Generative Exploration and Exploitation

Jiechuan Jiang [1] **Zongqing Lu** [1]

Abstract

Sparse reward is one of the biggest challenges in reinforcement learning (RL). In this paper, we propose a novel method called *Generative Exploration and Exploitation* (GENE) to overcome sparse reward. GENE dynamically changes the start state of agent to the generated novel state to encourage the agent to explore the environment or to the generated rewarding state to boost the agent to exploit the received reward signal. GENE relies on no prior knowledge about the environment and can be combined with any RL algorithm, no matter on-policy or off-policy, single-agent or multi-agent. Empirically, we demonstrate that GENE significantly outperforms existing methods in four challenging tasks with only binary rewards indicating whether or not the task is completed, including Maze, Goal Ant, Pushing, and Cooperative Navigation. The ablation studies verify that GENE can adaptively tradeoff between exploration and exploitation as the learning progresses by automatically adjusting the proportion between generated novel states and rewarding states, which is the key for GENE to solving these challenging tasks effectively and efficiently.

1. Introduction

Deep reinforcement learning (RL) has achieved great success in many sequential decision-making problems, such as Atari games (Mnih et al., 2015), Go (Silver et al., 2016; 2017), and robotic tasks (Levine et al., 2016; Duan et al., 2016). However, a common challenge in many real-world applications is the reward is extremely sparse or only binary. For example, in goal-based tasks, the agent can only receive the reward when it reaches the goal. However, the goal is usually hard to reach via random exploration, such as ϵ-greedy and Gaussian noise. Domain-specific knowledge can be used to construct a shaped reward function to guide the policy optimization. However, it often biases the policy in a suboptimal direction, and more importantly domain-specific knowledge is unavailable in many cases.

Some methods have been proposed to address sparse reward in RL. Nair et al. (2018) exploited imitation learning to enable the agent to learn a policy from expert behaviors. However collecting expert demonstrations is usually a challenge. HER (Andrychowicz et al., 2017) enables the agent to learn from undesired outcomes by replaying each episode with a different goal than the one the agent was originally trying to achieve. However, by random exploration the agent rarely obtains a real reward signal. Exploration is crucial for agents to learn in environments with sparse rewards. Count-based exploration (Bellemare et al., 2016; Ostrovski et al., 2017; Tang et al., 2017) and curiosity-driven exploration (Pathak et al., 2017; Burda et al., 2018a;b) quantify the novelty of the state and take it as an intrinsic reward to boost the agent to explore new states. However, intrinsic reward leads to deviation from the true target and causes the learning process detoured and unstable.

When we humans learn to solve a task, we never always start from the very beginning, but stand up from where we fall down and move forward. As we have already known how to reach these states, it is unnecessary to always start from the scratch. Moreover, we deliberately practice more on some unfamiliar and instructive states. Analogically, appropriate start states would also accelerate the learning of agent, which has been proven theoretically by (Kearns et al., 2002) and practically by (Popov et al., 2017) using start states collected from expert demonstrations. We go one step further following these intuitions.

In this paper, we propose a novel method called *Generative Exploration and Exploitation* (GENE) to overcome sparse reward. GENE dynamically changes the start state of agent to the generated novel state to encourage the agent to explore the environment or to the generated rewarding state (where it is easy for the agent to reach the goal under current policy) to boost the agent to exploit received reward signals. We adopt Variational Autoencoder (VAE) (Kingma & Welling, 2013) as the generative model to produce desired states and let the agent play from these states rather than the initial state. As the encoder of VAE compresses high dimensional states into a low dimensional latent space, it is easy to estimate the distribution of the states experienced by the agent via

[1]Department of Computer Science, Peking University. Correspondence to: Zongqing Lu <zongqing.lu@pku.edu.cn>.

Kernel Density Estimation (KDE) (Rosenblatt, 1956). Then, we sample from this distribution to feed into the decoder to reconstruct states. By deliberately giving high probability to the representations of states with low density, GENE is able to automatically generates novel states and rewarding states and adaptively adjust their proportion to guide the policy optimization as the learning progresses.

GENE can be combined with any RL algorithm, no matter on-policy or off-policy, single-agent or multi-agent. Driven by unsupervised VAE and statistical KDE, GENE relies on no prior knowledge about the environment. Taking advantage of embedding states into a latent space, GENE is practical and efficient in high dimensional environments. Moreover, in multi-agent environments with sparse rewards, where the search space exponentially increases with the number of agents, GENE can greatly help agents co-explore the environment. Empirically, we evaluate GENE in four tasks with binary rewards, including Maze, Goal Ant, Pushing, and Cooperative Navigation. We demonstrate GENE significantly outperforms existing methods in all the four tasks. The ablation studies verify the effectiveness of generative exploration and exploitation. Moreover, GENE can adaptively tradeoff between exploration and exploitation by automatically adjusting the proportion between generated novel states and rewarding states, which is the key to solving these challenging tasks effectively and efficiently.

2. Related Work

Exploration The main idea behind exploration methods is to impel the agent to discover novel states by intrinsic motivation which explains the need to explore the environment. Most exploration methods fall into two categories: count-based methods and curiosity-driven methods. Count-based methods directly use or estimate visit counts as an intrinsic reward to guide the agent towards reducing uncertainty, such as CTS-based pseudocounts (Bellemare et al., 2016), PixelCNN-based pseudocounts (Ostrovski et al., 2017) and hash-based counts (Tang et al., 2017). Curiosity-driven methods (Pathak et al., 2017; Burda et al., 2018a;b) use the prediction error in the learned feature space as the intrinsic reward. When facing unfamiliar states, the prediction error becomes high and the agent will receive high intrinsic reward. The error in feature space rather than raw observation space helps the agent to focus on the novelty in information relevant to predicting the action of agent rather than unpredictable and uncontrollable noise. However, the shaped reward is biased and the scale of the intrinsic reward might vary greatly at different timesteps, which leads to deviation from the true target and causes the learning process detoured and unstable.

Generative Methods in RL Some works try to combine generative model and RL. Huang et al. (2017) proposed an approach for pre-training the agent based on the enhanced GAN to shorten the training phase. Collecting a small set of data samples using random policy, the GAN learns to produce experiences for the agent. Goal GAN (Florensa et al., 2018) generates different tasks at the appropriate level of difficulty for the agent. First, a set of goals are labeled based on whether they are at the appropriate level for the current policy, and then the label is added into the loss function of GAN to produce the tasks not too easy and not too hard. However, in the environments with sparse rewards, the probability of reaching a goal might be very low, and thus the reward signal might be too weak to drive the training of GAN, especially in high dimensional environments. Reverse curriculum generation (Florensa et al., 2017) makes the agent gradually learn to reach the goal from a set of start states which are between the bounds on the probability of success. However, it requires start states uniformly distributed over all feasible states, which relies on prior knowledge about the environment. Goyal et al. (2018) trained a backtracking model to predict the preceding states that terminate at the given high-reward state. Then the generated traces are used to improving the policy via imitation learning.

Experience Replay The methods of experience replay focus on which experiences to store and how to use them to speed up the training. Prioritized experience replay (Schaul et al., 2015) measures the priority of experiences by the magnitude of their temporal-difference errors and replays transitions with high priority more frequently. Andrychowicz et al. (2017) proposed hindsight experience replay (HER) for goal-based tasks. HER is inspired by that one can learn almost as much from achieving an undesired outcome as from the desired one. After experiencing some episodes, HER arbitrarily selects a set of additional goals and uses them to replace the original goals of the transitions in the reply buffer. However, learning additional goals slows down the learning process and by random exploration the agent rarely sees a real reward signal. Moreover, HER only works on off-policy methods.

3. Background

3.1. Deep Reinforcement Learning

Consider a scenario where an agent lives in an environment. At every timestep t, the agent gets current state s_t of the environment, takes an action a_t to interact with the environment, receives a reward r_t from the environment, and the environment transitions to the next state. Deep RL tries to help the agent learn a policy which maximizes the expected return $R = \sum_{t=0}^{T} \gamma^t r_t$. The policy can be deterministic $a_t = \mu(s_t)$ or stochastic $a_t \sim \pi(\cdot|s_t)$.

There are two main approaches in RL: policy gradient

and Q-learning. Policy gradient methods directly adjust the parameters θ by maximizing the approximation of $J(\pi_\theta)$, *e.g.*, $J(\theta) = \mathbb{E}_{s\sim p^\pi, a\sim\pi_\theta}[R]$. They are almost always on-policy, which means that the parameters are only updated using data collected while acting according to the current policy. TRPO (Schulman et al., 2015) and PPO (Schulman et al., 2017) are typical policy gradient methods. They all maximize a surrogate objective function which estimates how much $J(\pi_\theta)$ will change as a result of the update. The advantage function, defined as $A_t = Q_t - V_t$, explains how much better the selected action is than average. TRPO maximizes $J(\theta) = \mathbb{E}_t[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} A_t]$, while PPO maximizes $J(\theta) = \mathbb{E}_t[\min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} A_t, \text{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}, 1-\epsilon, 1+\epsilon\right) A_t\right]$. Q-learning (*e.g.*, DQN) learns a value function $Q_\theta(s,a)$ and the action is selected by $a(s) = \arg\max_a Q^\pi(s,a)$. The function is updated based on Bellman equation $Q^\pi(s,a) = \mathbb{E}_{s'}[r(s,a) + \gamma \mathbb{E}_{a'\sim\pi}[Q_\pi(s',a')]]$. Q-learning methods are usually off-policy, which means that each update can use the past experiences.

DDPG (Lillicrap et al., 2015) learns a Q-function and a deterministic policy, where the Q-function provides the gradient to update the policy. MADDPG (Lowe et al., 2017) is an extension of DDPG for multi-agent environments, where each agent has an independent network, and the Q-function of the agent takes as input the actions and observations of all the agents and provides the gradient to its policy. MADDPG makes it feasible to train multiple agents acting in a globally coordinated way.

3.2. Variational Autoencoder

Variational Autoencoder (VAE) consists of an encoder and a decoder. The encoder takes a high dimensional datapoint x as the input and outputs parameters to $q_\theta(z|x)$. A constraint on the encoder forces the latent space roughly follow a unit Gaussian distribution. The decoder learns to reconstruct the datapoint x given the representation z, denoted by $p_\phi(x|z)$. VAE maximizes $\mathbb{E}_{z\sim q_\theta(z|x)}[\log p_\phi(x|z)] - \text{KL}(q_\theta(z|x)||p(z))$, where $p(z)$ is the unit Gaussian distribution. The first term is the reconstruction likelihood, which encourages the decoder to learn to reconstruct x. The second term is the Kullback-Leibler divergence that ensures $q_\theta(z|x)$ is similar to the prior distribution $p(z)$. This has the effect of keeping the representations of similar datapoints close together rather than separated in different regions of the latent space.

3.3. Kernel Density Estimation

Kernel density estimation (KDE) belongs to the class of non-parametric density estimations. Closely related to histograms, but KDE smooths out the contribution of each observed datapoint x_i over a local neighborhood of that datapoint by centering a kernel function. Formally, KDE can be formulated as

$$\hat{f}_h(x) = \frac{1}{nh} \sum_{i=1}^{n} K(\frac{x - x_i}{h}),$$

where K is the kernel function, and $h > 0$ is the bandwidth that controls the amount of smoothness. Due to the convenient mathematical properties, the Gaussian kernel is often used. The choice of bandwidth is a trade-off between the bias of estimator and its variance.

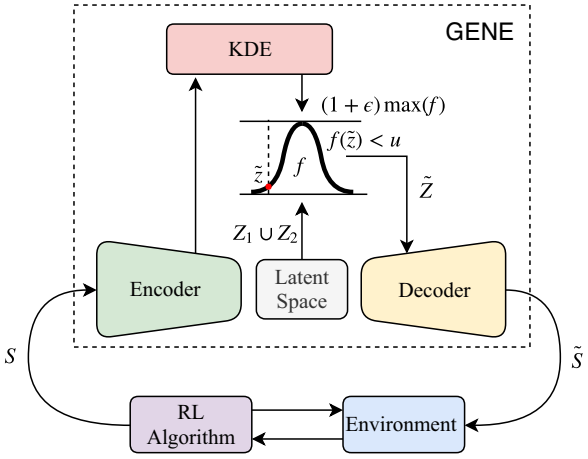


Figure 1. GENE consists a VAE and a KDE.

4. Methods

GENE consists of a VAE and a KDE and works with any RL algorithm, as illustrated in Figure 1. In training, we maintain a buffer $\mathcal{R}_1$ to store the states the agent has recently experienced and another buffer $\mathcal{R}_2$ to store the states where it is easy for the agent to reach the goal under current policy, which we call *rewarding states*[1]. As rewarding states can only be identified after the agent reaches the goal, $\mathcal{R}_2$ is usually empty at the beginning of training. We abuse the notation a little and let $\mathcal{R}_1$ and $\mathcal{R}_2$ also denote the set of states they contain, respectively.

The basic idea of GENE is to generate states with low density in the distribution of $\mathcal{R}_1$. Low density means the generated states are novel (*i.e.*, the agent is not familiar with), and starting from these states the agent is able to explore the environment further. When novel states become common (*i.e.*, higher density than before), new novel states will be generated. Therefore, GENE propels the agent to explore the environment gradually. The aim of exploration is to

[1]When the agent reaches the goal, we can trace back to find the states that lead the agent to the goal within certain timesteps $[\text{step}_{\min}, \text{step}_{\max}]$ under current policy. These state are defined as *rewarding states*.

obtain reward signals. When there are rewarding states in $\mathcal{R}_2$ (*i.e.*, the agent has reached the goal at least once), GENE will generate states according to the rewarding states and let the agent also start from these generated states. This will quickly strengthen the learned policy. Under the improved policy, the agent may spend less timesteps than before to reach the goal. Therefore, some states where the agent used to be far (difficult) from the goal are qualified as rewarding states and added into $\mathcal{R}_2$, which eventually makes the agent learn the policy that can reach the goal from the initial state. In short, GENE guides the agent to explore the environment by starting from the states with low density and reinforces learned policy by replaying rewarding states.

4.1. State Generation

In order to purposely generate states that the agent is not familiar with, it is necessary to estimate the distribution of the states the agent has recently experienced. However, the density estimation of high dimensional states is usually intractable. Fortunately, the encoder of VAE maps the high dimensional state to the latent space which is described as k-dimension mean and log-variance $(\mu, \log\sigma)$. We use μ as the representation of the input state. As the latent space is only k-dimension and roughly follows the unit Gaussian distribution, it is easy to estimate the probability density function (PDF) of the representations of the states in $\mathcal{R}_1$ using KDE, denoted by $f(z)$.

To guide the agent to explore the novel states and receive reward signals, the generated states should have two properties: it is difficult to reach the generated states from the initial state via random exploration; it is feasible to reach the goal from these states under current policy. To do so, we uniformly sample from the latent space to get a set of representations Z_1. Then if $|\mathcal{R}_2| \neq 0$, we also uniformly sample a state from $\mathcal{R}_2$, encode the state, sample a representation on $\mathcal{N}(\mu, \sigma^2)$ where $(\mu, \log\sigma)$ is the encoding of the state, and put the representation in Z_2. We repeat the process until $|Z_2| = |Z_1|$.

We apply rejection sampling to select eligible representations from $Z_1 \cup Z_2$. The principle is to give a high probability to the representation with low density in $f(z)$. We propose a uniform distribution with the PDF $(1+\epsilon) * \max(f(z))$. Every time we take out a representation $\tilde{z}$ from $Z_1 \cup Z_2$, we sample u from $\mathrm{Unif}(0, (1+\epsilon) * \max(f(z)))$. Then we check whether $f(\tilde{z}) < u$. If this holds, we accept $\tilde{z}$, otherwise we reject the representation, as illustrated Figure 1. We repeat the sampling process until the number of accepted samples $\tilde{Z}$ is equal to N_s. Then, we pass $\tilde{Z}$ to the decoder to reconstruct the states $\tilde{S}$, from which the agent will start new episodes.

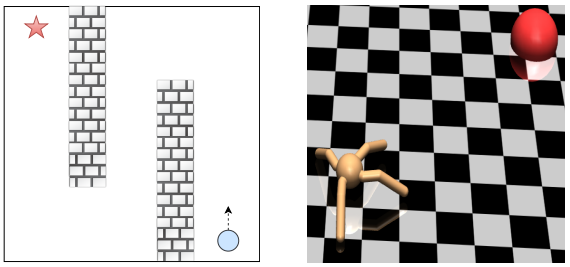
Let $\tilde{Z}_1$ and $\tilde{Z}_2$ denote $Z_1 \cap \tilde{Z}$ and $Z_2 \cap \tilde{Z}$, respectively. The generated states from $\tilde{Z}_1$ make the agent to explore novel states, while the generated states from $\tilde{Z}_2$ make the agent more feasible to reach the goal and strengthen the learned policy. Note that the states generated by VAE from a same state (*e.g.*, a state in $\mathcal{R}_2$) will be slightly different. This difference prevents the agent from always repeating the states it has experienced. It also makes GENE more sample-efficient since it can generate various samples even using only limited reference samples. As the policy updates, the distribution of experienced states also changes, which brings two benefits. On the one hand, novel states become common gradually, which boosts the agent to explore new novel states continuously. On the other hand, the varying distribution makes the proportion between $\tilde{Z}_1$ and $\tilde{Z}_2$ change adaptively without any prior knowledge, which makes GENE automatically tradeoff between exploration and exploitation to guide the policy optimization.

Algorithm 1 Generative Exploration and Exploitation

```
Initialize RL model (e.g., PPO, TRPO, DQN, DDPG)
Initialize buffer R_1 and R_2
for episode = 1, ..., M do
    Agent starts from the generated state or from the initial state with the probability 50%, respectively
    Store all visited states in R_1
    Store rewarding states in R_2 if any
    if episode%N_s = 0 then
        Initialize KDE and VAE
        Train VAE using R_1
        Fit the PDF of R_1 using the representations of R_1 via KDE
        Sampling from latent space and R_2 to obtain Z_1 and Z_2
        Applying rejection sampling to select Z̃ from Z_1 and Z_2
        Reconstruct states S̃ from Z̃ for next N_s episodes
    end if
    Update RL model
end for
```

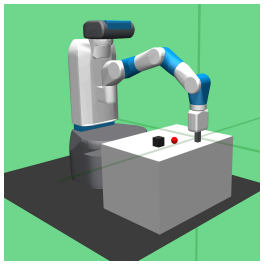
4.2. Training

Algorithm 1 details the training of GENE. Every episode, the agent starts from the generated state or initial state randomly. The probability is set to 50% to balance the learning of task itself (from initial state) and the learning from generated states. Every N_s episodes, we train the VAE using the states stored in $\mathcal{R}_1$ from the scratch. The reason why training from the scratch is to avoid overfitting and collapse when the distribution of experienced states changes slowly. The training of VAE is very easy and fast, which is also an advantage of VAE comparing against GAN. The PDF of the experienced states is estimated and fitted by their representations via KDE. Then, $\tilde{Z}$ is obtained by applying rejection

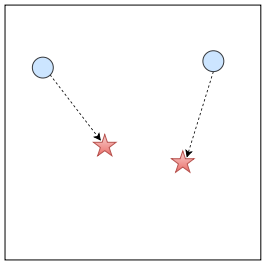


(a) Maze

(b) Goal Ant



(c) Pushing



(d) Cooperative Navigation

Figure 2. Illustrations of experimental tasks with binary rewards.

sampling to Z_1 and Z_2, and the states are generated by the decoder for the next N_s episodes. After that, the RL model is updated accordingly. GENE is very easy to implement. Moreover, as GENE does not directly interact with the RL algorithm, it is compatible with any RL algorithm.

5. Experiments

In this section, we focus on answering the following questions. (*i*) Can the mechanism and effectiveness of GENE be verified and interpreted empirically? (*ii*) Is GENE effective in high dimensional environments? (*iii*) Does GENE help in multi-agent environments?

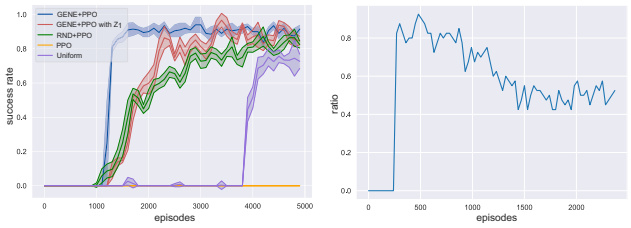
To answer these questions, we investigate GENE in four tasks with binary rewards indicating whether or not the task is completed. We compare GENE against existing methods for sparse/binary rewards. Both GENE and baselines work on a base RL algorithm. The parameters of the base RL algorithm are the same, which guarantees the comparison fairness. Note that in the four tasks, we may use different base RL algorithms. The main reason is that some baselines only work on certain base RL algorithms. Also, by this chance we can verify that GENE is compatible with any RL algorithm. To answer the first question, we demonstrate GENE in a challenging Maze task (Figure 2a) and compare it against RND (Burda et al., 2018b). To answer the second question, we study GENE in two robotic locomotion tasks, Goal Ant (Figure 2b) and Pushing (Figure 2c), and compare it against Goal GAN (Florensa et al., 2018) and HER (Andrychowicz et al., 2017), respectively. To answer the last question, we demonstrate GENE in Cooperative Navigation (Figure 2d) and compare it against MADDPG (Lowe et al., 2017). The details of each task and the hyperparameters of the algorithms used in the experiments are available in Appendix. The code of GENE is available in Supplementary.

5.1. Maze

In the 2D maze, the agent learns to navigate from an initial position (lower right corner) to the target position (upper left corner) within a given number of timesteps as depicted in Figure 2a. At every timestep, the agent can observe its position, speed and the target position, totally 6-dimension, and then choose to accelerate in one of four directions. Only if the agent reaches the target, it receives a reward. This scenario is built on Multi-Agent Particle Environment (Lowe et al., 2017). In Maze, we choose PPO (Schulman et al., 2017) as the base RL algorithm.

Figure 3a shows the learning curves of GENE and baselines. The base algorithm PPO is incapable of solving this task. The reason is that the agent never reaches the goal with random exploration and thus cannot learn an effective policy. GENE with PPO reaches 90% success rate by only 1500 episodes and greatly outperforms others.

Figure 4 gives more details of the learning process and explains the mechanism of GENE. At the beginning, $\mathcal{R}_2$ is empty, all the generated states come from $\mathcal{R}_1$. The agent is wandering around the start position, so the generated states are mostly distributed at the edge of the scope of activity. As the training progresses, the agent becomes familiar with the states which are originally novel and the scope of activity of agent gradually expands. Subsequently, the agent can reach the target occasionally, then rewarding states are generated from $\mathcal{R}_2$, where the agent is around the target. As the representations of $\mathcal{R}_2$ have low density as illustrated in Figure 4, they are more likely to be selected and thus much more rewarding states are generated. This is verified by Figure 3b, where the proportion of $\tilde{Z}_2$ to $\tilde{Z}_1 \cup \tilde{Z}_2$ quickly increases to more than 80%. Then, the proportion gradually decreases as the agent becomes familiar with these rewarding states. Moreover, in the newly generated states from $\mathcal{R}_2$ the distance between the agent and target increases. This is because the early states in $\mathcal{R}_2$ are easy for the agent now. That is, as the policy becomes better, the agent can reach the target in these states using less timesteps than before, and hence these states are not qualified to be taken into $\mathcal{R}_2$ any more. Therefore, more difficult states are generated. The learned policy is continuously optimized by the generated states with gradually increased difficulty. In the later phase, $\tilde{Z}_2$ becomes common to the agent, and the proportion goes down to 50% around 1500 episodes, which coincides with the learning curve of GENE reaching the plateau. From Figure 3a and 3b, we can see that *GENE automatically adjusts the proportion between $\tilde{Z}_1$ and $\tilde{Z}_2$ to*



(a) learning curves (b) $|\tilde{Z}_2|/|\tilde{Z}_1 \cup \tilde{Z}_2|$

Figure 3. Learing curves in terms of success rate of GENE and baselines (a) and the proportion of $\tilde{Z}_2$ to $\tilde{Z}_1 \cup \tilde{Z}_2$ as the training progresses (b) in Maze. In the experiments, all learning curves are plotted using three training runs with different random seeds, and the shade region is $[\mu - \sigma,\ \mu + \sigma]$.

adaptively tradeoff between exploration and exploitation as the learning progresses, which guides the agent to learning a good policy.

To verify the effectiveness of GENE, we compare it against RND (Burda et al., 2018b), a curiosity-driven exploration method using the prediction errors of network trained on the past experiences of the agent to quantify the novelty as an intrinsic reward. In GENE, the generated novel states encourage the agent to explore. We want to see how well GENE with only Z_1 (*i.e.*, GENE with only generative exploration) would work. We make $\mathcal{R}_2$ always empty and generate states only from Z_1. That is, the generated states mostly have low density in the distribution of the experienced states $\mathcal{R}_1$. As illustrated in Figure 3a, GENE with only Z_1 outperforms RND. Together with generative exploitation, GENE significantly outperforms RND. The difference between GENE and GENE with Z_1 verifies that replaying generated rewarding states truly accelerates the learning. In RND, the intrinsic reward varies largely at different timesteps, causing the learning process detoured.

Since GENE changes the start position of the agent to the generated state during training, we also compare GENE against a simple method that lets the agent start from a uniformly random location in the maze. Note that this simple method is a dense reward case (Pathak et al., 2017) and it requires prior knowledge about the map of the maze. We evaluate its success rate also from the initial position. As show in Figure 3a, the agent with uniform distribution eventually learns to solve this task, but converges much slowly. This is because too many start positions merely benefit the learning, especially those are both difficult to reach and far from the target. Even worse, from these positions the agent might learn the unreasonable policy, such as always moving in the same direction, which impacts on further learning. GENE allows the agent to focus on instructive states and adaptively adjusts them to make the agent learn gradually to solve the task.

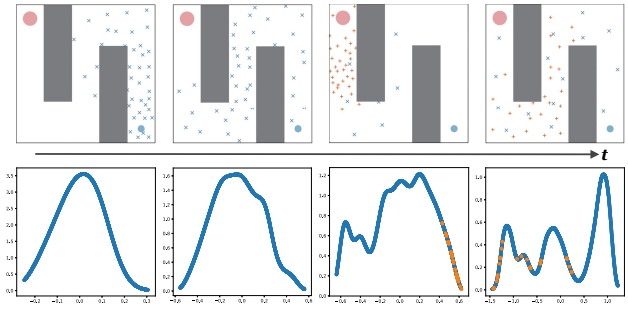


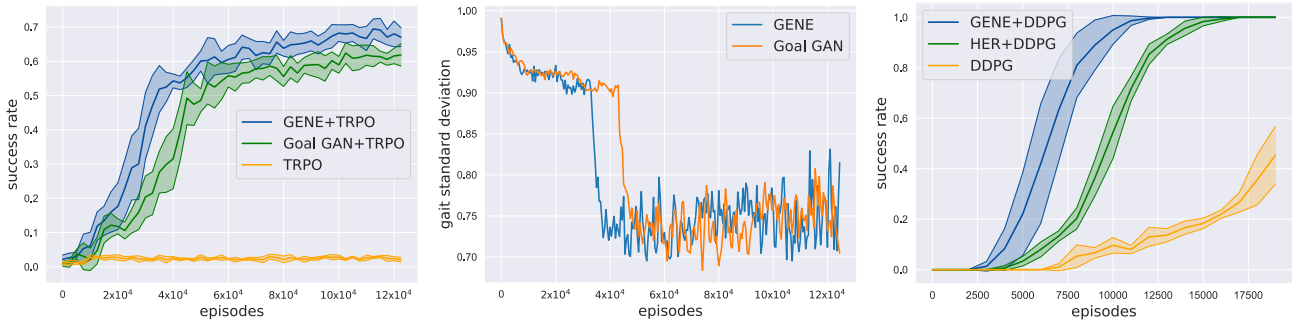
Figure 4. Top row shows the distribution of the positions of agent in generated states as the training progresses, where the blue crosses are the agent positions of $\tilde{Z}_1$ and the orange pluses are the agent positions of $\tilde{Z}_2$. As the goal position in generated states varies very slightly, they are not plotted for clear presentation. Bottom row shows the corresponding PDF $f(z)$ estimated via KDE, where the orange dots are $\tilde{Z}_2$.

5.2. High Dimensional Environments

To verify GENE is also effective and efficient in high dimensional environments, we demonstrate GENE in Goal Ant and Pushing, two robotic tasks in Mujoco (Todorov et al., 2012).

Goal Ant The ant (Duan et al., 2016) within a square and positioned at the center of the square tries to reach the goal with a random location with a given number of timesteps, as illustrated in Figure 2b. Only when the ant gets the goal, it receives a reward. The state space is 41-dimension, including the positions of the ant and goal, and the positions and velocities of joints of the ant. The action space of the ant is 8-dimension, controlling the movement. In Goal Ant, we choose TRPO (Schulman et al., 2015) as the base RL algorithm.

Figure 5a shows the learning curves of GENE and baselines. TRPO is unable to solve the task, and GENE outperforms Goal GAN (Florensa et al., 2018). Goal GAN generates the positions of the goal from easy to difficult to guide the ant to move towards the goal, while GENE generates the positions of both the ant and goal which together drive the learning more quickly. The difference between GENE and Goal GAN is well illustrated in Figure 6. The positions of generated goals by Goal GAN and GENE diffuse from the origin (the center of the square) over time. The learning of Goal GAN agent becomes more difficult as the goal position diffuses and the ant position remains fixed. However, GENE also generates the start position of the ant. In a generated state, the distance between the goal and ant is mostly shorter than the distance between the goal and origin, as illustrated in Figure 6. This means, for a same goal, GENE agent learns to reach the goal faster than Goal GAN agent who has to start from the origin. That is the reason why the generated goal positions of GENE are farther from the origin than Goal



(a) learning curves in Goal Ant (b) standard deviation of gait in Goal Ant (c) learning curves in Pushing

Figure 5. In high dimensional environments: learning curves in terms of success rate of GENE and baselines (a) and the standard deviation of gait as the training processes (b) in Goal Ant, and learning curves in terms of success rate of GENE and baselines (c) in Pusing.

GAN at the same training episode and GENE outperforms Goal GAN.

In addition to the positions of ant and goal, GENE also generates the positions and velocities of the joints of the ant. The control of multi-joint robot is complex due to the high degrees of freedom, and issues such as gimbal lock might happen. The success of GENE explains the generativity in high dimensional state space, which is attributed to that VAE could map the high dimensional state to a meaningful latent space. This also helps the learning. To reach the goal, the ant must learn how to crawl first. GENE generates adequate postures for the ant to explore how to crawl, so the ant learns

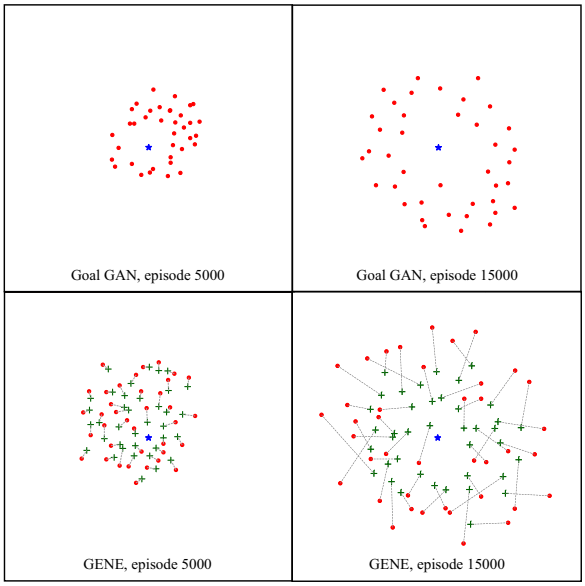


Figure 6. Top row shows the distribution of goal positions generated by Goal GAN at episode 5000 and 15000, where the red dot is the generated goal position and the blue star is the initial position of agent. Bottom row is the distribution of goal and agent positions generated by GENE also at episode 5000 and 15000, where the pair of connected goal and ant (green plus) indicates their positions in a generated state.

Table 1. Proportion in training time in Goal Ant

	Interaction	Training VAE	Training TRPO
proportion	78%	9%	13%

to crawl more quickly than Goal GAN as illustrated by the curve of the standard deviation of the ant gait in Figure 5b. When the ant masters how to crawl, the gait is more steady and hence the standard deviation decreases. Goal GAN only focuses on the position of the goal, ignoring the gait of the ant, and thus learning to crawl is also slower.

The characteristics of VAE perfectly match GENE. Firstly, the training of VAE is easier and more stable than GAN, which is a common view. Table 1 gives the training time proportion of each component in Goal Ant, where we can see training VAE only takes 9%. Second, one of the challenges in training GAN is mode collapse, meaning the generator only produces a small family of similar samples. However, the samples generated by VAE are various. Although the generated samples are a little noisy, which is usually a disadvantage for computer vision applications, they are particularly useful for the RL agent because noisy states make the policy more robust.

Pushing Another challenging task is Pushing, where the robot learns to move the box to the target location on the table within a given number of timesteps, as illustrated in Figure 2c. The robot fingers are locked to prevent grasping. The learned behavior is a mixture of pushing and rolling. Only when the box is pushed at the target location, the agent receives a reward. In Pushing, we choose DDPG (Lillicrap et al., 2015) as the base RL algorithm.

Figure 5c shows the learning curves of GENE and baselines, where GENE converges more quickly than HER (Andrychowicz et al., 2017) and DDPG converges much slower. In this task, the robotic arm first moves to the box and then push the box to the target location. When the robotic arm encouraged by the exploration of GENE pushes

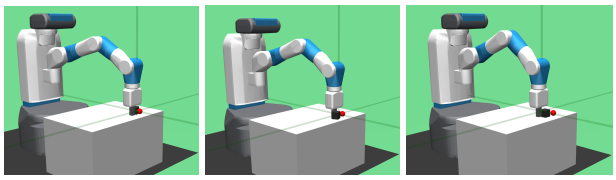
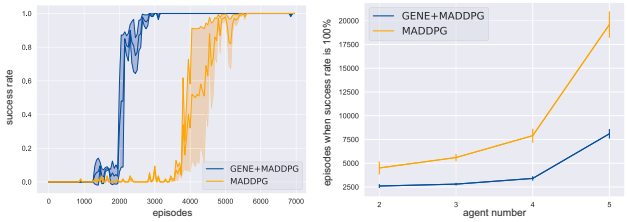


Figure 7. Three generated states based on a rewarding state in Pushing.



(a) learning curves (b) episodes to solve the task

Figure 8. Learning curves in terms of success rate of GENE and MADDPG (a) and the number of training episodes to obtain 100% success rate with different number of agents (b) in Cooperative Navigation.

the box once, the movement of the box will produce some novel states the agent has never seen. Then, GENE will generate similar novel states which will drive the robotic arm to continuously push the box around, which will eventually lead to the box being pushed at the target location. After that, rewarding sates are generated, the robotic arm gradually reinforces the learned policy. However, HER agent has to learn many additional goals and rarely sees a real reward signal, which slows down the learning processing. One of the most important properties of VAE is that the distance in the learned latent space is meaningful and similar states gather together in the latent space. Figure 7 shows three generated states based on a rewarding state. They are all similar but with subtle differences. The generativity of GENE makes it more sample-efficient in high dimensional environments.

5.3. Cooperative Navigation

In multi-agent environments, many tasks rely on collaboration among agents. However, the agent does not know the policies of others and their policies are always changing during training, and thus the task is much more difficult than the single-agent version. In this Cooperative Navigation task, there are a same number of landmarks and agents. The goal of agents is to occupy each landmark within a given number of timesteps, as illustrated in Figure 2d. Only when every landmark is occupied by an agent, each agent receives a reward. Therefore, this is a case of binary reward for the multi-agent environment. We choose MADDPG (Lowe et al., 2017) as the base multi-agent RL algorithm, where each agent has an independent actor-critic network, without weight sharing or communication.

Figure 8a illustrates the learning curves of GENE and MADDPG in the setting of two agents and two landmarks, where GENE converges much faster than MADDPG. In MADDPG, the critic network of each agent takes as input the actions and states of all the agents to estimate the Q-value. As the critic is centralized, it can guide the learned policy towards collaboration in this task. Therefore, MADDPG agents can obtain the reward signal sometimes, which eventually leads to the convergence. However, the exploration of MADDPG agent is independent, not coordinated. Unlike MADDPG, GENE uses the experienced states of all the agents to generate novel states and impels the agents to co-explore the environment from these states. Note that GENE has the same amount of information as MADDPG. The coordinated exploration among agents helps the agents to reach the goal much easier than MADDPG. When the agents receive the reward once, GENE can generate various rewarding states that cover many situations where the agents need to cooperate to obtain the reward. However, it takes much more timesteps for MADDPG agents to experience these states. These two aspects make GENE greatly outperform MADDPG.

When the number of agents increases, the search space increases exponentially and thus the task becomes much more difficult. As shown in Figure 8b, the number of training episodes to solve the task for MADDPG increases dramatically with the increase of the number of agents and landmarks. When more agents need to cooperate to complete the task, independent exploration becomes much less efficient. However, GENE still performs much better than MADDPG. The gain of GENE over MADDPG even increases with the number of agents (*i.e.*, from $2\times$ to $3\times$ speedup). This indicates GENE indeed accelerates the learning of multiple agents in the cooperative task regardless the number of agents.

6. Conclusion

In this paper, we have proposed GENE for overcoming sparse rewards in RL. By dynamically changing the start state of agent to the generated novel state or to the generated rewarding state, GENE can automatically tradeoff between exploration and exploitation to optimize the policy as the learning progresses. Driven by unsupervised VAE and statistical KDE, GENE relies on no prior knowledge about the environment and can be combined with any RL algorithm, no matter on-policy or off-policy, single-agent or multi-agent. Empirically, we demonstrate that GENE significantly outperforms existing methods in a variety of challenging tasks with binary rewards.

References

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, O. P., and Zaremba, W. Hindsight experience replay. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 5048–5058, 2017.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1471–1479, 2016.

Burda, Y., Edwards, H., Pathak, D., Storkey, A., Darrell, T., and Efros, A. A. Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355*, 2018a.

Burda, Y., Edwards, H., Storkey, A., and Klimov, O. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018b.

Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning (ICML)*, pp. 1329–1338, 2016.

Florensa, C., Held, D., Wulfmeier, M., Zhang, M., and Abbeel, P. Reverse curriculum generation for reinforcement learning. *arXiv preprint arXiv:1707.05300*, 2017.

Florensa, C., Held, D., Geng, X., and Abbeel, P. Automatic goal generation for reinforcement learning agents. In *International Conference on Machine Learning (ICML)*, pp. 1514–1523, 2018.

Goyal, A., Brakel, P., Fedus, W., Lillicrap, T., Levine, S., Larochelle, H., and Bengio, Y. Recall traces: Backtracking models for efficient reinforcement learning. *arXiv preprint arXiv:1804.00379*, 2018.

Huang, V., Ley, T., Vlachou-Konchylaki, M., and Hu, W. Enhanced experience replay generation for efficient reinforcement learning. *arXiv preprint arXiv:1705.08245*, 2017.

Kearns, M., Mansour, Y., and Ng, A. Y. A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine learning*, 49(2-3):193–208, 2002.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, O. P., and Mordatch, I. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 6379–6390, 2017.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529, 2015.

Nair, A., McGrew, B., Andrychowicz, M., Zaremba, W., and Abbeel, P. Overcoming exploration in reinforcement learning with demonstrations. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6292–6299, 2018.

Ostrovski, G., Bellemare, M. G., Oord, A., and Munos, R. Count-based exploration with neural density models. In *International Conference on Machine Learning (ICML)*, pp. 2721–2730, 2017.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning (ICML)*, pp. 2778–2787, 2017.

Popov, I., Heess, N., Lillicrap, T., Hafner, R., Barth-Maron, G., Vecerik, M., Lampe, T., Tassa, Y., Erez, T., and Riedmiller, M. Data-efficient deep reinforcement learning for dexterous manipulation. *arXiv preprint arXiv:1704.03073*, 2017.

Rosenblatt, M. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, pp. 832–837, 1956.

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

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In *International Conference on Machine Learning (ICML)*, pp. 1889–1897, 2015.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I.,

Panneershelvam, V., Lanctot, M., et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

Tang, H., Houthooft, R., Foote, D., Stooke, A., Chen, O. X., Duan, Y., Schulman, J., DeTurck, F., and Abbeel, P. # exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 2753–2762, 2017.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5026–5033, 2012.

A. Hyperparameters

GENE, RND (Burda et al., 2018b), Goal GAN (Florensa et al., 2018), and HER (Andrychowicz et al., 2017) all work on a base RL algorithm. In each task, the hyperparameters of the base RL algorithm, such as batch size, learning rate, and discount factor, are all the same for fair comparison, which are summarized in Table 2. The hyperparameters of GENE used in each task are also summarized in Table 2.

For GENE, the size of $\mathcal{R}_1$ should be adequate such that the distribution of $\mathcal{R}_1$ can approximate the state distribution under current policy of the agent. That is, the states in $\mathcal{R}_1$ should be experienced by the agent without substantial policy/Q-function updates. However, the size of $\mathcal{R}_1$ is easy to tune based on the setting of the task (*e.g.*, the maximum timesteps) and the parameters of the base RL algorithm. The size of $\mathcal{R}_2$ can be simply set to a number smaller than the size of $\mathcal{R}_1$. To select rewarding states, $\text{step}_{\min}$ and $\text{step}_{\max}$ are used, but they are easy to tune based the maximum timesteps for the task.

B. Maze

In the 2D maze, the agent learns to navigate from a initial position $P_a = (0.8 \pm 0.2, -0.8 \pm 0.2)$ to the target position $P_t = (-0.8 \pm 0.2, 0.8 \pm 0.2)$ within 200 timesteps. Only when the agent reaches the target ($||P_a - P_t|| < 0.15$), the agent receives a reward 10, otherwise always 0. RND uses a target network with hidden layers $(32, 32)$ to produce the representation with 8-dimension. The predictor network is trained to mimic the target network with the learning rate 10^{-4}. The prediction error is used as the intrinsic reward to guide the agent to explore the novel state.

Table 2. Hyperparameters

Hyperparameter	Maze	Goal Ant	Pushing	Cooperative Navigation
maximum timesteps	200	500	50	60
RL algorithm	PPO	TRPO	DDPG	MADDPG
discount (γ)	0.99	0.99	0.95	0.95
batch size	200	1024	256	1024
# actor MLP units	$(128, 128)$	$(32, 32)$	$(256, 256, 256)$	$(64, 64)$
# critic MLP units	$(128, 128)$	$(32, 32)$	$(256, 256, 256)$	$(64, 64)$
actor leanring rate	3×10^{-4}	3×10^{-3}	10^{-3}	10^{-2}
critic leanring rate	5×10^{-3}	3×10^{-3}	10^{-3}	10^{-2}
MLP activation			ReLU	
optimizer			Adam	
replay buffer size	-	-	10^6	
VAE latent space dimension	1	5	5	3
# VAE encoder MLP units	$(128, 128)$	$([128, 128], 128)$	$(128, 128)$	$(128, 128)$
# VAE decoder MLP units	$(128, 128)$	$(128, [128, 128])$	$(128, 128)$	$(128, 128)$
VAE learning rate			3×10^{-4}	
VAE training epochs			3	
KDE bandwidth			0.05	
KDE kernel			Gaussian	
$\mathcal{R}_1$ size	40000	50000	10000	20000
$\mathcal{R}_2$ size	20000	5000	4000	10000
$\text{step}_{\min}$	40	3	1	6
$\text{step}_{\max}$	100	40	5	30
# generated states (N_s)	20	50	50	80

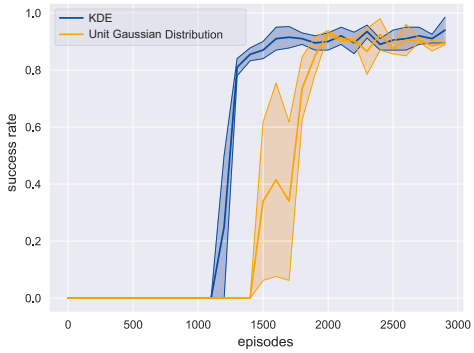


Figure 9. Learning curves of GENE with KDE and with a unit Gaussian distribution in Maze.

C. Goal Ant

The ant (Duan et al., 2016) within a square of $[-1, 1]^2$ and positioned at the center P_a tries to reach a goal with random location P_t within 500 timesteps. Only when the ant reaches the goal ($||P_a - P_t|| < 0.1$), it receives a reward of 1, otherwise always 0. Goal GAN consists of the discriminator with hidden layers $(256, 256)$ and the generator with hidden layers $(128, 128)$. The generator takes as input 4-dimension noise sampled from the unit Gaussian distribution. GAN is trained every 500 episodes with the learning rate 10^{-4} to generate goals for the following episodes.

D. Pushing

In Pushing, the robot learns to move a box randomly placed on the table of $[-0.16, 0.16]^2$ to a target location $(0.15, 0.15)$ in 50 timesteps. The robot obtains a reward of 0 only when the object is pushed at the target location, otherwise -1. The exploration noise of DDPG is $\mathcal{N}(0, 0.2^2)$. The setting of HER is the same as in (Andrychowicz et al., 2017), where HER is also evaluated in this task.

E. Cooperative Navigation

In Cooperative Navigation, there are the same number of landmarks (stationary) and agents (fixed initial positions) in a square of $[-1, 1]^2$. the radius of landmark is 0.15 and the radius of agent is 0.05. Only when each landmark is occupied by an agent ($||P_a - P_t|| < 0.2$), each agent receives a reward 1, otherwise always 0. In MADDPG, the networks are updated every 100 timesteps.

F. Sampling from Unit Gaussian Distribution

As VAE constrains that the latent space roughly follows a unit Gaussian distribution, we investigate the performance of GENE if we apply rejection sampling using the unit Gaussian distribution instead of the distribution estimated by KDE. We performed the experiment in Maze. As expected, GENE with KDE greatly outperforms GENE with unit Gaussian, as illustrated in Figure 9. The reason is that as the training progresses, the distribution of $\mathcal{R}_1$ also changes dynamically as shown in Figure 4. Therefore, we cannot simply use the unit Gaussian distribution for rejection sampling.