we can connect the above process with RL. In RL one task is an MDP with horizon H. This means that f_θ is a policy that maps states x_t to a distribution over actions a_t, with $t \in \{1, 2, ..., H\}$. Therefore, we can estimate the gradient for the model gradient update and the meta-optimization by using policy gradient methods. The complete process for applying MAML in a RL framework is: a) Sample a batch of tasks T_i, b) Sample K trajectories using an initial θ, c) Evaluate the gradient of the loss, d) Compute the optimal θ_i^*, e) Sample new trajectories using this new θ_i^*, f) Update the new optimal-common θ^* with the previous formula.

The authors of [14], with the help of the *REINFORCE* algorithm [85] for computing the gradient updates and the *TRPO* algorithm [67] as the meta-optimizer, obtained state-of-the-art results in both navigation and locomotion tasks.

2.5.2 Diversity Is All You Need - DIAYN

DIAYN [11] is a method of learning skills in the absence of any rewards. A "skill" here is defined as a latent-conditioned policy that can consistently change the state of the environment in a particular way. It is denoted as $p(a|s, z)$, where z represents a latent variable sampled from a distribution $p(z)$. It is assumed that in order to acquire skills that are useful, we have to train the skills so that they cover the spectrum of all the possible behaviours of the agent.

The key idea focuses on the diversity of the skills. Finding skills that have slight differences between them doesn't always mean that they are semantically meaningful. So, the primary target becomes the discovery of skills that are not just distinguishable, but as diverse as possible. This goal is formalized as maximizing an information objective with a maximum entropy policy. This way, the authors are trying to show that a good exploration objective can lead to unsupervised emergence of skills.

DIAYN is based on three ideas. The first idea is that if we want our skills to be useful and distinguishable, we need them to dictate the states that the agent visits so that different skills can visit different states. This can be achieved by maximizing the mutual information between states and skills [22, 17]. The second idea is that we will use states instead of actions to indicate the diversity of the skills, since actions that don't affect the environment cannot be seen by an outside observer. We can do that by minimizing the mutual information of actions and skills, given the states. The final idea is that more exploration is needed for our skills to be as diverse as possible, so we encourage them to also act as randomly as possible. This can be done by maximizing the policy entropy.

Now, we need to transform these three ideas mathematically. By repeating the above statements, we get the following process: a) We maximize the mutual information $I(S; Z)$ between states S and skills Z, b) We minimize the mutual information $I(A; Z|S)$ between actions A and skills Z given the states S, c) We maximize the policy entropy $H(A|S)$. So, formally, we need to maximize:

$$F(\theta) \triangleq I(S; Z) + H[A|S] - I(A; Z|S)$$

which can also be written as:

$$F(\theta) \triangleq H[Z] - H[Z|S] + H[A|S, Z]$$

The first term gives our prior distribution a motive over $p(z)$ to have high entropy, the second term suggests that it should be easy to infer the skill z from the current state and the last term indicates that each skill should act as randomly as possible. We can maximize the latter with

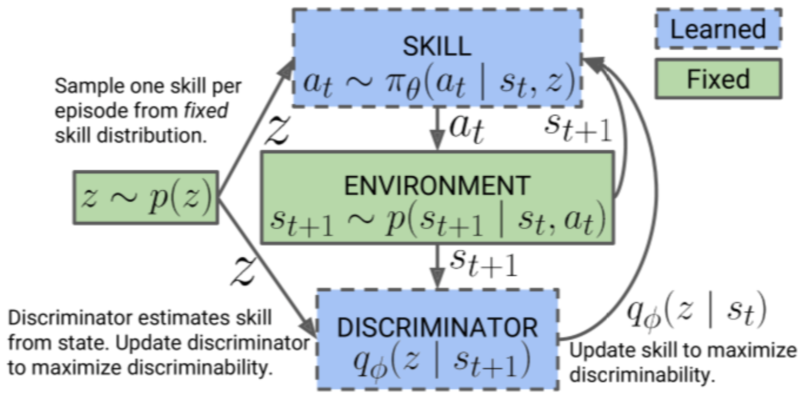

SKILL
$a_t \sim \pi_\theta(a_t \mid s_t, z)$
Learned
Fixed
Sample one skill per episode from *fixed* skill distribution.
z
a_t
s_{t+1}
$z \sim p(z)$
ENVIRONMENT
$s_{t+1} \sim p(s_{t+1} \mid s_t, a_t)$
z
s_{t+1}
$q_\phi(z \mid s_t)$
Discriminator estimates skill from state. Update discriminator to maximize discriminability.
DISCRIMINATOR
$q_\phi(z \mid s_{t+1})$
Update skill to maximize discriminability.

Figure 5: DIAYN algorithm (Reprinted from [11])

existing methods of maximum entropy RL like SAC [25], while for the first two terms the authors suggested to embody them in a pseudo-reward function:

$$r_z(s, a) \triangleq \log q_\phi(z|s) - \log p(z)$$

where a learned discriminator $q_\phi(z|s)$ is used to approximate $p(z|s)$. The above formula basically indicates that an agent is rewarded for visiting states that are easy to distinguish while q_ϕ is updated to deduce the skill z from the states more easily. We can see the structure of DIAYN in Figure 5.

2.5.3 Unsupervised Meta-Learning for Reinforcement Learning

Looking at the above two algorithms, someone may wonder if it is possible to mix them up in order to create something new; this idea was developed by the authors of [23]. Meta-RL algorithms usually sample from a pre-specified task distribution in order to be able to adapt to future given related tasks. But defining a distribution can sometimes prove difficult, especially when dealing with complex problems. Knowing that meta-RL needs a number of training tasks [64, 4, 14], the authors took advantage of the DIAYN algorithm and generated a task proposal distribution as input for MAML.

They also theoretically proved in their paper that when the meta-test task distribution is unknown, the meta-training task distribution is optimal only when it is uniform over the space of goals. The logic behind this is pretty clear; if the test time task distribution can be chosen adversarially, the algorithm must make sure it is uniformly good over all possible tasks that might be encountered.

The process of creating an unsupervised meta-RL technique is straightforward. In a given environment, we can generate tasks using the framework of URL, feed them in a meta-RL algorithm while keeping the environment and its dynamics the same, meta-train the agent on the these tasks and, at the meta-test time, check how well the agent adapts in any given unknown task(or any given reward function). The procedure can be seen in Figure 6. According to this structure the authors used DIAYN for the *Unsupervised Task Acquisition* and MAML for the *Meta-RL* .

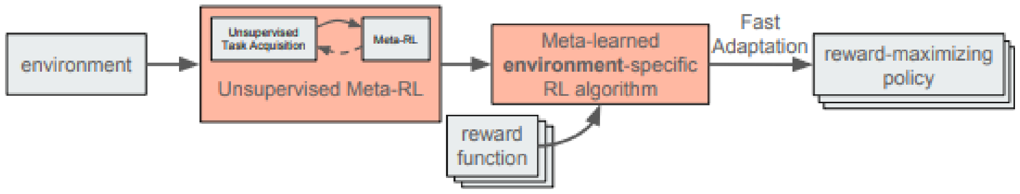


Figure 6: Unsupervised Meta-RL (Reprinted from [23])

The task distribution is defined as a mapping from a latent variable $z \sim p(z)$ to a reward function $r_z(s, a)$, meaning that we have a different reward function r_z for each variable z. We already know that DIAYN is able to train a discriminator that can predict a variable/skill z from the current state, always according to a policy $\pi(a|s, z)$. We can then use the learned discriminator $q_\phi(z|s)$ as the reward for the meta-training procedure in the same way as we are able to use it in imitation learning (we will not expand on that) [43, 28, 15]. Figure 7 shows the equality of a reward function with a classifier—in our case a discriminator network. So, the reward is computed according to this formula:

$$r_z(s, a) = \log q_\phi(z|s)$$

The MAML algorithm will use this reward function to update the model's parameters, resulting in a fast learning algorithm that can quickly adapt to any task.

2.5.4 Dynamics-Aware Unsupervised Discovery of Skills - DADS

DADS[72] is considered to be an improvement of DIAYN. It is a URL algorithm which learns skills that are optimized for predictability and diversity over the state space. Its main difference is that it is designed to learn skills by making the model-based reinforcement learning easier(we won't focus on that part). As in DIAYN, we are trying to "discover" skills without the use of any extrinsic rewards. We learn a skill-conditioned-policy $\pi(a|s, z)$ and we assume that the skills z are sampled from a prior distribution $p(z)$ over Z. We learn a skill-conditioned transition function $q(s'|s, z)$, which we refer to as *skill-dynamics*, that predicts the transition to the next state s' from the current state s for the skill z.

Based on the concepts of information theory, the authors maximize the mutual information between the next state s' and skill z, conditioned on the current state s.

$$I(s'; z|s) = H(s'|s) - H(s'|s, z)$$

If we maximize the above objective, we simultaneously maximize the diversity of transitions $H(s'|s)$ while making z informative about s' by minimizing $H(s'|s, z)$. This ensures that the transitions for a z are predictable and also that the skills are diverse from each other. Similarly with the pseudo-reward in DIAYN, we need to define an intrinsic reward for our algorithm to optimize. After a number of deductions that the authors did, the final formula can be seen below:

$$r_z(s, a) = \log \frac{q_\phi(s'|s, z)}{\sum_{i=1}^{L} q_\phi(s'|s, z_i)} + \log L, z_i \sim p(z)$$

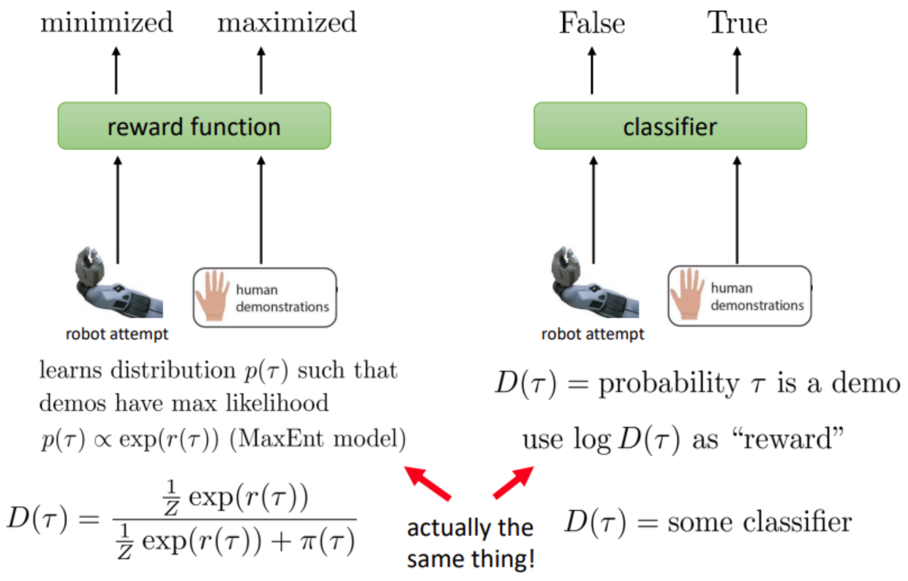

minimized
maximized
reward function
robot attempt
human demonstrations
learns distribution $p(\tau)$ such that demos have max likelihood
$p(\tau) \propto \exp(r(\tau))$ (MaxEnt model)
$D(\tau) = \frac{\frac{1}{Z}\exp(r(\tau))}{\frac{1}{Z}\exp(r(\tau)) + \pi(\tau)}$
actually the same thing!
False
True
classifier
robot attempt
human demonstrations
$D(\tau)$ = probability τ is a demo
use $\log D(\tau)$ as "reward"
$D(\tau)$ = some classifier

Figure 7: Reward function is equal to classifier (Reprinted from [43])

We can notice that the numerator will be high when the transition from the current state to the next state has a high log probability with the current skill z and the denominator will be low when the transition has low probability again under z_i.

In conclusion, the DADS algorithm works as follows: we sample a skill z from our distribution $p(z)$ for every episode, we collect on-policy samples and update our skill-dynamics using gradient descent, we compute the intrinsic reward and finally update our policy using a RL algorithm. Figure 8 displays the whole process.

2.5.5 Emergent Real-World Robotic Skills via Unsupervised Off-Policy Reinforcement Learning

This paper[71] is basically an application of DADS, extended into an off-policy method. The main reason for this extension is the inability of DADS to be sample efficient due to its on-policy sampling. Remember that in the DADS algorithm we were collecting on-policy samples and updating our skill-dynamics using gradient descent. The updated algorithm, called *off-DADS*, uses off-policy learning which enhances sample efficiency and asynchronous data collection through multiple actors. The authors, again after a number of deductions, use the Q-value function and redefine it as:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}[\sum_{i \geq t} \gamma^{i-t} r(s_t, z, s_{t+1})]$$

They instantiate a replay buffer R and use the current policy $p^{(t)}$ to sample transitions and add them to the buffer. A replay buffer is just a large table which we can store raw data, like states-actions-rewards etc, for future use [51]. Then they uniformly sample a batch of transitions from R

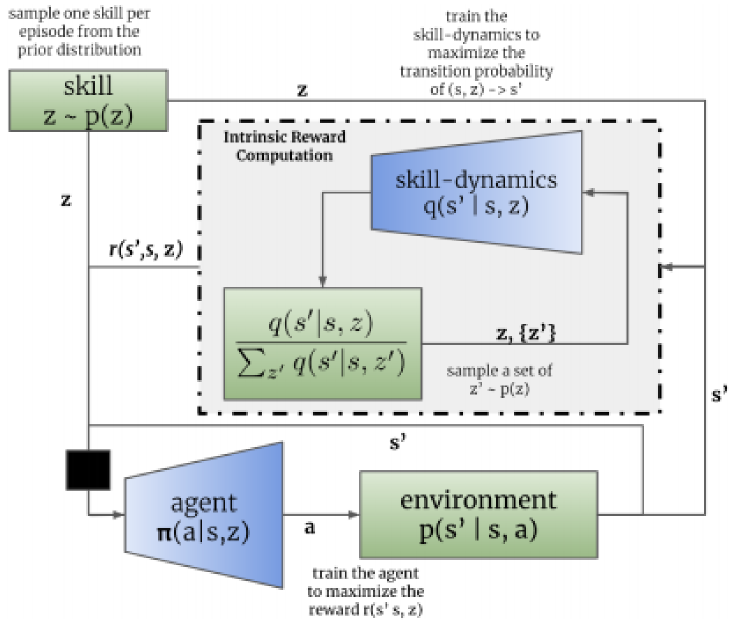

sample one skill per episode from the prior distribution
skill
z ~ p(z)
z
train the skill-dynamics to maximize the transition probability of (s, z) -> s'
Intrinsic Reward Computation
skill-dynamics
q(s' | s, z)
z
r(s',s, z)
$\frac{q(s'|s,z)}{\sum_{z'} q(s'|s,z')}$
z, {z'}
sample a set of z' ~ p(z)
s'
s'
agent
π(a|s,z)
a
environment
p(s' | s, a)
train the agent to maximize the reward r(s' s, z)

Figure 8: DADS algorithm (Reprinted from [72])

$B^{(t)} = \{(s_i, z_i, a_i, s'_i)\}_{i=0}^{B}$ and use them to update $\pi^{(t)}$ and $Q^{\pi^{(t)}}$.

The above process can face two problems: a) the reward is non-stationary and b) learning q_ϕ relies on the current policy π and the stationary distribution $p(s|z)$. The first problem can be solved by recomputing the reward for the current batch using the current q, while for the second problem we can either use samples from the current policy $\pi^{(t)}$ or just reuse off-policy data to learn q_ϕ. For the latter, because the data is sampled from a different distribution, we need to make some corrections in the sampling process. Instead of deriving an unbiased gradient estimator, the authors chose to make use of an estimator that is biased, but simpler and more stable.

Let's recap on the off-DADS algorithm; we use a number of actors in the environment which use the latest copy of the policy to collect data. We keep adding this new data to our buffer until we reach a specified limit in which we will uniformly sample from R to train the q_ϕ. We sample the buffer once more in order to update the policy π and compute the intrinsic reward for all the transitions using the latest q_ϕ. The authors use the *soft-actor critic* [25] algorithm to update π and Q^π, but they explicitly mention that these "labelled transitions" can be passed to any off-policy RL algorithm for the updates.

2.5.6 Efficient Off-Policy Meta-Reinforcement Learning via Probabilistic Context Variables - PEARL

The last algorithm that we will introduce is called PEARL [59]. It is an off-policy ML algorithm that improves the sample efficiency, especially in comparison with on-policy ML algorithms, performs structured exploration and also achieves rapid adaptation on unseen tasks at meta-test time. Similar

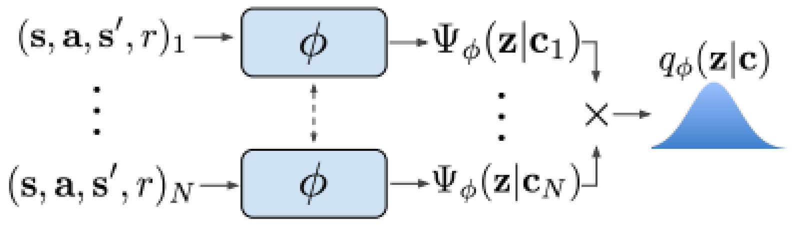

$(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)_1$
$(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)_N$
ϕ
ϕ
$\Psi_\phi(\mathbf{z}|\mathbf{c}_1)$
$\Psi_\phi(\mathbf{z}|\mathbf{c}_N)$
$\times$
$q_\phi(\mathbf{z}|\mathbf{c})$

Figure 9: Inference network (Reprinted from [59])

to the *context-based* meta-RL methods [10, 83]—methods that ensemble experience into latent representations to adapt to new tasks—in PEARL we represent task contexts with probabilistic latent variables in order to capture the uncertainty over the task and help the agent explore in a structured way in a new task.

Given a distribution of tasks $p(T)$, we meta-train an agent to learn a policy that can adapt to a given task by conditioning on the history of past transitions, which we refer to as *context* c. So, $c^T_{1:N}$ contains the experience that is collected for steps N in a task T. We represent our knowledge about the current task in a latent probabilistic context variable Z and we learn to infer it through a variational inference approach [36, 3]. We train an inference network $q_\phi(z|c)$ that estimates the posterior $p(z|c)$. Its architecture can be seen in Figure 9.

In meta-RL, we know that in order for the agent to be able to adapt to the task at meta-test time, it needs on-policy data; so the meta-training phase needs on-policy data too. But as we mentioned earlier, the biggest advantage of PEARL compared to other meta-RL algorithms is that it is an off-policy method. The authors had the idea of separating the data that we use to train the encoder with the data that is used to train the policy. The policy treats the context z as part of the state, while the uncertainty in the encoder $q(z|c)$ provides us with a stochastic exploration. A sampler S_c is defined in such a way so as to sample content batches from a replay buffer, in order to train the encoder. Instead of sampling from the entire buffer, which could possibly cause a distribution discrepancy, we sample only from the recently collected data for better efficiency. Figure 10 displays the whole meta-training procedure.

3 Methodology

The basic idea of our algorithm/process originated from [23]. As explained in subsection 2.5.3, the authors merged two algorithms, one for discovering skills (DIAYN) and one for using these skills as tasks for the agent to be meta-trained on (MAML). What would happen if we use the same process, but with new improved algorithms to substitute the aforementioned two? More importantly, what would happen if we use explicitly off-policy methods?

This section will focus on explaining the overall process of our algorithm called *SUMR - Semi Unsupervised Meta-Reinforcement Learning*. As we mentioned previously, we will follow the idea of [23], and try to build the same unsupervised Meta-RL framework, but more advanced—meaning with up-to-date algorithms. The formula/process will be the same: we have an environment in which we will run our unsupervised learning algorithm and extract some skills/tasks, feed them to our ML algorithm for meta-training, and at meta-test time we provide the agent with the goal

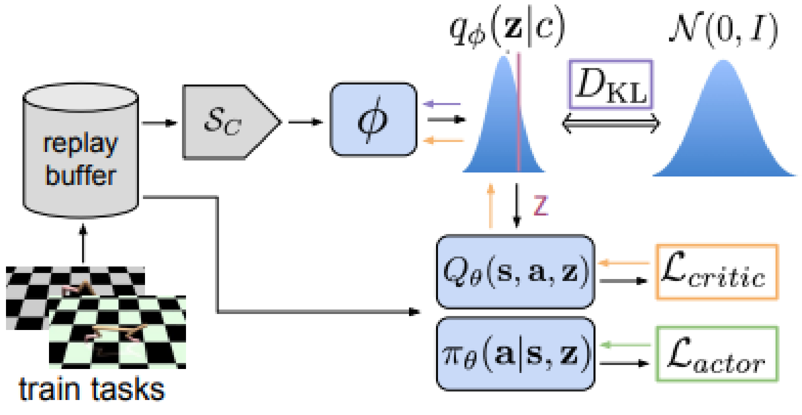


Figure 10: Meta-training process (Reprinted from [59])

(reward function) that it has to achieve in order for us to test if the adaptation to this goal is as fast and effective as we expect it to be.

3.1 First try

For our plan to succeed, we need to replace the reward function at meta-training time with the log probability of the discriminator network; the learning process won't be affected as proved by [28]. In the case of [23], the authors used the learned discriminator network of DIAYN to propose tasks to MAML. In our case, things are a little bit different.

The big difference between DIAYN and DADS is the network created to make the predictions of the skills in every state. While in DIAYN we are using a discriminator network that takes as input the current state and predicts which skill the agent is executing, the DADS algorithm uses a network called skill-dynamics, which takes as inputs the current state and the skill, and predicts the next state. For the latter we don't have any proof that it would have the same functional ability as the discriminator network if we try to place it as the reward function. Despite this fact, we wanted to try both ways; modify the DADS algorithm so we can follow the plan in every detail, by building and training a discriminator network, and if that worked, we would try to do the same with the original DADS (with the skill-dynamics network) and check if the results are close to each other. Unfortunately our plan was difficult to execute.

The DADS repository already contained a script for creating both a discriminator and a skill-dynamics network, but the algorithm was using only the last one. We tried to take advantage of the discriminator network script and straightly (before training it) place it as our reward function, which we were going to use in our ML framework, with random initialization of the weights (i.e. random dynamics). This test has already proved its robustness by [23], as they did the similar experiments with really good results in order to compare it with the pretrained discriminator network. But the implementation was extremely hard for us, leading to a non-productive period—we were working on this for nearly two months—without any success. During this time we came up with an alternative approach in order to achieve unsupervised meta-RL.

3.2 An Alternative Path

Most of the time, when people want to achieve something, they know their goal since the beginning of their task rather than learning it during the process. Based on this, when facing the above problem, we came up with a similar idea that may seem naive, but hadn't been explored yet. The key in our idea is the output at the last epoch of the agent's training. We wondered that if we cannot implement a discriminator network as the reward function, we could "cheat" and set the goal before the beginning of the unsupervised learning process. But, what does this mean? Are we still talking about unsupervised learning and about no extrinsic rewards or by knowing the goal beforehand we are just training an agent to create trajectories with a regular RL process? What will be the goal/reward-function at meta-test time? And would that even work?

If we were able to implement the idea of [23], then the meta-test's reward function wouldn't concern us at all; our agent would be able to train and adapt fast (optimistic view of the future) at any required task in meta-test time. In our case, since we can't achieve the implementation, the plan is to have the knowledge of the goal beforehand and somehow set it as one part of the output in the unsupervised learning framework, always without interfering at all on the DADS functioning. For instance, the main goal of our experiments was to train an ant on reaching a given position of the environment. In order to do that, we had to export the final positions of all the trajectories of the ant's unsupervised learning on the last epoch (the number of trajectories equals the number of skills) to later use them as tasks/goals in our ML framework. With this approach we first ensure that the ant would not take random positions as tasks, like we usually use as goals when using ML algorithms, but also provide the motivation to explore/reach positions that are far away from its initial state, hoping that the ant's trajectories could cover a broad state space of the environment.'For this to happen, we had to implement a reward function that could compute the distance between the agent's current position and the position we exported through the unsupervised process which in the meta-training phase becomes the goal of the agent (this is the reason behind the term "semi"). More specific, the ant receives a reward composed of a control cost, a contact cost, a survival reward and a penalty equal to its L1 distance to the target position, at each time step [81]. The formula can be seen below:

$$reward = goalreward - controlcost - contactcost + survivalreward$$

More details can be found at [8]. Similar data can be exported in the unsupervised learning framework for any goal (e.g. ant's velocity, ant's direction etc) that we want our ant/agent to meta-train on.

The process that was just described can be summed up to a series of the following steps: a) we train an agent with DADS algorithm and discover a number of skills, b) we export an output based on these skills and according to our goal, c) we feed this output as task to a ML algorithm for meta-training, d) we provide the algorithm with the appropriate reward function for reaching our goal and e) finally at meta-test time we test the performance of our algorithm by providing it with random tasks. Figure 11 displays the process of our algorithm.

3.3 The Final Plan

Although our alternative idea seemed pretty solid, the first results were not so convincing. After a number of experiments we encountered a problem that we should probably have expected. In the meta-training phase, our algorithm showed some signs of weakness, as the agent was learning really

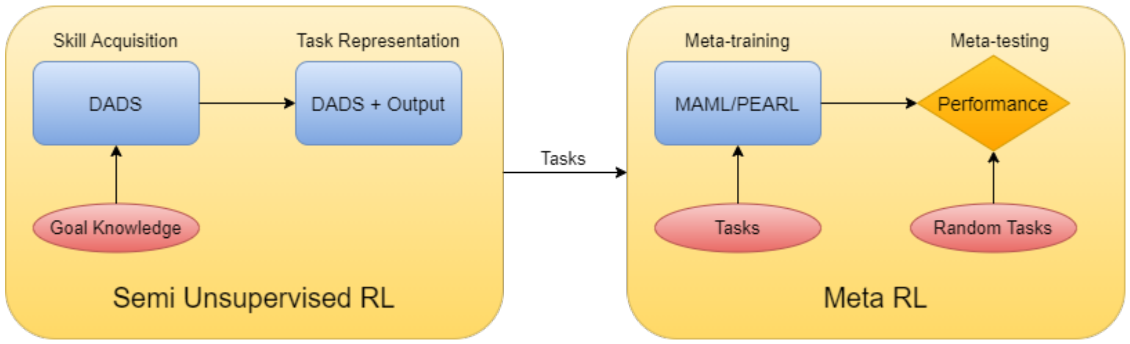


Figure 11: Semi Unsupervised Meta RL - SUMRL

slow and in many cases it couldn't learn at all. However, the problem was not the algorithm itself, but the data that we were feeding into the ML algorithms.

As previously claimed, the key in creating this algorithm was the output we would take on the last epoch of running in the unsupervised learning process. This means that the number of tasks (positions of the ant) that we will feed the ML framework would be the same exact number as the skills we discovered. The problem was that most of the final positions in the final epoch were far away from the initial position, causing the agent to "hunt" coordinates it couldn't possibly reach with zero training at the meta-training phase. For example, imagine that a baby is in the living room, trying to learn how to crawl, stand up or walk. Instead of teaching the baby how to reach the states closer to its initial state space, we showed the baby a destination in the house, like the kitchen or the bathroom, and told it to go there as fast as possible. The baby would first have to learn how to crawl in order to get there. And if we want to be a little more demanding, the baby would need to walk/run to reach the destination with an optimal speed. But, what can we do in our case?

Our solution was to extract the output of the final positions as before, but now not only in the last epoch, but in a number of epochs. Then, by sampling uniformly and at random all the positions, we could gain the benefit of providing goals to the ML algorithms that are covering an impressive amount of the state space, helping the agent to learn fast and steady. Now the question that arises has to do with the number of skills/tasks that we are setting in the unsupervised learning process and the number of epochs we are extracting these tasks from. This will be answered in the following section.

4 Experimentation and Results

In this section we will focus on showing the way we extracted our skill/tasks from the DADS algorithm, the hyperparameters we changed or experimented on, the way we fed these tasks to the off-policy ML algorithm PEARL (we also fed them to the on-policy MAML) and the results we got from the ML algorithms. We will divide our experimentation results into two categories: the unsupervised learning process and the Meta-RL process.

4.1 Unsupervised learning

DADS is an algorithm with many hyperparameters. In section 2, we already introduced two versions of DADS, the on-policy and the off-policy algorithm. Although we are interested in the offDADS algorithm, we will also examine the performance of the on-policy algorithm and compare the two approaches.

Apart from using the two versions of the DADS algorithm, we will also experiment on a number of hyperparameters. Testing all possible scenarios requires a lot of time, which our hardware cannot handle. For instance, we can choose to tune the number of steps in the environments (timesteps), the number of hidden layers in our networks, the type of the state distribution (continuous or discrete) or the batch size etc. From all these entities we chose to inspect four of them (three plus on-policy): the observation space, the number of the skills and the number of epochs, which we think are the ones that would have the largest impact on our results. The approximate real time for all the experiments was 40 days with an average time of 15 hours per day.

We took into consideration that all the values were already fine tuned by the authors of DADS, so we assumed that these are already near-optimal. Aside from these four that we will use later on, the other (most important) settings remained the same during all the experiments. More specific, we trained our agent for 5.000 epochs (approximate real time was 1 day for 1.100 epochs regardless of the set up), we learned in a continuous skill space in which we reduce the observation space only to the x and y axes. We do that to motivate our agent to keep moving away from its initial position and explore more state spaces in the environment. Our neural networks (skill dynamics, policy, critic) consisted of two hidden layers with a size of 512 neurons. We train our policy for 64 steps with a batch size of 256 and we use Adam[35] as our optimizer with a learning rate of $3 \cdot 10^{-4}$. For the off-policy mode we have a replay buffer of size 10.000 from which we sample our batches. Further details of the hyperparameters can be found on [71].

We ran our experiments on an Ubuntu 18.04.4 LTS system with an eight-core AMD FX-8320 Processor and a 8 GiB RAM. The simulation environment was specifically the Mujoco's Ant environment[81]. The ant's observation space is 111 dimensions (joint angles, orientation of the torso, directional and angular velocities etc) and it can move (action space) in 8 dimensions. Figure 12 shows a snapshot of the ant's environment during one of the tests.

4.1.1 Off-policy vs On-policy

First, we wanted to test the performance of DADS in terms of sample efficiency, in which we should expect offDADS to be better than the original version of DADS. We compared the two algorithms using two domains: one using 5 skills and one using 20 skills. Figures 13 and 14 show the results we obtained for every 1.000 epochs using 5 skills.

As we can see from the plots, off-policy learning returns better results than on-policy. Although the output for on-policy after the first 1.000 epochs seemed really promising, even better than off-policy, in the following epochs the skills failed to explore more states and diversify from each other, resulting in the improvement of only 1 or 2 skills towards the road of 5.000 epochs. In contrast, the off-policy method had a more steady road, but also did not succeed in improving all skills, since two of them are stuck near the initial position.

Following the above results, we decided to increase the number of skills to 20 in order to check if the "state space distance" they must keep due to the formula of mutual information (diversity), could play an important role in the performance. Results can be seen in Appendix A. Here, we have a clearer view of the difference between on and off-policy, but also a huge improvement on

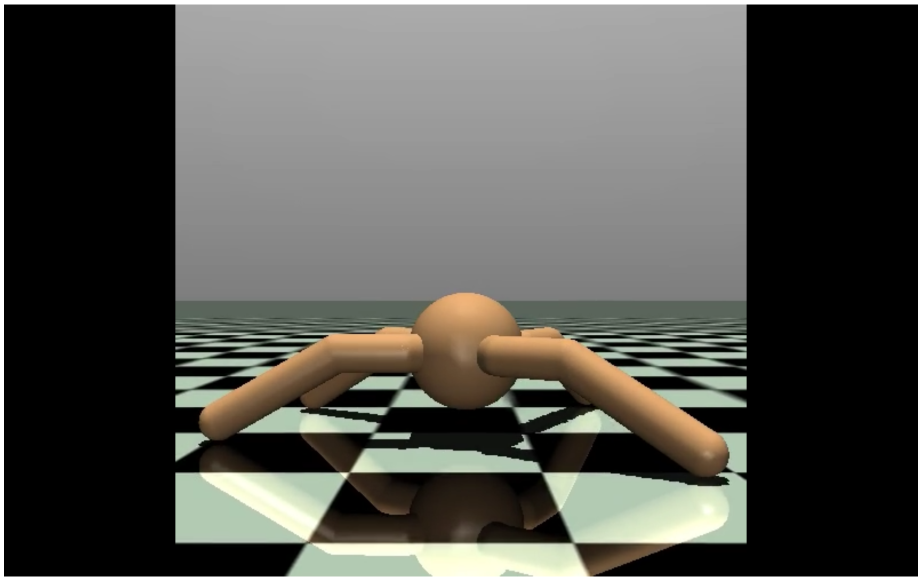


Figure 12: Ant's Environment

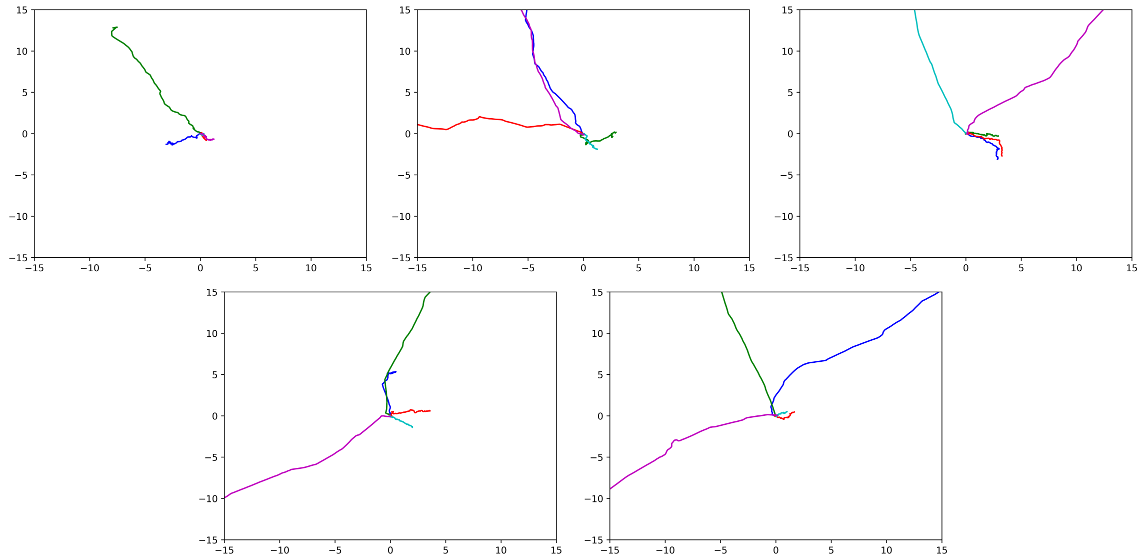


Figure 13: Off-policy learning on reduced observation space (x-y) with 5 skills. Each image corresponds to 1.000 epochs from left to right (i.e. first on the left=1.000 epochs, next on the right=2.000 epochs, etc)

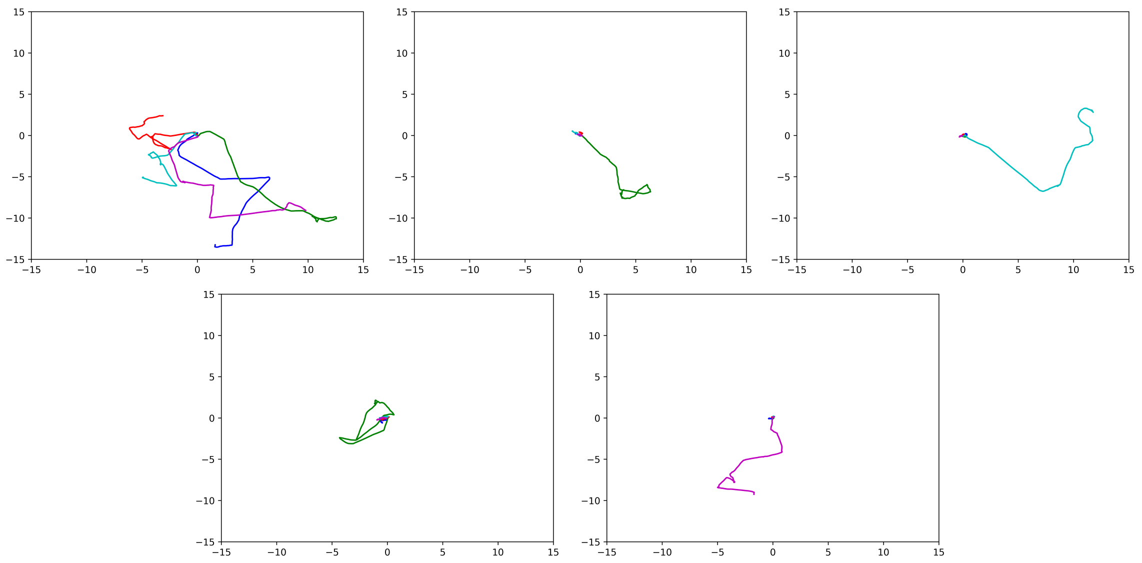


Figure 14: On-policy learning on reduced observation space (x-y) with 5 skills. Each image corresponds to 1.000 epochs from left to right (i.e. first on the left=1.000 epochs, next on the right=2.000 epochs, etc)

the performance of both the algorithms. Skills cover a large amount of the state space, not only near their initial position but also in the whole spectrum of the environment. Of course, we can easily notice that the skills in the off-policy algorithm are spreading in a more structured way than on-policy, and reaching greater distances.

By looking at the experiments on these two domains, we can conclude that off-policy learning, especially in the latter case, yields better results than the on-policy method, providing us with a better exploration technique, skill-wise, for our agent.

4.1.2 Observation space

So far our agent was running in a reduced observation space—remember only two dimensions (x and y axes) out of 111 dimensions— in which we focused on learning skills that could create trajectories in different directions on the environment. What exactly would happen if we were to free the agent from these observation limits? Is it going to change the outcome or just slow it down?

Our first guess, before running the experiment, was that the discovery of the skills would be at a lower scale. The complexity of the available moves would rise dramatically along with the number of different states, but that wouldn't cause any big issues. Unfortunately, this was totally incorrect. Figure 15 shows the results we obtained for every 1.000 epochs using 20 skills on the full observation space. We also tried the on-policy version, in which we used 5 skills on the full observation space, and can see the result in figure 16.

As we can see from the plots, the results in both cases are not so useful. Considering that the number of observations is now really large in comparison to the previous cases, our agent cannot discover useful skills, at least not to the human eye. We are adding the last remark, because although

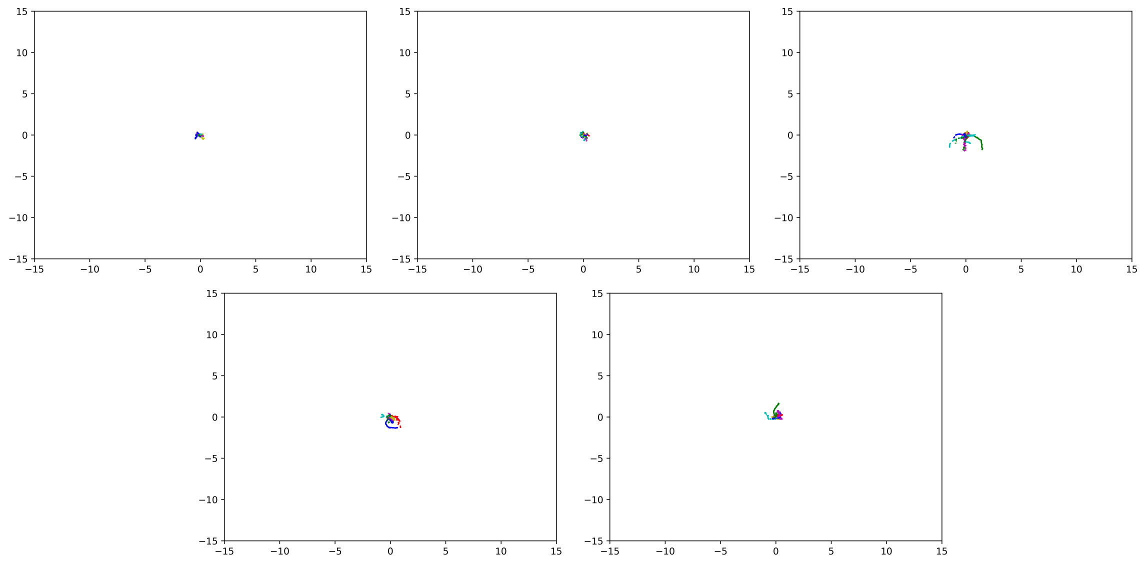


Figure 15: Off-policy learning on full observation space with 20 skills. Each image corresponds to 1.000 epochs from left to right (i.e. first on the left=1.000 epochs, next on the right=2.000 epochs, etc)

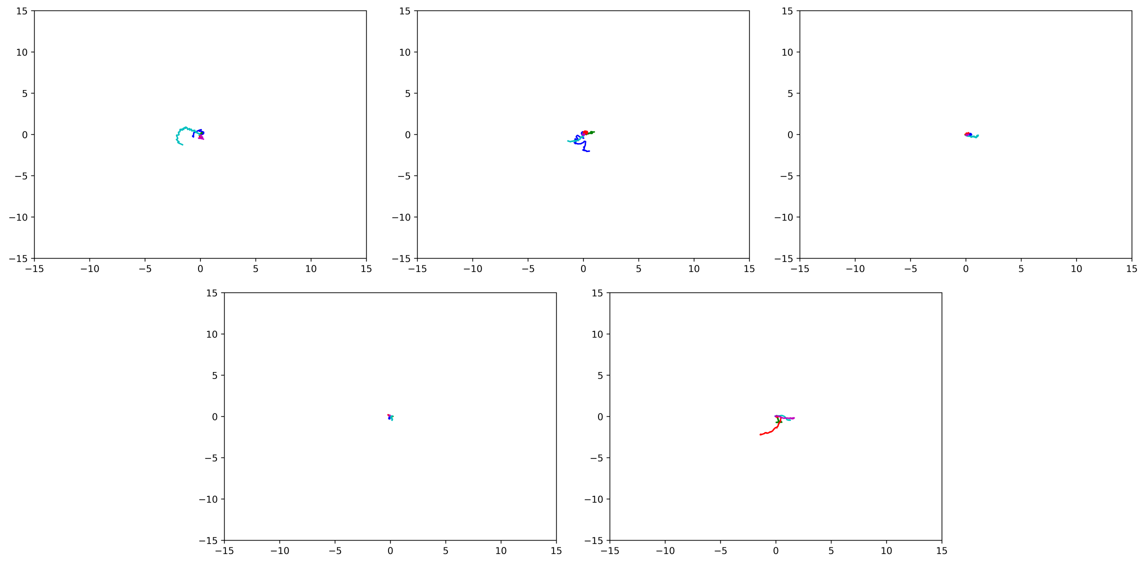


Figure 16: On-policy learning on full observation space with 20 skills. Each image corresponds to 1.000 epochs from left to right (i.e. first on the left=1.000 epochs, next on the right=2.000 epochs, etc)

we can't see trajectories far away from its initial position in the x-y axis, it doesn't necessarily imply that our agent is performing less actions than before or discovering skills that aren't useful. The difference is that now, its observational space became incredibly complex (111 dimensions) that in order to gain solid results we would either need more powerful hardware or a big amount of time (check the following subsection for the latter).

4.1.3 Epochs/Final preparation

It's time we start focusing more on the off-policy method since we know it improved the overall performance. The authors of DADS run both of their algorithms for 10.000 epochs, and not for 5.000 that we were experimenting so far. We decided that we have to make our skills more robust so they can provide us with better results when they would be fed as tasks on the ML algorithms; with that being said, we increased the number of epochs to 10.000 and continued the runs (remember we saw results from 5.000 epochs in the previous subsections), without starting from scratch. Along with 5 and 20 skills (we tested 20 skills on both reduced and full observation space) that we were testing until now, we also tried with 50 skills for two reasons: a) we wanted to check if the state space would be covered better than before by discovering more skills, and b) we wanted to see if the number of skills could have any big impact in a more complicated dimensional space.

We tested four domains: off-policy learning on reduced observation space with 5, 20 and 50 skills, and off-policy learning on full observation space with 20 skills. We knew that the latter wouldn't be useful for our algorithm in the future, but we wanted to try it anyway to better understand the power of DADS. Figures 17 and 18 show the results we obtained when we ran experiments on reduced observation space with 20 and 50 skills; the rest of the results can be found on Appendix B.

In the domain of 5 skills, compared with Figure 13, Figure 23 shows that there isn't any significant difference. Some of the skills kept expanding (the values of the coordinates of the finish-line of the ant's journey are increasing in parallel with the number of epochs-that is also for every domain we will discuss later on) covering more from the state space but the overall progress is not steady at all. We can see that raising the number of epochs, does not necessarily mean that our skills would move uniformly in the state space in a way that can prove the usefulness of this increase.

In the domain of 20 skills (x-y), if we compare the Figure 21 with the Figure 17, we can see that the "performance" is mixed. This means that some of our runs, with a lower number of epochs, resulted in having skills that cover more state space than the ones that run one or two thousand epochs more. By that we can say with certainty that we can't estimate the best number of epochs or finetune it with the algorithm of DADS. In the domain of 20 skills in the full observation space, we can't see any change (at least in the x-y state space coverage). However, as we mentioned before, we can't criticize the algorithm or the number of epochs in the full observation space, based only on the plots that display just the trajectories on x-y axis.

In the domain of 50 skills (x-y), we can say that the results look promising. The skills have a really good coverage over the state space and are constantly "growing" as the number of epochs increasing. The only drawback we observed from the plots, is that the skills seem to get divided into two regions at 4-5.000 epochs, and then follow this pattern until the end. We can't give a clear explanation on why this event is happening.

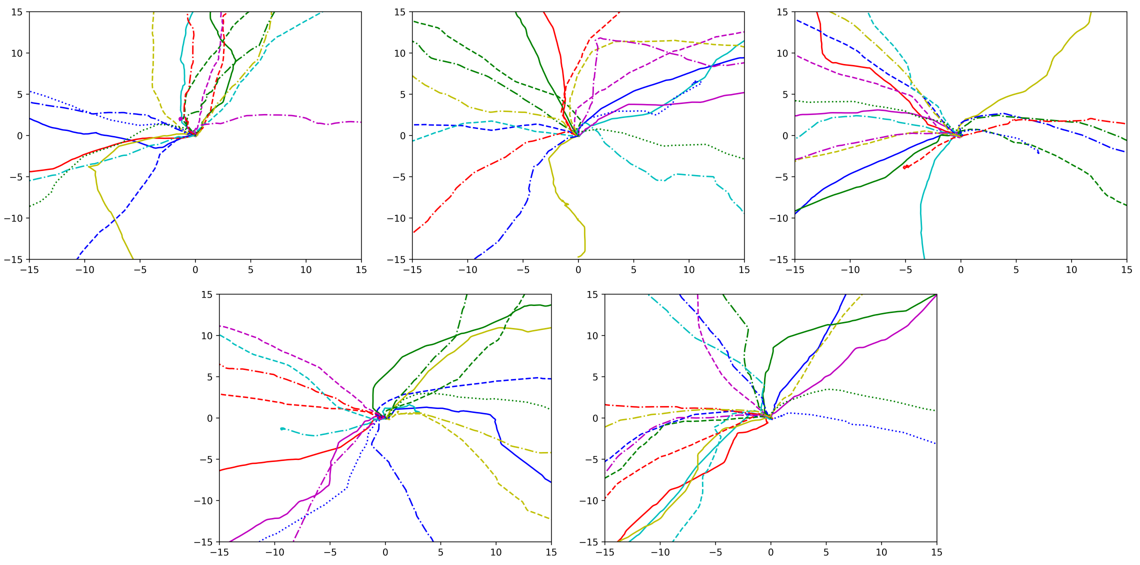


Figure 17: Off-policy learning on reduced observation space (x-y) with 20 skills. Each image corresponds to 1.000 epochs from left to right (i.e. first on the left=6.000 epochs, next on the right=7.000 epochs, etc)

4.1.4 Feeding the Meta-Learning algorithms

We are now ready to use our discovered skills as tasks for our agent in the ML framework. Due to the big number of different domains we presented in the previous subsection, we have to choose specific domains that we will meta-train our agent on. Otherwise, by adding different hyperparameters also for ML on top of all the domains we have already created, the number of the future runs will grow exponentially resulting in a variety of combinations that we won't be able to execute with the current hardware on a certain amount of time.

By observing the plots, we can say that the off-policy domain is "stronger" than the on-policy, regardless of the number of skills or epochs. Stronger means that they can cover more state space, faster and with better efficiency. With that in mind, we will choose only domains that are using the offDADS algorithm. Now the question is how many skills should we use and in what observation space? The latter can be answered almost immediately; since we didn't get any good results in the full observation space, we will use the skills that offDADS created using the restricted observation space (x-y). But what about the number of skills?

Our initial thought was to use either the 20 or the 50 skills, because we noticed that they cover a lot of the ant's state space. Unfortunately, we didn't have any strong arguments on why we should use either the domain with the 20 or the 50 skills, so it would be totally random. The first could mean faster results with maybe the same or worse efficiency, while the second could result in slower results with the same or better efficiency. But this is not our only dilemma. Remember, in the subsection 3.3, we discussed how both the number of skills and the number of epochs could dramatically affect the meta-training phase. The number of tasks which we will feed to the ML framework will be equal to the number of skills we will define, multiplied by the number of epochs

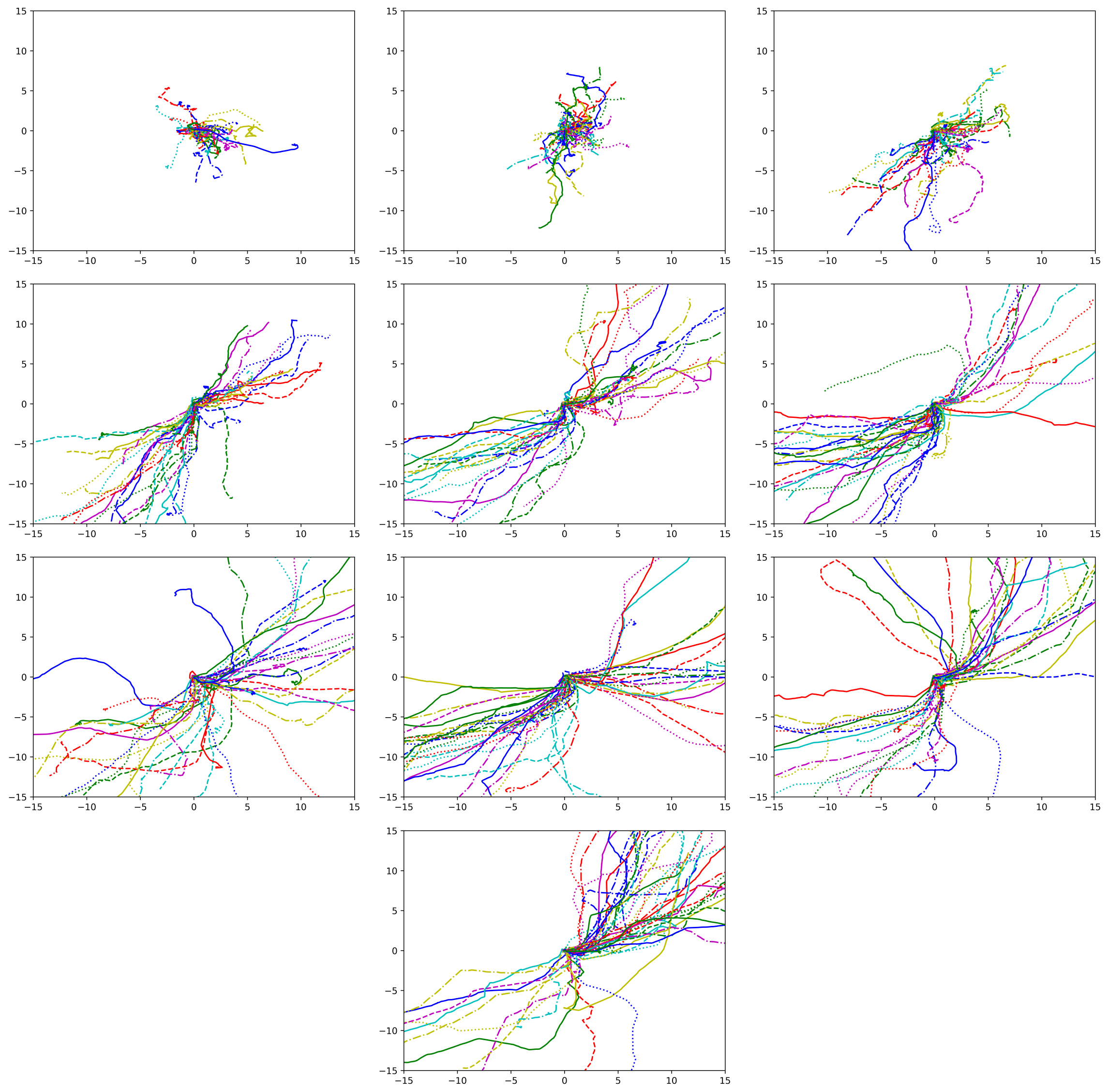


Figure 18: Off-policy learning on reduced observation space (x-y) with 50 skills. Each image corresponds to 1.000 epochs from left to right (i.e. first on the left=1.000 epochs, next on the right=2.000 epochs etc, until 10.000 epochs)

from which we will choose to extract the final positions of the ant. Keep in mind that the number of skills is the same as the number of final positions. With that in mind, it would not be so wise to extract tasks from all the epochs we run unsupervised learning for, because the number would be enormous with either 20 or 50 skills (10.000 epochs * number of skills). Since we have a visualization of the trajectories in every 1.000 epochs, then we can use this information to extract the final positions of the ant for every 1.000 epochs until the end; that would mean the number of skills multiplied by 10 (10.000 epochs / 1.000 epochs).

We based our final decision about the number of tasks simply on the machine learning ideology. Since more data usually (not always) indicate better results, we will select the domain with the 50 skills, we will feed the ML algorithms with 500 (50 *10) skills and expect that our choices will return the best results. Still, compared to the most ML algorithms that are randomly sampling tasks from continuous probability distributions, our fixed number doesn't seem large.

4.2 Meta-RL

We divided our experiments into two categories with two subcategories each. The first category consists of training our algorithms with our pretrained tasks from the unsupervised learning framework and the second consists of training the algorithms with random tasks. Needless to say that in both cases, we fed the algorithms with random tasks at meta-test time to check their performance when given new challenges. Now, the key element that separates their subcategories is the parameter of the environment's limit. While most researchers are training and testing the Ant using a space of coordinates with range $[-3, +3]$, we also wanted to try a different domain, so we expanded this range to $[-15, +15]$ (this doesn't mean that the Ant cannot move outside of this range; we will refer to range $[-3, +3]$ and range $[-15, +15]$, as limit 3 and limit 5, respectively). Like we did before, both the domains were tested in order to see if either the limits of the environment or our choice to extract the final positions of the Ant could have an impact on the results.

4.2.1 Technical Details

We have already trained our agent and discovered a number of skills as shown in subsection 4.1. We shall proceed to propose them as tasks to our algorithms and introduce the framework we used for our experiments.

Our ML framework is called *Garage* [19] and is formally considered a "toolkit" for developing and evaluating RL algorithms. Most importantly, we chose this specific framework because it already had our main ML algorithms (MAML, PEARL) available for use, saving us months of unnecessary work (updating the original repositories of the algorithms to the current version of python, tensorflow etc) and providing us with extra time to focus on our research.

Unfortunately, Garage did not have the Ant's environment available for their framework, so we had to either create a new one or borrow one. We adapted the code from [8] and modified it to be compatible with our needs, without changing the structure of the reward function. We created different sampling methods for meta-training and meta-testing processes; at meta-test time we are sampling random tasks while at meta-training time we need to sample tasks that were "created" by the DADS algorithm.

During the second phase of our research, our main PC broke down, so we had to run the experiments with a different hardware. We ran our meta-learning experiments on the same software (Ubuntu 18.04.4 LTS system), but with a six-core Intel Core i7-5820k Processor and a 16 GiB

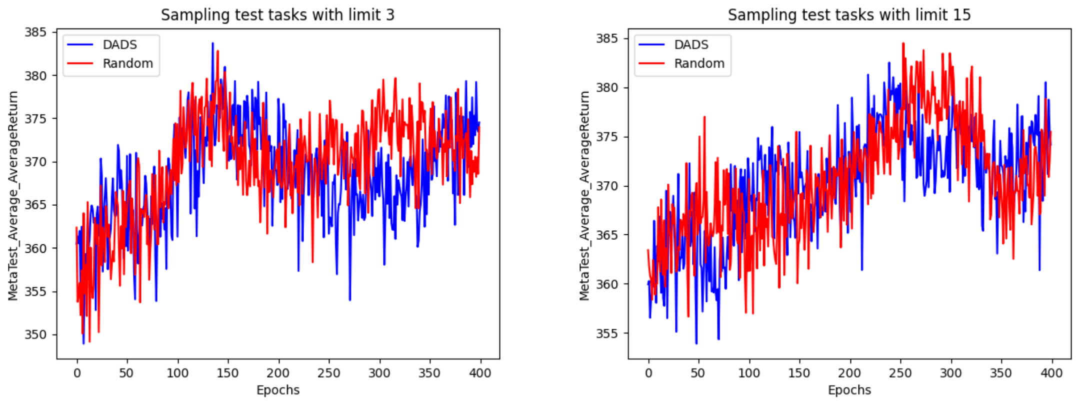


Figure 19: **Left**: Average return of MAML algorithm at meta-test time with limit 3, **Right**:Average return of MAML algorithm at meta-test time with limit 15

RAM. Each of the MAML run lasted approximately 10 hours, while each of the PEARL run lasted approximately 3 days.

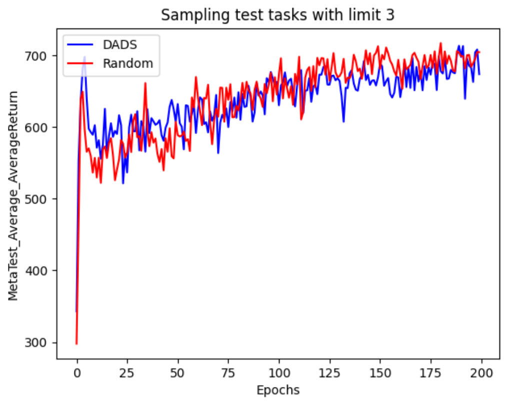
4.2.2 MAML

For the MAML algorithm, we set the values of the hyperparameters according to the original paper of the algorithm [14] (basic hyperparameters can also be seen in Appendix C). The only hyperparameter we changed was the number of epochs, in order to reduce the running time, from 500 to 400. The rest of them remained intact. Plot 19 displays the average return at the meta-test time for the domain of limit 3 and 15, respectively.

As we can see from the plots, the domains have more similarities than differences. Regardless of the limit applied, we can see a small, but "messy" improvement of the average return of the algorithm over time, whether we feed the MAML algorithm with random tasks or with pretrained tasks from the DADS algorithm. The values of the average return, as well as their highest and lowest points, are quite similar in both cases. However, we believe that a bigger number of epochs could improve the performance of the algorithm and therefore the results, but this will be determined in future research.

4.2.3 PEARL

For the PEARL algorithm, we also set the values of the hyperparameters according to the original paper of the algorithm [59] (basic hyperparameters can also be seen in Appendix C). Once more, the only hyperparameter we changed was the number of epochs (again to reduce the running time), but this time the reduction was more "severe". We measured the running time of 100 epochs with our current hardware and observed that we were running 60-70 epochs per day. The original value was set to 500 epochs, so we would need approximately 8 days of non-stop running for only one domain (out of four). This was extremely time-consuming for our research, so we decided to reduce the number of epochs to 200. Similarly with MAML, we created Figure 20 that displays the average return at the meta-test time for the domain of limit 3 and 15, respectively.



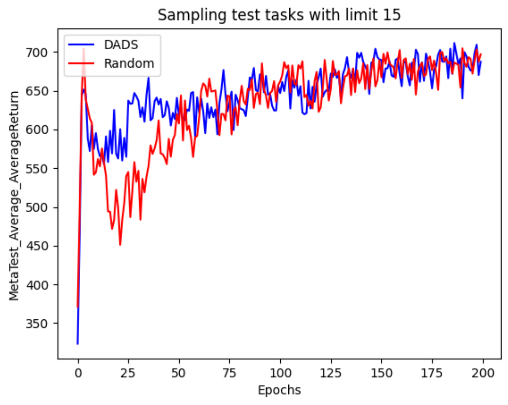


Figure 20: **Left**: Average return of PEARL algorithm at meta-test time with limit 3, **Right**:Average return of PEARL algorithm at meta-test time with limit 15

One can observe in the plots a sudden improvement in performance from the early epochs, and then a steady rise of the average return, in both cases of random and DADS-pretrained tasks. The results of PEARL are similar, in the overall performance of the algorithm, to those of MAML, so we cannot deduce anything important from the plots. In this case we also believe that a bigger number of epochs could improve the performance of the algorithm.

5 Discussion

There are a few things we would like to discuss considering our research. It has been an interesting journey that has given us the chance to wander into ”technological roads” that will play a significant part in humanity's future. We strongly believe that the frameworks of RL and ML will be important tools for the future of AI, which as years go by, is getting more fascinating and full of adventures. In this section we will talk about the findings of this research and criticize the results.

5.1 Discovery of skills/tasks

Our experiments showed that using the off-policy DADS algorithm was clearly better than the on-policy version in terms of sample efficiency. We also observed that by increasing the number of dimensions, the level of the algorithm's performance dropped dramatically. Of course both of the above results were expected as we have already seen them in previous researches. What surprised us the most in the unsupervised learning process was the different outcomes when we were changing the number of the skills. While we were expecting that trying a domain with 20 skills would be much better than the domain with 5 skills, the increase to 50 skills affected the performance, but not as much as we expected. This leads us back to the exact same question we had before running the experiments: what is the most efficient number of skills in an environment and how can we find it?

Unfortunately we do not have the answer to that, but we can probably guess that no matter the number of the skills, the optimal solution to this ”ongoing problem” would be to have a uniform diversity of the skills independent of the number of dimensions. This means that we needed to create a mechanism that would be able to stop the runs when the skills cover the whole state space uniformly. In this case, we also had to make sure that this wouldn't happen during the first runs, but after a certain number of epochs so that our algorithm can develop the ability for macro-actions. We leave this subject for future work.

5.2 Unsupervised Meta-RL

Both MAML and PEARL weren't strongly affected by the change of the limit, as we observed in the plots. The average return was more or less similar regardless of the limit (we are talking about each algorithm separately) which indicates that either the change in the limit was minor or the limit doesn't affect the agent at all. Apart from limit, we believe that a change in the conditions/hyperparameters of the experiments could have an impact on the efficiency of the algorithm (bigger number of epochs, different architectures of neural networks, other environments etc) so we will leave it for future work.

The comparison of feeding random tasks and pretrained tasks on the Meta-RL algorithms was executed with a small scale experimentation technique but successfully. The results of the Unsupervised Meta-RL process clearly showed that the pretrained tasks weren't able to return better results than the random tasks at the meta-test process, similar to the outcome of [23]. However we argue that this doesn't necessarily mean that pretrained tasks could not be more effective with a different process/algorithm than ours.

5.3 Few Limitations

During our research we came across a number of impediments and limitations. Some of them had an impact on our results, while others were minor compared to the overall progress.

Our biggest limitation was the running time combined with the hardware. Although we tried to find the optimal hardware to run our experiments, unfortunately we were unable to. As mentioned above, we used two different computers for our experiments, which were by no means optimal, but they were decent. In both cases, the time spent running our experiments was more than expected, at least according to our initial plan. Having in mind that the practical part of the research should approximately be finished within 6 months (it took more than that—nearly 9 months), we adjusted the hyperparameters and the number of the experiments, in order to reduce the running time and this led to a decrease in the performance of the algorithm; it was a "sacrifice" we were forced to do.

Another limitation of our research was the coding part. The inability to adapt the existing code according to our needs cost us a lot of time but also changed our plans, as we discussed in 3.2. The execution of the original idea 3.1 would yield different results. However we cannot be certain that these results would have been more promising. For the same reason, testing our algorithm in a different environment was extremely challenging from a programming point of view, limiting us in one specific 3D environment.

5.4 Future work

There are a number of things that can improve our algorithm and a lot more that could be done in order to improve the idea of Unsupervised Meta-RL. We will separate them into two subsections and create bullets to help the reader understand our ideas for the future.

5.4.1 Feasible Improvements

- Try the current set up of algorithms with the original idea (see 3.1
- Experiment with different hyperparameters/neural networks/environments
- Examine different algorithms for both the Unsupervised Learning and the ML
- Test a higher-scale experimentation process to improve the performance of the algorithm (more epochs, more episodes etc)

5.4.2 Hardcore Improvements

- Automate the process of feeding the tasks/skills from DADS to Garage
- Find a way to get the optimal number of skills in the unsupervised learning process
- Implement new/updated techniques for better sampling of the data (e.g. Bayesian methods)

5.5 Conclusion

This research aimed at creating an Unsupervised Meta-RL technique for fast adaptation to unknown tasks. The field of Information Theory along with the idea of an existing algorithm called DIAYN, gave us the opportunity to use an improved version of DIAYN called DADS in order to train an RL agent, without any human supervision, and create skills for the agent that would be later fed as tasks to a ML algorithm. Although the original idea, which we borrowed from existing research, was to train a discriminator network to be able to predict these skills and then use this discriminator to propose tasks for the ML process, we changed the process and extracted data regarding the goal of the agent in the unsupervised learning process, so that we could use this data as tasks in the ML process.

For the ML process we used two algorithms, one for on-policy learning (MAML) and one for off-policy learning (PEARL). Our evaluation metric was the average return of the algorithms at the meta-test time; we focused on comparing the performance of the two algorithms (not between them but separately for each algorithm) when provided with pretrained tasks (from DADS) or random tasks. The results showed that there isn't any significant difference among these two task proposals. There are many things that could be done to improve the current research and generally the framework of Unsupervised Meta-RL which seems to be a stepping stone for the evolution of AI.

References

[1] Joshua Achiam, Harrison Edwards, Dario Amodei, and Pieter Abbeel. Variational option discovery algorithms. *arXiv preprint arXiv:1807.10299*, 2018.

[2] Peter AR Ade, N Aghanim, M Arnaud, Mark Ashdown, J Aumont, C Baccigalupi, AJ Banday, RB Barreiro, JG Bartlett, N Bartolo, et al. Planck 2015 results-xiii. cosmological parameters. *Astronomy & Astrophysics*, 594:A13, 2016.

[3] Alexander A Alemi, Ian Fischer, Joshua V Dillon, and Kevin Murphy. Deep variational information bottleneck. *arXiv preprint arXiv:1612.00410*, 2016.

[4] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in neural information processing systems*, pages 3981–3989, 2016.

[5] Arthur Aubret, Laetitia Matignon, and Salima Hassas. A survey on intrinsic motivation in reinforcement learning. *arXiv preprint arXiv:1908.06976*, 2019.

[6] Andrew G Barto. Intrinsic motivation and reinforcement learning. In *Intrinsically motivated learning in natural and artificial systems*, pages 17–47. Springer, 2013.

[7] Nuttapong Chentanez, Andrew Barto, and Satinder Singh. Intrinsically motivated reinforcement learning. *Advances in neural information processing systems*, 17:1281–1288, 2004.

[8] Tristan Deleu. Model-Agnostic Meta-Learning for Reinforcement Learning in PyTorch, 2018. Available at: https://github.com/tristandeleu/pytorch-maml-rl.

[9] Yan Duan, Marcin Andrychowicz, Bradly Stadie, OpenAI Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. In *Advances in neural information processing systems*, pages 1087–1098, 2017.

[10] 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.

[11] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.

[12] Rasool Fakoor, Pratik Chaudhari, Stefano Soatto, and Alexander J Smola. Meta-q-learning. *arXiv preprint arXiv:1910.00125*, 2019.

[13] Chelsea Finn. Lecture notes on multi-task and meta-learning, 2019.

[14] 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*, pages 1126–1135. JMLR. org, 2017.

[15] Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International conference on machine learning*, pages 49–58, 2016.

[16] Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot visual imitation learning via meta-learning. *arXiv preprint arXiv:1709.04905*, 2017.

[17] Carlos Florensa, Yan Duan, and Pieter Abbeel. Stochastic neural networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1704.03012*, 2017.

[18] Mayumi Fukuyama. Society 5.0: Aiming for a new human-centered society. *Japan Spotlight*, 27:47–50, 2018.

[19] The garage contributors. Garage: A toolkit for reproducible reinforcement learning research. `https://github.com/rlworkgroup/garage`, 2019.

[20] Francisco Garcia and Philip S Thomas. A meta-mdp approach to exploration for lifelong reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 5692–5701, 2019.

[21] Ben Goertzel and Cassio Pennachin. *Artificial general intelligence*, volume 2. Springer, 2007.

[22] Karol Gregor, Danilo Jimenez Rezende, and Daan Wierstra. Variational intrinsic control. *arXiv preprint arXiv:1611.07507*, 2016.

[23] Abhishek Gupta, Benjamin Eysenbach, Chelsea Finn, and Sergey Levine. Unsupervised meta-learning for reinforcement learning. *arXiv preprint arXiv:1806.04640*, 2018.

[24] 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*, pages 5302–5311, 2018.

[25] 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*, 2018.

[26] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, SM Eslami, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

[27] Rob High. The era of cognitive systems: An inside look at ibm watson and how it works. *IBM Corporation, Redbooks*, pages 1–16, 2012.

[28] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in neural information processing systems*, pages 4565–4573, 2016.

[29] Sepp Hochreiter, A Steven Younger, and Peter R Conwell. Learning to learn using gradient descent. In *International Conference on Artificial Neural Networks*, pages 87–94. Springer, 2001.

[30] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *arXiv preprint arXiv:2004.05439*, 2020.

[31] Matthew B Hoy. Alexa, siri, cortana, and more: an introduction to voice assistants. *Medical reference services quarterly*, 37(1):81–88, 2018.

[32] Human Evolution. Human evolution — Wikipedia, the free encyclopedia. [Online; accessed 29-April-2020].

[33] Thomas Henry Huxley. Darwin on the origin of species. *Westminster Review*, 17(2):541–70, 1860.

[34] Shinpei Kato, Eijiro Takeuchi, Yoshio Ishiguro, Yoshiki Ninomiya, Kazuya Takeda, and Tsuyoshi Hamada. An open approach to autonomous vehicles. *IEEE Micro*, 35(6):60–68, 2015.

[35] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[36] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[37] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[38] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2. Lille, 2015.

[39] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

[40] Solomon Kullback. *Information theory and statistics*. Courier Corporation, 1997.

[41] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[42] Sergey Levine. Unsupervised reinforcement learning.

[43] Sergey Levine. Lecture notes on inverse reinforcement learning, 2019.

[44] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy M Hospedales. Learning to generalize: Meta-learning for domain generalization. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[45] Nan LI and Guo-dong LIU. Intrinsic motivation reinforcement learning and its application to robocup simulation [j]. *Computer Simulation*, 4, 2006.

[46] Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. Meta-sgd: Learning to learn quickly for few-shot learning. *arXiv preprint arXiv:1707.09835*, 2017.

[47] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[48] Eric V Linder. Exploring the expansion history of the universe. *Physical Review Letters*, 90(9):091301, 2003.

[49] Peter Menzel and Faith d'Aluisio. *Robo sapiens: Evolution of a new species*. Mit Press, 2000.

[50] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. *arXiv preprint arXiv:1707.03141*, 2017.

[51] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[52] Shakir Mohamed and Danilo Jimenez Rezende. Variational information maximisation for intrinsically motivated reinforcement learning. In *Advances in neural information processing systems*, pages 2125–2133, 2015.

[53] Tsendsuren Munkhdalai and Hong Yu. Meta networks. *Proceedings of machine learning research*, 70:2554, 2017.

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

[55] Pierre-Yves Oudeyer, Frdric Kaplan, and Verena V Hafner. Intrinsic motivation systems for autonomous mental development. *IEEE transactions on evolutionary computation*, 11(2):265–286, 2007.

[56] Pierre-Yves Oudeyer and Frederic Kaplan. What is intrinsic motivation? a typology of computational approaches. *Frontiers in neurorobotics*, 1:6, 2009.

[57] Dennis Overbye. Cosmos controversy: The universe is expanding, but how fast? *The New York Times*, 20, 2017.

[58] Principle of maximum entropy. Principle of maximum entropy — Wikipedia, the free encyclopedia. [Online; accessed 24-December-2020].

[59] Kate Rakelly, Aurick Zhou, Chelsea Finn, Sergey Levine, and Deirdre Quillen. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International conference on machine learning*, pages 5331–5340, 2019.

[60] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. 2016.

[61] Sebastian Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

[62] Richard M Ryan and Edward L Deci. Intrinsic and extrinsic motivations: Classic definitions and new directions. *Contemporary educational psychology*, 25(1):54–67, 2000.

[63] Bruno Salgues. *Society 5.0: Industry of the Future, Technologies, Methods and Tools*. John Wiley & Sons, 2018.

[64] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pages 1842–1850, 2016.

[65] Jürgen Schmidhuber. *Evolutionary principles in self-referential learning. On learning how to learn: The meta-meta-... hook.* PhD thesis, TU Munich, 1987.

[66] Jürgen Schmidhuber. Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE Transactions on Autonomous Mental Development*, 2(3):230–247, 2010.

[67] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

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

[69] Nicolas Schweighofer and Kenji Doya. Meta-learning in reinforcement learning. *Neural Networks*, 16(1):5–9, 2003.

[70] Claude E Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[71] Archit Sharma, Michael Ahn, Sergey Levine, Vikash Kumar, Karol Hausman, and Shixiang Gu. Emergent real-world robotic skills via unsupervised off-policy reinforcement learning. *arXiv preprint arXiv:2004.12974*, 2020.

[72] Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised discovery of skills. *arXiv preprint arXiv:1907.01657*, 2019.

[73] David Silver. Lecture notes on reinforcement learning, 2015.

[74] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[75] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. *Advances in neural information processing systems*, 30:4077–4087, 2017.

[76] Bradly C Stadie, Ge Yang, Rein Houthooft, Xi Chen, Yan Duan, Yuhuai Wu, Pieter Abbeel, and Ilya Sutskever. Some considerations on learning to explore via meta-reinforcement learning. *arXiv preprint arXiv:1803.01118*, 2018.

[77] Sainbayar Sukhbaatar, Zeming Lin, Ilya Kostrikov, Gabriel Synnaeve, Arthur Szlam, and Rob Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. *arXiv preprint arXiv:1703.05407*, 2017.

[78] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1199–1208, 2018.

[79] Flood Sung, Li Zhang, Tao Xiang, Timothy Hospedales, and Yongxin Yang. Learning to learn: Meta-critic networks for sample efficient learning. *arXiv preprint arXiv:1706.09529*, 2017.

[80] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* Second edition, 2017.

[81] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[82] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. *Advances in neural information processing systems*, 29:3630–3638, 2016.

[83] 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.

[84] Lilian Weng. Meta learning. *lilianweng.github.io/lil-log*, 2018.

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

[86] Tianbing Xu, Qiang Liu, Liang Zhao, and Jian Peng. Learning to explore with meta-policy gradient. *arXiv preprint arXiv:1803.05044*, 2018.

[87] A Steven Younger, Sepp Hochreiter, and Peter R Conwell. Meta-learning with backpropagation. In *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No. 01CH37222)*, volume 3. IEEE, 2001.

[88] Yu Zhang and Qiang Yang. A survey on multi-task learning. *arXiv preprint arXiv:1707.08114*, 2017.

[89] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3357–3364. IEEE, 2017.

[90] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

A Off-policy vs On-policy

Figures 21 and 22 show the results we obtained for every 1.000 epochs using 20 skills.

B Epochs/Final preparation

Figures 23 and 24 show the results we obtained for 5 skills on reduced observation space and 20 skills on full observation space, respectively.

C Hyperparameters

MAML Hyperparameters: For the MAML algorithm, the number of epochs is 400, the number of episodes per task is 40, the meta-batch size is 30, the inner learning rate is 0.1 and the outer learning rate is 1e-3.
PEARL Hyperparameters: For the PEARL algorithm, the number of epochs is 200, the number of training tasks is 100, the number of test tasks is 30 and the meta-batch size is 64.

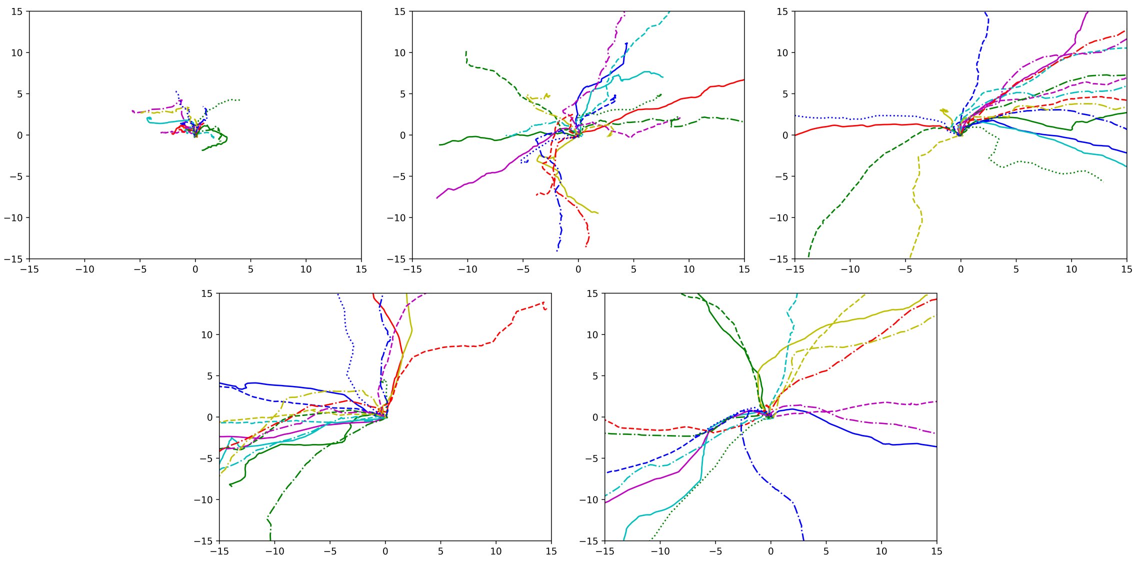


Figure 21: Off-policy learning on reduced observation space (x-y) with 20 skills. Each image corresponds to 1.000 epochs from left to right (i.e. first on the left=1.000 epochs, next on the right=2.000 epochs, etc)

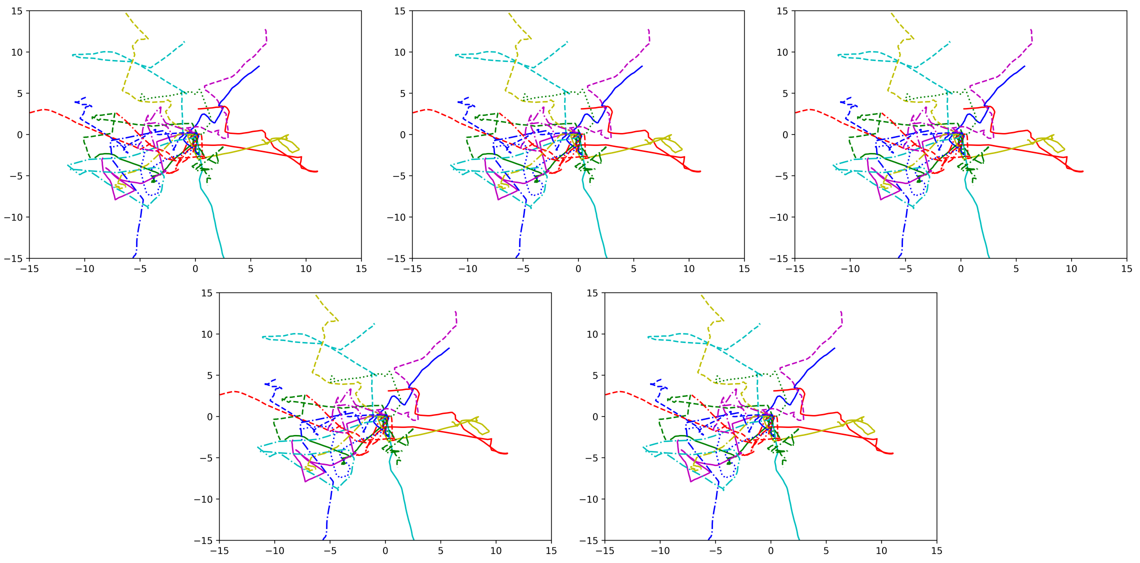


Figure 22: On-policy learning on reduced observation space (x-y) with 20 skills. Each image corresponds to 1.000 epochs from left to right (i.e. first on the left=1.000 epochs, next on the right=2.000 epochs, etc)

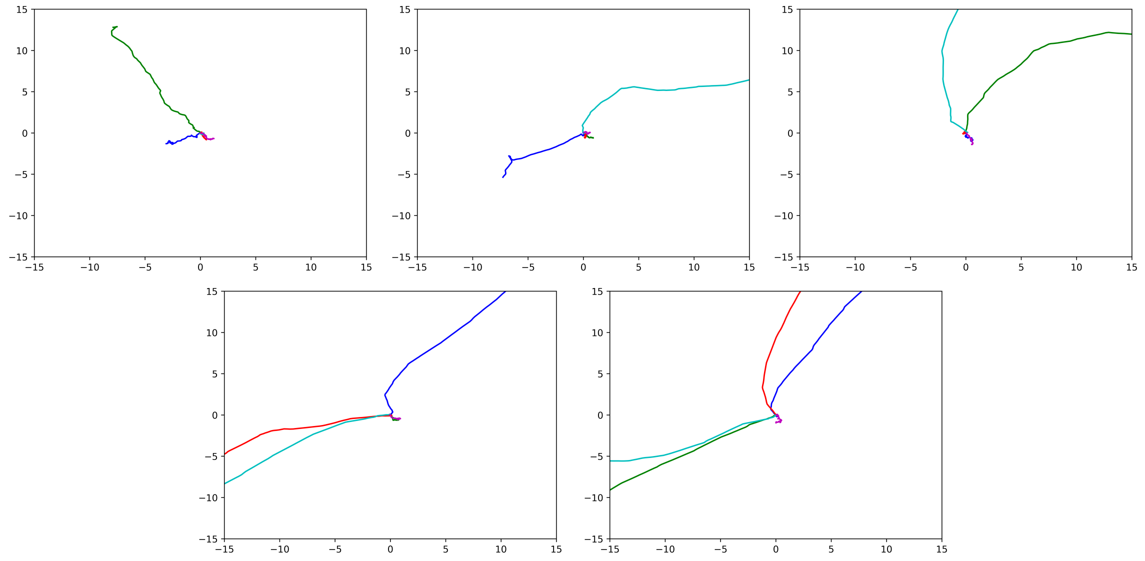


Figure 23: Off-policy learning on reduced observation space (x-y) with 5 skills. Each image corresponds to 1.000 epochs from left to right (i.e. first on the left=6.000 epochs, next on the right=7.000 epochs, etc)

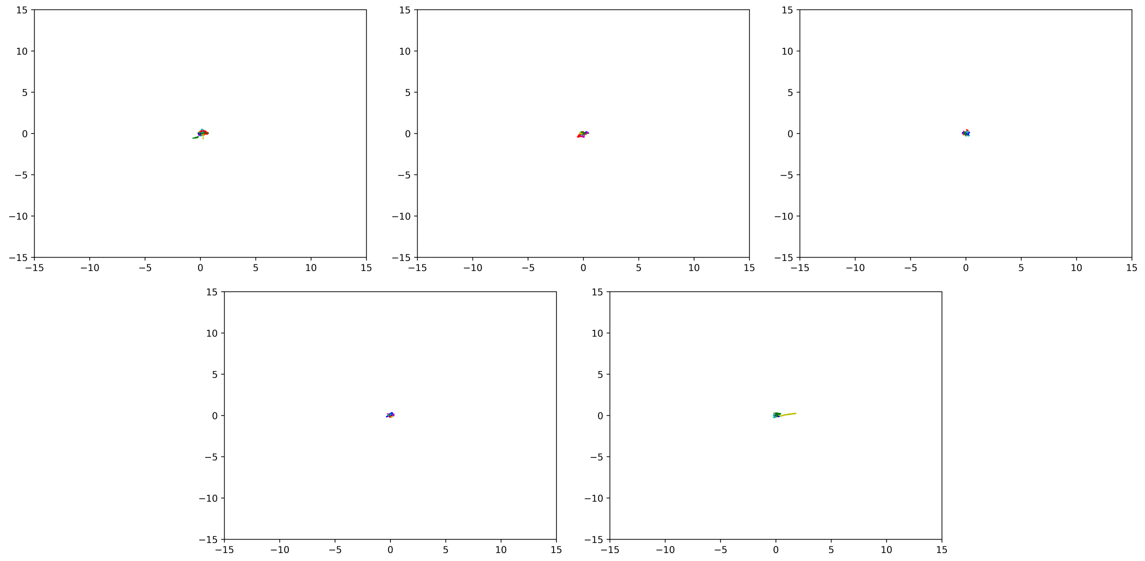


Figure 24: Off-policy learning on full observation space with 20 skills. Each image corresponds to 1.000 epochs from left to right (i.e. first on the left=6.000 epochs, next on the right=7.000 epochs, etc)