CURIOUS: Intrinsically Motivated Modular Multi-Goal Reinforcement Learning (Appendix)

Cédric Colas [1] **Pierre Fournier** [2] **Olivier Sigaud** [2] **Mohamed Chetouani** [2] **Pierre-Yves Oudeyer** [1]

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:

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

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).

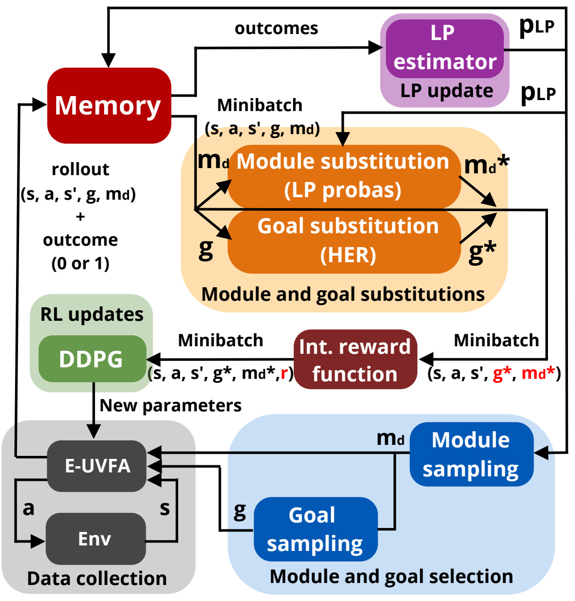


Figure 1. **Schematic view of CURIOUS.**

*Equal contribution [1]Flowers Team, INRIA Sud-Ouest, Bordeaux, FRANCE [2]ISIR, Sorbonne Université, Paris, FRANCE. Correspondence to: Cédric Colas <cedric.colas@inria.fr>.

Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

Additional Methods

Overview. Algorithm 1 and Fig. 1 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. 1). 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 $z =$ height(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 sampling 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 hold 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:

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

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

Algorithm 1 CURIOUS

```
1: Input: env, $\mathcal{M}$, $\mathcal{G}_{1:N}$, $noise$, internal_reward( ), $p_{eval\_c}$      {$\mathcal{M}$: set of $N$ modules, $\mathcal{G}_i$: goal space of module $M_i$ }
2: Initialize: $policy$, $memory$, $p_{LP}$      {random, empty and uniform respectively}
3: repeat
4:     $eval\_c \leftarrow random() < p_{eval\_c}$      {The agent evaluates its competence if True}
5:     $goal, m_d(M_i) \leftarrow$ ModuleGoalSelector()      {$M_i \sim p_{LP}$ (or uniform if $eval\_c$), $goal \sim \mathcal{U}(\mathcal{G}_{M_i})$}
6:     $s_0, outcome_0 \leftarrow$ env.reset()
7:     for $t = 0 : N_t$ do
8:         $policy\_input \leftarrow$ 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:        $s_{t+1}, outcome_{t+1} \leftarrow$ 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($outcome == goal, m_d$)      {If self evaluation, update LP with new result}
18:        $p_{LP} \leftarrow$ LP_Update( $memory$ )
19:    end if
20:    $modified\_batch \leftarrow$ ModuleGoalReplayPolicy($p_{LP}$, $memory$)  {Use Prioritized Interest Replay, $p_{LP}$ and HER }
21:    $modified\_batch \leftarrow$ RewardComputer($modified\_batch$, internal_reward())
22:    $policy \leftarrow$ RL_Updates($modified\_batch$)      {Using DDPG }
23: until learning over
```

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.[1] 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_{eval_c} that the agent performs self-evaluation for the next rollout. It does so to update its subjective competence measure for a module, $p_{eval\_c} = 0.1$. 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 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. 2, 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 (IMGEP) is much more general than the one used in this paper. Indeed, in IMGEP, *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 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.

[1] 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`.

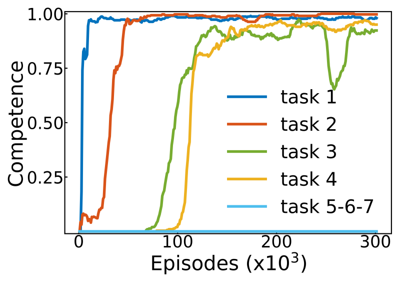
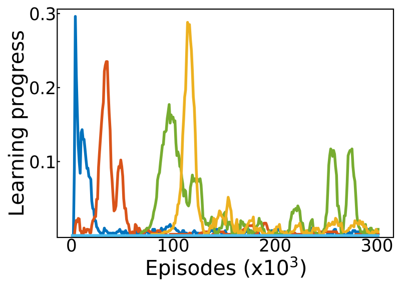
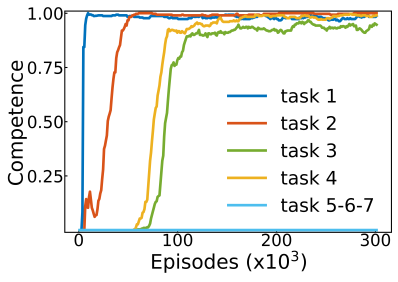
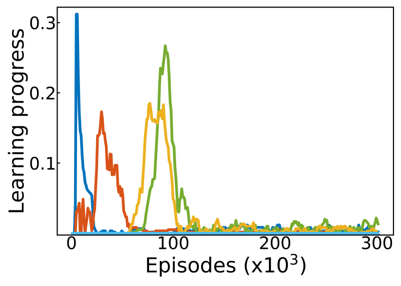
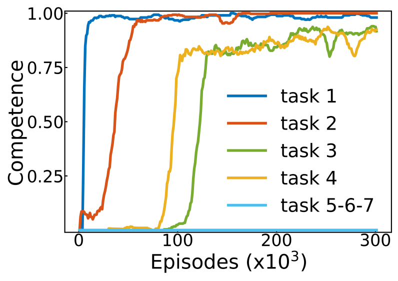
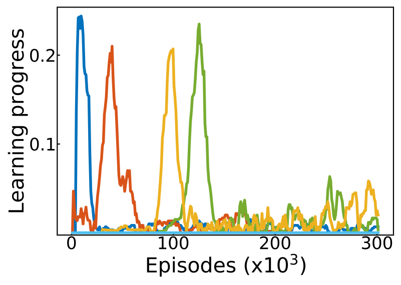
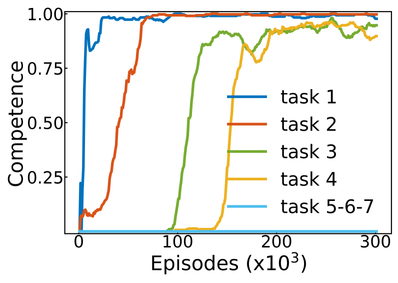
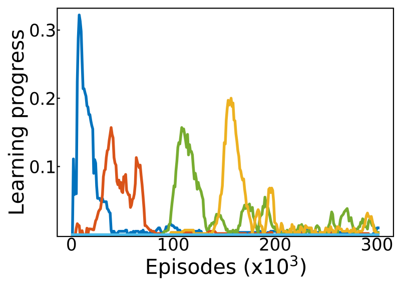
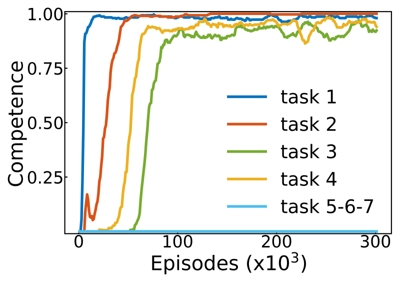
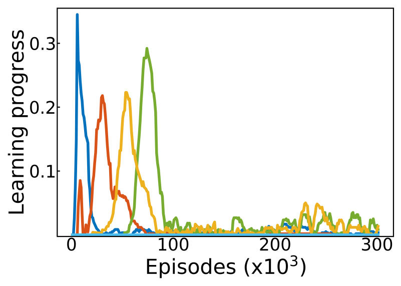
TRIALS	COMPETENCE	LEARNING PROGRESS
1		
2		
3		
4		
5		

Figure 2. **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).

Acknowledgments

Cédric Colas is partly funded by the French Ministère des Armées - Direction Générale de lArmement. 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^o 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.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

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.

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.

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.

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