GEP-PG: Decoupling Exploration and Exploitation in Deep Reinforcement Learning Algorithms

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

Abstract

In continuous action domains, standard deep reinforcement learning algorithms like DDPG suffer from inefficient exploration when facing sparse or deceptive reward problems. Conversely, evolutionary and developmental methods focusing on exploration like Novelty Search, Quality-Diversity or Goal Exploration Processes explore more robustly but are less efficient at fine-tuning policies using gradient-descent. In this paper, we present the GEP-PG approach, taking the best of both worlds by sequentially combining a Goal Exploration Process and two variants of DDPG. We study the learning performance of these components and their combination on a low dimensional deceptive reward problem and on the larger Half-Cheetah benchmark. We show that DDPG fails on the former and that GEP-PG improves over the best DDPG variant in both environments. Supplementary videos and discussion can be found at `frama.link/gep_pg`, the code at `github.com/flowersteam/geppg`.

1. Introduction

Deep reinforcement learning (RL) algorithms combine function approximation based on deep neural networks with RL. Outstanding results have been obtained recently by such systems in problems with either large discrete state and action sets (Mnih et al., 2015; Silver et al., 2016) or continuous state and action domains (Lillicrap et al., 2015). These results have attracted considerable attention on these techniques in the last two years, with the emergence of several competitive algorithms like PPO (Schulman et al., 2017), ACKTR (Wu et al., 2017), ACER (Wang et al., 2016) and Q-PROP (Gu et al., 2016), as well as easily accessible benchmarks (Brockman et al., 2016; Machado et al., 2017b) to perform thorough comparative evaluations (Duan et al., 2016; Islam et al., 2017; Henderson et al., 2017).

[1]INRIA, Flowers Team, Bordeaux, France [2]Sorbonne University, ISIR, Paris, France. Correspondence to: Cédric Colas <cedric.colas@inria.fr>.

Deep RL algorithms generally consist in applying Stochastic Gradient Descent (SGD) to the weights of neural networks representing the policy and an eventual critic, so as to minimize some reward-related cost function. The good performance of these methods mainly comes from the high sample efficiency of SGD, the gradient with respect to the network parameters being analytically available. To deal with the exploration-exploitation trade-off (Sutton & Barto, 1998), a deep RL algorithm uses its current policy to select the best action (exploitation), and adds random noise to explore (action perturbation). This could be a simple Gaussian noise or a more advanced Ornstein-Uhlenbeck (OU) correlated noise process in the case of Deep Deterministic Policy Gradient DDPG (Lillicrap et al., 2015) or a random selection of suboptimal actions in the case of DQN (Mnih et al., 2015). This approach may not be efficient enough in the case of RL problems with multi-dimensional continuous actions where exploration can be particularly challenging (e.g. in robotics).

Recently, a family of evolutionary computation techniques has emerged as a convincing competitor of deep RL in the continuous action domain (Salimans et al., 2017; Conti et al., 2017; Petroski Such et al., 2017). These techniques perform policy search directly in the policy parameter space, classifying them as "parameter perturbation" methods. These methods benefit from more focus on exploration, especially for problems with rare or deceptive rewards and continuous action spaces, as illustrated by many papers on Novelty Search (Lehman & Stanley, 2011; Conti et al., 2017), Quality-Diversity (Pugh et al., 2015; Cully & Demiris, 2017) and curiosity-driven learning (Baranes & Oudeyer, 2013; Forestier et al., 2017). Besides, policy parameter perturbation is generally considered superior to action perturbation in policy search (Stulp & Sigaud, 2013). But, as they do not rely on analytical gradient computations, these methods are generally less sample efficient than SGD-based methods (Salimans et al., 2017).

To summarize, deep RL methods are generally more efficient than evolutionary methods once they are fine-tuning a solution that is already close to the optimum, but their exploration might prove inefficient in problems with flat or deceptive gradients.

Contributions. In this paper, we define a new framework which takes the best of both worlds by sequentially combining elements of both families, focusing on its use and evaluation for continuous action RL problems. We call it GEP-PG for "Goal Exploration Process - Policy Gradient". In the first stage of GEP-PG, a method coming from the curiosity-driven learning literature called Goal Exploration Processes (GEPs) (Forestier et al., 2017) is used to efficiently explore the continuous state-action space of a given problem. The resulting samples are then stored into the replay buffer of a deep RL algorithm, which processes them to perform sample efficient policy improvement. In this paper, the chosen deep RL algorithm is DDPG, which can itself benefit from various exploration strategies (Lillicrap et al., 2015; Plappert et al., 2017).

For evaluation purposes, we use two benchmarks showing distinct properties: 1) the Continuous Mountain Car (CMC) environment, a small size ($2D$ state, $1D$ action) control benchmark with interesting deceptive reward properties and 2) Half-Cheetah (HC), a larger ($17D$ state, $6D$ action) benchmark where DDPG was recently the state-of-the-art (now behind SAC (Haarnoja et al., 2018)).

On these benchmarks, we study the learning performance of GEP, variants of DDPG, and of the sequential combination of both components. We study the learning performances from several point of views: final performance, sample efficiency to reach high-performance, and variability of the learning dynamics. These investigations reveal various aspects of the exploration and exploitation efficiency of these components. Overall, we finally show that GEP-PG provides results beyond DDPG in terms of performance, variability and sample efficiency on HC.

The paper is organized as follows. In Section 2, we identify three categories of methods to perform exploration in policy search and mention research works sharing some similarities with ours. In Section 3, we quickly describe DDPG, then GEPs, the way we combine them into GEP-PG, and the experimental setup and parameters. Performance, variance and sample efficiency results of the corresponding exploration strategies on CMC and HC are given in Section 4. Finally, we discuss these results, conclude and describe potential avenues for future work in Section 5.

2. Related work

As deep RL algorithms start addressing larger and more difficult environments, the importance of exploration, particularly in the case of sparse reward problems and continuous actions, is a more and more recognized concern.

Basic, *undirected exploration* methods rely mostly on noise. This is the case in methods like TRPO (Schulman et al., 2015), PPO (Schulman et al., 2017) and SAC (Haarnoja et al., 2018) which use a stochastic policy, but also in methods using a deterministic policy like DDPG, which add noise to the chosen action. Another approach consists in adding noise directly into the neural networks used as representation for learning (Fortunato et al., 2017; Plappert et al., 2017). The latter approach is generally considered superior (Stulp & Sigaud, 2013), even if this can depend on the respective sizes of the policy parameter space and the state action space.

A more directed exploration strategy strives to cover the state-action space as uniformly as possible. This idea has been studied both in the deep RL and in the evolutionary/developmental literatures. In deep RL, this is achieved using count-based exploration (Bellemare et al., 2016), as well as various forms of intrinsic rewards and exploration bonuses (Houthooft et al., 2016a;b; Pathak et al., 2017) and methods for option discovery (Machado et al., 2017a). Count-based exploration can also be applied in a more compact *latent space* obtained from representation learning techniques (Tang et al., 2016). While some of these strategies have been tested for continuous actions RL problems (Houthooft et al., 2016b; Tang et al., 2016), they remain complex to implement and computationally expensive. Besides, exploration bonuses focus on information gain associated to states (rather than state-action), thus they do not fully consider the exploration challenge of multi-dimensional continuous action spaces.

Evolutionary and developmental methods focused on exploration, like Novelty Search (Lehman & Stanley, 2011; Conti et al., 2017), Quality-Diversity (Pugh et al., 2015; Cully & Demiris, 2017) and Curiosity-Driven Learning (Baranes & Oudeyer, 2013; Forestier & Oudeyer, 2016) generalize the idea of covering well the state-action space. Indeed, instead of trying to cover the state-action space of a predefined RL problem, they try to cover a user-defined space of behavioral features, called "outcome space" or "behavior space", potentially characterizing the full state-action trajectories resulting from a policy roll-out. Within approaches of curiosity-driven exploration, Goal Exploration Processes are methods where this outcome space is used to stochastically sample goals, enabling to efficiently discover repertoires of policies that produce diverse outcomes. Learning such diverse repertoires of policies is often useful for finding approximate solutions to difficult problems with flat, deceptive or rare rewards. However, these methods are episode-based rather than step-based (Deisenroth et al., 2013), hence they are generally less sample efficient in fine-tuning the parameters of a policy (see Section 3.2 for more details).

Our approach sequentially combines evolutionary/ developmental methods for exploration and more traditional deep RL algorithms for fine-tuning policy parameters. This is similar to (Nair et al., 2017), in which demonstration trajec-

tories are used to inform DDPG through the replay buffer. Using Goal Exploration Processes provides extra information on what might be useful to explore via the design of the outcome space, unlike imitation learning which provides more directive information on how to behave. Besides, like the *Guided Policy Search* (GPS) algorithm (Levine & Koltun, 2013), our approach can first use simple policies to generate samples and then train a richer policy. Finally, like us, Zimmer & Doncieux (2017) first explore an environment with an evolutionary method, then call upon an RL algorithm to speed up policy convergence. However, they do so by extracting a discrete state-action representation and using the Q-LEARNING algorithm (Watkins, 1989), avoiding the burden of solving continuous action problems.

3. Methods

In this section we quickly define DDPG and Goal Exploration Processes (GEPs) as background methods of our study, then we present GEP-PG and we describe the experimental setup. Our code-base is made available online[1].

3.1. DDPG

The DDPG algorithm (Lillicrap et al., 2015) is a deep RL algorithm based on the Deterministic Policy Gradient (Silver et al., 2014). It borrows the use of a replay buffer and a target network from DQN (Mnih et al., 2015). In this paper, we use two versions of DDPG: 1) the standard implementation of DDPG with action perturbations (OU noise) (Lillicrap et al., 2015); 2) a variant in which the neural network used as actor is augmented with some noise (parameter perturbations), resulting in better exploration and more robust performance (Plappert et al., 2017). All the parameters of DDPG used in this paper are taken from baselines used in systematic studies (Islam et al., 2017; Henderson et al., 2017), and are described in Section 3.6.

3.2. Goal Exploration Processes

Goal Exploration Processes (GEPs) are a subpart of Intrinsically Motivated Goal Exploration Processes (IMGEPs), a more general framework addressing efficient exploration and curriculum learning in developmental scenarios (Forestier et al., 2017). Here we focus on the GEP part, describing it under the perspective of a policy search process endowed with efficient exploration properties.

In GEPs, two spaces are used: the policy parameter space Θ and the outcome space O. The latter is designed by the experimenter, and is often a set of behavioral features characterizing the state-action trajectory produced by running a policy in the environment. Thus, the outcome o (which is a vector of features) of a policy parameterized by θ can be measured by the system after each roll-out of this policy.

The general process can be described as follows. In a first *bootstrap* stage, N sets of policy parameters θ are drawn randomly from Θ. For each policy the resulting outcome o in O is observed. This stage implements a pure random policy search process. Both at this stage and at the next one, each time some new point o is observed, the corresponding $\langle \theta, o \rangle$ pair is stored. In the second stage, vectors o are randomly drawn from O. These points are considered as goals and are not related to the actual goal of a particular benchmark. GEP therefore implements an exploration that is *directed* towards the generation of behavioral diversity and is completely unaware of the external reward signal of a benchmark. Once these goals are drawn, the algorithm looks for policy parameters θ which could reach them using a simple search method. It looks for the closest stored point in the outcome space (a nearest-neighbor strategy) and randomly perturbs the corresponding θ. If the perturbation is adequate by chance, the generated outcome should be closer to the goal than the previous one. Importantly, even if the perturbations is not closer to the goal, it enables to discover a novel outcome and progress towards other goals as a side effect. Instead of random perturbations, more advanced interpolation-based techniques can be used (Forestier & Oudeyer, 2016), but here we stick to the simplest method and apply Gaussian noise $\mathcal{N}(0\ , \sigma^2)$.

By storing more and more $\langle \theta, o \rangle$ pairs, GEPs improve reaching capabilities to a variety of goals. As it stores a population of policies rather than trying to learn a single large goal-parameterized policy, and as generalization can be made on-the-fly through various forms of regression (Forestier & Oudeyer, 2016), this can be viewed as a form of *lazy learning* (Aha, 1997). Described this way, GEPs share a lot of similarities with Novelty Search (Lehman & Stanley, 2011; Conti et al., 2017) and Quality-Diversity (Pugh et al., 2015; Cully & Demiris, 2017) methods.

While GEPs do not use a potential reward information corresponding to an externally imposed RL problem when they explore, e.g. in a benchmark, they can observe this RL-specific reward during policy roll-outs and store it for later reuse. In the GEP-PG framework described below, this is reused both by the learning algorithm to initialize a replay buffer for DDPG and by the experimenter to measure the performance of the best policies found by GEP from the perspective of a target RL problem (and even if GEP does not try to optimize the associated reward).

3.3. The GEP-PG methodology

The GEP-PG methodology comes with two steps. In a first *exploration* stage, a large diversity of samples are generated and stored using GEP, using either a simple linear policy

[1] https://github.com/flowersteam/geppg

or the same policies as DDPG. In a second *exploitation* stage, DDPG's replay buffer is loaded with GEP-generated samples and it is used to learn a policy. As described above, in this paper we use two versions of DDPG coined *action-* and *parameter-* perturbation.

3.4. Experimental setups

Below we describe the two benchmark environments used, namely CMC and HC. For both setups, we use the implementations from OpenAI Gym (Brockman et al., 2016).

3.4.1. Continuous Mountain Car

The Mountain Car environment is a standard benchmark for the continuous state, discrete action RL setting. An underpowered car whose actions are in $\{-1, 0, 1\}$ must reach its goal at the top of a hill by gaining momentum from another hill. The state space contains the horizontal position and velocity and the action is a scalar acceleration. In the standard setting, the goal is to reach the flag as fast as possible. A bang-bang policy, using only $\{-1, 1\}$ as actions, is the optimal solution. The environment is reset after 10^3 steps or when the flag is reached.

In the *Continuous Mountain Car* (CMC) environment, the action is defined over $[-1, 1]$ and the car should reach the goal while spending as little energy as possible. To ensure this, the reward function contains a reward of 100 for reaching the goal and, at each step, an energy penalty for using acceleration a which is $-0.1 \times a^2$. This reward function raises a specific exploration issue: it is necessary to perform large accelerations to reach the goal but larger accelerations also bring larger penalties. As a result, the agent should perform the smallest sufficient accelerations, which results in a "Gradient Cliff" landscape as described in (Lehman et al., 2017). If the goal is not reached, the least negative policy would consist in performing null accelerations. Thus, as long as the agent did not reach the goal, the environment provides a *deceptive* reward signal which may drive policy updates in the wrong direction. As a consequence, when using an SGD-based approach, successful learning is conditioned on reaching the goal fast enough, so that the attractiveness of getting the goal reward overcomes the tendency to stop accelerating. Due to all these features, CMC is a favorable environment for our study.

3.4.2. Half-Cheetah

Our second benchmark is HC, in which the agent moves a 2D bipedal body made of 8 articulated rigid links. The agent can observe the positions and angles of its joints, its linear positions and velocities ($17D$) and can act on the torques of its legs joints ($6D$). The reward r is the sum of the instantaneous linear velocities on the x axis $v_x(t)$ at each step, minus the norm of the action vector a: $r = \sum_t (v_x(t) - 0.1 \times |a(t)|^2)$. The environment is reset after 10^3 steps, without any other termination criterion. As in CMC, HC shows a "Gradient Cliff" landscape property as we experienced that a cheetah running too fast might fall, resulting in a sudden drop of performance close to the optimum.

3.5. Evaluation methodology

Difficulties in reproducing reported results using standard algorithms on standard benchmarks is a serious issue in the field of deep RL. For instance, with the same hyper-parameters, two different codes can provide different results (Islam et al., 2017; Henderson et al., 2017). In an attempt to increase reproducibility, the authors of (Henderson et al., 2017) have put forward some methodological guidelines and a standardized evaluation methodology which we tried to follow as closely as possible. To avoid the code issue, we used the implementation of Henderson's fork of OpenAI Baselines (Dhariwal et al., 2017) for all the variants of DDPG. In our work as in theirs, DDPG is run in the environment for 100 roll-out steps before being trained with 50 random batches of samples from the replay buffer. This cycle is repeated 20 times (2.10^3 steps in the environment) before DDPG is evaluated off-line on 10^4 steps (10 episodes).

However, this standardized methodology still raises two problems. First, though the authors show that randomly averaging the results over two splits of 5 different random seeds can lead to statistically different results, they still perform evaluations with 5 seeds. In all the experiments below, we use 20 different seeds. In order to make sure averaging over 20 seeds was sufficient to counter the variance issue, we run 40 trials of the baseline DDPG with OU noise ($\mu = 0, \sigma = 0.3$) for both environments and computed our comparison tests for 1000 different sets of 20 randomly selected runs. To compare the performance of the various algorithms, we use the tests advised by (Henderson et al., 2017): a paired t-test and a bootstrap estimation of the 95% confidence interval of the means difference using 10^4 bootstraps. For all tests listed above, less than 5.01% of the sets were found statistically distinct, with most of the test showing a 0% error. Two groups of 20 seeds have therefore between 0% and 5% chances of being found separable depending on the considered test and metric (see Appendix A). In all figures, we show the mean and standard error of the mean across 20 runs using different random seeds while the x-axis is the number of time steps.

Second, the authors of (Henderson et al., 2017) report what we call a *final metric* of performance: the average over the last 100 evaluation episodes, 10 episodes for each of the last 10 controllers. This can be a problem for two reasons: 1) when progress is unstable, the final steps may not correspond to the overall best performing policies; 2) an average

over several different policies obtained along learning does not mean much. Using this metric, only the last policies are considered, forgetting about the potential good ones discovered in the past, good policies that we might want to save for reuse.

As an alternative, we find it more appropriate and more informative to report the performance over 100 evaluation episodes of the best policy found over the whole training process (called *absolute metric*). The process duration is set to $2M$ steps on HC to keep close to the standard methodology, and 5.10^5 steps on CMC.

Concerning the GEP part of GEP-PG, we keep the same structure and show the average performance over 10 episodes (10^4 steps) every 2.10^3 steps in the environment. Note that the termination condition of CMC implies that an episode does not always correspond to 10^3 steps, e.g. when the goal is reached before the end. We therefore show the performance evaluated at the nearest step being a multiple of the figure resolution (2.10^3 steps).

In the results below, we report the performance as evaluated by both the absolute and the final metrics, to facilitate comparison with the literature while providing more informative evaluations.

3.6. Experimental parameters

For DDPG, we use the default set of parameters provided in Henderson's version mentioned above. The replay buffer is a sliding window of size 10^6. We also tried a replay buffer size of 2.10^6 samples, but this led to lower performance in all cases. Three types of noise are used in our different comparisons: 1) an OU process with $\mu = 0$, $\sigma = 0.3$ and $\theta = 0.15$, 2) an OU version where σ linearly decreases from 0.6 at the first step to 0 at the final step, and 3) the noise on actor network parameters with variance $\sigma = 0.2$, which shows state-of-the-art performance on HC (Plappert et al., 2017; Henderson et al., 2017). The other meta-parameters of DDPG are as follows: the batch size is 64, the discount factor is 0.99, the actor and critic networks have two hidden layers of size $(64, 64)$ and RELU activation functions, the output layer activation function are $tanh$ and linear respectively. The learning rates are 10^{-4} and 10^{-3} respectively and the SGD algorithm is ADAM.

As stated above, the simple GEP policies are linear policies with $tanh$ activation functions. In CMC, it maps the position and velocity to the action and the corresponding search space is $2D$ instead of $4288D$ if we had used the DDPG policy. In HC, it maps the position and velocity of the joints to the torques (12 observations), it is $72D$ instead of $5248D$. In a bootstrap stage, the GEP policy parameters are sampled randomly in $[-1, 1]^P$ where P is the number of parameters. Bootstrap consists of 5 episodes for CMC, 50 for HC. Then 45 (respectively 450) random goals are drawn in a custom goal space which is $3D$ for CMC (range of position, maximum position and energy spent) and $2D$ for HC (mean velocity and minimum head position). Given a goal, the algorithm finds in its memory the closest state previously experienced using a k-nearest neighbor algorithm using an Euclidean distance and $(k = 1)$, and samples parameters around the associated set of policy parameters with centered Gaussian noise using $\sigma = 0.01$.

4. Results

In this section, we proceed as follows. First, we investigate the exploration efficiency of action perturbation and policy parameter perturbation methods in DDPG using the CMC and HC benchmarks. Second, we examine whether GEPs are better than the above undirected exploration methods at reaching the goal for the first time on CMC. Third, we use the GEP-PG algorithm to investigate whether the better exploration capability of GEPs translates into a higher learning performance on CMC and HC. We show that, though GEP alone performs better than GEP-PG on CMC, GEP-PG outperforms both GEP alone and the DDPG variants on HC, in terms of final performance and variance.

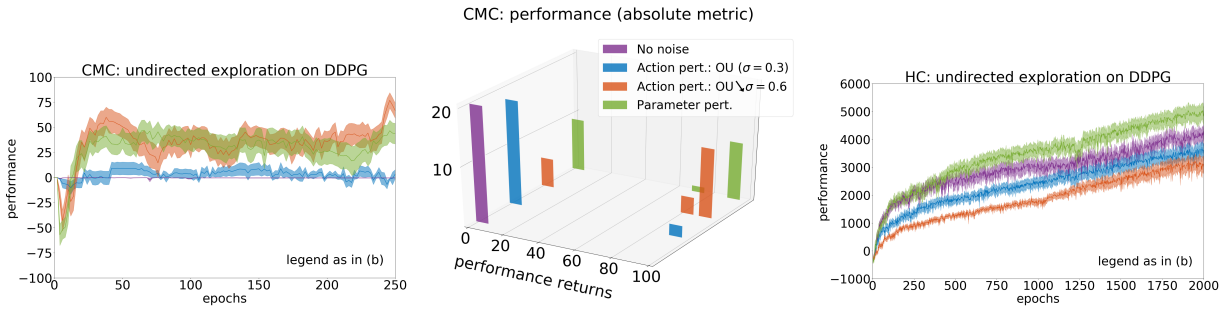
4.1. Undirected exploration in DDPG

First, we compare the effect of action perturbation versus policy parameter perturbation on DDPG performance on CMC and HC. More precisely, we compare in Figure 1: 1) a DDPG version without any exploration noise as a baseline, 2) the approach of (Lillicrap et al., 2015) using OU noise with $(\mu = 0, \sigma = 0.3)$, 3) the parameter perturbation method implemented in (Plappert et al., 2017) and 4) OU noise with σ linearly decreasing from 0.6 to 0.

Consistently with (Plappert et al., 2017), we found that parameter perturbation outperforms action perturbation on HC, but also on CMC on both metrics of performance (all tests significant at the 5% level, see Appendix D).

On CMC, the DDPG version without noise performs poorly as it does not have the exploration drive required to reach the reward a first time. The decreasing noise strategy significantly improves over the static one, as a stronger initial noise helps finding the goal more often. On HC, it seems that having no noise, a standard OU(0.3) or a decreasing noise do not lead to statistically different performances at 2M steps, although difference could be noted earlier on. We conclude that increasing noise in the first stage is beneficial in CMC but might be detrimental in HC.

These contrasting results might be due to several phenomena. First, there is probably less need to explore on HC than on CMC, as reward information is available anytime. Second, as explained in Section 3.4.1, reaching the goal early is



(a) (b) (c)

Figure 1. (a): Effect of action perturbation versus policy parameter perturbation on the performance of DDPG on CMC. (b): Histogram of absolute performances on CMC (c): Same as (a) for HC.

crucial on CMC, which is not the case on HC. Using a stronger exploration noise in the first stage might therefore bring a significant advantage on CMC. Third, it might be the case that too much noise is more detrimental to performance on HC than on CMC because on HC more noise may result in more falls.

However, the information depicted in Figure 1(a) must be taken with a grain of salt. Indeed, the curves depict averages over 20 runs, but the individual performances are not normally distributed (see Appendix C). Because learning is unstable, an algorithm having found a good policy during learning might present a bad one in the final steps. To present complementary evaluations, we chose to report the histograms of absolute metrics across seeds. This absolute metric for a given seed is computed as the average performance over 100 test episodes of the best policy found across training, see Figure 1(b).

As a conclusion, parameter perturbations perform better than action perturbations, whereas increasing exploration in a first stage might be either advantageous or detrimental to the performance depending on the environment properties.

4.2. Exploration efficiency of GEP and DDPG on CMC

As explained above, the CMC benchmark raises a specific exploration issue where the time at which the goal is first reached is crucial. The same is not true of HC. In this section, we study how fast two variants of GEP and two variants of DDPG can find the goal.

Figure 2 represents the histograms of the number of steps before the goal is reached for the first time on CMC. We compare four algorithms: GEP applied to the linear policy we use in GEP-PG, GEP applied to the same policy as DDPG $(64, 64)$ that we call *complex policy* thereafter, DDPG with noise on the network parameters and DDPG with OU noise $(\mu = 0, \sigma = 0.3)$ on actions. We also tried GEP applied to a policy with hidden layers $(400, 300)$, but the result is not statistically different from the one obtained with the $(64, 64)$ policy.

For each condition, we perform 1000 trials and we record the mean and the histogram of the number of steps necessary to reach the goal. However, because both variants of DDPG might never reach the goal, we stop experiments with these algorithms if they have not reached the goal after 5.10^4 steps, which is more than the maximum number of steps needed with GEP variants.

The GEP algorithm using a simple linear policy and the complex policy take on average 3875 and 3773 steps respectively to find the top. Using DDPG with parameter perturbation (respectively action perturbation), the goal is reached before 5.10^4 steps only in 42% (respectively 22%) of the trials. These results are enough to show that GEP finds the goal faster than the DDPG variants. This effect can be attributed to the deceptive gradient issue present in CMC, as performing SGD before finding the goal drives policy improvement into a wrong direction.

The GEP variants actually reach the goal as early as the bootstrap phase where the policy is essentially random. Thus, despite the deceptive gradient effect, random exploration is enough to solve CMC. This confirms the idea that CMC could be considered a simple problem when the algorithm is not blinded by the reward feedback.

Finally, the policy complexity of GEP does not seem important. What matters is the ratio between the size of Θ and the subspace of successful policy parameters, rather than the size of Θ itself.

4.3. Combining GEP and DDPG into GEP-PG

In this section, we compare the performance of DDPG with action perturbation and with parameter perturbation to the corresponding GEP-PG versions and to GEP alone. In Figure 3(c), the vertical dotted line signals the transition from

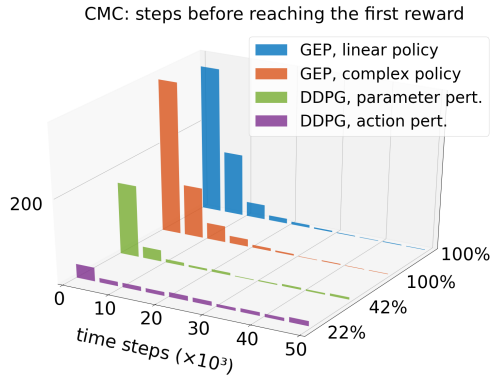


Figure 2. Histogram of number of steps to reach the goal for the first time with: 1)GEP applied to the linear policy; 2) GEP applied to a deep policy; 3) DDPG with noise on the network parameters and 4) DDPG with OU noise ($\mu = 0, \sigma = 0.3$) on actions. The percentages of times the goal was found before 5.10^4 steps is indicated on the y-axis.

the GEP part of GEP-PG to the DDPG part. In Figure 3(a), the dotted line is indicative, because GEPs perform 50 episodes and 1 episode does not always last 10^3 steps in CMC.

In GEP-PG performance curves, performance drops when switching from GEP to DDPG because the policy is not transfered from one algorithm to the other (only samples gathered by GEP are transmitted to the replay buffer of DDPG). In the case where GEP is using a DDPG-like policy, such transfer would be possible, but this is left for future work.

As in Section 4.1, the learning performance depicted in Figure 3(a) does reflect the ratio of trials which found the goal rather than an average performance and these ratios are better depicted in the histogram of Figure 3(b). Nevertheless, Figure 3(a) shows that GEP alone quickly finds an efficient policy and remains stable at a good performance. This stability is easily explained by the fact that, being a lazy learning approach, a GEP cannot forget a good solution to a stationary problem, in contrast with DDPG. We can also see that DDPG with standard OU noise performs surprisingly poorly on this low dimensional benchmark, and that DDPG with noise on the network parameters does not perform much better. Finally, the GEP-PG counterparts of the DDPG variants perform slightly better, but not at the level of GEP.

From these results, GEP alone being better, one may wonder whether GEP-PG and deep RL approaches in general are useful at all. The point that they are will be made when using the higher dimensional HC benchmark.

On HC, GEP alone using the linear policy performs statistically worse than others on both performance metrics, a performance that is not due to the simplicity of its policy since using the $(64, 64)$ neurons policy provides results that are not significantly different (see Appendix F).

More importantly, the GEP-PG versions significantly outperform their DDPG counterparts on both metrics and reach a new state-of-the-art performance on this benchmark where DDPG with noise on the network parameters was the current leader (Henderson et al., 2017). The performance of final policies and best policies found over the whole learning process are given in Table 1. Note that learning curves of DDPG without exploration noise and its GEP-PG counterpart are not represented here, as they were found to match closely the corresponding versions using OU(0.3) noise.

Table 1. Final and absolute performances on HC, mean (std)

Algorithm	final metric	absolute metric
DDPG action pert.	3596 (1102)	3943 (1209)
DDPG param. pert.	4961 (1288)	5445 (1265)
GEP-PG action pert.	4521 (738)	5033 (833)
GEP-PG param pert.	**5740 (573)**	**6118 (634)**
GEP param pert.	2105 (881)	2140 (881)

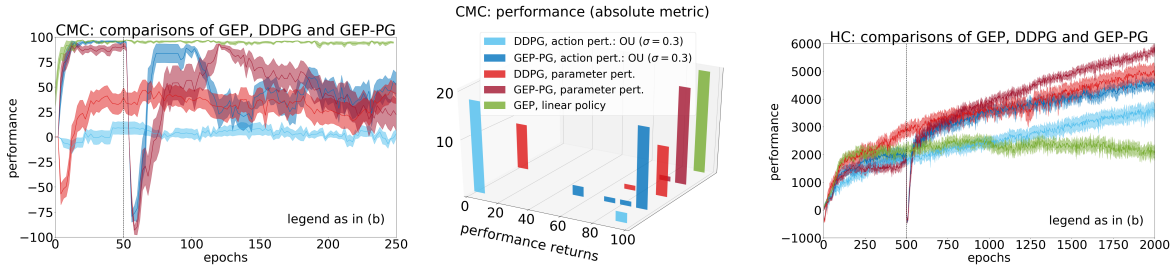
GEP-PG is found to be robust to the number of GEP episodes used to fill the DDPG replay buffer. Performance is stable from about 100 to 600 GEP episodes (see Appendix G). Finally, the GEP-PG versions seem to generate less variability than their DDPG counterparts, which means that an efficient policy can be found more consistently (see standard deviations in Table 1 and Appendix E, Figure 6). Performance predictability matters if performance should not fall below a specific level (e.g. for safety reasons).

From Figure 3, one may consider that, on both CMC and HC, GEP-PG with policy parameter noise outperforms GEP-PG with standard OU noise on both metrics. Further analyses of the impact of the content of a buffer filled with GEP on GEP-PG performance led to the following conclusions: 1) the size of the buffer does not impact GEP-PG performance (from 100 to 2.10^3 episodes); 2) GEP-PG performance correlates with GEP best performance and the average performance of training policies; 3) GEP-PG performance correlates to diversity in terms of observations and outcomes as measured by various metrics (see Appendix G). Therefore, a good buffer should contain both efficient and diverse trajectories.

Note that, as it is used here, GEP-PG only uses 0.3% additional storage and has a higher complexity than DDPG w.r.t the number of episodes: $\mathcal{O}(n^2 \log n)$ vs. $\mathcal{O}(n)$. However, due to complexity constants, it is much faster in practice for the number of episodes we use.

5. Discussion and conclusion

Reinforcement learning problems with sparse or deceptive rewards are challenging for continuous actions algorithms using SGD, such as DDPG. In sparse reward problems, the necessary gradient to perform policy improvement may not be available until a source of reward is found. Even worse,



(a) (b) (c)

Figure 3. (a): Learning performance on CMC of GEP, DDPG with OU noise ($\mu = 0, \sigma = 0.3$) on actions, DDPG with noise on policy parameters and GEP-PG with the same noise configurations. (b): Histogram of absolute performances on CMC (c): Same as (a) for HC.

in deceptive reward problems, the gradient may drive to wrong directions. The baseline version of DDPG, using random action perturbation is highly inefficient in these contexts as shown on the CMC and HC benchmarks. We have reproduced results showing that policy parameter noise is a better strategy, but our experiments show that this form of exploration is still insufficient. So how should we organize exploration, especially in the early stages of learning, to be robust to rare and deceptive rewards?

Decoupling exploration and exploitation. We have proposed GEP-PG as a two-phase approach, where a first exploration phase discovers a repertoire of simple policies maximizing behavioral diversity, ignoring the reward function. In particular, we used the developmental GEP approach and analyzed its exploration efficiency on CMC. While experiments presented in this paper show that GEP alone is already competitive, it is less efficient as it gets closer to the optimal policy in larger benchmarks such as HC. This is why, in GEP-PG, the exploration phase is followed by a more standard deep RL phase for fine-tuning, where DDPG uses a replay buffer filled with samples generated by GEP.

Using two benchmarks, we have shown that training DDPG after filling its replay buffer with a GEP 1) is more sample efficient, 2) leads to better absolute final performance and 3) has less variance than training it from scratch using standard action or parameter perturbation methods. Furthermore, we have shown that using a GEP alone was the method of choice for the simple CMC benchmark, partly due to its deceptive reward signal effect, but GEP-PG takes the lead and provides performance beyond the DDPG on the larger HC benchmark. While this paper focused on the detailed study of these algorithms in two benchmarks, future work will evaluate them more broadly.

Limits and future work. We saw that GEP-PG was robust to the number of episodes in the initial exploration phase. A next stage could be to extend GEP-PG towards an adaptive transition or mixing mechanism, where a multi-armed bandit could be used to dynamically switch between both learning strategies using measures like learning progress (Lopes & Oudeyer, 2012).

It would be interesting to compare other variants of GEP-PG. For the exploration phase, more advanced forms of GEP algorithms could be used, for example using curiosity-driven sampling of goals (Baranes & Oudeyer, 2013) or closely related exploration algorithms in the evolutionary computation literature like Novelty Search (Conti et al., 2017) or Quality-Diversity (Cully & Demiris, 2017). For the exploitation phase, DDPG could be replaced by the more recent ACKTR (Wu et al., 2017) which can in principle be trained off-policy. The GEP-PG approach could also be applied to discrete action RL problems with rare rewards, bootstrapping exploration to initialize the replay buffer of the DQN algorithm or its variants. On the contrary, this approach can not be used with an on-policy deep RL algorithm such as TRPO or PPO because they do not use a pivotal replay buffer.

As the GEP is intrinsically a multi-goal exploration algorithm, another line of investigation would be to study its impact for bootstrapping goal-parameterized deep RL approaches for continuous actions RL problems like Hindsight Experience Replay (Andrychowicz et al., 2017; Held et al., 2017). This would be interesting as goal-parameterized deep RL approaches sample goals stochastically as part of their exploration mechanism (like GEP), but differ from GEP approaches as they learn a single large policy as opposed to a population of simpler policies.

Finally, evolutionary and developmental exploration methods like GEP, Novelty Search or Quality-Diversity take advantage of a behavioral features space, which in most works characterizes properties of the state or state-action trajectories. Many instances of these algorithms have successfully used features that are relatively straightforward to define manually, but an open question is how they could be learned through unsupervised learning (see (Pere et al., 2018) for a potential approach) and how such learned features would impact the dynamics of the GEP-PG approach.

6. Acknowledgments

We thank Pierre Manceron for his strong investment in an initial version of this work and Pierre Fournier for useful discussions. This research is financially supported by the French Ministère des Armées - Direction Générale de l'Armement. This work was supported by the European Commission, within the DREAM project, and has received funding from the European Unions Horizon 2020 research and innovation program under grant agreement N^o 640891.

References

Aha, David W. Editorial. In *Lazy learning*, pp. 7–10. Springer, 1997.

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

Baranes, Adrien and Oudeyer, Pierre-Yves. Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems*, 61(1), 2013.

Bellemare, Marc, Srinivasan, Sriram, Ostrovski, Georg, Schaul, Tom, Saxton, David, and Munos, Remi. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pp. 1471–1479, 2016.

Brockman, Greg, Cheung, Vicki, Pettersson, Ludwig, Schneider, Jonas, Schulman, John, Tang, Jie, and Zaremba, Wojciech. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Conti, Edoardo, Madhavan, Vashisht, Such, Felipe Petroski, Lehman, Joel, Stanley, Kenneth O., and Clune, Jeff. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. *arXiv preprint arXiv:1712.06560*, 2017.

Cully, Antoine and Demiris, Yiannis. Quality and diversity optimization: A unifying modular framework. *IEEE Transactions on Evolutionary Computation*, 2017.

Deisenroth, Marc Peter, Neumann, Gerhard, Peters, Jan, et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142, 2013.

Dhariwal, Prafulla, Hesse, Christopher, Klimov, Oleg, Nichol, Alex, Plappert, Matthias, Radford, Alec, Schulman, John, Sidor, Szymon, and Wu, Yuhuai. Openai baselines. `https://github.com/openai/baselines`, 2017.

Duan, Yan, Chen, Xi, Houthooft, Rein, Schulman, John, and Abbeel, Pieter. Benchmarking deep reinforcement learning for continuous control. *arXiv preprint arXiv:1604.06778*, 2016.

Forestier, Sébastien and Oudeyer, Pierre-Yves. 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, Sébastien, Mollard, Yoan, and Oudeyer, Pierre-Yves. Intrinsically motivated goal exploration processes with automatic curriculum learning. *arXiv preprint arXiv:1708.02190*, 2017.

Fortunato, Meire, Azar, Mohammad Gheshlaghi, Piot, Bilal, Menick, Jacob, Osband, Ian, Graves, Alex, Mnih, Vlad, Munos, Remi, Hassabis, Demis, Pietquin, Olivier, et al. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*, 2017.

Gu, Shixiang, Lillicrap, Timothy, Ghahramani, Zoubin, Turner, Richard E., and Levine, Sergey. Q-prop: Sample-efficient policy gradient with an off-policy critic. *arXiv preprint arXiv:1611.02247*, 2016.

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

Held, David, Geng, Xinyang, Florensa, Carlos, and Abbeel, Pieter. Automatic goal generation for reinforcement learning agents. *arXiv preprint arXiv:1705.06366*, 2017.

Henderson, Peter, Islam, Riashat, Bachman, Philip, Pineau, Joelle, Precup, Doina, and Meger, David. Deep reinforcement learning that matters. *arXiv preprint arXiv:1709.06560*, 2017.

Houthooft, Rein, Chen, Xi, Duan, Yan, Schulman, John, De Turck, Filip, and Abbeel, Pieter. Curiosity-driven exploration in deep reinforcement learning via bayesian neural networks. *arXiv preprint arXiv:1605.09674*, 2016a.

Houthooft, Rein, Chen, Xi, Duan, Yan, Schulman, John, De Turck, Filip, and Abbeel, Pieter. Vime: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems*, pp. 1109–1117, 2016b.

Islam, Riashat, Henderson, Peter, Gomrokchi, Maziar, and Precup, Doina. Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. In *Proceedings of the ICML 2017 workshop on Reproducibility in Machine Learning (RML)*, 2017.

Lehman, Joel and Stanley, Kenneth O. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223, 2011.

Lehman, Joel, Chen, Jay, Clune, Jeff, and Stanley, Kenneth O. ES is more than just a traditional finite-difference approximator. *arXiv preprint arXiv:1712.06568*, 2017.

Levine, Sergey and Koltun, Vladlen. Guided policy search. In *Proceedings of the 30th International Conference on Machine Learning*, pp. 1–9, 2013.

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

Lopes, Manuel and Oudeyer, Pierre-Yves. The strategic student approach for life-long exploration and learning. In *Development and Learning and Epigenetic Robotics (ICDL), 2012 IEEE International Conference on*, pp. 1–8. IEEE, 2012.

Machado, Marlos C, Bellemare, Marc G, and Bowling, Michael. A laplacian framework for option discovery in reinforcement learning, 2017a.

Machado, Marlos C, Bellemare, Marc G, Talvitie, Erik, Veness, Joel, Hausknecht, Matthew, and Bowling, Michael. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *arXiv preprint arXiv:1709.06009*, 2017b.

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

Nair, Ashvin, McGrew, Bob, Andrychowicz, Marcin, Zaremba, Wojciech, and Abbeel, Pieter. Overcoming exploration in reinforcement learning with demonstrations. *arXiv preprint arXiv:1709.10089*, 2017.

Pathak, Deepak, Agrawal, Pulkit, Efros, Alexei A., and Darrell, Trevor. Curiosity-driven exploration by self-supervised prediction. *arXiv preprint arXiv:1705.05363*, 2017.

Pere, Alexandre, Forestier, Sebastien, Sigaud, Olivier, and Oudeyer, Pierre-Yves. Unsupervised learning of goal spaces for intrinsically motivated goal exploration. In *International Conference on Learning Representations (ICLR)*, 2018.

Petroski Such, Felipe, Madhavan, Vashisht, Conti, Edoardo, Lehman, Joel, Stanley, Kenneth O., and Clune, Jeff. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*, 2017.

Plappert, Matthias, Houthooft, Rein, Dhariwal, Prafulla, Sidor, Szymon, Chen, Richard Y., Chen, Xi, Asfour, Tamim, Abbeel, Pieter, and Andrychowicz, Marcin. Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*, 2017.

Pugh, Justin K, Soros, LB, Szerlip, Paul A, and Stanley, Kenneth O. Confronting the challenge of quality diversity. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pp. 967–974. ACM, 2015.

Salimans, Tim, Ho, Jonathan, Chen, Xi, and Sutskever, Ilya. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

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

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

Silver, David, Lever, Guy, Heess, Nicolas, Degris, Thomas, Wierstra, Daan, and Riedmiller, Martin. Deterministic policy gradient algorithms. In *Proceedings of the 30th International Conference in Machine Learning*, 2014.

Silver, David, Huang, Aja, Maddison, Chris J., Guez, Arthur, Sifre, Laurent, Van Den Driessche, George, Schrittwieser, Julian, Antonoglou, Ioannis, Panneershelvam, Veda, Lanctot, Marc, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Stulp, Freek and Sigaud, Olivier. Robot skill learning: From reinforcement learning to evolution strategies. *Paladyn Journal of Behavioral Robotics*, 4(1):49–61, august 2013. doi: 10.2478/pjbr-2013-0003.

Sutton, Richard S. and Barto, Andrew G. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

Tang, Haoran, Houthooft, Rein, Foote, Davis, Stooke, Adam, Chen, Xi, Duan, Yan, Schulman, John, De Turck, Filip, and Abbeel, Pieter. # exploration: A study of count-based exploration for deep reinforcement learning. *arXiv preprint arXiv:1611.04717*, 2016.

Wang, Ziyu, Bapst, Victor, Heess, Nicolas, Mnih, Volodymyr, Munos, Remi, Kavukcuoglu, Koray, and de Freitas, Nando. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*, 2016.

Watkins, Christopher J. C. H. *Learning with Delayed Rewards*. PhD thesis, Psychology Department, University of Cambridge, England, 1989.

Wu, Yuhuai, Mansimov, Elman, Liao, Shun, Grosse, Roger, and Ba, Jimmy. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. *arXiv preprint arXiv:1708.05144*, 2017.

Zimmer, Matthieu and Doncieux, Stephane. Bootstrapping Q-learning for robotics from neuro-evolution results. *IEEE Transactions on Cognitive and Developmental Systems*, 2017. doi: 10.1109/TCDS.2016.2628817.

Appendices

A. Study of DDPG variability

In this section, we investigate performance variability in DDPG. The authors of (Henderson et al., 2017) showed that averaging the performance of two randomly selected splits of 5 runs with different random seeds can lead to statistically different distributions. This considerably undermines previous results, as most of the community ((Henderson et al., 2017) included) has been using 5 seeds or less, see (Henderson et al., 2017). Here we use a larger statistical sample of 20 random seeds and show that it is enough to counter the variance effect. We run the baseline DDPG algorithm with OU($\sigma = 0.3$) noise 40 times on Continuous Mountain Car and Half-Cheetah. We randomly select two pairs of 20 sets and perform statistical tests to compare their performance. We repeat this procedure 1000 times and report the percentages of tests showing a significant difference between both sets (Table 2(b) and 2(a)). We report results for both evaluation metrics and two tests: two-sample t-test (t-test) and bootstrapped (BS) estimation of the 95% confidence interval for the mean difference (positive if the interval does not contain 0).

Table 2. Percentage of tests showing differences between two sets of the same algorithm: DDPG OU(0.3) on (a) CMC, (b) HC.

(a)

	abs. metric	final metric
t-test	0.0%	0.0%
BS	0.0%	5.1%

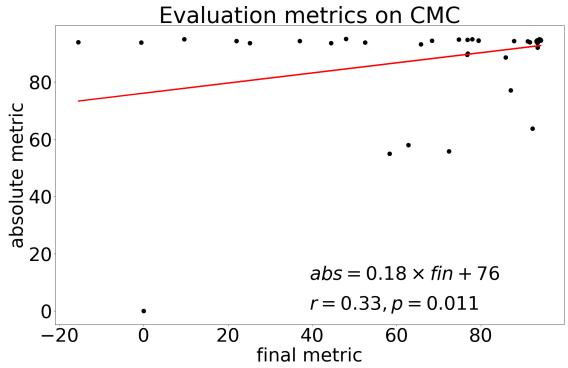
(b)

	abs. metric	final metric
t-test	0.0%	0.0%
BS	0.0%	0.0%

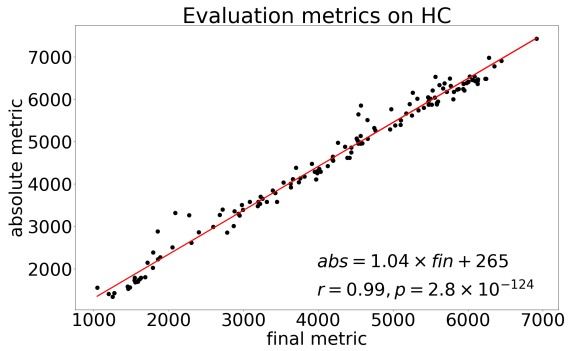
B. Correlation between evaluation metrics

We use two performance metrics: (1) the *absolute metric* is the average performance over 100 test episodes using the best controller over the whole learning process; (2) the *final metric*, corresponding to the evaluation methodology of (Henderson et al., 2017) is the average over the 100 last test episodes of the learning process, 10 episodes for each of the last 10 different controllers.

As highlighted in Appendix-Figure 5 of Appendix C, DDPG performance on CMC is highly unstable. The final metric only gives a measure of the final performance, which might not represent the algorithm's ability to find a good policy. Figure 1(a) shows this problem for the GEP-PG runs: even though the Pearson correlation coefficient is found significant, the line slope is far from 1. The final metric is highly variable whereas the absolute metric almost always shows a good performance. On the opposite, Appendix Figure 1(c) shows that final metrics and absolute metrics of DDPG performance on HC are highly correlated with a slope close to 1. This can be seen on Figures 1(c) and 2(c) where all learning curves are strictly increasing: the highest performance is always among the last ones. As a result, it is better to report the *absolute metric*, representing the performance of the best policy over a run. In the case of unstable learning as in CMC, we find it informative to present different runs so as to get a better sense of the learning dynamics. This is done in the next section.



(a)



(b)

Figure 4. Correlation of the two metrics of performance on (a) CMC and (b) HC. The equations of the lines of best fit (in red), the Pearson coefficients r and their associated p-values p are provided.

C. Individual runs on CMC

Figure 5 shows a representative example of 20 runs of DDPG with standard OU noise ($\mu = 0, \sigma = 0.3$). One can see that most runs never find the rewarding goal and that the learning performance of those which do so is unstable. As a result, the distribution of performances is not normal.

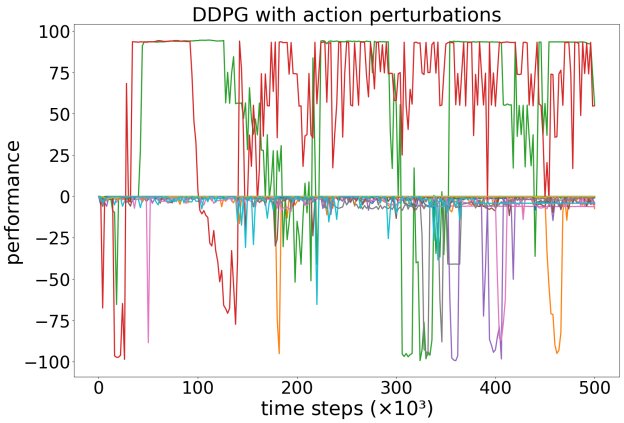


Figure 5. Twenty individual runs of DDPG OU(0.3) on CMC

D. Performance comparisons

Below we present statistical comparisons of the performance of various pairs of algorithms on HC. For comparisons, we use the 2-sample t-test (t-test) and a measure of the 95% confidence interval computed by bootstrap (BS) using 10^4 samples. The scipy implementation is used for t-test and the Facebook Bootstrapped implementation for BS[2]. For the t-test, we present the test value with the p-value in brackets, the difference being significant when $p \leq 0.05$. For BS, we present the mean and bounds of the 95% confidence interval for the difference between the two sets of performances. It is positive if the interval does not contain 0.

Table 3. DDPG param. pert. versus DDPG action pert.

	absolute metric	final metric
t-test	3.74 (6.0×10^{-4})	3.5 (1.2×10^{-3})
BS	1502 (736, 2272)	1364 (641, 2120)

DDPG with parameter perturbation achieves a significantly higher final (2/2) and absolute performance (2/2) than DDPG with action perturbation.

[2]https://github.com/facebookincubator/bootstrapped

Table 4. DDPG OU(0.3) versus decreasing OU(0.6), action pert.

	absolute metric	final metric
t-test	1.1 (0.29)	1.4 (0.16)
BS	425 (-330, 1196)	534 (-151, 1257)

There is no statistical evidence that DDPG with OU(0.3) achieves higher final or absolute performance than the decreasing OU(0.6) version.

Table 5. DDPG action pert. versus GEP

	absolute metric	final metric
t-test	5.3 (7.6×10^{-6})	4.6 (4.9×10^{-5})
BS	1804 (1151, 2476)	1491 (893, 2115)

DDPG with action perturbation achieves significantly higher final and absolute performance than GEP, 2/2 tests for both metrics.

Table 6. GEP linear policy versus GEP complex policy (64,64)

	absolute metric	final metric
KS	0.15 (0.96)	0.20 (0.77)
t-test	0.32 (0.75)	0.52 (0.60)
BS	787 (-404, 552)	128 (-350, 595)

The complexity of the GEP policy is not found to make any difference in absolute or final performance.

Table 7. GEP-PG versus DDPG with action perturbation

	absolute metric	final metric
t-test	3.2 (2.7×10^{-3})	3.0 (4.6×10^{-3})
BS	1089 (448, 1726)	924 (334, 1503)

GEP-PG achieves significantly higher absolute and final performance than DDPG, both using action perturbation (2/2 tests for both performance metrics)

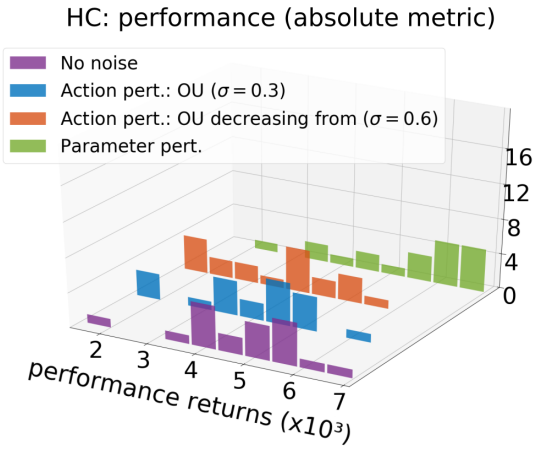
Table 8. GEP-PG versus DDPG, parameter perturbation

	absolute metric	final metric
t-test	2.1 (4.8×10^{-2})	2.4 (2.3×10^{-2})
BS	672 (39, 1261)	780 (143, 1378)

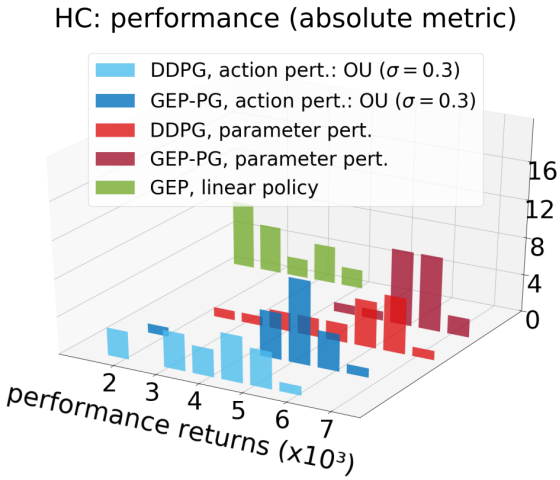
GEP-PG achieves significantly higher absolute and final performance than DDPG, both using parameter perturbation (2/2 tests with both performance metrics).

E. Histograms of performance on HC

Here we show the histograms of the absolute metrics of HC (Figure 6). We can see that the GEP-PG versions of DDPG algorithms show a smaller variance.



(a)



(b)

Figure 6. Absolute metric in HC for the various algorithms.(a) shows the influence of undirected exploration with perturbations on performance. (b) show the performances of GEP, DDPG and their combination GEP-PG on HC.

F. Influence of policy complexity in GEP

Using a linear or a more complex policy with GEP does not impact the final GEP performances on CMC or HC (no test over 5 shows significance). However, in the case of HC, the version of GEP using a simple linear policy achieves higher performance sooner. The statistical tests show significance at 2.10^5 steps with $p = 7.3 \times 10^{-4}$ for KS, $p = 2.3 \times 10^{-}4$ for t-test and a bootstrapped confidence interval of 738 $(395, 1080)$. This is important in terms of sample efficiency and a DDPG replay buffer of 2.10^5 samples filled by GEP would probably be of higher interest if the policy was linear. This supports the idea developed in Section 3.3 that a smaller policy parameter space might be faster to explore.

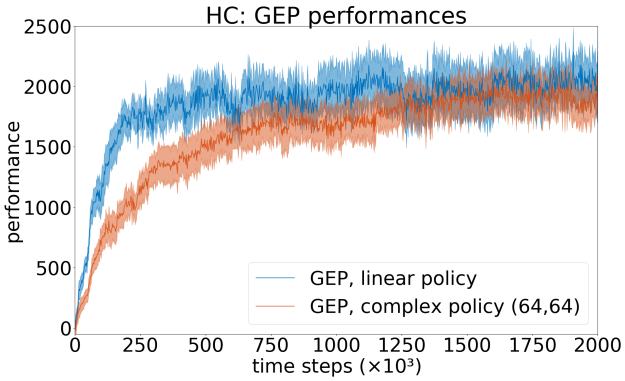


Figure 7. Performance of GEP for a linear and a policy with two hidden layers of (64,64) neurons.

G. Influence of the initial replay buffer content

Here we study the influence of the content of the replay buffer filled by GEP to bootstrap DDPG on the performance of GEP-PG. First, we found that the size of this buffer does not influence GEP-PG performance (from 100 to 2000 episodes), although too few episodes harms GEP-PG performance (< 100 episodes; Figure 8). Second, we found that GEP-PG performance correlates with the performance of the best GEP policy $(p < 2 \times 10^{-6})$ (Appendix-Figure 8) and the average performance of all GEP policies $(p < 4 \times 10^{-8})$. Third, we found correlations between GEP-PG performance and various measures of exploration diversity: 1) the standard deviation of GEP policies performances $(p < 3 \times 10^{-10})$; 2) the standard deviation of the observation vectors averaged across dimensions. This quantifies the diversity of sensory inputs. $(p < 3 \times 10^{-8})$; 3) the outcome diversity measured by the average distance to the k-nearest neighbors in outcome space (for various k). This measure is normalized by the average distance to the 1-nearest neighbor in the case of a uniform distribution, which makes it insensitive to the sample size $(p < 4 \times 10^{-10})$, see Appendix-Figure 9; 4) the percentage of cells filled when the outcome space is discretized (with various number of cells). We also use a number of cells equal to the number of points, which make the measure insensitive to this number $(p < 4 \times 10^{-5})$; 5) the discretized entropy with various number of cells $(p < 6 \times 10^{-7})$.

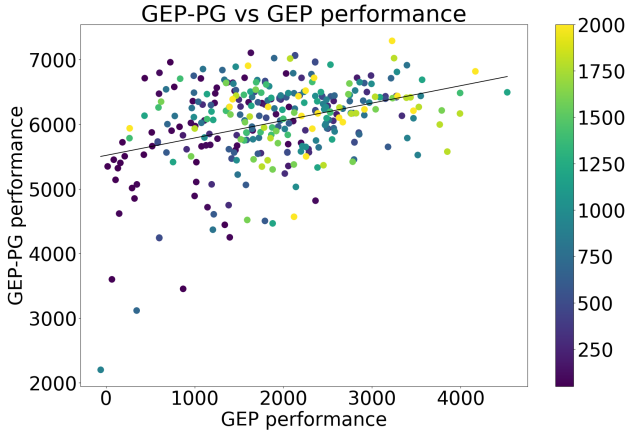


Figure 8. GEP-PG performance as a function of the GEP buffer performance. The GEP performance is evaluated by replaying 100 times the best policy found by GEP. Color maps for the size of the replay buffer (the number of episodes played by GEP).

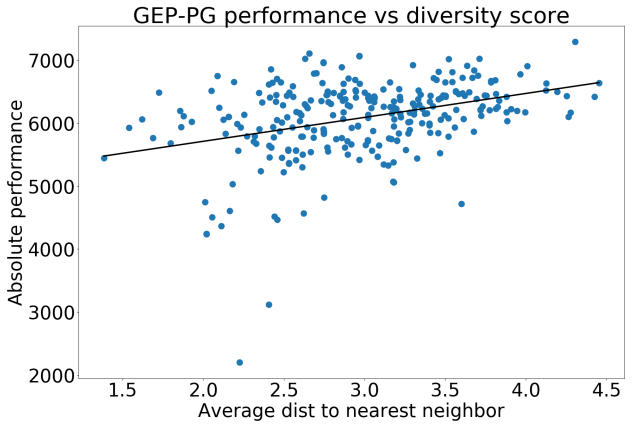


Figure 9. GEP-PG performance as a function of the buffer's diversity measure. The diversity measure is computed as the average distance to the nearest neighbor in outcome space. It is normalized by the expected value of this measure in the case of a uniform distribution of outcomes. This normalization makes the measure insensitive to the number of considered samples (here through the size of the buffer).

H. Sanity check

We could think of other exploration strategies to fill the replay buffer: a) using DDPG exploration with action perturbations to fill the buffer during 500 episodes (learning rate is zero); b) doing the same with DDPG parameter perturbations; c) using samples collected from random policies (RP-PG). Appendix-Figure 10 compares these strategies to DDPG with parameter perturbations and its corresponding GEP-PG version. Filling the replay buffer with exploration performed in the parameter space (b or c) seems not to impede DDPG performance and even reduces variance. Finally, GEP-PG still outperforms all versions of DDPG combined with undirected exploration strategies (2/2 tests positive on final metric, only bootstrap test on absolute metric).

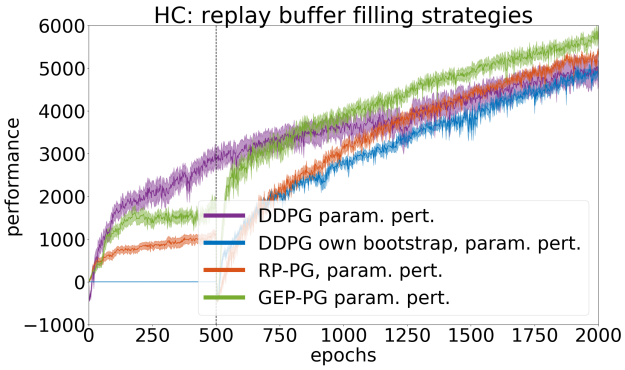


Figure 10. Influence of various exploration strategies to initialize DDPG's replay buffer on GEP-PG performance. In the second curve, the parameter perturbation exploration of DDPG is used. In the exploration phase (< 500 epochs), the networks are not updated. In the exploration phase of the third and fourth curves, Random Policy Search (RP) and Goal Exploration Process (GEP) are used respectively.