Multi-goal Reinforcement Learning has recently attracted a large amount of research interest. By allowing experience to be shared between related training tasks, this setting favors generalization for new tasks at test time, whenever some smoothness exists in the considered representation space of goals. However, in settings with discontinuities in state or goal spaces (e.g., walls in a maze), a majority of goals are difficult to reach, due to the sparsity of rewards in the absence of expert knowledge. This implies hard exploration, for which some curriculum of goals must be discovered, to help agents learn by adapting training tasks to their current capabilities. Building on recent automatic curriculum learning techniques for goal-conditioned policies, we propose a novel approach: Stein Variational Goal Generation (SVGG), which seeks at preferably sampling new goals in the zone of proximal development of the agent, by leveraging a learned model of its abilities, and a goal distribution modeled as particles in the exploration space. Our approach relies on Stein Variational Gradient Descent to dynamically attract the goal sampling distribution in areas of appropriate difficulty. We demonstrate the performances of the approach, in terms of success coverage in the goal space, compared to recent state-of-the-art RL methods for hard exploration problems.
For these difficult settings, curricula of goals must be discovered automatically, to help agents learn progressively, by adapting training tasks to their current capabilities. In that vein, many approaches have been developed recently to sample goals in the zone of proximal development of the agent [33], that is the current set of tasks an individual can make most progress on. Works such as [11] or [10] implement this notion from developmental psychology as sets of Goals of Intermediate Difficulty (GOIDs), that are dynamically updated according to the progression of the agent, arguing that those are the only goals providing some useful experience (i.e., non null gradients for policy optimization) for the learning process. Estimating the distribution of GOIDs in the state space is thus of crucial importance, to avoid many useless and often costly agent rollouts for tasks that are either too easy (i.e., nothing left to learn) or too difficult (i.e., no positive reward feedback). This is particularly true for large and continuous state spaces.

In this work, we introduce a novel model-free approach to overcome issues of existing automatic RL curricula methods [10, 32, 28], which either suffer from restricted exploration abilities or learning stability issues, especially in the setting of this work where the agent has no access to the desired goals distribution at train time. The aim is to pre-train agents to be able to reach any state of the environment, set as the goal that conditions their policy, in order to be ready for tasks that could be specified by the end user of the system. Our approach, called SVGG [2], relies on a model of the agent’s abilities, that is updated through time, from which we define a sampling goal distribution, that allocates important probability mass to areas of high entropy of success. Based on sets of particles to model this distribution, our approach outperforms methods working with a buffer of already reached states such as MEGA [28], and those that fully generate new goals from a generator distribution such as GoalGan [10] and Setter-Solver [30], to achieve state-of-the-art results for hard-exploration RL problems. The optimization of particles, which constitute goal candidates, is performed via Stein Variational Gradient Descent [20], that actively adapts goals to useful areas, even with a poor model of success or in the case of catastrophic forgetting.

2 Background and Related Work

In this paper, we consider the multi-goal reinforcement learning setting, defined as a Markov Decision Process (MDP) \( \mathcal{M}_g = < S, T, A, g, R_g > \), where \( S \) is a set of states, \( T \) is the set of transitions, \( A \) the set of actions and whose reward function \( R_g \) is parametrized by a goal \( g \) lying in the goal space \( G \). In our setting, each goal \( g \) is defined as a set of states \( S_g \subseteq S \) that are desirable situations for the corresponding task, with states in \( S_g \) being terminal states of the corresponding MDP. Thus, a goal \( g \) is considered achieved when the agent reached at step \( t \) any state \( s_t \in S_g \), which implies the following sparse reward function \( R_g : S \rightarrow \{0; 1\} \) in the absence of expert knowledge, where \( R_g(s_t, a_t, s_{t+1}) = I(s_{t+1} \in S_g) \) with \( I \) the indicator function. For simplicity reasons and without loss of generality, \( S_g \) reduces to a unique state in the following (i.e., \(|S_g| = 1\)).

Then, the objective is to learn a goal-conditioned policy \( \pi : S \times G \rightarrow A \) which maximizes the expected cumulative reward from any initial state of the environment, given a goal \( g \in G \): \( \pi^* = \arg \max_{\pi \in \Pi} \mathbb{E}_{s_0 \sim \rho_g, \tau \sim \pi(\cdot)}[\sum_{t=0}^{\infty} \gamma^t r^g_t] \), where \( r^g_t = R_g(s_t, a_t, s_{t+1}) \) stands for the goal-conditioned reward obtained at step \( t \) of trajectory \( \tau \) using goal \( g \), \( \gamma \) is a discount factor in \([0; 1]\) and \( \rho_g \) is the distribution of goals over \( G \). In our setting we consider that \( \rho_g \) is uniform over \( S \) (i.e., no known desired distribution), while the work could be extended to cover different distributions.

2.1 Automatic Curriculum for sparse Reward RL

Our SVGG method addresses automatic curriculum for sparse reward goal-conditioned RL (GCRL) problems, which allows to learn to achieve a continuum of related tasks.

Achieved Goals Distributions Our work is strongly related to the MEGA approach from [28], which (1) maintains a buffer of previously achieved goals, (2) models the distribution of achieved goals via a kernel density estimation (KDE), and (3) uses this distribution to define its behavior distribution. By preferably sampling from the buffer goals at the boundary of the set of already reached states, an increase of the support of that distribution is expected. In that way, MEGA aims at overcoming the limitations of previous related approaches which also model the distribution of achieved goals. For instance, DISCERN [34] only uses a replay buffer of goals whereas RIG [23] and Skew-fit [29] rather use variational auto-encoding [16] to encode the distribution. While RIG samples from the modeled achieved distribution, and DISCERN and Skew-Fit skew that distribution to sample more diverse achieved goals, MEGA rather focuses on low density regions of the distribution, with an explicit aim of expanding it. This results in improved exploration compared to competitors. Our work differs from all these works as they only model achieved goals, independently from what goal was targeted when they were achieved, whereas we model the capability of reaching target goals. This makes a strong difference since, while MEGA selects goals at the frontier of what it already discovered, nothing indicates
that goals \( g \) closer to the mode of the distribution can be achieved when they are targeted. MEGA is also prone to catastrophic forgetting, and limits exploration to goals from the replay buffer (goals need to be achieved once to serve as behavior goal candidates).

**Adversarial Goal Generation** Another trend of works proposed to adversarially learn a goal generator, that produces targets that are at the frontier of the agent’s capabilities. In that vein, GoalGAN \([10]\) simultaneously learns a discriminator to sort out non GOIDs or generated goals from GOIDs in the buffer of achieved goals, and a generator that aims at producing goals that fool the discriminator. While the approach is appealing, it is prone to many learning instabilities, with a generator that may usually diverge far from the space of reachable goals. Setter Solver \([30]\) stands as an extension of GoalGAN, where a goal setter is learned to provide goals of various levels of difficulty w.r.t. a judge network (similar to our success predictor, see next section), that remain close to already achieved goals, and that are sufficiently diverse to avoid mode collapse. One strong limitation of this approach lies on the relying on invertible networks to map from the latent space to the goal space, which severely limits the modeling power, and can reveal as very problematic for goals living in high dimensional spaces as images for instances. Asymmetric self-play \([32]\) is another way to generate goals, with a teacher agent that seeks at producing goals that are just behind the capabilities of the student agent. Both teacher and student are learned simultaneously, with an equilibrium of adverse rewards determined on their respective time to go. However, this equilibrium is particularly difficult to maintain, and the approach is prone to avoid many useful areas of the space. Finally, curiosity-driven exploration \([26]\) could be placed in that adversarial category, with an inverse model that aims at learning a compact latent representation space from which it is still possible to predict dynamics of the environment. The agent is then intrinsically rewarded if it explores areas whose dynamics are difficult to predict. Nevertheless, its model-based foundations, whose complex model learning may lead to hallucinations for the agent that can remain stuck in particularly difficult areas of the search space, greatly differs from our, simpler, model-free setting.

**Skills Discovery** Many recent works propose to deal with the problem of exploration based on a quantization of the state space in distinct skills, that are supposed to correspond to homogeneous sets of states that can be achieved with same kinds of policies. Based on variational empowerment \([13, 3]\) that approximates a posterior skill distribution via variational inference and mutual information maximization, the most famous approach of that category is certainly DIAYN \([9]\), which is based on a discriminator network which attempts at identifying for each trajectory the skill that the agent is currently using. Extensions such as \([13]\) propose to sequentially chain skills to deal with particularly complex environments. However, while this trend of approaches is appealing, their learning is particularly unstable, and the full exploration of the environment is not guaranteed.

**Population-based Exploration** Population-based reinforcement learning, using evolutionary strategies to explore environments, is getting increasing attention for hard exploration problems. Methods such as Novelty Search or Quality Diversity \([6, 21, 25]\) deal with populations of agents that they make evolve with diversity objectives considered as the fitness function to maximize. However, such approaches suffer from the hard problem of phenotype representation, which strongly impacts the performances. Also, such approaches are usually greatly less sample efficient than GCRL ones. Our SVGG approach can be considered as a trade-off between classical GCRL and evolutionary methods, by dealing with a population of goal particles, ensuring diversity in the space and benefiting from sample efficiency of gradient-based optimization.

### 2.2 Stein Variational Gradient Descent

Our method builds on Stein Variational Gradient Descent (SVGD) \([20]\) to approximate the distribution of goals of interest. SVGD is a powerful non-parametric tool for density estimation, when the partition function of the target distribution \( p \) to approximate is intractable. It stands as an efficient alternative to MCMC methods, which are proven to converge to the true distribution \( p \) but are usually too slow to be used in complex optimization processes. It also stands as an alternative to variational inference of parametric neural distributions \( q \), which are restricted to pre-specified families of distributions (e.g., Gaussian or mixtures of Gaussians) that may not fit target distributions. Its principle is to rather model the variational distribution \( q \) as a set of particles \( \{x_i\}_{i=1}^n \), each belonging to the support of \( p \).

The idea behind SVGD is to approximate the target distribution \( p \) with \( q \) by minimizing their KL-divergence: \( \min_q KL(q \parallel p) \). This objective is reached by iterative deterministic transforms as small perturbations of the identity map, on the set of particles: \( T(x) = x + \epsilon \phi(x) \), where \( \phi \) is a smooth transform function that indicates the direction of the perturbation, while \( \epsilon \) is the magnitude.

The authors draw a connection between KL-divergence and the **Stein operator** by showing that \( \nabla_\epsilon KL(q_T \parallel p) \mid_{\epsilon=0} = -E_{x \sim q} \text{trace}(A_p \phi(x)) \), where \( q_T \) is the distribution of particles after the transformation \( T \), \( A_p \phi(x) = \)
\( \phi(x) \nabla_x \log p(x)^T + \nabla_x \phi(x) \) being the Stein operator. The KL minimization objective is thus directly linked to the Stein Discrepancy, defined as: 
\[
S(q,p) = \max_{\phi \in \mathcal{F}} E_{x \sim q}[\text{trace}(A_{p,\phi}(x))].
\]
Note that \( S(q,p) = 0 \) only if \( q = p \).

Minimizing Stein Discrepancy being intractable as such, [19] and [4] introduce the Kernelised Stein Discrepancy (KSD) where the idea is to restrict to projections \( \phi \) that belong to the unit ball of a reproducing kernel Hilbert space \( \mathcal{H} \) (RKHS), for which there is a closed form solution. The KSD is defined as 
\[
S(q,p) = \max_{\phi \in \mathcal{H}} \left\{ E_{x \sim q}\left[ \text{trace}(A_{p,\phi}(x)) \right], \text{s.t. } ||\phi||_{\mathcal{H}} \leq 1 \right\},
\]
whose solution is given by: 
\[
\phi^*(\cdot) = E_{x \sim q}[A_{p,k}(x,\cdot)],
\]
where 
\[
k(x,x') = \exp(-\frac{1}{\theta} ||x-x'||^2_2)
\]
is commonly used.

Therefore, the steepest descent on the KL-objective is given by the optimal transform:
\[
x_i \leftarrow x_i + \epsilon \phi^*(x_i), \quad \forall i = 1 \cdots n,
\]
where
\[
\phi^*(x_i) = \frac{1}{n} \sum_{j=1}^{n} \left[ k(x_j,x_i) \nabla_{x_j} \log p(x_j) + \nabla_{x_j} k(x_j,x_i) \right].
\]

The “attractive force” in the update drives the particles toward high density areas of the target \( p \). The “repulsive force” pushes the particles away from each other, therefore fosters exploration and avoids mode collapse. Note that when \( n = 1 \), the update in (1) corresponds to a Maximum a Posteriori.

SVGD has already been successfully explored in the context of RL. The popular method Stein Variational Policy Gradient (SVPG) [22] employs SVGD to maintain a distribution of useful agents as particles. It strongly differs from our approach, since we consider particles as behavior goal candidates, while SVPG seeks at capturing the epistemic uncertainty about policy parameters. Please note that a mix between this orthogonal work with our SVGG could be considered as future work, to deal with policy and models uncertainty in addition to a useful distribution of GOIDs.

### 3 Stein Variational Goal Generation

We introduce our Stein Variational Goal Generation (SVGG) approach for curriculum learning in hard exploration RL problems. We build on SVGD to sample goals of appropriate difficulty for the RL agent.

We seek to sample goals from the following target distribution
\[
p_{\text{goals}}(g) \propto p_{\text{skills}}(g) \cdot p_{\text{valid}}(g).
\]

Figure 1: Overview of the SVGG method
The main component of this target is the distribution \( p_{\text{skills}} \), which assigns probability mass to areas of goals of appropriate difficulty, relying on a model of the agent’s current skills. The distribution \( p_{\text{valid}} \) acts as a prior that prevents the particles from moving too far away from the previously achieved goals, and thus from ending up in non-feasible areas of the goal space. Formal definitions of these two components are defined below.

**Model of the agent’s skills** The curriculum involved in \( p_{\text{skills}} \) relies on a learned model of the agent’s abilities to reach goals based on previous experiences. In our context, the model is a Neural Network \( D_{\phi} \) whose parameters \( \phi \) are learned by gradient descent on the following Binary Cross Entropy (BCE) loss:

\[
L_{\phi} = \sum_{(g^i, s^i) \in O} s^i (\log D_{\phi}(g^i)) + (1 - s^i) (\log (1 - D_{\phi}(g^i))),
\]

where \( O = \{ g^i, s^i \}_{i=1}^{n_B} \) is a batch of (goal, success) pairs coming from recent trajectories of the agent in the environment. We build on this model’s predictions to define our notion of GOIDs. We argue that the agent should avoid goals whose predicted probability of success is too high or too low (i.e., \( D_{\phi}(g) \approx 1 \) or \( D_{\phi}(g) \approx 0 \)). The experience that the agent can extract from such goals either because there is nothing else to learn on this task (i.e., null policy gradient) or because the agent never succeeds to reach the targeted goal (i.e., null reward).

To build \( p_{\text{skills}} \) based on the prediction of \( D_{\phi} \), we use a beta distribution whose maximum density point mass is determined according to the output of \( D_{\phi} \), by two hyperparameters \( \alpha \) and \( \beta \) that shape the distribution, as illustrated in Figure 2. In our experiments below, we report results for these 5 depicted settings (unless specified otherwise, the setting we use is the Medium one from the figure).

We define the distribution \( p_{\text{skills}} \) as an energy based density whose potential is the output of the beta distribution \( f \):

\[
p_{\text{skills}}(g) \propto \exp (f(\alpha, \beta, D_{\phi}(g))).
\]

**Prior distribution** In order to keep the sampled goals in the valid goal space of the environment \( G \), we need to define a prior distribution over the set of particles. We assume that the set of valid goals \( \mathcal{G} \) is not known a priori. However, we have access to the set of states already reached by the agent through an archive \( R \).

We would ideally have a prior that considers the posterior probability that a goal \( g \) belongs to \( \mathcal{G} \) given \( R \): \( p_{\text{valid}}(g) \propto \mathbb{P}(\text{g} \in \mathcal{G} | R) \). We progressively build such a distribution with One Class SVM (OCSVM). This model is mainly designed for outlier or novelty detection in absence of labeled data. Given a dataset \( X \in \mathbb{R}^d \), it defines a boundary of the data support in \( \mathbb{R}^d \), while keeping a small portion of the data points out of that boundary. Therefore, we use a calibrated version of a OCSVM model, using goals from \( R \) as the prior distribution:

\[
p_{\text{valid}}(g) \propto \mathbb{P}(\text{g} \in \tilde{\mathcal{G}} | R) = R_{\psi}(g),
\]

where \( R_{\psi}(g) \) is the output of the OCSVM model, with parameters \( \psi \), and \( \tilde{\mathcal{G}} \) is the set of states that the agent is or was able to reach until the current iteration.

As the agent progresses and expands its set of achieved states through training, it eventually reaches the environment boundary. In this case \( p_{\text{valid}}(g) \propto \mathbb{P}(\text{g} \in \tilde{\mathcal{G}} | R) \approx \mathbb{P}(\text{g} \in \mathcal{G} | R) \).

The pseudo-code of SVGG is given in Algorithm 1. After a random initialization of the particles set \( \Omega \), the approach collects data by running the agent with sampled goals corresponding to particles from \( \Omega \), which are used to feed learning of success predictor \( D_{\phi} \) and reachability predictor \( R_{\psi} \). From these adjusted models, \( p_{\text{goals}} \) is used to compute transforms of the particles in \( S \). At the end of each epoch, an off-Policy RL algorithm is used to improve the agent.

**Theorem 1. Recovery property:** Assume that particles are not allowed to leave a compact set \( C \), with \( \mathcal{G} \subset C \). Let us denote as \( \mathcal{G}^+ \) the set of goals \( g \in C \) such that \( D_{\phi}(g) \to 1 \). Assume that, at a given iteration \( l \), \( \tilde{\mathcal{G}} \subset \mathcal{G}^+ \) (i.e., the success
Algorithm 1 RL with Stein Variational Goal Generation

1: **Input**: a goal-conditioned policy \( \pi_\theta \), a parameterizable environment \( \mathcal{M} \), a number of particles \( n \), a success predictor \( D_\phi \), a reachability predictor \( R_\psi \), buffers of transitions \( \mathcal{B} \), reached states \( \mathcal{R} \) and success outcomes \( \mathcal{O} \), a kernel \( k \), numbers \( n, l(r), l(m), l(p), l(\tau) \) and \( l(\nu) \).

2: Sample a set of particles \( \Omega = \{x^i\}_{i=1}^n \) uniformly from states reached by pre-runs of \( \pi_\theta \);

3: for \( n \) epochs do \( \triangleright \) **Data Collection**

4: for \( l(\tau) \) iterations do

5: Sample a batch \( \{g^i\}_{i=1}^c \) from \( \Omega \);

6: for \( i \) from 1 to \( l(\tau) \) do

7: \( \tau \leftarrow \text{Rollout} \pi_\theta(\cdot|\cdot, g = g^i) \);

8: Store all \( (s_t, a_t, s_{t+1}, r_t, g^i) \) from \( \tau \) in \( \mathcal{B} \) and every \( s_t \) from \( \mathcal{R} \);

9: Optionally (HER): Store relabeled transitions from \( \tau \);

10: Store outcome \( (g^i, I(s_t|\tau) \approx g^i) \) in \( \mathcal{O} \);

11: end for

12: end for

13: for \( l(m) \) iterations do \( \triangleright \) **Model Update**

14: Sample a batch \( \{(g^i, s^i)\}_{i=1}^m \) from \( \mathcal{O} \);

15: Update model \( D_\phi \) (e.g., via ADAM), with gradient of \( \mathcal{L}_\phi \) (3);

16: end for

17: Update model \( R_\psi \) according to all states in \( \mathcal{R} \); \( \triangleright \) **Prior Update**

18: for \( l(p) \) iterations do \( \triangleright \) **Particles Update**

19: Compute the density of the target \( p_{\text{goals}} \) for the set of particles \( \Omega \) using 2.

20: Compute transforms: \( \phi^*(x_i) = \frac{1}{n} \sum_{j=1}^{n} \left[ k(x_j, x_i) \nabla x_j \log p_{\text{goals}}(x_j) + \nabla x_j k(x_j, x_i) \right] \);

21: Update particles \( x_i \leftarrow x_i + \epsilon \phi^*(x_i) \), \( \forall i = 1 \cdots n \);

22: end for

23: Improve agent with any Off-Policy RL algorithm \( \triangleright \) **Agent Improvement**

(e.g., DDPG) using transitions in \( \mathcal{B} \);

24: end for

The above theorem ensures that, even if the success model over-estimates the capacities of the agent for some area \( \omega \) (e.g., due to changes in the environment, catastrophic forgetting or success model error), some particles are likely to go back to this area once every goal in \( \mathcal{G} \) looks well covered by the agent, with an increasing probability for more particles. This way, the process can reconsider over-estimated areas, by sampling again goals in them, and hence correcting the corresponding predictions, which leads to attract attention of \( p_{\text{goals}} \) back to that difficult areas. Approaches such as MEGs do not exhibit such recovery properties, since they always sample at the boundary of \( \mathcal{G} \), which is likely to incrementally grow towards \( \mathcal{G} \). This theoretical guarantee of SVGG is empirically validated by the experiment from Figure (4) which shows the good recovering property of our approach, after a sudden change in the environment.

4 Experiments

To measure SVGG’s performance and compare it to our baselines, we evaluate the resulting policy on the entire space of valid goals \( \mathcal{G} \) in our environment. We consider a success coverage metric, defined as \( S(\pi) = \frac{1}{|\mathcal{G}|} \int_{\mathcal{G}} P(\pi \text{ reaches } g) \, dg \)

\[\text{as assumed for instance in } [18]. \text{ This is not always true, but gives strong insights about the behavior of SVGD. Please refer to } [12] \text{ for more discussions about KSD.} \]
The goal space being continuous, we evaluate the policy on a finite uniform subset $\hat{G} \subset \mathcal{G}$. Then our objective reduces to:

$$S(\pi) = \frac{1}{|\hat{G}|} \sum_{i=1}^{|\hat{G}|} P(\pi \text{ reaches } g_i) = \frac{1}{|\hat{G}|} \sum_{i=1}^{|\hat{G}|} E_{\tau \sim \pi} [\mathbb{1}\{\exists s \in \tau, ||s - g_i||_2 < \delta\}].$$

(6)

To build $\hat{G}$, we split $\mathcal{G}$ into areas following a regular grid, and then uniformly sample 30 goals inside each part of the division.

In this section, we provide experimental results to investigate the following questions: 1) Does SVGG maximize the success coverage objective compared to recent intrinsic motivation work? 2) What target difficulty (i.e., beta distribution as described above) is more efficient to maximize coverage? 3) Is SVGG robust to environment changes? 4) Is SVGG Robust to agent’s and model catastrophic forgetting?

Our main results are based on a Point-Maze environment: an agent moves a point without mass within a 2d maze with continuous spaces of states and actions. Due to the walls, the goal space presents many discontinuities. In the absence of expert knowledge, introduced via reward shaping or a given curriculum, the success coverage objective is very hard to maximize. Additional results on different environments are reported in appendix. To conduct our experiments, we rely on the modular RL code base [27].

4.1 Compared Approaches

As baselines, we choose two methods from the literature that span the existing trends in unsupervised RL with goal conditioned policies (GCP), MEGA [28] and Goal-GAN [10]. In addition, the Random baseline randomly selects the behavior goal among past achieved states. We also perform two SVGG ablations: a No Prior version, which considers $p_{\text{goals}} \propto p_{\text{skills}}$ and a Only Prior version, where $p_{\text{goals}} \propto p_{\text{valid}}$. All compared approaches use DDPG and HER with a mixed strategy described in the appendix. In every version of SVGG, we use $n = 100$ particles to approximate the target distribution. Implementation details about all considered architectures are given in Appendix.

4.2 Main results

Our main results in Figure 3 shows the success coverage of our baselines over 4 different maze environments. SVGG is the only method that discovers and reaches all valid goals in 4M agent steps in all considered environments.

![Figure 3: Success coverage results, evaluated throughout training in 4 different mazes environment (4 seeds each). SVGG outperforms recent intrinsic motivation algorithms (MEGA and Goal-GAN), as well as naive baselines.](image)

Due to the known stability issues in GAN training, Goal-GAN is the least efficient of our baselines. Another explanation of the failure of Goal-GAN is that the generator is likely to output goals outside the valid space, which is not the case for MEGA or SVGG. MEGA chooses the behavior goal in a Replay Buffer of achieved goals, while the prior distribution $p_{\text{valid}}$ considered in SVGG keeps particles inside valid areas. Nevertheless, we observe that Goal-GAN performs better in environments with linear structures, such as Mazes 3 and 4. On the other hand, it has a hard time to generate goals of intermediate difficulty when the structure is random, as in Mazes 1 and 2.
The minimum density heuristic of MEGA efficiently discovers all feasible goals in the environment, but our results show that its success plateaus in almost all the considered environments. MEGA’s intrinsic motivation only relies on state novelty. Thus, when the agent has discovered all feasible states, it is unable to target areas that the agent has reached in the past but has not mastered yet.

**Recovery Property** In Figure 4, we demonstrate the advantages of SVGG’s intrinsic motivation over MEGA’s in a changing environment. Walls are suddenly added in the mazes during the training process, after the methods had solved the entire goal space (dot red line from the figure). Curves show that MEGA is unable to fully adapt to the new maze setting, which corresponds to Maze 0 from Figure 1 (although MEGA solved Maze 0 in 1M steps in that experiment with constant conditions). On the other hand, SVGG is able to discover new difficulties due to the environment modification and focuses on those to finally solve the entire maze in less than 1.5M steps.

We also observe that the advantages of our method over MEGA in terms of recovery ability are more significant when changes in the environment are more drastic (i.e., when starting from maze A).

**Control of Sampling Difficulty** Using beta distributions reveals what goals difficulty an intrinsic motivation algorithm should aim at for an agent to efficiently explore and control an environment. In Figure 5, we compare 5 versions of SVGG using different distributions from Figure 2, on the 4 previously considered mazes. One can observe that extreme targets difficulties are the least effective ones, especially the Very easy, which is too conservative to efficiently explore new areas of the space. On the other hand, SVGG performs very well with Medium and Hard distributions. This suggests that the optimal goal difficulty is somewhere between medium and hard.

**SVGG analysis** To gain further intuition on how SVGG maximizes the success coverage, we show in Figure 6 the evolution of the particles throughout training. As the agent progresses and reaches novel and harder goals, the $D_\phi$ model updates its predictions. Thus, the target distribution $p_{\text{goals}}$ is updated accordingly (background color of the 2nd row of the figure). The particles $q = \{g_i\}_{i=1}^n$ are then moved toward new areas of intermediate difficulty through SVGD to minimize $KL(q || p_{\text{goals}})$.

Figure 6 also highlights the recovery property of SVGG. When the model detects that the agent has nothing else to learn in the environment, $p_{\text{goals}}$ reduces to $p_{\text{valid}}$, that is at this point uniform on the entire goal space. Therefore, the particles spread uniformly over the environment and thus prevent SVGG from catastrophic forgetting, as the model rediscovers areas that the agent has forgotten how to reach (cf. rightmost column in Figure 6).
5 Conclusion

This paper introduces a new approach for multi-goal reinforcement learning in deterministic environments. Our algorithm, SVGG, leverages Stein Variational Gradient Descent to monitor a model w.r.t. its goal reaching capabilities. Using this model, the agent addresses goals of intermediate difficulty, resulting in an efficient curriculum for finally covering the whole goal space. Moreover, SVGG can recover from catastrophic forgetting by means of SVGD particle optimization, which is a classic pitfall in intrinsically motivated RL.

Studying the impact of the number of particles is left for future work. Actually, the target distribution being in constant evolution, the KL divergence minimization objective is hard to reach at all times, which makes it difficult to claim that using more particles is always better. Furthermore, a previous work [7] spotted exploration failures in SVGD, and suggests that periodically annealing the attraction force in particle optimization is required to enable particles to cover non-trivial distributions, e.g. in multimodal settings or in high dimensions.

In the future, we also envision to address larger and stochastic environments, and the case where the goal space is learned from representation learning techniques.

References

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

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[3] Jongwook Choi, Archit Sharma, Honglak Lee, Sergey Levine, and Shixiang Shane Gu. Variational empowerment as representation learning for goal-based reinforcement learning. arXiv preprint arXiv:2106.01404, 2021.

[4] Kacper Chwialkowski, Heiko Strathmann, and Arthur Gretton. A kernel test of goodness of fit. In Maria Florina Balcan and Kilian Q. Weinberger, editors, Proceedings of The 33rd International Conference on Machine Learning, volume 48 of Proceedings of Machine Learning Research, pages 2606–2615, New York, New York, USA, 20–22 Jun 2016. PMLR.

[5] Cédric Colas, Pierre Fournier, Olivier Sigaud, and Pierre-Yves Oudeyer. CURIOUS: intrinsically motivated multi-task multi-goal reinforcement learning. In ICML, 2018.

[6] Edoardo Conti, Vashisht Madhavan, Felipe Petroski Such, Joel Lehman, Kenneth Stanley, and Jeff Clune. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. Advances in neural information processing systems, 31, 2018.

[7] Francesco D’Angelo and Vincent Fortuin. Annealed stein variational gradient descent. CoRR, abs/2101.09815, 2021.
[8] Benjamin Eysenbach, Xinyang Geng, Sergey Levine, and Ruslan Salakhutdinov. Rewriting history with inverse RL: hindsight inference for policy improvement. CoRR, abs/2002.11089, 2020.

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

[10] Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In International conference on machine learning, pages 1515–1528. PMLR, 2018.

[11] Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. In Conference on robot learning, pages 482–495. PMLR, 2017.

[12] Jackson Gorham and Lester Mackey. Measuring sample quality with kernels. In International Conference on Machine Learning, pages 1292–1301. PMLR, 2017.

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

[14] Leslie Pack Kaelbling. Learning to achieve goals. In IJCAI, pages 1094–1099. Citeseer, 1993.

[15] Pierre-Alexandre Kamienny, Jean Tarbouriech, Alessandro Lazaric, and Ludovic Denoyer. Direct then diffuse: Incremental unsupervised skill discovery for state covering and goal reaching. CoRR, abs/2110.14457, 2021.

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

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

[18] Qiang Liu. Stein variational gradient descent as gradient flow. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 30. Curran Associates, Inc., 2017.

[19] Qiang Liu, Jason Lee, and Michael Jordan. A kernelized stein discrepancy for goodness-of-fit tests. In Maria Florina Balcan and Kilian Q. Weinberger, editors, Proceedings of The 33rd International Conference on Machine Learning, volume 48 of Proceedings of Machine Learning Research, pages 276–284, New York, New York, USA, 20–22 Jun 2016. PMLR.

[20] Qiang Liu and Dilin Wang. Stein variational gradient descent: A general purpose bayesian inference algorithm. arXiv preprint arXiv:1608.04471, 2016.

[21] Qihao Liu, Xiaofeng Liu, and Guoping Cai. Pns: Population-guided novelty search learning method for reinforcement learning. arXiv preprint arXiv:1811.10264, 2018.

[22] Yang Liu, Prajit Ramachandran, Qiang Liu, and Jian Peng. Stein variational policy gradient. arXiv preprint arXiv:1704.02399, 2017.

[23] Ashvin Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. arXiv preprint arXiv:1807.04742, 2018.

[24] Ian Osband, Benjamin Van Roy, and Zheng Wen. Generalization and exploration via randomized value functions. In International Conference on Machine Learning, pages 2377–2386. PMLR, 2016.

[25] Giuseppe Paolo, Alexandre Coninx, Stéphane Doncieux, and Alban Laflaquière. Sparse reward exploration via novelty search and emitters. arXiv preprint arXiv:2102.03140, 2021.

[26] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In Doina Precup and Yee Whye Teh, editors, Proceedings of the 34th International Conference on Machine Learning, volume 70 of Proceedings of Machine Learning Research, pages 2778–2787. PMLR, 06–11 Aug 2017.

[27] Silviu Pitis, Harris Chan, and Stephen Zhao. mrl: modular rl. https://github.com/spitis/mrl 2020.

[28] Silviu Pitis, Harris Chan, Stephen Zhao, Bradly Stadie, and Jimmy Ba. Maximum entropy gain exploration for long horizon multi-goal reinforcement learning. In Hal Daumé III and Aarti Singh, editors, Proceedings of the 37th International Conference on Machine Learning, volume 119 of Proceedings of Machine Learning Research, pages 7750–7761. PMLR, 13–18 Jul 2020.

[29] Vitchyr H Pong, Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl, and Sergey Levine. Skew-fit: State-covering self-supervised reinforcement learning. arXiv preprint arXiv:1903.03698, 2019.

[30] Sebastien Racaniere, Andrew Lampinen, Adam Santoro, David Reichert, Vlad Firoiu, and Timothy Lillicrap. Automated curriculum generation through setter-solver interactions. In International Conference on Learning Representations, 2019.
A Proof of Theorem 1

**Theorem 1. Recovery property:** Assume that particles are not allowed to leave a compact set \( C \), with \( G \subseteq C \). Let us denote as \( G^+ \) the set of goals \( g \in C \) such that \( D_\omega(g) \rightarrow 1 \). Assume that, at a given iteration \( l \), \( \hat{G} \subseteq G^+ \) (i.e., the success model \( D_\omega \) is confident in capabilities of the agent for all already observed goals), \( R_\omega(g) \rightarrow 1 \) for every \( g \in \hat{G} \) (i.e., the reachability predictor \( R_\omega \) identifies correctly all already reached goals - very mild assumption) and \( R_\omega(g) \rightarrow 0 \) for every \( g \in G \) such that \( D_\omega(g) < 1 \) (i.e., the reachability model is more conservative than the success one). Assume also that we use a kernel which ensures that the Kernelized Stein Discrepancy (KSD) of any distribution \( q \) with \( p_{\text{goals}} \) is 0 only if \( q \) weakly converged to \( p_{\text{goals}} \). Then, with no updates of the models after iteration \( l \), and a sufficiently high number of particles \( n \), for any area \( \omega \subseteq G^+ \) with volume \( v(\omega) \), the probability of having at least one particle in \( \omega \) at convergence is at least \( 1 - \left(1 - \frac{v(\omega)}{v(C)}\right)^n \).

**Proof.** In settings where the KSD of a distribution \( \mu \) from a target \( p \) is 0 only if \( \mu \) weakly converged to \( p \), [18] previously proved that, for any compact set, the empirical measure \( \mu^n \) of \( \mu \), computed on a set of \( n \) particles, converges weakly towards the target \( p \) when a sufficiently number of particles is used. Thus, under our hypotheses, the set of particles of SVGG appropriately represents \( p_{\text{goals}} \) after a sufficient number of steps of Stein transforms.

Now, we know that \( p_{\text{goals}} \) is maximal and constant for any \( g \in G^+ \), since \( p_{\text{skills}}(g) \) and \( p_{\text{valid}}(g) \) are both constant over this set and \( p_{\text{valid}}(g) \rightarrow 0 \) outside \( G^+ \). Thus, particles are attracted towards \( G^+ \), which corresponds to the mode of \( p_{\text{goals}} \), over which they are supposed to be uniformly spread. In that setting, we thus know that, for any particle \( x_i \) and area \( \omega \in G^+ \) with volume \( v(\omega) \), we have:

\[
P(x_i \in \omega) = P(x_i \in \omega | x_i \in G^+)P(x_i \in G^+) + P(x_i \in \omega | x_i \not\in G^+)P(x_i \not\in G^+) \\
= P(x_i \in \omega | x_i \in G^+)P(x_i \in G^+) \\
= \frac{v(\omega)}{v(G^+)}P(x_i \in G^+) \\
\geq \frac{v(\omega)}{v(G^+)} \frac{v(G^+)}{v(C)} = \frac{v(\omega)}{v(C)},
\]

where the inequality comes from the higher concentration of particles in \( G^+ \) than in other parts of \( C \), since \( G^+ \) corresponds to the mode of \( p_{\text{goals}} \).

Thus, the probability for having no particle inside a given area \( \omega \in G^+ \) with volume \( v(\omega) \) is:

\[
P(\bar{\omega}i \in [1, n], x_i \in \omega) = \prod_{i=1}^n P(x_i \not\in \omega | \bar{\omega}j \in [1, i-1], x_j \in \omega) \\
\leq \prod_{i=1}^n P(x_i \not\in \omega) = \left(1 - \frac{v(\omega)}{v(C)}\right)^n.
\]

The probability of having at least one particle in any area \( \omega \in G^+ \) is thus lower bounded by \( 1 - \left(1 - \frac{v(\omega)}{v(C)}\right)^n \).

---

\(^5\)As assumed for instance in [18]. This is not always true, but gives strong insights about the behavior of SVGD. Please refer to [12] for more discussions about KSD.
B Visualization of goals

![Visualization of goals](image)

Figure 7: Visualization of behavioral goal (1st row), achieved goals (2nd row) and success coverage (3rd row) for 1M steps in the Maze 1 environment.

We visualize behavioral and achieved goals in Maze 1 (Figure 7) and Maze 2 (Figure 8), in parallel with the success coverage after training. The main advantages of our method lie in the capacity to target difficult areas and avoid catastrophic forgetting, which results in nearly optimal success coverage. We observe that MEGA efficiently discover the environment, but is unable to detect where the agent has not mastered yet a goal in order to target it. This also leads to catastrophic forgetting and a lack of adaptation in front of environment changes, a case studied in the main paper.

One can see that the generation of GOIDs in Goal-GAN is very instable and tricky in such discontinuous goal space, especially as the generator is susceptible to output goals outside the environment boundary, which SVGG avoids with the prior distribution.

C Experiments and Implementation details

C.1 Environments

**Pointmaze** We use a 2D pointmaze environment where the state space and goal space are (x, y) coordinates (the agent cannot see the walls), and the agent moves according to its action, which is a 2-dimensional vector with dimensions constrained to \([-0.95, 0.95]\]. All methods are tested on 4 5 × 5 mazes with different shapes and a highly discontinuous goal space. The maximum number of steps in one trajectory is set to 30. We argue that the difficulty of these environments does not lie in their size, but rather in the complexity of their goal space and thus of the trajectories adopted by the agent to reach these goals.

**Fetch Pick and Place (Hard version)** To test the versatility of our method, we conduct additional experiments on a hard version of FetchPickAndPlace-v1 from OpenAI gym [2]. The agent controls a robotic arm that must pick and
place a block to a 3D desired location. In the hard version, the target goals are placed between 20 and 45cm in the air, while in the original version, 50% of target goals are on the table and the remaining ones are placed between 0 and 45cm in the air.

While this environment presents a greater difficulty in terms of control (the action dimension is 4), the goal generation component is significantly easier as the goal space is smooth, there is no obstacle for the agent to bypass. As we can see in Figure 9, SVGG is able to solve the environment within 1.5M steps, but does not significantly outperform MEGA.

We argue that the interest of our goal generation algorithm resides in environment with highly discontinuous goal space as the action control is largely supported by the RL algorithm (e.g. DDPG). Therefore, the smaller difference between MEGA and SVGG in this environment was expected, as SVGG is mainly designed to target non-smooth goal spaces, and to avoid pitfalls such as catastrophic forgetting. Experiment on environments that are complex both in terms of goals and control are left for future work.

C.2 Baselines

The authors of MEGA [28] train a GCP with previously achieved goals from a replay buffer. Their choice of goals relies on a minimum density heuristic, where they model the distribution of achieved goals with a KDE. They argue that aiming at novel goals suffices to efficiently discover and control the environment. We use the original implementation of the authors, the pseudocode is described in Algorithm 2.

Goal-GAN [10] uses a procedural goal generation method based on GAN training. As our SVGG, it aims at sampling goals of intermediate difficulty, which they define as $G_{\text{GOID}} = \{g \mid P_{\text{min}} < P_{\pi}(g) < P_{\text{max}}\}$, $P_{\pi}(g)$ being the probability for the policy $\pi$ to reach goal $g$, $P_{\text{min}}$ and $P_{\text{max}}$ are hyperparameters. A Discriminator $D_{\theta_d}$ is trained to distinguish between goals in $G_{\text{GOID}}$ and other goals, while a generator $G_{\theta_g}$ is trained to output goals in $G_{\text{GOID}}$ by relying on the
Algorithm 2 MEGA

1: Input: a goal-conditioned policy $\pi_\theta$, a parameterizable environment $M$, reached states $\mathcal{R}$, a KDE model $P_{as}$ of the achieved states, numbers $c, n, N, l^{(r)}$ and $l^{(c)}$.
2: Initialize $\mathcal{R}$ with states reached by pre-runs of $\pi_\theta$;
3: for $n$ epochs do ▷ Data Collection
   4: for $t^{(r)}$ iterations do
      5: Sample a batch $\{g_i\}_{i=1}^N$ uniformly from $\mathcal{R}$;
      6: Eliminate unachievable candidates if their $Q$-values are below cutoff $c$;
      7: Choose goals $\{g_i\}_{i=1}^{l^{(r)}} = \arg\min_{g_i} P_{as}(g_i)$
      8: for $i$ from 1 to $l^{(r)}$ do
         9: $\tau \leftarrow$ Rollout $\pi_\theta(.,.|g=g^i)$;
      10: Store all $(s_t, a_t, s_{t+1}, r_t, g^i)$ from $\tau$ in $\mathcal{B}$ and every $s_t$ from $\tau$ in $\mathcal{R}$;
     11: Optionally (HER): Store relabeled transitions from $\tau$;
     12: Store outcome $(g^i, I(s_{|\tau|} \approx g^i))$ in $\mathcal{O}$;
   13: end for
   14: end for
15: Update KDE model $P_{as}$ with uniform sample from $\mathcal{R}$; ▷ KDE Update
16: Improve agent with any Off-Policy RL algorithm (e.g., DDPG) using transitions in $\mathcal{B}$; ▷ Agent Improvement
17: end for

discriminator outputs. They optimize $G_{\theta_g}$ and $D_{\theta_d}$ in a manner similar to the Least-Squares GAN (LSGAN) with the following losses

$$\begin{align*}
\min_{\theta_d} V(D_{\theta_d}) &= \mathbb{E}_{g \sim \mathcal{R}} [y_g (D_{\theta_d}(g) - 1)^2 + (1 - y_g) (D_{\theta_d}(g) + 1)^2] \\
\min_{\theta_g} V(G_{\theta_g}) &= \mathbb{E}_{z \sim \mathcal{N}(0,1)} [D_{\theta_d}(G_{\theta_g}(z))^2],
\end{align*}$$

(7)

where $y_g$ is the label that indicates whether $g$ belongs to $G_{\text{GOLD}}$ or not. In [10], the authors use Monte-Carlo sampling of the policy to estimate $y_g$. For efficiency reasons, we use a learned model of the agent’s capabilities as in SVGG. The pseudocode is described in Algorithm 3.

C.3 Hindsight Experience Replay

The original goal relabeling strategy introduced in HER [1] is the future, which consist in relabeling a given transition with a goal achieved on the trajectory later on. This is very effective in sparse reward setting to learn a GCP. However, many works suggested that relabeling transitions with goals outside the current trajectory helps the agent generalize.
Algorithm 3 Goal-GAN

1: **Input:** a goal-conditioned policy $\pi_\theta$, a parameterizable environment $\mathcal{M}$, a goal Generator $G_{\theta_g}$, a Discriminator $D_{\theta_d}$, a success predictor $D_{\phi}$, buffers of transitions $\mathcal{B}$, reached states $\mathcal{R}$ and success outcomes $\mathcal{O}$, numbers $n$, $P_{\text{min}}$, $P_{\text{max}}$, $t(r)$, $l(r)$, $l(m)$, $k(m)$, $l(g)$ and $k(g)$.

2: Initialize $G_{\theta_g}$ and $D_{\theta_d}$ with pre-runs of $\pi_\theta$;

3: for $n$ epochs do

\[\text{Data Collection}\]

4: for $t(r)$ iterations do

5: Sample noise $\{z_i\}_{i=1}^{l(r)} \sim \mathcal{N}(0, 1)$

6: generate $\{g^i\}_{i=1}^{l(r)} = G_{\theta_g}(\{z_i\}_{i=1}^{l(r)})$

7: for $i$ from 1 to $l(r)$ do

8: $\tau \leftarrow$ Rollout $\pi_\theta(\cdot, g = g^i)$;

9: Store all $(s_t, a_t, s_{t+1}, r_t, g^i)$ from $\tau$ in $\mathcal{B}$ and every $s_t$ from $\tau$ in $\mathcal{R}$;

10: Optionally (HER): Store relabeled transitions from $\tau$;

11: Store outcome $(g^i, l(s|\tau| \approx g^i))$ in $\mathcal{O}$;

end for

12: end for

\[\text{GAN training}\]

13: Sample a batch $\{(g^i, s^i)\}_{i=1}^{l(g)}$ from $\mathcal{O}$;

14: Label goals (GOID or not) with model $D_{\phi}$: $\{y_{gi}\}_{i=1}^{l(g)} = \{P_{\text{min}} < D_{\phi}(g_i) < P_{\text{max}}\}_{i=1}^{l(g)}$

15: for $k(g)$ iterations do

16: Update $G_{\theta_g}$ and $D_{\theta_d}$ (e.g. with ADAM) with gradients of LSGAN losses; (7)

end for

\[\text{Model Update}\]

17: Sample a batch $\{(g^i, s^i)\}_{i=1}^{l(m)}$ from $\mathcal{O}$;

18: for $k(m)$ iterations do

19: Update model $D_{\phi}$ (e.g., via ADAM), with gradient of $\mathcal{L}_{\phi}$ (3);

end for

20: Improve agent with any Off-Policy RL algorithm (e.g., DDPG) using transitions in $\mathcal{B}$;

end for

\[\text{Agent Improvement}\]

across trajectories. For example, one can use inverse RL to determine the optimal goal to relabel a given transition [8]. We use a naive version of this method. As in [28], we relabel transitions for DDPG optimization using a mixed strategy. All methods presented in this work use the same strategy.

10% of real experience are kept while 40% of the transitions are relabeled using the future strategy of HER. We relabel the remaining 50% transitions with goals outside of their trajectory, with randomly sampled goals among the past behavioral and achieved goals. The latter part of the strategy helps share information between different trajectories that often contains similar transitions.
## C.4 Architectures and Hyperparameters

| Hyper-Parameter                                                                 | Value          |
|-------------------------------------------------------------------------------|---------------|
| **DDPG**                                                                      |               |
| Batch size for replay buffer                                                   | 2000          |
| Discount factor $\gamma$                                                      | 0.99          |
| Action L2 regularization                                                       | 1e-1          |
| Action noise maximum std (Gaussian noise)                                     | 0.1           |
| Warm up steps before training                                                  | 2500          |
| Actor learning rate                                                            | 1e-3          |
| Critic learning rate                                                           | 1e-3          |
| Target Network update fraction for soft parameters update update               | 0.05          |
| Activation for actor and critic networks                                       | Gelu          |
| Actor dense layers sizes                                                       | (512, 512, 512)|
| Critic dense layers sizes                                                      | (512, 512, 512)|
| Replay buffer size                                                            | max training steps |
| **SVGD**                                                                       |               |
| Number of particles $m$                                                        | 100           |
| Optimization every $k$ agent’s steps                                           | 20            |
| Number of particle moves for one optimization $k^{(p)}$                       | 1             |
| Bandwidth $\sigma$ for RBF kernel $k(.,.)$ for RKHS                           | 1             |
| Learning rate $\epsilon$                                                      | 1e-3          |
| **Agent’s skill model** $D_\phi$                                              |               |
| Dense layers sizes                                                             | (64, 64)      |
| K gradient steps per optimization                                              | 10            |
| Learning rate $\epsilon$                                                      | 1e-3          |
| Training batch size $l^{(m)}$                                                  | 100           |
| Training history length (trajectories)                                        | 1000          |
| Optimize every (steps)                                                        | 4000          |
| Number of training steps $k^{(m)}$                                             | 100           |
| Activations                                                                   | Gelu          |
| **OCSVM prior**                                                               |               |
| Bandwidth $\sigma$ for RBF kernel $k(.,.)$ for RKHS                           | 1             |
| Optimize every (agent’s steps)                                                | 4000          |
| **MEGA**                                                                      |               |
| Bandwidth of KDE RBF kernel                                                    | 0.1           |
| KDE optimize every (agent’s steps)                                            | 1             |
| Number of state samples for KDE optimization                                  | 10000         |
| Number of candidate sampled goals $N$                                          | 100           |
| $Q$-value cutoff $c$                                                           | -3            |
| **Goal-GAN**                                                                  |               |
| Generator input dimension (gaussian noise)                                    | 4             |
| Generator dense layers sizes                                                  | (64, 64)      |
| Discriminator dense layers sizes                                              | (64, 64)      |
| Optimize every (agent’s steps)                                                | 2000          |
| GAN training batch size $l^{(g)}$                                             | 200           |
| Number of GAN steps in for one optimization $k^{(g)}$                         | 100           |
| Discriminator/Generator Learning rate                                          | 1e-3          |
| $P_{min}$ minimum probability for GOID                                         | 0.1           |
| $P_{max}$ minimum probability for GOID                                         | 0.9           |
C.5 Ressources

Every seed of each experiment was run on 1 GPU in the following list \{Nvidia RTX A6000, Nvidia RTX A5000, Nvidia TITAN RTX, Nvidia Titan Xp, Nvidia TITAN V\}. We estimate the amount of training time to 4000 hours, most of it executed in parallel, as we had full time access to 12 GPUs.