

CURIOS: Intrinsically Motivated Modular Multi-Goal Reinforcement Learning

Cédric Colas¹ Pierre Fournier² Olivier Sigaud² Mohamed Chetouani² Pierre-Yves Oudeyer¹

Abstract

In open-ended environments, autonomous learning agents must set their own goals and build their own curriculum through an intrinsically motivated exploration. They may consider a large diversity of goals, aiming to discover what is controllable in their environments, and what is not. Because some goals might prove easy and some impossible, agents must actively select which goal to practice at any moment, to maximize their overall mastery on the set of learnable goals. This paper proposes CURIOUS, an algorithm that leverages 1) a modular Universal Value Function Approximator with hindsight learning to achieve a diversity of goals of different kinds within a unique policy and 2) an automated curriculum learning mechanism that biases the attention of the agent towards goals maximizing the absolute learning progress. Agents focus sequentially on goals of increasing complexity, and focus back on goals that are being forgotten. Experiments conducted in a new modular-goal robotic environment show the resulting developmental self-organization of a learning curriculum, and demonstrate properties of robustness to distracting goals, forgetting and changes in body properties.

1. Introduction

In *autonomous continual learning*, agents aim to discover repertoires of skills in an ever-changing open-ended world, and without external rewards. In such realistic environments, the agent must be endowed with intrinsic motivations to explore the diversity of ways in which it can control its environment. One important form of intrinsic motivation system is the ability to autonomously set one's own goals

¹Flowers Team, Inria and Ensta ParisTech, FR. ²ISIR, Sorbonne Univ., Paris, FR.. Correspondence to: Cédric Colas <cedric.colas@inria.fr>.

Proceedings of the 36th International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

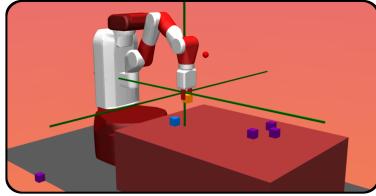


Figure 1. The **Modular Goal Fetch Arm** environment. An intrinsically motivated agent can set its own (modular) goals (Reach, Push, Pick and Place, Stack), with multiple objects and distractors.

and self-organize one's own curriculum. This challenge can be tackled within the framework of Intrinsically Motivated Goal Exploration Processes (IMGP) (Baranes & Oudeyer, 2013; Forestier et al., 2017), leveraging computational models of autonomous development in human infants.

Modular goal representation. In a same environment, an agent might want to ‘put the cube in position x ’ or to ‘reach position y ’ for any x or y . Here, describing the full goal space requires modular goal representations. Goals are organized by *modules*, where *module* refers to the pair of a reward function and a goal space $M_i = (R_{M_i, g_i \in \mathcal{G}_{M_i}}, \mathcal{G}_{M_i})$. The reward function describes a set of constraints that must be satisfied by the agent’s state (e.g. Reach), given a continuous parameter (e.g. $g_i = y$) evolving in the associated goal space (e.g. 3D Euclidean space), see Fig. 1.

While flat multi-goal problems with continuous (Schaul et al., 2015a; Andrychowicz et al., 2017; Plappert et al., 2018) or discrete goals (Mankowitz et al., 2018; Riedmiller et al., 2018) have been explored in the past, only few works tackle the problem of modular multi-goal learning (Forestier & Oudeyer, 2016), none in an RL setting. Here, we present CURIOUS¹, a modular multi-goal reinforcement learning (RL) algorithm that uses intrinsic motivations to efficiently learn a continuous set of diverse goals using modular goal representations. To build an algorithm able to learn modular goals, one must answer the following questions: 1) How to choose the action policy architecture? 2) How to select

¹ CURIOUS stands for Continual Universal Reinforcement learning with Intrinsically mOtivated sUbstitutionS.

the next module and goal to practice and learn about? 3) How to efficiently transfer knowledge between modules and goals?

Related work. Kaelbling (1993) proposed the first algorithm able to leverage cross-goal learning to address a discrete set of goals. For each goal, the algorithm learned a specific value function using Q-learning (*goal-experts* approach). More recently, Schaul et al. (2015a) proposed Universal Value Function Approximators (UVFA), a unique policy able to address an infinity of goals by concatenating the current state and goal to feed both the policy and the value function. In UNICORN, UVFA is used to address a discrete set of goals in parallel: reaching different objects in a visual world (Mankowitz et al., 2018). SAC-X implements multi-task RL where easy tasks are considered as auxiliary tasks to help learning about the hardest task (placing cubes inside a closed box) (Riedmiller et al., 2018). Here, one network is trained for each task and the collected transitions are shared (*goal-experts* approach). In other works from multi-task RL (Teh et al., 2017; Espeholt et al., 2018; Hessel et al., 2018), agents do not represent explicitly the current task and aim at maximizing the overall reward. Finally, within the Intrinsically Motivated Goal Exploration Processes framework (IMGEP), Forestier & Oudeyer (2016) proposed MACOB, an algorithm able to target modular goals using a population-based algorithm that mutates and replays controllers experienced in the past. MACOB maintains a population of solutions, one for each goal (*modular goal-experts* approach), see Nguyen & Oudeyer (2012) for a similar approach. This enables efficient cross-goal learning in high-dimensional goal spaces, but is limited by the memory-based representation of policies.

Multi-goal approaches prove better than simply training a policy per goal because knowledge can be transferred between different goals using off-policy learning and hindsight learning (Andrychowicz et al., 2017). Off-policy learning enables the use of any transition to improve the current policy: transitions collected from an older version of the current policy (Lillicrap et al., 2015), from a population of exploratory policies (Colas et al., 2018), or even from demonstrations (Večerík et al., 2017). Transitions collected while aiming at a particular goal can be used to learn about any other. With finite sets of goals, each transition is generally used to update the policy on every other goal (Mankowitz et al., 2018; Kaelbling, 1993). With continuous sets of goals, *imaginary goals* are sampled from the goal space (Andrychowicz et al., 2017). In the case of UVFA policies, this consists in the substitution of the goal that is part of the input by the imaginary one, a technique called *goal replay* or *goal substitution*. Building on UVFA, Andrychowicz et al. (2017) proposed Hindsight Experience Replay (HER), a method leveraging hindsight for transferring knowledge

between goals. The original goal of a transition can be substituted by any outcome experienced later in the trajectory (*imaginary goal*). This helps to increase the probability to observe rewards in reward-sparse environments.

In the literature, environments usually provide goals that the agent is asked to solve. In the IMGEP framework however, autonomous agents are intrinsically motivated to set their own, possibly learning their representation (Laversanne-Finot et al., 2018). Forestier & Oudeyer (2016) in particular, biased the selection of the next goal to attempt towards modules showing high absolute measures of learning progress (LP). This mechanism helps the agent to focus on learnable goals and to disengage from goals that are currently too hard or already solved. Veeriah et al. (2018) uses LP computed from Bellman errors for goal selection, but this form of LP does not improve over random goal selection.

Additional background can be found in the supplementary document. Table 1 presents a classification of the multi-goal approaches most related to our work.

Contributions. The contributions of this paper are:

1. A modular encoding of goals to enable learning of continuous sets of diverse goals within a single policy using UVFA (Reach, Push, Pick and Place, Stack). This enables to tackle different kinds of goals, each with their own continuous parameterization, and facilitates transfer between modules and goals. See Sec. 2.1.
2. An active strategy for cross-module goal replay. Off-policy learning enables to use any experience to learn about any goal from any module. We propose to guide the selection of module for replay using absolute learning progress measures (in addition to LP-based goal sampling to interact with environment). See Sec. 2.2.
3. From the IMGEP perspective, a single monolithic modular multi-goal action policy. This is an alternative to the population-based algorithms studied so far (Forestier & Oudeyer, 2016; Forestier et al., 2017) and provides the flexibility of RL methods.
4. An environment for modular goal RL. See Sec. 3.
5. Empirical comparisons to other architectures: a goal-parameterized RL with HER (flat multi-goal RL) and a multi-goal module-experts approach (one multi-goal expert per module). See Sec. 4.1.
6. A study of the self-organization of learning phases demonstrated by our algorithm (automatic curriculum). See Sec. 4.2.
7. Experimental evidence of robustness to *distracting goals, forgetting and body changes* in comparison with random goal selection. See Sec. 4.3 and 4.4.

2. CURIOUS

2.1. A Modular Multi-Goal Architecture using Universal Approximators

UVFA concatenates the goal of the agent with its current state to form the input of the policy and the value function implemented by deep neural networks (Schaul et al., 2015a). With CURIOUS, we propose a new encoding of goals using modular representations. This enables to target a rich diversity of modular goals within a single network (*modular multi-goal approach*), see Fig. 2. Given \mathcal{G}_{M_i} the goal space of module M_i , the current goal g is defined as a vector of dimension $|\mathcal{G}| = \sum_{i=1}^N |\mathcal{G}_{M_i}|$, where the \mathcal{G}_{M_i} can have different dimensionalities. g is set to 0 everywhere except in the indices corresponding to the current module M_i , where it is set to $g_i \in \mathcal{G}_{M_i}$. By masking the goal-inputs corresponding to unconsidered modules, the corresponding weights are frozen during backpropagation. In addition, a module descriptor m_d of size N (one-hot encoding) encodes the current module. The overall input to the policy network is $[s_t, g, m_d]$, see Fig. 2. We call this modular goal-parameterized architecture Modular-UVFA (M-UVFA).

In Fig. 2, we can see the underlying learning architecture (actor-critic). The actor implements the action policy and maps the input $[s_t, g, m_d]$ to the next action a_t . The action is then concatenated to a copy of the actor’s input to feed the critic $[s_t, g, m_d, a_t]$. The critic provides an estimate of the Q -value: $Q(s_t, a_t, g, m_d)$. The critic and the actor are then trained using DDPG (Lillicrap et al., 2015), although any other off-policy learning method could be used (e.g. TD3 (Fujimoto et al., 2018), or DQN for the discrete case (Mnih et al., 2013)). More details about DDPG can be found in the supplementary document or in Lillicrap et al. (2015).

2.2. Module and Goal Selection, Cross-Module Learning, Cross-Goal Learning

In UVFA, HER and UNICORN, the next goal to target is selected at random (Schaul et al., 2015a; Andrychowicz et al., 2017; Mankowitz et al., 2018). This is coherent with the common view that the agent must comply with the desires of an engineer and target the goal it is asked to target. Here

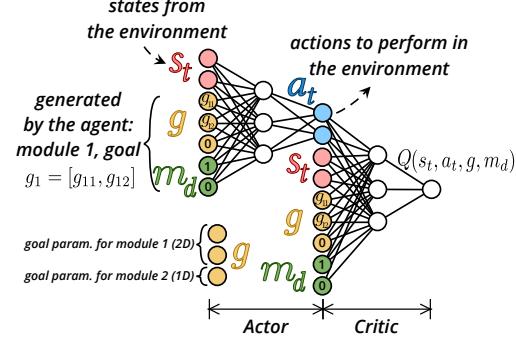


Figure 2. Modular goal-parameterized actor-critic architecture (M-UVFA). Toy example with 2 modules, parameterized by g_1 (2D) and g_2 (1D) respectively. Here, the agent is attempting goal g_1 in module M_1 , as specified by the one-hot module descriptor $m_d = (1, 0)$. The actor (left) computes the action a_t . The critic (right) computes the Q -value.

on the other hand, agents have the capacity to select which goal to target next. Because goals are not equivalent, the agent can benefit from intrinsic motivations towards LP (Schmidhuber, 1991; Kaplan & Oudeyer, 2004). This can be useful: 1) when there are distracting goals on which the agent cannot progress; 2) when some goals are already mastered. This idea comes from the IMGP framework and was used in Baranes & Oudeyer (2013) to guide goal selection and in Forestier & Oudeyer (2016) to guide module selection. The problem of selecting a module can be modeled as a non-stationary multi-armed bandit (MAB), where the value of each arm (module) is the current absolute LP. Learning progress (LP) is defined as the derivative of the agent’s competence on a particular module: $LP_{M_i} = \frac{dC_{M_i}}{dt}$, where the competence $C_{M_i} : t \rightarrow p_{\text{success}}(t)$ is the probability of success at time t . Here, the agent focuses its attention on modules for which it is making the largest absolute progress, and pays little attention to modules that are already solved or unsolvable, i.e. for which $|LP|$ stays small. Using the absolute value of LP also leads to the prioritization of modules for which the agent is showing decreasing performances. This helps to deal with forgetting: the agent reallocates learning resources to the modules being forgotten, Sec. 4.3.

Table 1. Classification of multi-goal approaches. Underlined: Algorithms internally generating goals (IMGP), (*) using LP-based intrinsic motivations. *Italic*: Population-based algorithms (non-RL). **Bold**: Algorithms proposed in this paper.

	<i>n GOALS, n POLICIES</i>		<i>n GOALS, 1 POLICY</i>
FLAT REPR.	GOAL-EXPERTS: (Kaelbling, 1993) SAC-X (Riedmiller et al., 2018) <u>SAGG-RIAC</u> * (Baranes & Oudeyer, 2013)		MULTI-GOAL: UVFA (Schaul et al., 2015a) HER (Andrychowicz et al., 2017) UNICORN (Mankowitz et al., 2018)
MODULAR REPR.	MOD-GOAL-EXPERTS: <u>MACOB</u> * (Forestier & Oudeyer, 2016) MULTI-GOAL MODULE-EXPERTS: <u>MG-ME</u>		MODULAR-MULTI-GOALS: M-UVFA , <u>CURIOS</u> *

Learning Progress Estimation. Since an autonomous agent is not externally provided its true competence or LP, it needs to approximate them for each module. To measure its competence, it uses some episodes (with $p_{eval} = 0.1$) to evaluate itself on random modules and targets without exploration noise. The results (success 1 or failure 0) of these rollouts are stored in competence queues $results^{(i)}$ for all M_i . In a similar way as Forestier & Oudeyer (2016), the agent computes its subjective competence as

$$C_{M_i}(n_{eval}^{(i)}) = \frac{1}{l} \sum_{j=0}^{l-1} results^{(i)}(n_{eval}^{(i)} - j),$$

where $n_{eval}^{(i)}$ is the number of self-evaluation rollouts performed by the agent in module M_i . The subjective LP_{M_i} after $n_{eval}^{(i)}$ self-evaluation rollouts is then computed as:

$$LP_{M_i}(n_{eval}^{(i)}) = C_{M_i}(n_{eval}^{(i)}) - C_{M_i}(n_{eval}^{(i)} - l).$$

Given the subjective LP measures, we tackle the multi-armed bandit problem by implementing a simple approach called proportional probability matching, with an additional ϵ -greedy strategy for exploration. More precisely, we compute the LP probabilities $p_{LP}(M_i)$ as:

$$p_{LP}(M_i) = \epsilon \times \frac{1}{N} + (1 - \epsilon) \times \frac{|LP_{M_i}|}{\sum_{j=1}^N |LP_{M_j}|},$$

where N is the number of modules. The ratio ϵ implements a mixture between random exploration of modules (left term) and exploitation through a biased selection/replay of modules (right term). The random exploration term enables sampling modules that do not show any LP (i.e. already solved, too hard, or at a plateau). This way, the agent can check that it stays competent on modules that are already learned, or can insist on modules that are currently too hard.

Note that we use LP for two distinct purposes: 1) Before data collection, to select the module from which to draw the next goal to attempt in the environment; 2) Before training, to select the substitute module descriptor (module replay). Recall that, once transitions are sampled from the replay buffer, they can be modified (replayed) by substituting the original module descriptor (or goal) by a new one. The substitute module is the one the agent is going to learn about. When replaying a particular module more than others, the agent allocates more resources to that module. While the use of LP for module selection is not new (Forestier & Oudeyer, 2016), we are the first to consider its use for cross-module goal replay.

Module and Goal Selection. Before interacting with the environment, the agents selects the next goal to target by first sampling a module from \mathcal{M} using p_{LP} , and second, sampling the goal uniformly from the corresponding goal space \mathcal{G}_{M_i} .

Cross-Module and Cross-Goal Learning. In an example with three modules, an agent computed $p_{LP} = [0.6, 0.2, 0.2]$. The agent uses these probabilities to guide learning towards modules with high absolute LP. If the size of the minibatch is N_{mb} , the agent will sample $\lfloor N_{mb} \times 0.6 \rfloor$ transitions relevant to module 1, $\lfloor N_{mb} \times 0.2 \rfloor$ transitions relevant to module 2 etc. A transition that is *relevant for module M_i* (e.g. Push module), means that it comes from an episode during which the corresponding outcome has changed (e.g. cube position). This sampling bias towards “eventful” transitions is similar to Energy-Based Prioritization (Zhao & Tresp, 2018) (see supp. doc.). In this minibatch, every transition has been sampled to train on a specific module (e.g. m_d^*), although it could have been collected while targeting another module (e.g. m_d). To perform this cross-module learning, we simply substitute the latter by the former. Goal substitution is then performed using hindsight, which means the goal g of a transition is sometimes ($p = 0.8$) replaced by an outcome reached later in the same episode g^* (Andrychowicz et al., 2017).

Internal Reward. After module descriptors and goals have been substituted, the agent computes an internal reward for each transition using a reward function parameterized by the new m_d^* and goal g^* . Thus it answers: *What would have been my reward for experiencing this transition, if I were aiming at that imagined goal from that imagined module?* The reward is non-negative (0) when the outcome satisfies the constraints described by the imagined module m_d^* , relative to the imagined g^* ; negative otherwise (-1). In a reaching module for instance (see Fig. 1), a positive reward is generated when the Euclidean distance between the 3D target (goal) and the gripper (outcome) falls below a precision parameter ϵ_{reach} (reward constraint associated to the reaching module).

2.3. Combining Modular-UVFA and Intrinsically Motivated Goal Exploration

A schematic view of CURIOUS is given in Fig. 8. The detailed algorithm is given in the supplementary document.

1. **Module and goal selection.** The agent selects module M_i and goal g_i for the next rollout (blue), respectively sampled from the set of potential modules \mathcal{M} using p_{LP} (purple), and uniformly from the corresponding goal space \mathcal{G}_{M_i} .
2. **Data collection.** The agent interacts with the environment using its current M-UVFA policy (grey), collects transitions and stores them in memory (red).
3. **LP update.** If it was a self-evaluation rollout, the agent updates its measures of competence, LP and p_{LP} given the new result (success or failure, purple).

4. **Module and goal substitution.** The agent decides on which modules and goals to train. To update the policy and critic, the algorithm first samples a minibatch from the replay buffers (red) using p_{LP} and implements module and goal substitutions to perform cross-module and cross-goal learning (orange), see Sec. 2.2.
5. **Internal reward.** The agent computes its reward r for each transition, using $R_{M,g}$ parameterized by the substitute module m_d^* and goal g^* (brown).
6. **RL updates.** The agent updates its policy and value function with DDPG using the modified minibatch (green).

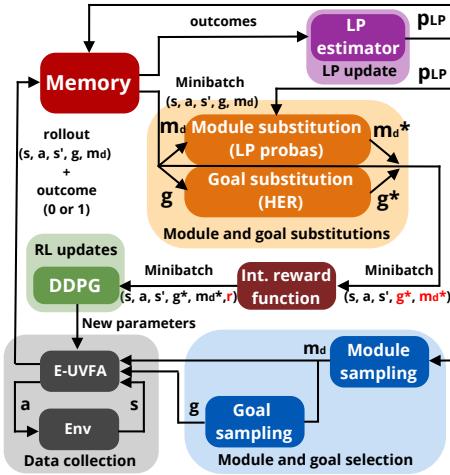


Figure 3. Schematic view of CURIOUS.

2.4. Evaluation Methodology

The performance of the agents are evaluated offline in terms of success rates over sets of achievable goals (defined by the experimenter). Every point of a learning curve represents the success rate over 95 offline evaluation rollouts (5×19 actors), using random achievable goals. Evaluation is completely independent from training, i.e. agents cannot train on evaluation transitions. We use the non-parametric one-tail Mann-Whitney U-test with confidence level $\alpha = 0.01$ for all comparisons. More details and justifications can be found in the supplementary document.

3. A Modular Goal Environment

Modular Goal Fetch Arm is a new simulated environment adapted from the OpenAI Gym suite (Brockman et al., 2016). The agent is embodied by a robotic arm facing 2 cubes randomly positioned on a table. The agent controls the position of its gripper and the gripper opening (4D). It can target a diverse set of modular goals: (M_1) *Reach* a 3D

target with the gripper; (M_2) *Push* cube 1 onto a 2D target on the table; (M_3) *Pick and Place* cube 1 on a 3D target; (M_4) *Stack* cube 1 over cube 2. Additional Push modules concerning additional out-of-reach and moving cubes can be defined (impossible, distracting goals). Further details can be found in the supplementary document.

4. Experiment and Results

In this section, we present ablative studies to assess the relative importance of: 1) the policy and value function architecture and 2) the use of intrinsically motivated module selection for practice and replay. We call M-UVFA the algorithm using a modular goal-parameterized policy and random module choices, while the intrinsically motivated version is called CURIOUS. We do not investigate the efficiency of HER or the efficiency of the sampling bias towards interesting transitions as they were already studied in Andrychowicz et al. (2017); Plappert et al. (2018) and Zhao & Tresp (2018) respectively. For fair comparisons, we apply both mechanisms to all the tested algorithms.

4.1. Impact of Policy and Value Function Architecture

Experiments. In this section, we investigate the impact of using an M-UVFA architecture for the policy and value function. The module-set is composed of four achievable modules and four distracting modules. We test this algorithm against two baselines:

1. A *flat multi-goal* architecture (HER). This algorithm does not represent goals in a modular fashion but in a linear way. The corresponding goal is selected uniformly inside \mathcal{G} , a holistic goal space such that $\mathcal{G} = \prod_{i=1}^N \mathcal{G}_{M_i}$. To generate a reward, the agent needs to satisfy the constraints described by all the modules at once. This goal-parameterized architecture is equivalent to UVFA, which makes the algorithm equivalent to HER +DDPG.
2. A *multi-goal module-experts* architecture (MG-ME) where an expert multi-goal policy is trained for each of the N modules. Each policy is trained one epoch every N on its designated module and shares the collected transitions with other experts. When evaluated on a particular module, the algorithm uses the corresponding module-expert.

Results. Fig. 5 shows the evolution of the average success rate computed over achievable goals for M-UVFA and the two baselines described above. The learning curve of HER stays flat. This can be easily understood as none of the goals expressed in the complete goal space \mathcal{G} corresponds to a real situation (e.g. the agent cannot reach a 3D target with its gripper while placing a cube at another). The agent cannot

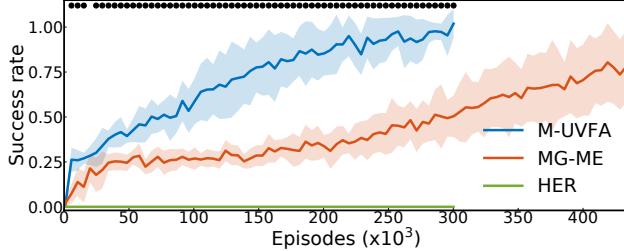


Figure 5. Impact of the policy and value function architecture. Average success rates computed over achievable goals. Mean \pm std over 10 trials are plotted, while dots indicate significance when testing M-UVFA against MG-ME.

fulfill the constraints of all modules simultaneously, thus receives no reward. This motivates the use of a modular representation with separated modules. Comparing MG-ME and M-UVFA, we can see that the achievable goals are learned much faster in the multi-modular-goals approach (one, policy, $\approx 250 \cdot 10^3$ vs. $\approx 450 \cdot 10^3$ episodes). From now on, all experiments use the M-UVFA architecture.

4.2. Visualizing the Intrinsic Motivation towards Learning Progress

Experiments. This section aims at showing the inner working of CURIOUS’s intrinsic motivation towards LP. Here we focus on a setting with four achievable modules (Reach, Push, Pick and Place, and Stack).

Results. Fig. 4(a) shows the evolution of the module-dependent competence measures as subjectively perceived by the agent, while Fig. 4(b) shows the evolution of the corresponding LP measures. Finally, Fig. 4(c) shows the corresponding module selection probabilities p_{LP} , a mixture of random selection with probability ϵ and active selection proportional to LP measures with probability $1 - \epsilon$. These figures demonstrate the existence of successive learning phases, that can be interpreted as developmental phases (Oudeyer & Smith, 2016). The robot first learns how to

control its gripper (M_1), then to push objects on a desired target on the table (M_2) before it learns how to place the cube on a 3D target (M_3) and how to stack the two cubes (M_4). Fig. 4(b) shows that LP stays small for modules that are already solved (e.g. M_1 after 10^4 episodes) or too hard to solve (e.g. M_3 and M_4 before $35 \cdot 10^3$ episodes), and increases when a module is being learned. We further discuss the link between these learning phases, developmental learning and curriculum learning in the supplementary document.

4.3. Impact of the Intrinsic Motivation: Resilience to Forgetting and Sensor Perturbations

Experiments. During learning, the agent can forget about a previously mastered module. This can happen because is not targeting it often (catastrophic forgetting), because of environmental changes (e.g. icy floor) or because of body changes (e.g. sensor failure). Ideally, CURIOUS should be able to detect and react when such situations arise. This section investigates the resilience of our algorithm to such perturbations and compares it to the M-UVFA baseline.

We first look at a run where forgetting occurs and explain how CURIOUS detects the situation and reacts. Since forgetting cannot be triggered, we add more emphasis to a second experiment, where we simulate a time-locked sensory failure. We present the following setup to the agent: first, it learns about a set of 4 modules (Reach, Push, Pick and Place for cube 1, and Push for cube 2). Then, a sensory perturbation is triggered at a precise time ($epoch = 250$, $episode = 237.5 \cdot 10^3$) such that the perception of cube 2 gets shifted by 0.05 (simulation units) until the end of the run. The performance on this module suddenly drops and we compare the recoveries of CURIOUS and M-UVFA.

Results - Forgetting. Looking at Fig. 4(a), we can observe a drop in the competence on M_3 around episode $80 \cdot 10^3$. This phenomenon is usually described as *catastrophic forgetting*: because it is trained on other modules, the network can forget about the previously mastered mod-

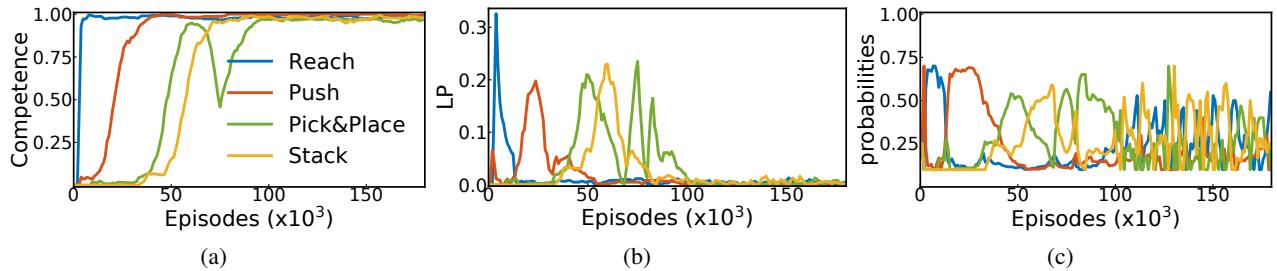


Figure 4. Visualization of a single run. a: Module-dependent subjective measures of competence for CURIOUS (1 run). b: Corresponding module-dependent subjective measures of absolute LP. c: Corresponding probabilities p_{LP} to select modules to practice or to learn about.

ule M_3 , without any obvious reason. The corresponding period of Fig. 4(b) shows an increase in LP for M_3 , which in turn triggers an additional focus of the agent towards that module (see the corresponding probability increase in Fig. 4(c)). Using LP to bias its attention, the agent monitors its competence on the modules and can react when it forgets about a previously mastered module. This mechanism helps to deal with the problem of forgetting and facilitates learning of multiple modules in parallel. To prove its efficiency, we need to compare CURIOUS to its baseline M-UVFA using a time-locked perturbation.

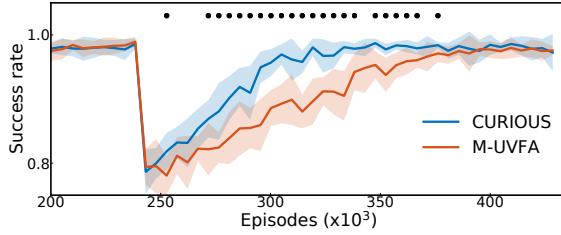


Figure 6. Impact of the intrinsic motivation towards LP for sensory failure recovery. Mean success rates over the four modules \pm std over 10 trials are plotted. The dashed line indicates the onset of the perturbation, while the dots indicate significance when testing CURIOUS against M-UVFA.

Results - Sensor Perturbation. In Fig. 6, we can observe the drop in average success rate after the perturbation (around $240 \cdot 10^3$ episodes). This perturbation only affects one of the four modules (Push cube 2), which triggers a drop in the average performance of about $1/4^{th}$. As described above, CURIOUS agents detect that perturbation and react by using more transitions to improve on the corresponding module. This translates into a significantly faster recovery when compared to M-UVFA. The agents recover 95% of their pre-perturbation performance in 78 and $43 \cdot 10^3$ episodes (random and active respectively), which translates in a 45% faster recovery for CURIOUS ($p < 10^{-4}$), see Fig. 6.

4.4. Impact of the Intrinsic Motivation: Resilience to Distracting Modules

Experiments. In this section, we investigate the resilience of our learning algorithm when the number of distracting modules increases (0, 4, 7). The agent faces four achievable modules in addition to the distracting modules. The distracting modules are all Push modules relative to the randomly moving and out-of-reach cubes. The agent receives extra noisy inputs corresponding to the random movements of these cubes.

Results. In Fig. 7, we see that the number of distracting modules faced by the agents highly impacts their learn-

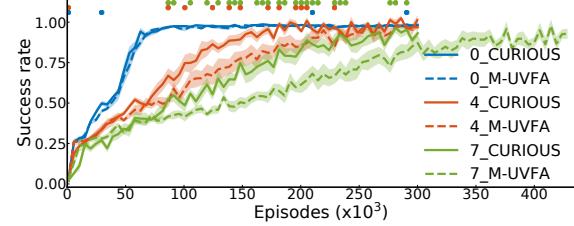


Figure 7. Impact of the intrinsic motivation towards LP when the number of distracting modules grows. Average success rates computed over achievable goals when the number of distracting modules increases (0, 4, 7). Mean and standard error of the mean (for visualization purpose) over 10 trials are plotted. The dots indicate significance when testing CURIOUS against M-UVFA.

ing speed on achievable goals. In particular, M-UVFA random agents do not know that these goals are impossible to achieve and waste time and resources trying to improve on them. Since these agents sample distracting modules just like others, we can expect the learning speed to be scaled by $\frac{\#achievablemodules}{\#modules}$. On the other hand, CURIOUS agents try to learn which modules are too difficult at the moment to target them less often. Note that CURIOUS agents still need to choose them sporadically to keep updated measures of their LP: they sample a random module with probability ϵ . In Fig. 7, we see that the advantage of CURIOUS over its random counterpart increases as the number of distracting modules grows (see colored dots indicating significant differences). Although the addition of distracting modules might sound a bit ad-hoc here, it is important to note that autonomous agents evolving in the real world face numerous modules such as these. For humans, quantity of potential modules are impossible (predicting the movement of leaves on a tree, trying to walk on walls etc.). Just as humans, artificial agents need to discard them based on experience and LP.

5. Discussion

Leveraging Environment Modularity. In some environments, representing all the potential goals requires modular representations. Because Sec. 4.1 proved that a simple UVFA architecture could not deal with this situation, we proposed M-UVFA. Note that, although our Modular Goal Fetch-Arm environment only contains goals that can be represented in a modular way, M-UVFA can also target discrete sets of goals using flat representations (by setting the goal g_i of module M_i to the null vector and letting m_d encode for the goal). In short, M-UVFA enables traditional UVFA to target a richer diversity of goals than what was possible with traditional UVFA implementations.

Pros and Cons of Monolithic Policies. As noted in Mankowitz et al. (2018), representations of the world state

are learned in the first layers of a neural network policy/value function. A representation learned to achieve goals from one module could probably be useful for learning goals from another similar module. Our monolithic modular goal policy leverages that fact, by re-using subparts of the same network to learn different but similar modules and goals. This might explain why M-UVFA outperforms the *multi-goal module-experts* (MG-ME) policy architecture (Fig. 5). However, such monolithic policies are more prone to forgetting. Although this phenomenon is partially mitigated by the use of the absolute value of LP, it might still be an issue when the number of potential modules increases. To answer this problem, we could think of combining several M-UVFA policies for different subsets of modules.

A Monolithic IMGP. Contrary to the vision shared by many multi-goal RL papers where agents must comply to the engineer desires (*do goal 1, do goal 3 ...*), our work takes the perspective of agents empowered by intrinsic motivations to choose their own goals (*do whatever you want, but be curious.*). This vision comes from the IMGP framework which defines agents able to set their own parameterized problems to explore their surrounding and master their environment (Forestier et al., 2017). Contrary to previous IMGP algorithms grounded on memory-based representations of policies, CURIOUS uses a single monolithic policy for all modules and goals (M-UVFA). Because it is memory-based, MACOB does not handle well the variety of initial states which limits its generalization capacity.

Active Learning using Learning Progress. Although LP-based module selection already brings significant advantages compared to random module selection, CURIOUS could benefit from a more advanced LP estimator. Our current estimator uses moving averages. It is fast and requires small amounts of memory, but could be more reactive to changes in true LP. This delay causes the agent to persevere on modules that are already mastered, or not to react quickly to newly learnable modules. These drawbacks could be mitigated with more advanced measures of competence or LP (e.g. approximate Bayesian methods like in Mathys et al. (2011)).

6. Further Work

Hierarchical Extension. The idea of using a high-level policy to select goals for a lower-level policy was also studied in the field of *hierarchical RL*. Yet, while hierarchical RL agents choose their own subgoals, they usually do so to achieve higher-level goals imposed by the engineer (Vezhnevets et al., 2017; Nachum et al., 2018; Levy et al., 2018). A natural extension of our work could be to replace our high-level MAB module selection policy by another CURIOUS agent targeting self-generated higher-level goals, in a

hierarchical manner.

Learning a Goal Selection Policy. In this work we provide the policy for goal sampling inside modules: sampling uniformly from a pre-defined (reachable) goal space. In the future, the agents could learn it autonomously using adaptations of existing algorithms such as SAGG-RIAC (Baranes & Oudeyer, 2013) or GOAL-GAN (Held et al., 2017). SAGG-RIAC enables to split recursively a wide continuous goal space and to focus on sub-regions where LP is higher, while GOAL-GAN proposes to generate goals of intermediate difficulty using a Generative Adversarial Network.

Learning Representations for Modules and Goals. Another assumption of our work, is that agents should already know a modular representation of goals and their modules. Although modules and goal spaces were hand-defined in the experiments of this paper, this was a scaffolding for the studies we presented. In a general IMGP setting, autonomous agents must be able to construct their own set of modules and goal representations. The idea of autonomously learning modular goal representations from experience has been explored in Laversanne-Finot et al. (2018), using β -VAEs. This was used for goal exploration using a population-based IMGP algorithm. Combining CURIOUS to this unsupervised learning of disentangled goal spaces is an interesting avenue to explore, in the quest of more autonomous learning agents.

7. Conclusion

This paper presents CURIOUS, a learning algorithm that combines an extension of UVFA to enable modular goal RL in a single policy (M-UVFA), and active mechanisms that bias the agent’s attention towards modules where the absolute LP is maximized. This self-organizes distinct learning phases, some of which are shared across agents, others dependent on the agent experience. With this mechanism, agents spend less time on impossible modules and focus on achievable ones. It also helps to deal with forgetting, by refocusing learning on modules that are being forgotten because of model faults, changes in the environment or body changes (e.g. sensory failures). This mechanism is important for autonomous continual learning in the real world, where agents must set their own goals and might face goals with diverse levels of difficulty, some of which might be required to solve others later on.

Links. The [environment](#), [code](#) and [video](#) of the CURIOUS agent are made available at <https://github.com/flowersteam/curious>.

Acknowledgments

Cédric Colas is partly funded by the French Ministère des Armées - Direction Générale de l'Armement. Olivier Sigaud is partly funded by the European Commission, within the DREAM project. The DREAM project has received funding from the European Unions Horizon 2020 research and innovation program under grant agreement N° 640891.

References

- Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, O. P., and Zaremba, W. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pp. 5048–5058, 2017.
- Baranes, A. and Oudeyer, P.-Y. Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems*, 61(1):49–73, 2013.
- Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.
- Colas, C., Sigaud, O., and Oudeyer, P.-Y. GEP-PG: Decoupling exploration and exploitation in deep reinforcement learning algorithms. *arXiv preprint arXiv:1802.05054*, 2018.
- Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*, 2018.
- Forestier, S. and Oudeyer, P.-Y. Modular active curiosity-driven discovery of tool use. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pp. 3965–3972. IEEE, 2016.
- Forestier, S., Mollard, Y., and Oudeyer, P.-Y. Intrinsically motivated goal exploration processes with automatic curriculum learning. *arXiv preprint arXiv:1708.02190*, 2017.
- Fujimoto, S., van Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.
- Held, D., Geng, X., Florensa, C., and Abbeel, P. Automatic goal generation for reinforcement learning agents. *arXiv preprint arXiv:1705.06366*, 2017.
- Hessel, M., Soyer, H., Espeholt, L., Czarnecki, W., Schmitt, S., and van Hasselt, H. Multi-task deep reinforcement learning with popart. *arXiv preprint arXiv:1809.04474*, 2018.
- Kaelbling, L. P. Learning to achieve goals. In *IJCAI*, pp. 1094–1099. Citeseer, 1993.
- Kaplan, F. and Oudeyer, P.-Y. Maximizing learning progress: an internal reward system for development. In *Embodied artificial intelligence*, pp. 259–270. Springer, 2004.
- Laversanne-Finot, A., Pérez, A., and Oudeyer, P.-Y. Curiosity driven exploration of learned disentangled goal spaces. In *Proceedings of CoRL 2018 (PMLR)*, 2018.
- Levy, A., Platt, R., and Saenko, K. Hierarchical reinforcement learning with hindsight. *arXiv preprint arXiv:1805.08180*, 2018.
- Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.
- Mankowitz, D. J., Žídek, A., Barreto, A., Horgan, D., Hessel, M., Quan, J., Oh, J., van Hasselt, H., Silver, D., and Schaul, T. Unicorn: Continual learning with a universal, off-policy agent. *arXiv preprint arXiv:1802.08294*, 2018.
- Mann, H. B. and Whitney, D. R. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, pp. 50–60, 1947.
- Mathys, C., Daunizeau, J., Friston, K. J., and Stephan, K. E. A bayesian foundation for individual learning under uncertainty. *Frontiers in human neuroscience*, 5:39, 2011.
- Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.
- Nachum, O., Gu, S., Lee, H., and Levine, S. Data-efficient hierarchical reinforcement learning. *arXiv preprint arXiv:1805.08296*, 2018.
- Nguyen, S. M. and Oudeyer, P.-Y. Active choice of teachers, learning strategies and goals for a socially guided intrinsic motivation learner. *Paladyn*, 3(3):136–146, 2012.
- Oudeyer, P.-Y. and Smith, L. B. How evolution may work through curiosity-driven developmental process. *Topics in Cognitive Science*, 8(2):492–502, 2016.
- Plappert, M., Andrychowicz, M., Ray, A., McGrew, B., Baker, B., Powell, G., Schneider, J., Tobin, J., Chociej, M., Welinder, P., et al. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*, 2018.

Riedmiller, M., Hafner, R., Lampe, T., Neunert, M., Degrave, J., Van de Wiele, T., Mnih, V., Heess, N., and Springenberg, J. T. Learning by playing-solving sparse reward tasks from scratch. *arXiv preprint arXiv:1802.10567*, 2018.

Schaul, T., Horgan, D., Gregor, K., and Silver, D. Universal value function approximators. In *International Conference on Machine Learning*, pp. 1312–1320, 2015a.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015b.

Schmidhuber, J. Curious model-building control systems. In *Neural Networks, 1991. 1991 IEEE International Joint Conference on*, pp. 1458–1463. IEEE, 1991.

Teh, Y., Bapst, V., Czarnecki, W. M., Quan, J., Kirkpatrick, J., Hadsell, R., Heess, N., and Pascanu, R. Distral: Robust multitask reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 4496–4506, 2017.

Večerík, M., Hester, T., Scholz, J., Wang, F., Pietquin, O., Piot, B., Heess, N., Rothörl, T., Lampe, T., and Riedmiller, M. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

Veeriah, V., Oh, J., and Singh, S. Many-goals reinforcement learning. *arXiv preprint arXiv:1806.09605*, 2018.

Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., and Kavukcuoglu, K. Feudal networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1703.01161*, 2017.

Zhao, R. and Tresp, V. Energy-based hindsight experience prioritization. *arXiv preprint arXiv:1810.01363*, 2018.

Additional Background

DDPG. Deep Deterministic Policy Gradient (DDPG) is an off-policy model-free RL algorithm for continuous action spaces (Lillicrap et al., 2015). It concurrently learns a policy π (the actor) and a Q -function Q (the critic) using neural networks. The actor implements the controller and maps the current state to the next action: $\pi : \mathcal{S} \rightarrow \mathcal{A}$. The critic approximates the optimal action-value function Q^* , $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$.

For exploration purposes, transitions are collected using a noisy version of the policy (behavioral policy), e.g. using Gaussian noise on the actions. The transitions are stored in a replay buffer of finite size. The actor and critic updates are then conducted using transitions sampled uniformly from

the replay buffer. The critic is trained to minimize the mean-squared Bellman error such that:

$$L(\phi, \mathcal{D}) = E_{t \sim \mathcal{D}} \left[\left(Q_\phi(s, a) - (r + \gamma \max_{a'} Q_\phi(s', a')) \right)^2 \right],$$

where ϕ represents the parameters of the critic, \mathcal{D} is the dataset, $t = (s, a, r, s')$ is a transition with r a reward. The actor is trained so as to maximize the output of the critic. Further details can be found in the original paper (Lillicrap et al., 2015).

UVFA. Universal Value Function Approximators (UVFA) is an extension of the traditional actor and critic networks to target multiple goals (Schaul et al., 2015a). The goal is represented by a vector defined in space \mathcal{G} . It is concatenated both to the input of the actor and to the input of the critic such that: $\pi : \mathcal{S} \times \mathcal{G} \rightarrow \mathcal{A}$ for the actor and $Q : \mathcal{S} \times \mathcal{G} \times \mathcal{A} \rightarrow \mathbb{R}$ for the critic. The reward function as well, is now parameterized by the goal: $R_g : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. The original paper shows that a UVFA architecture trained with Deep Q-Network (DQN) can generalize to previously unseen goals (Schaul et al., 2015a). Further details can be found in the original paper.

HER. Hindsight Experience Replay (HER) proposes an efficient mechanism leveraging hindsight to learn more efficiently in multi-goal settings (Andrychowicz et al., 2017). When an agent makes an attempt towards a goal and fails to reach it, HER proposes to probabilistically substitute the goal vector by an outcome achieved later in the trajectory. In simpler words, the agents *pretends* it was trying to reach some other goal it actually reached later on, hence it increases the probability to observe a reward. Note that this mechanism requires the agent to have access to the true reward function R_g and to be able to compute the reward corresponding to any combination of goal, state and action vectors. This mechanism can be used with any off-policy algorithms and was initially presented with DQN and DDPG. Further details can be found in the original paper (Andrychowicz et al., 2017).

Additional Methods

Overview. Algorithm 1 and Fig. 8 present the pseudo-code. We go through the different steps:

1. **Module and goal selection.** The agent selects module M_i and goal g_i for the next rollout (blue in Fig. 8). First, it decides whether it will perform self-evaluation, using probability p_{eval_c} [line 4 in the algorithm]. If it does so, it selects a module M_i at random. Otherwise,

the next module is sampled from the set of modules \mathcal{M} according to the LP probabilities p_{LP} (purple). The goal is sampled uniformly from the corresponding goal space \mathcal{G}_{M_i} [line 5]. All the considered goal spaces are either the 2D Euclidean space (for the *Push module*, with $z = \text{height}(\text{table})$) or the 3D Euclidean space (for other modules). These spaces are simply subspaces of the state space $\mathcal{G}_{\subset \mathcal{S}}$.

2. **Data collection.** The agent interacts with the environment and collects transitions (grey) using its current M-UVFA policy [lines 8 to 13] before storing them in memory [line 15]. If it is performing self-evaluation it does not use exploration noise.
3. **LP update.** If the agent was performing self-evaluation, it can now update its measures of LP given the new result (success or failure, purple) [lines 17 and 18]. It computes the corresponding probability measures p_{LP} using an ϵ -greedy version of the proportional probability matching method such that:

$$p_{LP}(M_i) = \epsilon \times \frac{1}{N} + (1 - \epsilon) \times \frac{|LP_{M_i}|}{\sum_{j=1}^N |LP_{M_j}|},$$

where LP_{M_i} is the learning progress of module M_i , and ϵ implements a mixture between random exploration (left-term) and exploitation guided by the absolute learning progress (right term).

4. **Module and goal substitution.** The agent decides on which modules and goals to train. To update the policy and critic, the algorithm first samples a minibatch from the replay buffers (red) using p_{LP} , see *Prioritized Interest Replay* below. If it wants to train on module M_i , it samples a transition relevant to module M_i and substitutes the original module descriptor stored in the transition (m_d) by the module descriptor corresponding to M_i , (m_d^* in the figure). With probability $p = 0.8$, it also substitutes the current goal g by an outcome achieved later in the episode using hindsight (g^* in the figure). These substitutions enable cross-module and cross-goal learning [line 20].
5. **Internal reward.** The agent computes its reward r for each transition, using the internal reward function $R_{M,g}$ parameterized by the substitute module description (m_d^*) and the substitute goal (g^*) (brown) [line 21].
6. **RL updates.** The agent updates its policy and value function using DDPG (green) [line 22].

Prioritized Interest Replay. In *Prioritized Experience Replay* (PER), Schaul et al. (2015b) suggests that RL agents can learn more efficiently from some transitions than from others. They propose to bias transition replay towards transitions with high temporal-difference (TD) error. In multi-goal settings, Zhao & Tresp (2018) proposes to bias sam-

Algorithm 1 CURIOUS

```

1: Input: env,  $\mathcal{M}$ ,  $\mathcal{G}_{1:N}$ , noise, internal_reward( ),  $p_{eval\_c}$ 
2: Initialize: policy, memory,  $p_{LP}$ 
3: repeat
4:    $eval\_c \leftarrow \text{random}() < p_{eval\_c}$ 
5:    $goal, m_d(M_i) \leftarrow \text{ModuleGoalSelector}()$ 
6:    $s_0, outcome_0 \leftarrow \text{env.reset}()$ 
7:   for  $t = 0 : N_t$  do
8:      $policy\_input \leftarrow \text{concatenate}(s_t, m_d, goal)$ 
9:      $a_t \leftarrow policy(policy\_input)$ 
10:    if not  $eval\_c$  then
11:       $a_t \leftarrow a_t + noise$ 
12:    end if
13:     $s_{t+1}, outcome_{t+1} \leftarrow \text{env.step}(a_t)$ 
14:   end for
15:    $memory.add(s, a, s', goal, outcome, m_d)$ 
16:   if  $eval\_c$  then
17:      $memory.add(outcome == goal, m_d)$ 
18:      $p_{LP} \leftarrow \text{LP\_Update}(memory)$ 
19:   end if
20:    $modified\_batch \leftarrow \text{ModuleGoalReplayPolicy}(p_{LP}, memory)$  {Use Prioritized Interest Replay,  $p_{LP}$  and HER }
21:    $modified\_batch \leftarrow \text{RewardComputer}(modified\_batch, \text{internal\_reward}())$ 
22:    $policy \leftarrow \text{RL\_Updates}(modified\_batch)$ 
23: until learning over
    
```

$\{\mathcal{M}: \text{set of } N \text{ modules}, \mathcal{G}_i: \text{goal space of module } M_i\}$
 $\{\text{random, empty and uniform respectively}\}$

$\{\text{The agent evaluates its competence if True}\}$
 $\{M_i \sim p_{LP} (\text{or uniform if } eval_c), goal \sim \mathcal{U}(\mathcal{G}_{M_i})\}$

$\{\text{If self evaluation, update LP with new result}\}$

$\{\text{Using DDPG }\}$

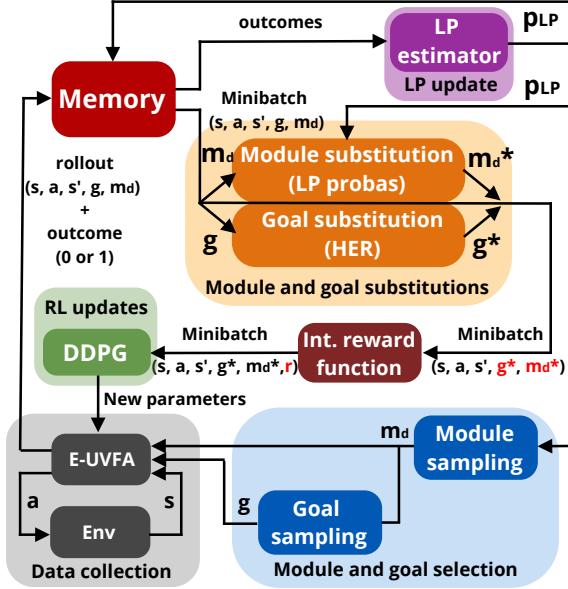


Figure 8. Schematic view of CURIOUS.

pling towards episodes of higher energy, where energy is the sum of the potential energy, the kinetic energy and the rotational energy of the considered object (Energy-Based Prioritization or EBP). They show that EBP performs higher than PER in their experiments. Here, we propose to apply a simpler version of EBP that we call *Prioritized Interest Replay*. To learn about module M_i (e.g. *Push(cube 1)* module), we only use transitions from episodes where the corresponding outcome (e.g. cube 1 position) has changed during the episode. This is an heuristic modeling attention towards *interesting* episodes relative to the module at hand. Note that, contrary to PER where transitions are prioritized based on a criterion at the transition level (TD-error of the transition), we apply a criterion at the episode level (did the outcome change during the episode?), as in EBP. Although we pick interesting episodes, we still sample uniformly from those. We assume this heuristic to induce a small sampling bias and do not use any correction by importance sampling as it was done in PER. We apply that heuristic to all algorithms in our paper.

After the interaction with the environment, transitions are stored in $N + 1$ buffers. A transition is stored in buffer i when the outcome corresponding to M_i has changed during the episode. The additional buffer collects the remaining episodes. Before training, the agent decides how to allow its computational resources among the different modules using p_{LP} . If the size of the minibatch is N_{mb} , the agent will sample $\lfloor N_{mb} \times p_{LP}(M_i) \rfloor$ transitions from buffer i , this for all modules M_i . In this minibatch, every transition has been sampled to train on a specific module (m_d^*), although

it could have been collected while targeting another module (m_d). To perform this cross-module learning, we simply substitute the module descriptor of each transition m_d by m_d^* .

Internal Rewards. The internal reward function is parameterized by both the module and the goal $R_{M,g}$. It is positive ($r = 0$) when the constraints defined by the modules are satisfied for the current goal and outcome. For the *Push* and *Pick and Place* modules, the reward is positive when the distance between the cube and the target position is within $\epsilon_{reach} = 0.05$ (simulation units), negative otherwise. For the *Reach* module, the same criterion is applied to the distance between the gripper and the goal. Finally, for the *Stack* module, in addition to the constraints on the goal-cube distance, the goal-gripper distance must stay larger than $1.2 \times \epsilon_{reach}$, to ensure that the two cubes are actually stacked and not held by the gripper. With this example we see that a module simply defines a set of constraints. Although here it is applied to a single state, we could imagine constraints defined over multiple time-steps (e.g. keep the cube above the table for more than 2 seconds).

Environment

Modular Goal Fetch Arm is a new simulated environment based on the Fetch environments included in the OpenAI Gym suite (Brockman et al., 2016). The agent is embodied by a 7-DoF Fetch robotic arm facing 2 cubes randomly positioned on a table and can target a large set of diverse goals. Additional out-of-reach cubes can be added as distracting modules. The agent controls the 3D Cartesian position of its gripper in velocity as well as its two-fingered parallel gripper. The agent can target several modules of goals: (M_1) reaching a 3D target with the gripper; (M_2) reaching a 2D target on the table with cube 1; (M_3) reaching a 3D target with cube 1; (M_4) stacking cube 1 over cube 2; (M_5) reaching a 2D target on the table with one of the randomly moving cube which are out of reach (distracting module). The observation space has 40 dimensions (see original paper presenting the Fetch environments for details (Plappert et al., 2018)) + 3 per additional out-of-reach cube, while the action space has 4 (3D actions + gripper).

Evaluation Methodology

Every epoch (50×19 actors = 950 episodes), the 19 actors are evaluated offline on 5 rollouts each. For each evaluation rollout, the experimenter asks the agent to perform a goal selected at random in the set of achievable goals (subset of goals the agent trained on). This external evaluation of the agent’s true competence is distinct from the self-evaluation performed by the agent during learning. Note that the agent policy is frozen during evaluation, and that the agent cannot use the evaluation trajectories for later training as opposed

to trajectories obtained by self-evaluation. Since all actors share the same policy, each point \hat{p} of a learning curve represents the maximum likelihood estimate of a Bernoulli probability p (i.e. success rate) using a sample size of $n = 19 \times 5 = 95$ rollouts. With this sample size, the confidence interval can be estimated using a normal distribution:

$$\text{error} = |p - \hat{p}| \leq 1.96 \sqrt{\frac{\hat{p}(1 - \hat{p})}{n}},$$

with probability 95%, which is itself upper-bounded by $1.96 \sqrt{\frac{0.25}{95}} = 0.1$.

Each experiment is replicated with 10 different random seeds. Because the sample size is relatively small (< 30), we cannot make any assumption about the distribution of the performance measures. Comparisons are therefore conducted using a one-tail Mann-Whitney U-test, a non-parametric test that does not make such assumptions (Mann & Whitney, 1947). We use a type-I error (confidence level) $\alpha = 0.01$. The one-tail version is used because we expect our algorithm to perform better.

Hyperparameters

The CURIOUS algorithm is built on top of the OpenAI Baselines implementation of HER-DDPG.² This consists in a parallel implementation with 19 actors. The actors share the same parameters and their updates are averaged to compute the next set of parameters. We use the same hyperparameters as Plappert et al. (2018). Note that this paper does not focus on the underlying learning algorithm, which could be replaced by any off-policy reinforcement learning algorithm.

CURIOUS uses three extra hyperparameters. 1) The probability $p_{\text{eval_c}}$ that the agent performs self-evaluation for the next rollout. It does so to update its subjective competence measure for a module, $p_{\text{eval_c}} = 0.1$. 2) The length of the windows considered to compute the subjective measures of competence and learning progress is set to $l = 300$ episodes. 3) The exploration parameter ϵ which controls the mixture between random module selection and active module selection based on absolute learning progress is set to $\epsilon = 0.4$.

Additional Results

Developmental Learning: Regularities and Diversity

Traditionally, the study of learning phases is based on behavior. This is true in Psychology, which can only access

² The OpenAI Baselines implementation of HER-DDPG can be found at <https://github.com/openai/baselines>, our implementation of CURIOUS can be found at <https://github.com/flowersteam/curious>.

the human behaviors but cannot access internal representations. Multi-goal RL papers as well, when they do provide behavioral studies, usually present the test performance of the agent on various goals as a function of time (e.g. (Mankowitz et al., 2018)). In the present paper however, we have a direct access to the internal representations of our agents, to their beliefs about their current competence and learning progress for each of the modules. It is from these internal representations that we study developmental progressions. This enables us to distinguish active curriculum learning from passive experience of the environmental structure. Indeed, the structure of the environment can be such that some modules are simpler than others, leading random agents to learn them faster than others. On the other hand, agents performing active module selection such as CURIOUS, actively guide their learning trajectories as a function of their internal representations of competence and progress.

In Fig. 9, we present 5 sets of subjective competence curves and their associated learning progress. We can see that M_1 (*Reach* module) is always learned first and M_2 (*Push* module) is always learned second. However, M_3 (*Pick and Place*) and M_4 (*Stack*) can be learned in various order or even simultaneously depending on the individual learning trajectories. Once the agent has experienced a few successes on a module, LP increases and the agent focuses on it, which generates even more successes. What happened by chance in the early stages of learning leads this agent to focus first on either M_3 or M_4 . Although some modules might be easier to learn first, or necessary to learn others, individual experience can influence learning trajectories just as for humans (Oudeyer & Smith, 2016).

Scaling Properties

It is important to discuss the ability of CURIOUS to scale to larger sets of modules. Quick tests showed that CURIOUS could scale to at least 10 achievable modules (Reach, Push, Pick and Place and Stack with different cubes). However, we can expect CURIOUS to fail when the number of modules gets even larger. This could be mitigated by enabling CURIOUS to deal with multiple actors and critics, each pair dealing with a subset of the modules.

Meaning of the Term *Goal*

It is important to note that the term *goal* used in the context of Intrinsically Motivated Goal Exploration Processes (IMGP) is much more general than the one used in this paper. Indeed, in IMGP, *self-generated goal* denotes any self-defined parameterized problem, which solution should be found through ones own actions (e.g. it can as diverse as “Grasp *obj1* and place it at *pos3*”, “Move to (x,y,z) ”, “Find a *blue key*”, “Collect an even number of *obj3*”, or “generate a

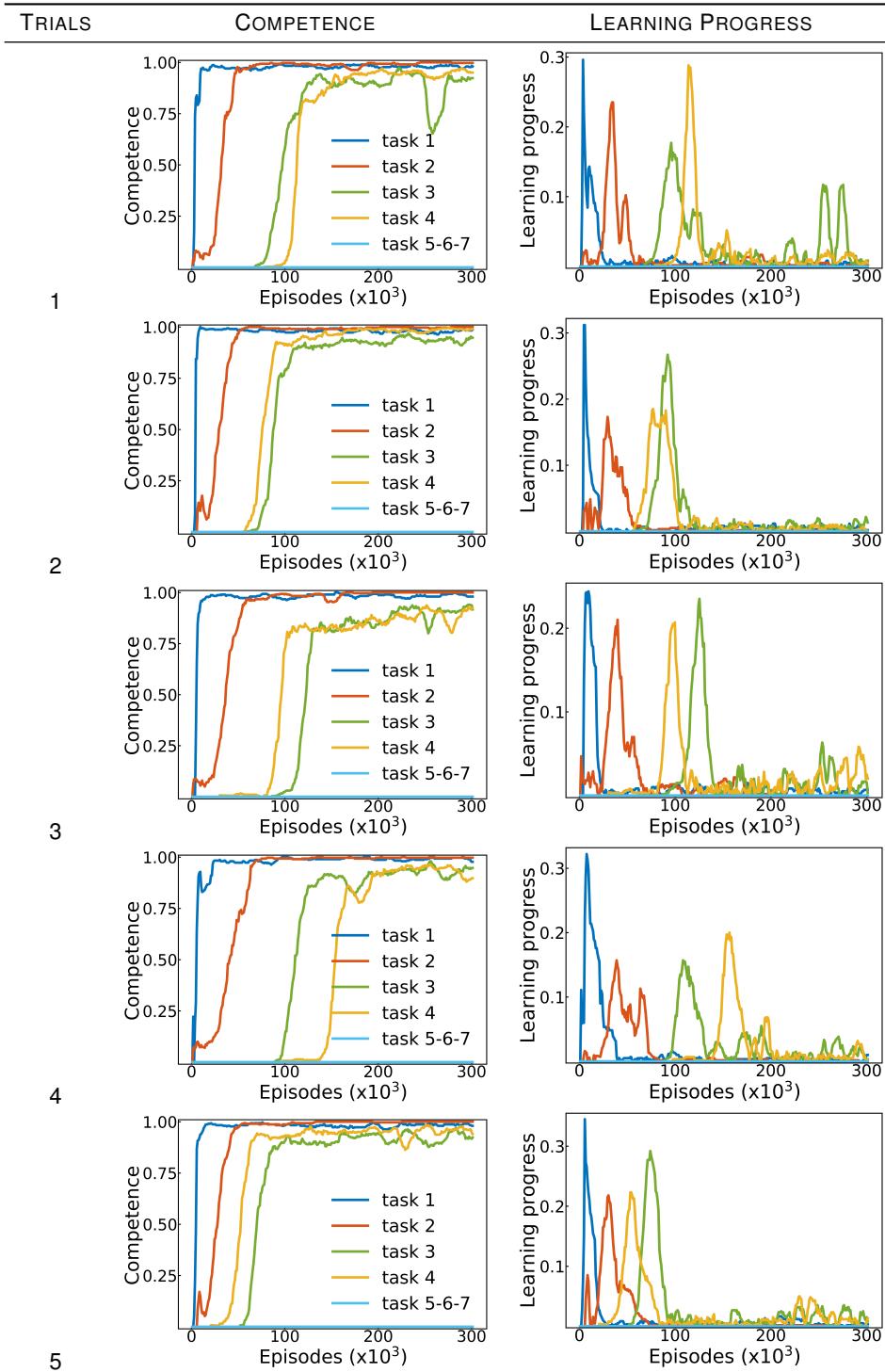


Figure 9. Learning Phases. Competence (left) and absolute learning progress (right) for 5 trials of the CURIOUS algorithm (Reach, Push, Pick and Place, Stack + 3 distracting modules).

trajectory that contains 3 loops”).

Computational Resources

One trial of one algorithm takes around 20 hours to run on 19 cpus. This paper contains around 130 trials, which sums to a total of $130 \times 20 \times 19 \approx 5.6$ cpu years.