Intrinsically-Motivated Reinforcement Learning: A Brief Introduction

Mingqi Yuan
Tencent Robotics X Lab
Shenzhen, CN 518172
mingqiyuan@tencent.com

Abstract

Reinforcement learning (RL) is one of the three basic paradigms of machine learning. It has demonstrated impressive performance in many complex tasks like Go and StarCraft, which is increasingly involved in smart manufacturing and autonomous driving. However, RL consistently suffers from the exploration-exploitation dilemma. In this paper, we investigated the problem of improving exploration in RL and introduced the intrinsically-motivated RL. In sharp contrast to the classic exploration strategies, intrinsically-motivated RL utilizes the intrinsic learning motivation to provide sustainable exploration incentives. We carefully classified the existing intrinsic reward methods and analyzed their practical drawbacks. Moreover, we proposed a new intrinsic reward method via Rényi state entropy maximization, which overcomes the drawbacks of the preceding methods and provides powerful exploration incentives. Finally, extensive simulation demonstrated that the proposed module achieve superior performance with higher efficiency and robustness.

Preprint. Under Review.

Contents

1 Introduction

1.1 Background

Learning behavior permeates the whole life of human beings. However, it is extremely intricate to characterize or summarize the nature of learning. A natural idea is that we learn by interacting with our environment [1]. For instance, we observes the road condition in real time to adjust direction and speed to drive safely and smoothly. Take conversation as another example, we respond according to the statement of the other party. In this two cases, we are acutely aware of how our environment reacts to what we do, and try to influence what happens through our actions. Learning from interaction is a fundamental idea underlying almost all theories of learning and intelligence.

Reinforcement learning (RL) is a computational approach to learning from interaction. This term comes from behavioral psychology, which indicates that organisms take favorable strategies more frequently to draw on the advantages and avoid disadvantages. For instance, a general will make various deployments following some strategy in a battle. If one of his decisions leads him to victory, he will use this strategy more in future battles. [2] first introduced the term "reinforcement" to describe the phenomenon that specific incentives make organisms tend to adopt certain strategies. Furthermore, [3] classified reinforcements as positive reinforcements and negative reinforcements. Positive reinforcements make organisms tend to gain more benefits, while negative reinforcements make organisms avoid damage. RL has become one of the three paradigms of machine learning (ML), and achieved great success like AlphaGo [4].

A typical RL scenario is composed of an agent and an environment. The interactions between the agent and the environment are often modeled as a Markov decision process (MDP), which comprises a state space, an action space, a reward function and a transition probability [5]. In each time step, the agent observes the state of environment before selecting an action from the action space. After that, the state-action pair will be evaluated by the reward function. Finally, the objective of RL is to learn the optimal policy that maximizes the accumulative rewards in the long run. For policy learning, RL developed a large number of approaches that can be broadly categorized into optimal value algorithms and policy gradient algorithms. The former learns the optimal policy via estimating the optimal value functions, such as Q-learning [6], while the latter learns the optimal policy via policy gradient theorem, such as trust-region-policy-optimization (TRPO) and proximal-policy-optimization (PPO) [7, 8]. Take Q-learning for instance, its convergence condition is to visit all possible state-action pairs infinitely. However, many existing RL algorithms suffer from insufficient exploration mechanism. The agent cannot keep exploring the environment across episodes, especially when handling the complex environment with high-dimensional observations. As a result, the learned policy will prematurely fall into local optima after finite iterations [9].

The aforementioned phenomenon is called exploration-exploitation dilemma in RL. A representative example for demonstrating this dilemma is the K-armed bandit problem [10]. Each slot machine has a certain payout probability unknown to the gambler. The goal for the gambler is to identify the slot machine of the highest payout probability with the minimum number of attempts. One trivial approach for the gambler is to follow the exploration-only method by uniformly attempting all slot machines. However, such an approach usually requires a large number of attempts before the best slot machine is identified. On the other hand, the gambler can follow the exploitation method by focusing on the slot machine that gave the highest payout after a limited number of attempts. Clearly, this so-called the “best” slot machine may not be optimal at all. Both these methods cannot effectively maximize the payout with limited attempts. Thus, a comprehensive algorithm must be found by striking an expedient balance between exploitation and exploration.

Inspired by the discussions above, this thesis focuses on the exploration problem of RL to establish efficient and robust exploration methods. More specifically, we delve into the intrinsically-motivated RL that utilizes intrinsic learning motivation to improve exploration. In the following contents, we will introduce the identity of intrinsic learning motivation and transform the concept into a computational model. Finally, the effectiveness of the proposed methods is verified using realistic control tasks to demonstrate their advantages.

1.2 Related Work

The exploration problem of RL has been extensively studied. Exploration methods encourage RL agents to fully explore the state space in various manners, for instance by rewarding surprise [11, 12], curiosity [13, 14], information gain (IG) [15, 16], feature control [17] or diversity [18]. Another class of exploration methods following the insight of Thompson sampling [19, 20, 21, 22]. For instance, [21] utilizes a family of randomized Q-functions trained on bootstrapped data to choose actions, while [22] inserts noise into parameter space to encourage exploration. In this thesis, we focus on intrinsic motivation methods, which are extensively used and proven to be effective for extensive hard-exploration tasks. Intrinsic motivation is very useful in leading the exploration of RL agents, especially in environments where the extrinsic rewards are sparse or missing [23]. In particular, most effective and popular intrinsic motivation can be broadly classified into two approaches, namely state novelty-based methods and prediction error-based methods.

State novelty-based methods encourage the agent to visit as many novel states as possible. A representative approach is the count-based exploration that utilizes state visitation counts as exploration bonus in tabular settings [24]. [25, 26] further extend such methods to complex environments with high-dimensional observations. [27, 19] designs a state pseudo-count method based on context-tree switching density model, while [19] uses neural network as a state density estimator. In contrast, prediction error-based methods encourage the agent to fully explore the environment to reduce the uncertainty or error in predicting the consequences of its own actions. For instance, [14] utilizes the prediction error of random networks as exploration bonus to reward infrequently-seen states. [9] designs an intrinsic reward as the prediction error of predicting the encoded next state based on the current state-action pair. [13] further introduces an embedding network to learn the representation of state space and designs a inverse-forward pattern to generate intrinsic rewards. However, the prediction-error based methods suffer from the television dilemma, *i.e.*, the agent may hover in a novel state and stop exploration [28]. Moreover, the two kinds of methods generate vanishing rewards and cannot provide sustainable exploration incentives. In the following sections, we will dive into these problems and propose efficient solutions.

1.3 Contributions

The main contributions of this thesis are summarized as follows:

- We systematically introduce and analyze the exploration-exploitation dilemma in RL, and summarize the traditional strategic exploration algorithms;
- We detail the intrinsic learning motivation, and categorize various intrinsic reward generation methods. We further analyze the practical drawbacks of the existing intrinsic reward methods, and introduce the corresponding solutions;
- We designed a brand new intrinsic reward method entitled **R**ény**I** **S**tate **E**ntropy (RISE), which overcomes the drawbacks of the preceding methods and provides powerful exploration incentives;
- Finally, extensive simulation is performed to compare the performance of RISE against existing methods using both discrete and continuous control tasks as well as several hard exploration games. Simulation results confirm that the proposed module achieve superior performance with high efficiency and robustness.

1.4 Organization

The remainder of the thesis is organized as follows:

- Chapter 2 introduces the fundamentals of RL, including Markov decision process and some representative RL algorithms. These definitions and algorithms will run through the full text;
- Chapter 3 first demonstrates the exploration-exploitation dilemma in RL. After that, we introduce several classical exploration strategies both for tabular setting and deep RL, such as ϵ-greedy policy and Boltzmann exploration.
- Chapter 4 first analyze the intrinsic learning motivation of the agent before introducing the reward shaping method in RL. After that, we discussed two important kinds of intrinsic

reward methods, namely state novelty-based and prediction error-based approaches. Finally, we investigated some practical drawbacks of the discussed methods and proposed the corresponding solutions.

- Chapter 5 first analyzes the exploration problem from a new perspective before proposing a powerful intrinsic reward method, which provides high-quality intrinsic rewards via state entropy maximization. Extensive simulation demonstrates that this method can provide sustainable exploration incentives with less computational complexity and higher robustness.
- Chapter 6 concludes the full thesis and discusses the future work.

2 Fundamentals of Reinforcement Learning

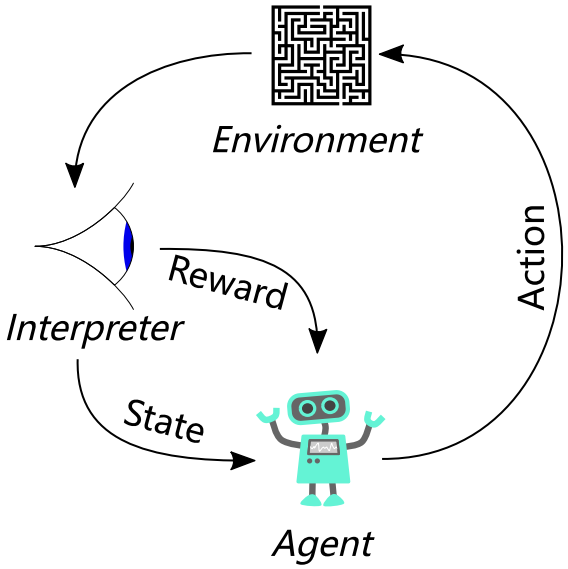

Figure 1: The typical framing of a reinforcement learning scenario.

2.1 Markov Decision Process

Figure 1 illustrates a typical RL scenario. We refer to the learner and decision maker as the *agent*, and the object it interacts with, *i.e.*, everything other than the agent, is called the *environment*. In RL, the interactions between the agent and the environment are often depicted as a Markov decision process (MDP), which is defined as [29]

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, r, \rho(\boldsymbol{s}_0), \gamma \rangle, \tag{1}$$

where

- $\mathcal{S}$ is a state space that can be finite or infinite. We assume that $\mathcal{S}$ is finite or countable infinite for convenience of discussion.
- $\mathcal{A}$ is an action space that can be also finite or infinite. A finite action space is often referred to the discrete control tasks, while an infinite action space is often referred to the continuous control tasks.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the transition probability, where $\Delta(\mathcal{S})$ is the space of probability distributions over $\mathcal{S}$. $\mathcal{T}(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})$ is the probability of transitioning into state $\boldsymbol{s}'$ if an action $\boldsymbol{a}$ is taken in state $\boldsymbol{s}$.
- $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function. $r(\boldsymbol{s}, \boldsymbol{a})$ is the immediate reward if an action $\boldsymbol{a}$ is taken in state $\boldsymbol{s}$.
- $\rho(\boldsymbol{s}_0) \in \Delta(\mathcal{S})$ is an initial state distribution to generate the initial state $\boldsymbol{s}_0$.
- $\gamma \in [0, 1]$ is a discount factor.

For the transition probability, we have:

$$\sum_{s'} \mathcal{T}(s'|s, a) = 1. \tag{2}$$

Figure 2 illustrates an example of Markov decision process, which has three possible states and two actions.

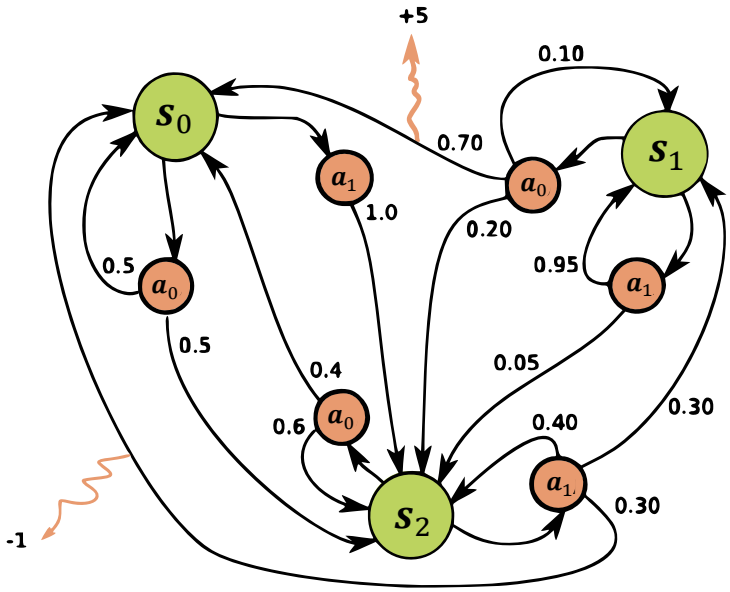

Figure 2: An example of Markov decision process.

When the agent takes the action a_0 in state s_1, the environment has a probability of 0.7 of transitioning into the state s_0, and the agent will get a reward of $+5$. In contrast, if the agent takes the action a_1 in state s_2, the environment has a probability of 0.3 of transitioning into s_0, and the agent will receive a negative reward of -1, which can be considered as a punishment.

2.1.1 Returns

So far we have introduced the basic components of a MDP, we next define the learning objective. Assume that a sequence of rewards is received after time step t:

$$R_t, R_{t+1}, R_{t+2}, R_{t+3}, \ldots, \tag{3}$$

our objective is to maximize the *expected discounted return*:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \tag{4}$$

In particular, the specific target of the agent is affected by the value of γ. If $\gamma < 1$, the sum has a finite value as long as the reward sequence is bounded. Furthermore, setting $\gamma = 0$ leads to $G_t = R_{t+1}$, which effectively restrict the agent to maximize the immediate reward. Finally, the return objective is more farsighted and thinks more highly of the future rewards when γ approaches 1. In particular, returns at successive time steps satisfy:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \ldots \\ &= R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \ldots \right) \\ &= R_{t+1} + \gamma G_{t+1}. \end{aligned} \tag{5}$$

This property is very important for the theory and algorithms of RL.

2.1.2 Policies and Value Functions

In the most general case, a *policy* specifies a decision strategy in which the agent adaptively selects actions based on the observations history. More specifically, a policy is a mapping from states to probabilities of selecting each possible action. A stationary policy π allows the agent to select actions based on the current state, *i.e.*, $\boldsymbol{a}_t \sim \pi(\cdot|\boldsymbol{s}_t)$, while a deterministic and stationary policy is of the form $\pi : \mathcal{S} \to \mathcal{A}$.

To evaluate the performance of policies, we define the *value function* to characterize the expected return. Denote by $V^\pi(\boldsymbol{s})$ the value function of a state $\boldsymbol{s}$ under a policy π, which represents the expected return when starting in $\boldsymbol{s}$ and following π thereafter. For Markov decision processes, we can formally define $V^\pi(\boldsymbol{s})$ as

$$\begin{aligned} V^\pi(\boldsymbol{s}) &= \mathbb{E}_\pi \left[G_t | S_t = \boldsymbol{s} \right] \\ &= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = \boldsymbol{s} \right], \forall \boldsymbol{s} \in \mathcal{S}, \end{aligned} \tag{6}$$

where V^π is called the state-value function for policy π.

Similarly, we define the value function $Q^\pi(\boldsymbol{s}, \boldsymbol{a})$ of a state-action pair under a policy π, which represents the expected return when starting from $\boldsymbol{s}$, taking the action $\boldsymbol{a}$, and thereafter following policy π:

$$\begin{aligned} Q^\pi(\boldsymbol{s}, \boldsymbol{a}) &= \mathbb{E}_\pi \left[G_t | S_t = \boldsymbol{s}, A_t = \boldsymbol{a} \right] \\ &= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = \boldsymbol{s}, A_t = \boldsymbol{a} \right], \forall \boldsymbol{s} \in \mathcal{S}, \end{aligned} \tag{7}$$

where Q^π is called the action-value function for policy π. Next, we introduce a fundamental property of value functions that is widely used in RL and dynamic programming, which satisfies recursive relationships similar to the return definition above. For any policy π and any state $\boldsymbol{s}$, we have the following Bellman equation:

$$\begin{aligned} V^\pi(\boldsymbol{s}) &= \sum_{\boldsymbol{a}} \pi(\boldsymbol{a}|\boldsymbol{s}) \sum_{\boldsymbol{s}'} \mathcal{T}\left(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a}\right) \left[r + \gamma V^\pi(\boldsymbol{s}') \right], \\ Q^\pi(\boldsymbol{s}, \boldsymbol{a}) &= \sum_{\boldsymbol{a}'} \pi(\boldsymbol{a}'|\boldsymbol{s}') \sum_{\boldsymbol{s}'} \mathcal{T}\left(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a}\right) \left[r + \gamma Q^\pi(\boldsymbol{s}', \boldsymbol{a}') \right]. \end{aligned} \tag{8}$$

The two equations will be used to design a popular RL algorithm in the following sections.

2.1.3 Optimal Policies and Optimal Value Functions

Given two policies π and π', we say π is better than or equal to π' if and only if its expected return is greater than or equal to that of π' for all states, *i.e.* $\pi \succeq \pi'$ if and only if

$$V^\pi(\boldsymbol{s}) \geq V^{\pi\prime}(\boldsymbol{s}), \forall \boldsymbol{s} \in \mathcal{S}. \tag{9}$$

We denote by π^* the optimal policy. Despite the fact that multiple optimal policies may exist, they share the same state-value function, called the optimal state-value function defined as

$$V^*(\boldsymbol{s}) = \max_\pi V^\pi(\boldsymbol{s}), \forall \boldsymbol{s} \in \mathcal{S}. \tag{10}$$

They also share the same optimal action-value function, denoted Q^*, and defined as

$$Q^*(\boldsymbol{s}, \boldsymbol{a}) = \max_\pi Q^\pi(\boldsymbol{s}, \boldsymbol{a}), \tag{11}$$

for all $\boldsymbol{s} \in \mathcal{S}$ and $\boldsymbol{a} \in \mathcal{A}$. To find the optimal policy, a trivial approach is to exhaustively search all the possible policies and compare the resulting expected returns. However, such an approach will incur prohibitive computational complexity. To address this problem, two representative RL algorithms have been introduced in the literature.

2.2 Optimal Value Algorithms

2.2.1 Q-learning

Q-learning is a value-based, model-free and off-policy RL algorithm [6]. An off-policy learner learns the value of the optimal policy that is independently of the actions of the agent. We first introduce a Bellman optimality equation before detailing the Q-learning. After substituting the optimal action-value function into the Bellman equation, we have:

$$\begin{aligned} Q^*(\boldsymbol{s}, \boldsymbol{a}) &= \sum_{\boldsymbol{a}'} \pi(\boldsymbol{a}'|\boldsymbol{s}') \sum_{\boldsymbol{s}'} \mathcal{T}\left(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a}\right) \left[r + \gamma Q^*(\boldsymbol{s}', \boldsymbol{a}')\right] \\ &= \sum_{\boldsymbol{s}'} \mathcal{T}\left(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a}\right) \left[r + \gamma \max_{\boldsymbol{a}'} \{Q^*(\boldsymbol{s}', \boldsymbol{a}')\}\right] \end{aligned} \tag{12}$$

Therefore, Q-learning defines the following iterative formula:

$$Q\left(S_t, A_t\right) \leftarrow Q\left(S_t, A_t\right) + \alpha \left[R_{t+1} + \gamma \max_{\boldsymbol{a}} Q\left(S_{t+1}, \boldsymbol{a}\right) - Q\left(S_t, A_t\right)\right], \tag{13}$$

where α is a step size. In this formula, the learned action-value function straightforwardly approximates Q^*, which is independent of the policy being followed. Such a setting significantly simplifies the analysis of the algorithm and convergence proofs. It is worth mentioning that the policy still has an effect because it determines which state-action pairs will be visited and updated. Furthermore, the convergence condition of Q-learning is to visit all possible state-action pairs infinitely. Finally, we summarize the detailed workflow of the Q-learning in Algorithm 1.

Algorithm 1 Q-learning for estimating $\pi \approx \pi^*$

Initialize $Q(\boldsymbol{s}, \boldsymbol{a})$ for all $\boldsymbol{s} \in \mathcal{S}, \boldsymbol{a} \in \mathcal{A}$ randomly, except $Q(\text{terminal}, \cdot) = 0$;
Initialize a step size $\alpha \in (0, 1]$ and a small $\epsilon > 0$;
Loop for each episode:
    Initialize S;
    Loop for each step of episode:
        Choose A from S using policy derived from Q (*e.g.*, "ϵ-greedy");
        Take action A, observe R, S';
        $Q\left(S_t, A_t\right) \leftarrow Q\left(S_t, A_t\right) + \alpha \left[R_{t+1} + \gamma \max_{\boldsymbol{a}} Q\left(S_{t+1}, \boldsymbol{a}\right) - Q\left(S_t, A_t\right)\right]$;
        $S \leftarrow S'$;
    until S is terminal.

2.2.2 Deep Q-learning

When handling complex environments with high-dimensional observations, it is critical to utilize function approximation as a compact representation of action values [30]. We represent the action-value function $Q(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\theta})$ by a neural network with parameters $\boldsymbol{\theta}$. Thus, the optimization objective is to find $\boldsymbol{\theta}$ such that $Q(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\theta}) \approx Q^*(\boldsymbol{s}, \boldsymbol{a})$. Therefore, the optimal parameters can be approximated by minimizing the squared temporal difference error:

$$L(\boldsymbol{\theta}) = \left[r + \gamma \max_{\boldsymbol{a}'} Q(\boldsymbol{s}', \boldsymbol{a}', \boldsymbol{\theta}) - Q(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\theta})\right]^2. \tag{14}$$

Despite its simplicity, Q-learning suffers from low efficiency and inadequate convergence when handling complex environments like Atari games.

2.3 Policy Gradient Algorithms

The optimal value algorithms learn the optimal policies by estimating the optimal action-value function. Alternatively, it is feasible to omit the estimation procedure and straightforwardly approximate the optimal policies via a parameterized function, and update its parameters iteratively. Since the iterative procedure corresponds to the gradient of policy, such algorithms are called as policy gradient algorithms.

2.3.1 Policy Gradient Theorem

Recalling the identity of the stationary policy, it satisfies

$$\sum_{\boldsymbol{a}} \pi(\boldsymbol{a}|\boldsymbol{s}) = 1, \tag{15}$$

for all $\boldsymbol{s} \in \mathcal{S}$. We denote by $\pi(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{\theta})$ the parameterized policy, it should also holds the normalization condition and permits differentiability [31]. Therefore, we introduce an action preference function $h(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\theta})$ and define the following policies:

$$\pi(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{\theta}) = \frac{\exp\{h(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\theta})\}}{\sum_{\boldsymbol{a}' \in \mathcal{A}} \exp\{h(\boldsymbol{s}, \boldsymbol{a}', \boldsymbol{\theta})\}}, \tag{16}$$

where h may have multiple forms, such as linear combinations, neural network and so on. Next, we introduce the policy gradient theorem as follows:

Theorem 1. *For episodic case, denote by $J(\boldsymbol{\theta}) = V^{\pi}(\boldsymbol{s}_0)$ the expected return of policy π, the gradients of $J(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$ can be expressed as*

$$\nabla J(\boldsymbol{\theta}) \propto \sum_{\boldsymbol{s}} \mu(\boldsymbol{s}) \sum_{\boldsymbol{a}} Q^{\pi}(\boldsymbol{s}, \boldsymbol{a}) \nabla \pi(\boldsymbol{a}|\boldsymbol{s}), \tag{17}$$

where μ is the on-policy distribution under policy π.

Proof. See proof in [1]. □

Theorem 1 indicates that we can obtains the gradients of the expected return as long as the parameters of the policy are given. As a result, we can update $\boldsymbol{\theta}$ following the gradients to increase the expected return. Note that the right side of the policy gradient is a sum over states weighted by the frequency of the states occurrence under the target policy. Thus, the policy gradient can be expressed as

$$\begin{aligned} \nabla J(\boldsymbol{\theta}) &\propto \sum_{\boldsymbol{s}} \mu(\boldsymbol{s}) \sum_{\boldsymbol{a}} Q^{\pi}(\boldsymbol{s}, \boldsymbol{a}) \nabla \pi(\boldsymbol{a}|\boldsymbol{s}), \\ &= \mathbb{E}_{\pi}\left[\sum_{\boldsymbol{a}} Q^{\pi}(S_t, \boldsymbol{a}) \nabla \pi(\boldsymbol{a}|S_t, \boldsymbol{\theta}) \right] \\ &= \mathbb{E}_{\pi}\left[G_t \frac{\nabla \pi\left(A_t|S_t, \boldsymbol{\theta}\right)}{\pi\left(A_t|S_t, \boldsymbol{\theta}\right)} \right] \\ &= \mathbb{E}_{\pi}\left[G_t \nabla \ln \pi\left(A_t|S_t, \boldsymbol{\theta}\right) \right]. \end{aligned} \tag{18}$$

Therefore, we can derive the following vanilla policy gradient (VPG) algorithm [32]:

Algorithm 2 Vanilla Policy Gradient

Randomly initialize the policy π using parameters $\boldsymbol{\theta}$;
Initialize a step size α;
Loop for each episode:
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi_{\boldsymbol{\theta}}$;
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$;
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi\left(A_t|S_t, \boldsymbol{\theta}\right)$.

2.3.2 Proximal Policy Optimization

In Algorithm 2, we use Monte-Carlo sampling to calculate the return G, and the policy can only be updated at the end of the episode. To promote the learning efficiency, it is reasonable to leverage a parameterized function to approximate the value function and estimate the expected return. After that, the estimated return is used to compute the policy gradients and update the policy parameters. Such a setting is called *actor-critic* method. In particular, an efficient algorithm entitled proximal

policy optimization (PPO) was developed in [7]. PPO is a model-free, on-policy, and actor-critic RL algorithm .

We consider updating the policy $\pi(\boldsymbol{\theta})$ iteratively. Suppose that a new policy $\pi(\boldsymbol{\theta}_k)$ is derived at time step k, the following equality holds:

$$\mathbb{E}_{\pi(\boldsymbol{\theta})}\left[G_0\right] = \mathbb{E}_{\pi(\boldsymbol{\theta}_k)}\left[G_0\right] + \mathbb{E}_{\pi(\boldsymbol{\theta})}\left[\sum_{t=0}^{+\infty}\gamma^t Z_{\pi(\boldsymbol{\theta}_k)}\left(S_t, A_t\right)\right], \tag{19}$$

where $Z_\pi = Q^\pi - V^\pi$ is the advantage function. To maximize $\mathbb{E}_{\pi(\boldsymbol{\theta})}\left[G_0\right]$, it suffices to maximize

$$\mathbb{E}_{\pi(\boldsymbol{\theta})}\left[\sum_{t=0}^{+\infty}\gamma^t Z_{\pi(\boldsymbol{\theta}_k)}\left(S_t, A_t\right)\right]. \tag{20}$$

By applying importance sampling, we have:

$$\mathbb{E}_{S_t, A_t\sim\pi(\boldsymbol{\theta})}\left[Z_{\pi(\boldsymbol{\theta}_k)}\left(S_t, A_t\right)\right] = \mathbb{E}_{S_t\sim\pi(\boldsymbol{\theta}), A\sim\pi(\boldsymbol{\theta}_k)}\left[\frac{\pi\left(A_t|S_t, \boldsymbol{\theta}\right)}{\pi\left(A_t|S_t, \boldsymbol{\theta}_k\right)} Z_{\pi(\boldsymbol{\theta}_k)}\left(S_t, A_t\right)\right] \tag{21}$$

However, it is difficult to calculate the expectation with respect to $S_t \sim \pi(\boldsymbol{\theta})$. To address this problem, we approximate the expectation of $S_t \sim \pi(\boldsymbol{\theta})$ as $S_t \sim \pi(\boldsymbol{\theta}_k)$, which is called surrogate advantage, *i.e.*,

$$\mathbb{E}_{S_t, A_t\sim\pi(\boldsymbol{\theta})}\left[Z_{\pi(\boldsymbol{\theta}_k)}\left(S_t, A_t\right)\right] \approx \mathbb{E}_{S_t, A_t\sim\pi(\boldsymbol{\theta}_k)}\left[\frac{\pi\left(A_t|S_t, \boldsymbol{\theta}\right)}{\pi\left(A_t|S_t, \boldsymbol{\theta}_k\right)} Z_{\pi(\boldsymbol{\theta}_k)}\left(S_t, A_t\right)\right]. \tag{22}$$

Therefore, we obtain the approximation of $\mathbb{E}_{\pi(\boldsymbol{\theta})}\left[G_0\right]$:

$$L(\boldsymbol{\theta}) = \mathbb{E}_{\pi(\boldsymbol{\theta}_k)}\left[G_0\right] + \mathbb{E}_{S_t, A_t\sim\pi(\boldsymbol{\theta}_k)}\left[\sum_{t=0}^{+\infty}\gamma^t \frac{\pi\left(A_t|S_t, \boldsymbol{\theta}\right)}{\pi\left(A_t|S_t, \boldsymbol{\theta}_k\right)} Z_{\pi(\boldsymbol{\theta}_k)}\left(S_t, A_t\right)\right]. \tag{23}$$

It is straightforward to show that $\mathbb{E}_{\pi(\boldsymbol{\theta})}\left[G_0\right]$ and $L(\boldsymbol{\theta})$ have same gradients when $\boldsymbol{\theta} = \boldsymbol{\theta}_k$. Thus, it is possible to learn better policies via optimizing the surrogate advantage. PPO redesigned the optimization objective as

$$\mathbb{E}_{\pi(\boldsymbol{\theta}_k)}\left[\min\left(\frac{\pi\left(A_t|S_t, \boldsymbol{\theta}\right)}{\pi\left(A_t|S_t, \boldsymbol{\theta}_k\right)} Z_{\pi(\boldsymbol{\theta}_k)}, Z_{\pi(\boldsymbol{\theta}_k)} + \epsilon|Z_{\pi(\boldsymbol{\theta}_k)}|\right)\right], \tag{24}$$

where $\epsilon \in (0, 1)$ is a weighting coefficient. This objective can stabilize the update process by controlling the gap between the new policy and the old policy. Moreover, PPO is very efficient and easy to implement, serving as the baseline in many RL experiments.

3 Classical Exploration Strategies

3.1 K-Armed Bandit Problem

In Chapter 2, the convergence condition of RL algorithms requires visiting all possible state-action pairs infinitely. In other words, the agent need to fully explore the environment to find the optimal policies. However, there is a critical tradeoff between the exploitation and exploration in RL, which can be shown via the K-armed bandit problem. As shown in Figure 3, a gambler can choose to press one of the arms after putting in a coin. Each arm will spit out a coin with a certain probability, but this probability is unknown for the gambler. The goal of gamblers is to maximize their return through certain policies, *i.e.*, to get the most coins.

If we want to know the expected return of each arm, we can distribute the trial opportunities equally to all the arms, *i.e.*, the exploration-only method. Based on the outcome, we can estimate the expected return of each arm using the average payout probability. In contrast, if we want to execute the action with maximum reward, it suffices to apply the exploitation-only method, *i.e.*, to press the current optimal arm. Therefore, the exploration-only method can estimate the return of each arm accurately at the cost of the large resources spent on identifying the optimal arm. In contrast, the exploitation-only method only focus the current return without spending further efforts in finding the true optimal arm, which is shortsighted. Both these two methods cannot effectively maximize the long-term rewards. Since the number of total attempts is finite, a comprehensive algorithm must be found by striking an appropriate balance between exploitation and exploration.

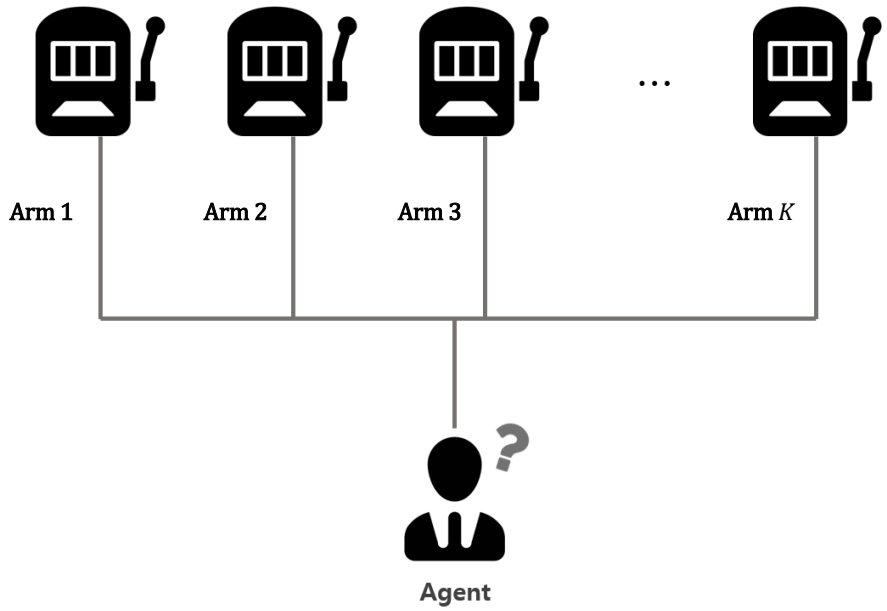

Figure 3: A K-armed bandit.

3.2 Soft Policy

To cope with the problem above, a well-known method entitled ϵ-greedy algorithm was proposed in the literature. In on-policy control, we say that a policy is generally *soft*, if it holds

$$\pi(\boldsymbol{a}|\boldsymbol{s}) > 0, \tag{25}$$

for all $\boldsymbol{s} \in \mathcal{S}$ and all $\boldsymbol{a} \in \mathcal{A}$. A soft policy can select all possible actions, which allows the agent to visit more possible states and state-action pairs. The ϵ-greedy policy is a special case of the soft policies. It selects an action corresponding to the maximal estimated action value with probability $1-\epsilon$, while any random action with probability ϵ. More specifically, it follows

$$\pi\left(\boldsymbol{a}|S_t\right) \leftarrow \begin{cases} 1-\epsilon+\epsilon/\left|\mathcal{A}\left(S_t\right)\right| & \text{if } \boldsymbol{a}=A^*, \\ \epsilon/\left|\mathcal{A}\left(S_t\right)\right| & \text{if } \boldsymbol{a} \neq A^*, \end{cases} \tag{26}$$

where $A^* \leftarrow \underset{\boldsymbol{a}}{\operatorname{argmax}}\, Q(S_t, \boldsymbol{a})$. Moreover, we have the following theorem:

Theorem 2. *The ϵ-greedy policy satisfies the policy improvement theorem.*

Proof. See proof in [1]. □

Now we are ready to design the ϵ-greedy algorithm for K-armed bandit problem. Let $Q_n(k)$ denote the average return of the k-th arm after the n-th attempt, we have the following update formula:

$$Q_n(k) = \frac{1}{n}\left((n-1) \times Q_{n-1}(k) + v_n\right), \tag{27}$$

where v_n denotes the reward of the n-th attempt. Finally, we summarize the ϵ-greedy algorithm for K-armed bandit problem in Algorithm 3. Note that $Q(i)$ denotes the average return of the i-th arm, $\mathrm{count}(i)$ denotes the number of attempts, and $\mathrm{rand}()$ will randomly generates a number from $[0, 1]$.

When the reward distribution has heavier tails, a larger ϵ is needed to encourage more exploration. Furthermore, for a sufficiently large number of attempts, all the arm rewards can be accurately estimated. As a result, no further exploration is required. Therefore, it suffices to make ϵ decay with attempts, such as $\epsilon = 1/\sqrt{t}$.

Algorithm 3 ϵ-greedy Algorithm

Input: Number of arms K, a reward function r, number of attempts T, exploration probability ϵ;
Set $G = 0$;
Initialize $Q(i) = 0, \text{count}(i) = 0$ for $i = 1, 2, \ldots, K$;
for $t = 1, 2, \ldots, T$ **do**
  if $\text{rand}() < \epsilon$ **then**
    Randomly select a k from $1, 2, \ldots, K$;
  else
    $k = \underset{i}{\text{argmax}}\, Q(i)$;
  end if
  $v = r(k)$;
  $G = G + v$;
  $Q(k) = \frac{Q(k) \cdot \text{count}(k) + v}{\text{count}(k) + 1}$;
  $\text{count}(k) = \text{count}(k) + 1$;
end for
Output: The accumulative rewards G.

3.3 Upper Confidence Bound

The ϵ-greedy method selects non-greedy actions with equal probability, regardless of their estimated returns derived from the past attempts. However, it is more reasonable to choose actions based on their estimated returns by taking into account both their expected return and the uncertainty associated with these estimates. Inspired by this observation, the upper confidence bound (UCB) algorithm selects actions by [33]

$$k = \underset{i}{\text{argmax}} \left[Q(i) + c \sqrt{\frac{\ln t}{N_t(i)}} \right], \tag{28}$$

where $N_t(i)$ is the number of times that the i-th arm has been selected prior to time t, and $c > 0$ controls the degree of exploration. The square-root term measures the uncertainty in the return estimation of the i-th arm. The uncertainty term decreases when $N_t(i)$ increases. On the opposite, if the i-th arm is not selected but t increases, the uncertainty estimate increases. As a result, all arms will eventually be selected with arms of either low estimated return or high selection frequency being selected with decaying frequency.

3.4 Thompson Sampling

In Thompson sampling (TS), the agent tracks a belief over the probability distribution of the optimal arms and samples from this distribution, which is shown to solve the K-armed bandit problem more effectively [34]. More specifically, TS assumes that $Q(i)$ follows a Beta distribution that is a family of continuous probability distributions defined on the interval $[0, 1]$ parameterized by two positive shape parameters, namely α and β. At each time step t, TS samples an expected reward $\tilde{Q}(i)$ from the prior distribution $\text{Beta}(\alpha_i, \beta_i)$ for every arm selection. After that, the action is chosen by

$$k_{TS} = \underset{i}{\text{argmax}}\, \tilde{Q}(i). \tag{29}$$

After the true reward is observed, the Beta distribution is updated following

$$\begin{aligned} \alpha_i &\leftarrow \alpha_i + r(k) \mathbb{1}[k_{TS} = i], \\ \beta_i &\leftarrow \beta_i + (1 - r(k)) \mathbb{1}[k_{TS} = i], \end{aligned} \tag{30}$$

where $\mathbb{1}$ is the indicator function. Since TS samples reward estimations from prior distributions and the reward probability corresponding to each action is currently considered to be the optimal, it actually implements the idea of probability matching.

3.5 Boltzmann Exploration

Next, we review a more straightforward but efficient exploration method entitled the Boltzmann Exploration. The Boltzmann exploration compromises the exploitation and exploration based on

the known average return. The key insight of the Boltzmann exploration is to give higher selection probability to arms with higher average returns. The Boltzmann distribution is defined as [35]

$$P(k) = \frac{e^{\frac{Q(k)}{\tau}}}{\sum_{i=1}^{K} e^{\frac{Q(i)}{\tau}}}, \tag{31}$$

where $Q(i)$ is the average return of the i-th arm, $\tau > 0$ is a coefficient called as *temperature*. The preference for exploration increases with the value of τ.

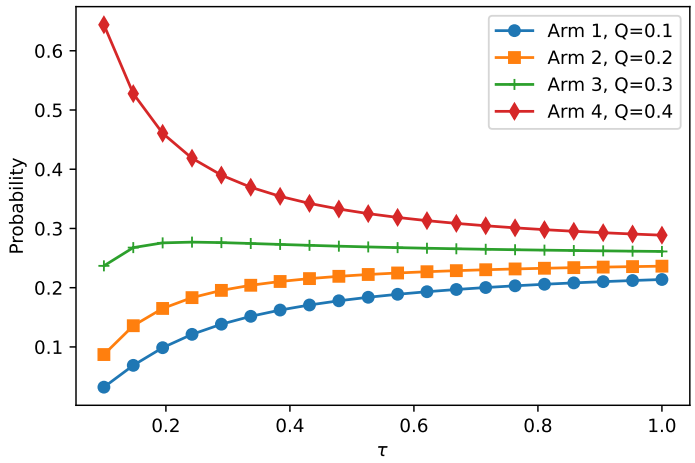

Figure 4: Selection probabilities versus the value of τ.

For illustration purposes, We consider an example of four arms with average return 0.1, 0.2, 0.3 and 0.4, Figure 4 illustrates the relation between the selection probabilities and τ. Inspection of Figure 4 suggests that the selection probabilities tend to be equal when the temperature increases. Equipped with the Boltzmann distribution, we summarize the detailed exploration algorithm in Algorithm 4.

Algorithm 4 Boltzmann Exploration Algorithm

Input: Number of arms K, a reward function r, number of attempts T, temperature coefficient τ;
Set $G = 0$;
Initialize $Q(i) = 0, \text{count}(i) = 0$ for $i = 1, 2, \ldots, K$;
for $t = 1, 2, \ldots, T$ **do**
  Sample k based on $P(k)$;
  $v = r(k)$;
  $G = G + v$;
  $Q(k) = \frac{Q(k) \cdot \text{count}(k) + v}{\text{count}(k) + 1}$;
  $\text{count}(k) = \text{count}(k) + 1$;
end for
Output: The accumulative rewards G.

3.6 Action Entropy Maximization

Exploration methods like ϵ-greedy policy and Boltzmann exploration are prone to eventually learn the optimal policy in tabular setting. However, they are inefficient and may be futile when handling complex environments with high-dimensional observations. To address the exploration problem in deep RL, an effective method is action entropy maximization. In contrast to the aforementioned methods, action entropy maximization is a passive exploration method that adds an entropy term into the loss function while utilizing neural networks for function approximation.

We first formally define the Shannon entropy as follows [36]:

Definition 1. *Let $X \in \mathbb{R}^m$ be a random vector that has a density function $f(\boldsymbol{x})$ with respect to Lebesgue measure on $\mathbb{R}^m$, and let $\mathcal{X} = \{\boldsymbol{x} \in \mathbb{R}^m : f(\boldsymbol{x}) > 0\}$ be the support of the distribution. The Shannon entropy is defined as:*

$$H(f) = -\int_{\mathcal{X}} f(\boldsymbol{x}) \log f(\boldsymbol{x}) d\boldsymbol{x}. \tag{32}$$

Equipped with Definition 1, the action entropy is calculated as

$$H(\pi(\cdot|\boldsymbol{s})) = -\int_{\mathcal{A}} \pi(\boldsymbol{a}|\boldsymbol{s}) \log \pi(\boldsymbol{a}|\boldsymbol{s}) d\boldsymbol{a}. \tag{33}$$

For discrete action space, it has a simpler form:

$$H(\pi(\cdot|\boldsymbol{s})) = \sum_{\boldsymbol{a} \in \mathcal{A}} -\pi(\boldsymbol{a}|\boldsymbol{s}) \log \pi(\boldsymbol{a}|\boldsymbol{s}). \tag{34}$$

Similar to Boltzmann exploration, action entropy evaluates the preference of exploration and exploitation. For instance, let $|\mathcal{A}| = 4, \pi(\cdot|\boldsymbol{s}_0) = \{0.1, 0.2, 0.3, 0.4\}$, then we have

$$H(\pi(\cdot|\boldsymbol{s}_0)) = 1.28. \tag{35}$$

Let $\pi(\cdot|\boldsymbol{s}_1) = \{0.25, 0.25, 0.25, 0.25\}$, then

$$H(\pi(\cdot|\boldsymbol{s}_1)) = 1.39. \tag{36}$$

It is natural to find that the decentralized action probabilities produce higher action entropy, which encourages the agent to visit the action space more evenly. Therefore, it is feasible to utilize the action entropy as an additional reward, and redefine the objective of RL as

$$\pi^* = \underset{\pi \in \Pi}{\operatorname{argmax}}\, \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{T-1} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \alpha H(\pi(\cdot|\boldsymbol{s}_t)) \right], \tag{37}$$

where Π is the set of all stationary polices, $\tau = (\boldsymbol{s}_0, \boldsymbol{a}_0, \ldots, \boldsymbol{a}_{T-1}, \boldsymbol{s}_T)$ is the trajectory collected by the agent, and α is the temperature parameter that determines the importance of the entropy item [37]. Moreover, we have the following convergence theorem:

Theorem 3. *Repeated application of soft policy evaluation and soft policy improvement from any $\pi \in \Pi$ can converge to the optimal policy π^*.*

Proof. See proof in [38]. □

A well-known algorithm for solving Eq. (37) is soft actor-critic that can be found in [37]. Note that the entropy regularizer can be used in conjunction with any RL algorithms.

3.7 Noise-Based Exploration

Another representative exploration method is the noise-based exploration, which adds noise into observation, action and even parameter space. Active exploration methods like ϵ-greedy policy and Boltzmann exploration are often used for discrete action spaces. In fact, the ϵ-greedy policy can also be used in continuous control tasks, *i.e.*, to randomly sample an action with probability ϵ in each time step. In [39], a deep deterministic policy gradient (DDPG) is proposed that solves the problem of exploration independently from the learning algorithm. DDPG constructs an exploration policy by adding noise sampled from a noise process:

$$\pi'(\boldsymbol{s}) = \pi(\boldsymbol{s}, \boldsymbol{\theta}) + \mathcal{N}, \tag{38}$$

where π is a deterministic policy and $\mathcal{N}$ can be chosen to adapt the environment. For instance, we can use Ornstein-Uhlenbeck process (OUP) [40] or Gaussian process.

In contrast, [22] proposes to improve exploration via parameter space noise. As shown in Figure 5, parameter space noise directly injects randomness into the parameters of the agent, disturbing the types of decisions it makes such that they always fully depend on what the agent currently senses.

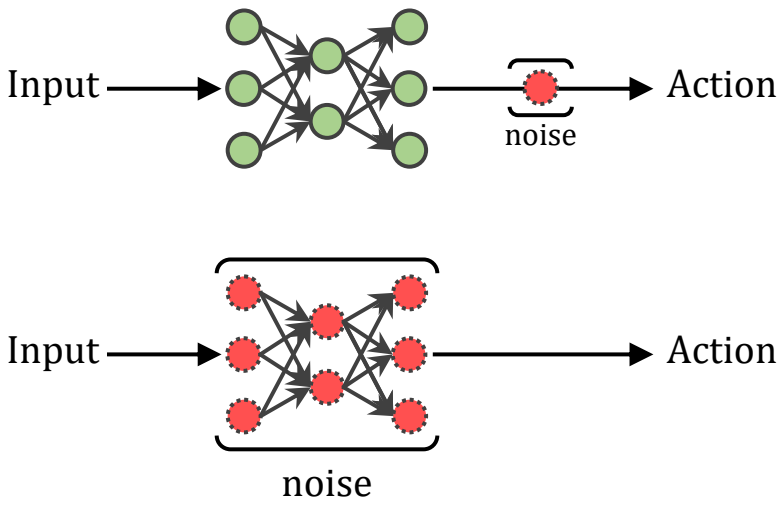

Figure 5: Action space noise (top), compared to parameter space noise (bottom).

Denote by $\boldsymbol{\theta}$ the parameters of the agent, then the perturbed parameters $\tilde{\boldsymbol{\theta}}$ are computed as

$$\tilde{\boldsymbol{\theta}} = \boldsymbol{\theta} + \mathcal{N}, \tag{39}$$

where $\mathcal{N}$ is a noise process, such as Gaussian noise. In particular, an adaptive variance σ is defined as

$$\sigma_{k+1} = \begin{cases} \alpha\sigma_k & \text{if } d(\pi, \tilde{\pi}) \leq \delta \\ \frac{1}{\alpha}\sigma_k & \text{otherwise} \end{cases}, \tag{40}$$

where $\alpha \in \mathbb{R}_+$ is a scaling factor, $\delta \in \mathbb{R}_+$ is a threshold value, and $d(\cdot, \cdot)$ is a distance measure that depends on the concrete algorithm. Figure 6 illustrates the performance comparison with action-noise methods.

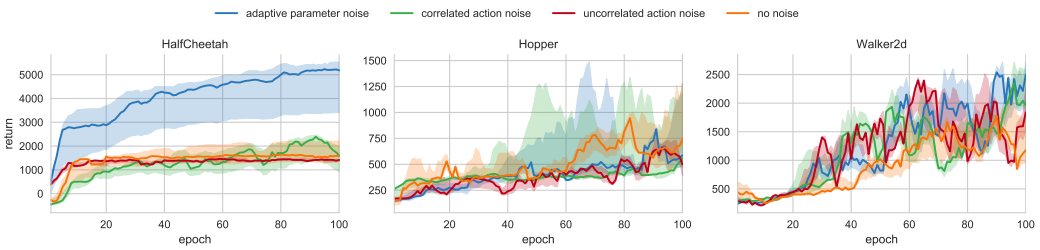

Figure 6: Median DDPG returns for continuous control environments plotted over epochs.

It is interesting to find in Figure 6 that parameter noise actually helped algorithms explore their environments more effectively, leading to higher scores. This is because adding noise to the parameters will make the exploration consistent across episodes, while adding noise to the action space will lead to unpredictable exploration.

4 Intrinsically-Motivated Exploration

4.1 Intrinsic Motivation of Learning

Most exploration methods in Chapter 3 promotes the exploration of action space, which encourages the agent to visit more unknown state-action pairs to learn better policies. However, these techniques can eventually learn the optimal policy in the tabular setting, but they are inefficient when handling

complex environments with high-dimensional observations [13]. Despite the advantages of the action entropy maximization and noise-based exploration, they cannot significantly improve the critical exploration problem of state space. To address this problem, [41] analyzed the learning motivation of the agent, which can be distinguished as extrinsic motivation and intrinsic motivation. Extrinsic motivation refers to being moved to do something because of some specific rewarding mechanism. For instance, we want to get more points in Atari games or more grades in examination. In contrast, intrinsic motivation refers to being moved to do something because it is inherently pleasant. When an infant plays, waves its arms or hums, he has no explicit teacher. Most trappings of modern life like college and a good job are so far into the future, which provides no effective reinforcement signal. Therefore, intrinsic motivation leads agents to participate in exploration, games and other behaviors driven by curiosity without explicit reward. In this chapter, we attempt to evaluate intrinsic motivation and transform it into quantized rewards for improving exploration.

4.2 Reward Shaping

Reward shaping is a very powerful technique for scaling up RL methods to handle complex problems by supplying additional rewards to the agent to guide its learning process [42]. The most well-known reward shaping method is potential-based reward shaping (PBRS) [43]. We first formally define the potential-based shaping function as follows:

Definition 2. *A shaping reward function $f : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is potential-based if there exists $\phi : \mathcal{S} \to \mathbb{R}$, such that*

$$f(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') = \gamma\phi(\boldsymbol{s}') - \phi(\boldsymbol{s}), \tag{41}$$

for all $\boldsymbol{s} \neq \boldsymbol{s}_0, \boldsymbol{s}'$.

For instance, we can utilize the state-value function as the shaping function:

$$f(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') = \gamma V^{\pi}(\boldsymbol{s}') - V^{\pi}(\boldsymbol{s}). \tag{42}$$

Furthermore, we have the following theorem:

Theorem 4. *If f is a potential-based shaping function, then every optimal policy in $\mathcal{M}' = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, r + f, \rho(\boldsymbol{s}_0), \gamma \rangle$ will also be an optimal policy in $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, r, \rho(\boldsymbol{s}_0), \gamma \rangle$ (and vice versa).*

This theorem demonstrates that PBRS can guarantee the policy invariance while adjust the learning process. Consider the following GridWorld game, in which the agent needs to move from the black node to the red node with minimum steps, and the reward is -1 per step.

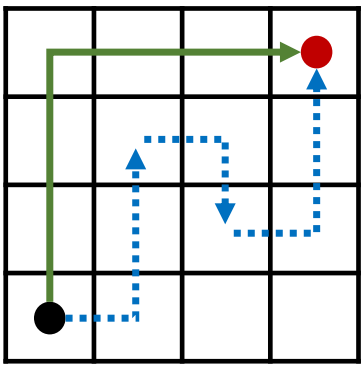

Figure 7: A GridWorld game.

Therefore, one estimate of the state-value function is

$$\phi_0(\boldsymbol{s}) = -D_{\mathrm{M}}(\boldsymbol{s}, \mathrm{GOAL}), \tag{43}$$

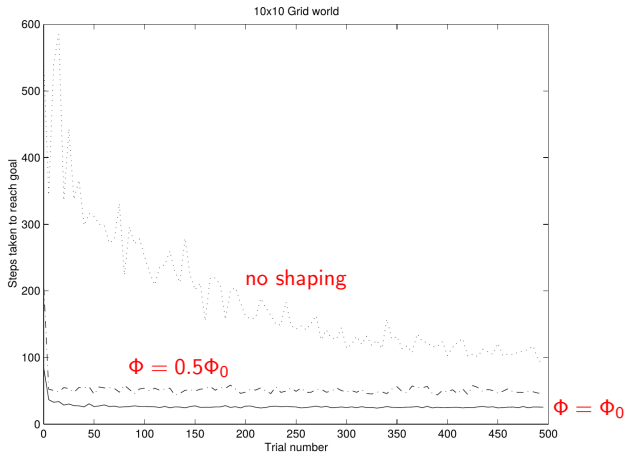

Figure 8: Experiments for 10×10 GridWorld.

where D_{M} is the Manhattan distance. Then we use ϕ_0 and $0.5\phi_0$ as potential functions, and the performance comparison is illustrated in Figure 8. It is obvious that the training process is significantly accelerated with the shaped rewards.

In the following sections, we will introduce multifarious intrinsic rewards. In general, all of these methods belong to reward shaping approaches. But they may not follow the form of the potential-based functions, *i.e.*, they may change what the optimal policy is. However, extensive experiments demonstrate that the intrinsic rewards effectively improve the exploration efficiency and allow the agent to learn better policies [44]. In particular, most formulations of intrinsic reward can be broadly categorized into two classes:

- Encourage the agent to explore novel states or state-action pairs;
- Encourage the agent to take actions that reduce the uncertainty of predicting the consequence of its own actions, *i.e.*, the knowledge of the environment.

4.3 Novelty-Based Intrinsic Rewards

4.3.1 Count-Based Exploration

We begin with a count-based exploration method proposed by [19], which leverages the pseudo-count method to evaluate the novelty of states. Let ψ denotes a density model on a finite space $\mathcal{S}$, and $\psi_n(\boldsymbol{s})$ is the probability after being trained on a sequence of states $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n$. Assume $\psi_n(\boldsymbol{s}) > 0$ for all $\boldsymbol{s}, n$, let $\psi'_n(\boldsymbol{s})$ be the recording probability the model would assign to $\boldsymbol{s}$ if it was trained on the same $\boldsymbol{s}$ one more time. We say that ψ is learning-positive if $\psi'_n(\boldsymbol{s}) > \psi_n(\boldsymbol{s})$, and the prediction gain (PG) of ψ is

$$\mathrm{PG}_n(\boldsymbol{s}) = \log \psi'_n(\boldsymbol{s}) - \log \psi_n(\boldsymbol{s}). \tag{44}$$

Then we define the pseudo-count as

$$\hat{\mathrm{N}}_n(\boldsymbol{s}) = \frac{\psi_n(\boldsymbol{s})\,(1 - \psi'_n(\boldsymbol{s}))}{\psi'_n(\boldsymbol{s}) - \psi_n(\boldsymbol{s})}, \tag{45}$$

derived by assuming that a single observation of $\boldsymbol{s} \in \mathcal{S}$ leads to a unit increase in pseudo-count:

$$\psi_n(\boldsymbol{s}) = \frac{\hat{\mathrm{N}}_n(\boldsymbol{s})}{\hat{n}}, \quad \psi'_n(\boldsymbol{s}) = \frac{\hat{\mathrm{N}}_n(\boldsymbol{s}) + 1}{\hat{n} + 1}, \tag{46}$$

where $\hat{n}$ is the pseudo-count total. Upon certain assumptions on ψ_n, pseudo-counts grow approximately linearly with real counts. Moreover, the pseudo-count can be approximated as

$$\hat{\mathrm{N}}_n(\boldsymbol{s}) \approx \left(e^{\mathrm{PG}_n(\boldsymbol{s})} - 1\right)^{-1}. \tag{47}$$

Finally, we define the intrinsic reward at step n as

$$\hat{r}(\boldsymbol{s}) = \sqrt{1/\hat{N}_n(\boldsymbol{s})}, \tag{48}$$

which encourages the agent to try to re-experience surprising states. On the contrary, if a state is visited multiple times, the agent will visit it with decay frequency.

4.3.2 Random Network Distillation

Count-based exploration is a straightforward method for evaluating the state novelty. But it suffers from complicated computation procedures and the pseudo-count method may produce large variance. To address the problem, [14] proposed a random network distillation (RND) method that utilizes neural network to record the state novelty. As shown in Figure 9, RND consists of two neural networks, namely a *target* network and a *predictor* network. The target network is fixed and randomly initialized, while the predictor network is trained on the observation data collected by the agent.

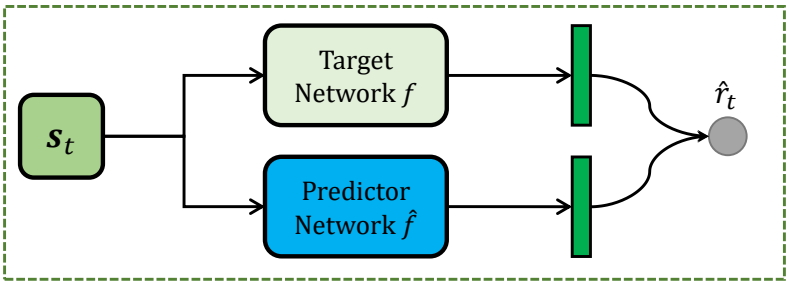

Figure 9: The overview of RND.

Let $f : \mathcal{S} \to \mathbb{R}^m$ denote the target network, and $\hat{f} : \mathcal{S} \to \mathbb{R}^m$ denote the predictor network with parameters $\boldsymbol{\theta}$, where m is the embedding size. Then the predictor network is trained by minimizing the following loss function:

$$L(\boldsymbol{\theta}) = \|\hat{f}(\boldsymbol{s}, \boldsymbol{\theta}) - f(\boldsymbol{s})\|_2^2. \tag{49}$$

Thus the predictor network will be distilled into the target network, and novel states will produce higher prediction error. Finally, we summarize the RND algorithm in Algorithm 5.

Algorithm 5 Random Network Distillation

Input: A target network f, a predictor network $\hat{f}$, a policy π, a number N_{opt} of optimization steps, a replay buffer $\mathcal{B}$.
Loop for each episode:
  Sample state $\boldsymbol{s}_0$ from $\rho(\boldsymbol{s}_0)$;
  Set $t = 0$;
  Loop for each step of episode:
    Sample $\boldsymbol{a}_t \sim \pi(\cdot|\boldsymbol{s}_t)$;
    Sample $\boldsymbol{s}_{t+1} \sim \mathcal{T}(\boldsymbol{s}_{t+1}|\boldsymbol{s}, \boldsymbol{a})$;
    Calculate the intrinsic reward $\hat{r}_t = \|\hat{f}(\boldsymbol{s}_{t+1}) - f(\boldsymbol{s}_{t+1})\|$;
    Save the transition $(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}, r_t, \hat{r}_t)$ into $\mathcal{B}$.
  Until $\boldsymbol{s}_{t+1}$ is terminal;
  for $i = 1, 2, \ldots, N_{\text{opt}}$ **do**
    Sample transitions from $\mathcal{B}$;
    Optimize the policy using PPO method;
    Optimize the predictor network;
  end for
Output: The trained policy π.

We next test RND method on six hard-exploration Atari games, *i.e.*, the environments have very sparse and even deceptive rewards. The performance comparison is illustrated in Figure 10. RND achieves the best performance on Gravitar, Montezuma's Revenge, and Venture, significantly outperforming PPO on the latter two.

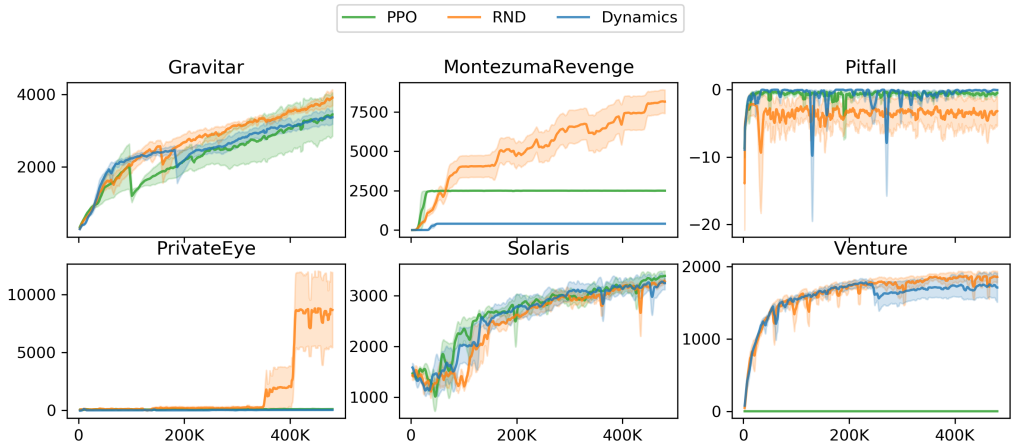

Figure 10: Mean episodic return of RND, dynamics-based exploration method, and PPO.

4.4 Prediction Error-Based Intrinsic Rewards

4.4.1 Exploration with Deep Predictive Models

The second class of exploration methods often learn the dynamic model of the environment, and utilize the prediction error as the intrinsic reward. For instance, [9] proposed a scalable and efficient method for assigning exploration bonuses in large RL problems with complex observations. Let $\sigma(\boldsymbol{s})$ denote the encoding of the state $\boldsymbol{s}$ and $f : \sigma(\mathcal{S}) \times \mathcal{A} \to \sigma(\mathcal{S})$ be a predictor. f takes an encoded state $\boldsymbol{s}$ at time t and the action to predict the encoded successor state $\boldsymbol{s}$. For each transition $(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})$, we have the following prediction error:

$$e(\boldsymbol{s}_t, \boldsymbol{a}_t) = \|\sigma(\boldsymbol{s}_{t+1}) - f(\sigma(\boldsymbol{s}_t), \boldsymbol{a}_t)\|. \tag{50}$$

Then the intrinsic reward is defined as

$$\hat{r}(\boldsymbol{s}_t, \boldsymbol{a}_t) = \frac{e(\boldsymbol{s}_t, \boldsymbol{a}_t)}{t * c}, \tag{51}$$

where $c > 0$ is a decaying constant. If we can accurately model the dynamics of a particular state-action pair, it means we have fully comprehended the state and its novelty is lower. On the contrary, if large prediction error is produced, it means more knowledge about that particular area is needed and hence a higher exploration bonus will be assigned.

4.4.2 Intrinsic Curiosity Module

Performing predictions in the raw sensory space like pixels may produce poor behavior and assign same exploration bonuses to all the states. In [9], the encoding function of state space is learned via an auto-encoder. In contrast, [13] proposed an intrinsic curiosity module (ICM) that learns the state feature space using self-supervision method. ICM suggests that an appropriate state feature space needs to exclude the factors that cannot affect the behavior of the agent. Thus it is unnecessary for the agent to learn them. To that end, ICM designs an inverse dynamic model to predict the action of the agent based on its current and next states. The learned feature space only focuses on the changes related to the actions of the agent and ignores the rest. Then this feature space is used to train a forward dynamics model that predicts the feature representation of the next state. The overview of ICM is illustrated in Figure 11.

Denote by I and F the inverse model and forward model, and let ϕ be the embedding network. The inverse model is optimized by minimizing the following loss function:

$$L_I = D_I(\hat{\boldsymbol{a}}_t, \boldsymbol{a}_t), \tag{52}$$

where $\hat{\boldsymbol{a}}_t = I\big(\phi(\boldsymbol{s}_t), \phi(\boldsymbol{s}_{t+1})\big)$, and D_I is some measures that evaluate the discrepancy between the predicted and actual actions. For discrete action space, D_I is a softmax distribution across all

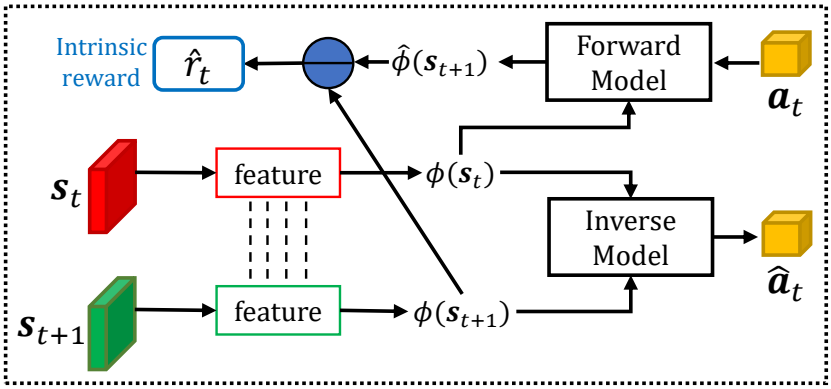

Figure 11: The overview of ICM, where $\ominus$ denotes the Euclidean distance.

possible actions. For the forward model, it is trained using a mean squared error (MSE) function:

$$L_F = \frac{1}{2}\|\hat{\phi}(\boldsymbol{s}_{t+1}) - \phi(\boldsymbol{s}_{t+1})\|_2^2, \tag{53}$$

where $\hat{\phi}(\boldsymbol{s}_{t+1}) = F\big(\phi(\boldsymbol{s}_t), \boldsymbol{a}_t\big)$. Finally, the intrinsic reward is defined as

$$\hat{r}_t = \frac{\eta}{2}\|\hat{\phi}(\boldsymbol{s}_{t+1}) - \phi(\boldsymbol{s}_{t+1})\|_2^2, \tag{54}$$

where $\eta > 0$ is a scaling factor. Since there is no incentive for this feature space to encode any environmental features that are not affected by the actions of agent, it will receive no rewards for reaching inherently unpredictable states. Thus the agent will be robust to the variation in the environment, such as changes in illumination or presence of distractor objects.

4.4.3 Generative Intrinsic Reward Module

ICM trains the reward module using supervised learning, which can only provide deterministic rewards and suffers from overfitting. To address this problem, [45] proposed a generative intrinsic reward module (GIRM) based on variational auto-encoder (VAE), which empowers the generalization of intrinsic rewards on unseen state-action pairs [46]. As illustrated in Figure 12, GIRM consists of two neural network-based components, namely a recognition network (*i.e.* the probabilistic *encoder*) and a generative network (*i.e.* the probabilistic *decoder*).

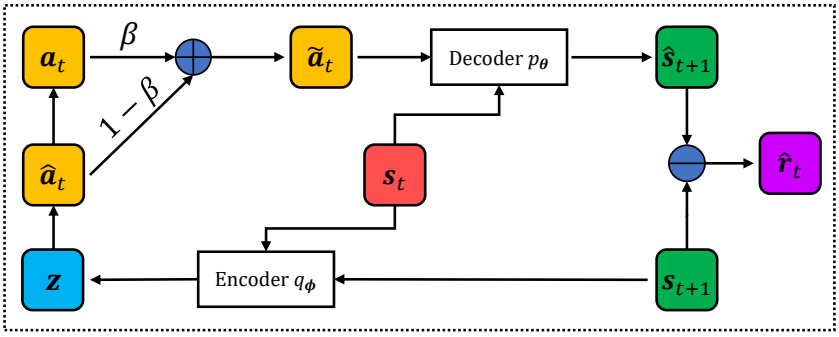

Figure 12: The overview of GIRM, where $\ominus$ denotes the Euclidean distance and $\oplus$ denotes the weighted summation operation.

The encoder $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{s}_t, \boldsymbol{s}_{t+1})$ takes the states $\boldsymbol{s}_t$ and $\boldsymbol{s}_{t+1}$ to perform backward action encoding, and we have

$$\hat{\boldsymbol{a}}_t = \mathrm{Softmax}(\boldsymbol{z}). \tag{55}$$

We then derive an intermediate action data $\tilde{\boldsymbol{a}}_t$ by

$$\tilde{\boldsymbol{a}}_t = \beta \cdot \boldsymbol{a}_t + (1-\beta) \cdot \hat{\boldsymbol{a}}_t, \tag{56}$$

where $\beta \in (0, 1]$ is a weighting coefficient. After that, the decoder $p_{\boldsymbol{\theta}}(\boldsymbol{s}_{t+1}|\boldsymbol{z}, \boldsymbol{s}_t)$ performs forward state transition by taking the action $\tilde{\boldsymbol{a}}_t$ and reconstruct the successor state. The GIRM is trained via optimizing the following evidence lower bound:

$$\begin{aligned} L(\boldsymbol{s}_t, \boldsymbol{s}_{t+1}; \boldsymbol{\theta}, \boldsymbol{\phi}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{s}_t, \boldsymbol{s}_{t+1})} \left[\log p_{\boldsymbol{\theta}}(\boldsymbol{s}_{t+1}|\boldsymbol{z}, \boldsymbol{s}_t)\right] \\ + D_{\mathrm{KL}}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{s}_t, \boldsymbol{s}_{t+1}) \| p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{s}_t)), \end{aligned} \tag{57}$$

where D_{KL} is the Kullback–Leibler divergence. Finally, the intrinsic reward is defined as

$$\hat{r}_t = \lambda \|\hat{\boldsymbol{s}}_{t+1} - \boldsymbol{s}_{t+1}\|_2^2, \tag{58}$$

where λ is a positive scaling weight. In particular, GIRM can generate a family of reward functions by adjusting the mean and variance of the Gaussian distribution of the encoder.

4.5 Vanishing Intrinsic Rewards

So far we have introduced two representative classes of intrinsic rewards. However, both of them suffer from the vanishing intrinsic rewards, *i.e.*, the intrinsic rewards decrease with visits. The agent will have no additional motivation to explore the environment further once the intrinsic rewards decay to zero. To maintain exploration across episodes, [47] proposed a never-give-up (NGU) framework that learns mixed intrinsic rewards composed of episodic and life-long state novelty. As illustrated in Figure 13, the episodic novelty module has an episodic memory and an embedding function f. At the beginning of each episode, the episodic memory is erased completely. At each time step, it computes an episodic intrinsic reward r_t^{episodic} and inserts the embedded state into the memory $\mathcal{E}$.

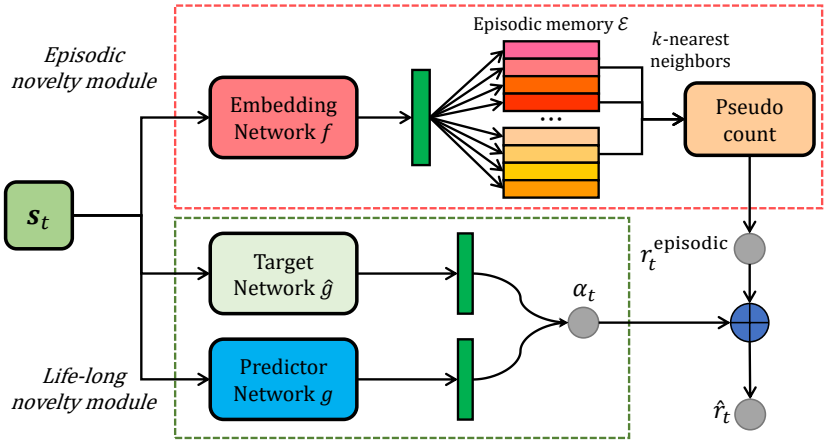

Figure 13: The overview of NGU, where $\oplus$ denotes the weighted summation operation.

Consider the episodic memory until time step t:

$$\mathcal{E} = \{\boldsymbol{e}_0, \boldsymbol{e}_1, \ldots, \boldsymbol{e}_t\}, \tag{59}$$

where $\boldsymbol{e}_t = f(\boldsymbol{s}_t)$. The episodic intrinsic reward is defined as

$$r_t^{\text{episodic}} = \frac{1}{\sqrt{N(\boldsymbol{e}_t)}} \approx \frac{1}{\sqrt{\sum_{\boldsymbol{e}_i \in N_k} K(\boldsymbol{e}_t, \boldsymbol{e}_i)} + c}, \tag{60}$$

where $N(\boldsymbol{e}_t)$ is the counts of visits of $\boldsymbol{e}_t$, $N_k = \{\boldsymbol{e}_i\}_{i=1}^k$ is the k-nearest neighbors of $\boldsymbol{e}_t$, K is a kernel function, and c guarantees a minimum amount of pseudo-count. The episodic intrinsic rewards encourage the agent to visit as many distinct states as possible within one episode. Since it ignores the novelty interactions across episodes, a state will also be treated as a novel state in the current episode even it has been visited many times.

The life-long novelty module captures the long-term state novelty via RND to control the amount of exploration across episodes. It is realized by multiplicatively modulating the exploration bonus

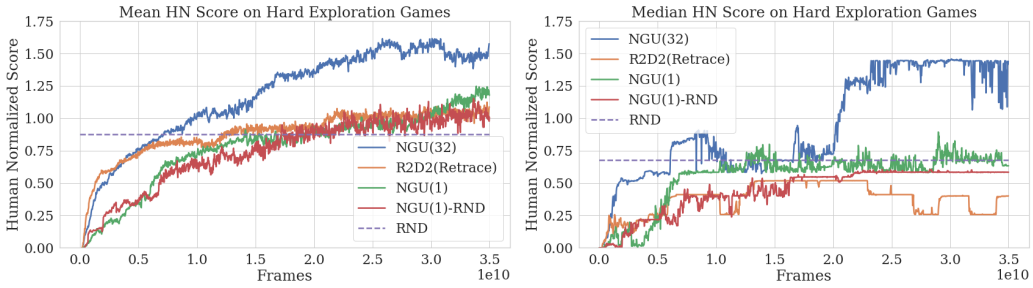

Figure 14: Performance comparison on the 6 hard-exploration games..

r_t^{episodic} with a life-long curiosity factor α_t. This factor will vanish over time and reduce the method to using non-shaped rewards. Finally, we define the mixed intrinsic reward as follows:

$$\hat{r}_t = r_t^{\text{episodic}} \cdot \min\{\max\{\alpha_t, 1\}, \beta\}, \tag{61}$$

where β is a maximum reward threshold.

Figure 14 illustrates the performance comparison on six hard-exploration games. NGU achieves on similar or higher average return than the baselines like RND on all hard-exploration tasks. In particular, NGU is the first method that obtains a positive score on *Pitfall!* without using privileged information.

4.6 Noisy-TV Problem

Noisy-TV problem is also a critical challenge of the intrinsic reward methods, especially for the prediction error-based approaches [14, 28]. Consider a RL agent walking in a complex maze, and it is rewarded by seeking novel experience. As shown in Figure 15, a television with unpredictable random noise will keep producing high prediction error and attract the attention of the agent. Since the agent will consistently receive exploration bonuses from the noisy television, it will make no further progress and just standstill.

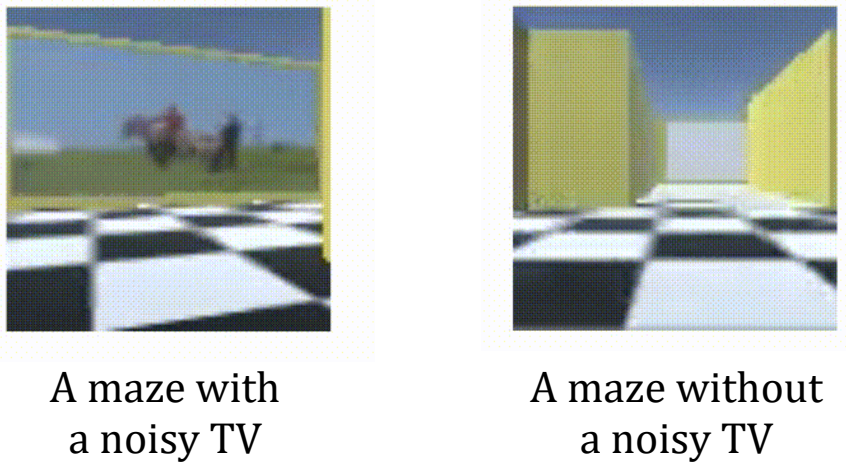

Figure 15: The noisy television will keep playing different pictures.

To address the problem, [28] proposed an episodic curiosity module that utilizes reachability between states as novelty bonus. In contrast, [48] propsoed a more straightforward framework entitled rewarding-impact-driven-exploration (RIDE). In particular, RIDE considers the procedurally-generated environments, in which the environment is constructed differently in every episode. Therefore, learning in such environments requires the agent to perform exploration more actively.

As shown in Figure 16, RIDE inherits the inverse-forward pattern of ICM, in which two dynamic models are leveraged to reconstruct the transition process. RIDE defines the intrinsic reward as the

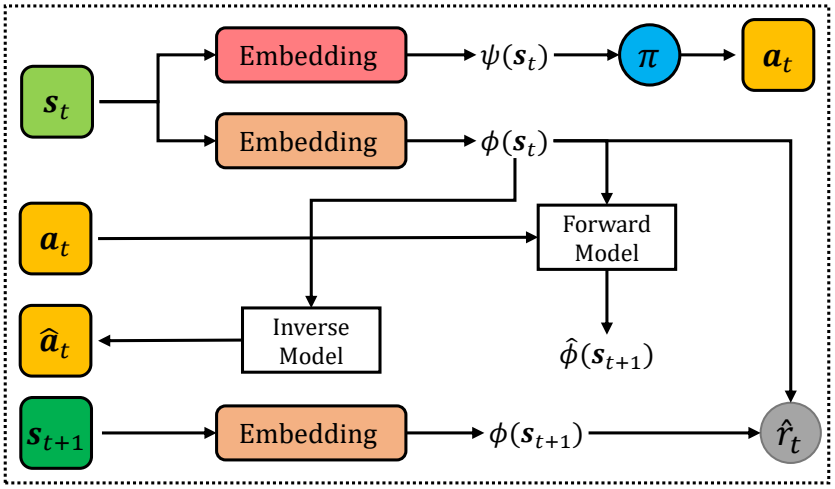

Figure 16: The overview of RIDE.

difference between the consecutive embedded states:

$$\hat{r}_t = \frac{\|\phi(\boldsymbol{s}_{t+1}) - \phi(\boldsymbol{s}_t)\|_2}{\sqrt{N_{\text{ep}}(\boldsymbol{s}_{t+1})}}, \tag{62}$$

where $\phi(\cdot)$ is the embedding network and N_{ep} is the episodic visitation counts. N_{ep} is used to discount the intrinsic reward, which guarantees that the agent will not linger between a sequence of states with a large difference in their embeddings. Through this intrinsic reward, the agent is encouraged to take actions that results in large state change to improve the exploration. In particular, RIDE can also overcome the problem of vanishing intrinsic rewards.

5 Rényi State Entropy Maximization

In Chapter 4, we discussed various intrinsically-motivated methods that utilize the reward shaping to improve the exploration of state space. For instance, NGU leverages the mixed state novelty to encourage the agent to visit as many distinct states as possible. NGU overcomes the problem of the vanishing intrinsic rewards and provides sustainable exploration incentives. However, methods like NGU and RIDE pay excessive attention to specific states while failing to reflect the global exploration completeness. Furthermore, they suffer from high computational complexity and performance loss incurred by abundant auxiliary models. In this chapter, we will first propose to reformulate the exploration problem before establishing a more efficient exploration method.

5.1 Coupon Collector's Problem

In probability theory, the coupon collector's problem (CCP) is described as the contest of collecting all coupons and winning [49]. Given n coupons, how many coupons do you need to draw with replacement before having drawn each coupon at least once? For instance, it takes about 172 trials on average to collect about all coupons when $n = 40$. To guarantee the completeness of exploration, the agent is expected to visit all possible states during training. Such an objective can be also considered as a CCP conditioned upon a nonuniform probability distribution, in which the agent is the collector and the states are the coupons. Denote by $d^\pi(\boldsymbol{s})$ the state distribution induced by the policy π. Assuming that the agent takes $\tilde{T}$ environment steps to finish the collection, we can compute the expectation of $\tilde{T}$ as

$$\mathbb{E}_\pi(\tilde{T}) = \int_0^\infty \left(1 - \prod_{i=1}^{|\mathcal{S}|}(1 - e^{-d^\pi(\boldsymbol{s}_i)t})\right) dt, \tag{63}$$

where $|\cdot|$ stands for the cardinality of the enclosed set $\mathcal{S}$.

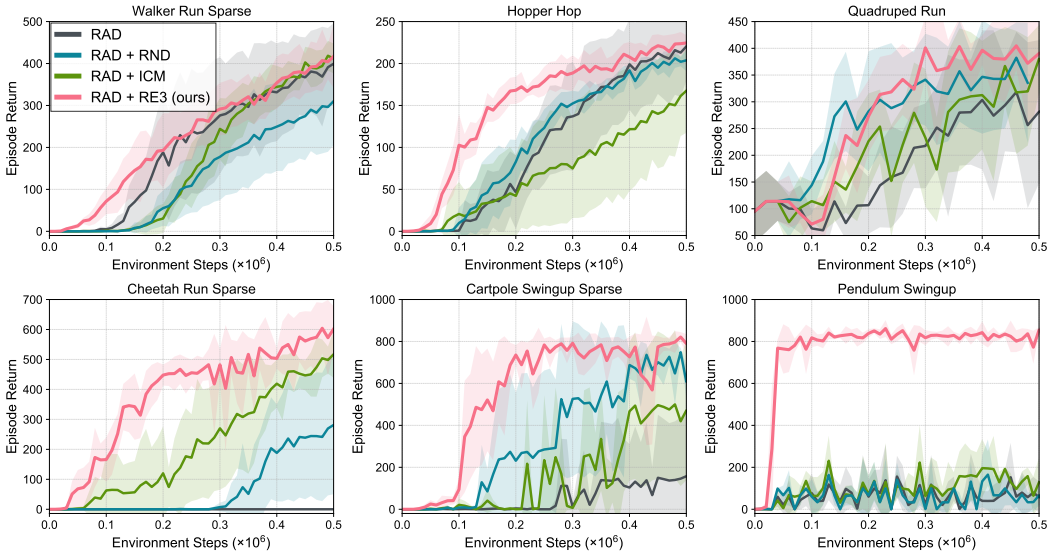

Figure 17: Performance on locomotion tasks from DeepMind control suite.

For simplicity of notation, we omit the superscript in $d^\pi(\boldsymbol{s})$ in the sequel. Efficient exploration aims to find a policy that optimizes $\min_{\pi\in\Pi} \mathbb{E}_\pi(\tilde{T})$. However, it is non-trivial to evaluate Eq. (63) due to the improper integral, not to mention solving the optimization problem. To address the problem, it is common to leverage the Shannon entropy to make a tractable objective function, which is defined as

$$H(d) = -\mathbb{E}_{\boldsymbol{s}\sim d(\boldsymbol{s})}\big[\log d(\boldsymbol{s})\big]. \tag{64}$$

5.2 Random Encoders for Efficient Exploration

To estimate the state entropy, [50] introduced a k-nearest neighbor entropy estimator [51] and proposed a random-encoders-for-efficient-exploration (RE3) method. Given a trajectory $\tau = (\boldsymbol{s}_0, \boldsymbol{a}_0, \ldots, \boldsymbol{a}_{T-1}, \boldsymbol{s}_T)$, RE3 first uses a randomly initialized DNN to encode the visited states. Denote by $\{\boldsymbol{x}_i\}_{i=0}^{T-1}$ the encoding vectors of observations, RE3 estimates the entropy of state distribution $d(\boldsymbol{s})$ using a k-nearest neighbor entropy estimator [51]:

$$\begin{aligned}\hat{H}_T^k(d) &= \frac{1}{T}\sum_{i=0}^{T-1}\log\frac{T\cdot\|\boldsymbol{x}_i-\tilde{\boldsymbol{x}}_i\|_2^m\cdot\pi}{k\cdot\Gamma(\frac{m}{2}+1)} + \log k - \Psi(k)\\ &\propto \frac{1}{T}\sum_{i=0}^{T-1}\log\|\boldsymbol{x}_i-\tilde{\boldsymbol{x}}_i\|_2,\end{aligned} \tag{65}$$

where $\tilde{\boldsymbol{x}}_i$ is the k-nearest neighbor of $\boldsymbol{x}_i$ within the set $\{\boldsymbol{x}_i\}_{i=0}^{T-1}$, m is the dimension of the encoding vectors, and $\Gamma(\cdot)$ is the Gamma function, and $\Psi(\cdot)$ is the digamma function. Note that π in Eq. (65) denotes the ratio between the circumference of a circle to its diameter. Equipped with Eq. (65), the total reward for each transition $(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1})$ is computed as:

$$r^{\text{total}} = r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \lambda_t\cdot\log(\|\boldsymbol{x}_t-\tilde{\boldsymbol{x}}_t\|_2+1), \tag{66}$$

where $\lambda_t = \lambda_0(1-\kappa)^t, \lambda_t \geq 0$ is a weight coefficient that decays over time, κ is a decay rate.

In sharp contrast to the methods of Chapter 4, RE3 does not need abundant auxiliary models or storages to record and analyze the learning procedure. Thus it is very efficient and can be easily employed in arbitrary tasks. We next tested the RE3 on locomotion tasks from DeepMind control suite. As illustrated in Figure 17, RE3 significantly outperforms other exploration methods in terms of sample-efficiency. Despite the advantages of RE3, the Shannon entropy-based objective may lead to a policy that visits some states with a vanishing probability. In the following section, we will employ a representative example to demonstrate the practical drawbacks of Eq. (64) and introduce the Rényi entropy to address the problem.

5.3 Rényi State Entropy Maximization

5.3.1 Rényi State Entropy

We first formally define the Rényi entropy as follows:

Definition 3 (Rényi Entropy)**.** *Let $X \in \mathbb{R}^m$ be a random vector that has a density function $f(\boldsymbol{x})$ with respect to Lebesgue measure on $\mathbb{R}^m$, and let $\mathcal{X} = \{\boldsymbol{x} \in \mathbb{R}^m : f(\boldsymbol{x}) > 0\}$ be the support of the distribution. The Rényi entropy of order $\alpha \in (0,1) \cup (1, +\infty)$ is defined as [52]:*

$$H_\alpha(f) = \frac{1}{1-\alpha} \log \int_{\mathcal{X}} f^\alpha(\boldsymbol{x}) d\boldsymbol{x}. \tag{67}$$

Using Definition 3, we propose the following Rényi state entropy (RISE):

$$H_\alpha(d) = \frac{1}{1-\alpha} \log \int_{\mathcal{S}} d^\alpha(\boldsymbol{s}) d\boldsymbol{s}. \tag{68}$$

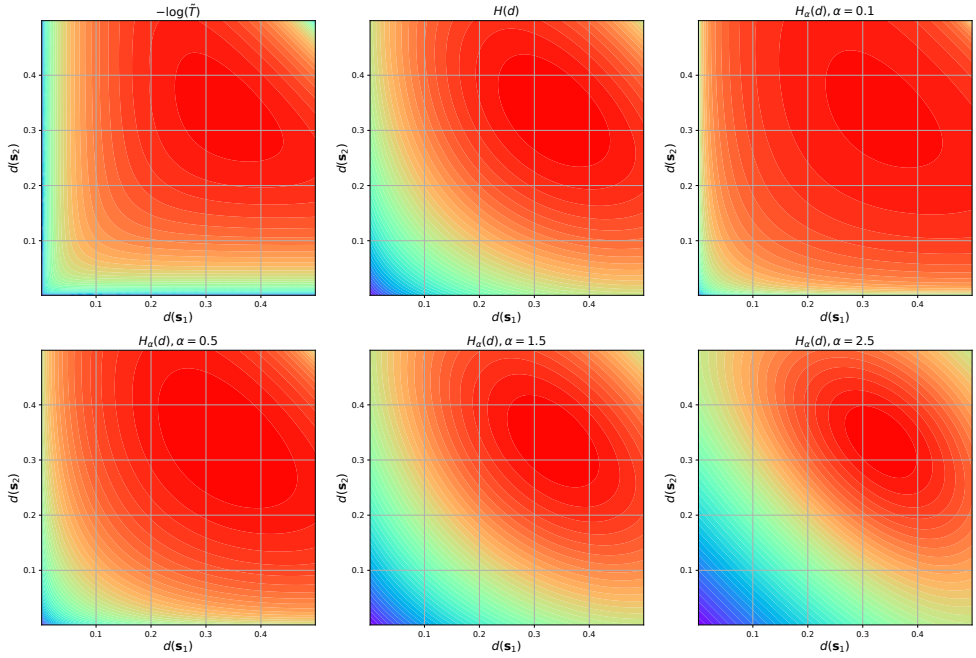

Figure 18: The contours of different objective functions when $|\mathcal{S}| = 3$.

Fig. 18 use a toy example to visualize the contours of different objective functions when an agent learns from an environment characterized by only three states. As shown in Fig. 18, $-\log(\tilde{T})$ decreases rapidly when any state probability approaches zero, which prevents the agent from visiting a state with a vanishing probability while encouraging the agent to explore the infrequently-seen states. In contrast, the Shannon entropy remains relatively large as the state probability approaches zero. Interestingly, Fig. 18 shows that this problem can be alleviated by the Rényi entropy as it better matches $-\log(\tilde{T})$. The Shannon entropy is far less aggressive in penalizing small probabilities, while the Rényi entropy provides more flexible exploration intensity.

5.3.2 Theoretical Analysis

To maximize $H_\alpha(d)$, we consider using a maximum entropy policy computation (MEPC) algorithm proposed by [53], which uses the following two oracles:

Definition 4 (Approximating planning oracle)**.** *Given a reward function $r : \mathcal{S} \to \mathbb{R}$ and a gap ϵ, the planning oracle returns a policy by $\pi = O_{\mathrm{AP}}(r, \epsilon_1)$, such that*

$$V^\pi \geq \max_{\pi \in \Pi} V^\pi - \epsilon, \tag{69}$$

where V^π is the state-value function.

Definition 5 (State distribution estimation oracle)**.** *Given a gap ϵ and a policy π, this oracle estimates the state distribution by $\hat{d} = O_{\mathrm{DE}}(\pi, \epsilon)$, such that*

$$\|d - \hat{d}\|_\infty \leq \epsilon. \tag{70}$$

Given a set of stationary policies $\hat{\Pi} = \{\pi_0, \pi_1, \dots\}$, we define a mixed policy as $\pi_{\mathrm{mix}} = (\boldsymbol{\omega}, \hat{\Pi})$, where ω contains the weighting coefficients. Then the induced state distribution is

$$d^{\pi_{\mathrm{mix}}} = \sum_i \boldsymbol{\omega}_i d^{\pi_i}(\boldsymbol{s}). \tag{71}$$

Finally, the workflow of MEPC is summarized in Algorithm 6.

Algorithm 6 MEPC

1: Set the number of iterations T, step size η, planning oracle error tolerance $\epsilon_1 > 0$, and state distribution oracle error tolerance $\epsilon_2 > 0$;
2: Initialize $\hat{\Pi}_0 = \{\pi_0\}$, where π_0 is an arbitrary policy;
3: Initialize $\omega_0 = 1$;
4: **for** $t = 0, \dots, T-1$ **do**
5: Invoke the state distribution oracle on $\pi_{\mathrm{mix},t} = (\omega, \hat{\Pi}_t)$:

$$\hat{d}^{\pi_{\mathrm{mix},t}} = O_{\mathrm{AP}}(r, \epsilon_1);$$

6: Define the reward function r_t as

$$r_t(\boldsymbol{s}) = \nabla H_\alpha(\hat{d}^{\pi_{\mathrm{mix},t}});$$

7: Approximate the optimal policy on r_t:

$$\pi_{t+1} = O_{\mathrm{DE}}(\pi, \epsilon_2);$$

8: Update $\pi_{\mathrm{mix},t} = (\omega_{t+1}, \hat{\Pi})$

$$\begin{aligned} \hat{\Pi}_{t+1} &= (\pi_0, \dots, \pi_t, \pi_{t+1}) \\ \omega_{t+1} &= ((1-\eta)\omega_t, \eta)\,; \end{aligned}$$

9: **end for**
10: Return $\pi_{\mathrm{mix},T} = (\omega_T, \hat{\Pi}_T)$.

Consider the discrete case of Rényi state entropy and set $\alpha \in (0, 1)$, we have

$$H_\alpha(d) = \frac{1}{1-\alpha} \log \sum_{\boldsymbol{s} \in \mathcal{S}} d^\alpha(\boldsymbol{s}). \tag{72}$$

Since the logarithmic functions are monotonically increasing functions, the maximization of $\tilde{H}_\alpha(d)$ can be achieved by maximizing the following function:

$$\tilde{\mathcal{H}}_\alpha(d) = \frac{1}{1-\alpha} \sum_{\boldsymbol{s} \in \mathcal{S}} d^\alpha(\boldsymbol{s}). \tag{73}$$

As $\tilde{\mathcal{H}}_\alpha(d)$ is not smooth, we propose to replace $\tilde{\mathcal{H}}_\alpha(d)$ with a smoothed $\tilde{H}_{\alpha,\sigma}(d)$ defined as

$$\tilde{H}_{\alpha,\sigma}(d) = \frac{1}{1-\alpha} \sum_{\boldsymbol{s} \in \mathcal{S}} (d(\boldsymbol{s}) + \sigma)^\alpha, \tag{74}$$

where $\sigma > 0$.

Lemma 1. *$\tilde{H}_{\alpha,\sigma}(d)$ is β-smooth, such that*

$$\|\nabla \tilde{H}_{\alpha,\sigma}(d) - \nabla \tilde{H}_{\alpha,\sigma}(d')\|_\infty \leq \beta \|d - d'\|_\infty, \tag{75}$$

where $\beta = \alpha \sigma^{\alpha-2}$.

Proof. Since $\nabla^2 \tilde{H}_{\alpha,\sigma}(d) = -\alpha(d(\boldsymbol{s}) + \sigma)^{\alpha-2}$ is a diagonal matrix, we have

$$\begin{aligned} &\|\nabla \tilde{H}_{\alpha,\sigma}(d) - \nabla \tilde{H}_{\alpha,\sigma}(d')\|_\infty \\ &\leq \max_{\varsigma \in [0,1]} |\nabla^2 \tilde{H}_{\alpha,\sigma}(\varsigma d + (1-\varsigma)d')| \cdot \|d - d'\|_\infty \\ &\leq \alpha \sigma^{\alpha-2} \|d - d'\|_\infty, \end{aligned} \tag{76}$$

where the first inequality follows the Taylor's theorem. This concludes the proof. □

From Lemma 1, the following theorem can be derived.

Theorem 5. *For any $\epsilon > 0$ with $\epsilon_1 = 0.1\epsilon, \epsilon_2 = 0.1\beta^{-1}\epsilon$ and $\eta = 0.1\beta^{-1}\epsilon$, the following inequality holds*

$$\tilde{H}_{\alpha,\sigma}(d^{\pi_{\mathrm{mix},T}}) \geq \max_{\pi \in \Pi} \tilde{H}_{\alpha,\sigma}(d^\pi) - \epsilon, \tag{77}$$

if Algorithm 6 is run for T iterations with

$$T \geq \frac{10\alpha\sigma^{\alpha-2}}{\epsilon} \log \frac{10\alpha\sigma^{\alpha-1}}{(1-\alpha)\epsilon}. \tag{78}$$

Proof. Equipped with Lemma 1, let $\pi^* = \operatorname*{argmax}_{\pi \in \Pi} \tilde{H}_{\alpha,\sigma}(d)$, we have (proved by [53]):

$$\begin{aligned} &\tilde{H}_{\alpha,\sigma}(d^{\pi^*}) - \tilde{H}_{\alpha,\sigma}(d^{\pi_{\mathrm{mix},T}}) \\ &\leq B \exp(-T\eta) + 2\beta\epsilon_2 + \epsilon_1 + \eta\beta, \end{aligned} \tag{79}$$

where $\|\nabla \tilde{H}_{\alpha,\sigma}(d)\|_\infty \leq B = \frac{\alpha}{1-\alpha}\sigma^{\alpha-1}$. Thus it suffices to set $\epsilon_1 = 0.1\epsilon, \epsilon_2 = 0.1\beta^{-1}\epsilon, \eta = 0.1\beta^{-1}\epsilon, T = \eta^{-1} \log 10B\epsilon^{-1}$. When Algorithm 6 is run for

$$T \geq 10\beta\epsilon^{-1} \log 10B\epsilon^{-1}, \tag{80}$$

it holds

$$\tilde{H}_{\alpha,\sigma}(d^{\pi_{\mathrm{mix},T}}) \geq \max_{\pi \in \Pi} \tilde{H}_{\alpha,\sigma}(d^\pi) - \epsilon. \tag{81}$$

This concludes the proof. □

Theorem 5 demonstrates that the proposed Rényi state entropy can be practically maximized with computational complexity given in Theorem 5 by exploiting MEPC. Inspection of Eq. (78) reveals that the lower bound of T is a function of α. In other words, we can expedite the exploration process through reducing the lower bound of T by adjusting the value of α. This observation is consistent with the analytical results reported in [52].

5.3.3 Fast Entropy Estimation

However, it is non-trivial to apply MEPC when handling complex environments with high-dimensional observations. To address the problem, we propose to utilize the following k-nearest neighbor estimator to realize efficient estimation of the Rényi entropy [54]. Note that π in Eq. (82) denotes the ratio between the circumference of a circle to its diameter.

Theorem 6 (Estimator)**.** *Denote by $\{X_i\}_{i=1}^N$ a set of independent random vectors from the distribution X. For $k < N, k \in \mathbb{N}$, $\tilde{X}_i$ stands for the k-nearest neighbor of X_i among the set. We estimate the Rényi entropy using the sample mean as follows:*

$$\hat{H}_N^{k,\alpha}(f) = \frac{1}{N} \sum_{i=1}^{N} \left[(N-1) V_m C_k \|X_i - \tilde{X}_i\|^m \right]^{1-\alpha}, \tag{82}$$

where $C_k = \left[\frac{\Gamma(k)}{\Gamma(k+1-\alpha)} \right]^{\frac{1}{1-\alpha}}$ and $V_m = \frac{\pi^{\frac{m}{2}}}{\Gamma(\frac{m}{2}+1)}$ is the volume of the unit ball in $\mathbb{R}^m$, and $\Gamma(\cdot)$ is the Gamma function, respectively. Moreover, it holds

$$\lim_{N \to \infty} \hat{H}_N^{k,\alpha}(f) = H_\alpha(f). \tag{83}$$

Proof. See proof in [54]. □

Given a trajectory $\tau = \{\boldsymbol{s}_0, \boldsymbol{a}_0, \ldots, \boldsymbol{a}_{T-1}, \boldsymbol{s}_T\}$ collected by the agent, we approximate the Rényi state entropy in Eq. (67) using Eq. (82) as

$$\begin{aligned} \hat{H}_T^{k,\alpha}(d) &= \frac{1}{T}\sum_{i=0}^{T-1}\left[(T-1)V_m C_k \|\boldsymbol{y}_i - \tilde{\boldsymbol{y}}_i\|^m\right]^{1-\alpha}, \\ &\propto \frac{1}{T}\sum_{i=0}^{T-1}\|\boldsymbol{y}_i - \tilde{\boldsymbol{y}}_i\|^{1-\alpha}, \end{aligned} \tag{84}$$

where $\boldsymbol{y}_i$ is the encoding vector of $\boldsymbol{s}_i$ and $\tilde{\boldsymbol{y}}_i$ is the k-nearest neighbor of $\boldsymbol{y}_i$. After that, we define the intrinsic reward that takes each transition as a particle:

$$\hat{r}(\boldsymbol{s}_i) = \|\boldsymbol{y}_i - \tilde{\boldsymbol{y}}_i\|^{1-\alpha}, \tag{85}$$

where $\hat{r}(\cdot)$ is used to distinguish the intrinsic reward from the extrinsic reward $r(\cdot)$. Eq. (85) indicates that the agent needs to visit as more distinct states as possible to obtain higher intrinsic rewards.

Such an estimation method requires no additional auxiliary models, which significantly promotes the learning efficiency. Equipped with the intrinsic reward, the total reward of each transition $(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})$ is computed as

$$r_t^{\text{total}} = r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \lambda_t \cdot \hat{r}(\boldsymbol{s}_t) + \zeta \cdot H(\pi(\cdot|\boldsymbol{s}_t)), \tag{86}$$

where $H(\pi(\cdot|\boldsymbol{s}_t))$ is the action entropy regularizer for improving the exploration on action space, $\lambda_t = \lambda_0(1-\kappa)^t$ and ζ are two non-negative weight coefficients, and κ is a decay rate.

5.4 Robust Representation Learning

While the Rényi state entropy encourages exploration in high-dimensional observation spaces, several implementation issues have to be addressed in its practical deployment. First of all, observations have to be encoded into low-dimensional vectors in calculating the intrinsic reward. While a randomly initialized neural network can be utilized as the encoder as proposed in [50], it cannot handle more complex and dynamic tasks, which inevitably incurs performance loss. Moreover, since it is less computationally expensive to train an encoder than RL, we propose to leverage the VAE to realize efficient and robust embedding operation, which is a powerful generative model based on the Bayesian inference [46]. As shown in Fig. 19, a standard VAE is composed of a recognition model and a generative model. These two models represent a probabilistic *encoder* and a probabilistic *decoder*, respectively.

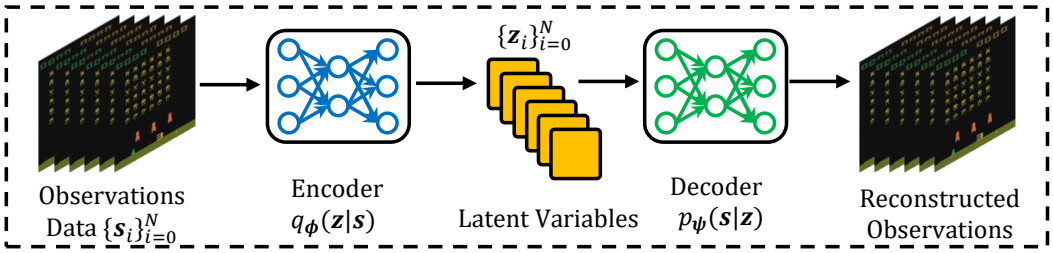

Figure 19: VAE model for embedding observations.

We denote by $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{s})$ the recognition model represented by a neural network with parameters $\boldsymbol{\phi}$. The recognition model accepts an observation input before encoding the input into latent variables. Similarly, we represent the generative model as $p_{\boldsymbol{\psi}}(\boldsymbol{s}|\boldsymbol{z})$ using a neural network with parameters $\boldsymbol{\psi}$, accepting the latent variables and reconstructing the observation. Given a trajectory $\tau = \{\boldsymbol{s}_0, \boldsymbol{a}_0, \ldots, \boldsymbol{a}_{T-1}, \boldsymbol{s}_T\}$, the VAE model is trained by minimizing the following loss function:

$$\begin{aligned} L(\boldsymbol{s}_t; \boldsymbol{\phi}, \boldsymbol{\psi}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{s}_t)}\left[\log p_{\boldsymbol{\psi}}(\boldsymbol{s}_t|\boldsymbol{z})\right] \\ - D_{\text{KL}}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{s}_t)\|p_{\boldsymbol{\psi}}(\boldsymbol{z})\right), \end{aligned} \tag{87}$$

where $t = 0, \ldots, T$, $D_{\mathrm{KL}}(\cdot)$ is the Kullback-Liebler (KL) divergence.

Next, we will elaborate on the design of the k value to improve the estimation accuracy of the state entropy. [51] investigated the performance of this entropy estimator for some specific probability distribution functions such as uniform distribution and Gaussian distribution. Their simulation results demonstrated that the estimation accuracy first increased before decreasing as the k value increases. To circumvent this problem, we propose our k-value searching scheme as shown in Fig. 20. We first divide the observation dataset into K subsets before the encoder encodes the data into low-dimensional embedding vectors. Assuming that all the data samples are independent and identically distributed, an appropriate k value should produce comparable results on different subsets. By exploiting this intuition, we propose to search the optimal k value that minimizes the min-max ratio of entropy estimation set. Denote by $\pi_{\boldsymbol{\theta}}$ the policy network, the detailed searching algorithm is summarized in Algorithm 7.

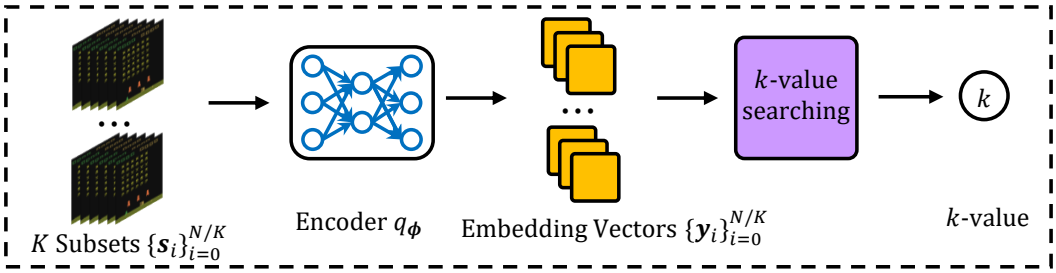

Figure 20: k-value searching.

Algorithm 7 k-value searching method

1: Initialize a policy network $\pi_{\boldsymbol{\theta}}$;
2: Initialize the number of sample steps N, the threshold $k_{\max}$ of k, a null array δ with length $k_{\max}$, and the number of subsets K;
3: Execute policy $\pi_{\boldsymbol{\theta}}$ and collect the trajectory $\tau = \{\boldsymbol{s}_0, \boldsymbol{a}_0, \ldots, \boldsymbol{a}_{N-1}, \boldsymbol{s}_N\}$;
4: Divide the observations dataset $\{\boldsymbol{s}_i\}_{i=0}^{N}$ into K subsets randomly;
5: **for** $k = 1, 2, \ldots, k_{\max}$ **do**
6: Calculate the estimated entropy on K subsets using Eq. (82):

$$\hat{\boldsymbol{H}}_k = (\hat{H}_{k,N/K,\alpha}[1], \ldots, \hat{H}_{k,N/K,\alpha}[K]) \tag{88}$$

7: Calculate the min-max ratio $\delta(\hat{\boldsymbol{H}}_k)$ and $\boldsymbol{\delta}[k] \leftarrow \delta(\hat{\boldsymbol{H}}_k)$;
8: **end for**
9: Output $k = \underset{k}{\operatorname{argmin}}\, \boldsymbol{\delta}[k]$.

Finally, we are ready to propose our RISE framework by exploiting the optimal k value derived above. As shown in Fig. 21, the proposed RISE framework first encodes the high-dimensional observation data into low-dimensional embedding vectors through $q : \mathcal{S} \to \mathbb{R}^m$. After that, the Euclidean distance between $\boldsymbol{y}_t$ and its k-nearest neighbor is computed as the intrinsic reward. Algorithm 8 and Algorithm 9 summarize the on-policy and off-policy RL versions of the proposed RISE, respectively. In the off-policy version, the entropy estimation is performed on the sampled transitions in each step. As a result, a larger batch size can improve the estimation accuracy. It is worth pointing out that RISE can be straightforwardly integrated into any existing RL algorithms such as Q-learning and soft actor-critic, providing high-quality intrinsic rewards for improved exploration.

5.5 Experiments

In this section, we will evaluate our RISE framework on both the tabular setting and environments with high-dimensional observations. We compare RISE against two representative intrinsic reward-based methods, namely RE3 and MaxRényi [52]. We also train the agent without intrinsic rewards for ablation studies. As for hyper-parameters setting, we only report the values of the best experiment results.

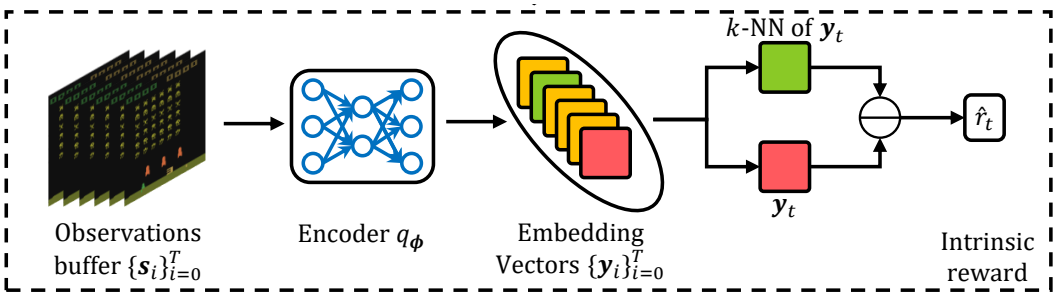

Figure 21: The generation of intrinsic rewards, where k-NN is k-nearest neighbor and $\ominus$ denotes the Euclidean distance.

Algorithm 8 On-policy RL Version of RISE

Phase 1: k-value searching and encoder training
Initialize the policy network $\pi_{\boldsymbol{\theta}}$, encoder $q_{\boldsymbol{\phi}}$ and decoder $p_{\boldsymbol{\psi}}$;
Initialize the number of sample steps N, the threshold $k_{\max}$ of k-value, the embedding size m and the number subsets K ;
Execute policy and collect observations data $\{\boldsymbol{s}_i\}_{i=1}^N$;
Use $\{\boldsymbol{s}_i\}_{i=1}^N$ to train the encoder;
Use Algorithm 7 to select k-value;
Phase 2: Policy update
Initialize the maximum episodes E, order α, coefficients λ_0, ζ and decay rate κ;
for episode $\ell = 1, \ldots, E$ **do**
  Collect the trajectory $\tau_\ell = \{\boldsymbol{s}_0, \boldsymbol{a}_0, \ldots, \boldsymbol{a}_{T-1}, \boldsymbol{s}_T\}$;
  Compute the embedding vectors $\{\boldsymbol{z}_t\}_{t=0}^T$ of $\{\boldsymbol{s}_t\}_{t=0}^T$ using the encoder $q_{\boldsymbol{\phi}}$;
  Compute $\hat{r}(\boldsymbol{s}_t) \leftarrow (\|\boldsymbol{z}_t - \tilde{\boldsymbol{z}}_t\|_2)^{1-\alpha}$;
  Update $\lambda_\ell = \lambda_0(1-\kappa)^\ell$;
  Let $r_t^{\text{total}} = r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \lambda_\ell \cdot \hat{r}(\boldsymbol{s}_t) + \zeta \cdot H(\pi(\cdot|\boldsymbol{s}_t))$;
  Update the policy network with transitions $\{\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}, r_t^{\text{total}}\}_{t=0}^T$ using any on-policy RL algorithms.
end for

5.5.1 Maze Games

In this section, we first leverage a simple but representative example to highlight the effectiveness of the Rényi state entropy-driven exploration. We introduce a grid-based environment Maze2D [55] illustrated in Fig. 22. The agent can move one position at a time in one of the four directions, namely left, right, up, and down. The goal of the agent is to find the shortest path from the start point to the end point. In particular, the agent can teleport from a portal to another identical mark.

Experimental Setting The standard Q-learning (QL) algorithm [6] is selected as the benchmarking method. We perform extensive experiments on three mazes with different sizes. Note that the problem complexity increases exponentially with the maze size. In each episode, the maximum environment step size was set to $10M^2$, where M is the maze size. We initialized the Q-table with zeros and updated the Q-table in every step for efficient training. The update formulation is given by:

$$Q(\boldsymbol{s}, \boldsymbol{a}) \leftarrow Q(\boldsymbol{s}, \boldsymbol{a}) + \eta[r + \gamma \max_{\boldsymbol{a}'} Q(\boldsymbol{s}', \boldsymbol{a}') - Q(\boldsymbol{s}, \boldsymbol{a})], \tag{89}$$

where $Q(\boldsymbol{s}, \boldsymbol{a})$ is the action-value function. The step size was set to 0.2 while a ϵ-greedy policy with an exploration rate of 0.001 was employed.

Performance Comparison

To compare the exploration performance, we choose the minimum number of environment steps taken to visit all states as the key performance indicator (KPI). For instance, a 10×10 maze of 100 grids corresponds to 100 states. The minimum number of steps for the agent to visit all the possible states is evaluated as its exploration performance. As seen in Fig. 23, the proposed *Q-learning+RISE* achieved the best performance in all three maze games. Moreover, RISE with smaller α takes less

Algorithm 9 Off-policy RL Version of RISE

Phase 1 of Algorithm 8;
Phase 2: Policy update
Initialize the maximum environment steps $t_{\max}$, order α, coefficients λ_0, ζ, decay rate κ, and replay buffer $\mathcal{B} \leftarrow \emptyset$;
for step $t = 1, \ldots, t_{\max}$ **do**
  Collect the transition $(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1})$ and let $\mathcal{B} \leftarrow \mathcal{B} \cup \{(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1}, \boldsymbol{z}_t)\}$, where $\boldsymbol{z}_t = q_{\boldsymbol{\theta}}(\boldsymbol{s}_t)$;
  Sample a minibatch $\{(\boldsymbol{s}_i, \boldsymbol{a}_i, r_i, \boldsymbol{s}_{i+1}, \boldsymbol{z}_i)\}_{i=1}^{B}$ from $\mathcal{B}$ randomly;
  Compute $\hat{r}(\boldsymbol{s}_i) \leftarrow (\|\boldsymbol{z}_i - \tilde{\boldsymbol{z}}_i\|_2)^{1-\alpha}$;
  Update $\lambda_t = \lambda_0(1-\kappa)^t$;
  Let $r_i^{\text{total}} = r(\boldsymbol{s}_i, \boldsymbol{a}_i) + \lambda_t \cdot \hat{r}(\boldsymbol{s}_i) + \zeta \cdot H(\pi(\cdot|\boldsymbol{s}_i))$;
  Update the policy network with transitions $\{\boldsymbol{s}_i, \boldsymbol{a}_i, \boldsymbol{s}_{i+1}, r_i^{\text{total}}\}_{i=1}^{B}$ using any off-policy RL algorithms.
end for

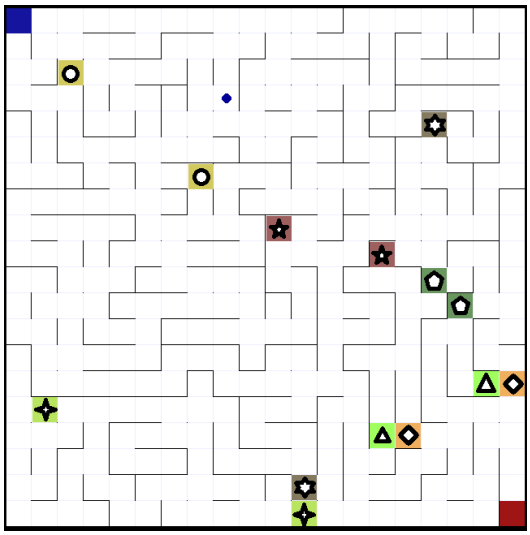

Figure 22: A maze game with grid size 20×20.

steps to finish the exploration phase. This experiment confirmed the great capability of the Rényi state entropy-driven exploration.

5.5.2 Atari Games

Next, we will test RISE on the Atari games with a discrete action space, in which the player aims to achieve more points while remaining alive [56]. To generate the observation of the agent, we stacked four consecutive frames as one input. These frames were cropped to the size of $(84, 84)$ to reduce the required computational complexity.

Experimental Setting To handle the graphic observations, we leveraged convolutional neural networks (CNNs) to build RISE and the benchmarking methods. For fair comparison, the same policy network and value network are employed for all the algorithms, and their architectures can be found in Table 3. For instance, "8×8 Conv. 32" represents a convolutional layer that has 32 filters of size 8×8. A categorical distribution was used to sample an action based on the action probability of the stochastic policy. The VAE block of RISE and MaxRényi need to learn an encoder and a decoder. The encoder is composed of four convolutional layers and one dense layer, in which each convolutional layer is followed by a batch normalization (BN) layer [57]. Note that "Dense 512 & Dense 512" in Table 3 means that there are two branches to output the mean and variance of the latent variables, respectively. For the decoder, it utilizes four deconvolutional layers to perform upsampling while a dense layer and a convolutional layer are employed at the top and the bottom of the decoder.

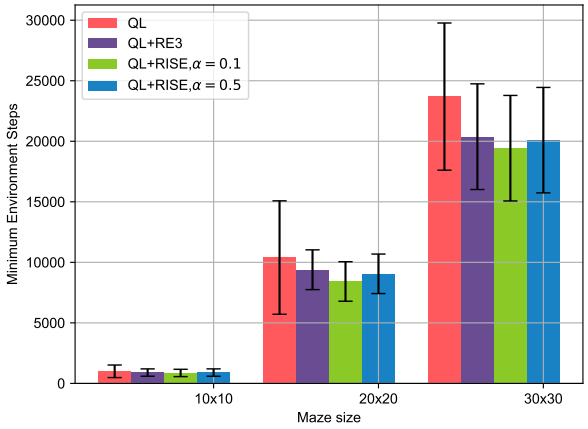

Figure 23: Average exploration performance comparison over 100 simulation runs.

Finally, no BN layer is included in the decoder and the ReLU activation function is employed for all components.

Figure 24: Average episode return versus number of environment steps on Atari games.

Table 1: Performance comparison in nine Atari games.

Game	PPO	PPO+RE3	PPO+MaxRényi	PPO+RISE
Assault	5.24k±1.86k	5.54k±2.12k	5.68k±1.99k	**5.78k±2.44k**
Battle Zone	18.63k±1.96k	20.47k±2.73k	19.17k±3.48k	**21.80k±5.44k**
Demon Attack	13.55k±4.65k	17.40k±9.63k	16.18k±8.19k	**18.39k±7.61k**
Kung Fu Master	21.86k±8.92k	23.21k±5.74k	27.20k±5.59k	**27.76k±5.86k**
Riverraid	7.99k±0.53k	8.14k±0.37k	8.21k±0.36k	**10.07k±0.77k**
Space Invaders	0.72k±0.25k	0.89k±0.33k	0.81k±0.37k	**1.09k±0.21k**

In the first phase, we initialized a policy network $\pi_{\boldsymbol{\theta}}$ and let it interact with eight parallel environments with different random seeds. We first collected observation data over ten thousand environment steps before the VAE encoder generates the latent vectors of dimension of 128 from the observation data. After that, the latent vectors were sent to the decoder to reconstruct the observation tensors. For parameters update, we used an Adam optimizer with a learning rate of 0.005 and a batch size of 256. Finally, we divided the observation dataset into $K = 8$ subsets before searching for the optimal k-value over the range of $[1, 15]$ using Algorithm 7.

Equipped with the learned k and encoder $q_{\boldsymbol{\phi}}$, we trained RISE with ten million environment steps. In each episode, the agent was also set to interact with eight parallel environments with different random seeds. Each episode has a length of 128 steps, producing 1024 pieces of transitions. After that, we calculated the intrinsic reward for all transitions using Eq. (85), where $\alpha = 0.1, \lambda_0 = 0.1$.

Finally, the policy network was updated using a proximal policy optimization (PPO) method [7]. More specifically, we used a PyTorch implementation of the PPO method, which can be found in [58]. The PPO method was trained with a learning rate of 0.0025, a value function coefficient of 0.5, an action entropy coefficient of 0.01, and a generalized-advantage-estimation (GAE) parameter of 0.95 [59]. In particular, a gradient clipping operation with threshold $[-5, 5]$ was performed to stabilize the learning procedure. As for benchmarking methods, we trained them following their default settings reported in the literature [52, 50].

Performance Comparison

The average one-life return is employed as the KPI in our performance comparison. Table 1 illustrates the performance comparison over eight random seeds on nine Atari games. For instance, 5.24k$\pm$1.86k represents the mean return is 5.24k and the standard deviation is 1.86k. The highest performance is shown in bold. As shown in Table 1, RISE achieved the highest performance in all nine games. Both RE3 and MaxRényi achieved the second highest performance in three games. Furthermore, Fig. 24 illustrates the change of average episode return during training of two selected games. It is clear that the growth rate of RISE is faster than all the benchmarking methods.

Next, we compare the training efficiency between RISE and the benchmarking methods, and the frame per second (FPS) is set as the KPI. For instance, if a method takes 10 second to finish the training of an episode, the FPS is computed as the ratio between the time cost and episode length. The time cost involves only interaction and policy updates for the vanilla PPO agent. But the time cost needs to involve further the intrinsic reward generation and auxiliary model updates for other methods. As shown in Fig. 25, the vanilla PPO method achieves the highest computation efficiency, while RISE and RE3 achieve the second highest FPS. In contrast, MaxRényi has far less FPS that RISE and RE3. This mainly because RISE and RE3 require no auxiliary models, while MaxRényi uses a VAE to estimate the probability density function. Therefore, RISE has great advantages in both the policy performance and learning efficiency.

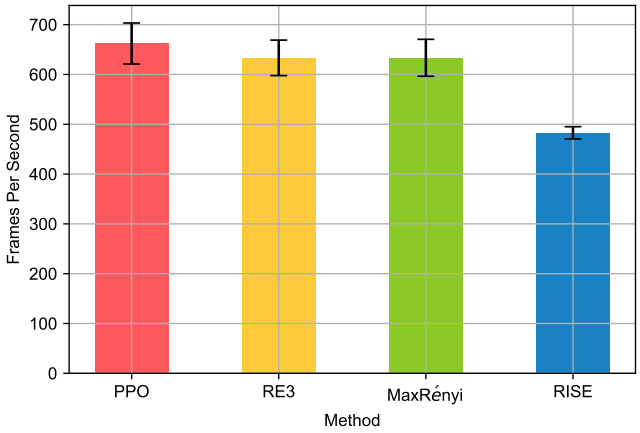

Figure 25: Average computational complexity on Atari games. The experiments were performed in Ubuntu 18.04 LTS operating system with a Intel 10900x CPU and a NVIDIA RTX3090 GPU.

5.5.3 Bullet Games

Experimental Setting Finally, we tested RISE on six Bullet games [60] with continuous action space, namely *Ant*, *Half Cheetah*, *Hopper*, *Humanoid*, *Inverted Pendulum* and *Walker 2D*. In all six games, the target of the agent is to move forward as fast as possible without falling to the ground. Unlike the Atari games that have graphic observations, the Bullet games use fixed-length vectors as observations. For instance, the "Ant" game uses 28 parameters to describe the state of the agent, and its action is a vector of 8 values within $[-1.0, 1.0]$.

We leveraged the multilayer perceptron (MLP) to implement RISE and the benchmarking methods. The detailed network architectures are illustrated in Table 4. Note that the encoder and decoder were designed for MaxRényi, and no BN layers were introduced in this experiment. Since the state space is far simpler than the Atari games, the entropy can be directly derived from the observations while the

Table 2: Performance comparison of six Bullet games.

Game	PPO	PPO+RE3	PPO+MaxRényi	PPO+RISE
Ant	2.25k±0.06k	2.36k±0.01k	2.43k±0.03k	**2.71k±0.07k**
Half Cheetah	2.36k±0.02k	2.41k±0.02k	2.40k±0.01k	**2.47k±0.07k**
Hopper	1.53k±0.57k	2.08k±0.55k	2.23k±0.31k	**2.44k±0.04k**
Humanoid	0.83k±0.25k	0.95k±0.64k	1.17k±0.54k	**1.24k±0.92k**
Inverted Pendulum	1.00k±0.00k	1.00k±0.00k	1.00k±0.00k	**1.00k±0.00k**
Walker 2D	1.66k±0.37k	1.85k±0.71k	1.73k±0.21k	**1.96k±0.34k**

training procedure for the encoder is omitted. We trained RISE with ten million environment steps. The agent was also set to interact with eight parallel environments with different random seeds, and Gaussian distribution was used to sample actions. The rest of the updating procedure was consistent with the experiments of the Atari games.

Performance Comparison Table 2 illustrates the performance comparison between RISE and the benchmarking methods. Inspection of Table 2 suggests that RISE achieved the best performance in all six games. In summary, RISE has shown great potential for achieving excellent performance in both discrete and continuous control tasks.

6 Conclusion

6.1 Contributions

In this thesis, we have investigated the problem of improving exploration in RL. We first systematically analyze the exploration-exploitation dilemma via multi-armed bandit problem before introducing several classical exploration strategies, such as Thompson sampling and noise-based exploration. Since those methods cannot effectively adapt to hard-exploration environments, we further introduce the intrinsically-motivated RL that utilizes intrinsic learning motivation to encourage exploration. We carefully classify the existing intrinsic reward methods, and analyzed their practical drawbacks, such as vanishing rewards and noisy-TV problem. Finally, we proposed a brand new intrinsic reward method via Rényi state entropy maximization that overcomes the drawbacks of the preceding methods and provides powerful exploration incentives. Extensive simulation is performed to compare the performance of RISE against existing methods using both discrete and continuous control tasks as well as several hard exploration games. Simulation results confirm that the proposed module achieve superior performance with high efficiency and robustness.

6.2 Future Work

Intrinsically-motivated RL has achieved great progresses in hard-exploration problem. However, there remains several critical challenges in its consequent development. Most existing intrinsic reward methods cannot guarantee the policy invariance. Despite they improve the exploration performance, the agent may never learn the desired optimal policy. On other hand, the mechanism of intrinsic reward still lacks rigorous mathematical interpreability, which greatly limits its practical application and development. In particular, the existing RL algorithms still produce ultra-high computation complexity, even with the promotion of those exploration methods. Therefore, more research efforts should be devoted to improving the efficiency of these algorithms. This paper is expected to inspire more subsequent research.

A Experimental Settings

A.1 Network Architectures

Table 3: The CNN-based network architectures.

Module	Policy network $\pi_{\boldsymbol{\theta}}$	Value network
Input	States	States
Arch.	8×8 Conv 32, ReLU 4×4 Conv 64, ReLU 3×3 Conv 32, ReLU Flatten Dense 512, ReLU Dense $\lvert\mathcal{A}\rvert$ Categorical Distribution	8×8 Conv 32, ReLU 4×4 Conv 64, ReLU 3×3 Conv 32, ReLU Flatten Dense 512, ReLU Dense 1
Output	Actions	Predicted values
Module	Encoder $p_{\boldsymbol{\psi}}$	Decoder $q_{\boldsymbol{\phi}}$
Input	States	Latent variables
Arch.	3×3 Conv. 32, ReLU 3×3 Conv. 32, ReLU 3×3 Conv. 32, ReLU 3×3 Conv. 32 Flatten Dense 512 & Dense 512 Gaussian sampling	Dense 64, ReLU Dense 1024, ReLU Reshape 3×3 Deconv. 64, ReLU 3×3 Deconv. 64, ReLU 3×3 Deconv. 64, ReLU 8×8 Deconv. 32 1×1 Conv. 4
Output	Latent variables	Predicted states

Table 4: The MLP-based network architectures.

Module	Policy network $\pi_{\boldsymbol{\theta}}$	Value network
Input	States	States
Arch.	Dense 64, Tanh Dense 64, Tanh Dense $\lvert\mathcal{A}\rvert$ Gaussian Distribution	Dense 64, Tanh Dense 64, Tanh Dense 1
Output	Actions	Predicted values
Module	Encoder $p_{\boldsymbol{\psi}}$	Decoder $q_{\boldsymbol{\phi}}$
Input	States	Latent variables
Arch.	Dense 32, Tanh Dense 64, Tanh Dense 256 Dense 256 & Dense 512 Gaussian sampling	Dense 32, Tanh Dense 64, Tanh Dense observation shape
Output	Latent variables	Predicted states

A.2 Hyper-parameter Settings

Table 5: Hyperparameters of RISE used for Atari games.

Phase	**Hyperparameter**	**Value**
k-value searching and encoder training	Number of parallel environments	8
	Number of sample steps N	1e+5
	Threshold of k-value $k_{\max}$	15
	Embedding size m	128
	Number of subsets K	8
	Learning rate	5e-3
	Batch size	256
	Coefficient of divergence λ_D	1.0
	Coefficient of reconstruction loss λ_R	1.0
Policy update	Maximum environment steps	1e+7
	Observation size	(84, 84)
	Stacked frames	4
	Frame skipping	False
	Learning rate for actor & critic	2.5e-4
	Batch size	256
	k	5
	Order of Rényi entropy α	0.1
	Initial coefficient of intrinsic reward β_0	0.1
	Decay rate κ	1e-5
	Coefficient of GAE	0.95
	Coefficient of value function	0.05
	Gradient clipping threshold	[-5,5]

Table 6: Hyperparameters of RISE used for Bullet games.

Phase	**Hyperparameter**	**Value**
k-value searching and encoder training	Number of parallel environments	8
	Number of sample steps	1e+5
	Threshold of k-value	15
	Number of subsets K	8
Policy update	Maximum environment steps	1e+7
	Learning rate for actor & critic	2.5-4
	Batch size	512
	k	5
	Order of Rényi entropy α	0.1
	Coefficient of intrinsic reward β_0	0.1
	Decay rate κ	1e-5
	Coefficient of GAE	0.95
	Coefficient of value function	0.05
	Gradient clipping threshold	[-5,5]

References

[1] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[2] Ivan Petrovitch Pavlov and William Gantt. Lectures on conditioned reflexes: Twenty-five years of objective study of the higher nervous activity (behaviour) of animals. 1928.

[3] Jack Michael. Positive and negative reinforcement, a distinction that is no longer necessary; or a better way to talk about bad things. *Behaviorism*, 3(1):33–44, 1975.

[4] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[5] James R Norris and James Robert Norris. *Markov chains*. Number 2. Cambridge university press, 1998.

[6] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

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

[8] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[9] Bradly C Stadie, Sergey Levine, and Pieter Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*, 2015.

[10] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

[11] Jürgen Schmidhuber. Developmental robotics, optimal artificial curiosity, creativity, music, and the fine arts. *Connection Science*, 18(2):173–187, 2006.

[12] Jürgen Schmidhuber. Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE transactions on autonomous mental development*, 2(3):230–247, 2010.

[13] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pages 2778–2787. PMLR, 2017.

[14] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

[15] Daniel Ying-Jeh Little and Friedrich Tobias Sommer. Learning and exploration in action-perception loops. *Frontiers in neural circuits*, 7:37, 2013.

[16] Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational information maximizing exploration. *Advances in neural information processing systems*, 29, 2016.

[17] Nat Dilokthanakul, Christos Kaplanis, Nick Pawlowski, and Murray Shanahan. Feature control as intrinsic motivation for hierarchical reinforcement learning. *IEEE transactions on neural networks and learning systems*, 30(11):3409–3418, 2019.

[18] 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.

[19] Georg Ostrovski, Marc G Bellemare, Aäron Oord, and Rémi Munos. Count-based exploration with neural density models. In *International conference on machine learning*, pages 2721–2730. PMLR, 2017.

[20] Brendan O'Donoghue, Ian Osband, Remi Munos, and Volodymyr Mnih. The uncertainty bellman equation and exploration. In *International Conference on Machine Learning*, pages 3836–3845, 2018.

[21] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. *Advances in neural information processing systems*, 29, 2016.

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

[23] Pierre-Yves Oudeyer and Frederic Kaplan. What is intrinsic motivation? a typology of computational approaches. *Frontiers in neurorobotics*, 1:6, 2009.

[24] Alexander L Strehl and Michael L Littman. An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331, 2008.

[25] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

[26] Jarryd Martin, Suraj Narayanan Sasikumar, Tom Everitt, and Marcus Hutter. Count-based exploration in feature space for reinforcement learning. *arXiv preprint arXiv:1706.08090*, 2017.

[27] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016.

[28] Nikolay Savinov, Anton Raichuk, Raphaël Marinier, Damien Vincent, Marc Pollefeys, Timothy Lillicrap, and Sylvain Gelly. Episodic curiosity through reachability. *arXiv preprint arXiv:1810.02274*, 2018.

[29] Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[30] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[31] Alekh Agarwal, Nan Jiang, Sham M. Kakade, and Wen Sun. *Reinforcement learning: Theory and Algorithms*. https://rltheorybook.github.io/, 2020.

[32] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

[33] Aurélien Garivier and Eric Moulines. On upper-confidence bound policies for switching bandit problems. In *International Conference on Algorithmic Learning Theory*, pages 174–188. Springer, 2011.

[34] Daniel Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, and Zheng Wen. A tutorial on thompson sampling. *arXiv preprint arXiv:1707.02038*, 2017.

[35] Linda E Reichl. A modern course in statistical physics, 1999.

[36] Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE mobile computing and communications review*, 5(1):3–55, 2001.

[37] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[38] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *International Conference on Machine Learning*, pages 1352–1361. PMLR, 2017.

[39] 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.

[40] George E Uhlenbeck and Leonard S Ornstein. On the theory of the brownian motion. *Physical review*, 36(5):823, 1930.

[41] Robert W White. Motivation reconsidered: the concept of competence. *Psychological review*, 66(5):297, 1959.

[42] Marco Dorigo and Marco Colombetti. Robot shaping: Developing autonomous agents through learning. *Artificial intelligence*, 71(2):321–370, 1994.

[43] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287, 1999.

[44] Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A Efros. Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355*, 2018.

[45] Xingrui Yu, Yueming Lyu, and Ivor Tsang. Intrinsic reward driven imitation learning via generative model. In *International Conference on Machine Learning*, pages 10925–10935. PMLR, 2020.

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

[47] Adrià Puigdomènech Badia, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, Bilal Piot, Steven Kapturowski, Olivier Tieleman, Martín Arjovsky, Alexander Pritzel, Andew Bolt, et al. Never give up: Learning directed exploration strategies. *arXiv preprint arXiv:2002.06038*, 2020.

[48] Roberta Raileanu and Tim Rocktäschel. Ride: Rewarding impact-driven exploration for procedurally-generated environments. *arXiv preprint arXiv:2002.12292*, 2020.

[49] Philippe Flajolet, Daniele Gardy, and Loÿs Thimonier. Birthday paradox, coupon collectors, caching algorithms and self-organizing search. *Discrete Applied Mathematics*, 39(3):207–229, 1992.

[50] Younggyo Seo, Lili Chen, Jinwoo Shin, Honglak Lee, Pieter Abbeel, and Kimin Lee. State entropy maximization with random encoders for efficient exploration. In *International Conference on Machine Learning*, pages 9443–9454. PMLR, 2021.

[51] Harshinder Singh, Neeraj Misra, Vladimir Hnizdo, Adam Fedorowicz, and Eugene Demchuk. Nearest neighbor estimates of entropy. *American journal of mathematical and management sciences*, 23(3-4):301–321, 2003.

[52] Chuheng Zhang, Yuanying Cai, Longbo Huang, and Jian Li. Exploration by maximizing rényi entropy for reward-free rl framework. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 10859–10867, 2021.

[53] Elad Hazan, Sham Kakade, Karan Singh, and Abby Van Soest. Provably efficient maximum entropy exploration. In *International Conference on Machine Learning*, pages 2681–2691. PMLR, 2019.

[54] Nikolai Leonenko, Luc Pronzato, and Vippal Savani. A class of rényi information estimators for multidimensional densities. *The Annals of Statistics*, 36(5):2153–2182, 2008.

[55] Matthew Chan. A simple maze environment. *URL https://github.com/MattChanTK/gym-maze*, 2016.

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

[57] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[58] Kostrikov. Pytorch implementation of the reinforcement learning algorithms. *URL https://github.com/ikostrikov/ pytorch-a2c-ppo-acktr-gail*, 2018.

[59] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[60] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. 2016.