Rényi State Entropy Maximization for Exploration Acceleration in Reinforcement Learning

Mingqi Yuan, Man-On Pun, Senior Member, IEEE and Dong Wang

Abstract—One of the most critical challenges in deep reinforcement learning is to maintain the long-term exploration capability of the agent. To tackle this problem, it has been recently proposed to provide intrinsic rewards for the agent to encourage exploration. However, most existing intrinsic reward-based methods proposed in the literature fail to provide sustainable exploration incentives, a problem known as vanishing rewards. In addition, these conventional methods incur complex models and additional memory in their learning procedures, resulting in high computational complexity and low robustness. In this work, a novel intrinsic reward module based on the Rényi entropy is proposed to provide high-quality intrinsic rewards. It is shown that the proposed method actually generalizes the existing state entropy maximization methods. In particular, a \(k\)-nearest neighbor estimator is introduced for entropy estimation while a \(k\)-value estimator is designed to guarantee the estimation accuracy. Extensive simulation results demonstrate that the proposed Rényi entropy-based method can achieve higher performance as compared to existing schemes. The simulation code used in this work is available in the following GitHub website \(^1\).

Impact Statement—Reinforcement learning (RL) has demonstrated impressive performance in many complex games such as Go and StarCraft. However, the existing RL algorithms suffer from prohibitively expensive computational complexity, poor generalization ability, and low robustness, which hinders its practical applications in the real world. Thus, it is essential to develop more effective RL algorithms for real-life applications such as autonomous driving and smart manufacturing. To tackle this problem, one critical design challenge is to improve the exploration mechanism of RL to realize efficient policy learning. This work proposes a simple and yet, effective method that can significantly improve the exploration ability of RL algorithms, which can be easily applied to real-life applications. For instance, it will facilitate the development of more powerful autonomous driving systems that can adapt to more complex and challenging environments. Finally, this work is also expected to inspire more subsequent research.

Index Terms—Reinforcement learning, exploration, intrinsic reward, Rényi entropy.

I. INTRODUCTION

REINFORCEMENT learning (RL) algorithms are designed to strike an appropriate balance between exploitation and exploration [1]. However, many existing RL algorithms suffer from insufficient exploration, i.e., the agent cannot keep exploring the environment to visit all possible state-action pairs [2]. As a result, the learned policy prematurely falls into local optima after finite iterations [3]. To address the problem, a simple approach is to employ stochastic policies such as the \(\epsilon\)-greedy policy and the Boltzmann exploration [4]. These policies randomly select one action with a non-zero probability in each state. For continuous control tasks, an additional noise term can be added to the action to perform limited exploration. Despite the fact that such techniques can eventually learn the optimal policy in the tabular setting, they are futile when handling complex environments with high-dimensional observations.

To cope with the exploration problems above, recent approaches proposed to leverage intrinsic rewards to encourage exploration. In sharp contrast to the extrinsic rewards explicitly given by the environment, intrinsic rewards represent the inherent learning motivation or curiosity of the agent [5]. Most existing intrinsic reward modules can be broadly categorized into two approaches, namely the novelty-based and the prediction error-based approaches [6], [7], [8], [9]. For instance, [10], [11], [12] employed a state visitation counter to evaluate the novelty of states, and the intrinsic rewards are defined to be inversely proportional to the visiting frequency. As a result, the agent is encouraged to revisit those infrequent states while increasing the probability of exploring new states. In contrast, [13], [14], [3] followed an alternative approach in which the prediction error of a dynamic model is utilized as intrinsic rewards. Given a state transition, an auxiliary model was designed to predict a successor state based on the current state-action pair. After that, the intrinsic reward is computed as the Euclidean distance between the predicted and the true successor states. In particular, [15] attempted to perform RL using only the intrinsic rewards, showing that the agent could achieve considerable performance in many experiments. Despite their good performance, these count-based and prediction error-based methods suffer from vanishing intrinsic rewards, i.e., the intrinsic rewards decrease with visits [16]. The agent will have no additional motivation to explore the environment further once the intrinsic rewards decay to zero. To maintain exploration across episodes, [17] proposed a never-give-up (NGU) framework that learns mixed intrinsic rewards composed of episodic and life-long state.

---

1https://github.com/yuanmingqi/RISE
n novelty. NGU evaluates the episodic state novelty using a
slot-based memory and pseudo-count method [12], which
encourages the agent to visit as many distinct states as possible
in each episode. Since the memory is reset at the beginning
of each episode, the intrinsic rewards will not decay during
the training process. Meanwhile, NGU further introduced a
random network distillation (RND) module to capture the
life-long novelty of states, which prevents the agent from
visiting familiar states across episodes [8]. However, NGU
suffers from complicated architecture and high computational
complexity, making it difficult to be applied in arbitrary
tasks. A more straightforward framework entitled rewarding-
impact-driven-exploration (RIDE) is proposed in [18]. RIDE
inherits the inverse-forward pattern of [13], in which two
dynamic models are leveraged to reconstruct the transition
process. More specifically, the Euclidean distance between two
consecutive encoded states is utilized as the intrinsic reward,
which encourages the agent to take actions that result in more
state changes. Moreover, RIDE uses episodic state visitation
counts to discount the generated rewards, preventing the agent
from staying at states that lead to large embedding differences
while avoiding the television dilemma reported in [19].

However, both NGU and RIDE pay excessive attention to
specific states while failing to reflect the global exploration
extent. Furthermore, they suffer from poor mathematical inter-
pretability and performance loss incurred by auxiliary models.
To circumvent these problems, a state entropy maximiza-
tion method entitled random-encoder-for-efficient-exploration
(RE3) was proposed in [20] to force the agent to visit the
state space more equitably. In each episode, the observation
data is collected and encoded using a randomly initialized deep
neural network. After that, a $k$-nearest neighbor estimator is
leveraged to realize efficient entropy estimation [21]. Simu-
lation results demonstrated that RE3 significantly improved
the sampling efficiency of both model-free and model-based
RL algorithms with less computational complexity. Despite its
many advantages, RE3 ignores the important $k$-value selection
while its default random encoder entails low adaptability and
robustness. Furthermore, motivated by the observation that
the Shannon entropy-based objective function may lead to
a policy with vanishing probabilities to visit some states,
[22] proposed to maximize the Rényi entropy of state-action
distribution (MaxRényi). However, [22] suffers from high
computational complexity and may mislead the agent due to
its imperfect estimation of the state-action distribution derived
from a variation auto-encoder (VAE) [23].

Inspired by the discussions above, we propose to devise a
more efficient and robust method for state entropy maxi-
mization to improve exploration in RL. In this paper, we propose a
RényI State Entropy (RISE) maximization framework to
provide high-quality intrinsic rewards. Our main contributions
are summarized as follows:

- We propose a Rényi entropy-based intrinsic reward mod-
ule that generalizes the existing state entropy maxi-
mization methods such as RE3, and provide theoretical
analysis for the Rényi entropy-based learning objective.
The new module can be applied in arbitrary tasks with
significantly improved exploration efficiency for both
model-based and model-free RL algorithms;
- By leveraging the VAE model, the proposed module can
realize efficient and robust encoding operation for
accurate entropy estimation, which guarantees its gen-
eralization capability and adaptability. Moreover, a search
algorithm is devised for the $k$-value selection to reduce the
uncertainty of performance loss caused by random
selection;
- 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
higher efficiency.

It is worth pointing out that the proposed RISE aims to
maximize the Rényi entropy of the state probability by leverag-
ing a $k$-nearest-neighbor estimator to realize efficient entropy
estimation without requiring additional auxiliary models and
complicated computation procedures. Thus, RISE can be re-
garded as the synergy of MaxRényi in [22] and RE3 in [20] by
exploiting a more appropriate learning objective and a more
efficient intrinsic reward generation method.

II. PROBLEM FORMULATION

We study the following RL problem that considers a
Markov decision process (MDP) characterized by a tuple
$M = (S, A; T, r, \rho(s_0), \gamma)$ [1], in which $S$ is the state space,
$A$ is the action space, $T(s', a, s) = \rho(s_0)$ is the transition probability,
$r(s, a) : S \times A \rightarrow \mathbb{R}$ is the reward function, $\rho(s_0)$ is
the initial state distribution, and $\gamma \in (0, 1]$ is a discount factor,
respectively. We denote by $\pi(a|s)$ the policy of the agent
that observes the state of the environment before choosing an
action from the action space. The objective of RL is to find
the optimal policy $\pi^*$ that maximizes the expected discounted
return given by:

$$
\pi^* = \arg\max_{\pi \in \Pi} \mathbb{E}_{T \sim \pi} \sum_{t=0}^{T-1} \gamma^t r_t(s_t, a_t),
$$

where $\Pi$ is the set of all stationary policies, and $\tau = (s_0, a_0, \ldots, a_{T-1}, s_T)$ is the trajectory collected by the agent.

In this paper, we aim to improve the exploration in RL.
To guarantee the completeness of exploration, the agent is
required to visit all possible states during training. Such an
objective can be regarded as the Coupon collector’s problem
conditioned upon a nonuniform probability distribution [24], in
which the agent is the collector and the states are the coupons.
We denote by $d^\tau(s)$ the state distribution induced by the policy
$\pi$. Assuming that the agent takes $T$ environment steps to finish
the collection, we can compute the expectation of $T$ as

$$
\mathbb{E}_\pi(T) = \int_0^{\infty} \left(1 - \prod_{i=1}^{|S|} (1 - e^{-d^\tau(s_i)\gamma})\right) dt,
$$

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

For notational simplicity, we will omit the superscript in
$d^\tau(s)$ in the sequel. Efficient exploration aims to find a policy
that optimizes $\min_{\pi \in \Pi} \mathbb{E}_\pi(T)$. However, it is non-trivial to
evaluate Eq. (2) 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

$$H(d) = -\mathbb{E}_{d \sim d(s)} \left[ \log d(s) \right].$$

(3)

However, this objective function may lead to a policy that visits some states with a vanishing probability. In the following section, we will first employ a representative example to demonstrate the practical drawbacks of Eq. (3) before introducing the Rényi entropy to address the problem.

III. RÉNYI STATE ENTROPY MAXIMIZATION

A. Rényi State Entropy

We first formally define the Rényi entropy as follows:

**Definition 1** (Rényi Entropy). Let $X \in \mathbb{R}^m$ be a random vector that has a density function $f(x)$ with respect to Lebesgue measure on $\mathbb{R}^m$, and let $\mathcal{X} = \{x \in \mathbb{R}^m : f(x) > 0\}$ be the support of the distribution. The Rényi entropy of order $\alpha \in (0, 1) \cup (1, +\infty)$ is defined as [22]:

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

(4)

Using Definition 1, we propose the following Rényi state entropy (RISE):

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

(5)

In Fig. 1, a toy example is employed to visualize the contours of different objective functions when an agent learns from an environment characterized by only three states. As shown in Fig. 1, $-\log(T)$ 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. 1 shows that this problem can be alleviated by the Rényi entropy as it better matches $\log(T)$.

The Shannon entropy is far less aggressive in penalizing small probabilities whereas the Rényi entropy provides more flexible exploration capability.

B. Theoretical Analysis

To maximize $H_\alpha(d)$, we consider using a maximum entropy policy computation (MEPC) algorithm proposed in [25] that uses the following two oracles:

**Definition 2** (Approximating planning oracle). Given a reward function $r : \mathcal{S} \rightarrow \mathbb{R}$ and a sub-optimality gap $\epsilon_1$, the planning oracle returns a stationary policy $\pi = \mathcal{O}_{AP}(r, \epsilon_1)$ where $\mathcal{O}_{AP}(\cdot)$ is the approximating planning oracle and

$$V^\pi \geq \max_{\pi \in \Pi} V^\pi - \epsilon_1,$$

(6)

with $V^\pi$ being the state-value function [25].

**Definition 3** (State distribution estimation oracle). Given a sub-optimality gap $\epsilon_2$ and a policy $\pi$, this oracle estimates the state distribution by $\hat{d} = \mathcal{O}_{DE}(\pi, \epsilon_2)$, such that

$$\|\hat{d} - d^\pi\|_\infty \leq \epsilon_2,$$

(7)

where $d^\pi$ is the true state distribution induced by the policy $\pi$ [25].
Given a set of stationary policies $\Pi = \{\pi_0, \pi_1, \ldots\}$, we define a mixed policy denoted by $\pi_{\text{mix}} = (\omega, \tilde{\Pi})$ where $\omega$ is the weighting coefficient vector. The induced state distribution is given by:

$$d^{\text{mix}} = \sum_i \omega_i d^{\pi_i}(s).$$

Finally, the proposed MEPC algorithm is summarized in Algorithm 1.

Algorithm 1 Maximum Entropy Policy Computation (MEPC)
1: Set the number of iterations $T$, step size $\eta$, planning oracle error tolerance $\epsilon_1 > 0$, and state distribution oracle error tolerance $\epsilon_2 > 0$;
2: Initialize $\tilde{\Pi}_0 = \{\pi_0\}$, where $\pi_0$ is an arbitrary policy;
3: Initialize $\omega_0 = 1$;
4: for $t = 0, \ldots, T - 1$ do
5: Invoke the state distribution oracle on $\pi_{\text{mix},t} = (\omega, \tilde{\Pi}_t)$:
6: Define the reward function $r_t$ as $r_t(s) = \nabla H_\alpha(d^{\text{mix},t})$;
7: Approximate the optimal policy on $r_t$:
$$\pi_{t+1} = O_{\text{DE}}(\pi_t, \epsilon_2);$$
8: Update $\pi_{\text{mix},t} = (\omega_{t+1}, \tilde{\Pi}_t)$:
$$\tilde{\Pi}_{t+1} = (\tilde{\pi}_0, \ldots, \tilde{\pi}_t, \pi_{t+1});$$
$$\omega_{t+1} = (1 - \eta)\omega_t, \eta;$$
9: end for
10: Return $\pi_{\text{mix},T} = (\omega_T, \tilde{\Pi}_T)$.

Consider the discrete case of Eq. (5), we have

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

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

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

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

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

where $\sigma > 0$.

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

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

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

Proof: See proof in Appendix B.

From Lemma 1, the following theorem can be derived.

Theorem 1. 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

$$\hat{H}_{\alpha,\sigma}(d^{\text{mix},T}) \geq \max_{\pi \in \Pi} \hat{H}_{\alpha,\sigma}(d^\pi) - \epsilon,$$

if Algorithm 1 is run for $T$ iterations with

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

Proof: See proof in Appendix C.

C. 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 [26].

Theorem 2 (Estimator). Denote by $\{X_i\}_{i=1}^N$ a set of independent random vectors from the distribution $X$. For $k < N$, $k \in \mathbb{N}$, $X_i$ stands for the $k$-nearest neighbor of $X_i$ among the set. We estimate the Rényi entropy using the sample mean as follows:

$$\hat{H}_{N,\alpha}^k(f) = \frac{1}{N} \sum_{i=1}^N [(N - 1)V_mC_k \|X_i - \tilde{X}_i\|_m^{1 - \alpha}],$$

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

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

Proof: See proof in [26].

Note that the sample mean can be derived in a straightforward manner as shown in [26], assuming that all samples are independent identically distributed.

Given a trajectory $\tau = \{s_0, a_0, \ldots, a_{T-1}, s_T\}$ collected by the agent, we approximate the Rényi state entropy in Eq. (4) using Eq. (15) as

$$\hat{H}_{T,\alpha}^k(d) = \frac{1}{T} \sum_{i=0}^{T-1} [(T - 1)V_mC_k \|y_i - \tilde{y}_i\|_m^{1 - \alpha}],$$

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

$$r_i(s_i) = \|y_i - \tilde{y}_i\|_1^{-\alpha},$$

where
where \( \hat{r}(\cdot) \) is used to distinguish the intrinsic reward from the extrinsic reward \( r(\cdot) \). Eq. (18) indicates that the agent needs to visit as many distinct states as possible to obtain higher intrinsic rewards.

Since the estimation method above requires no additional auxiliary models, which significantly improves the learning efficiency. Equipped with the intrinsic reward, the total reward of each transition \((s_t, a_t, s_{t+1})\) is computed as

\[
    r^\text{total}_t = r(s_t, a_t) + \lambda_t \cdot \hat{r}(s_t) + \zeta \cdot H(\pi(\cdot|s_t)),
\]

where \( H(\pi(\cdot|s_t)) \) is the action entropy regularizer for improving the exploration on action space. Furthermore, \( \lambda_t = \lambda_0(1 - \kappa)^t \) and \( \zeta \) are two non-negative weight coefficients with \( \kappa \) being a decay rate.

IV. ROBUST REPRESENTATION LEARNING

While the Rényi state entropy encourages exploration in high-dimensional observation spaces, several implementation issues have to be resolved 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 [20], 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 [23]. As shown in Fig. 2(a), 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.

We denote by \( q_\phi(z|s) \) the recognition model represented by a neural network with parameters \( \phi \). The recognition model accepts an observation input before encoding the input into latent variables. Similarly, we represent the generative model as \( p_\psi(s|z) \) using a neural network with parameters \( \psi \), accepting the latent variables and reconstructing the observation. Given a trajectory \( \tau = \{s_0, a_0, \ldots, a_{T-1}, s_T\} \), the VAE model is trained by minimizing the following loss function:

\[
    L(s_t; \phi, \psi) = \mathbb{E}_{q_\phi(z|s_t)} \left[ \log p_\psi(s_t|z) \right] - D_{KL}(q_\phi(z|s_t) || p_\psi(z)),
\]

where \( D_{KL} \) is the Kullback-Leibler divergence.
where \( t = 0, \ldots, T \), \( D_{\text{KL}}(\cdot) \) 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. [21] 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. 2(b). 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 \( \pi_\theta \) the policy network, the detailed searching algorithm is summarized in Algorithm 2.

**Algorithm 2** \( k \)-value searching method

1. Initialize a policy network \( \pi_\theta \);
2. Initialize the number of sample steps \( N \), the threshold \( k_{\text{max}} \) of \( k \), a null array \( \delta \) with length \( k_{\text{max}} \), and the number of subsets \( K \);
3. Execute policy \( \pi_\theta \) and collect the trajectory \( \tau = \{s_0, a_0, \ldots, a_{T-1}, s_T\} \);
4. Divide the observations dataset \( \{s_i\}_{i=0}^N \) into \( K \) subsets randomly;
5. for \( k = 1, 2, \ldots, k_{\text{max}} \) do
6. Calculate the estimated entropy on \( K \) subsets using Eq. (15):
   \[
   \hat{H}_k = (\hat{H}_{k,N/K,1}, \ldots, \hat{H}_{k,N/K,K})
   \]
7. Calculate the min-max ratio \( \delta(\hat{H}_k) \) and \( \delta[k] \leftarrow \delta(\hat{H}_k) \);
8. end for
9. Output \( k = \arg\min_k \delta[k] \).

Finally, we are ready to propose our RISE framework by exploiting the optimal \( k \) value derived above. As shown in Fig. 2(c), 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 \( y_i \) and its \( k \)-nearest neighbor is computed as the intrinsic reward. Algorithm 3 and Algorithm 4 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.

**V. 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. A brief introduction of these benchmarking methods can be found in Appendix A. 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.

**Algorithm 3** On-policy RL Version of RISE

1. Phase 1: \( k \)-value searching and encoder training
2. Initialize the policy network \( \pi_\theta \), encoder \( q_\phi \) and decoder \( p_\psi \);
3. Initialize the number of sample steps \( N \), the threshold \( k_{\text{max}} \) of \( k \)-value, the embedding size \( m \) and the number subsets \( K \);
4. Execute policy and collect observations data \( \{s_i\}_{i=1}^N \);
5. Use \( \{s_i\}_{i=1}^N \) to train the encoder;
6. Use Algorithm 2 to select \( k \)-value;
7. Phase 2: Policy update
8. Initialize the maximum episodes \( E \), order \( \alpha \), coefficients \( \lambda_0, \zeta \) and decay rate \( \kappa \);
9. for episode \( l = 1, \ldots, E \) do
10. Collect the trajectory \( \tau_l = \{s_0, a_0, \ldots, a_{T-1}, s_T\} \);
11. Compute the embedding vectors \( \{z_i\}_{i=0}^T \) of \( \{s_i\}_{i=0}^T \)
   using the encoder \( q_\phi \);
12. Compute \( \hat{r}(s_i) \leftarrow \|z_i - \hat{z}_i\|_2 \);\(~\alpha;\)
13. Update \( \lambda_l = \lambda_0(1-k) \);
14. Let \( r_{i,i}^t = r(s_i, a_i) + \lambda_i \cdot \hat{r}(s_i) + \zeta \cdot H(\pi(\cdot|s_i)) \);
15. Update the policy network with transitions \( \{s_i, a_i, s_{i+1}, r_{i,i}^t\}_{i=1}^B \) using any on-policy RL algorithms.
16. end for

**Algorithm 4** Off-policy RL Version of RISE

1. Phase 1 of Algorithm 3;
2. Phase 2: Policy update
3. Initialize the maximum environment steps \( t_{\text{max}} \), order \( \alpha \), coefficients \( \lambda_0, \zeta \) and replay buffer \( \mathcal{B} \leftarrow \emptyset \);
4. for step \( t = 1, \ldots, t_{\text{max}} \) do
5. Collect the transition \( (s_t, a_t, r_t, s_{t+1}) \) and let \( \mathcal{B} \leftarrow \mathcal{B} \cup \{(s_t, a_t, r_t, s_{t+1}, z_t)\} \), where \( z_t = q_\phi(s_t) \);
6. Sample a minibatch \( \{(s_i, a_i, r_i, s_{i+1}, z_i)\}_{i=1}^B \) from \( \mathcal{B} \) randomly;
7. Compute \( \hat{r}(s_i) \leftarrow \|z_i - \hat{z}_i\|_2 \);\(~\alpha;\)
8. Update \( \lambda_t = \lambda_0(1-k) \);
9. Let \( r_{i,i}^t = r(s_i, a_i) + \lambda_t \cdot \hat{r}(s_i) + \zeta \cdot H(\pi(\cdot|s_i)) \);
10. Update the policy network with transitions \( \{s_i, a_i, s_{i+1}, r_{i,i}^t\}_{i=1}^B \) using any off-policy RL algorithms.
11. end for
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.

![A maze game with grid size 20 x 20.](image)

Fig. 3. A maze game with grid size 20 x 20.

1) Experimental Setting: The standard Q-learning (QL) algorithm [2] 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:

$$Q(s, a) \leftarrow Q(s, a) + \eta [r + \gamma \max_{a'} Q(s', a') - Q(s, a)] , \quad (22)$$

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

2) Performance Comparison: We compare the exploration performance by investigating the minimum number of environment steps required to visit all states. In this experiment, we consider three mazes of size $M \times M$ for $M = 10, 20$ and 30. As there are in total $M^2$ grids, this amounts to $M^2$ states. The minimum number of steps for the agent to visit all $M^2$ states was recorded. The averaged minimum step numbers over 100 simulation runs are shown in Fig. 4. Inspection of Fig. 4 reveals that the proposed Q-learning+RISE achieved the best performance in all three maze games. Moreover, RISE with smaller $\alpha$ takes less steps to finish the exploration phase. This experiment confirmed the effectiveness of the proposed Rényi state entropy-driven exploration.

B. 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 [28]. 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.

![Average exploration performance comparison over 100 simulation runs.](image)

Fig. 4. Average exploration performance comparison over 100 simulation runs.

<table>
<thead>
<tr>
<th>Module</th>
<th>Policy network $\pi_{\theta}$</th>
<th>Value network $\phi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>States</td>
<td>States</td>
</tr>
<tr>
<td>Arch.</td>
<td>8x8 Conv 32, ReLU</td>
<td>8x8 Conv 32, ReLU</td>
</tr>
<tr>
<td>Flatten</td>
<td>Dense 512, ReLU</td>
<td>Dense 512, ReLU</td>
</tr>
<tr>
<td>Dense $</td>
<td>A</td>
<td>$</td>
</tr>
<tr>
<td>Flatten</td>
<td>Dense 512 &amp; Dense 512</td>
<td>Gaussian sampling</td>
</tr>
<tr>
<td>Decoder $\psi_{\phi}$</td>
<td>Dense 64, ReLU</td>
<td>Dense 64, ReLU</td>
</tr>
<tr>
<td>Reshape</td>
<td>3x3 Deconv. 64, ReLU</td>
<td>3x3 Deconv. 64, ReLU</td>
</tr>
<tr>
<td>Flatten</td>
<td>3x3 Deconv. 64, ReLU</td>
<td>8x8 Deconv. 32</td>
</tr>
<tr>
<td>Dense $</td>
<td>A</td>
<td>$</td>
</tr>
<tr>
<td>Output</td>
<td>Latent variables</td>
<td>Predicted states</td>
</tr>
</tbody>
</table>

1) 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 under consideration, and their architectures can be found in Table I. For instance, “8x8 Conv 32” represents a convolutional layer that has 32 filters of size $8 \times 8$. 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 [29]. Note that “Dense 512 & Dense 512” in Table I means that there are two branches to output the mean and variance of the
was trained with a learning rate of the PPO method, which can be found in [31]. The PPO method [30]. More specifically, we used a PyTorch implementation of \( \alpha \) calculated the intrinsic reward for all transitions using Eq. (18), steps, producing \( 128 \) agent was also set to interact with eight parallel environments RISE with ten million environment steps. In each episode, the over the range of \( [1, 15] \) using Algorithm 2.

Equipped with the learned \( k \) and encoder \( q_{\phi} \), 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. (18), where \( \alpha = 0.1, \lambda_0 = 0.1 \). Finally, the policy network was updated using a proximal policy optimization (PPO) method [30]. More specifically, we used a PyTorch implementation of the PPO method, which can be found in [31]. 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 [32]. 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 [22], [20].

2) Performance Comparison: Fig. 5 illustrates the average episode return during the training process for two of the six selected games. It is clear that the growth rate of RISE is faster than all the benchmarking methods. To quantitatively compare the performance of RISE and existing methods, the average return over the last 10 steps is computed for each method in each game. Table II illustrates the performance comparison over eight random seeds on six Atari games. For instance, 5.24k±1.86k represents the mean return is \( 5.24k \pm 1.86k \). The highest performance is shown in bold. As shown in Table II, RISE achieved the highest performance in all six games whereas both RE3 and MaxRényi achieved the second highest performance in three games.

Next, we compare the training efficiency between RISE and other existing methods in terms of frame per second (FPS). More specifically, the FPS is defined as the ratio between the time cost and episode length. It is worth pointing out that the time cost for the vanilla PPO agent involves only interaction and policy updates while that for other methods further include the intrinsic reward generation and auxiliary model updates. As a result, the vanilla PPO method achieved the highest FPS followed by RISE and RE3 as shown in Fig. 6. In contrast, MaxRényi had much lower FPS performance than

<table>
<thead>
<tr>
<th>Game</th>
<th>PPO</th>
<th>PPO+RE3</th>
<th>PPO+MaxRényi</th>
<th>PPO+RISE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assault</td>
<td>5.24k±1.86k</td>
<td>5.54k±2.12k</td>
<td>5.68k±1.99k</td>
<td>5.78k±2.44k</td>
</tr>
<tr>
<td>Battle Zone</td>
<td>18.63k±1.96k</td>
<td>20.47k±2.73k</td>
<td>19.17k±3.48k</td>
<td>21.80k±5.44k</td>
</tr>
<tr>
<td>Demon Attack</td>
<td>13.55k±4.65k</td>
<td>17.40k±9.63k</td>
<td>16.18k±8.19k</td>
<td>18.39k±7.61k</td>
</tr>
<tr>
<td>Kung Fu Master</td>
<td>21.86k±8.92k</td>
<td>23.21k±5.74k</td>
<td>27.20k±5.59k</td>
<td>27.76k±5.86k</td>
</tr>
<tr>
<td>Riverraid</td>
<td>7.99k±0.53k</td>
<td>8.14k±0.37k</td>
<td>8.21k±0.36k</td>
<td>10.07k±0.77k</td>
</tr>
<tr>
<td>Space Invaders</td>
<td>0.72k±0.25k</td>
<td>0.89k±0.33k</td>
<td>0.81k±0.37k</td>
<td>1.09k±0.21k</td>
</tr>
</tbody>
</table>
RISE and RE3. This is mainly because MaxRényi used a VAE to estimate the probability density function while RISE and RE3 are exempted from auxiliary models. As a result, RISE is more advantageous in terms of both policy performance and learning efficiency.

C. Bullet Games

1) Experimental Setting: Finally, we tested RISE on six Bullet games [33] 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 onto 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 \([-1, 0, 1]\).

We leveraged the multilayer perceptron (MLP) to implement RISE and the benchmarking methods. The detailed network architectures are illustrated in Table IV. Note that the encoder and decoder were designed for MaxRényi while no BN layers were employed in this experiment. Furthermore, as the state space of the Bullet games is much simpler than the Atari games, the entropy was directly derived from the observations while the training procedure for the encoder was 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.

2) Performance Comparison: Table III illustrates the performance comparison between RISE and the benchmarking methods in terms of the average return for the last 10 steps over 8 random runs. Inspection of Table III 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.

VI. CONCLUSION

In this paper, we have investigated the problem of improving exploration in RL by proposing a Rényi state entropy (RISE) maximization method to provide high-quality intrinsic rewards. Our method generalizes the existing state entropy maximization method to achieve higher generalization capability and flexibility. Moreover, a $k$-value search algorithm has been developed to obtain efficient and robust entropy estimation by leveraging a VAE model, which makes the proposed method practical for real-life applications. Finally, extensive simulation has been performed on both discrete and continuous tasks from the Open AI Gym library and Bullet library. Our simulation results have confirmed that the proposed method can substantially outperform conventional methods through efficient exploration.

![Fig. 6. 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.](image)

### TABLE III

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

### TABLE IV

<table>
<thead>
<tr>
<th>Module</th>
<th>Policy network $\pi_\theta$</th>
<th>Value network $q_\phi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>States</td>
<td>States</td>
</tr>
<tr>
<td>Arch.</td>
<td>Dense 64, Tanh</td>
<td>Dense 64, Tanh</td>
</tr>
<tr>
<td></td>
<td>Dense 64, Tanh</td>
<td>Dense 64, Tanh</td>
</tr>
<tr>
<td></td>
<td>Gaussian Distribution</td>
<td>Dense 1</td>
</tr>
<tr>
<td>Output</td>
<td>Actions</td>
<td>Predicted values</td>
</tr>
<tr>
<td>Module</td>
<td>Encoder $p_\psi$</td>
<td>Decoder $q_\phi$</td>
</tr>
<tr>
<td>Input</td>
<td>States</td>
<td>Latent variables</td>
</tr>
<tr>
<td>Arch.</td>
<td>Dense 32, Tanh</td>
<td>Dense 32, Tanh</td>
</tr>
<tr>
<td></td>
<td>Dense 64, Tanh</td>
<td>Dense 64, Tanh</td>
</tr>
<tr>
<td></td>
<td>Dense 526</td>
<td>Dense 526 &amp; Dense 512</td>
</tr>
<tr>
<td></td>
<td>Gaussian sampling</td>
<td>Dense observation shape</td>
</tr>
<tr>
<td>Output</td>
<td>Latent variables</td>
<td>Predicted states</td>
</tr>
</tbody>
</table>
APPENDIX

A. Benchmarking Methods

1) RE3: Given a trajectory \( \tau = (s_0, a_0, \ldots, a_{T-1}, s_T) \), RE3 first uses a randomly initialized DNN to encode the visited states. Denote by \( \{x_i\}_{i=0}^{T-1} \) the encoding vectors of observations, RE3 estimates the entropy of state distribution \( d(s) \) using a k-nearest neighbor entropy estimator [21]:

\[
\tilde{H}^k_i(d) = \frac{1}{T} \sum_{i=0}^{T-1} \log \frac{T \cdot \|x_i - \tilde{x}_i\|^2 \cdot \pi}{k \cdot \Gamma(\frac{d}{2} + 1)} + \log k - \Psi(k)
\]

\[
\approx \frac{1}{T} \sum_{i=0}^{T-1} \log \|x_i - \tilde{x}_i\|_2,
\]

where \( \tilde{x}_i \) is the k-nearest neighbor of \( x_i \) within the set \( \{x_i\}_{i=0}^{T-1} \), \( m \) is the dimension of the encoding vectors, and \( \Gamma(\cdot) \) is the Gamma function, and \( \Psi(\cdot) \) is the digamma function. Note that \( \pi \) in Eq. (23) denotes the ratio between the circumference of a circle to its diameter. Equipped with Eq. (23), the total reward for each transition \( (s_t, a_t, r_t, s_{t+1}) \) is computed as:

\[
r^{\text{total}} = r(s_t, a_t) + \lambda_t \cdot \log (\|x_t - \tilde{x}_t\|^2 + 1),
\]

where \( \lambda_t = \lambda_0(1 - k)^t, \lambda_t \geq 0 \) is a weight coefficient that decays over time, \( k \) is a decay rate. Our RIME method is a generalization of RE3, which provides more aggressive exploration incentives.

2) MaxRényi : The MaxRényi method aims to maximize the Rényi entropy of state-action distribution \( d^R_\alpha(s, a) \). The gradient of its objective function is given by:

\[
\nabla H_\alpha(d^R_\alpha) \propto \frac{1}{1 - \alpha} \mathbb{E}(a | s)_{d^R_\alpha} \left[ \nabla \log \pi(a | s) \right]
\]

\[
\left( \frac{(d^R_{s,a})^\alpha - 1}{\gamma} + \frac{(d^R_{s,a})^\alpha - 1}{\alpha - 1} \right)
\]

It uses VAE to estimate \( d(s) \) and take the evidence lower bound (ELBO) as the density estimation [23], which suffers from low efficiency and high variance.

B. Proof of Lemma 1

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

\[
\|\nabla \tilde{H}_{\alpha,\sigma}(d) - \nabla \tilde{H}_{\alpha,\sigma}(d')\|_\infty
\]

\[
\leq \max_{\zeta \in [0, 1]} \|\nabla^2 \tilde{H}_{\alpha,\sigma}(s(d + (1 - \zeta)d')) \cdot \|d - d'\|_\infty
\]

\[
\leq \alpha \sigma^{\alpha-2} \|d - d'\|_\infty,
\]

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

C. Proof of Theorem 1

Let \( \pi^* = \arg\max_{\pi \in \Pi} \tilde{H}_{\alpha,\sigma}(d) \). Using Lemma 1, we have [25]:

\[
\tilde{H}_{\alpha,\sigma}(d^\pi) - \tilde{H}_{\alpha,\sigma}(d^\text{mix,}\tau) \leq B \exp(-T\eta) + 2\beta\epsilon^2 + \epsilon_1 + \eta\beta,
\]

where \( \|\nabla \tilde{H}_{\alpha,\sigma}(d)\|_\infty \leq B = \frac{\alpha}{1 - \alpha} \sigma^{\alpha-1} \).

Thus, it suffices to set \( \epsilon_1 = 0.1\epsilon, \epsilon_2 = 0.1\beta^{-1}\epsilon, \eta = 0.1\beta^{-1}\epsilon, T = \eta^{-1} \log 10B\epsilon^{-1} \). After Algorithm 1 is run for \( T \) iterations, where

\[
T \geq 10\beta^{-1}\epsilon^{-1} \log 10B\epsilon^{-1},
\]

the following inequality holds

\[
\tilde{H}_{\alpha,\sigma}(d^\text{mix,}\tau) \geq \max_{\pi \in \Pi} \tilde{H}_{\alpha,\sigma}(d^\pi) - \epsilon.
\]

This concludes the proof.

REFERENCES


Mingqi Yuan received his BSc. degree in Information and Computing Science from the Minzu University of China (MUC) in 2020. He had worked at JD.com, Inc. and Tencent Robotics X Lab as a research intern in 2019 and 2022, developing intelligent algorithms for smart agriculture and humanoid robot.

Currently, he is pursing the M.Phil. degree at The Chinese University of Hong Kong, Shenzhen (CUHK SZ), China. His research interests include reinforcement learning, robotics and computer vision.

Man-On Pun (Senior Member, IEEE) received his BEng. degree in Electronic Engineering from the Chinese University of Hong Kong (CUHK) in 1996, the MEng. degree in Computer Science from University of Tsukuba, Japan in 1999 and the Ph.D. degree in Electrical Engineering from University of Southern California (USC) in Los Angeles, U.S.A in 2006, respectively. He was a postdoctoral research associate at Princeton University from 2006 to 2008.

Currently, he is an associate professor at the School of Science and Engineering, the Chinese University of Hong Kong, Shenzhen (CUHKSZ). Prior to joining CUHKSZ in 2015, he held research positions at Huawei (USA), Mitsubishi Electric Research Labs (MERL) in Boston and Sony in Tokyo, Japan. Prof. Pun’s research interests include AI Internet of Things (AIoT) and applications of machine learning in communications and satellite remote sensing.

Prof. Pun has received best paper awards from IEEE VTC’06 Fall, IEEE ICC’08 and IEEE Infocom’09. He served as associate editor for the IEEE Transactions on Wireless Communications in 2010 - 2014. He is the founding chair of the IEEE Joint SPS-ComSoc Chapter, Shenzhen.

Dong Wang is an assistant professor in the School of Science and Engineering at the Chinese University of Hong Kong, Shenzhen since 2020. He has broad interests in analytical and computational methods for problems in applied mathematics, especially in computational fluid dynamics, computational material science, image processing, optimization and machine learning. After receiving his bachelor’s degree in mathematics from Sichuan University in 2013, Dong earned his Ph.D. in Applied Mathematics at the Hong Kong University of Science and Technology in 2017. Before moving to CUHK(SZ), he was an Assistant Professor Lecturer in the Department of Mathematics at the University of Utah.

First A. Author et al.: Bare Demo of IEEEtran.cls for IEEE Journals of IEEE Transactions on Artificial Intelligence 11